

Instytut Podstaw Informatyki
Polskiej Akademii Nauk
ul. Jana Kazimierza 5
01-248 Warszawa
za pośrednictwem:
Rady Doskonałości Naukowej
pl. Defilad 1
00-901 Warszawa
(Pałac Kultury i Nauki, p. XXIV, pok. 2401)

.....Marek Nowicki.....
(imię i nazwisko wnioskodawcy)
.....Uniwersytet Mikołaja Kopernika w Toruniu.....
.....Uniwersytet Warszawski.....
(miejsce pracy/jednostka naukowa)

Wniosek
z dnia 26 listopada 2025

o przeprowadzenie postępowania w sprawie nadania stopnia doktora habilitowanego w dziedzinie nauk inżyneryjno-technicznych w dyscyplinie informatyka techniczna i telekomunikacja.

Wnoszę – na podstawie art. 221 ust. 10 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce – aby komisja habilitacyjna podejmowała uchwałę w sprawie nadania stopnia doktora habilitowanego w głosowaniu ~~tajnym~~/jawnym*.

Zostałem poinformowany, że:

Administratorem w odniesieniu do danych osobowych pozyskanych w ramach postępowania w sprawie nadania stopnia doktora habilitowanego jest Przewodniczący Rady Doskonałości Naukowej z siedzibą w Warszawie (pl. Defilad 1, XXIV piętro, 00-901 Warszawa).

Kontakt za pośrednictwem e-mail: kancelaria@rdn.gov.pl, tel. 22 656 60 98 lub w siedzibie organu. Dane osobowe będą przetwarzane w oparciu o przesłankę wskazaną w art. 6 ust. 1 lit. c) Rozporządzenia UE 2016/679 z dnia 27 kwietnia 2016 r. w związku z art. 220-221 oraz art. 232-240 ustawy z dnia 20 lipca 2018 roku – Prawo o szkolnictwie wyższym i nauce, w celu przeprowadzenia postępowania o nadanie stopnia doktora habilitowanego oraz realizacji praw i obowiązków oraz środków odwoławczych przewidzianych w tym postępowaniu.

Szczegółowa informacja na temat przetwarzania danych osobowych w postępowaniu dostępna jest na stronie www.rdn.gov.pl/klauzula-informacyjna-rodo.html

 **PODPIS ZAUFANY**
MAREK NOWICKI
26.11.2025 15:57:40 GMT+1
Dokument podpisany elektronicznie
podpisem zaufanym

podpis wnioskodawcy

*Niepotrzebne skreślić.

Załączniki:

1. dane_wnioskodawcy.pdf – dane wnioskodawcy (po polsku)
2. autoreferat.pdf – autoreferat (po polsku)
3. autoreferat_en.pdf – autoreferat (po angielsku)
4. wykaz_osiagniec.pdf – wykaz osiągnięć naukowych (po polsku)
5. wykaz_osiagniec_en.pdf – wykaz osiągnięć naukowych (po angielsku)
6. [1]_201801-jcomputbiol_cmb.2018.0079.pdf – publikacja [1], pt. *Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library*
7. [2]_201805-hpcs18_pcj_tutorial.pdf – publikacja [2], pt. *PCJ – Java Library for Highly Scalable HPC and Big Data Processing*
8. [3]_202007-jbigdata_terasort.pdf – publikacja [3], pt. *Comparison of sort algorithms in Hadoop and PCJ*
9. [4]_202012-cpe.6536.pdf – publikacja [4], pt. *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*
10. [5]_202101-jbigdata_hpc_bigdata_ai.pdf – publikacja [5], pt. *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*
11. [6]_202103-ISGC2021_007.pdf – publikacja [6], pt. *Scalable computing in Java with PCJ Library. Improved collective operations*
12. [7]_202409-bbaf155_hpv-kite.pdf – publikacja [7], pt. *HPV-KITE: sequence analysis software for rapid HPV genotype detection*
13. oswiadczenia_wspolautorow.pdf – oświadczenia współautorów prac o ich wkładzie
14. dyplom-a3.pdf – kopia dokumentu potwierdzającego posiadanie stopnia doktora
15. Karolinska Institutet - Certificate 251112.pdf – potwierdzenie statusu naukowca afiliowanego w *Karolinska Institutet*
16. zakres_kierownika_projektu_HEAP.pdf – potwierdzenie funkcji kierownika pakietu roboczego projektu *HEAP* w *ICM UW*
17. IBM - document2014-09-01-120446-reference.pdf – potwierdzenie wdrożenia w *IBM*
18. HPC SC14 - Most Elegant.pdf – potwierdzenie zdobycia nagrody *HPC Challenge Class 2 Award: Best Productivity – Most Elegant Solution*
19. Use of PCJ library by IMGT.pdf – potwierdzenie wdrażania w *international ImMunoGenetics information system*

Dane wnioskodawcy

1. **Imię i Nazwisko:**

Marek Nowicki

2. **Miejsce pracy:**

Wydział Matematyki i Informatyki

Uniwersytet Mikołaja Kopernika w Toruniu

Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego

Uniwersytet Warszawski

3. **Adres korespondencyjny:**

[REDACTED]

[REDACTED]

4. **Nr telefonu:**

[REDACTED]

5. **Adres e-mail:**

faramir@mat.umk.pl

6. **Numer PESEL:**

[REDACTED]

7. **Numer i seria dokumentu tożsamości w przypadku braku nadania numeru PESEL:**

nie dotyczy



PODPIS ZAUFANY

**MAREK
NOWICKI**

26.11.2025 15:59:18 GMT+1

Dokument podpisany elektronicznie
podpisem zaufanym

podpis wnioskodawcy

Autoreferat

Marek Nowicki

26 listopada 2025

1 Imię i nazwisko

Marek Nowicki

2 Posiadane dyplomy i stopnie naukowe

Posiadane dyplomy, stopnie naukowe lub artystyczne – z podaniem podmiotu nadającego stopień, roku ich uzyskania oraz tytułu rozprawy doktorskiej

- 2015 Stopień naukowy doktora nauk matematycznych w zakresie informatyki
nadany uchwałą Rady Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu
Warszawskiego z dnia 18 czerwca 2015 r.
Rozprawa doktorska pt.: *Opracowanie nowych metod programowania równoległego
w Javie w oparciu o paradygmat PGAS (Partitioned Global Address Space)*
Promotor: dr hab. Piotr Bała, prof. UW; Promotor Pomocniczy: dr Michał Chlebiej
- 2010 Tytuł zawodowy magistra informatyki,
Wydział Matematyki i Informatyki, Uniwersytet Mikołaja Kopernika w Toruniu
praca magisterska pt.: *Webowy system zarządzania użytkownikami systemu Kardionet*
Promotor: dr hab. Piotr Bała, prof. UMK

3 Zatrudnienie w jednostkach naukowych

Informacja o dotychczasowym zatrudnieniu w jednostkach naukowych lub artystycznych

- od 02/2021 adiunkt w Interdyscyplinarnym Centrum Modelowania Matematycznego
i Komputerowego, Uniwersytet Warszawski
- od 10/2019 adiunkt w Katedrze Informatyki, Wydział Matematyki i Informatyki,
Uniwersytet Mikołaja Kopernika w Toruniu
- 10/2018-09/2019 adiunkt w Zakładzie Obliczeń Równoległych i Rozproszonych, Wydział
Matematyki i Informatyki, Uniwersytet Mikołaja Kopernika w Toruniu
- 10/2014-09/2018 asystent w Zakładzie Obliczeń Równoległych i Rozproszonych, Wydział
Matematyki i Informatyki, Uniwersytet Mikołaja Kopernika w Toruniu
- 05/2013-11/2013 staż w laboratorium badawczym IBM Zurich Research Laboratory w Zurychu,
w Szwajcarii (Mathematical & Computational Science Department)
- 02/2012-01/2013 programista w Interdyscyplinarnym Centrum Modelowania Matematycznego
i Komputerowego Uniwersytetu Warszawskiego

4 Omówienie osiągnięć

Omówienie osiągnięć, o których mowa w art. 219 ust. 1 pkt. 2 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2021 r. poz. 478 z późn. zm.). Omówienie to winno dotyczyć merytorycznego ujęcia przedmiotowych osiągnięć, jak i w sposób precyzyjny określać indywidualny wkład w ich powstanie, w przypadku, gdy dane osiągnięcie jest dziełem współautorskim, z uwzględnieniem możliwości wskazywania dorobku z okresu całej kariery zawodowej.

Tytuł:

Paradygmat PGAS programowania równoległego w języku Java: rozwój i analiza biblioteki PCJ na przykładzie wybranych algorytmów, w tym bioinformatycznych

Cykl powiązanych tematycznie artykułów naukowych

- [1] Marek Nowicki, Davit Bzhalava, and Piotr Bała. Massively parallel implementation of sequence alignment with Basic Local Alignment Search Tool using parallel computing in Java library. *Journal of Computational Biology*, 25(8):871–881, 2018. doi:10.1089/cmb.2018.0079.
- [2] Marek Nowicki, Łukasz Górski, and Piotr Bała. PCJ - Java library for highly scalable HPC and Big Data processing. In *Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS 2018), Orléans, France, July 16-20, 2018*, pages 12–20. IEEE, 2018. doi:10.1109/HPCS.2018.00017.
- [3] Marek Nowicki. Comparison of sort algorithms in Hadoop and PCJ. *Journal of Big Data*, 7, 101(1):1–28, 2020. doi:10.1186/s40537-020-00376-9.
- [4] Marek Nowicki, Łukasz Górski, and Piotr Bała. Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version). *Concurrency and Computation-Practice & Experience*, 35(15):1–15, 2023. doi:10.1002/cpe.6536.
- [5] Marek Nowicki, Łukasz Górski, and Piotr Bała. PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads. *Journal of Big Data*, 8, 62(1):1–21, 2021. doi:10.1186/s40537-021-00454-6.
- [6] Marek Nowicki, Łukasz Górski, and Piotr Bała. Scalable computing in Java with PCJ Library. Improved collective operations. In *Proceedings of International Symposium on Grids & Clouds 2021 – PoS (ISGC2021), Taipei, March 22-26, 2021*, volume 378, 007, pages 1–19. Sissa Medialab Srl, 2021. doi:10.22323/1.378.0007.
- [7] Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, and Laila Sara Arroyo Mühr. HPV-KITE : sequence analysis software for rapid HPV genotype detection. *Briefings in Bioinformatics*, 26(2):1–11, 2025. doi:10.1093/bib/bbaf155.

Omówienie wyników

W dzisiejszym świecie taktowanie procesorów praktycznie przestało rosnąć z powodu ograniczeń fizycznych. W celu poprawy wydajności, nowoczesne urządzenia są wyposażone w wiele rdzeni, co sprawia, że programowanie równoległe jest niezbędne do pełnego wykorzystania ich potencjału obliczeniowego. Co więcej, wraz ze wzrostem złożoności obliczeń, na przykład przez wzrost wolumenu

danych czy spodziewaną dokładność wyników, pojedyncza maszyna często nie pozwala na wykonanie odpowiednich zadań. W takim przypadku wykorzystywane są systemy wielowęzłowe, takie jak klastry obliczeniowe czy superkomputery. Takie systemy wymagają dedykowanego wsparcia programowego. Tradycyjnie środowiska klastrowe opierają się na językach programowania takich jak C i Fortran oraz na narzędziach takich jak OpenMP (*Open Multi-Processing*) do obliczeń równoległych na jednym węźle i MPI (*Message Passing Interface*) do wykonywania zadań na wielu węzłach. Odpowiednio przygotowane aplikacje z wykorzystaniem tych języków pozwalają na wykonywanie bardzo wydajnych obliczeń naukowych. Jednak cena, którą się za to płaci, to trudniejszy rozwój czy mniejsza przenośność aplikacji.

Problematyka prowadzonych przeze mnie badań jest związana z programowaniem równoległym z wykorzystaniem paradygmatu PGAS (*Partitioned Global Address Space*) w środowisku wielowęzłowym z wykorzystaniem języka Java. Tematyką tą zajmuję się od czasu poprzedzającego uzyskanie stopnia doktora. Moje zainteresowania tą tematyką związane są z moim doświadczeniem z wykorzystaniem języka Java (m.in. uzyskany certyfikat *Sun Certified Programmer for the Java Platform, SE 6* w trakcie studiów magisterskich – data uzyskania certyfikatu: 16.12.2009), a także tym, że język Java od samego początku istnienia miał wbudowaną obsługę wielowątkowości na poziomie języka i biblioteki standardowej. Od pierwszej wersji języka Java (w 1996) istniała klasa `java.lang.Thread` reprezentująca wątek obliczeniowy, interfejs `java.lang.Runnable` służący do definiowania zadania, które będzie wykonywane przez wątek, oraz dostępne były podstawowe mechanizmy synchronizacji (słowa kluczowe `volatile`, `synchronized` i metody w każdym obiekcie `wait()`, `notify()` i `notifyAll()`). Równoległość była dalej rozwijana w kolejnych wersjach języka Java. W wersji 1.5 (inaczej nazywana wersją 5) zdefiniowany został nowy model pamięci *Java Memory Model* (JMM) oraz wprowadzono pakiet `java.util.concurrent` wraz z klasami narzędziowymi, takimi jak `Semaphore`, `CountDownLatch`, `ConcurrentHashMap`, `Executor` i `Future`. W kolejnych wersjach dodawano nowe funkcjonalności równoległe: Java 7 wprowadziła *Fork/Join Framework* (model obliczeń oparty na dzieleniu zadań – *divide-and-conquer*, z mechanizmami podkradania zadań – *work-stealing*), Java 8 wprowadziła *strumienie danych* (*Java Streams*) i możliwość ich przetwarzania równoległego z wykorzystaniem wyrażeń lambda, a także nowe klasy narzędziowe jak `CompletableFuture`, a wersja Java 21 ostatecznie wprowadziła lekkie wątki wirtualne, których wykonanie w kolejnych wersjach ulega optymalizacji. Ponadto, począwszy od wersji 16, rozwijany jest projekt *Panama*, którego jednym z kluczowych elementów jest *Vector API* (JEP 529; *JDK Enhancement Proposal*) – międzyplatformowy interfejs pozwalający na wykonywanie obliczeń wektorowych na procesorze. Rozwiązanie to ma umożliwić definiowanie operacji numerycznych w sposób pozwalający kompilatorowi na ich automatyczne odwzorowanie w najbardziej efektywne instrukcje wektorowe dostępne na danej architekturze wykonawczej, co może przełożyć się na istotny wzrost wydajności w porównaniu z obliczeniami skalarnymi. Wszystkie te funkcjonalności oparte są na równoległości jednowęzłowej – na pojedynczej maszynie, a dokładniej na pojedynczej wirtualnej maszynie Java (JVM, ang. *Java Virtual Machine*).

Wybór języka Java wynikał nie tylko z mojego dotychczasowego doświadczenia, ale także z popularności tego języka i szerokiej bazy jego użytkowników, w szczególności w dziedzinie analizy danych. Istotnym czynnikiem była również jego przenośność – możliwość uruchamiania tego samego programu na bardziej zaawansowanym sprzęcie komputerowym bez konieczności mozolnego dostosowywania kodu do wersji kompilatora czy czasochłonnej kompilacji. Wydajność obliczeń może być ponadto zwiększana dzięki wykorzystaniu nowszych i lepiej zoptymalizowanych wersji JVM. Dodatkowo, mechanizmy środowiska JVM, takie jak *garbage collector*, silne typowanie oraz

brak wskaźników i ręcznego zarządzania pamięcią, umożliwiając tworzenie programów łatwiejszych w utrzymaniu.

Tematem mojej pracy badawczej związanej z rozprawą doktorską było opracowanie nowych metod programowania równoległego w języku Java w oparciu o paradygmat PGAS. W ramach tych badań od podstaw przygotowałem bibliotekę PCJ (*Parallel Computing in Java*) dla języka Java pozwalającą na wykonywanie obliczeń równoległych w środowisku wielowęzłowym i wielordzeniowym. Biblioteka PCJ jest otwartoźródłową biblioteką dla języka Java dostępną na licencji BSD. Nie wymaga ona żadnych modyfikacji języka Java, żadnych dodatkowych kompilatorów ani nie zależy od żadnej zewnętrznej biblioteki. Biblioteka ta pozwala na uruchamianie kodu na klastrach obliczeniowych i superkomputerach, ale także ze względu na wysoką przenośność kodu i motto języka Java *napisz raz, uruchom wszędzie* (ang. *Write Once, Run Anywhere*), na uruchomienie tego samego kodu na stacji roboczej, komputerze stacjonarnym czy nawet laptopie. Po uzyskaniu stopnia doktora, w toku badań naukowych, biblioteka jest przeze mnie dalej rozwijana. Biblioteka w aktualnej wersji (5.3.3) działa na każdej JVM wspierającej język Java 8 i powyżej, aż do najnowszych wersji.

Dzięki bibliotece PCJ możliwe jest badanie paradygmatu programowania PGAS w oparciu o rozwiązania stworzone w języku Java. Kolejne wersje biblioteki były wykorzystywane w badaniach, które posłużyły do przygotowania następujących rozpraw doktorskich: *Adaptacja wybranych równoległych algorytmów do modelu PGAS i ich implementacja w języku Java z wykorzystaniem biblioteki PCJ* autorstwa mgr. Łukasza Górskiego, któremu stopień doktora został nadany w 2017 roku przez Radę Naukową Instytutu Podstaw Informatyki Polskiej Akademii Nauk (IPI PAN) w dziedzinie nauk technicznych w zakresie informatyki, oraz *Opracowanie w modelu PGAS wybranych, równoległych algorytmów grafowych i ich implementacja przy użyciu języka Java* autorstwa mgr. Magdaleny Ryczkowskiej, której stopień doktora w tej samej dziedzinie ta sama Rada nadała w 2018 roku.

Do tej pory nie istniała udana realizacji modelu programowania PGAS dla języka Java, choć rozwiązania bazujące na paradygmacie PGAS zyskują popularność. Wśród nich można wymienić biblioteki SHMEM [O1], Global Arrays [O2] i Charm++ [O3] czy języki takie jak UPC [O4], Fortran [O5], Chapel [O6] i X10 [O7]. Pewnym podejściem do użycia języka Java w modelu PGAS był język Titanium [O8], ale nie wykorzystywał on JVM, tylko tłumaczył kod Titanium do języka C++ i dopiero wygenerowany kod był kompilowany do postaci wykonywalnej. Inną potencjalną implementacją PGAS dla języka Java jest biblioteka APGAS oferująca rozwiązania programistyczne podobne do X10 [O9].

We wniosku habilitacyjnym przedstawiam cykl powiązanych ze sobą artykułów naukowych poświęconych problematyce obliczeń równoległych wielowęzłowych z wykorzystaniem języka Java. Cykl ten dokumentuje i dowodzi istotne osiągnięcia naukowe w dziedzinie nauk inżyniersko-technicznych, w dyscyplinie informatyka techniczna i telekomunikacja. Cykl ten zbiorczo zatytułowałem *Paradygmat PGAS programowania równoległego w języku Java: rozwój i analiza biblioteki PCJ na przykładzie wybranych algorytmów, w tym bioinformatycznych*. Zawarte w cyklu prace przedstawiają rozwój i zastosowania biblioteki PCJ jako implementacji paradygmatu PGAS w języku Java oraz analizują algorytmy równoległe zaimplementowane z jej wykorzystaniem. Wśród omawianych algorytmów znajduje się m.in. algorytm TeraSort, służący do sortowania wielkich zbiorów danych, wraz z analizą jego różnych wariantów implementacji, a także algorytmy zastosowane do praktycznego rozwiązywania problemów bioinformatycznych, w tym wyszukiwania wirusów w sekwencjach DNA/RNA. Badania te przyczyniły się do rozwoju obszaru badawczego programowania równoległego w ramach dyscypliny informatyka techniczna, ze szczególnym uwzględnieniem zastosowań w naukach przyrodniczych.

4.1 Omówienie poszczególnych prac

Moja rozprawa doktorska dotyczyła biblioteki PCJ w wersji 3. W trakcie studiów doktoranckich powstały dwie wcześniejsze wersje. Szczegółową historię tego rozwoju, aż do wstępnej wersji *5-alpha*, można znaleźć w pracy [A1] (w języku polskim).

PCJ 1

Pierwsza, prototypowa wersja biblioteki powstała w listopadzie 2011 roku. Biblioteka została opracowana w oparciu o najnowszą w ówczesnym czasie wersję języka Java 7. Początkowo zawierała jedynie podstawowe mechanizmy komunikacyjne, takie jak pobieranie danych (`get`), wysyłanie danych (`put`) oraz synchronizację wykonania wątków (`sync`). W późniejszym etapie dodałem mechanizm rozgłaszania wiadomości (`broadcast`) i tworzenie podgrup wątków. Operacje `put`, `get` i `broadcast` były jednostronne i asynchroniczne. Operacje `put` i `broadcast` były dodatkowo nieblokujące, dlatego dodałem również nieblokujący wariant metody `get`, który umożliwiał kontynuowanie pracy wątku w trakcie pobierania danych. Wprowadziłem również mechanizm pozwalający sprawdzać, czy nastąpił komunikat mogący modyfikować lokalną zmienną współdzieloną (`monitor`, `waitFor`).

Wersja ta zakładała wykonywanie tylko jednego wątku PCJ na pojedynczej JVM.

PCJ 2

Druga wersja, której rozwój rozpoczął się w sierpniu 2012 roku, zmodyfikowała podstawową koncepcję, umożliwiając uruchamianie wielu wątków PCJ na jednym węźle. Zmiana ta spowodowała brak wstecznej kompatybilności, ponieważ na jednej JVM mogło działać więcej wątków PCJ, co wiązało się ze współdzieleniem pól statycznych klas – co było wcześniej niemożliwe. Problem współdzielenia pól statycznych klas rozwiązałem poprzez zastosowanie własnych ładowarek klas (ang. *class loaders*) dla poszczególnych wątków PCJ.

Dodatkowo nieblokujący wariant metody `get` zastąpiłem metodą `getFutureObject`, która samodzielnie tworzyła i zwracała obiekt `Future`. W metodach `get`, `getFutureObject` i `put` dodałem opcjonalny parametr umożliwiający korzystanie z indeksów tablicowej zmiennej współdzielonej. Wprowadziłem także metody `getLocal` i `putLocal`, które pozwalały lokalnie na korzystanie z referencji obiektów współdzielonych, a nie ich pełnych kopii.

PCJ 3

Trzecia wersja biblioteki wprowadziła uporządkowanie nazw metod (np. `barrier` zamiast `sync`, `myId` zamiast `myNode`, czy `threadCount` zamiast `numNodes`). W tej wersji zaimplementowano aplikacje stanowiące część zestawu *HPC Challenge Benchmark Suite*. Biblioteka PCJ w wersji 3 zdobyła nagrodę *HPC Challenge Class 2 Award: Best Productivity – Most Elegant Solution* na międzynarodowej konferencji *SC14 – The International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing 2014)*. Nagroda została przyznana na podstawie opinii ekspertów oceniających implementację podzbioru benchmarku *HPC Challenge* przy użyciu biblioteki PCJ, za prezentację efektywnego i eleganckiego sposobu programowania aplikacji równoległych.

4.1.1 PCJ-BLAST [1] – PCJ 4.1.0

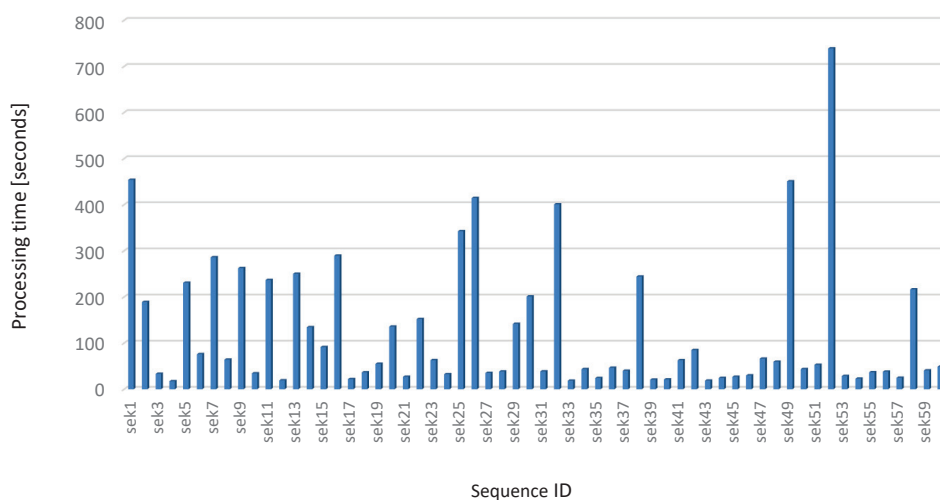
Biblioteka PCJ w wersji 4.1.0 była podstawą rozwiązania opisanego w pierwszej pracy z cyklu [1], która jest rozszerzoną wersją pracy [A2]. W pracy tej opisaliśmy rozwiązanie nazwane *PCJ-BLAST*. Jest to aplikacja służąca do dopasowywania sekwencji biologicznych zaimplementowana w języku Java z wykorzystaniem biblioteki PCJ. Aplikacja pozwala na uruchomienie wielu niezmodyfikowanych instancji *NCBI-BLAST* w sposób równoległy na systemach wielowęzłowych z możliwością automatycznego przetwarzania uzyskanych wyników. Równoległość tego rozwiązania bazuje na dystrybucji sekwencji wejściowych pomiędzy wykorzystane węzły. Sekwencje wejściowe są następnie wyszukiwane w bazie referencyjnej.

PCJ 4

Czwarta wersja biblioteki rozwijana była w trakcie realizacji projektu HPDCJ (Heterogeneous Parallel and Distributed Computing with Java), w którym pełniłem funkcję koordynatora technicznego pakietu roboczego (ang. *technical coordinator of the work package*). Rozwój wersji 4.0 rozpoczął się od kwestii porządkowych, takich jak zmiana nazw pakietów z *pl.umk.mat.pcj.** na *org.pcj.**. W tej wersji dodałem również metodę umożliwiającą wykonywanie operacji *porównaj-i-ustaw* (CAS; ang. *compare-and-set*). Wprowadziłem też modyfikacje pozwalające na wykorzystanie protokołu SDP (*Socket Direct Protocol*), obsługującego połączenia strumieniowe w infrastrukturze *InfiniBand*. W ramach projektu partnerzy prowadzili prace nad wykonywaniem obliczeń GPU z wykorzystaniem języka Java i biblioteki PCJ. Ze względu na ograniczenie JVM polegające na możliwości tylko jednorazowego ładowania biblioteki natywnej, w kolejnych wersjach biblioteki starałem się opracować mechanizm umożliwiający ich ładowanie i wspólne wykorzystywanie przez różne wątki PCJ.

Jedno z rozwiązań opierało się na adnotacji klasy zawierającej odwołania do metod natywnych, by poinformować mechanizm ładowania klas, że ma być zastosowana systemowa ładowarka klas. Podejście to nie sprawdzało się jednak w przypadku zewnętrznych bibliotek, ponieważ wymagałoby modyfikacji gotowego kodu. Alternatywnie rozważałem możliwość wskazania w parametrach uruchomieniowych JVM, które pakiety lub klasy mają być ładowane przez systemową ładowarkę klas. To rozwiązanie okazało się bardziej praktyczne, choć podatne na błędy, gdyż wymagało od użytkownika wiedzy o wszystkich nazwach pakietów wykorzystywanych przez daną bibliotekę. Dodatkowo, w sytuacjach, gdy do biblioteki przekazywano obiekty, których klasy nie były załadowane przez systemową ładowarkę, mogły występować błędy w trakcie działania aplikacji.

Ostatecznie, w wersji 4.1 zrezygnowałem z obsługi własnych ładowarek klas dla poszczególnych wątków PCJ. Dodatkowym argumentem za tą zmianą był brak możliwości całkowitego rozdzielenia przestrzeni klas – klasy ze standardowej biblioteki musiały być ładowane przez systemową ładowarkę, wspólną dla wszystkich wątków PCJ działających na jednej JVM. W efekcie pewne klasy i ich pola statyczne były i tak współdzielone. Najważniejszym skutkiem usunięcia własnych ładowarek klas jest natomiast to, że pola statyczne wszystkich klas są współdzielone w obrębie jednej maszyny JVM, a nie tylko klas systemowych. Ponadto, w wersji 4.1 biblioteki po raz pierwszy zastosowałem konstrukcje składniowe i funkcje języka Java 8, co wiązało się z koniecznością uruchamiania aplikacji tworzonych z jej wykorzystaniem w tym środowisku.



Rysunek 1: Czas przetwarzania pojedynczych próbek. (z [A2])

Dane wejściowe powstały z wykorzystaniem urządzenia *Next-Generation Sequencing* (*Illumina*) i składają się z miliona krótkich (do 150 znaków) sekwencji w formacie FASTA. PCJ-BLAST wykorzystuje oryginalne, niezmodyfikowane dane wejściowe. Przetwarzanie każdej z wejściowych sekwencji może trwać od kilku sekund do kilku minut (Rysunek 1) – czasu analizy nie można łatwo oszacować na podstawie budowy samej sekwencji. Z tego względu zastosowany algorytm bazuje na dynamicznym podziale danych wejściowych między węzły wykorzystane do obliczeń. Decyzja, do którego węzła trafi kolejna porcja sekwencji do przetworzenia jest podejmowana na podstawie aktualnego obciążenia pracą.

Równoległe przetwarzanie sekwencji wykorzystuje wiele instancji NCBI-BLAST na węzeł. Sam NCBI-BLAST daje możliwość przetwarzania danych wejściowych w sposób równoległy z wykorzystaniem wątków, co należało uwzględnić przydzielając liczbę wątków, na których mają być prowadzone obliczenia. Wątek PCJ #0 (*Scatterer*) czyta wejściowy plik FASTA i dystrybuuje sekwencje do pozostałych wątków. Pozostałe wątki PCJ (*Workers*) uruchamiają NCBI-BLAST na przekazanych sekwencjach i przetwarzają uzyskiwane wyniki do odpowiedniego formatu. *Scatterer* wysyła zestaw sekwencji do dostępnych procesów *Worker*, a jeśli takich nie ma, czeka, aż się pojawią. Po zakończeniu obliczeń proces *Worker* przetwarza wynik (format XML) do postaci odpowiedniej do dalszej obróbki danych i zapisuje je do swojego pliku wynikowego (np. jako plik CSV). Cała komunikacja między wątkami PCJ wykonywana jest asynchronicznie. Rysunek 2 przedstawia schemat działania algorytmu.

Poza przedstawieniem samego algorytmu, w pracy omówiłem wyniki wydajnościowe. Sprawdziłem dla jakiej liczby wątków NCBI-BLAST i dla jakiej liczby procesów NCBI-BLAST na węzeł czas wykonania jest najkrótszy. Okazało się, że na wykorzystywanym klastrze HPC *Topola* (ICM UW; każdy węzeł z 28-rdzeniowym procesorem Intel Xeon E5-2697 v3 @ 2.60 GHz i przynajmniej 64 GB pamięci RAM) uruchomienie 4 procesów NCBI-BLAST, każdy po 7 wątków, jest najbardziej wydajne.

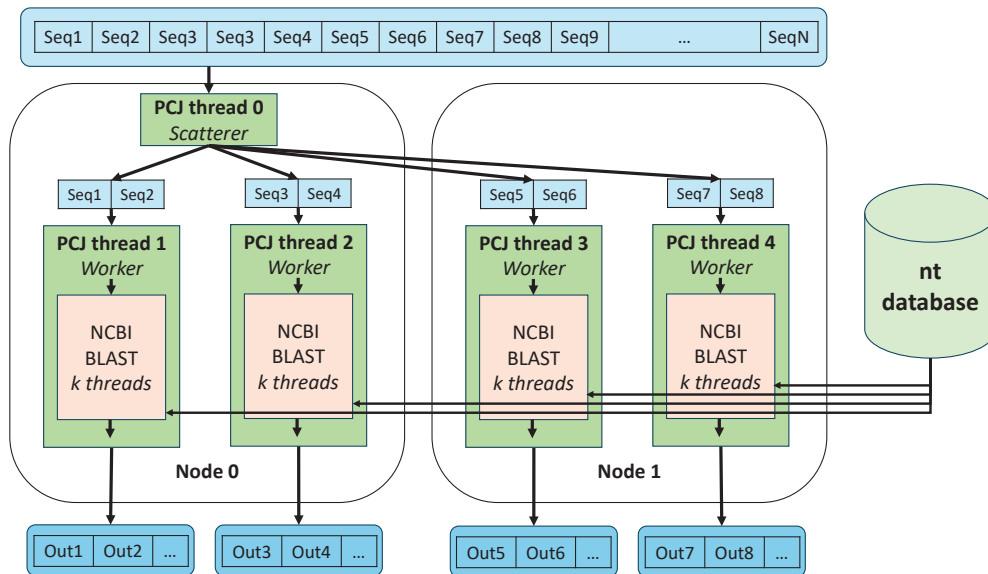
Testy wydajnościowe wykonane na 64 węzłach (1792 rdzeniach) pokazały, że efektywność równoległości osiąga ponad 70%: czas analizy 1 066 288 sekwencji zmniejszył się ze spodziewanych 52 dni przy użyciu pojedynczego 28-rdzeniowego węzła, do niecałych 27 godzin używając 64 węzłów.

W pracy porównałem też wydajność stworzonego przeze mnie rozwiązania z rozwiązaniem bazującym na podziale bazy referencyjnej na superkomputerze *Okeanos* (ICM UW; Cray XC40). Węzły tego komputera posiadają po dwa 12-rdzeniowe procesory Intel Xeon E5-2690 v3 @ 2.60 GHz (z możliwością włączenia *hyper-threading*, co daje 24 rdzenie logiczne; łącznie 48 wątków obliczeniowych) i 128 GB pamięci RAM.

Liczba węzłów	Liczba rdzeni	NCBI-BLAST		PCJ-BLAST	
		baza podzielona na 4 części		4 NCBI-BLAST na węzeł	
		Czas wykonania w minutach	Efektywność równoległa	Czas wykonania w minutach	Efektywność równoległa
1	48	1470.5	1.000	756.0	1.000
2	96	736.4	0.998	379.0	0.997
4	192	378.8	0.970	190.8	0.990
8	384	216.1	0.851	96.3	0.981
16	768	113.7	0.808	49.5	0.955
32	1536	60.3	0.762	26.4	0.893
64	3072	32.2	0.714	14.1	0.837
128	6144	21.2	0.542	7.8	0.755

Tabela 1: Czas wykonania i efektywność równoległa dla PCJ-BLAST oraz rozwiązania opartego na dzieleniu bazy referencyjnej.

Początkowo bazę dzieliłem na 32 w przybliżeniu równe bloki, ale czas przetwarzania w takiej konfiguracji był długi – przy 8 węzłach było to ok. 315 minut dla przetwarzania pierwszych 12 288 sekwencji. Przy podziale bazy na 4 części wydajność była lepsza, bo czas przetwarzania tych samych danych na 8 węzłach zmniejszył się do 216 minut. Rozwiązanie oparte o podział bazy wykazało bardzo dobrą skalowalność (dwukrotne zwiększenie liczby węzłów zmniejsza czas wykonywania obliczeń blisko dwukrotnie), ale czas przetwarzania był dwukrotnie wyższy niż przy zastosowaniu PCJ-BLAST, gdzie czas przetwarzania tych samych danych przy użyciu 8 węzłów wynosił 97 minut. Było to



Rysunek 2: Schemat działania algorytmu. (z [1])

związane głównie z tym, że PCJ-BLAST implementuje dynamiczny mechanizm równoważenia pracy. Szczegółowe wyniki są przedstawione w Tabeli 1.

W ramach badań będących podstawą niniejszej pracy, poza samym rozwojem biblioteki PCJ, zaprojektowałem i zaimplementowałem mechanizm przygotowywania i równoległego przetwarzania danych wejściowych i zapisu danych wyjściowych. Ponadto zaimplementowałem mechanizm wyrównujący obciążenie poszczególnych składników systemu. Wykonałem również testy wydajnościowe swojego rozwiązania i innych rozwiązań porównywanych w pracy. Byłem odpowiedzialny za przygotowanie opisu przedstawianego rozwiązania i wykonanych badań a także wyników wydajnościowych wraz z dyskusją osiągniętych rezultatów.

PCJ 5.0

Rozwój wersji 5.0 biblioteki PCJ rozpoczął się w czerwcu 2016 roku w ramach projektu HPDCJ. Tworząc piątą wersję biblioteki, postanowiłem całkowicie usunąć elementy związane z własnymi ładowarkami klas, które w czwartej wersji zostały jedynie ukryte i pozostawały nieaktywne. Nie była to jednak wersja wyłącznie porządkująca.

Wśród głównych, choć dla użytkownika niewidocznych, zmian znalazła się przebudowa całego mechanizmu komunikacji między węzłami. Była to modyfikacja o charakterze wręcz rewolucyjnym. Umożliwiła uruchamianie aplikacji wykorzystujących bibliotekę PCJ na ponad 1024 węzłach obliczeniowych. Dodatkowo zmiana architektury wewnętrznej pozwoliła na przesyłanie komunikatów o rozmiarze przekraczającym 2 GB, co w poprzednich wersjach prowadziłoby do błędów. Nowa struktura przekazywania wiadomości ułatwiła także wprowadzanie nowych typów komunikatów oraz umożliwiła prostą implementację mechanizmów powiadamiania o zakończeniu operacji komunikacyjnych. W rezultacie wersja 5 przyniosła również szereg zmian w programistycznym interfejsie aplikacji (API, ang. *Application Programming Interface*).

Najważniejszą ze zmian było usystematyzowanie podstawowych metod biblioteki: `get`, `put`, `broadcast` oraz `barrier`. Wcześniej metody te miały spójne nazwy, lecz różniły się zachowaniem: `get` i `barrier` były metodami synchronicznymi, natomiast `put` i `broadcast` – asynchronicznymi, umożliwiającymi dalsze działanie aplikacji bez oczekiwania na potwierdzenie odbioru wiadomości. W nowej wersji zachowania wszystkich tych metod zostało ujednolicone: działają synchronicznie. Dodatkowo dla każdej z nich dodano odpowiednik asynchroniczny: `asyncGet`, `asyncPut`, `asyncBroadcast` oraz `asyncBarrier`. Wersje asynchroniczne stały się podstawowymi, natomiast metody synchroniczne, dostępne wyłącznie w głównej klasie `org.pcj.PCJ`, stanowią nakładki (ang. *wrapper*) wywołujące odpowiednie wersje asynchroniczne. Przykładowo, wywołanie `PCJ.barrier()` realizowane jest poprzez uruchomienie metody asynchronicznej i czekania w metodzie blokującej zwróconego obiektu typu `Future<Void>`: `PCJ.asyncBarrier().get()`. Zastosowanie obiektów typu `Future` umożliwiło ponadto czasowe oczekiwanie na zakończenie operacji.

Wprowadziłem również zmiany w zakresie parametrów przekazywanych do metod biblioteki, tak aby były one możliwie najbardziej intuicyjne i spójne. Od wersji 5 nazwa zmiennej znajduje się bezpośrednio przed opcjonalnymi parametrami indeksów tablicowych, a wcześniej wskazywany jest identyfikator wątku, do którego zmienna należy. Ponieważ parametr opcjonalny musi występować na końcu listy argumentów,

przekazywana wartość w metodach `put` i `broadcast` występuje jako pierwszy argument, np. `PCJ.put(newValue, threadId, variableName[, indices...])`.

Kolejna istotna zmiana dotyczyła sposobu definiowania nazw zmiennych współdzielonych. We wcześniejszych wersjach biblioteki nazwy te były reprezentowane jako ciągi tekstowe (ang. *String*), co wiązało się z ryzykiem popełnienia literówki, brakiem wsparcia dla refaktoryzacji i ograniczeniem możliwości korzystania z wielu klas pełniących rolę *magazynów* zmiennych współdzielonych. W nowej wersji zdecydowałem o zastąpieniu nazw tekstowych stałymi typów wyliczeniowych, co znacząco ułatwiło utrzymanie i refaktoryzację kodu, a także zmniejszyło ryzyko błędów. Dodatkowo, zaimplementowany w bibliotece procesor adnotacji umożliwia wykrywanie potencjalnych niezgodności już na etapie kompilacji, a nie dopiero podczas wykonywania programu.

W bibliotece PCJ 5.0 wprowadziłem również obsługę wyjątków mogących wystąpić podczas przesyłania zmiennych współdzielonych – w przypadku błędu w wątku docelowym informacja o nim jest przekazywana do wątku źródłowego poprzez obiekt typu *Future*, przechowujący stan operacji asynchronicznej.

Ponadto operacja *compare-and-swap* (CAS), wprowadzona w PCJ 4, została usunięta ze względu na niską wydajność w sytuacjach silnego współzawodnictwa między wątkami, wynikającą z konieczności przesyłania wielu drobnych komunikatów sieciowych przy próbie zastosowania klasycznego wzorca pętli CAS. W trakcie prac nad nową wersją rozważałem możliwość przeniesienia obliczania nowej wartości zmiennej na wątek docelowy zamiast źródłowego, co mogłoby ograniczyć liczbę przesyłanych komunikatów. Ostatecznie, zamiast implementować to rozwiązanie wyłącznie dla operacji CAS, zdecydowałem się wprowadzić bardziej ogólny mechanizm. W wersji 5.0.5 biblioteki PCJ dodałem operację `asyncAt`, umożliwiającą zlecenie dowolnych zadań do wykonania w kontekście zdalnego wątku PCJ.

Wcześniejsze wersje (od 5.0.0 do 5.0.4) stanowiły etapy rozwojowe, w których koncentrowałem się głównie na poprawkach i optymalizacji działania biblioteki.

4.1.2 Tutorial [2] – PCJ 5.0.6

Niejako równolegle, poza tworzeniem aplikacji wykorzystujących bibliotekę PCJ, podejmujemy działania mające na celu jej upowszechnienie i przedstawienie szerszemu gronu odbiorców.

Biblioteka PCJ jest wykorzystywana w procesie dydaktycznym m.in. na studiach *Inżynierii Obliczeniowej* prowadzonych przez Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego Uniwersytetu Warszawskiego, w ramach przedmiotu *Programowanie równoległe*. Jest również nauczana na studiach informatycznych prowadzonych na Wydziale Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika w Toruniu, w ramach przedmiotu *Wprowadzenie do obliczeń równoległych*, będącym przedmiotem obowiązkowym na studiach drugiego stopnia kierunku *informatyka* dla specjalności *Obliczenia wielkoskalowe*, a także – przed zmianą planów i programów studiów – w ramach przedmiotu *Programowanie równoległe i rozproszone*, oferowanego jako przedmiot do wyboru dla studentów studiów pierwszego stopnia kierunku *informatyka* oraz obowiązkowego dla studentów drugiego stopnia kierunku *informatyka*.

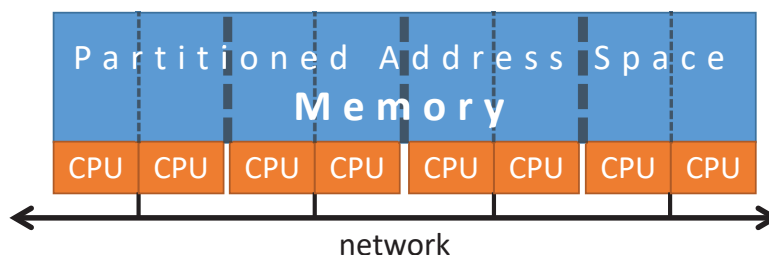
Inną formą popularyzacji biblioteki jest prowadzenie szkoleń (ang. *tutorials*) prezentujących jej możliwości oraz umożliwiających uczestnikom szybkie zdobycie wiedzy i umiejętności potrzebnych do praktycznego wykorzystania PCJ. Druga omawiana w cyklu praca [2] jest publikacją powstałą

w związku ze szkoleniem przeprowadzonym podczas konferencji *High Performance Computing & Simulation* (HPCS) w 2018 roku. Do przygotowania i poprowadzenia tego szkolenia oraz opracowania publikacji zostaliśmy zaproszeni przez organizatorów konferencji. Wcześniej współuczestniczyłem w przygotowaniu koncepcji oraz prowadzeniu warsztatów podczas konferencji *Parallel Processing & Applied Mathematics* (PPAM'15, PPAM'17) oraz *Supercomputing Frontiers Europe 2018*. W omawianej pracy byłem odpowiedzialny za przygotowanie prawie całego opisu biblioteki PCJ – w tym ilustracji i większości przykładowych kodów źródłowych.

PCJ 5.0.6

Wersja 5.0.6 biblioteki wprowadziła między innymi optymalizacje wykonania poprzez zastosowanie puli obiektów `ByteBuffer` do obsługi komunikatów sieciowych. Była to pierwsza wersja biblioteki udostępniona nie tylko za pośrednictwem strony domowej projektu, lecz także w głównym repozytorium pakietów dla języka Java – *Maven Central Repository*. W tym repozytorium pojawiła się w październiku 2018 roku.

Praca omawia podstawy oraz sposób działania biblioteki PCJ w wersji 5.0.6. Publikacja, choć nawiązuje do warsztatów, nie stanowi bezpośredniego zapisu ich przebiegu ani pełnych kodów aplikacji tworzonych przez uczestników. Zawiera natomiast zwięzłe omówienie zagadnień prezentowanych podczas zajęć.

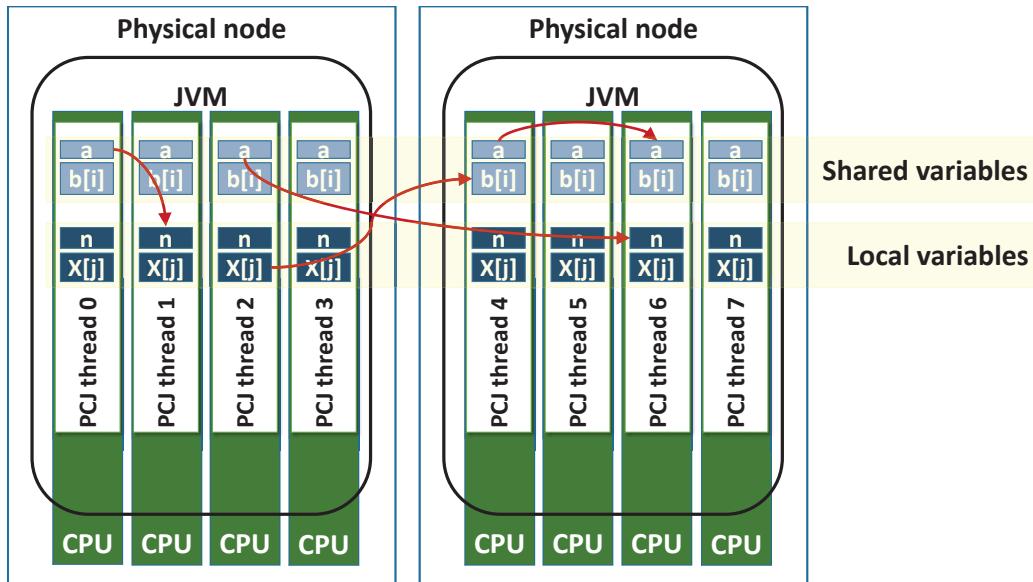


Rysunek 3: Widok pamięci w modelu PGAS. Każdy węzeł obliczeniowy, który może składać się z wielu procesorów (CPU), posiada własną pamięć. Węzły obliczeniowe są połączone ze sobą poprzez sieć. Przestrzeń adresowa pamięci wszystkich węzłów jest koncepcyjnie traktowana jako jedna wspólna, globalna przestrzeń adresowa. (z [3])

Praca zaczyna się od krótkiego wprowadzenia do paradygmatu PGAS (Partitioned Global Address Space), gdzie widok globalnej pamięci jest realizowany przez podzielenie jej między wątki (jednostki) wykonujące obliczenia (Rysunek 3). Pamięć jest podzielona na część prywatną i współdzieloną. Pamięć prywatna jest dostępna tylko dla właściciela – zawiera jego lokalne zmienne. Pamięć współdzielona jest pamięcią, do której mogą odwoływać się inne wątki. W zależności od języka programowania nie mamy bezpośredniego dostępu do obszaru pamięci, który jest prywatny lub współdzielony, lecz jedynie mamy dostęp do nazw zmiennych. W takiej sytuacji domyślnie wszystkie zmienne są prywatne, jednak część z nich można oznaczyć jako współdzielone. Zmienne współdzielone są dostępne dla wszystkich wątków – ich stan może być odczytany i modyfikowany zdalnie.

W pracy przedstawiliśmy wprowadzenie do udostępnionego przez bibliotekę PCJ interfejsu programistycznego (API). Biblioteka PCJ bazuje na modelu obliczeń SPMD (*Single Program, Multiple Data*), gdzie ten sam program uruchomiony jest jednocześnie na n węzłach obliczeniowych, na każdym węźle uruchamiane są wątki PCJ, a każdy wątek pracuje na swoim zestawie danych; liczba wątków obliczeniowych na poszczególnych węzłach jest stała i ustalona na początku działania programu.

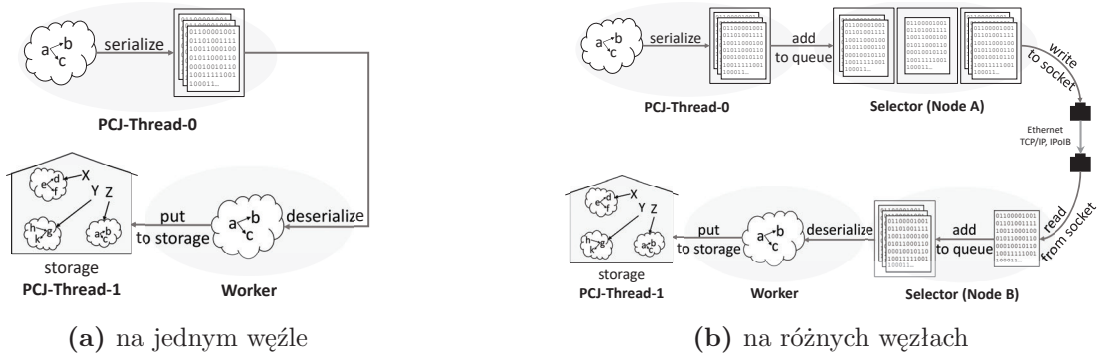
Biblioteka PCJ dostarcza niezbędne mechanizmy, takie jak numerowanie i synchronizację wątków oraz transfer danych za pomocą prostych komend – pojedynczych metod biblioteki. Pamięć



Rysunek 4: Schemat modelu obliczeniowego PCJ. Strzałki przedstawiają możliwą komunikację z wykorzystaniem metod `put()` i `get()` działających na zmiennych współdzielonych. (z [2])

współdzielona realizowana jest poprzez pola klasy zwanej magazynem (ang. *storage*). Domyślnie wszystkie pola są lokalne, a część pól klasy może być oznaczona, jako współdzielona (ang. *shared* lub *shareable*). Rysunek 4 przedstawia schemat modelu obliczeniowego w PCJ i komunikacji za pomocą metod `put()` i `get()`. Szczegóły komunikacji są ukryte przed użytkownikami, a sama komunikacja jest jednostronna i asynchroniczna.

Praca opisuje również wewnętrzne mechanizmy komunikacyjne. Jeden z ważniejszych mechanizmów jest związany z przekazaniem danych pomiędzy wątkami PCJ. Przekazanie danych w ramach jednego węzła różni się od przekazania danych pomiędzy różnymi węzłami. W przypadku przekazywania danych w ramach jednego węzła nie można po prostu przekazać wartości pola. Przekazanie takiej wartości wiązałoby się ze współdzieleniem referencji do tego samego obiektu. W związku z tym należy wykonać głęboką kopię obiektu. W bibliotece PCJ wykonywane jest to za pomocą serializacji obiektu źródłowego i deserializacji w pamięci wątku docelowego. Jest to operacja czasochłonna, ale bezpieczna – zwłaszcza w przypadku złożonych obiektów z wieloma referencjami do innych obiektów. Schemat tego procesu przedstawiony jest na Rysunku 5a. Natomiast w przypadku komunikacji pomiędzy węzłami wykonywana jest komunikacja wykorzystująca gniazda sieciowe i wątek implementujący *Selector*, którego zadaniem jest przekazywanie danych do kanałów komunikacyjnych. W tym przypadku serializacja obiektu jest naturalna. Schemat tego procesu przedstawia Rysunek 5b.



Rysunek 5: Diagram wykonania metody `PCJ.put()` pomiędzy dwoma wątkami PCJ. (z [2])

W pracy przedstawiliśmy również podstawowe konstrukcje: zrównoleglenie pętli, komunikacja typu *stencil* (wymiana danych brzegowych, ang. *halo exchange*), redukcja. Dla podanych konstrukcji pokazaliśmy i omówiliśmy też wyniki wydajnościowe implementacji przykładowych problemów je zawierających. Zrównoleglenie pętli zaprezentowaliśmy na przykładzie problemu przybliżenia wartości π metodą *Monte Carlo*. Komunikacja typu *stencil* została zaprezentowana za pomocą implementacji algorytmu wygładzania krzywych Savitzky’ego-Golay’a [O10]. Redukcja została zaprezentowana z wykorzystaniem problemu *WordCount*, którego zadaniem jest zliczenie liczby wystąpień poszczególnych słów w tekście.

Dzięki przedstawionym konstrukcjom można w prosty sposób implementować bardziej zaawansowane algorytmy rozwiązujące złożone problemy.

4.1.3 Cloud [4] – PCJ 5.0.8

Do tej pory pokazywaliśmy działanie aplikacji stworzonych z wykorzystaniem biblioteki PCJ implementującej paradygmat PGAS na klastrach obliczeniowych i superkomputerze. Jednak w związku z coraz większą popularnością platform chmurowych, postanowiliśmy zweryfikować możliwość uruchamiania aplikacji wykorzystujących PCJ w takiej infrastrukturze. Nasze badania na tym polu przedstawiliśmy w pracy [4] będącej wersją rozszerzoną pracy [A3] przedstawionej na warsztatach *Second International Workshop on Parallel Programming Models in High-Performance Cloud (ParaMo 2020)* organizowanych w ramach konferencji *26th International European Conference on Parallel and Distributed Computing (Euro-Par 2020)* w Warszawie (ze względu na COVID-19 referaty przedstawiane były zdalnie). Niejako przy okazji zweryfikowaliśmy poprawność i wydajność działania aplikacji na różnych procesorach typu ARM. Prace badawcze prowadzone były z wykorzystaniem biblioteki PCJ w wersji 5.0.8.

PCJ 5.0.8

W bibliotece PCJ w wersji 5.0.8 oprócz poprawek dotyczących działania mechanizmów sieciowych, istotną modyfikacją była zmiana zachowania procesora adnotacji, który od tej wersji zamiast zgłaszać błąd wyświetla jedynie ostrzeżenie w przypadku pól współdzielonych o typach nieserializowalnych i nieoznaczonych jako `final`.

W ramach prac badawczych wykorzystaliśmy zasoby chmurowe dostarczane przez Amazon (*Amazon Web Services (AWS) Elastic Compute Cloud (EC2)*) i Linaro (*Linaro Developer Cloud*).

W ramach chmury AWS wykorzystaliśmy instancje *t2.xlarge* z procesorami Intel Xeon CPU E5-2686 v4 @ 2.3 GHz, gdzie do dyspozycji były 4 vCPU i 16 GB pamięci RAM. Instancje z procesorami typu ARM to instancje z procesorami AWS Graviton @ 2.3 GHz, które zawierały 64-bitowe rdzenie ARM Neoverse opracowane i zoptymalizowane przez AWS. Wykorzystaliśmy instancje *a1.xlarge* z 4 vCPU i 8 GB pamięci RAM, oraz *a1.4xlarge* i *a1.metal*, które zawierały 16 vCPU (lub 16 CPUs w przypadku wersji *metal*, gdzie bezpośrednio były używane fizyczne rdzenie) i 32 GB pamięci RAM. Wydajność sieciowa była określona jako *umiarkowana* (ang. *moderate*), co przekłada się na przepustowość sieci między instancjami na poziomie do 1 Gb/s. Jako implementację wirtualnej maszyny Java użyliśmy OpenJDK Runtime Environment Corretto-11.0.6.10.1.

W ramach chmury Linaro wykorzystywaliśmy wyłącznie procesory ARM. Większość instancji posiadała 64-bitowe procesory Cavium ThunderX 88XX @ 1.9 GHz, a różnice między nimi to liczba vCPU i pamięci RAM: *m1.tiny* (1 vCPUs, 512MB), *uk.large* (8 vCPUs, 16GB), *uk.medium* (4 vCPUs, 8GB), *uk.small* (2 vCPUs, 4GB), *uk.nano* (1 vCPUs, 2GB), *nano* (1 vCPUs, 4GB). Instancja *m1.small*

była wyposażona w 64-bitowy procesor Qualcomm Falkor @ 2.2 GHz z 2 vCPU i 1 GB pamięci RAM. Dokumentacja wskazywała, że przepustowość sieci między instancjami w chmurze wynosiła do 10 Gb/s. Użyta implementacja wirtualnej maszyny Java to OpenJDK Runtime Environment (*build 11.0.6+10-post-Debian-1deb10u1*).

W celach porównawczych wykorzystaliśmy również superkomputer *Okeanos* (ICM UW). W ramach tego superkomputera uruchamiałem aplikacje zarówno bezpośrednio na fizycznych rdzeniach, jak również przez platformę wirtualizacyjną *Singularity*. Superkomputer wykorzystuje wysokowydajną sieć komunikacyjną Cray Aries opracowaną przez firmę Cray. Wirtualna maszyna Javy użyta w tym środowisku to OpenJDK Runtime Environment (*build 11.0.4+11-suse-3.33.1-x8664*). Tabela 2 zawiera podsumowanie wszystkich wykorzystanych typów instancji.

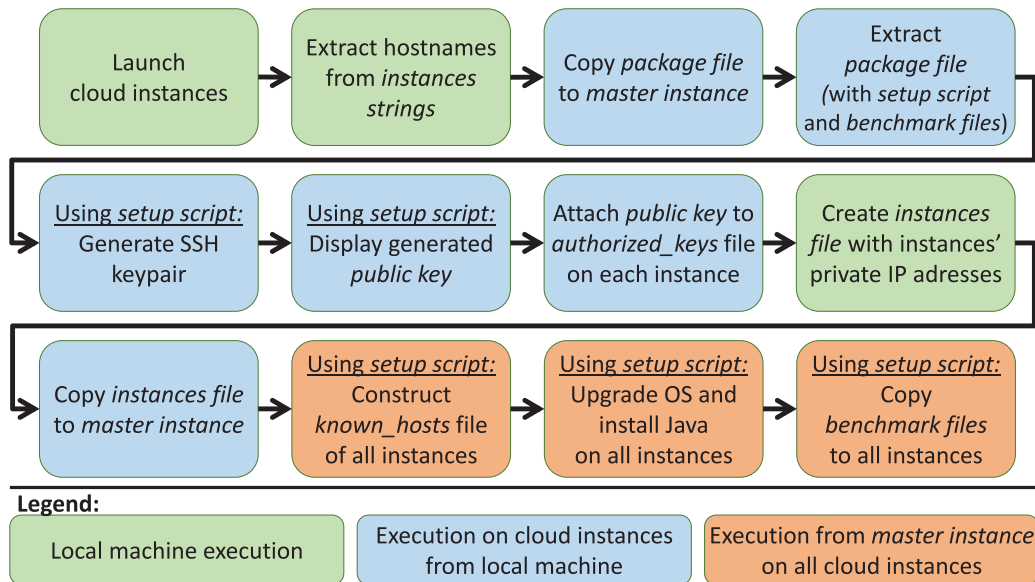
Dostawca	Typ	Procesor	CPU	vCPU	RAM
Amazon	t2.xlarge	Intel Xeon CPU E5-2686 v4 @ 2.3 GHz	-	4	16 GB
Amazon	a1.xlarge	AWS Graviton @ 2.3 GHz	-	4	8 GB
Amazon	a1.4xlarge	"	-	16	32 GB
Amazon	a1.metal	"	16	-	32 GB
Linaro	m1.small	Qualcomm Falkor @ 2.2 GHz	-	2	1 GB
Linaro	m1.tiny	Cavium ThunderX @ 1.9 GHz	-	1	512 MB
Linaro	uk.large	"	-	8	16 GB
Linaro	uk.medium	"	-	4	8 GB
Linaro	uk.small	"	-	2	4 GB
Linaro	uk.nano	"	-	1	2 GB
Linaro	nano	"	-	1	4 GB
ICM UW	bare metal	2×Intel Xeon E5-2690 v3 @ 2.60 GHz	48	-	128 GB
ICM UW	singularity	"	-	48	128 GB

Tabela 2: Podsumowanie wykorzystanych typów instancji (węzłów obliczeniowych).

Konfiguracja Wykorzystywanie zasobów chmurowych do obliczeń różni się w podstawach w stosunku do używania klastrów obliczeniowych czy superkomputerów. Przede wszystkim obrazy systemów operacyjnych są z reguły pozbawione wielu pakietów – zainstalowane są wyłącznie niezbędne pakiety a inne trzeba samodzielnie doinstalować. Dodatkowo, normalnie uruchomione instancje mające brać udział we wspólnych obliczeniach nie znają swoich nazw (ang. *hostname*), a systemy plików nie są współdzielone. W związku z tym, konieczne jest skonfigurowanie systemu operacyjnego i całego środowiska do współpracy przy wykonywaniu aplikacji.

Aby połączyć się z lokalnego komputera do instancji w chmurze należy przygotować parę kluczy: klucz publiczny i prywatny. Klucz prywatny jest przechowywany wyłącznie lokalnie i będzie służył do autentykacji w czasie łączenia się z instancją w chmurze, na której zostanie umieszczony klucz publiczny przy tworzeniu instancji. Mimo tego, że z lokalnego komputera będzie można połączyć się do wszystkich utworzonych instancji, to instancje nie posiadają poświadczeń potrzebnych do nawiązywania połączeń między sobą.

Od pierwszych wersji biblioteki istniały dwa sposoby na uruchomienie obliczeń: mechanizm `start()` oraz `deploy()`. Pierwszy ze sposobów uruchamiania obliczeń (`start()`) wymaga aby w systemie istniał mechanizm jednoczesnego uruchamiania aplikacji na węzłach obliczeniowych (instancjach). Jeśli go nie ma, biblioteka może skorzystać z drugiego sposobu uruchamiania obliczeń (`deploy()`): samodzielnie uruchamia nowe instancje z tzw. instancji głównej (ang. *master instance*). W takim



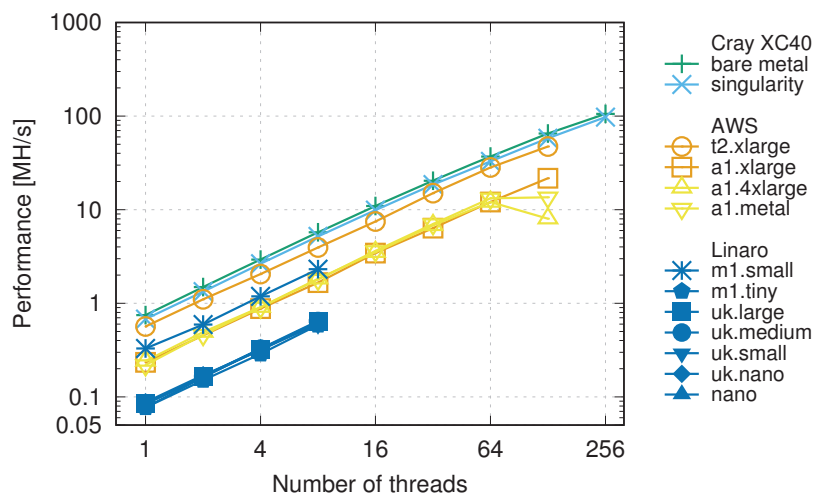
Rysunek 6: Schemat konfiguracji środowisk chmurowych. (z [4])

przypadku konfiguracja powinna pozwalać na nieinteraktywny sposób logowania poprzez SSH – za pomocą kluczy – bez pytania o hasło czy poprawność odcisku palca poszczególnych węzłów. Skopiowanie stworzonego lokalnie klucza prywatnego na instancję w chmurze nie jest bezpiecznym rozwiązaniem – w przypadku ataku typu *man-in-the-middle* klucz prywatny może ulec ujawnieniu. W związku z tym należy wygenerować parę kluczy bezpośrednio w instancji głównej. Ponadto, instancja główna musi znać adresy IP pozostałych instancji używanych w obliczeniach. Rysunek 6 przedstawia schemat konfiguracji środowisk chmurowych z podziałem na akcje wykonywane lokalnie jak i na instancjach w chmurze. Aby ułatwić konfigurację środowiska stworzyłem skrypty powłoki BASH, które mają być uruchamiane lokalnie i po nawiązaniu połączenia z instancją główną (zawartość skryptów przedstawiona jest w *Appendix A* pracy [4] jako, odpowiednio, Listing 1 i Listing 2). Dzięki wykorzystaniu przygotowanych skryptów konfiguracja środowiska na AWS EC2 z wykorzystaniem 32 instancji zajmuje mniej niż 2,5 minuty.

Warte podkreślenia jest to, że tak prosta konfiguracja wystarczyła, by z powodzeniem uruchomić aplikacje oparte o bibliotekę PCJ i wykorzystać w jednym zadaniu wiele instancji maszyn chmurowych. Omawiane w pracy badania pokazały, że biblioteka PCJ pozwala na zrównoleganie obliczeń nie tylko na klastrach obliczeniowych czy superkomputerach, ale także wykorzystując zasoby chmurowe. Ponadto pokazaliśmy, że wystarczy, by na danej maszynie istniała implementacja standardowej wirtualnej maszyny Java (JVM), by uruchamiać aplikacje stworzone w PCJ, a jeśli te maszyny połączone są w sieć, możliwe jest działanie wielowęzłowe. Możliwość uruchomienia nie ogranicza się wyłącznie do procesorów typu x86 czy do konkretnej implementacji JVM.

Do porównania wydajności przedstawiłem trzy typy aplikacji: *DES Decryption* – aplikacja intensywna obliczeniowo, *FFT* – aplikacja intensywna komunikacyjnie, *WordCount* – aplikacja typu Big Data. Taki wybór umożliwił analizę różnych aspektów wydajności aplikacji. Aby ujednolicić uruchomienia między typami instancji, maksymalny rozmiar pamięci dla JVM został ustawiony na 1 GB niezależnie od rzeczywistej ilości dostępnej pamięci. Ponadto, w przypadku każdego typu testu wybrałem najbardziej efektywne wyniki – te o najkrótszym czasie wykonania, a tym samym najwyższej wydajności.

DES Decryption Pierwszy test to *DES Decryption*. Jest to intensywny obliczeniowo, a jednocześnie trywialnie równoległy problem. W pracy, ewaluacji poddałem trzy implementacje równoległe: jednowęzłową opartą o natywne wątki Java, wielowęzłową z wykorzystaniem biblioteki PCJ oraz zoptymalizowaną implementację wielowęzłową. Wszystkie implementacje opierały się na najprostszym, siłowym sposobie ataku na szyfr, w którym badane były wszystkie możliwe kombinacje klucza deszyfrującego. Dla klucza długości n -bitów istnieje 2^n różnych kombinacji klucza. Normalnie po odnalezieniu klucza wyszukiwanie powinno być zakończone, ale w przedstawionych testach wydajnościowych sprawdzany był pełen zakres możliwych kluczy. Do kodowania i dekodowania używane były klasy i interfejsy wbudowane w bibliotekę standardową Java (m.in. `javax.crypto.Cipher` i `java.security.Key`). W zoptymalizowanej implementacji mojego autorstwa, aby zwiększyć wydajność obliczeń, zamiast tworzyć osobny obiekt dla każdego testowanego klucza, każdy wątek PCJ posiada pojedynczy obiekt reprezentujący klucz, który jest zmieniany z każdą iteracją.

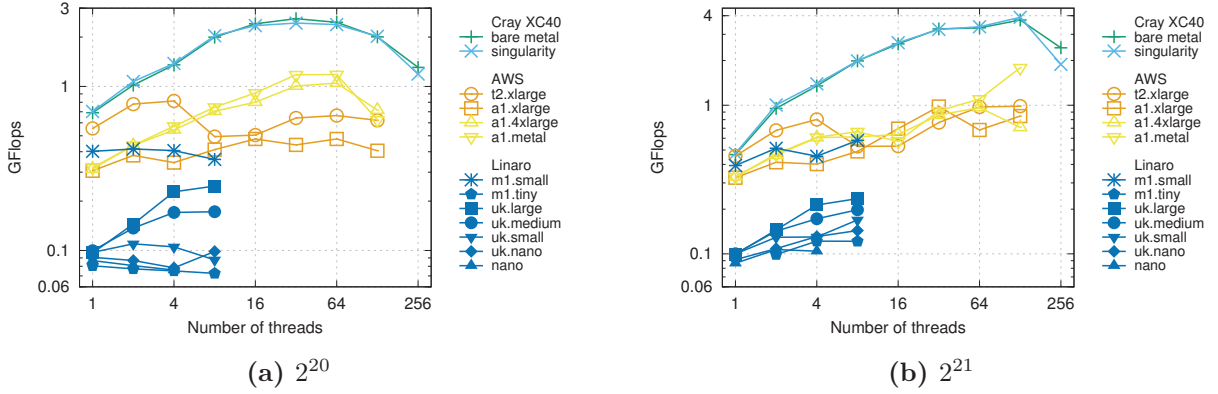


Rysunek 7: Wyniki wydajnościowe dla problemu *DES Decryption* z kluczem 26-bitowym. Wyższa wartość oznacza lepszy wynik. (z [4])

Wyniki wydajnościowe zoptymalizowanej implementacji zostały przedstawione na Rysunku 7. Wartości są podane w jednostce *mega-hashes/s* (liczba sprawdzonych kluczy na sekundę, H/s). Niezależnie od typu maszyny skalowalność rozwiązań jest bardzo dobra. Jednakże całkowita wydajność większości instancji chmury Linaro jest blisko 10-krotnie mniejsza w porównaniu do rozwiązania uruchomionego na Cray XC40 i 3-krotnie mniejsza od rezultatów uzyskanych na chmurze AWS. Różnica odzwierciedla szybkość procesora na każdym z systemów. Jedynie instancja *m1.small* z procesorem Qualcomm Falkor pokazuje wydajność zbliżoną do instancji AWS. Wydajność uzyskana na AWS jest niższa niż na Cray XC40, a różnica między instancjami *a1* a *t2* pokazuje różnicę między wykonaniami na procesorach x86 i ARM. Dodatkowo warto zauważyć, że wydajność na Cray XC40 czy to używając wirtualizacji, czy też bez niej, praktycznie jest taka sama. Pokazuje to, że wirtualizacja nie wprowadza znaczącego narzutu do aplikacji intensywnie wykorzystujących CPU.

FFT Szybkie przekształcenie Fouriera (*FFT*, ang. *Fast Fourier Transform*) zostało wybrane jako test intensywny komunikacyjnie. Wersja w PCJ została oparta na implementacji PGAS opracowanej dla Coarray Fortran 2.0 [O11] i powstała w ramach pracy doktorskiej [O12] współautora pracy – Łukasza Górskiego. Wykorzystuje ona algorytm wymiany binarnej *radix-2*, którego celem jest ograniczenie komunikacji międzyprocesowej. Schemat działania wygląda następująco: na początku lokalnie wykonywany jest algorytm *FFT* bazujący na permutacji odwróconych bitów, a następnie wykonywana jest globalna transpozycja danych z układu blokowego do układu cyklicznego, po czym

następuje kolejne lokalne obliczenie *FFT* i na koniec wykonywana jest globalna odwrotna transpozycja przywracająca dane do pierwotnego układu blokowego. Komunikacja między wszystkimi wątkami (ang. *all-to-all*) jest wykonywana w procedurze odpowiedzialnej za transpozycję i odwrotną transpozycję danych.



Rysunek 8: Wyniki wydajnościowe dla problemu *FFT*. Wartości są podane w GFlops. Wyższa wartość oznacza lepszy wynik. (z [4])

Wykresy przedstawione na Rysunku 8 przedstawiają wyniki wydajnościowe. Wartości zostały podane w jednostce *giga-flops* (liczba operacji zmiennoprzecinkowych na sekundę, ang. *number of floating-point operations per second*). Ze względu na naturę algorytmu, gdzie liczba wątków musi być potęgą dwójki, rdzenie obliczeniowe na węzłach obliczeniowych maszyny Cray XC40 zostały częściowo niewykorzystane.

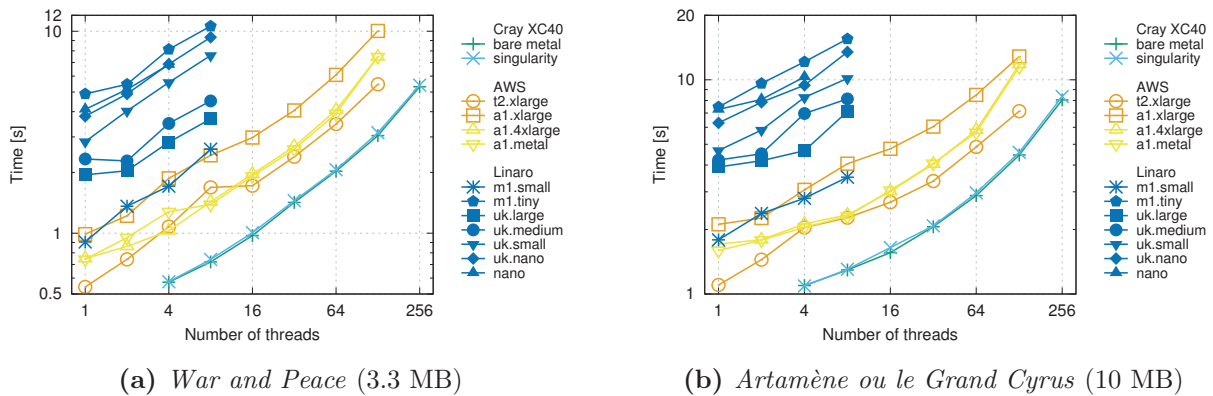
Podobnie jak w poprzednim teście wydajnościowym, rozwiązanie działające na Cray XC40 działa najwydajniej a wirtualizacja nie wprowadziła znaczącego narzutu na obliczenia. Skalowalność tego rozwiązania jest ograniczona i jest lepsza dla większych rozmiarów danych, co również pokazaliśmy w innych naszych pracach: w pracy [A4] wykonaliśmy *FFT* na danych rozmiaru 2^{26} , a w pracy [5] przedstawiliśmy wyniki dla rozmiaru 2^{30} .

Rozwiązanie uruchomione na chmurze AWS wykazuje dobrą skalowalność ale tylko do kilku wątków oraz mniejszą całkowitą wydajność. Wyniki uzyskane dla 16 wątków są podobne w przypadku procesora ARM (Graviton) i procesora x86. Dla mniejszej liczby wątków (4 na instancję) wydajność jest wyższa na architekturze x86. Jednak wraz ze wzrostem liczby instancji wydajność jest podobna, zwłaszcza dla większych danych. Warto zauważyć również, że dla większej tablicy (Rysunek 8b) i małej liczby wątków wydajność uzyskana przy użyciu Cray XC40 i chmury AWS (zwłaszcza dla instancji *t2.large*) jest zbliżona – jest to związane z tym, że obliczenia wykonane zostały na jednej instancji i nie wykorzystywały połączenia sieciowego. Odzwierciedla to charakter aplikacji *FFT*, której wydajność jest związana głównie z komunikacją, a zatem wydajność jest ograniczona przepustowością kanału komunikacyjnego, a nie szybkością procesora. Cray XC40 jest wyposażony w wydajniejszą infrastrukturę komunikacyjną między węzłami, co przekłada się na lepszą wydajność aplikacji związanych z komunikacją.

Wyniki uzyskane w ramach chmury Linaro pokazują mniejszą całkowitą wydajność. Jedyna instancja, która dorównywała instancjom uruchomionym na chmurze AWS, to *m1.small* z procesorem Qualcomm Falkor. Co więcej, wyniki z chmury Linaro pokazują, że wydajność jest lepsza w przypadku większych tablic (2^{21}), ponieważ w przypadku tej chmury decydującym czynnikiem są obliczenia a nie komunikacja. Z drugiej strony, narzut komunikacyjny ogranicza wydajność w przypadku mniejszych tablic.

WordCount Ostatnim przedstawionym testem jest aplikacja *WordCount*. Jest to szandarowa aplikacja pokazująca sposób działania modelu *map-reduce*. W tym problemie zliczana jest liczba wystąpień każdego słowa w danych wejściowych. Aplikacja działa w dwóch fazach. W pierwszej plik wejściowy jest czytany linia za linią a każda linia dzielona jest na osobne słowa tworząc w ten sposób listę par (*słowo*, 1). W drugiej fazie następuje sumowanie wyników częściowych bazując na kluczu, którym jest pierwszy element pary (*słowo*).

Implementację tego problemu z wykorzystaniem biblioteki PCJ przygotował współautor pracy – Łukasz Górski – na potrzeby wcześniejszego naszego wspólnego artykułu [A5]. W przedstawionym rozwiązaniu każda linia pliku wejściowego dzielona jest na słowa względem białych znaków i tworzona jest suma częściowa wystąpień w obiekcie (słowniku) typu `Map<String, Integer>`. Każdy wątek przechowuje swoją niezależną kopię tego obiektu. Dla prostoty, implementacja nie wykonuje żadnego przetwarzania wstępnego słów. Na koniec wykonywana jest operacja redukcji a ostateczny wynik zapisany jest w wątku #0. Fazy przetwarzania nie nakładają się na siebie – redukcja następuje dopiero, gdy wszystkie wątki zakończą czytanie danych wejściowych.



Rysunek 9: Wyniki wydajnościowe dla problemu *WordCount*. Wartości są podane w sekundach. Niższa wartość oznacza lepszy wynik. (z [4])

W testach wydajnościowych wykorzystałem zaimplementowany prosty schemat redukcji, gdzie wątek #0 pobiera po kolei z każdego wątku jego sumę częściową i wykonuje jej łączenie. Wydajność może być zwiększona korzystając z innego algorytmu redukcji, co jest przedstawione w innych naszych pracach [A5, 5, 6]. Ewaluację wydajności przeprowadziłem używając reżimu słabej skalowalności, czyli rozmiar problemu był zwiększany proporcjonalnie do liczby wątków, utrzymując stałe obciążenie jednostkowe. Inaczej mówiąc, każdy wątek przetwarza cały plik wejściowy, więc zwiększanie liczby wątków powoduje zwiększenie ilości danych do przetworzenia.

Analizę wykonałem dla dwóch plików będących powieściami zgodnie z metodologią przyjętą w poprzedniej pracy [A5]. Pierwszy plik tekstowy zakodowany w standardzie UTF-8 zawierał angielskie tłumaczenie powieści *Wojna i pokój* (ang. *War and Peace*) Lwa Tołstoja [O13] i zajmował 3.3 MB. Drugi plik tekstowy zakodowany w standardzie ISO 8859-1 zawierał oryginalną francuską wersję powieści *Artamène ou le Grand Cyrus* Georgesa de Scudéry’ego [O14] i zajmował 10 MB.

Rezultaty wydajnościowe przedstawione na Rysunku 9 pokazują podobną skalowalność zarówno dla procesorów Intel jak i ARM. Podobnie jak poprzednio, rozwiązanie uruchomione na superkomputerze Cray XC40 wykonywało się dwukrotnie szybciej niż na chmurze AWS. Wykonanie na chmurze Linaro trwało od 4 do 10 razy dłużej niż na Cray XC40 i przynajmniej dwukrotnie dłużej niż na chmurze AWS. Jedynym wyjątkiem była instancja *m1.small* w chmurze Linaro, której wydajność była podobna do wydajności instancji *a1.xlarge* w chmurze AWS.

W ramach badań będących podstawą niniejszej pracy zaprojektowałem i zaimplementowałem mechanizm uruchamiania zadań równoległych wykorzystujących bibliotekę PCJ na obu chmurach obliczeniowych. Ponadto byłem odpowiedzialny za kontakt z firmami Amazon i Linaro w celu otrzymania zasobów obliczeniowych do celów naukowych w ilościach wystarczających do przeprowadzenia miarodajnych badań. Dodatkowo zajmowałem się obsługą maszyn chmurowych i wykonywaniem obliczeń z wykorzystaniem tych zasobów. Opisałem mechanizm przygotowania środowiska chmurowego do wykonania obliczeń z wykorzystaniem biblioteki PCJ. Wykonałem wszystkie przedstawione w publikacji testy wydajnościowe aplikacji równoległych wykorzystujących bibliotekę PCJ na obu chmurach jak również na superkomputerze Okeanos w ramach grantu obliczeniowego, którego byłem kierownikiem. Byłem odpowiedzialny za przygotowanie opisu konfiguracji eksperymentu a także wyników wydajnościowych wraz z dyskusją osiągniętych rezultatów.

4.1.4 HPC, Big Data & AI [5] – PCJ 5.0.8

Sztuczna inteligencja (AI, ang. *Artificial Intelligence*), staje się coraz bardziej popularna. Umożliwia rozwiązywanie problemów, dla których stworzenie tradycyjnego algorytmu jest niemożliwe lub wyjątkowo złożone. W ostatnich latach szczególne znaczenie zdobyło podejście oparte na uczeniu głębokim (ang. *deep learning*), polegające na analizie bardzo dużych zbiorów danych uczących i dostosowywaniu wewnętrznych parametrów modelu w taki sposób, aby mógł on samodzielnie wydobywać wzorce i zależności. Im większy jest wolumen i różnorodność danych treningowych, tym lepiej sieć neuronowa *uczy się* je rozpoznawać, a w konsekwencji udzielać trafniejszych odpowiedzi dla wcześniej nieobserwowanych danych. Przetwarzanie ogromnych zbiorów danych oraz trenowanie sieci neuronowych o dużej liczbie parametrów wymaga jednak znacznych zasobów obliczeniowych, niedostępnych w typowych laptopach czy stacjach roboczych. Z tego względu istotne jest opracowanie narzędzi umożliwiających integrację przetwarzania danych typu Big Data z obciążeniami charakterystycznymi dla sztucznej inteligencji w środowiskach obliczeń wysokowydajnych (HPC, ang. *High-Performance Computing*).

Kolejna praca w cyklu [5] pokazuje, że rozwiązania oparte o bibliotekę PCJ mogą służyć do rozwiązywania zadań z różnych obszarów, takich jak HPC, Big Data i AI. Otwiera to możliwość integracji rozwiązań HPC z aplikacjami typu Big Data i Artificial Intelligence. Ponadto, dzięki przenośności rozwiązań opartych o język Java, użytkownik może rozwijać oprogramowanie na swoim laptopie czy stacji roboczej, a później, nawet bez ponownej kompilacji, uruchomić rozwiązanie korzystając z chmury obliczeniowej czy systemu HPC.

W omawianej pracy przedstawiliśmy problemy z obszarów HPC, Big Data i AI. Problemami HPC były *2D stencil* oraz wcześniej omawiany *FFT*, problemem typu Big Data był również wcześniej omawiany *WordCount*, a zadania związane z AI to modelowanie konektomu nicienia *C. Elegans* za pomocą algorytmu ewolucji różnicowej oraz rozproszone uczenie sieci neuronowej.

Wyniki wydajnościowe zostały zebrane na superkomputerach Cray XC40: *Okeanos* (ICM UW) oraz *Hazel Hen* (HLRS, University of Stuttgart, Niemcy). Węzły obliczeniowe superkomputerów posiadały po dwa procesory Intel Xeon E5-2690 v3 (*Okeanos*) lub Intel Haswell E5-2680 (*Hazel Hen*), każdy po 12 rdzeni oraz dostępny był też mechanizm *hyper-threading* dający dwa wątki na rdzeń, co łącznie daje 48 wątki obliczeniowe na węzeł. Rozwiązania bazujące na języku Java testowaliśmy korzystając z Oracle JDK 1.8.0_51 w przypadku PCJ i Oracle JDK 10.0.2 w przypadku APGAS. Dla rozwiązań opartych o C/MPI użyliśmy implementacji Cray MPICH w wersjach 8.3 (*Okeanos*) i 8.4 (*Hazel Hen*). Dodatkowo użyliśmy OpenMPI w wersji 4.0.0 oferującej interfejs MPI dla języka Java (ang. *Java bindings*) do wykonań Java/MPI.

W ramach niniejszej pracy byłem odpowiedzialny w szczególności za szczegółowy opis biblioteki PCJ, jej nowych funkcjonalności oraz za przygotowanie i uruchomienie testów wydajnościowych badających wydajność biblioteki w problemie typu *2D stencil* na wybranych superkomputerach. W ramach badań zdobyłem grant na uruchomienie programów wykorzystujących bibliotekę PCJ na superkomputerze *Hazel Hen* w HLRS w Niemczech, będącym w tamtym czasie na 17. miejscu listy TOP500 [O15]. Dzięki temu możliwe było wykonanie testów wydajnościowych na ponad 4000 węzłach obliczeniowych tego superkomputera, co pozwoliło na uruchomienie aplikacji na blisko 200 000 wątkach obliczeniowych. Ponadto opisałem uzyskane wyniki wraz z dyskusją osiągniętych rezultatów. Byłem również odpowiedzialny za całość pracy.

HPC Zadania HPC, które przedstawiliśmy w pracy to: dwuwymiarowy szablon obliczeniowy (ang. *2D stencil*) oraz *FFT*. Poniżej skupię się przede wszystkim na *2D stencil* jako rozwiązaniu stworzonym i omówionym wyłącznie przeze mnie.

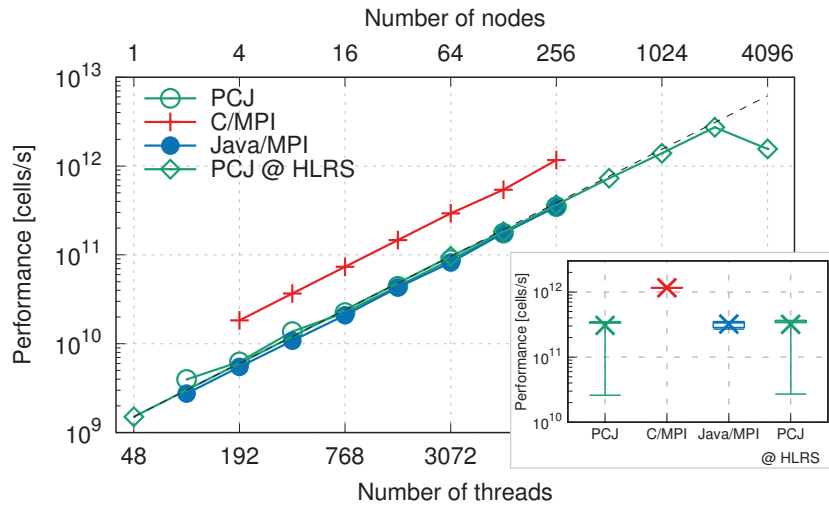
2D stencil został zaprezentowany na przykładzie automatu komórkowego *Game of Life* Johna Conway’a [O16]. W mojej implementacji plansza (wszechświat) nie jest nieskończona ani nie jest cykliczna – ma swoją wysokość i szerokość oraz brzeg. Każdy wątek posiada swoją podplanszę (zawierającą fragment wszechświata) – plansza dzielona jest na jednakowe części w sposób blokowy. Mimo iż istnieją szybkie algorytmy generowania kolejnego stanu automatu, zdecydowałem, że moja implementacja będzie bezpośrednią implementacją sprawdzającą stan każdej komórki. Aby zaoszczędzić pamięć, każda komórka jest reprezentowana jako pojedynczy bit, gdzie 0 oznacza, że komórka jest martwa, a 1, że jest żywa. Po wyliczeniu nowego stanu planszy-wszechświata (po kroku czasowym), komórki znajdujące się na brzegach podplansz są asynchronicznie wymieniane z odpowiednimi wątkami-sąsiadami.

Dla celów porównawczych przygotowałem również wersję napisaną w języku C i korzystającą z MPI (C/MPI) oraz w języku Java korzystającą z MPI (Java/MPI). Główny kod bazował na implementacji wykorzystującej PCJ, ale komunikacja odbywała się za pomocą funkcji MPI.

Do celów wydajnościowych dla każdego kroku czasowego liczyłem liczbę komórek przetworzonych w jednostce czasu (*cells/s*). Każdy test składał się z 11 kroków czasowych. Zapewniło to *rozgrzanie* JVM, aby umożliwić jej wykonanie kompilacji JIT (ang. *just-in-time*) do optymalizacji wykonania. Upewniłem się również, że GC (ang. *garbage collector*) nie ma wpływu na uzyskaną wydajność. W tym celu wziąłem pod uwagę wydajność szczytową (maksymalną) dla całego obliczenia.

Rysunek 10 przedstawia porównanie wydajności aplikacji *Game of Life* dla planszy rozmiaru 604 800×604 800 komórek. Wydajność obu implementacji (PCJ i Java/MPI) jest bardzo podobna i zapewnia niemal idealną skalowalność. Wersja C/MPI oferuje trzykrotnie wyższą wydajność i podobną skalowalność. Dane dotyczące wydajności pokazują skalowalność do 98 304 wątków (na 2048 węzłach). W przypadku większej liczby wątków wydajność przetwarzania równoległego spada ze względu na niewielkie obciążenie każdego procesora w porównaniu z czasem komunikacji wymaganym do wymiany danych brzegowych.

Wykres znajdujący się na wstawce przedstawia statystyki wydajności obliczone na podstawie 11 kroków czasowych aplikacji *Game of Life* wykonanej na 256 węzłach (12 288 wątków) superkomputera *Okeanos*. Końce *wąsów* oznaczają wartości minimalne i maksymalne, krzyżyk (×) reprezentuje wartość średnią, pudełko reprezentuje wartości między pierwszym a trzecim kwartyłem, a pasek wewnątrz oznacza wartość drugiego kwartyłu (mediany). W przypadku C/MPI pudełko i wąsy nie są widoczne, ponieważ wykonanie wykazuje bardzo zbliżoną wydajność dla wszystkich kroków. W przypadku wykonań PCJ i Java/MPI minimalne wartości pochodzą z pierwszych kroków



Rysunek 10: Wyniki wydajnościowe dla *Game of Life* dla planszy rozmiaru 604 800×604 800 komórek. Linie przerywane oznaczają idealne skalowanie dla PCJ. Wyższa wartość oznacza lepszy wynik. (z [5])

wykonania, gdy wykonanie zostało wykonane w trybie interpretowanym. Kompilacja JIT szybko zoptymalizowała uruchamiany kod i zdecydowana większość kroków została wykonana z najwyższą wydajnością. Wyraźnie widać, że aplikacje Java po kompilacji JIT mają bardzo stabilne wyniki wydajności, ponieważ dane maksymalne i kwartyle (mediana oraz kwartyle pierwszy i trzeci) są na rysunku niemal nieodróżnialne.

Big Data Omówienie problemu z obszaru *Big Data* było zadaniem współautorów pracy. W omawianej pracy wybrali *WordCount* jako przykładową aplikację przetwarzającą duże zbiory danych. Porównali wydajność między rozwiązaniem PCJ a APGAS oraz implementacją bazującą na C++/MPI. W przypadku PCJ zastosowali implementację zawierającą redukcję wykorzystującą samodzielnie zaimplementowane drzewo binarne do komunikacji [A4]. W przypadku APGAS zastosowali bazową implementację APGAS_{stat} jak i implementację APGAS_{dyn} rozszerzoną o dynamiczne równoważenie pracy. Biblioteka APGAS i implementacja problemu *WordCount* bazowała na wersji przedstawionej w pracy [O17]. W przypadku rozwiązania natywnego wybrali język C++ zamiast C, jako bardziej odpowiedni do tego zadania ze względu na to, że posiada w standardowej bibliotece wbudowany typ map a także wysokopoziomowe funkcje operujące na łańcuchach znaków.

Wyniki wydajnościowe przedstawione w pracy pokazują dobrą skalowalność implementacji PCJ, a kluczowym etapem wpływającym na ogólną wydajność jest faza redukcji. Implementacja PCJ osiąga lepsze wyniki niż wersja oparta na APGAS, co wynika z zastosowania efektywnego mechanizmu redukcji opartego na komunikacji bazującej na strukturze drzewa binarnego. Z kolei implementacja w języku C++ wykazuje idealną skalowalność, jednak jej czas wykonania jest około czterokrotnie dłuższy, co wynika z bardziej kosztownego etapu dzielenia linii tekstu na pojedyncze słowa. Warto w tym miejscu wspomnieć, że wyniki przedstawione w pracy [A5] uzyskane na klastrze Hadoop wskazują, że implementacja PCJ jest przynajmniej trzykrotnie szybsza niż implementacja Hadoop.

Artificial Intelligence W ramach badań łączących bibliotekę PCJ ze sztuczną inteligencją współautorzy sprawdzili możliwość zastosowania biblioteki w dwóch rodzajach zadań.

W pierwszym biblioteka została wykorzystana do zamodelowania konektomu nicienia *Caenorhabditis Elegans*. Nicień ten jest jedynym organizmem z konektomem poznanym w całości. Były przeprowadzane eksperymenty, których celem było połączenie struktury konektomu z rzeczywistym

zachowaniem nicienia [O18]. Wyniki tych eksperymentów pozwoliły na stworzenie modelu matematycznego odpowiedniego fragmentu konektomu przez ekspertów biofizycznych. Model został zdefiniowany jako zestaw równań różniczkowych zwyczajnych z 8 parametrami. Aby zbadać przestrzeń rozwiązań i dopasować parametry modelu tak, aby jego przewidywania były zgodne z danymi empirycznymi, zastosowano algorytm ewolucji różnicowej (ang. *differential evolution*). Model matematyczny został zaimplementowany w języku Java i zrównoleglony za pomocą biblioteki PCJ. Warto wspomnieć, że prototyp rozwiązania powstał w bardzo krótkim czasie (w około 2 miesiące) mimo zmieniających się wymagań eksperta biofizycznego.

Drugi rodzaj zadań był związany z uczeniem maszynowym. Współautorzy opracowali dedykowany program wykorzystujący TensorFlow Java API do rozproszonego uruchamiania oraz instrumentacji procesu trenowania sieci neuronowych. Zaimplementowana w rozwiązaniu operacja redukcji oparta została na algorytmie *all-reduce* realizowanym w topologii hipersześcianu [O19]. Biblioteka PCJ była odpowiedzialna za rozproszone uruchamianie obliczeń, ich monitorowanie i komunikację, podczas gdy obliczenia były wykonywane przez TensorFlow.

W celu zademonstrowania poprawności działania wykonali trening sieci rozpoznającej ręcznie zapisane liczby ze zbioru MNIST [O20]. Zbiór ten zawiera 60 000 obrazów treningowych, z czego w przedstawionym teście 5000 zostało zarezerwowanych do celów walidacyjnych. Sieć neuronowa składała się z trzech w pełni połączonych warstw o rozmiarach odpowiednio 300, 100 i 10 neuronów [O21]. Była trenowana przez 20 epok małymi partiami po 50 obrazów. Współautorzy sprawdzili dwa algorytmy nauki. W pierwszym obliczenia były wykonywane równolegle, a gradienty były uśredniane po każdej partii obrazów, podobnie jak to robi Horovod [O22]. Drugi algorytm implementuje asynchroniczny równoległy spadek gradientu w sposób opisany przez innych autorów [O23]. Wydajność tych rozwiązań została porównana z rozwiązaniem Horovod opartym o C, MPI i Python. Rozwiązania osiągały dokładność rzędu 96%.

Wyniki wydajnościowe zaprezentowane w pracy pokazują, że asynchroniczna implementacja wykorzystująca PCJ ma porównywalną wydajność co Horovod, ale osiągnięcie pełnej wydajności Horovod wymagało zmiany domyślnej konfiguracji instalacji TensorFlow, podczas gdy użyta wersja TensorFlow dla języka Java nie wymagała dodatkowej konfiguracji.

Przedstawione wyniki badań pokazują, że PCJ może być efektywnie wykorzystywana do równoległego przetwarzania zadań AI, a dzięki zastosowaniu języka i ekosystemu Java integracja z istniejącymi narzędziami była stosunkowo prosta. Biblioteka PCJ pozwoliła na uruchomienie TensorFlow na klastrze HPC w konfiguracji wielowęzłowej i gwarantującej wysoką wydajność.

PCJ 5.1.0

Wersja 5.1.0 biblioteki PCJ wprowadziła szereg zmian funkcjonalnych, wykraczających poza poprawki porządkujące i naprawiające błędy.

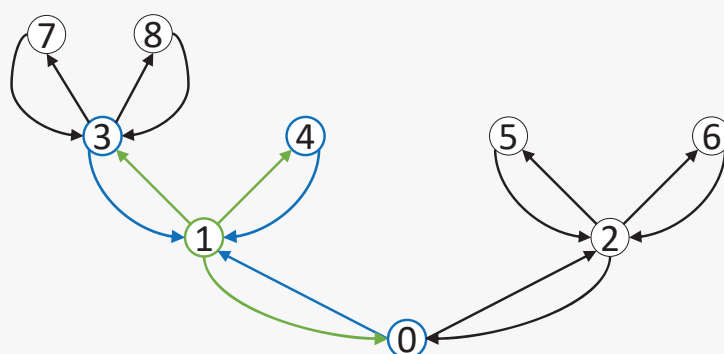
Po pierwsze, w operacji `broadcast` zmodyfikowałem mechanizm potwierdzania dostarczenia wiadomości – znacząco ograniczyłem liczbę wiadomości przesyłanych do węzła #0, w taki sposób, że zamiast bezpośredniego przesyłania komunikatów zwrotnych do węzła #0 przez wszystkie węzły, zastosowałem do komunikacji strukturę drzewa binarnego.

Ponadto zmieniłem interfejs uruchamiania aplikacji: w miejsce metod `PCJ.start` i `PCJ.deploy` wprowadziłem konstrukcję opartą na wzorcu projektowym *builder* – `PCJ.executionBuilder` – z analogicznymi metodami `start` i `deploy`. Rozwiązanie to umożliwia

wcześniejsze przygotowanie konfiguracji uruchomieniowej, w tym m.in. przekazanie listy węzłów obliczeniowych oraz dodatkowych właściwości.

Dodałem również metodę `PCJ.accumulate`, służącą do modyfikacji zmiennej współdzielonej zgodnie z przekazaną funkcją i wartością – na przykład w celu zwiększenia przechowywanej wartości o jeden. W tej wersji pojawiły się także nowe operacje kolektywne, takie jak `PCJ.reduce` i `PCJ.collect`, które zostaną omówione w dalszej części.

Mechanizm fail-safe W wersji 5.1.0 wprowadziłem także mechanizm *fail-safe*, którego opis zamieściłem w publikacji [5]. Jego zadaniem jest umożliwienie bezpiecznego zakończenia pracy aplikacji w sytuacji wystąpienia awarii. Bez tego mechanizmu obliczenia mogłyby utknąć w miejscu, podczas gdy aplikacja cały czas byłaby aktywna. W przypadku systemów klastrowych taka sytuacja powodowałaby niepotrzebne zużywanie CPU-godzin bez żadnej użytecznej pracy, aż nadejdzie limit czasu zadania.



Rysunek 11: Drzewo komunikatów dla mechanizmu *fail-safe* z zaznaczonymi sąsiadami węzła #1. Zielone strzałki reprezentują komunikaty o aktywności wysyłane do sąsiadów, a niebieskie otrzymane przez węzeł #1 od sąsiadów. (z [5])

Mechanizm *fail-safe* w bibliotece PCJ działa w taki sposób, że w przypadku awarii obliczenia są bezpiecznie przerywane. Wykrywanie awarii odbywa się na dwa sposoby. Pierwszy z nich związany jest z błędem wykonania w wątku PCJ, na przykład wynikającym z nieprzechwycenia wyjątku. W takiej sytuacji wewnętrzny nadzorca wykonania oznacza dany węzeł jako uległy awarii. Drugi sposób opiera się na mechanizmie *bicia serca* (ang. *heartbeat*). Komunikaty o aktywności są okresowo wysyłane przez każdy węzeł do jego sąsiadów – węzła nadrzędnego i potomnych (Rysunek 11). Jeżeli węzeł nie otrzyma komunikatu o aktywności od jednego z sąsiadów w określonym czasie, zakłada wystąpienie awarii. Analogicznie, awaria zostaje rozpoznana, gdy nie można wysłać komunikatu o aktywności.

W momencie wykrycia awarii węzeł natychmiast wysyła komunikaty o przerwaniu do swoich aktywnych sąsiadów i zatrzymuje obliczenia. Każdy węzeł, który otrzyma taki komunikat, przekazuje go dalej do pozostałych sąsiadów (z pominięciem węzła źródłowego) i również przerywa swoje działanie. W rezultacie komunikat o przerwaniu propaguje się do wszystkich węzłów, a aplikacja zostaje w kontrolowany sposób zakończona.

4.1.5 TeraSort [3] – PCJ 5.1.0

Praca [3] opisuje problem sortowania wielkich zbiorów danych w środowisku wielowęzłowym. W pracy przedstawiony został algorytm *TeraSort*, jego implementacje i porównanie ich wydajności

z wykorzystaniem Apache Hadoop i PCJ. W przypadku implementacji PCJ wykorzystana była wersja 5.1.0 biblioteki. Praca ta powstała w wyniku badań prowadzonych w ramach zdobytych przeze mnie grantów obliczeniowych o nazwie *Benchmarking Java on Ethernet Cluster* na klastrze *1000 Core Cluster* w infrastrukturze Future SOC Lab, Hasso-Plattner Institute (HPI; Poczdam, Niemcy). Każdy węzeł posiadał cztery 10-rdzeniowe procesory Intel Xeon E7-4870 @ 2.4 GHz (z *hyper-threading* łącznie 80 wątków obliczeniowych) i 1 TB pamięci RAM. Wyniki pośrednie ukazały się w raportach technicznych z grantów: [A6, A7, A8].

Algorytm TeraSort jest jednym z szeroko stosowanych narzędzi do testowania wydajności Hadoop. Przy badaniu wydajności mierzy się czas sortowania 100-bajtowych rekordów. Każdy rekord składa się z 10-bajowego klucza i 90 bajtowej wartości. W przypadku omawianym w pracy, liczba rekordów do sortowania wynosiła od 10^7 do 10^{10} , czyli rozmiar pliku wejściowego wynosił od ok. 1 GB do ok. 1 TB.

Algorytm *samplesort* Algorytm TeraSort bazuje na algorytmie *samplesort* (lub *sample sort*). Algorytm *samplesort* wykorzystuje metodę dziel i zwyciężaj (ang. *divide-and-conquer*). Na początek działania algorytmu *samplesort* wybieranych jest $p - 1$ elementów rozdzielających (ang. *pivots*). Elementy rozdzielające są próbkowane z danych wejściowych, a następnie sortowane innym algorytmem sortowania. Następnie dane wejściowe są dzielone do p kubeków (ang. *buckets*) na podstawie $p - 1$ elementów rozdzielających. W kolejnym kroku zawartość kubeków jest sortowana. W oryginalnym algorytmie *samplesort*, kubki są sortowane rekurencyjnie przy użyciu tego samego algorytmu, jednak jeśli rozmiar kubka spadnie poniżej pewnego progu, stosowany jest inny algorytm sortowania. Ostatecznie, proste połączenie kubeków daje posortowany wynik.

Algorytm *samplesort* bardzo dobrze nadaje się do zrównoleglenia. Liczba kubeków powinna być równa liczbie jednostek obliczeniowych (np. procesorów) – p . W idealnym przypadku liczba elementów w każdym z kubeków będzie taka sama, dzięki czemu każdy procesor spędzi podobny czas na sortowaniu.

Algorytm *samplesort* jest uogólnieniem algorytmu *quicksort* z tą różnicą, że *quicksort* wykorzystuje tylko 1 element osiowy. Również podobnie jak w algorytmie *quicksort* – odpowiedni dobór *pivotów* jest kluczowym krokiem algorytmu, ponieważ to od nich zależą rozmiary kubeków, a co za tym idzie – wydajność rozwiązania.

Algorytm TeraSort W omawianej pracy przedstawiam implementację algorytmu TeraSort zawartą w pakiecie Apache Hadoop zapisaną zgodnie z modelem map-reduce. Tutaj jednak ten opis pominię i skupię się wyłącznie na rozwiązaniu stworzonym przeze mnie z wykorzystaniem biblioteki PCJ. Sam algorytm składa się z 5 kroków:

1. wybór elementów rozdzielających,
2. odczytanie danych wejściowych i wkładanie ich do odpowiednich kubeków,
3. wymiana kubeków między wątkami,
4. sortowanie danych w kubkach,
5. zapisywanie danych z kubków.

Pierwszą częścią algorytmu jest wybór elementów rozdzielających. Jest to kluczowy element algorytmu wpływający na praktyczną wydajność aplikacji. Wybór elementów wykonałem w ten

sposób, że każdy wątek czyta $\lceil \frac{100\,000}{p} \rceil$ pierwszych elementów swoich danych wejściowych, a następnie te elementy są zbierane w wątku #0 za pomocą operacji redukcji w jedną listę 100 000 kandydatów. Wątek #0 sortuje otrzymaną listę i równomiernie wybiera $p - 1$ elementów rozdzielających – powstaje lista $pivots_{0..p-1}$. Elementy rozdzielające są rozgłaszane do każdego wątku i następuje drugi krok algorytmu. W pracy [A7] przedstawiłem również prostszy i szybszy sposób wyboru elementów rozdzielających, ale nie generuje on tak dobrego podziału danych.

W drugim kroku algorytmu każdy wątek czyta swoją porcję danych i od razu wkłada je do odpowiednich lokalnych kubeków.

W kolejnym kroku wątki wysyłają dane z lokalnych kubeków do odpowiednich wątków. W ten sposób wątek #0 ma dane z kubeków wszystkich wątków, w których to kubkach znajdują się liczby mniejsze od najmniejszego elementu rozdzielającego ($pivots_0$). Wątek t ma dane, które są większe lub równe $pivots_{t-1}$ a mniejsze niż $pivots_t$. Wątek o najwyższym indeksie ma elementy, które są większe lub równe największemu elementowi rozdzielającemu ($pivots_{p-1}$).

W kroku czwartym każdy wątek sortuje otrzymane elementy za pomocą algorytmu sortowania z biblioteki standardowej Javy.

Ostatni krok to zapisanie wyniku. W pracy przedstawiłem trzy podejścia do zapisywania wyników:

1. zapisywanie w kolejności – jeden wątek pisze do pojedynczego pliku wynikowego, pozostałe czekają na swoją kolej,
2. zapisywanie jednoczesne – wszystkie wątki mogą jednocześnie pisać do tego samego pliku korzystając z możliwości mapowania w pamięci rozdzielnych fragmentów pliku,
3. zapisywanie do osobnych plików – każdy wątek zapisuje wyniki w osobnym pliku.

Jak można się domyślić, zapisywanie w kolejności do jednego pliku jest najwolniejszym podejściem. Przy sortowaniu 10^9 rekordów (plik wejściowy rozmiaru 100 GB) przez 640 wątków czas samego zapisu wyniósł 24 minuty, podczas gdy jednoczesny zapis skrócił ten czas do 15 minut, natomiast zapisywanie do osobnych plików zajęło niecałe 5 minut. Oczywiście zapisywanie do osobnych plików wymaga, by pliki połączyć, by uzyskać dokładnie taki sam wynik jak w pozostałych przypadkach, co również zabiera czas. Wykonane testy pokazały, że średni czas łączenia to niecałe 14 minut. Ze względu na to, że algorytm działający w Hadoop tworzy osobne pliki z wynikami, w dalszych pracach wybrano trzecie podejście i pominięto etap łączenia plików.

Kolejną optymalizacją algorytmu, którą wykonałem, było spostrzeżenie, że krok 2 i 3 mogą być wykonywane jednocześnie – wysyłanie danych z kubeków może być wykonywane w tym samym czasie co czytanie pliku. Dzięki temu uwzględniamy naturę modelu PGAS pozwalającą na jednostronną asynchroniczną komunikację. Zamiast czekać na zakończenie kroku 2, sprawdzamy przy dodawaniu elementów do kubka, czy znajduje się w nim odpowiednia liczba elementów (np. 100, 1000, 10 000 czy 100 000). Jeśli kubelek staje się pełny, to jest on wysyłany do odpowiedniego wątku. Testy wydajnościowe pokazały, że dla największych danych (10^{10} rekordów) najwydajniejsze jest ustawienie limitu na 1000. W przypadku mniejszych rozmiarów danych wejściowych nie było widać znaczącego przyspieszenia w stosunku do wersji pierwotnej.

Oszacowanie złożoności obliczeniowej Mając podany pełen algorytm można policzyć jego optymistyczną, średnią i pesymistyczną złożoność obliczeniową. Obliczenia wykonałem dla p

procesorów, a więc $p-1$ elementów rozdzielających oraz dla rozmiaru danych wejściowych wynoszącym n ($n \gg p$). Poniższe obliczenia przedstawiłem z perspektywy pojedynczego procesora w przypadku optymistycznym. Wybór kandydatów na elementy rozdzielające to koszt $O(p)$, gdyż każdy z p procesorów wysyła stałą liczbę elementów do procesora 0. Sortowanie kandydatów przez procesor 0 to koszt $O(p \log p)$, ich równomierny wybór to $O(p)$, a rozgłoszenie wybranych elementów rozdzielających do wszystkich procesorów to koszt $O(p \log p)$, gdyż p elementów rozgłaszane jest przy użyciu drzewa komunikacji będącym zrównoważonym drzewem binarnym o głębokości $O(\log p)$. Czytanie danych przez procesor i wkładanie ich do kubelków to koszt $O(\frac{n}{p} \log p)$, bo dla każdego z $\frac{n}{p}$ elementów należy znaleźć poprawny kubelek, co ma złożoność $O(\log p)$ przy zastosowaniu algorytmu wyszukiwania binarnego, a wkładanie do kubelka można wykonać w czasie stałym $O(1)$. Rozdzielenie (rozesłanie) kubelków do odpowiednich procesorów to koszt $O(\frac{n}{p})$, a sortowanie *swojego* kubelka przez pojedynczy procesor to koszt $O(\frac{n}{p} \log \frac{n}{p})$.

Podobnie jak w przypadku algorytmu *quicksort*, złożoność czasowa zależy od wyboru elementów rozdzielających, czyli podziału na kubelki. Najlepszy podział jest wówczas, gdy w kubelkach znajdzie się taka sama liczba elementów. Okazuje się, że średnia złożoność obliczeniowa algorytmu jest taka sama jak optymistyczna złożoność obliczeniowa, ale z większą stałą ukrytą w notacji O , z tych samych powodów, dla których średnia złożoność algorytmu *quicksort* jest taka sama jak optymistyczna złożoność tego algorytmu. Ostatecznie średnia złożoność obliczeniowa jest rzędu:

$$O\left(\frac{n}{p} \log \frac{n}{p} + p \log p\right)$$

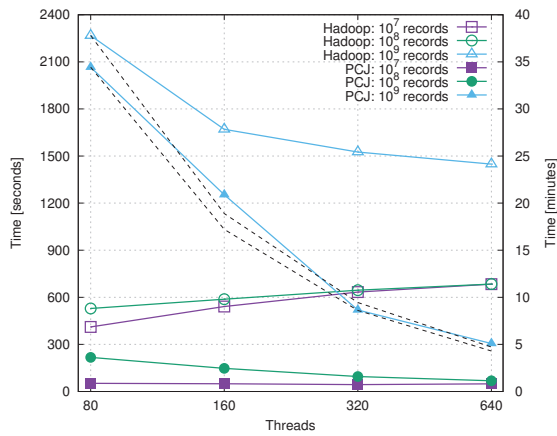
W przypadku pesymistycznym wszystkie kubelki z wyjątkiem jednego posiadają 1 element, a do tego jednego trafiają pozostałe liczby. Złożoność wówczas wynosi:

$$O(n \log n + p \log p)$$

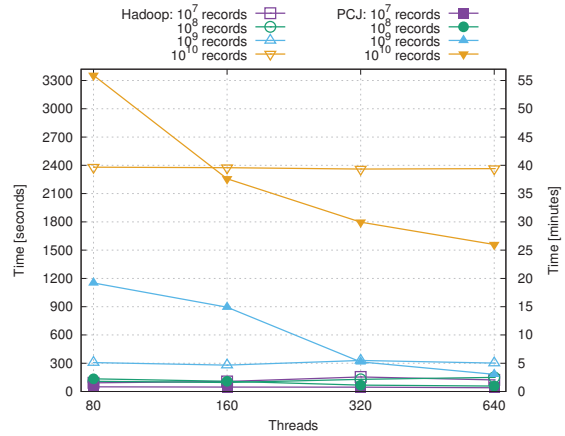
W powyższych obliczeniach zakładałem, że średnia i najgorsza złożoność wewnętrznego algorytmu sortowania n -elementów wynosi $O(n \log n)$.

Wyniki wydajnościowe Jako, że celem było porównanie implementacji algorytmu z wykorzystaniem Hadoop i PCJ, kolejnym krokiem było odczytywanie i zapisywanie danych przez PCJ z wykorzystaniem systemu plików HDFS (Hadoop Distributed File System), który jest naturalny dla Hadoopa. Dla największych danych czas działania algorytmu przy wykorzystaniu HDFS wiązał się z narzutem 3%, a dla małych danych (10^7 rekordów) czas działania z wykorzystaniem HDFS był nawet dwukrotnie dłuższy.

Gdy implementacja algorytmu TeraSort z wykorzystaniem PCJ była gotowa, wykonałem badanie wydajności rozwiązania opartego o Hadoop. Ze względu na konieczność używania systemu kolejkowego SLURM naturalnym było, że jedno zadanie jest klastrem Hadoop z tzw. *datanode* przechowującymi dane, a inne zadania będą odpowiedzialne za przetwarzanie danych. Dodatkowo w większości konfiguracji SLURM uruchomione zadania otrzymują wyłączny dostęp do węzłów, na których są uruchamiane. W związku z tym zadania przetwarzające dane, w przypadku Hadoop z komponentami *nodemanager*, były wykonywane na osobnych węzłach obliczeniowych (*tryb zewnętrzny*). Ponadto ten mechanizm pozwalał na ustalenie limitu na liczbę węzłów, które przetwarzają dane. Rysunek 12a przedstawia czasy wykonania algorytmu przy różnej liczbie rekordów. Wyniki pokazały, że wydajność rozwiązania opartego o PCJ była dużo wyższa niż rozwiązanie w Hadoop. Było to spowodowane tym, że ten tryb jest *nienaturalny* dla Hadoop, który nie może wykorzystać zalet związanych z rozmieszczeniem danych.



(a) *Tryb zewnętrzny*: węzły obliczeniowe były umieszczone na innych węzłach niż węzły z danymi



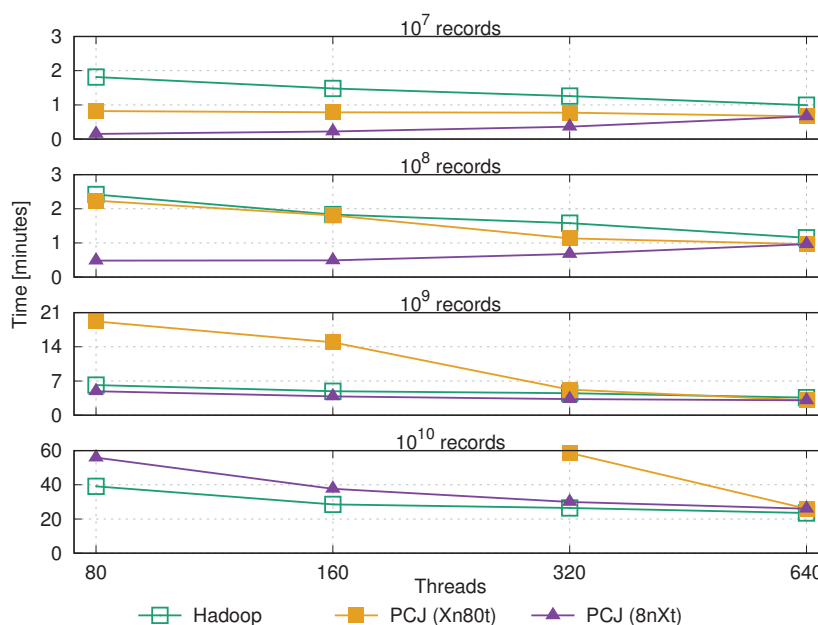
(b) *Tryb wewnętrzny*: węzły obliczeniowe były umieszczone na tych samych węzłach co węzły z danymi

Rysunek 12: Całkowity czas wykonania implementacji algorytmu TeraSort w Hadoop i PCJ. Niższa wartość oznacza lepszy wynik. (z [3])

Postanowiłem więc sprawdzić inne podejście, gdzie tworząc klaster Hadoop *datanode* i *nodemanager* były uruchamiane obok siebie na tych samych węzłach obliczeniowych w ramach tego samego zadania SLURM (*tryb wewnętrzny*). Spowodowało to, że liczba *nodemanager* była stała w trakcie wszystkich testów, co oznacza, że zadania mogły być umieszczane na każdym węzle bez limitów. W tym przypadku zadania związane z PCJ były uruchamiane w nienaturalny dla SLURM sposób, jako dołączenie zewnętrzne do zadania z klastrem Hadoop. Wyniki tego podejścia przedstawione są na Rysunku 12b. Rozwiązanie wykorzystujące Hadoop daje w przybliżeniu stały czas, co sugeruje, że nie ma limitu na liczbę kontenerów uruchomionych w ramach klastra. Klaster wybrał, aby uruchomić zadania mapowania i redukcji na wszystkich węzłach, aby zminimalizować dane, które muszą być transmitowane między węzłami z danymi a węzłami przetwarzającymi. Rozwiązanie w PCJ również korzysta na tym, że zostało wykonane na tych samych węzłach co dane i ostatecznie, używając 640 wątków, rozwiązanie to jest bardziej wydajne niż rozwiązanie oparte o Hadoop.

Duża różnica w całkowitym czasie wykonania pomiędzy PCJ a Hadoop wymagała weryfikacji konfiguracji klastra Hadoop, a przede wszystkim ustalenia odpowiedniej wartości *maksymalnej przydzielanej pamięci* zadaniom map i reduce. Dotychczas wartość ta była ustawiona na 40 GB. Pamiętając, że każdy węzeł dysponuje co najmniej 820 GB wolnymi pamięci RAM oraz 40 rdzeniami fizycznymi (80 rdzeniami logicznymi z wykorzystaniem technologii *hyper-threading*) maksymalna liczba zadań map i reduce została określona na 80 dla każdego typu. Uwzględniając te parametry, przeprowadziłem testy wydajnościowe dla różnych wartości *maksymalnej przydzielanej pamięci* zadaniom: 5 GB, 10 GB, 20 GB oraz 40 GB. Najkrótsze czasy były w przypadku, gdy ta wartość była ustalona na 10 GB i 20 GB, ale dla największych danych wejściowych (10^{10} rekordów) największa wydajność była dla ustawienia wynoszącego 10 GB i tę wartość wykorzystałem w kolejnych badaniach wydajnościowych. Ustawienie to pozwala na pełne wykorzystanie dostępnej pamięci bez zbyt dużej nadmiernej alokacji.

W omawianej pracy zawarłem wiele innych wyników wydajnościowych. Ostatni wynik, pokazany na Rysunku 13 jest najważniejszy, bo podsumowuje uzyskane rezultaty. Przedstawia on silne skalowanie implementacji algorytmu TeraSort stworzonego z wykorzystaniem Apache Hadoop jak i PCJ dla różnej liczby rekordów (tj. 10^7 , 10^8 , 10^9 and 10^{10} rekordów). W ramach testów przygotowałem



Rysunek 13: Porównanie wydajności implementacji algorytmu TeraSort w Hadoop i PCJ dla różnej liczby rekordów w reżimie silnej skalowalności. Każde zadanie mapowania i redukcji w Hadoop było wykonywane z użyciem 10 GB pamięci. Węzły obliczeniowe umieszczono na tych samych węzłach co węzły danych. Wyniki dla PCJ zebrano przy użyciu 1, 2, 4 i 8 węzłów (z 80 wątkami na węzeł; Xn80t) oraz na 8 węzłach (z 10, 20, 40 i 80 wątkami na węzeł; 8nXt). Niższa wartość oznacza lepszy wynik. (z [3])

środowisko uruchomieniowe, tak by wykonanie Hadoop było jak najbardziej optymalne. W tym samym środowisku, z tymi samymi warunkami, uruchomiłem również rozwiązanie bazujące na bibliotece PCJ.

Całkowity czas działania aplikacji napisanej w PCJ zależał w dużej mierze od liczby użytych węzłów i wątków – przy stałej liczbie wątków wydajniejsze było uruchomienie aplikacji na większej liczbie węzłów. W przypadku mniejszych rozmiarów danych wejściowych (10^7 i 10^8 rekordów) implementacja w PCJ jest bardziej wydajna. Czas sortowania 10^9 rekordów używając Hadoop i PCJ jest zbliżony. Przy największym zbiorze danych (10^{10}) rozwiązanie wykorzystujące Hadoop jest bardziej wydajne dla mniejszej liczby wątków, a różnica jest prawie niewidoczna przy największej liczbie przetestowanych wątków (640).

Podsumowując, przedstawione w pracy badania wykazały podobną wydajność dla Hadoop i PCJ. Model programowania PGAS, reprezentowany przez bibliotekę PCJ, może stanowić wysoce konkurencyjne rozwiązanie względem modelu map-reduce – zarówno pod względem potencjalnych korzyści wydajnościowych, jak i efektywności tworzenia oprogramowania. Model ten oferuje przejrzystą abstrakcję umożliwiającą konstruowanie aplikacji równoległych w oparciu o globalną przestrzeń adresową, a ponadto zapewnia kontrolę nad układem danych oraz ich rozmieszczeniem pomiędzy procesorami. Co istotne, PGAS jest modelem ogólnego przeznaczenia, który może znaleźć zastosowanie w szerokiej gamie problemów obliczeniowych, w przeciwieństwie do modelu map-reduce, w którym to formułowany problem musi zostać dostosowany do ograniczeń i charakterystyki samego modelu.

PCJ 5.2.0

Wersja 5.2.0 biblioteki PCJ wprowadziła istotne zmiany w mechanizmie tworzenia grup wątków.

Dotychczas, aby dołączyć do grupy, należało wykonać operację `PCJ.joinGroup("groupName")`, podając w parametrze nazwę grupy, do której wątek

miał zostać przypisany. Operację tę można było wykonać w dowolnym momencie. Jednakże, aby zachować poprawność wykonania i spójność informacji o grupach między wątkami, konieczne było zapewnienie, że wszystkie wątki odnotują wykonanie tej operacji, na przykład poprzez wywołanie bariery na grupie nadrzędnej (np. globalnej).

Dodatkowym ograniczeniem wcześniejszego rozwiązania był brak możliwości łatwego ustalenia kolejności wątków w tworzonej grupie – aby to osiągnąć, każdy wątek musiał być dołączany indywidualnie, z koniecznością wykonania bariery pomiędzy kolejnymi dołączeniami. Rozwiązanie to było niewygodne w użyciu i w praktyce prowadziło niekiedy do trudnych do zdiagnozowania błędów.

Wersja 5.2.0 wprowadziła nowy mechanizm tworzenia grup, który rozwiązał opisane problemy kosztem ograniczenia możliwości rozszerzania grup po ich utworzeniu. Nowe podejście pozwala jednak na określenie kolejności wątków w grupie już w momencie jej tworzenia. Wszystkie wątki grupy nadrzędnej (np. globalnej) wykonują wspólnie operację `PCJ.splitGroup`, przekazując w parametrach tzw. *kolor* grupy (lub `null`, jeśli wątek nie ma do żadnej grupy dołączyć) oraz *klucz*. Wątki o tym samym kolorze trafiają do tej samej grupy, a ich kolejność ustalana jest na podstawie wartości klucza. Dzięki temu za pomocą jednej operacji kolektywnej można utworzyć wiele grup, w których kolejność wątków jest jednoznacznie określona i zgodna z intencją użytkownika.

W tej wersji wprowadzono również nową operację kolektywną: `PCJ.scatter`. Szczegółowe omówienie operacji kolektywnych znajduje się w kolejnej podsekcji.

4.1.6 Collectives [6] – PCJ 5.3.0

Operacje kolektywne (ang. *collective operations*) to operacje komunikacyjne w środowiskach równoległych, które angażują jednocześnie grupę procesów. W odróżnieniu od komunikacji punkt-punkt (ang. *point-to-point*), która odbywa się pomiędzy dwoma konkretnymi procesami, operacje kolektywne koordynują wymianę lub synchronizację danych w sposób zorganizowany i grupowy. Standaryzacja operacji kolektywnych nastąpiła w roku 1994 wraz z publikacją specyfikacji MPI-1 [O24, O25]. Przykładami operacji kolektywnych są: *barrier* (mechanizm zapewniający, że wszystkie procesy osiągną ten sam punkt wykonania przed kontynuacją obliczeń), *broadcast* (przesyłanie danych z jednego procesu do wszystkich pozostałych), *scatter* i *gather* (dystrybucja i zbieranie danych wśród procesów) oraz *reduce* (łączenie wyników cząstkowych z wielu procesów w jeden wynik globalny).

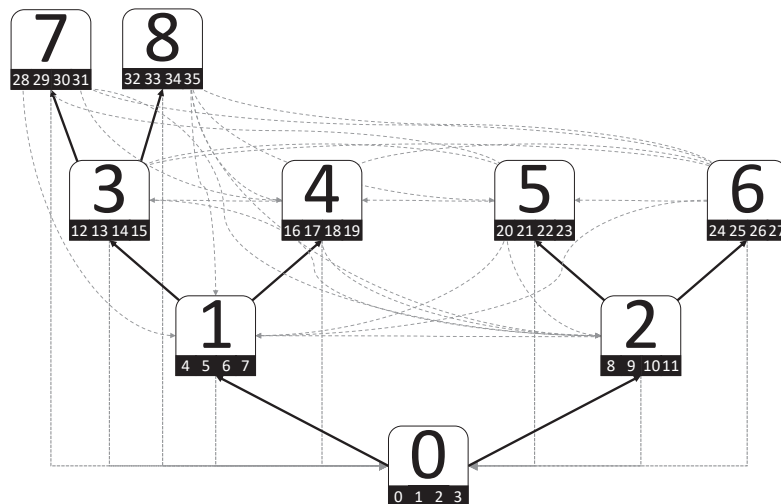
Większość języków PGAS udostępnia operacje typu *barrier* i *broadcast*, natomiast inne operacje kolektywne rzadko są dostępne jako gotowe metody czy procedury. W praktyce oznacza to, że użytkownik musi samodzielnie implementować te mechanizmy. Zoptymalizowane i skalowalne operacje kolektywne są kluczowym czynnikiem wpływającym na wydajność stworzonych rozwiązań.

Do wersji 5.1.0 wśród operacji kolektywnych dostępnych w bibliotece PCJ znajdowały się jedynie `barrier()` oraz `broadcast(T newValue, Enum<?> name)` (wraz z ich asynchronicznymi odpowiednikami: `asyncBarrier()` i `asyncBroadcast(...)`). Rozwój operacji kolektywnych w bibliotece PCJ przedstawia kolejna praca z cyklu [6]. W ramach tej pracy byłem odpowiedzialny za przygotowanie koncepcji badań, implementacji mechanizmów kolektywnych w bibliotece PCJ i przeprowadzenie testów wydajnościowych problemów *FFT* i *WordCount*. Rozwinąłem wcześniejsze implementacje *FFT*

i *WordCount* dodając wykorzystanie operacji kolektywnych wprowadzonych w nowej wersji biblioteki. Byłem również odpowiedzialny za przygotowanie prawie całego tekstu manuskryptu.

PCJ 5.3.0

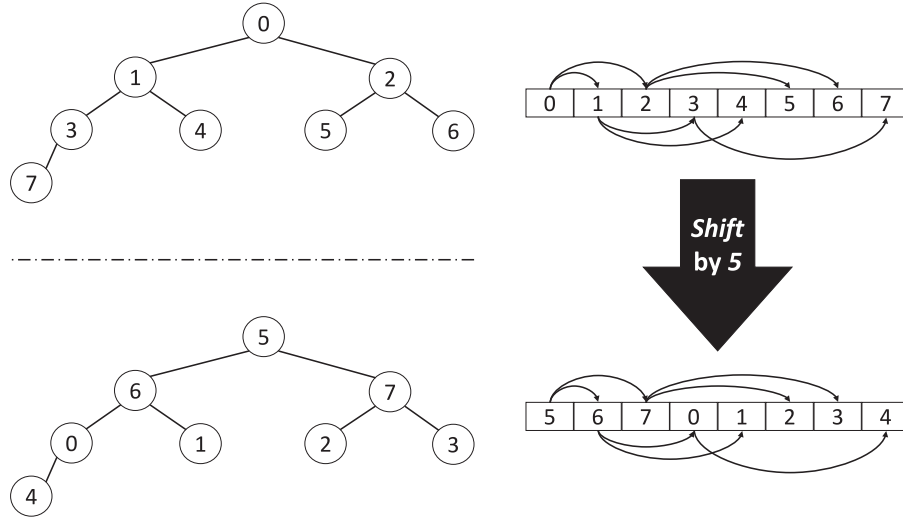
Już od dłuższego czasu trwały prace nad dodaniem kolejnych operacji kolektywnych do interfejsu biblioteki PCJ. W wersji 5.1.0 bibliotekę wyposażyłem w operacje `reduce(ReduceOperation<T> function, Enum<?> name)` oraz `collect(Enum<?> name)`, natomiast w wersji 5.2.0 dodałem operację `scatter(...)`. Wersja 5.3.0 przyniosła bardzo istotne rozszerzenia funkcjonalne w tym zakresie. W szczególności zmodyfikowałem sygnaturę metody `scatter`, która przyjmuje obecnie parametr w postaci mapy: `scatter(Map<Integer, T> newValueMap, Enum<?> name)`. Dodatkowo zmieniłem nazwę metody `collect` na `gather`, aby uniknąć nieporozumień wśród programistów języka Java – w *Java Streams*, wprowadzonych w Java 8, istnieje metoda `collect` o szerszym znaczeniu i możliwościach. Jednocześnie wprowadziłem nową metodę `collect(SerializableSupplier<Collector<T, ?, R>> collectorSupplier, Enum<?> variable)`, której działanie jest zbliżone do wspomnianej metody `collect` z *Java Streams*. Każda z powyższych operacji posiada również swój asynchroniczny odpowiednik.



Rysunek 14: Schemat komunikacji węzłów w PCJ. Każdy węzeł jest przedstawiony jako prostokąt z wpisanymi liczbami – większa liczba oznacza identyfikator węzła, a mniejsze to identyfikatory wątków. Różne rodzaje strzałek ilustrują połączenia sieciowe nawiązane w fazie rozruchu aplikacji: strzałka łamana wskazuje połączenie z węzłem #0; pogrubiona, prosta strzałka reprezentuje połączenie z węzłem potomnym tworzące binarne drzewo komunikacji; zakrzywione strzałki oznaczają pozostałe połączenia między parami węzłów. (z [6])

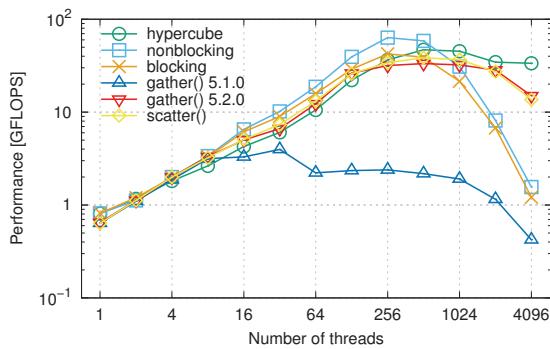
Wprowadzone operacje kolektywne implementują najbardziej wydajną komunikację za pomocą zrównoważonego drzewa binarnego (Rysunek 14). Operacje kolektywne są etapowe – wykonywane na poziomie wątków i następnie węzłów. Przykładowo przy redukcji następuje najpierw zsumowanie wartości po wszystkich wątkach węzła, a następnie przekazanie zsumowanej wartości do węzła będącego rodzicem. Dzięki temu operacje kolektywne z wykorzystaniem n węzłów i t wątków na węzeł wykonywane są w czasie $O(\log n + t)$ i są dużo szybsze niż proste iterowanie po wątkach, gdzie złożoność czasowa wynosi $O(nt)$. Dodatkowo trzeba zwrócić uwagę, że operacje komunikacji, zwłaszcza przesyłania danych między węzłami, są czasochłonne i wprowadzenie etapowego wykonania operacji kolektywnych pozwala na szybsze wykonanie żądanej operacji.

W wersji 5.1.0 schemat komunikacji przy wykonaniu operacji kolektywnych był prosty. Węzeł, w którym operacja kolektywna została zainicjowana (*requester*), wysyłał żądanie do węzła #0, a następnie komunikacja odbywała się poprzez drzewo – od korzenia, przez węzły pośrednie, do liści, a następnie w kierunku powrotnym: od liści, przez węzły pośrednie, do korzenia. Wartość wyniku lub powiadomienie o zakończeniu operacji była gromadzona w węźle #0, a następnie przesyłana do węzła *requester*. Jednak w sytuacji, gdy wiele wątków próbowało jednocześnie wykonać operację kolektywną, wydajność spadała, ponieważ wszystkie korzystały z węzła #0 jako korzenia, który stawał się wąskim gardłem.

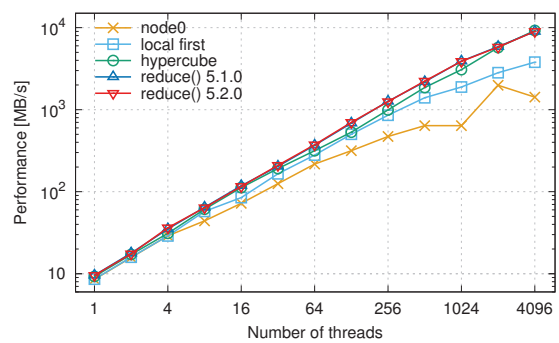


Rysunek 15: Schemat komunikacji dla operacji kolektywnych wykorzystujący przesunięcie węzłów w drzewie komunikacji. (z [6])

W wersji 5.2.0 wprowadziłem optymalizację, która polega na *przesunięciu* węzłów w drzewie komunikacji. Dzięki tej optymalizacji nie tylko węzeł *requester* staje się korzeniem drzewa, ale też całe drzewo jest przebudowane, co znacząco poprawiło wydajność komunikacji. Rysunek 15 przedstawia zarówno oryginalne, jak i *przesunięte* drzewa, w których węzeł *requester* to węzeł #5.



(a) FFT na 2^{27} elementach



(b) WordCount dla pliku 10 MB w reżimie słabej skalowalności.

Rysunek 16: Wyniki wydajnościowe z różnymi implementacjami operacji kolektywnych. (z [6])

Na Rysunku 16 zostały przedstawione wyniki wydajnościowe aplikacji *FFT* i *WordCount*. Wyniki zebrałem korzystając z superkomputera *Okeanos* (ICM UW).

Rysunek 16a przedstawia wyniki wydajnościowe jednowymiarowego *FFT* dla 2^{27} elementów. Porównałem na nim rozwiązania wykorzystujące trzy samodzielnie zaimplementowane przez współautora pracy schematy komunikacji *wszyscy-do-wszystkich* oraz nowo wprowadzone do biblioteki

PCJ operacje kolektywne: `gather(...)` i `scatter(...)`. Łatwo zauważyć poprawę wydajności operacji *gather* wynikającą ze zmiany wersji biblioteki z 5.1.0 do 5.2.0. Wersja korzystająca z *przesuniętego* drzewa osiąga wydajność niemal porównywalną z dopracowaną implementacją opartą na *hypercube*. Słaba wydajność *gather* w wersji 5.1.0 wynika z wąskiego gardła, jakim jest węzeł #0. W wersji 5.2.0 wydajność metod *gather* i *scatter* jest nieco niższa niż w przypadku *hypercube*, ponieważ wersja *hypercube* jest zoptymalizowana do konkretnego zadania i wysyła dane bezpośrednio do odpowiednich wątków, podczas gdy *gather* i *scatter* angażują do komunikacji wszystkie węzły. Z drugiej strony, kod korzystający z *gather* i *scatter* jest bardziej czytelny i łatwiejszy w utrzymaniu.

Rysunek 16b przedstawia wyniki wydajności dla problemu *WordCount*. Do analizy wykorzystałem tekst powieści *Artamène ou le Grand Cyrus* autorstwa Georges de Scudéry’ego, zakodowany w standardzie ISO 8859-1, o rozmiarze 10 MB. Pomiary przeprowadziłem w reżimie słabej skalowalności, w którym każdy wątek odczytuje cały plik, a na koniec obliczeń wykonywana jest operacja redukcji. Podobnie jak w teście *FFT*, porównałem rozwiązania wykorzystujące trzy samodzielnie zaimplementowane przez współautora pracy schematy redukcji oraz nowo wprowadzone operacje kolektywne: `gather(...)` i `scatter(...)`. Wyniki wskazują na dobrą skalowalność implementacji PCJ, choć wydajność stopniowo spada wraz ze wzrostem liczby wątków ze względu na zwiększający się rozmiar danych przekazywanych w komunikacji. Skalowalność jest niemal liniowa w przypadku implementacji *hypercube* i *reduce()*, co potwierdza wysoką efektywność części redukcyjnej algorytmu. W tym przypadku nie zaobserwowano różnic w wydajności między wersjami 5.1.0 i 5.2.0 biblioteki PCJ, ponieważ operacja redukcji jest wykonywana jednorazowo przez wątek #0. Pozostałe dwie implementacje charakteryzują się niższą wydajnością.

Operacje kolektywne wprowadzone w wersji 5.1.0 biblioteki PCJ i udoskonalone w wersjach 5.2.0 i 5.3.0, redukują nakład pracy programistycznej i umożliwiają jeszcze łatwiejsze tworzenie wydajnego i skalowalnego kodu.

4.1.7 HPV-KITE [7] – PCJ 5.3.3

Rozwój biblioteki PCJ pozwala na tworzenie aplikacji do uruchamiania na systemach wielordzeniowych i wielowęzłowych w sposób elastyczny i intuicyjny.

W ramach projektu HEAP (Human Exposome Assessment Platform), w którym przez ostatni rok projektu pełniłem funkcję kierownika pakietu roboczego WP9, powstała ostatnia praca z cyklu [7] opisująca stworzone przeze mnie rozwiązanie o nazwie HPV-KITE.

HPV-KITE to aplikacja umożliwiająca błyskawiczne przeszukiwanie dużych zbiorów danych genomowych w celu identyfikacji w nich sekwencji wirusa brodawczaka ludzkiego (HPV, *Human papillomaviruses*). Została opracowana do szybkiego i dokładnego wykrywania wirusów typu HPV w praktycznie dowolnych danych pochodzących z sekwencjonowania nowej generacji (NGS, ang. *Next-Generation Sequencing*). W aplikacji rdzeń związany z równoległością stworzyłem z wykorzystaniem biblioteki PCJ.

W dotychczasowych rozwiązaniach, po otrzymaniu danych sekwencyjnych z urządzenia, należało je jeszcze przetworzyć, m.in. wykonując usuwanie fragmentów sekwencji związanych z sekwencjami ludzkimi. Wstępne przetwarzanie danych, w zależności od ich wielkości (przykładowo 260 plików, łącznie 302 GB), mogło zajmować cały tydzień. W przypadku opracowanego rozwiązania wstępne przetwarzanie nie jest konieczne i analiza tych samych danych zajęła niecałe 16 minut korzystając z 8 węzłów superkomputera Puhti (CSC, Finlandia) dając porównywalne rezultaty jakościowe.

Główną ideą stojącą za HPV-KITE jest wyliczanie indeksu Tversky’ego (ang. *Tversky index*) [O26] zbiorów zawierających *shingle* (ciągi n -elementowe, ang. *shingles*; inaczej: *k-mers*, *n-grams*) powstałych z przetworzenia referencyjnej bazy danych (format FASTA) i sekwencji wejściowych (*sample*; format FASTQ). Tworzenie zbioru *shingli* polega na przejściu przez sekwencję oknem kroczącym (z krokiem wynoszącym 1) o stałej szerokości (domyślnie 31 znaków) i dodawanie widocznych ciągów do zbioru. W omawianej w pracy implementacji wykorzystane są wbudowane klasy implementujące interfejs `java.util.Set` do przechowywania zbiorów *shingli* i wyliczania mocy ich przekroju, ale trwają prace nad zastosowaniem w tym celu słownikowych struktur danych [A9].

Mając te zbiory można policzyć indeks Tversky’ego w następujący sposób:

$$S(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X \setminus Y| + \beta|Y \setminus X|}$$

gdzie X jest zbiorem powstałym z danych wejściowych (pliki FASTQ), Y jest zbiorem referencyjnej bazy zawierającej sekwencje rodziny wirusów HPV, natomiast α (alfa) i β (beta) są parametrami (wagami) indeksu Tversky’ego. W pracy parametry α i β zostały ustawione odpowiednio na 0 i 1, ponieważ celem jest identyfikacja sekwencji w pliku FASTQ (X), które wykazują wysokie podobieństwo do znanych sekwencji HPV obecnych w bazie referencyjnej (Y), przy czym jedynie elementy unikatowe dla bazy referencyjnej (Y) mają wpływ na wartość współczynnika podobieństwa. W związku z tym równanie można uprościć do następującej postaci:

$$C(X, Y) = \frac{|X \cap Y|}{|Y|}$$

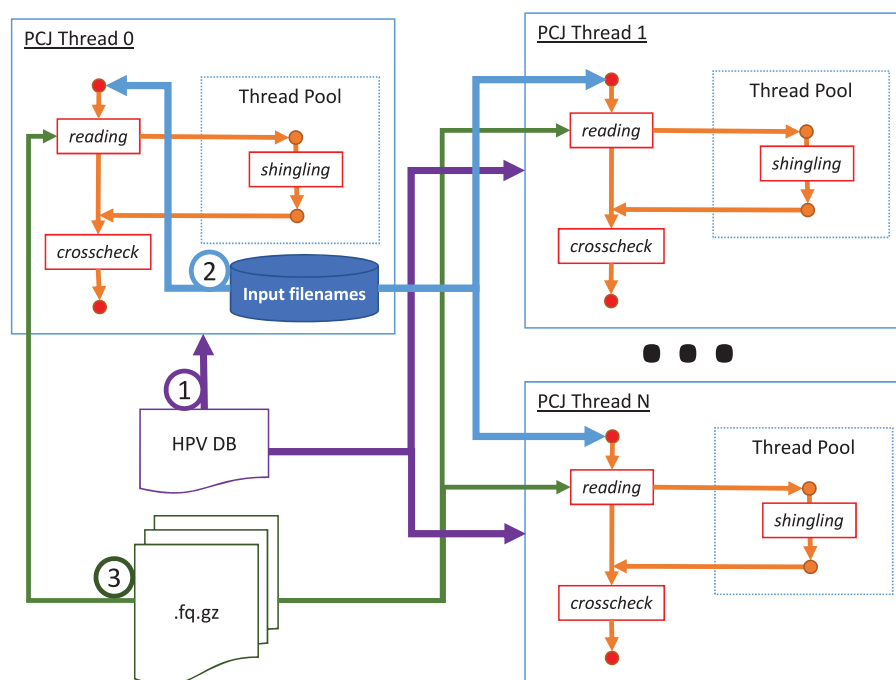
Obliczając wartość według powyższego wzoru można wyznaczyć *pokrycie* (ang. *coverage*), rozumiane jako liczbę unikatowych *shingli* należących do danego typu HPV obecnego w bazie danych, dla analizowanej sekwencji wejściowej. Do zaliczenia próbki jako zarażonej, ustaliliśmy próg na poziomie 10% pokrycia wirusowego (co odpowiada pokryciu 10% genomu HPV). Jako, że wartości są liczone osobno dla każdego wirusa, w wyniku dostajemy również informacje o współzakażeniach wieloma typami HPV. Dokładna liczba pokrytych k -merów zależy od konkretnego wirusa HPV oraz szerokości okna kroczącego. W domyślnej konfiguracji, przy szerokości okna równej 31, w używanej referencyjnej bazie HPV (PaVE [O27] z 23 lutego 2024 r.), wirus HPV213 charakteryzuje się najmniejszą liczbą unikatowych *shingli* (7065), natomiast HPV83 – największą liczbą unikatowych *shingli* (8073), zatem próg 10% odpowiada wartościom powyżej 706 i 807 *shingli*. Średnio wirusy mają około 7484 *shingli*, natomiast mediana liczby *shingli* wynosi 7367.

Zaletą stworzonego rozwiązania, poza szybkością i dokładnością wyników, jest możliwość wykorzystania wielu węzłów obliczeniowych dzięki zastosowaniu biblioteki PCJ. W rozwiązaniu tym zastosowano PCJ w najnowszej dostępnej do dziś wersji 5.3.3. Kod rozwiązania opiera się o Java 17, a testy były wykonywane z użyciem OpenJDK Runtime Environment w wersji 21.0.2.

PCJ 5.3.3

Zmiany wprowadzone od wersji 5.3.0 do aktualnie najnowszej wersji biblioteki 5.3.3 miały przede wszystkim charakter porządkujący. W wersji 5.3.1 poprawiłem mechanizm dodawania węzłów w `ExecutionBuilder` – tak, aby był bardziej odporny na dane przekazywane przez użytkowników (m.in. usuwanie białych znaków z początku i końca nazwy węzła, sprawdzanie poprawności nazwy, brak możliwości przekazania pustej nazwy oraz pomijanie pustych linii w pliku z listą węzłów). Wersja 5.3.2 ułatwiła korzystanie z adnotacji `@Storage` i `@RegisterStorage`, umożliwiając pominięcie ich parametrów – wartości zostają wówczas

automatycznie wywnioskowane z kontekstu. W wersji 5.3.3 ujednoliciłem nazewnictwo metod służących do lokalnych operacji na zmiennych współdzielonych; przykładowo, metoda `getLocal` zmieniła nazwę na `localGet` (analogicznie do `asyncGet`).

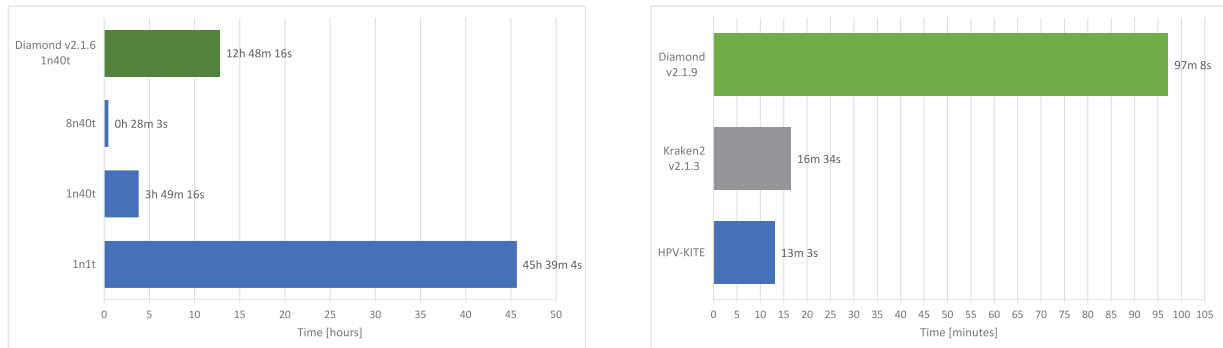


Rysunek 17: Schemat działania HPV-KITE. (z [7])

Rysunek 17 przedstawia schemat działania aplikacji. Do optymalnego przetwarzania równoległego zastosowałem algorytm równoważenia obciążenia (ang. *load-balancing*). Każdy węzeł uruchamia pojedynczy wątek PCJ odpowiedzialny za równoległe przetwarzanie plików wejściowych przy wykorzystaniu lokalnych wątków. Parametry wykonania, takie jak nazwa pliku bazy danych HPV, długość *shingla* czy lista plików, były przekazywane do procesu głównego. Wątek nadzorujący (wątek PCJ #0) tworzy obiekt współdzielony, którego zadaniem jest przechowywanie listy plików przeznaczonych do przetworzenia. Jest to obiekt klasy `java.util.concurrent.ConcurrentLinkedQueue` ze względu na jej właściwości zapewniające bezpieczeństwo współbieżnego dostępu. Następnie wszystkie wątki jednocześnie odczytują referencyjną bazę danych (początkowo planowano, aby wątek nadzorujący przetwarzał bazę referencyjną i przekazywał ją pozostałym wątkom, jednak testy wydajnościowe wykazały, że takie rozwiązanie jest wolniejsze). Mając gotową bazę referencyjną, każdy wątek PCJ pobiera od wątku nadzorującego ścieżkę do kolejnego pliku do przetworzenia. Pobieranie ścieżki do kolejnych plików odbywa się za pomocą mechanizmu pozwalającego na zlecenie wykonania zadań w kontekście innego wątku PCJ. Mechanizm ten jest dostępny przez wywołanie metody `at` (lub w wersji asynchronicznej: `asyncAt`). Po zakończeniu przetwarzania danego pliku wątek żąda ścieżki do następnego pliku, aż do momentu przetworzenia wszystkich plików, po czym kończy działanie.

W części udostępnionych mi danych pojedyncza próbka (*sample*) składała się z kilku plików (zazwyczaj 2, 4 lub 8). Przykładowo, 8 plików dla jednej próbki powstaje w sytuacji, gdy podczas sekwencjonowania wykorzystano cztery *lane'y* (kanały, ścieżki sekwencjonowania), a każdy *lane* zawiera odczyty w obu kierunkach (*paired-end*). Wszystkie te pliki można jednak jednoznacznie powiązać z daną próbką, ponieważ mają wspólny prefiks nazwy.

W rozwiązaniu HPV-KITE zaprojektowałem i zaimplementowałem również obsługę przetwarzania równoległego próbek składających się z wielu plików, co umożliwia uzyskanie pojedynczego wyniku dla każdej z nich. Przykładowo, w przypadku 8 plików składających się na jedną próbkę i 8 wątków PCJ przetwarzających dane, każdy wątek PCJ przetwarza osobno jeden plik, a następnie generowany jest wynik skumulowany dla danej próbki.



(a) Całkowity czas przetwarzania całego fińskiego zbioru sekwencji dla HPV-KITE i Diamond na superkomputerze Puhti. (b) Całkowity czas przetwarzania podzbioru TCGA dla HPV-KITE, Diamond i Kraken2 na superkomputerze Mahti.

Rysunek 18: Czasy przetwarzania danych sekwencyjnych za pomocą HPV-KITE. Niższa wartość oznacza lepszy wynik. (z [7])

Rysunek 18a zawiera czasy przetwarzania zbioru sekwencji z fińskiego badania klinicznego szczepionki przeciwko HPV (96 próbek) na superkomputerze Puhti (CSC, Finlandia). Pojedynczy węzeł składał się z dwóch 20-rdzeniowych procesorów Intel Xeon Gold 6230 CPU @ 2.10 GHz (łącznie 40 rdzeni) i 384 GB pamięci RAM. Wyniki wskazały, że poleganie wyłącznie na jednym węźle i wątku skutkuje nieoptymalną wydajnością, co podkreśliło konieczność zrównoleglenia rozwiązania. Zrównoleglenie generowania *shingli* znacznie zwiększyło wydajność – zbiór danych został przetworzony w około 4 godziny (11 razy szybciej), a porównanie tej wartości do czasu wykonania Diamond dało ponad trzykrotną poprawę. Co więcej, rozwinięcie tego rozwiązania o uruchomienie wielowęzłowe skróciło czas obliczeń do mniej niż pół godziny.

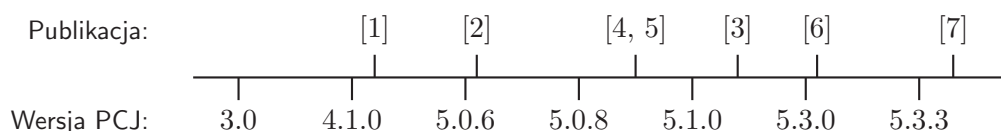
Rysunek 18b zawiera czasy przetwarzania podzbioru sekwencji z bazy projektu *The Cancer Genome Atlas* (TCGA). TCGA zawierają zsekwencjonowane pełne genomy ponad 20 000 próbek nowotworowych. Z całej bazy wybraliśmy wyłącznie próbki z guzami pierwotnymi szyjki macicy (304 próbki) i endometrium (590 próbek) – łącznie 894 próbki. Do przetwarzania wykorzystany był pojedynczy węzeł superkomputera Mahti (CSC, Finlandia) z dwoma 64-rdzeniowymi procesorami AMD EPYC 7H12 @ 2.6 GHz (łącznie 128 rdzeni) i 256 GB pamięci RAM. Najbardziej wydajnym rozwiązaniem okazał się HPV-KITE kończąc przetwarzanie w 13 minut, następnie Kraken2 (16 minut) a najwolniejszy okazał się Diamond (97 minut).

W pracy przedstawiliśmy również wyniki oceniające poprawność generowanych wyników przez HPV-KITE. Wyniki eksperymentów pokazują, że HPV-KITE zapewnia dokładne wykrywanie (czułość > 96%) najliczniejszych typów HPV w metagenomie DNA i RNA w porównaniu ze standardowymi i wymagającymi dużej mocy obliczeniowej algorytmami mapowania bezpośredniego, takimi jak Diamond. Ponadto, w porównaniu z genotypowaniem opartym na PCR z wykorzystaniem testu Luminex, HPV-KITE osiągnął czułość 89% i swoistość 91% w wykrywaniu wszystkich przypadków zakażenia pojedynczym i wieloma wirusami HPV (18 próbek).

Podsumowanie

W przedstawionym cyklu publikacji zaprezentowałem rozwój i analizę biblioteki PCJ, będącej implementacją paradygmatu PGAS dla języka Java, w kontekście jej zastosowań w różnych obszarach obliczeń równoległych w środowiskach wielowęzłowych. Prace w cyklu obejmują m.in. zastosowania bioinformatyczne (PCJ-BLAST, HPV-KITE), badania nad wykorzystaniem środowisk chmurowych, integrację obciążeń HPC, Big Data i sztucznej inteligencji, implementację i analizę algorytmu TeraSort w modelu PGAS, a także rozwój mechanizmów operacji kolektywnych w bibliotece PCJ. Osiągnięcia te stanowią istotny wkład w rozwój badań nad programowaniem równoległym, ze szczególnym uwzględnieniem zastosowań w naukach przyrodniczych.

Kolejne wersje biblioteki PCJ wraz z odpowiadającymi im publikacjami naukowymi wchodzącymi w skład cyklu zostały przedstawione na Rysunku 19.



Rysunek 19: Zestawienie kolejnych wersji biblioteki PCJ i powiązanych z nimi publikacji wchodzących w skład cyklu.

4.2 Przegląd pozostałych publikacji

W niniejszej podsekcji przedstawiam zorganizowany tematycznie krótki przegląd moich pozostałych publikacji.

4.2.1 Inne prace

Wśród innych prac dotyczących rozwoju biblioteki PCJ, nieujętych w głównej części omówienia wyników, znajdują się publikacje i abstrakty opublikowane przed uzyskaniem przeze mnie stopnia doktora [B1, B2, B3, B4, B5, B6] oraz po jego uzyskaniu [A10, A11]. Dodatkowo, w pracach [A12, A13, A14] przedstawiliśmy implementacje algorytmów grafowych z wykorzystaniem biblioteki PCJ. W publikacji [A15] opisaliśmy wersję 4.1 biblioteki oraz, między innymi, omówiliśmy fragmenty implementacji testu *MolDyn* z pakietu *Java Grande Benchmark Suite*, służącego do symulacji ruchu cząstek w problemie *N-body*. W pracy [A16] zaprezentowaliśmy możliwości implementacji mechanizmów odporności na awarie (*Fault Tolerance*) w ramach biblioteki PCJ, natomiast w publikacji [A17] przedstawiliśmy wyniki badań nad wydajnością biblioteki w obliczeniach prowadzonych na specjalnych, wysokowielordzeniowych procesorach zaprojektowanych z myślą o masywnej równoległości, tj. *Intel Xeon Phi Knights Landing (KNL)*.

Praca [A18] stanowi krótki raport techniczny z grantu uzyskanego w *Future SOC Lab HPI* przez Bensona Muite z Uniwersytetu w Tartu, w ramach którego uruchamiałem zadania typu *PingPong* z wykorzystaniem biblioteki PCJ, w celu weryfikacji możliwości uruchamiania aplikacji PCJ oraz wydajności komunikacji na klastrze o przepustowości sieci 10 Gb/s.

Praca [A19] jest abstraktem z *e-posterowej* sesji zorganizowanej podczas *55th European Society of Human Genetics (ESHG) Conference* i krótko przedstawia aktualny na rok 2022 stan badań grupy roboczej WP9 w ramach projektu HEAP.

4.2.2 Edukacja

ZawodyWeb Prace [E1, E2], opublikowane przed uzyskaniem stopnia doktora (w trakcie studiów magisterskich), przedstawiają system *ZawodyWeb*, którego jestem autorem. System ten umożliwia automatyczną weryfikację poprawności rozwiązań programistycznych. W przedstawionej wersji obsługiwał on języki takie jak Pascal, C, C++, Java, PHP i Python, a także pozwalał na ręczne ocenianie rozwiązań. System *ZawodyWeb* ułatwiał sprawdzanie zadań algorytmicznych opracowanych z wykorzystaniem technologii internetowych w ramach *Konkursu Informatycznego WMiI UMK*. Był również stosowany m.in. podczas zajęć *Uniwersyteckiego Koła Informatycznego* prowadzonych dla uczniów szkół średnich i gimnazjalnych.

Praca [E3], opublikowana została również przed uzyskaniem stopnia doktora. Opisuje ona dalszy rozwój systemu *ZawodyWeb* oraz jego zastosowanie w procesie dydaktycznym na studiach stacjonarnych i niestacjonarnych kierunku Informatyka na Wydziale Matematyki i Informatyki UMK.

Praca [E4], opublikowana już po uzyskaniu stopnia doktora, prezentuje rozwój systemu *ZawodyWeb* w kontekście nauczania programowania równoległego oraz jego integrację z oprogramowaniem UNICORE, umożliwiającym zlecanie zadań na infrastrukturze wielowęzłowej. Pokazaliśmy w niej, że system *ZawodyWeb* pozwala na weryfikację poprawności rozwiązań wykorzystujących technologie OpenMP i MPI w językach C i C++ oraz rozwiązań stworzonych w języku Java wykorzystujących bibliotekę PCJ.

Szkoła Prace [E5, E6], opublikowane przed uzyskaniem stopnia doktora, opisują projekt *Regionalne Koło Informatyczne* – pierwsze w kraju tego typu przedsięwzięcie mające na celu popularyzację i naukę programowania wśród uczniów szkół ponadgimnazjalnych. Zajęcia prowadzone na Wydziale Matematyki i Informatyki UMK realizowaliśmy w nowatorskiej formule zdalnego nauczania. Wykorzystywaliśmy w tym celu m.in. system *ZawodyWeb*.

Praca [E7], opublikowana po uzyskaniu stopnia doktora, zawiera krótki opis warsztatów przeprowadzonych podczas konferencji *Informatyka w Edukacji XV* w 2018 roku, poświęconych nauce podstaw programowania w języku Java.

Praca [E8], również opublikowana po uzyskaniu stopnia doktora, omawia zagadnienia nauczania programowania równoległego w szkołach ponadpodstawowych i na uczelniach wyższych. Zaprezentowaliśmy w niej bibliotekę PCJ oraz miniklaster zbudowany z minikomputerów Raspberry Pi jako platformę umożliwiającą efektywne wprowadzenie nauczania programowania równoległego już na etapie szkoły średniej.

Scratch W pracy [E9], opublikowanej przed uzyskaniem stopnia doktora, przedstawiliśmy wizualne środowisko programistyczne *Scratch*. Zaprezentowaliśmy w niej podstawowe zasady tworzenia aplikacji w tym środowisku oraz przykłady jego zastosowania w nauce programowania w szkołach podstawowych i gimnazjach.

Praca [E10], opublikowana po uzyskaniu stopnia doktora, zawiera krótki opis warsztatów przeprowadzonych podczas konferencji *Informatyka w Edukacji XV* w 2018 roku, poświęconych nowej wersji środowiska *Scratch*.

W pracy [E11], również opublikowanej po uzyskaniu stopnia doktora, przedstawiłem scenariusz oraz doświadczenia z realizacji zajęć w przedszkolu w ramach *Dnia Scratcha* w latach 2019 i 2022.

Praca [E12], opublikowana po uzyskaniu stopnia doktora, zawiera pełny opis warsztatów przeprowadzonych przeze mnie podczas konferencji *Informatyka w Edukacji XX* w 2024 roku.

Uczestnicy warsztatów zostali zapoznani z wizualnym językiem programowania *Scratch* oraz krok po kroku zostali poprowadzeni do stworzenia w tym środowisku gry typu *Wąż*.

Inne prace opublikowane po uzyskaniu stopnia doktora

- [A1] Marek Nowicki and Piotr Bała. Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ. In Łukasz Koźmiński, Jacek Daskoczek, and Piotr Kardasz, editors, *Innowacje w polskiej nauce w obszarze matematyki i informatyki : przegląd aktualnej tematyki badawczej / red. nauk. Łukasz Koźmiński, Jacek Daskoczek, Piotr Kardasz*, pages 130–140. 2016.
- [A2] Marek Nowicki, Davit Bzhalava, and Piotr Bała. Massively parallel sequence alignment with BLAST through work distribution implemented using PCJ library. In Shadi Ibrahim, Kim-Kwang Raymond Choo, Zheng Yan, and Witold Pedrycz, editors, *Algorithms and architectures for parallel processing : 17th International Conference, ICA3PP 2017, Helsinki, Finland, August 21-23, 2017 : proceedings / eds. Shadi Ibrahim, Kim-Kwang Raymond Choo, Zheng Yan, Witold Pedrycz*, volume 10393 of *Lecture Notes In Computer Science*, pages 503–512. Springer International Publishing, 2017.
- [A3] Marek Nowicki, Łukasz Górski, and Piotr Bała. Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud. In Bartosz Balis, Dora B. Heras, Laura Antonelli, Andrea Bracciali, Thomas Gruber, Jin Hyun-Wook, Michael Kuhn, Stephen L. Scott, Didem Unat, and Roman Wyrzykowski, editors, *Euro-Par 2020 : Parallel Processing Workshops : Euro-Par 2020 International Workshops : Warsaw, Poland, August 24-25, 2020 : revised selected papers / ed. by Bartosz Balis, Dora B. Heras, Laura Antonelli, Andrea Bracciali, Thomas Gruber, Jin Hyun-Wook, Michael Kuhn, Stephen L. Scott, Didem Unat, Roman Wyrzykowski*, number 12480 in *Lecture Notes In Computer Science*, pages 213–224, 2021.
- [A4] Marek Nowicki, Łukasz Górski, and Piotr Bała. Performance evaluation of parallel computing and Big Data processing with Java and PCJ library. In *Visionary Computing: CUG 2018, Stockholm, Sweden, 20-24 May 2018.*, pages 1–14. Cray User Group, 2018.
- [A5] Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, and Piotr Bala. Big Data analytics in Java with PCJ library : performance comparison with Hadoop. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski*, volume 10778 of *Lecture Notes In Computer Science*, pages 318–327. Springer International Publishing, 2018.
- [A6] Marek Nowicki. Benchmarking Java on Ethernet Cluster: benchmarking PCJ – a library for parallel computing in Java. In Christoph Meinel, Andreas Polze, Karsten Beins, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, and Jürgen Müller, editors, *HPI Future SOC Lab: proceedings 2019*, volume 158 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 263–271. Universitätsverlag Potsdam, 2024.
- [A7] Marek Nowicki. Benchmarking the sort algorithm on Ethernet Cluster : Benchmarking PCJ : a library for parallel computing in Java. In Christoph Meinel, Andreas Polze, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, Jürgen Müller, and Jürgen Sommer, editors, *HPI Future SOC Lab*

- : *proceedings 2020*, volume 159 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 53–63. Universitätsverlag Potsdam, 2024.
- [A8] Marek Nowicki. Benchmarking the TeraSort algorithm on Ethernet Cluster : Benchmarking Apache Hadoop based implementation. In Christoph Meinel, Andreas Polze, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, Jürgen Müller, and Jürgen Sommer, editors, *HPI Future SOC Lab : proceedings 2020*, volume 159 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 97–108. Universitätsverlag Potsdam, 2024.
- [A9] Marcin Piątkowski, Magdalena Mroczek, and Marek Nowicki. Słownikowe struktury danych i ich zastosowania. In Anna Beata Kwiatkowska, editor, *Informatyka w edukacji : edukacja informatyczna a kompetencje cyfrowe XXI wieku*, pages 80–90, 2025.
- [A10] Marek Nowicki and Piotr Bała. Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ. In *Materiały konferencyjne - Innowacyjne projekty badawcze: [II Ogólnopolska e-Konferencja Innowacyjnych Projektów Badawczych IPB]*, Wrocław, 2 września 2016., pages 30–30. Dolnośląski Akcelerator Technologii Innowacji Sp. z o.o., 2016.
- [A11] Marek Nowicki, Łukasz Górski, Magdalena Ryczkowska, and Piotr Bała. PCJ as a tool for massively parallel data processing. In *CGW Workshop’17, Kraków, Poland, October 23-25, 2017: proceedings / eds. Marian Bubak, Michał Turała, Kazimierz Wiatr.*, pages 39–40. Academic Computer Centre Cyfronet. AGH, 2017.
- [A12] Magdalena Ryczkowska, Marek Nowicki, and Piotr Bała. Level-synchronous BFS algorithm implemented in Java using PCJ Library. In Hamid R. Arabnia, Leonidas Deligiannidis, and Mary Yang, editors, *2016 International Conference on Computational Science and Computational Intelligence (CSCI 2016), 15-17 December 2016, Las Vegas, Nevada, USA : proceedings / ed. by Hamid R. Arabnia, Leonidas Deligiannidis, Mary Yang*, pages 596–601, 2016.
- [A13] Magdalena Ryczkowska, Marek Nowicki, and Piotr Bała. The performance evaluation of the Java implementation of Graph500. *Lecture Notes In Computer Science*, 9574:221–230, 2016.
- [A14] Magdalena Ryczkowska and Marek Nowicki. Performance comparison of graph BFS implemented in MapReduce and PGAS programming models. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski*, volume 10778 of *Lecture Notes In Computer Science*, pages 328–337. Springer International Publishing, 2018.
- [A15] Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, Michał Szynekiewicz, and Piotr Bała. PCJ : a Java library for heterogenous parallel computing. In *Recent advances in information science: proceedings of the 15th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS ’16) ; proceedings of the 10th International Conference on Computer Engineering and Applications (CEA ’16), Barcelona, Spain, February 13-15, 2016 / ed.: Xiaodong Zhuang.*, pages 66–72. World Scientific and Engineering Academy and Society Press, 2016.
- [A16] Michał Szynekiewicz and Marek Nowicki. Fault-tolerance mechanisms for the Java parallel codes implemented with the PCJ library. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International*

Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski, volume 10778 of *Lecture Notes In Computer Science*, pages 298–307. Springer International Publishing, 2018.

- [A17] Marek Nowicki, Łukasz Górski, and Piotr Bała. Evaluation of the parallel performance of the Java and PCJ on the Intel KNL based systems. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski*, volume 10778 of *Lecture Notes In Computer Science*, pages 288–297. Springer International Publishing, 2018.
- [A18] Marek Nowicki, Benson K. Muite, and Mehmet Can Boysan. Benchmarking FFT on an Ethernet Cluster: Latency, Bandwidth and MPI benchmarks. In Christoph Meinel, Andreas Polze, Karsten Beins, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, and Jürgen Müller, editors, *HPI Future SOC Lab: proceedings 2019*, volume 158 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 93–98. Universitätsverlag Potsdam, 2024.
- [A19] Magdalena Mroczek, Marek Nowicki, Łukasz Górski, Sara Arroyo Mühr, Alex Ormenisan, and Piotr Bała. High performance AI based tool for mining and classification of viral genomes in human exposome metadata. *European Journal of Human Genetics*, 31(suppl. 1):292–292, 2023.

Inne prace opublikowane przed uzyskaniem stopnia doktora

- [B1] Marek Nowicki and Piotr Bała. Parallel computations in Java with PCJ library. In *Proceedings of the 2012 International Conference on High Performance Computing & Simulation : HPCS 2012, July 2 - July 6 2012, Madrid, Spain / ed. Waleed W. Smari.*, pages 381–387. IEEE, 2012.
- [B2] Marek Nowicki and Piotr Bała. PCJ - new approach for parallel computations in Java. *Lecture Notes In Computer Science*, 7782:115–125, 2013.
- [B3] Łukasz Górski, Marek Nowicki, and Piotr Bała. Komunikacja wątków w bibliotece PCJ. In *XXVII Forum Informatyki Teoretycznej, Toruń, 11-14 kwietnia 2013.*, pages 28–28. [s.n.], 2013.
- [B4] Marek Nowicki and Piotr Bała. The performance of the PCJ library for massively parallel computing. In *Proceedings of the 7th International Conference on PGAS Programming Models, 3-4th October 2013, Edinburgh, UK / ed. by M. Weiland, A. Jackson and N. Johnson.*, pages 254–254. The University of Edinburgh, 2013.
- [B5] Rafał Kluszczyński, Marcin Stolarek, Grzegorz Marczak, Łukasz Górski, Marek Nowicki, and Piotr Bała. Development of parallel codes using PL-Grid infrastructure. In *Cracow '14 Grid Workshop, October 27-29 2014, Krakow, Poland.*, pages 81–82. Academic Computer Centre Cyfronet. AGH, 2014.
- [B6] Marek Nowicki, Łukasz Górski, Patryk Grabarczyk, and Piotr Bała. PCJ - Java library for high performance computing in PGAS model. In Waleed W. Smari and Vesna Zeljkovic, editors, *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014) / ed. by W.W. Smari, V. Zeljkovic*, pages 202–209. 2014.

Prace związane z edukacją

- [E1] Marek Nowicki, Rafał Kluszczyński, and Piotr Bała. Internetowe narzędzia do sprawdzania zadań programistycznych i ich zastosowania. In *V Konferencja "Informatyka w Edukacji" - IwE 2008, Toruń, 3-6 lipca 2008 / red. nauk. Anna Beata Kwiatkowska, Maciej M. Sysło.*, pages 88–94. Agencja Wydawnicza Argi, 2008.
- [E2] Rafał Kluszczyński, Łukasz Mikulski, Marek Nowicki, and Piotr Bała. Contests Hosting Service as a tool to teach programming. *Annales Universitatis Mariae Curie-Skłodowska. Sectio AI: Informatica*, 9:213–224, 2009.
- [E3] Marek Nowicki, Łukasz Mikulski, Marcin Piątkowski, Rafał Kluszczyński, and Piotr Bała. On-line contests hosting service as a tool to teach computer science students programming. In Nicholas Reynolds, Mary Webb, Valentina Dagiene, and Maciej Sysło, editors, *WCCE 2013 : 10th IFIP World Conference on Computers in Education, Toruń, Poland; July 1-7, 2013. Vol. 2 / eds. Nicholas Reynolds, Mary Webb, in coop. with Valentina Dagiene, Maciej M. Sysło*, pages 61–71. 2013.
- [E4] Marek Nowicki, Maciej Marchwiany, Maciej Szpindler, and Piotr Bała. On-line service for teaching parallel programming. *Lecture Notes In Computer Science*, 9523:78–89, 2015.
- [E5] Piotr Bała, Anna Kwiatkowska, Marek Nowicki, Mariusz Strzelecki, and Maciej Sysło. Regionalne Koło Informatyczne. In *Informatyka w edukacji: materiały z warsztatów VII Konferencji IwE 2010 / red. Anna Beata Kwiatkowska.*, pages 61–74. Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011.
- [E6] Marek Nowicki, Michał Matuszak, Anna Kwiatkowska, and Maciej Sysło. Teaching secondary school students programming using distance learning : a case study. In Nicholas Reynolds, Mary Webb, Valentina Dagiene, and Maciej Sysło, editors, *WCCE 2013 : 10th IFIP World Conference on Computers in Education, Toruń, Poland; July 1-7, 2013. Vol. 2 / eds. Nicholas Reynolds, Mary Webb, in coop. with Valentina Dagiene, Maciej M. Sysło*, pages 246–254. 2013.
- [E7] Piotr Bała, Łukasz Górski, and Marek Nowicki. Java w szkole. In Anna Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : myśl komputacyjnie! / red. Anna Beata Kwiatkowska, Maciej M. Sysło*, pages 171–172. 2018.
- [E8] Piotr Bała and Marek Nowicki. Czy jesteśmy gotowi na naukę programowania równoległego w szkole? In Anna Beata Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : edukacja informatyczna : wyzwania współczesnego świata*, pages 127–136, 2023.
- [E9] Piotr Bała and Marek Nowicki. Scratch - środowisko do nauki programowania. In *Informatyka w edukacji: materiały z warsztatów VII Konferencji IwE 2010 / red. Anna Beata Kwiatkowska.*, pages 33–40. Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011.
- [E10] Piotr Bała and Marek Nowicki. Scratch 3.0 : nowe możliwości. In Anna Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : myśl komputacyjnie! / red. Anna Beata Kwiatkowska, Maciej M. Sysło*, pages 173–174. 2018.
- [E11] Marek Nowicki. Scratch w przedszkolu. In Anna Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : uczniowie i nauczyciele w szkole przyszłości*, pages 414–424, 2022.

- [E12] Marek Nowicki. Scratch : podstawy na przykładzie gry Wąż. In Anna Beata Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : kształcenie informatyczne drogą do cyfrowej transformacji społeczeństwa*, pages 284–294, 2024.

5 Aktywność naukowa

Informacja o wykazywaniu się istotną aktywnością naukową albo artystyczną realizowaną w więcej niż jednej uczelni, instytucji naukowej lub instytucji kultury, w szczególności zagranicznej.

Poza zatrudnieniem na **Uniwersytecie Mikołaja Kopernika w Toruniu** jak również **Uniwersytecie Warszawskim** odbywałem staże w następujących jednostkach naukowych zagranicznych i krajowych, gdzie wykazywałem się istotną aktywnością naukową:

- wizyta badawcza w **ośrodku superkomputerowym HLRS** – High Performance Computing Center Stuttgart, University of Stuttgart, Niemcy (czerwiec-lipiec 2017);
- półroczny staż w laboratorium badawczym **IBM Zurich Research Laboratory** w Zurychu, w Szwajcarii (Mathematical & Computational Science Department) (maj-listopad 2013);
- semestralny staż w Laboratorium Maszyn Biomolekularnych Uniwersytetu Warszawskiego (październik 2011 – marzec 2012).

Poza zatrudnieniem w więcej niż jednej uczelni i odbywaniem staży naukowych, moja aktywność naukowa związana jest z współpracą m.in. z naukowcami z **Karolinska Institutet** (Szwecja), gdzie posiadałem status naukowca afiliowanego (*Affiliated Researcher*) w Centrum Eliminacji Raka Szyjki Macicy (*Center for Cervical Cancer Elimination*) na Wydziale Nauk Klinicznych, Interwencji i Technologii (*Department of Clinical Science, Intervention and Technology*), oraz Oulun Yliopisto (*University of Oulu*; Finlandia).

Brałem udział w międzynarodowych projektach badawczych:

- HEAP (Human Exposome Assessment Platform; finansowany ze środków Unii Europejskiej w ramach programu Horyzont 2020 - Grant agreement ID: 874662; DOI: 10.3030/874662): od 02/2021 – **wykonawca**, 05/2024-06/2025 – **kierownik** (ICM UW; WP9)
- HPDCJ (Heterogenous Parallel and Distributed Computing with Java; finansowany przez konsorcjum CHIST-ERA – w Polsce finansowane przez NCN: 2014/14/Z/ST6/0007): 10/2014-09/2017 – *technical coordinator of the work package* (**wykonawca**)
- APOS (Application Performance Optimization and Scalability; EU 7th Project Framework): 2011-2013 – programista (przed uzyskaniem stopnia doktora)
- Kardionet (EEA Funds): 2008-2011 – programista (w trakcie studiów magisterskich, przed uzyskaniem stopnia doktora)

Uczestniczyłem w konferencjach krajowych i międzynarodowych oraz szkołach letnich, gdzie wygłaszałem referaty, wykłady i przeprowadzałem *tutoriale*, m.in.:

- *HPV-KITE for high-performance metagenomics data analysis* (referat oraz poster), BioInformatics in Toruń 2025; 25-28/06/2025, Toruń

- *HPV-KITE for high-performance metagenomics data analysis*, EHEN Scientific Meeting 2024; 26-27/11/2024, Bruksela, Belgia
- *Parallel Programming in Java in the PCJ Library (wykład na zaproszenie)*, TSSP ExSci – Toruń Students Summer Program in Exact Sciences, 14/07/2022, Toruń
- *Big Data, processing and storage (wykład na zaproszenie)*, 4EU+ Against Cancer Summer School, 6/07/2022, Warszawa
- *PCJ - Java Library that integrates HPC, Big Data and AI workloads*, International Symposium on Grids & Clouds 2021 (ISGC2021); 22-26/03/2021; Academia Sinica Computing Centre (ASGC), Taipei, Taiwan (Online due to COVID-19)
- *Performance Evaluation of Java/PCJ Implementation of Parallel Algorithms on the Cloud, 26th International European Conference on Parallel and Distributed Computing (Euro-Par 2020)*, 24-28/08/2020, Warsaw, Poland (Online due to COVID-19)
- *Benchmarking Java on Ethernet Cluster / Benchmarking PCJ - a library for parallel computing in Java*, 19th HPI Future SOC Lab Day – Fall 2019, 12/11/2019, Poczdam, Niemcy
- *High Performance Computing And Big Data Analytics With PCJ And Java (tutorial)*, Supercomputing Frontiers Europe 2018; 12-15/03/2018; Warszawa, Polska
- *Evaluation of the parallel performance of the Java and PCJ on the Intel KNL based systems*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM'2017), 10-13/09/2017, Lublin, Poland
- *Parallel Computing in Java (tutorial)*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM'2017), 10/09/2017, Lublin, Poland
- *Massively Parallel Sequence Alignment with BLAST Through Work Distribution Implemented Using PCJ Library, 17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2017)*, 21-23/08/2017, Helsinki, Finlandia
- *Level-synchronous BFS Algorithm Implemented in Java Using PCJ Library*, The International Conference on Computational Science and Computational Intelligence (CSCI'16); 15-17/12/2016, Las Vegas, Nevada, USA
- *PCJ5 (wykład na zaproszenie)*, Seminarium zespołu, Cognitive Computing & Computational Sciences Department, IBM Research Lab – Zurich, 14/10/2016, Zürich, Szwajcaria,
- *Parallel Computing in Java (tutorial)*, 11th International Conference On Parallel Processing And Applied Mathematics (PPAM'2015), 6-9/09/2015, Kraków
- *The performance of the PCJ library for massively parallel computing (poster)*, 7th International Conference on PGAS Programming Models, 3-4/10/2013, Edinburgh, Scotland, Great Britain (przed uzyskaniem stopnia doktora)
- *PCJ – a Partioned Global Address Space Approach for Parallel Computations in Java*, Exascale Applications and Software Conference (EASC2013): Solving Software Challenges for Exascale, 9-11/04/2013 r., Edynburg, Szkocja, Wielka Brytania (przed uzyskaniem stopnia doktora)

- *Parallel computations in Java with PCJ library*, The 2012 International Conference on High Performance Computing & Simulation (HPCS 2012), 2-6/07/2012, Madrid, Hiszpania (przed uzyskaniem stopnia doktora)
- *PCJ – New Approach for Parallel Computations in Java*, PARA 2012: Workshop on State-of-the-Art in Scientific and Parallel Computing, 11-13/06/2012, Helsinki, Finlandia (przed uzyskaniem stopnia doktora)

Byłem również członkiem komitetów organizacyjnych i naukowych konferencji międzynarodowych, m.in.:

- **członek komitetu naukowego** warsztatów: Workshop on PGAS Programming, Konferencja PPAM 2019: 13th International Conference on Parallel Processing and Applied Mathematics, 8-11/09/2019, Białystok
- **członek komitetu organizacyjnego** międzynarodowej konferencji: Petri Nets 2016 // ACSD 2016: 37th International Conference On Applications And Theory Of Petri Nets And Concurrency, 16th International Conference On Application Of Concurrency To System Design, 19-24/06/2016 r., Toruń

Byłem **promotorem pomocniczym** dla doktorantki mgr. Magdaleny Ryczkowskiej, której tematem rozprawy było *Opracowanie w modelu PGAS wybranych, równoległych algorytmów grafowych i ich implementacja przy użyciu języka Java* (data obrony: 5 listopada 2018; data nadania stopnia: 8 listopada 2018; promotor: prof. dr hab. Piotr Bała; dziedzina i specjalność: nauki techniczne w zakresie informatyki; miejsce: Instytut Podstaw Informatyki Polskiej Akademii Nauk).

Moja aktywność naukowa związana jest również z **recenzowaniem prac naukowych** przesłanych do czasopism (m.in. *Bioinformatics*, *BMC Bioinformatics*, *Journal of Big Data*, *SoftwareX*) i na konferencje (*Petri Nets 2020*, *PPAM'2022*). **Recenzowałem też wnioski konkursowe** w ramach komisji konkursowej *Grants4NCUStudents* IV edycja (2022 r.) na zadania badawcze i wyjazdy zagraniczne realizowane przez studentów i doktorantów Uniwersytetu Mikołaja Kopernika w Toruniu – panel ST.

W czasie studiów magisterskich i doktoranckich (w latach 2009-2015) byłem członkiem towarzystwa naukowego *Association for Computing Machinery (ACM)* (ACM Student Member).

Pełna lista moich osiągnięć jest dostępna w dokumencie *Wykaz osiągnięć*.

6 Osiągnięciach dydaktyczne, organizacyjne oraz popularyzujące naukę

Informacja o osiągnięciach dydaktycznych, organizacyjnych oraz popularyzujących naukę lub sztukę.

- Prowadzone zajęcia dydaktyczne:
 - ze studentami studiów magisterskich na kierunku informatyka: *Inżynieria wsteczna oprogramowania* (zajęcia laboratoryjne), *Projekt badawczo-rozwojowy* (opiekun zespołu), *Wprowadzenie do obliczeń równoległych* (wykłady i zajęcia laboratoryjne), *Programowanie równoległe i rozproszone* (wykłady i zajęcia laboratoryjne), *Programowanie Zespołowe* (opiekun zespołu), *Wybrane Zastosowania Informatyki* (zajęcia laboratoryjne)

- ze studentami studiów licencjackich i inżynierskich na kierunku informatyka: *Programowanie równoległe i rozproszone* (wykłady i zajęcia laboratoryjne), *Programowanie III (Java)* (zajęcia laboratoryjne), *Programowanie II (C,C++)* (zajęcia laboratoryjne), *Podstawy Programowania* (zajęcia laboratoryjne), *Przygotowanie do Olimpiad* (zajęcia laboratoryjne)
- ze studentami kierunku matematyka, specjalność nauczanie matematyki i informatyki: *Programowanie II (Java)* (zajęcia laboratoryjne)
- na kursie podyplomowym: zajęcia praktyczne z zakresu *zastosowania narzędzi Big Data w analizach omiczych* na kursie podyplomowym *OMICS Data Science – Bioinformatyka i analiza wielkoskalowych danych biomedycznych*
- z dziećmi w ramach Uniwersytetu Dziecięcego: *zajęcia z programowanie gier w języku Scratch*, *Programowanie w języku Scratch*,
- przygotowywanie zadań i pomoc w prowadzeniu *Regionalnego Koła Informatycznego*
- Wykłady, warsztaty i inne aktywności na festiwalach naukowych:
 - *Kto najszybciej rozwiąże problem, mając tysiąc procesorów?* (wspólnie z prof. Piotrem Bałą), wykład na XIX Festiwal Nauki Warszawa 2015, 26 września 2015 r., Warszawa
 - *I choćby przyszło tysiąc atletów... czyli o programowaniu równoległym* (wspólnie z prof. Piotrem Bałą), warsztaty na XIX Festiwalu Nauki Warszawa 2015, 26 września 2015 r., Warszawa
 - pomoc w przeprowadzeniu *mini konkursu programistycznego* w ramach *24. Toruńskiego Festiwalu Nauki i Sztuki* w kwietniu 2006 r.
- Promotor wielu obronionych prac inżynierskich (15) i magisterskich (7)
- Udział w Akademickich Mistrzostwach Polski w Programowaniu Zespołowym, Wrocław, 26-27 października 2018 jako zastępca trenera (co-coach) trzech zespołów studentów studiów informatycznych Wydziału Matematyki i Informatyki UMK
- Administrowanie, utrzymanie i rozwijanie systemem *ZawodyWeb* do automatycznego testowania rozwiązań programistycznych studentów (od 10/2009)
- udział w pracach komisji konkursowych:
 - członek Komitetu Okręgowego w Toruniu *Olimpiady Informatycznej* (od 01/2014)
 - członek jury hackathonu *BrainCode 2019* organizowanego przez Allegro, 12-13 kwietnia 2019, Toruń
 - **przewodniczący** komisji konkursowej: *Hackaton Wielkie Wyzwania Programistyczne – edycja III: Switch to Smart*, ICM UW, 23-24 listopada 2018, Warszawa
 - członek Jury hackathonu *BrainCode 2018* organizowanego przez Allegro, 20-21 kwietnia 2018, Toruń
 - członek komisji sędziowskiej konkursu *FRI: III Kujawsko-Pomorskich Zawodów Programistycznych*, 20 kwietnia 2018, Toruń
 - **przewodniczący** komisji konkursowej: *Hackaton Wielkie Wyzwania Programistyczne – edycja II: Smart Charge*, ICM UW, 15-16 lutego 2018, Warszawa
 - członek jury hackathonu *BrainCode 2017* organizowanego przez Allegro, 23-24 marca 2017, Toruń

- członek komisji sędziowskiej konkursu *FRI: II Kujawsko-Pomorskich Zawodów Programistycznych*, 24 marca 2017, Toruń
- członek komisji konkursowej: Konkurs *Kodowanie ze Scratchem 2017* – dla uczniów szkół podstawowych, 26 maja 2017, Toruń
- **przewodniczący** komisji konkursowej: *Hackaton Wielkie Wyzwania Programistyczne* – edycja I: *Map the Gap*, ICM UW, 16-17 lutego 2017, Warszawa
- członek komisji konkursowej: Konkurs *Kodowanie ze Scratchem 2016* – dla uczniów szkół podstawowych, 17 czerwca 2016, Toruń
- członek komisji konkursowej: Konkurs *Kodowanie ze Scratchem 2015* – dla uczniów szkół podstawowych, 19 czerwca 2015, Toruń
- pomoc w przeprowadzaniu *Konkursu Informatycznego* (edycje: 2006/2007, 2007/2008, 2008/2009) na Wydziale Matematyki i Informatyki UMK
- artykuły przedstawiające wyniki prac naukowych w serwisach internetowych
 - artykuł opisujący HPV-KITE w **Forum Akademickie**
<https://forumakademickie.pl/naukowcy-z-uw-autorami-metody-szybkiego-wykrywania-sekwencji-wirusow-hpv/>
 - artykuł opisujący HPV-KITE w **Medonet**
<https://www.medonet.pl/biznes-system-i-zdrowie/digital-health-innovators,polscy-naukowcy-opracowali-hpv-kite--przelom-w-diagnostyce-wirusow-hpv,artykul,27980435.html>
 - artykuł opisujący HPV-KITE w **Serwisie Naukowym UW**
<https://serwisnaukowy.uw.edu.pl/na-tropie-hpv-matematyka-przychodzi-z-pomoca-medycynie/>
- artykuł w czasopiśmie popularnonaukowym
 - artykuł opisujący HPV-KITE w **czasopiśmie Wiedza i Życie**, Lipiec 2025 nr 7 (1087), ISSN 0137-8929: *HPV-KITE – wygrać z czasem* (str. 21)
- administrowanie serwerem i wsparcie w organizacji polskiej edycji Międzynarodowego Konkursu *Kangur Matematyczny*, rozwój oprogramowania zgłaszania uczestników, wyników i sprawozdań (od 09/2009)
- stworzenie polskiego tłumaczenia wizualnego środowiska programowania *Scratch* rozwijanego przez Lifelong Kindergarten Group w MIT Media Lab (08-10/2008)

7 Inne istotne osiągnięcia

Oprócz kwestii wymienionych w pkt. 1-6, wnioskodawca może podać inne informacje, ważne z jego punktu widzenia, dotyczące jego kariery zawodowej.

- Wdrożone technologie
 - system *ZawodyWeb* – wdrożony m.in. na Wydziale Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika w Toruniu
 - biblioteka *PCJ* – dostępna w centralnym repozytorium Maven, wdrażana m.in. w *International ImMunoGenetics Information System (IMGT)*

- aplikacja *PCJ-BLAST* – wersja binarna dostępna przy repozytorium w serwisie *GitHub*
 - aplikacja *HPV-KITE* – dostępna m.in. w repozytorium *Conda-Forge*
 - aplikacja *KITE* – dostępna m.in. w repozytorium *Conda-Forge*
 - wewnętrzny system wsparcia decyzji sprzedażowych **IBM** (*IBM wide Sales Decision Support system*) wykorzystywany przez ponad 5000 sprzedawców IBM na całym świecie
- Wyróżnienia i nagrody
 - **Indywidualne wyróżnienie Rektora** Uniwersytetu Mikołaja Kopernika w Toruniu za osiągnięcia uzyskane w dziedzinie naukowej w 2020 roku, 25 czerwca 2021 r.
 - **Indywidualna nagroda III stopnia Rektora** Uniwersytetu Mikołaja Kopernika w Toruniu za osiągnięcia w dziedzinie naukowo-badawczej w 2018 roku, 26 września 2019 r.
 - **Wyróżnienie za wybitny dorobek naukowy** w uczelnianym konkursie na priorytetowe zespoły badawcze, Uniwersytet Mikołaja Kopernika, 28 marca 2019 – nagroda zespołowa; zespół Formal Languages and Concurrency (FoLCo)
 - *HPC Challenge Class 2 Award: **Best Productivity – Most Elegant Solution*** na międzynarodowej konferencji *SC14 – the International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing 2014)* – 16-21 listopada 2014, New Orleans, LA, USA – za bibliotekę PCJ
 - Zwycięzca konkursu *IBM Great Minds student internships 2013*
 - Sun Certified Programmer for the Java Platform, SE 6 (data certyfikacji: 16.12.2009)



podpis wnioskodawcy

Literatura

- [O1] Karl Feind. Shared memory access (SHMEM) routines. *Cray Research*, 1995.
- [O2] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global Arrays: A Non-Uniform-Memory-Access Programming Model For High-Performance Computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.
- [O3] Laxmikant V. Kale and Gengbin Zheng. *Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects*, chapter 13, pages 265–282. John Wiley & Sons, Ltd, 2009. doi: 10.1002/9780470558027.ch13.

- [O4] William W. Carlson, Jesse M. Draper, David E. Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. Introduction to UPC and language specification. Technical report, Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, 1999.
- [O5] John Reid. The new features of Fortran 2008. In *ACM SIGPLAN Fortran Forum*, volume 27, pages 8–21. ACM, 2008.
- [O6] B.L. Chamberlain, D. Callahan, and H.P. Zima. Parallel Programmability and the Chapel Language. *The International Journal of High Performance Computing Applications*, 21(3):291–312, 2007. doi:10.1177/1094342007078442.
- [O7] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph Von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *Acm Sigplan Notices*, 40(10):519–538, 2005.
- [O8] Kathy Yelick, Luigi Semenzato, Geoff Pike, Carleton Miyamoto, Ben Liblit, Arvind Krishnamurthy, Paul Hilfinger, Susan Graham, David Gay, Phil Colella, et al. Titanium: a high-performance java dialect. *Concurrency and Computation: Practice and Experience*, 10(11-13):825–836, 1998.
- [O9] Olivier Tardieu. The apgas library: resilient parallel and distributed programming in java 8. In *Proceedings of the ACM SIGPLAN Workshop on X10*, pages 25–26. ACM, 2015.
- [O10] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.
- [O11] John Mellor-Crummey, Laksono Adhianto, Guohua Jin, Mark Krentel, Karthik Murthy, William Scherer, and Chaoran Yang. Class II submission to the HPC Challenge award competition Coarray Fortran 2.0. URL: http://www.hpcchallenge.org/presentations/sc2011/hpcc11_report_caf2_0.pdf.
- [O12] Łukasz Górski. *Adaptacja wybranych równoległych algorytmów do modelu PGAS i ich implementacja w języku Java z wykorzystaniem biblioteki PCJ*. PhD thesis, Instytut Podstaw Informatyki Polskiej Akademii Nauk, 2017.
- [O13] Leo Tolstoy. War and Peace. Accessed: 12.05.2021. URL: <http://www.gutenberg.org/files/2600/2600-0.txt>.
- [O14] Madeleine de Scudéry. Artamène ou le grand Cyrus. Accessed: 12.05.2021. URL: <https://github.com/xbsd/cyrusfiles/blob/master/cyrusprint.txt>.
- [O15] TOP500. Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect. Accessed: 30.10.2025. URL: <https://top500.org/system/178446/>.
- [O16] Martin Gardener. Mathematical Games: The fantastic combinations of John Conway’s new solitaire game ”Life”. *Scientific American*, 223(4):120–123, 1970.
- [O17] Jonas Posner, Lukas Reitz, and Claudia Fohry. Comparison of the HPC and Big Data Java Libraries Spark, PCJ and APGAS. In *2018 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM)*, pages 11–22. IEEE, 2018.
- [O18] Franciszek Rakowski and Jan Karbowski. Optimal synaptic signaling connectome for locomotory behavior in *Caenorhabditis elegans*: Design minimizing energy cost. *PLoS computational biology*, 13(11):e1005834, 2017.

- [O19] Ananth Grama, Vipin Kumar, Anshul Gupta, and George Karypis. *Introduction to parallel computing*. Pearson Education, 2003.
- [O20] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST Database of handwritten digits. Accessed: 17.03.2021. URL: <http://yann.lecun.com/exdb/mnist/>.
- [O21] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., 2017.
- [O22] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow, 2018. [arXiv:1802.05799](https://arxiv.org/abs/1802.05799).
- [O23] Janis Keuper and Franz-Josef Pfreundt. Asynchronous parallel stochastic gradient descent: A numeric core for scalable distributed machine learning algorithms. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 1. ACM, 2015.
- [O24] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI message passing interface standard. In *Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994*, pages 213–218. Springer, 1994.
- [O25] Message Passing Interface Forum. MPI: A message-passing interface standard, 1994.
- [O26] Amos Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.
- [O27] The PapillomaVirus Episteme (PaVE). <https://pave.niaid.nih.gov/>, Accessed: 2024-02-23.

Summary of Professional Accomplishments

Marek Nowicki

November 26, 2025

1 Name

Marek Nowicki

2 Diplomas, degrees conferred in specific areas of science

Diplomas, degrees conferred in specific areas of science or arts, including the name of the institution which conferred the degree, year of degree conferment, title of the PhD dissertation

- 2015 PhD degree – Doctor of Mathematical Sciences in Informatics,
awarded by the resolution of the Council of the Faculty of Mathematics, Informatics,
and Mechanics, University of Warsaw of June 18, 2015
Dissertation: *New programming methods for parallel programming in Java based on the
PGAS (Partitioned Global Address Space) paradigm*
(in Polish: *Opracowanie nowych metod programowania równoległego w Javie w oparciu
o paradygmat PGAS (Partitioned Global Address Space)*)
Supervisor: dr hab. Piotr Bała, prof. UW; Assistant Supervisor: dr Michał Chlebiej
- 2010 MSc in Computer Science,
Faculty of Mathematics and Computer Science, Nicolaus Copernicus University (NCU)
in Toruń, Poland
Master thesis: *Web based system for user management in Kardionet*
(in Polish: *Webowy system zarządzania użytkownikami systemu Kardionet*)
Supervisor: dr hab. Piotr Bała, prof. UMK

3 Employment in research institutes

Information on employment in research institutes or faculties/departments or school of arts

- since 02/2021 Assistant Professor at the Interdisciplinary Centre for Mathematical and
Computational Modelling, University of Warsaw
- since 10/2019 Assistant Professor at the Department of Computer Science, Faculty of
Mathematics and Computer Science, Nicolaus Copernicus University in Toruń
- 10/2018–09/2019 Assistant Professor at the Division of Parallel and Distributed Computing,
Faculty of Mathematics and Computer Science, Nicolaus Copernicus University
in Toruń

- 10/2014–09/2018 Research Assistant at the Division of Parallel and Distributed Computing, Faculty of Mathematics and Computer Science, Nicolaus Copernicus University in Toruń
- 05/2013–11/2013 Intern at IBM Zurich Research Laboratory, Switzerland (Mathematical & Computational Science Department)
- 02/2012–01/2013 Software Developer at the Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw

4 Description of the achievements

Description of the achievements, set out in art. 219 para 1 point 2 of the Act.

Title:

The PGAS Paradigm of Parallel Programming in Java: Development and Analysis of the PCJ Library on Selected Algorithms, including Bioinformatics

A cycle of thematically related scientific articles

- [1] Marek Nowicki, Davit Bzhalava, and Piotr Bała. Massively parallel implementation of sequence alignment with Basic Local Alignment Search Tool using parallel computing in Java library. *Journal of Computational Biology*, 25(8):871–881, 2018. doi:10.1089/cmb.2018.0079.
- [2] Marek Nowicki, Łukasz Górski, and Piotr Bała. PCJ - Java library for highly scalable HPC and Big Data processing. In *Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS 2018), Orléans, France, July 16-20, 2018*, pages 12–20. IEEE, 2018. doi:10.1109/HPCS.2018.00017.
- [3] Marek Nowicki. Comparison of sort algorithms in Hadoop and PCJ. *Journal of Big Data*, 7, 101(1):1–28, 2020. doi:10.1186/s40537-020-00376-9.
- [4] Marek Nowicki, Łukasz Górski, and Piotr Bała. Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version). *Concurrency and Computation-Practice & Experience*, 35(15):1–15, 2023. doi:10.1002/cpe.6536.
- [5] Marek Nowicki, Łukasz Górski, and Piotr Bała. PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads. *Journal of Big Data*, 8, 62(1):1–21, 2021. doi:10.1186/s40537-021-00454-6.
- [6] Marek Nowicki, Łukasz Górski, and Piotr Bała. Scalable computing in Java with PCJ Library. Improved collective operations. In *Proceedings of International Symposium on Grids & Clouds 2021 – PoS (ISGC2021), Taipei, March 22-26, 2021*, volume 378, 007, pages 1–19. Sissa Medialab Srl, 2021. doi:10.22323/1.378.0007.
- [7] Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, and Laila Sara Arroyo Mühr. HPV-KITE : sequence analysis software for rapid HPV genotype detection. *Briefings in Bioinformatics*, 26(2):1–11, 2025. doi:10.1093/bib/bbaf155.

Discussion of the Results

In today's world, processor clock frequencies have virtually stopped increasing due to physical limitations. To improve performance, modern devices are equipped with multiple cores, which makes parallel programming essential to fully utilize their computational potential. Moreover, as the complexity of computations grows, for example, due to larger data volumes or higher expected accuracy of results, a single machine often becomes insufficient to perform the required tasks. In such cases, multi-node systems are used, such as computing clusters or supercomputers. These systems require dedicated software support. Traditionally, such environments are based on programming languages such as C and Fortran, together with tools like OpenMP (*Open Multi-Processing*) for parallel computations on a single node and MPI (*Message Passing Interface*) for executing tasks across multiple nodes. Properly designed applications using these languages enable highly efficient scientific computations. However, the price to pay for such performance is more complex development and lower portability of applications.

The research I conduct is related to parallel programming using the PGAS (*Partitioned Global Address Space*) paradigm in a multi-node environment with the use of the Java language. I have been working on this topic since before obtaining my doctoral degree. My interest in this subject is related to my experience with the Java language (including the *Sun Certified Programmer for the Java Platform, SE 6* certificate obtained during my master's studies – certificate issue date: December 16, 2009), as well as the fact that Java has had built-in multithreading support at both the language and standard library level from the very beginning. From the first version of Java (in 1996), there existed the `java.lang.Thread` class representing a computational thread, the `java.lang.Runnable` interface used to define a task executed by a thread, and basic synchronization mechanisms such as the keywords `volatile`, `synchronized`, and the object methods `wait()`, `notify()`, and `notifyAll()`. Parallelism was further developed in subsequent versions of the Java language. Version 1.5 (also known as version 5) introduced a new *Java Memory Model* (JMM) and added the `java.util.concurrent` package with utility classes such as `Semaphore`, `CountDownLatch`, `ConcurrentHashMap`, `Executor`, and `Future`. Later versions introduced additional parallel features: Java 7 added the *Fork/Join Framework* (a computation model based on the divide-and-conquer approach with work-stealing mechanisms), Java 8 introduced *data streams* (*Java Streams*) and the possibility of their parallel processing using lambda expressions, as well as new utility classes such as `CompletableFuture`, and Java 21 finally introduced lightweight virtual threads, whose implementation continues to be optimized in subsequent releases. Moreover, starting from version 16, the *Project Panama* has been under development. One of its key components is the *Vector API* (JEP 529; *JDK Enhancement Proposal*) – a cross-platform interface that enables the use of processor vector computations. This solution allows defining numerical operations in a way that enables the compiler to automatically map them to the most efficient vector instructions available on a given execution architecture, which can lead to a significant performance increase compared to scalar computations. All these functionalities are based on single-node parallelism – on a single machine, more precisely, within the single JVM (*Java Virtual Machine*).

The choice of the Java language was motivated not only by my prior experience but also by its popularity and the large user base, particularly in the field of data analysis. Another important factor was its portability – the ability to run the same program on more advanced hardware without the need for tedious code adjustments to a specific compiler version or time-consuming recompilation. The computational performance can also be improved by using newer and more optimized versions of the JVM. Additionally, JVM mechanisms such as the garbage collector, strong typing, and the absence

of pointers and manual memory management enable the development of programs that are easier to maintain.

The subject of my research related to my doctoral dissertation was the development of new methods for parallel programming in the Java language based on the PGAS paradigm. As part of this work, I developed from scratch the PCJ (*Parallel Computing in Java*) library for Java, which enables parallel computations in multi-node and multi-core environments. The PCJ library is an open-source library for Java, available under the BSD license. It does not require any modifications to the Java language, any additional compilers, nor does it depend on any external libraries. This library allows running code on computing clusters and supercomputers, but due to the high portability of Java code and the language's motto *Write Once, Run Anywhere*, it is also possible to execute the same code on a workstation, desktop computer, or even a laptop. After obtaining my doctoral degree, I have continued to develop the library as part of my ongoing scientific research. The current version of the library (5.3.3) runs on any JVM supporting Java 8 and later versions, up to the newest releases.

The PCJ library enables research on the PGAS programming paradigm based on solutions implemented in the Java language. Subsequent versions of the library were used in research that contributed to the preparation of the following doctoral dissertations: *Adaptation of selected parallel algorithms to the PGAS model and their implementation in Java using the PCJ library* by M.Sc. Łukasz Górski, who was awarded the doctoral degree in 2017 by the Scientific Council of the Institute of Computer Science of the Polish Academy of Sciences (IPI PAN) in the field of technical sciences in the discipline of computer science, and *Development of selected parallel graph algorithms in the PGAS model and their implementation using the Java language* by M.Sc. Magdalena Ryczkowska, who was awarded the doctoral degree in 2018 by the same Council in the same field.

Until now, there has been no successful implementation of the PGAS programming model for the Java language, although solutions based on the PGAS paradigm are gaining popularity. Among them are libraries such as SHMEM [O1], Global Arrays [O2], and Charm++ [O3], and languages such as UPC [O4], Fortran [O5], Chapel [O6], and X10 [O7]. One approach to using Java within the PGAS model was the Titanium language [O8], which, however, did not use the JVM but instead translated Titanium code into C++, and then the generated code was compiled into an executable form. Another potential PGAS implementation for the Java language is the APGAS library, which offers programming solutions similar to those of X10 [O9].

In my habilitation application, I present a cycle of related scientific papers devoted to the topic of multi-node parallel computations using the Java language. This cycle documents and demonstrates significant scientific achievements in the field of engineering and technical sciences, in the discipline of technical computer science and telecommunications I have collectively titled this cycle *The PGAS Paradigm of Parallel Programming in Java: Development and Analysis of the PCJ Library on Selected Algorithms, including Bioinformatics*. The papers included in the cycle present the development and applications of the PCJ library as an implementation of the PGAS paradigm in the Java language and analyze parallel algorithms implemented with its use. Among the discussed algorithms is the TeraSort algorithm used for sorting large datasets, along with an analysis of its various implementation variants, as well as algorithms applied to solving practical bioinformatics problems, including virus detection in DNA/RNA sequences. These studies contributed to the development of the research area of parallel programming within the discipline of technical computer science, with particular emphasis on applications in the natural sciences.

4.1 Discussion of individual works

My doctoral dissertation concerned the PCJ library, version 3. During my PhD studies, two earlier versions were developed. A detailed history of this development, up to the preliminary version *5-alpha*, can be found in [A1] (in Polish).

PCJ 1

The first, prototype version of the library was created in November 2011. The library was developed based on the latest version of Java 7 available at that time. Initially, it included only the basic communication mechanisms, such as data retrieval (`get`), data sending (`put`), and thread execution synchronization (`sync`). At a later stage, I added a message broadcasting mechanism (`broadcast`) and the ability to create thread subgroups. The `put`, `get`, and `broadcast` operations were one-sided and asynchronous. The `put` and `broadcast` operations were also non-blocking, which led me to introduce a non-blocking variant of the `get` method, allowing a thread to continue its execution while retrieving data. I also implemented a mechanism to check whether a message that could modify a local shared variable had occurred (`monitor`, `waitFor`).

This version assumed the execution of only a single PCJ thread on a single JVM.

PCJ 2

The second version, whose development began in August 2012, modified the core concept by allowing multiple PCJ threads to run on a single node. This change broke backward compatibility, as multiple PCJ threads could now operate on a single JVM, which involved sharing the static fields of classes – something that had previously been impossible. I solved the problem of sharing static class fields by using custom class loaders for individual PCJ threads.

Additionally, the non-blocking variant of the `get` method was replaced with `getFutureObject`, which automatically creates and returns a `Future` object. In the `get`, `getFutureObject`, and `put` methods, I added an optional parameter allowing the use of indices of a shared array variable. I also introduced the `getLocal` and `putLocal` methods, which enabled local access to references of shared objects, rather than their full copies.

PCJ 3

The third version of the library introduced a standardized naming of methods (e.g., `barrier` instead of `sync`, `myId` instead of `myNode`, and `threadCount` instead of `numNodes`). This version was also used to implement applications that are part of the *HPC Challenge Benchmark Suite*. PCJ version 3 received the *HPC Challenge Class 2 Award: Best Productivity – Most Elegant Solution* at the international conference *SC14 – The International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing 2014)*. The award was granted based on expert evaluations of the implementation of a subset of the *HPC Challenge* benchmark using the PCJ library, recognizing the presentation of an efficient and elegant approach to programming parallel applications.

4.1.1 PCJ-BLAST [1] – PCJ 4.1.0

The PCJ library version 4.1.0 served as the basis for the solution described in the first paper of the cycle [1], which is an extended version of the paper [A2]. In this paper, we described a solution called *PCJ-BLAST*. It is an application for biological sequence alignment implemented in Java using the PCJ library. The application enables the execution of multiple unmodified *NCBI-BLAST* instances in parallel on multi-node systems, with automatic processing of the obtained results. The parallelism of this solution is based on distributing input sequences among the utilized nodes. The input sequences are then searched within a reference database.

PCJ 4

The fourth version of the library was developed during the HPDCJ project (Heterogeneous Parallel and Distributed Computing with Java), in which I served as the *technical coordinator of the work package*. The development of version 4.0 began with organizational matters, such as changing package names from *pl.umk.mat.pcj.** to *org.pcj.**. In this version, I also added a method enabling compare-and-set (CAS) operations. I introduced modifications allowing the use of the SDP (*Socket Direct Protocol*), which supports streaming connections in the *InfiniBand* infrastructure. Within the project, partners worked on performing GPU computations using Java and the PCJ library. Due to the JVM limitation of loading a native library only once, in subsequent versions of the library, I worked on developing a mechanism that would allow native libraries to be loaded and shared among different PCJ threads.

One of the solutions was based on annotating a class containing references to native methods to inform the class loading mechanism that the system class loader should be used. However, this approach did not work well for external libraries, as it would require modifying existing code. Alternatively, I considered specifying, in the JVM startup parameters, which packages or classes should be loaded by the system class loader. This solution proved to be more practical, although prone to errors, since it required the user to know all package names used by a given library. Additionally, in cases where objects whose classes were not loaded by the system class loader were passed to the library, runtime errors could occur.

Finally, in version 4.1, I abandoned the use of custom class loaders for individual PCJ threads. An additional argument for this change was the impossibility of completely separate the class space – classes from the standard library had to be loaded by the system class loader, shared among all PCJ threads running on a single JVM. As a result, certain classes and their static fields were shared anyway. The most important consequence of removing custom class loaders, however, is that the static fields of all classes are now shared within a single JVM, not just system classes. Furthermore, in version 4.1 of the library, I introduced Java 8 language constructs and features for the first time, which meant that applications developed with the library had to be run in a Java 8 environment.

The input data were obtained using a *Next-Generation Sequencing* device (*Illumina*) and consist of one million short sequences (up to 150 characters) in the FASTA format. PCJ-BLAST uses the original, unmodified input data. Processing each of the input sequences may take from a few seconds to several minutes (Figure 1) – the processing time cannot be easily estimated based on the structure of the sequence itself. For this reason, the algorithm used is based on the dynamic distribution of input data among the nodes used for computation. The decision on which node receives the next portion of sequences to process is made based on the current workload.

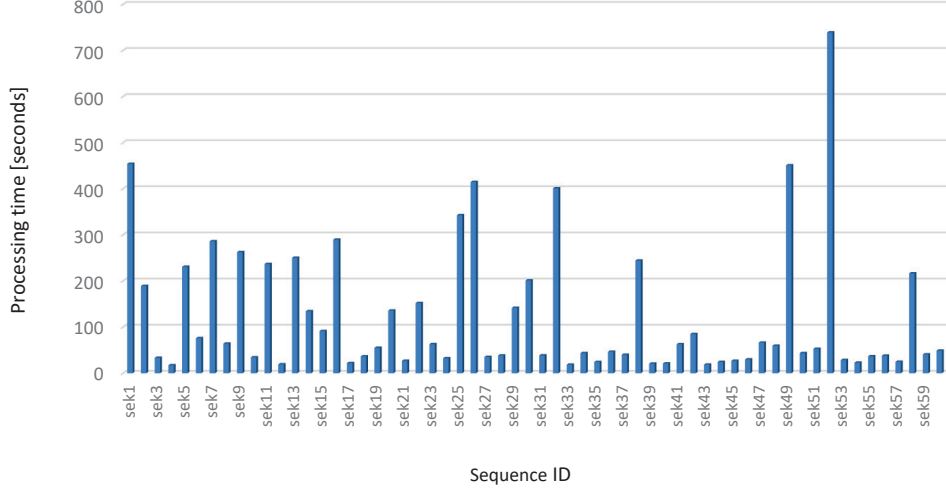


Figure 1: Processing time for the selected reads from the input file. (from [A2])

Parallel sequence processing uses multiple NCBI-BLAST instances per node. The NCBI-BLAST tool itself provides the capability to process input data in parallel using threads, which had to be taken into account when assigning the number of threads to be used for computations. PCJ thread #0 (*Scatterer*) reads the input FASTA file and distributes the sequences to the remaining threads. The other PCJ threads (*Workers*) execute NCBI-BLAST on the received sequences and process the obtained results into the appropriate format. The *Scatterer* sends batches of sequences to available *Worker* processes, and if none are available, it waits until they become free. After completing the computation, a *Worker* process converts the result (in XML format) into a form suitable for further data processing and writes it to its own output file (e.g., as a CSV file). All communication between PCJ threads is performed asynchronously. Figure 2 presents the algorithm workflow.

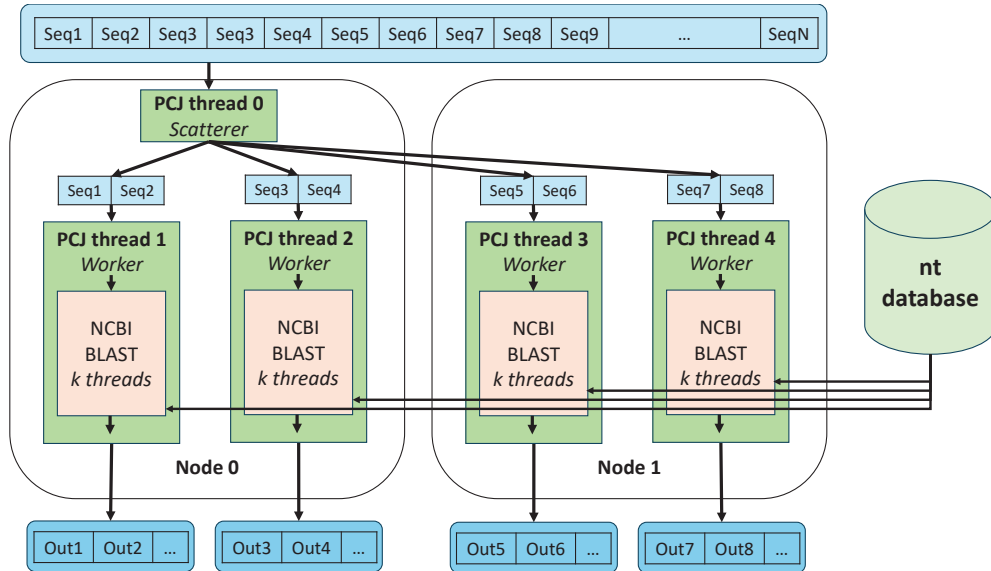


Figure 2: Schematic view of PCJ-BLAST execution. (from [1])

In addition to presenting the algorithm, the paper also discusses performance results. I examined for which number of NCBI-BLAST threads and for which number of NCBI-BLAST processes per node the execution time is the shortest. It was found that on the *Topola* HPC cluster (ICM UW; each node

equipped with a 28-core Intel Xeon E5-2697 v3 @ 2.60 GHz processor and at least 64 GB of RAM), running four NCBI-BLAST processes, each using seven threads, provides the highest performance.

Performance tests conducted on 64 nodes (1792 cores) demonstrated that the parallel efficiency exceeds 70%: the analysis time for 1 066 288 sequences was reduced from the expected 52 days when using a single 28-core node to less than 27 hours when using 64 nodes.

In the paper, I also compared the performance of the developed solution with an approach based on dividing the reference database, tested on the *Okeanos* supercomputer (ICM UW; Cray XC40). The compute nodes of this system are equipped with two 12-core Intel Xeon E5-2690 v3 @ 2.60 GHz processors (with optional *hyper-threading*, providing 24 logical cores and a total of 48 computational threads) and 128 GB of RAM.

Number of nodes	Number of cores	NCBI-BLAST		PCJ-BLAST	
		db splitted to four blocks		4 NCBI-BLAST per node	
		Execution time in minutes	Parallel efficiency	Execution time in minutes	Parallel efficiency
1	48	1470.5	1.000	756.0	1.000
2	96	736.4	0.998	379.0	0.997
4	192	378.8	0.970	190.8	0.990
8	384	216.1	0.851	96.3	0.981
16	768	113.7	0.808	49.5	0.955
32	1536	60.3	0.762	26.4	0.893
64	3072	32.2	0.714	14.1	0.837
128	6144	21.2	0.542	7.8	0.755

Table 1: Execution time and parallel efficiency comparison between PCJ-BLAST and the reference database partitioning approach.

Initially, the reference database was divided into 32 approximately equal blocks, but the processing time in this configuration was long – with 8 nodes, it took about 315 minutes to process the first 12 288 sequences. When the database was divided into 4 parts, the performance improved, and the processing time for the same data on 8 nodes decreased to 216 minutes. The solution based on database partitioning demonstrated very good scalability (doubling the number of nodes nearly halved the execution time), but the total processing time was roughly twice as long as when using PCJ-BLAST, where processing the same data on 8 nodes took 97 minutes. This was mainly due to the fact that PCJ-BLAST implements a dynamic load-balancing mechanism. Detailed results are presented in Table 1.

As part of the research underlying this work, in addition to the development of the PCJ library itself, I designed and implemented a mechanism for preparing and parallel processing of input data, as well as for storing output data. I also implemented a mechanism for balancing the load among individual system components. In addition, I conducted performance tests of my solution and other approaches compared in this work. I was responsible for preparing the description of the proposed solution and the conducted experiments, as well as for presenting the performance results and discussing the achieved outcomes.

The development of version 5.0 of the PCJ library began in June 2016 as part of the HPDCJ project. When creating the fifth version of the library, I decided to completely remove the elements related to custom class loaders, which in version 4 had only been hidden and remained inactive. However, this was not merely an organizational release.

Among the main changes, although invisible to the user, was a complete redesign of the inter-node communication mechanism. This modification was almost revolutionary in nature. It enabled the execution of applications using the PCJ library on more than 1024 compute nodes. Additionally, the change in the internal architecture allowed messages larger than 2 GB to be transmitted, which would have caused errors in previous versions. The new message-passing structure also facilitated the introduction of new message types and enabled a straightforward implementation of mechanisms for notifying the completion of communication operations. As a result, version 5 also brought a number of changes to the *Application Programming Interface* (API).

The most important change was the systematization of the library's core methods: `get`, `put`, `broadcast`, and `barrier`. Previously, these methods had consistent names but differed in behavior: `get` and `barrier` were synchronous, while `put` and `broadcast` were asynchronous, allowing the application to continue without waiting for message acknowledgment. In the new version, the behavior of all these methods was unified: they now operate synchronously. Additionally, an asynchronous counterpart was introduced for each method: `asyncGet`, `asyncPut`, `asyncBroadcast`, and `asyncBarrier`. The asynchronous versions became the primary ones, while the synchronous methods, available only in the main class `org.pcj.PCJ`, serve as *wrappers* that invoke the corresponding asynchronous versions. For example, calling `PCJ.barrier()` is implemented by executing the asynchronous method and waiting in the blocking method on the returned `Future<Void>` object: `PCJ.asyncBarrier().get()`. The use of `Future` objects also allows for waiting with a timeout for the completion of an operation.

I also introduced changes to the parameters passed to the library methods to make them as intuitive and consistent as possible. Starting from version 5, the variable name appears directly before the optional array index parameters, and the thread identifier to which the variable belongs is specified before them. Since the optional parameter must appear at the end of the argument list, the value passed in the `put` and `broadcast` methods is now the first argument, e.g., `PCJ.put(newValue, threadId, variableName[, indices...])`.

Another significant change concerned the way shared variable names are defined. In previous versions of the library, these names were represented as strings, which carried the risk of typographical errors, lacked support for refactoring, and limited the use of multiple classes serving as *repositories* of shared variables. In the new version, I decided to replace string names with enumeration constants, which greatly facilitated code maintenance and refactoring, while also reducing the risk of errors. Additionally, the annotation processor implemented in the library allows potential inconsistencies to be detected at compile time, rather than during program execution.

In PCJ 5.0, I also introduced support for exceptions that may occur during the transmission of shared variables – in the event of an error in the target thread, the information is communicated back to the source thread via a `Future` object, which stores the state of the asynchronous operation.

Furthermore, the *compare-and-swap* (CAS) operation introduced in PCJ 4 was removed due to its low performance in situations of high thread contention, which resulted from the need to transmit many small messages through network when applying the classic CAS loop pattern. During the development of the new version, I considered moving the computation of the new variable value to the target thread instead of the source thread, which could reduce the number of transmitted messages. Ultimately, rather than implementing this solution solely for the CAS operation, I decided to introduce a more general mechanism. In version 5.0.5 of the PCJ library, I added the `asyncAt` operation, allowing arbitrary tasks to be scheduled for execution in the context of a remote PCJ thread.

Earlier versions (from 5.0.0 to 5.0.4) were development stages in which I focused primarily on bug fixes and performance optimizations of the library.

4.1.2 Tutorial [2] – PCJ 5.0.6

In parallel, alongside the development of applications using the PCJ library, we have also undertaken efforts to promote it and present it to a broader audience.

The PCJ library is used in the teaching process, among others, in the *Computational Engineering* program offered by the Interdisciplinary Centre for Mathematical and Computational Modelling at the University of Warsaw, as part of the *Parallel Programming* course. It is also taught in computer science programs at the Faculty of Mathematics and Computer Science of the Nicolaus Copernicus University in Toruń, as part of the *Introduction to Parallel Computing* course, which is a compulsory course in the second-cycle *Computer Science* program, specializing in *Large-Scale Computing*. In addition – before recent curriculum and program changes – it was also included in the course *Parallel and Distributed Programming*, offered as an elective for first-cycle students of *Computer Science* and as a compulsory course for second-cycle *Computer Science* students.

Another form of promoting the library is through the organization of tutorials that present its capabilities and enable participants to quickly acquire the knowledge and skills necessary for the practical use of PCJ. The second paper discussed in the cycle [2] is a publication resulting from a tutorial conducted during the *High Performance Computing & Simulation* (HPCS) conference in 2018. We were invited by the conference organizers to prepare and deliver this tutorial, as well as to write the accompanying publication. Previously, I co-developed the concept and participated in conducting workshops at the *Parallel Processing & Applied Mathematics* (PPAM'15, PPAM'17) conferences and at *Supercomputing Frontiers Europe 2018*. In the discussed paper, I was responsible for preparing nearly the entire description of the PCJ library, including illustrations and most of the example source codes.

PCJ 5.0.6

Version 5.0.6 of the library introduced, among other things, performance optimizations through the use of a pool of `ByteBuffer` objects for handling network messages. It was also the first version of the library made available not only through the project's homepage but also in the main package repository for the Java language – the *Maven Central Repository*. The library appeared in this repository in October 2018.

The paper discusses the fundamentals and operation of version 5.0.6 of the PCJ library. However, it is not a direct transcript of the tutorial, nor does it include the complete source code of the

applications developed by the participants. Instead, it presents in a condensed form the topics covered during the training session.

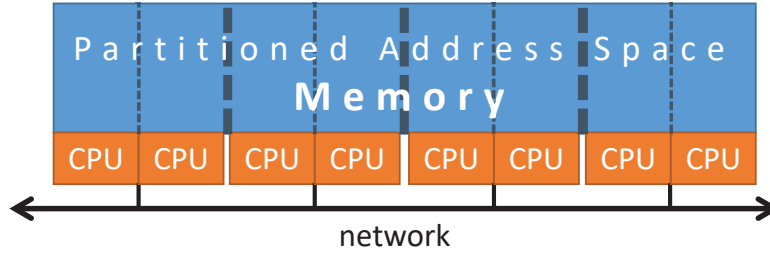


Figure 3: View of the memory in the PGAS model. Each computing node, possibly consisting of many CPUs, has its own memory. The computing nodes are connected through the network. All of the nodes memories' address space conceptually is treated as a single global memory address space. (from [3])

The work begins with a short introduction to the PGAS (Partitioned Global Address Space) paradigm, where the view of global memory is realized by dividing it among the threads (units) performing computations (Figure 3). The memory is divided into a private part and a shared part. The private memory is accessible only to its owner and contains its local variables. The shared memory, on the other hand, can be accessed by other threads. Depending on the programming language, we do not have direct access to the region of memory that is private or shared, but only to variable names. In such cases, all variables are private by default, but some of them can be declared as shared. Shared variables are accessible to all threads – their state can be read and modified remotely.

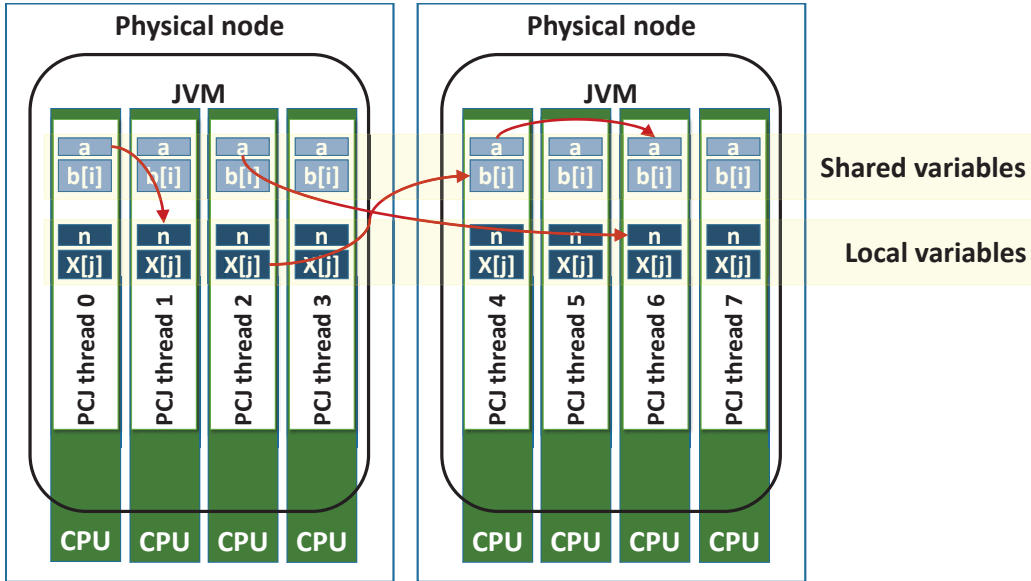


Figure 4: Diagram of PCJ computing model. Arrows present possible communication using `put()` or `get()` methods acting on shared variables. (from [2])

In the paper, we presented an introduction to the programming interface (API) provided by the PCJ library. The PCJ library is based on the SPMD (*Single Program, Multiple Data*) computation model, in which the same program is executed simultaneously on n compute nodes. On each node, PCJ threads are launched, and each thread operates on its own set of data. The number of computation threads on individual nodes is fixed and determined at the beginning of the program execution.

The PCJ library provides the essential mechanisms such as thread numbering and synchronization, as well as data transfer through simple commands – single library methods. The shared memory is implemented through the fields of a class called the *storage*. By default, all fields are

local, but some of the class fields can be marked as shared (also referred as *shareable*). Figure 4 shows the schematic representation of the computational model in PCJ and the communication using the `put()` and `get()` methods. The communication details are hidden from users, and the communication itself is one-sided and asynchronous.

The work also describes the internal communication mechanisms. One of the important mechanisms is related to data transfer between PCJ threads. Data transfer within a single node differs from data transfer between nodes. In the case of transferring data within one node, it is not possible to simply pass the value of a field. Such a transfer would result in sharing a reference to the same object. Therefore, a deep copy of the object must be created. In the PCJ library, this is done by serializing the source object and deserializing it in the memory of the target thread. This operation is time-consuming but safe, especially in the case of complex objects with many references to other objects. The diagram of this process is shown in Figure 5a. In contrast, when communication occurs between nodes, data transfer is handled using network sockets and a thread implementing the *Selector*, whose task is to send and receive data through communication channels. In this case, object serialization is natural. The diagram of this process is shown in Figure 5b.

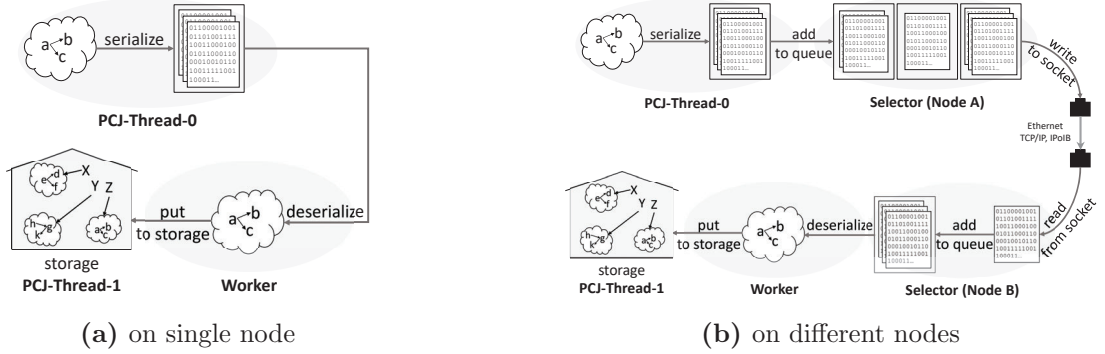


Figure 5: Execution diagram of `PCJ.put()` method between two PCJ threads. (from [2])

The paper also presented basic constructs such as loop parallelization, *stencil* communication (boundary data exchange, also known as *halo exchange*), and reduction. For these constructs, we also demonstrated and discussed performance results of example problem implementations that use them. Loop parallelization was illustrated with the example of approximating the value of π using the *Monte Carlo* method. The *stencil* communication was demonstrated through the implementation of the Savitzky-Golay curve smoothing algorithm [O10]. The reduction construct was presented using the *WordCount* problem, whose goal is to count the number of occurrences of individual words in a text.

Thanks to the presented constructs, it is possible to easily implement more advanced algorithms that solve complex problems.

4.1.3 Cloud [4] – PCJ 5.0.8

So far, we have demonstrated the operation of applications developed using the PCJ library implementing the PGAS paradigm on computing clusters and a supercomputer. However, due to the growing popularity of cloud platforms, we decided to verify the possibility of running applications using PCJ in such an infrastructure. Our research in this area was presented in the paper [4], which is an extended version of the paper [A3] presented at the *Second International Workshop on Parallel Programming Models in High-Performance Cloud (ParaMo 2020)*, organized as part of the *26th International European Conference on Parallel and Distributed Computing (Euro-Par 2020)*

in Warsaw (because of COVID-19, presentations were delivered remotely). In the process, we also verified the correctness and performance of applications running on various ARM processors. The research work was conducted using the PCJ library version 5.0.8.

PCJ 5.0.8

In PCJ version 5.0.8, besides fixes related to network mechanisms, an important change was introduced in the behavior of the annotation processor, which, starting from this version, issue a warning instead of an error for shared fields that are non-serializable and not marked as `final`.

As part of our research, we utilized cloud resources provided by Amazon (*Amazon Web Services (AWS) Elastic Compute Cloud (EC2)*) and Linaro (*Linaro Developer Cloud*).

Within the AWS cloud, we used *t2.xlarge* instances with Intel Xeon CPU E5-2686 v4 @ 2.3 GHz, providing 4 vCPUs and 16 GB of RAM. The ARM-based instances were equipped with AWS Graviton processors @ 2.3 GHz, featuring 64-bit ARM Neoverse cores designed and optimized by AWS. We used *a1.xlarge* instances with 4 vCPUs and 8 GB of RAM, as well as *a1.4xlarge* and *a1.metal*, which had 16 vCPUs (or 16 CPUs in the *metal* version, where physical cores were used directly) and 32 GB of RAM. Network bandwidth was rated as *moderate*, corresponding to inter-instance network capacity of up to 1 Gb/s. We used the OpenJDK Runtime Environment Corretto-11.0.6.10.1 as our Java virtual machine.

Within the Linaro cloud, we used only ARM processors. Most instances had 64-bit Cavium ThunderX 88XX processors @ 1.9 GHz, and the differences between them were in the number of vCPUs and RAM: *m1.tiny* (1 vCPU, 512MB), *uk.large* (8 vCPUs, 16GB), *uk.medium* (4 vCPUs, 8GB), *uk.small* (2 vCPUs, 4GB), *uk.nano* (1 vCPU, 2GB), *nano* (1 vCPU, 4GB). The *m1.small* instance was equipped with a 64-bit Qualcomm Falkor processor @ 2.2 GHz with 2 vCPUs and 1 GB of RAM. Documentation indicated that the network bandwidth between instances in the cloud was up to 10 Gb/s. The Java virtual machine used was the OpenJDK Runtime Environment (*build 11.0.6+10-post-Debian-1deb10u1*).

For comparison purposes, we also used the *Okeanos* supercomputer (ICM UW). On this supercomputer, I ran applications both directly on physical cores and through the *Singularity* virtualization platform. The supercomputer uses the high-performance Cray Aries communication network developed by Cray. The Java virtual machine used in this environment was the OpenJDK Runtime Environment (*build 11.0.4+11-suse-3.33.1-x8664*). Table 2 provides a summary of all the types of instances used.

Configuration Using cloud resources for computation differs fundamentally from using computing clusters or supercomputers. First of all, operating system images are usually stripped of many packages – only the essential packages are installed, and the others must be installed manually. Additionally, normally launched instances participating in a joint computation do not know each other’s *hostnames*, and their file systems are not shared. Therefore, it is necessary to configure the operating system and the entire environment to cooperate properly when running applications.

To establish a connection from a local computer to a cloud instance, a key pair must be prepared: a public key and a private key. The private key is stored locally and is used for authentication when connecting to the cloud instance, where the public key is placed during instance creation. Although it is possible to connect from the local computer to all created instances, the instances themselves do not have the credentials needed to connect to each other.

Provider	Type	Processor	CPU	vCPU	RAM
Amazon	t2.xlarge	Intel Xeon CPU E5-2686 v4 @ 2.3 GHz	-	4	16 GB
Amazon	a1.xlarge	AWS Graviton @ 2.3 GHz	-	4	8 GB
Amazon	a1.4xlarge	"	-	16	32 GB
Amazon	a1.metal	"	16	-	32 GB
Linaro	m1.small	Qualcomm Falkor @ 2.2 GHz	-	2	1 GB
Linaro	m1.tiny	Cavium ThunderX @ 1.9 GHz	-	1	512 MB
Linaro	uk.large	"	-	8	16 GB
Linaro	uk.medium	"	-	4	8 GB
Linaro	uk.small	"	-	2	4 GB
Linaro	uk.nano	"	-	1	2 GB
Linaro	nano	"	-	1	4 GB
ICM UW	bare metal	2×Intel Xeon E5-2690 v3 @ 2.60 GHz	48	-	128 GB
ICM UW	singularity	"	-	48	128 GB

Table 2: Summary of instance types (computing nodes) used.

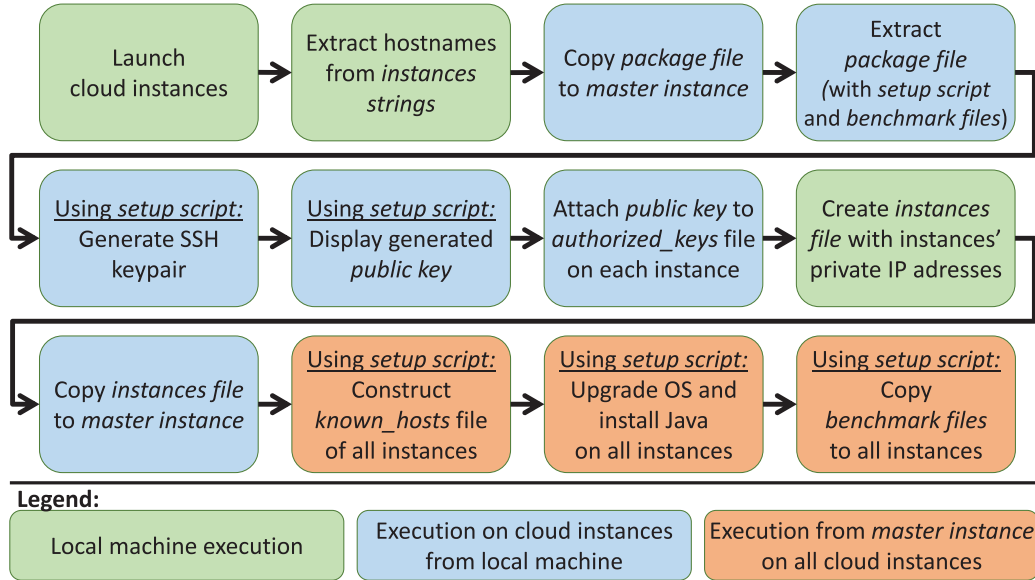


Figure 6: Cloud environment configuration diagram. (from [4])

Since the earliest versions of the library, there have been two ways to start computations: `start()` and `deploy()` mechanisms. The first method (`start()`) requires a system mechanism for simultaneously launching applications on compute nodes (instances). If such a mechanism is not available, the library can use the second method (`deploy()`), which independently launches new instances from the so-called *master instance*. In this case, the configuration should allow non-interactive SSH login using keys, without prompting for passphrases or verifying the fingerprint of each node. Copying a locally created private key to a cloud instance is not a secure solution – in a *man-in-the-middle* attack, the private key could be exposed. Therefore, the key pair should be generated directly on the master instance. Moreover, the master instance must know the IP addresses of all other instances used in the computation. Figure 6 shows a diagram of the cloud environment configuration, with a distinction between actions performed locally and on cloud instances. To facilitate environment setup, I created BASH shell scripts to be executed locally and after connecting to the master instance (the content of the scripts is presented in *Appendix A* of the paper [4] as Listing 1 and Listing 2, respectively). By

using these scripts, the environment configuration on AWS EC2 with 32 instances takes less than 2.5 minutes.

It is worth emphasizing that such a simple configuration was sufficient to successfully run applications based on the PCJ library and utilize multiple cloud machine instances for a single computation task. The studies presented in this work showed that the PCJ library allows parallelization of computations not only on computing clusters or supercomputers but also using cloud resources. Furthermore, we demonstrated that as long as a standard Java virtual machine (JVM) is available on a given machine, PCJ applications can be run, and if these machines are networked, multi-node operation is possible. The ability to run is not limited to x86 processors or a specific JVM implementation.

For performance comparison, I presented three types of applications: *DES Decryption* – a computation-intensive application, *FFT* – a communication-intensive application, and *WordCount* – a Big Data application. This selection allowed for the analysis of different aspects of application performance. To standardize executions across instance types, the maximum memory size for the JVM was set to 1 GB, regardless of the actual available memory. Moreover, for each test type, I selected the most efficient results – those with the shortest execution time, and therefore the highest performance.

DES Decryption The first test is *DES Decryption*. This is a computation-intensive, trivially parallel problem. In the paper, I evaluated three parallel implementations: a single-node version based on native Java threads, a multi-node version using the PCJ library, and an optimized multi-node implementation. All implementations were based on the simplest brute-force approach, in which all possible combinations of the decryption key were tested. For a key of n bits, there are 2^n possible key combinations. Normally, the search would stop once the key is found, but in the presented performance tests, the full range of possible keys was examined. Classes and interfaces from the Java standard library were used for encoding and decoding (including `javax.crypto.Cipher` and `java.security.Key`). In my optimized implementation, to improve computational performance, each PCJ thread uses a single key object that is modified in each iteration, instead of creating a separate object for every tested key.

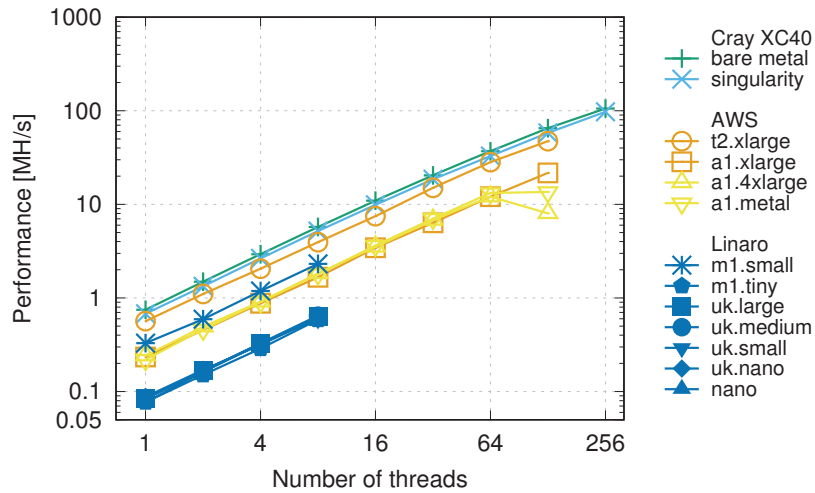


Figure 7: Performance results for the *DES Decryption* problem with a 26-bit key. A higher value indicates a better result. (from [4])

The performance results of the optimized implementation are presented in Figure 7. The values are given in *mega-hashes/s* (the number of keys tested per second, H/s). Regardless of the machine type, the scalability of the solutions is very good. However, the overall performance of most Linaro

cloud instances is nearly ten times lower compared to the solution running on the Cray XC40, and about three times lower than the results obtained on the AWS cloud. This difference reflects the processor speed in each system. Only the *m1.small* instance with the Qualcomm Falkor processor shows performance similar to AWS instances. The performance achieved on AWS is lower than on the Cray XC40, and the difference between *a1* and *t2* instances illustrates the performance gap between x86 and ARM processors. It is also worth noting that the performance on the Cray XC40, whether using virtualization or not, is practically the same. This shows that virtualization does not introduce a significant overhead for CPU-intensive applications.

FFT The Fast Fourier Transform (*FFT*) was chosen as a communication-intensive test. The PCJ version was based on a PGAS implementation developed for Coarray Fortran 2.0 [O11] and created as part of the doctoral dissertation [O12] by the co-author of the study, Łukasz Górski. It uses a radix-2 binary exchange algorithm designed to minimize interprocess communication. The operation scheme is as follows: first, a local *FFT* algorithm based on bit-reversed permutation is executed; then, a global transposition of data from block to cyclic distribution is performed, followed by another local *FFT* computation, and finally a global inverse transposition restores the data to the original block distribution. Communication between all threads (*all-to-all*) takes place in the procedure responsible for the transposition and inverse transposition of data.

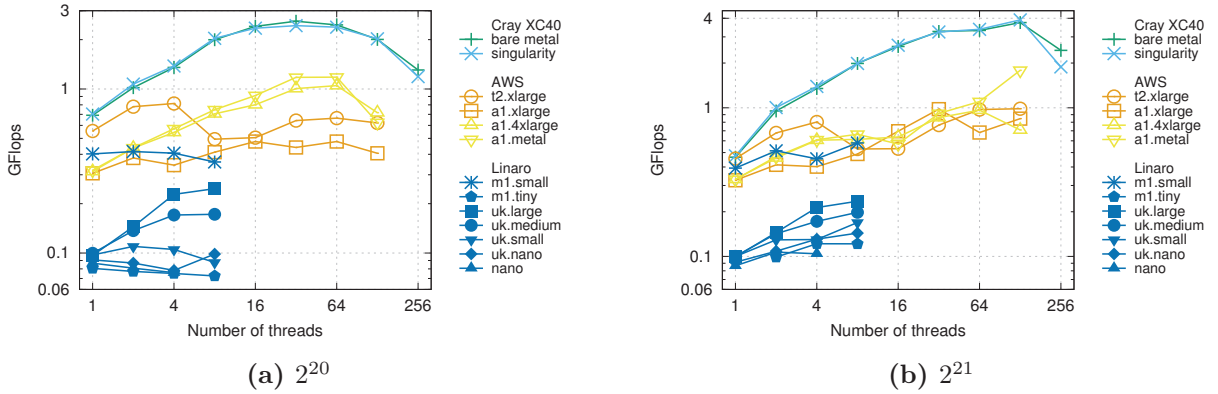


Figure 8: Performance results for the *FFT* problem. Values are given in GFlops. Higher values indicate better results. (from [4])

The plots presented in Figure 8 show the performance results. The values are given in *giga-flops* (the number of floating-point operations per second). Due to the nature of the algorithm, where the number of threads must be a power of two, some of the compute cores on the Cray XC40 nodes remained partially unused.

As in the previous performance test, the solution running on the Cray XC40 achieved the highest performance, and virtualization did not introduce a significant computational overhead. The scalability of this solution is limited and improves with larger data sizes, which we also demonstrated in our other studies: in [A4], we performed *FFT* on data of size 2^{26} , and in [5], we presented results for data of size 2^{30} .

The solution running on the AWS cloud shows good scalability, but only up to a few threads, and exhibits lower overall performance. The results obtained for 16 threads are similar for both the ARM (Graviton) and x86 processors. For a smaller number of threads (4 per instance), performance is higher on the x86 architecture. However, as the number of instances increases, performance becomes comparable, especially for larger datasets. It is also worth noting that for a larger array (Figure 8b)

and a small number of threads, the performance achieved using the Cray XC40 and AWS cloud (especially for the *t2.large* instance) is similar – this is because the computations were performed on a single instance and did not involve network communication. This reflects the nature of the *FFT* application, whose performance is primarily determined by communication; therefore, performance is limited by communication channel bandwidth rather than processor speed. The Cray XC40 is equipped with a more efficient inter-node communication infrastructure, which results in better performance for communication-intensive applications.

The results obtained on the Linaro cloud show lower overall performance. The only instance that matched the performance of AWS cloud instances was *m1.small* with the Qualcomm Falkor processor. Moreover, the Linaro cloud results indicate that performance is better for larger arrays (2^{21}), because in this cloud environment, computation rather than communication is the limiting factor. On the other hand, communication overhead limits performance for smaller arrays.

WordCount The last presented test is the *WordCount* application. This is a flagship application demonstrating the operation of the *map-reduce* model. In this problem, the number of occurrences of each word in the input data is counted. The application operates in two phases. In the first phase, the input file is read line by line, and each line is split into individual words, creating a list of pairs (*word*, 1). In the second phase, the partial results are summed based on the key, which is the first element of the pair (*word*).

The implementation of this problem using the PCJ library was prepared by the co-author of this work, Łukasz Górski, for our earlier joint article [A5]. In the presented solution, each line of the input file is split into words using whitespace as delimiter, and a partial count of occurrences is stored in a `Map<String, Integer>` dictionary object. Each thread maintains its own independent copy of this object. For simplicity, the implementation does not perform any preprocessing of the words. Finally, a reduction operation is performed, and the final result is stored in thread #0. The processing phases do not overlap – reduction occurs only after all threads have finished reading the input data.

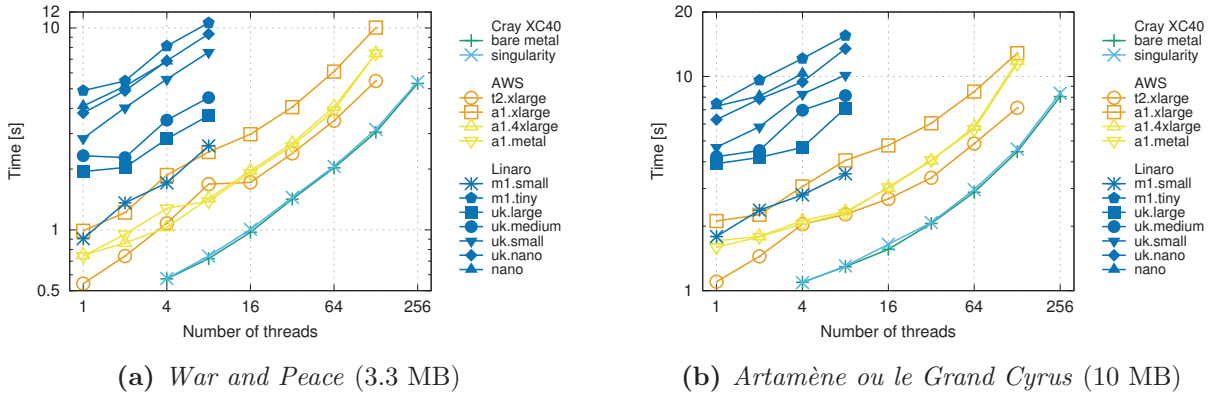


Figure 9: Performance results for the *WordCount* problem. Values are in seconds. Lower values indicate better results. (from [4])

In the performance tests, I used a simple reduction scheme, where thread #0 sequentially retrieves the partial counts from each thread and merges them. Performance can be improved by using a different reduction algorithm, as presented in our other works [A5, 5, 6]. The performance evaluation was conducted using the weak scaling regime, meaning that the problem size was increased proportionally to the number of threads, maintaining a constant workload per thread. In other words, each thread processes the entire input file, so increasing the number of threads increases the total amount of data to be processed.

The analysis was performed on two files containing novels, following the methodology adopted in our previous work [A5]. The first text file, encoded in UTF-8, contained the English translation of Leo Tolstoy’s *War and Peace* [O13] and its size was 3.3 MB. The second text file, encoded in ISO 8859-1, contained the original French version of Georges de Scudéry’s *Artamène ou le Grand Cyrus* [O14] and its size was 10 MB.

The performance results presented in Figure 9 show similar scalability for both Intel and ARM processors. As before, the solution running on the Cray XC40 supercomputer executed twice as fast as on the AWS cloud. Execution on the Linaro cloud took between 4 and 10 times longer than on the Cray XC40 and at least twice as long as on the AWS cloud. The only exception was the *m1.small* instance on the Linaro cloud, whose performance was similar to that of the *a1.xlarge* instance on AWS.

As part of the research that is the basis of this work, I designed and implemented a mechanism for running parallel tasks using the PCJ library on both cloud platforms. Additionally, I was responsible for contacting Amazon and Linaro in order to obtain computing resources for scientific purposes in amounts sufficient to conduct meaningful experiments. I also managed the cloud machines and performed computations using these resources. I described the mechanism for preparing the cloud environment to execute computations with the PCJ library. I conducted all the performance tests of parallel applications using the PCJ library presented in this publication on both clouds, as well as on the Okeanos supercomputer, within the computing grant that I led. I was responsible for preparing the description of the experiment configuration, as well as the performance results and discussion of the achieved outcomes.

4.1.4 HPC, Big Data & AI [5] – PCJ 5.0.8

Artificial Intelligence (AI) is becoming more and more popular. It enables solving problems for which creating a traditional algorithm is either impossible or exceptionally complex. In recent years, approaches based on *deep learning* have gained particular significance. Deep learning involves analyzing very large training datasets and adjusting the internal parameters of a model so that it can independently extract patterns and relationships. The larger the volume and diversity of training data, the better the neural network *learns* to recognize them, and consequently, to provide more accurate responses to previously unseen data. However, processing huge datasets and training neural networks with a large number of parameters requires significant computational resources, which are not available on typical laptops or workstations. For this reason, it is important to develop tools that enable the integration of Big Data processing with workloads characteristic of Artificial Intelligence in *High-Performance Computing* (HPC) environments.

Another paper in the cycle [5] demonstrates that solutions based on the PCJ library can be used to solve tasks in various areas, such as HPC, Big Data, and AI. This opens the possibility of integrating HPC solutions with Big Data and Artificial Intelligence applications. Moreover, thanks to the portability of Java-based solutions, a user can develop software on their laptop or workstation and later, even without recompilation, run the solution using a cloud platform or an HPC system.

In the discussed paper, we presented problems from the areas of HPC, Big Data, and AI. The HPC problems were the *2D stencil* and the previously discussed *FFT*; the Big Data problem was *WordCount* and was also previously discussed; the AI tasks included modeling the connectome of the *C. elegans* nematode using a differential evolution algorithm and distributed training of a neural network.

The performance results were collected on the Cray XC40 supercomputers: *Okeanos* (ICM UW) and *Hazel Hen* (HLRS, University of Stuttgart, Germany). The compute nodes of the supercomputers were equipped with two Intel Xeon E5-2690 v3 processors (*Okeanos*) or Intel Haswell E5-2680 processors (*Hazel Hen*), each with 12 cores, and also supported hyper-threading, providing two threads per core, for a total of 48 compute threads per node. Java-based solutions were tested using Oracle JDK 1.8.0.51 for PCJ and Oracle JDK 10.0.2 for APGAS. For C/MPI-based solutions, we used the Cray MPICH implementations, versions 8.3 (*Okeanos*) and 8.4 (*Hazel Hen*). Additionally, we used OpenMPI version 4.0.0, which provides Java bindings for MPI, for Java/MPI executions.

As part of the work for this paper, I was in particular responsible for a detailed description of the PCJ library and its new functionalities, as well as for preparing and running performance tests evaluating the library’s efficiency in solving the *2D stencil* problem on selected supercomputers. As part of this research, I received a grant to run programs using the PCJ library on the *Hazel Hen* supercomputer at the HLRS in Germany, which at the time was ranked 17th on the TOP500 list [O15]. This made it possible to perform performance tests on over 4000 compute nodes of this supercomputer, allowing the application to run on nearly 200 000 compute threads. Additionally, I described the obtained results along with a discussion of the achieved outcomes. I was also responsible for the work as a whole.

HPC The HPC problems presented in the paper are the *2D stencil* and *FFT*. Below, I will focus primarily on the *2D stencil* as a solution created and discussed solely by me.

The *2D stencil* was demonstrated using John Conway’s *Game of Life* cellular automaton [O16]. In my implementation, the board (universe) is neither infinite nor cyclic – it has a defined height and width, as well as edges. Each thread has its own sub-board (containing a portion of the universe) – the board is divided into equal parts in a block-wise manner. Although there are fast algorithms for generating the next state of the automaton, I decided that my implementation would directly check the state of each cell. To save memory, each cell is represented as a single bit, where 0 indicates a dead cell and 1 indicates a live cell. After calculating the new state of the board-universe (after a time step), the cells at the edges of the sub-boards are asynchronously exchanged with the corresponding neighboring threads.

For comparison purposes, I also prepared a version written in C using MPI (C/MPI) and a version in Java using MPI (Java/MPI). The main code was based on the implementation using PCJ, but communication was carried out using MPI functions.

For performance measurements, for each time step I calculated the number of cells processed per unit of time (*cells/s*). Each test consisted of 11 time steps. This ensured a *warm-up* of the JVM, allowing it to perform *just-in-time* (JIT) compilation for execution optimization. I also made sure that the *garbage collector* (GC) did not affect the measured performance. For this purpose, I used the peak (maximum) performance measured over the entire computation.

Figure 10 presents a performance comparison of the *Game of Life* application for a board of size 604 800×604 800 cells. The performance of both implementations (PCJ and Java/MPI) is very similar and provides nearly ideal scalability. The C/MPI version offers three times higher performance while maintaining similar scalability. The performance data demonstrate scalability up to 98 304 threads (on 2048 nodes). For a larger number of threads, the parallel processing performance decreases due to the small workload per processor relative to the communication time required for exchanging edge data.

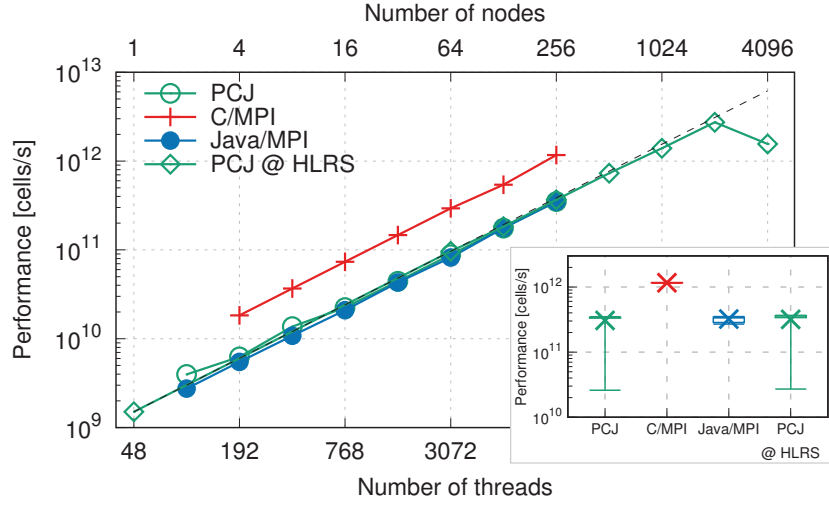


Figure 10: Performance results for *Game of Life* for a board size of $604\,800 \times 604\,800$ cells. Dashed lines indicate ideal scaling for PCJ. Higher values indicate better results. (from [5])

The inset chart shows performance statistics calculated over 11 time steps of the *Game of Life* application executed on 256 nodes (12288 threads) of the *Okeanos* supercomputer. The ends of the *whiskers* indicate the minimum and maximum values, the cross ($\times$) represents the mean, the box represents the values between the first and third quartiles, and the line inside the box indicates the second quartile (median). In the case of C/MPI, the box and whiskers are not visible because the execution exhibits very similar performance across all steps. For PCJ and Java/MPI executions, the minimum values come from the first steps, when the code was running in interpreted mode. The JIT compilation quickly optimized the running code, and the vast majority of steps were executed at peak performance. It is clearly visible that Java applications, after JIT compilation, have very stable performance results, as the maximum values and quartiles (median, first and third quartiles) are almost indistinguishable in the figure.

Big Data The discussion of the Big Data problem was the task of the co-authors. For the paper, they chose *WordCount* as an example application for processing large datasets. They compared the performance between the PCJ solution, APGAS, and a C++/MPI-based implementation. For PCJ, they used an implementation that included a reduction using a self-implemented binary tree for communication [A4]. For APGAS, they used both the base APGAS_{stat} implementation and the APGAS_{dyn} implementation extended with dynamic load balancing. The APGAS library and the *WordCount* implementation were based on the version presented in [O17]. For the native solution, they chose C++ instead of C, as it was more suitable for this task, due to its standard library contains a built-in `map` type and high-level functions for operating on strings.

The performance results presented in the paper show good scalability of the PCJ implementation, with the reduction phase being the key stage affecting overall performance. The PCJ implementation achieves better results than the APGAS-based version, due to the use of an efficient reduction mechanism based on binary tree communication. In turn, the C++ implementation demonstrates ideal scalability, but its execution time is about four times longer, which results from the more expensive step of splitting text lines into individual words. It is also worth noting that the results presented in [A5] obtained on a Hadoop cluster indicate that the PCJ implementation is at least three times faster than the Hadoop implementation.

Artificial Intelligence As part of research combining the PCJ library with artificial intelligence, the co-authors examined the possibility of using the library in two types of tasks.

In the first case, the library was used to model the connectome of the nematode *Caenorhabditis Elegans*. This nematode is the only organism with a fully mapped connectome. Experiments were conducted with the goal of linking the connectome structure to the actual behavior of the nematode [O18]. The results of these experiments allowed biophysical experts to create a mathematical model of the relevant part of the connectome. The model was defined as a set of ordinary differential equations with 8 parameters. To explore the solution space and fit the model parameters so that its predictions matched empirical data, a differential evolution algorithm was applied. The mathematical model was implemented in Java and parallelized using the PCJ library. It is worth noting that the prototype solution was developed in a very short time (around 2 months), despite changing requirements from the biophysical expert.

The second type of tasks was related to machine learning. The co-authors developed a dedicated program using the TensorFlow Java API for distributed execution and instrumentation of the neural network training process. The reduction operation implemented in this solution was based on an *all-reduce* algorithm executed in a hypercube topology [O19]. The PCJ library was responsible for distributing the computations, monitoring them, and handling communication, while the actual computations were performed by TensorFlow.

To demonstrate correctness, they trained a neural network to recognize handwritten digits from the MNIST dataset [O20]. This dataset contains 60 000 training images, of which 5000 were reserved for validation in the presented test. The neural network consisted of three fully connected layers with 300, 100, and 10 neurons, respectively [O21]. It was trained for 20 epochs with mini-batches of 50 images. The co-authors tested two learning algorithms. In the first, computations were performed in parallel, and gradients were averaged after each batch, similarly to Horovod [O22]. The second algorithm implements asynchronous parallel gradient descent as described by other authors [O23]. The performance of these solutions was compared to a Horovod implementation based on C, MPI, and Python. The solutions achieved an accuracy of approximately 96%.

The performance results presented in the paper show that the asynchronous PCJ implementation has comparable performance to Horovod, but achieving full Horovod performance required changing the default TensorFlow installation configuration, whereas the TensorFlow version used for Java did not require any additional configuration.

The results of the study demonstrate that PCJ can serve as an efficient framework for parallel execution of AI tasks, and leveraging the Java language and its ecosystem made it relatively straightforward to integrate with existing tools. Using the PCJ library, TensorFlow could be run on an HPC cluster in a multi-node setup while maintaining high performance.

PCJ 5.1.0

Version 5.1.0 of the PCJ library introduced a number of functional changes that go beyond simple bug fixes and organizational improvements.

First, in the **broadcast** operation, I modified the message acknowledgment mechanism – significantly reducing the number of messages sent to node #0. Instead of all nodes sending acknowledgment messages directly to node #0, I implemented a binary tree structure for this communication.

In addition, I changed the application launch interface: instead of the `PCJ.start` and `PCJ.deploy` methods, I introduced a construction based on the *builder* pattern – `PCJ.executionBuilder` – with analogous `start` and `deploy` methods. This approach allows the execution configuration to be prepared in advance, including the specification of computation nodes and other properties.

I also added the `PCJ.accumulate` method, which allows modifying a shared variable according to a given function and value – for instance, to increment the stored value by one. This version also introduced new collective operations, such as `PCJ.reduce` and `PCJ.collect`, which will be discussed in the following sections.

Fail-safe mechanism In version 5.1.0, I introduced a fail-safe mechanism, described in detail in [5]. The purpose of this mechanism is to ensure the safe and controlled termination of an application in the event of a system or runtime failure. Without such protection, computations could enter a deadlock state while the application itself remained active. In high-performance cluster environments, this would lead to unnecessary consumption of CPU-hours without producing any useful computational output until the job’s time limit was reached.

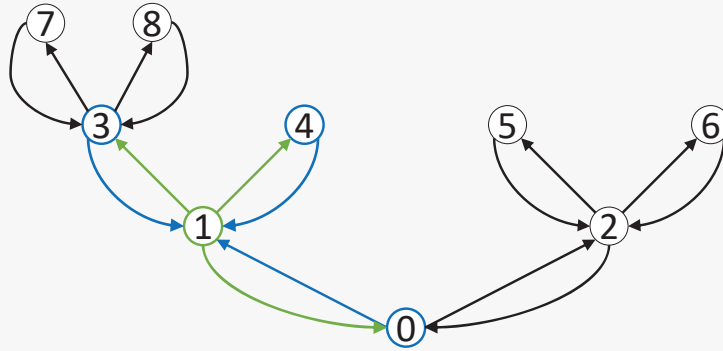


Figure 11: Message tree for the *fail-safe* mechanism with node #1’s neighbors highlighted. Green arrows represent activity messages sent to neighbors, and blue arrows represent messages received by node #1 from neighbors. (from [5])

The *fail-safe* mechanism in the PCJ library operates in such a way that, in the event of a failure, the computation is safely terminated. Failure detection is performed in two ways. The first method is related to a runtime error in a PCJ thread, such as one caused by an uncaught exception. In such a case, the internal execution supervisor marks the affected node as failed. The second method is based on a *heartbeat* mechanism. Activity messages are periodically sent by each node to its neighbors – the parent node and its children (Figure 11). If a node does not receive an activity message from one of its neighbors within a specified time, it assumes that a failure has occurred. Similarly, a failure is detected when it is not possible to send an activity message.

When a failure is detected, the node immediately sends termination messages to its active neighbors and stops the computation. Each node that receives such a message forwards it to its remaining neighbors (excluding the source node) and also terminates its execution. As a result, the termination message propagates to all nodes, and the application is shut down in a controlled manner.

4.1.5 TeraSort [3] – PCJ 5.1.0

The paper [3] describes the problem of sorting large datasets in a multi-node environment. It presents the *TeraSort* algorithm, its implementations, and a performance comparison using Apache Hadoop and PCJ. In the case of the PCJ implementation, version 5.1.0 of the library was used. This study was carried out as part of the computational grants I obtained, titled *Benchmarking Java on Ethernet Cluster*, on the *1000 Core Cluster* at the Future SOC Lab, Hasso-Plattner Institute (HPI; Potsdam, Germany). Each node was equipped with four 10-core Intel Xeon E7-4870 processors @ 2.4 GHz (with *hyper-threading*, a total of 80 computation threads) and 1 TB of RAM. Preliminary results were published in technical reports from these grants: [A6, A7, A8].

The TeraSort algorithm is one of the widely used tools for testing the performance of Hadoop. When evaluating performance, the time required to sort 100-byte records is measured. Each record consists of a 10-byte key and a 90-byte value. In the case discussed in this work, the number of records to be sorted ranged from 10^7 to 10^{10} , which means that the input file size varied from approximately 1 GB to about 1 TB.

***Samplesort* algorithm** The TeraSort algorithm is based on the *samplesort* (or *sample sort*) algorithm. The *samplesort* algorithm uses the *divide-and-conquer* method. At the beginning of the *samplesort* procedure, $p - 1$ pivots are selected. The pivots are sampled from the input data and then sorted using another sorting algorithm. Next, the input data are divided into p buckets based on the $p - 1$ pivots. In the following step, the contents of the buckets are being sorted. In the original *samplesort* algorithm, the buckets are sorted recursively using the same algorithm; however, if the bucket size falls below a certain threshold, a different sorting algorithm is applied. Finally, a simple concatenation of the buckets produces the sorted result.

The *samplesort* algorithm is very well-suited for parallelization. The number of buckets should be equal to the number of computing units (e.g., processors) – p . In the ideal case, the number of elements in each bucket will be the same, ensuring that each processor spends a similar amount of time on sorting.

The *samplesort* is a generalization of the *quicksort* algorithm, differing in that *quicksort* uses only one pivot element. Similarly to the *quicksort* algorithm, the proper selection of *pivots* is a key step in the algorithm, as the sizes of the buckets – and consequently the performance of the solution – depend on them.

TeraSort algorithm In the paper, I present an implementation of the TeraSort algorithm included in the Apache Hadoop package, written according to the map-reduce model. Here, however, I will omit that description and focus exclusively on the solution I developed using the PCJ library. The algorithm itself consists of 5 steps:

1. selection of pivots,
2. reading the input data and placing them into the appropriate buckets,
3. exchange of buckets between threads,
4. sorting the data within the buckets,
5. writing the data from the buckets.

The first part of the algorithm is the selection of pivots. This is a key component of the algorithm that affects the practical performance of the application. I performed the selection in the following way: each thread reads the first $\lceil \frac{100\,000}{p} \rceil$ elements of its input data, and then these elements are collected by thread #0 using the reduce operation as a single list of 100 000 candidates. Thread #0 sorts the list and evenly selects $p - 1$ dividing elements, producing the $pivots_{0..p-1}$ list. Then the broadcast operation is used to pass the pivots to each thread, and the second step of the algorithm follows. In the paper [A7], I also presented a simpler and faster method for selecting dividing elements, but it does not produce as balanced a data distribution as the one presented here.

In the second step of the algorithm, each thread reads its portion of the data and immediately places it into the appropriate local buckets.

In the following step, the threads send the data from their local buckets to the corresponding threads. In this way, thread #0 receives the data from the buckets of all threads where the numbers are smaller than the smallest pivot ($pivots_0$). Thread t receives the data that are greater than or equal to $pivots_{t-1}$ and smaller than $pivots_t$. The thread with the highest index has elements that are greater or equal to the largest dividing element ($pivots_{p-1}$).

In the fourth step, each thread sorts the received elements using a sorting algorithm from the standard Java library.

The final step is writing the result. In the paper, I presented three approaches to writing the results:

1. sequential writing – one thread writes to a single output file while the others wait for their turn,
2. concurrent writing – all threads can write simultaneously to the same file using memory-mapped separate sections of the file,
3. writing to separate files – each thread writes its results to a separate file.

As can be expected, sequential writing to a single file is the slowest approach. When sorting 10^9 records (an input file of 100 GB) using 640 threads, the writing alone took 24 minutes, whereas concurrent writing reduced this time to 15 minutes, and writing to separate files took less than 5 minutes. Of course, writing to separate files requires merging files to obtain exactly the same result as in the other approaches, which also takes time. The tests performed showed that the average merging time was just under 14 minutes. Since the algorithm running in Hadoop creates separate output files, the third approach was chosen for further work, and the file merging step was omitted.

Another optimization of the algorithm that I implemented was the observation that steps 2 and 3 can be performed in parallel – sending data from the buckets can take place at the same time as reading the file. This takes advantage of the PGAS model, which allows one-sided asynchronous communication. Instead of waiting for the completion of step 2, the number of elements in a bucket is checked as elements are added (e.g., 100, 1000, 10 000, or 100 000). If the bucket becomes full, it is sent to the appropriate thread. Performance tests showed that for the largest data sets (10^{10} records), the most efficient limit is 1000. For smaller input data sizes, no significant speedup was observed compared to the original version.

Estimation of Computational Complexity Given the full algorithm, its optimistic, average, and pessimistic computational complexity can be calculated. The calculations were performed for p processors, i.e. $p-1$ dividing elements, and for an input data size of n ($n \gg p$). The following analysis is

presented from the perspective of a single processor in the optimistic case. Selecting candidates for the dividing elements has a cost of $O(p)$, since each of the p processors sends a constant number of elements to processor 0. Sorting the candidates by processor 0 has a cost of $O(p \log p)$, their even selection is $O(p)$, and broadcasting the selected dividing elements to all processors has a cost of $O(p \log p)$, as p elements are broadcasted using a communication tree, which is a balanced binary tree of depth $O(\log p)$. Reading the data by a processor and placing it into buckets has a cost of $O(\frac{n}{p} \log p)$, because for each of the $\frac{n}{p}$ elements the correct bucket must be found, which has a complexity of $O(\log p)$ when using a binary search algorithm, and insertion into the bucket can be performed in constant time $O(1)$. Distributing the buckets to the appropriate processors has a cost of $O(\frac{n}{p})$, and sorting its own bucket by a single processor has a cost of $O(\frac{n}{p} \log \frac{n}{p})$.

Similarly to the *quicksort* algorithm, the time complexity depends on the selection of pivots, i.e. the partitioning into buckets. The best partition occurs when each bucket contains the same number of elements. It turns out that the average computational complexity of the algorithm is the same as the optimistic complexity, but with a larger constant hidden in the O notation, for the same reasons that the average complexity of the *quicksort* algorithm is the same as its optimistic complexity. Ultimately, the average computational complexity is of the order:

$$O\left(\frac{n}{p} \log \frac{n}{p} + p \log p\right)$$

In the pessimistic case, all buckets except one contain 1 element, while the remaining elements go into the last bucket. The complexity in this case is:

$$O(n \log n + p \log p)$$

In the above calculations, it is assumed that the average and worst-case complexity of the internal sorting algorithm for n elements is $O(n \log n)$.

Performance Results Since the goal was to compare the implementation of the algorithm using Hadoop and PCJ, the next step was to read and write data through PCJ using the HDFS (Hadoop Distributed File System), which is the native file system for Hadoop. For the largest data sets, the algorithm's execution time when using HDFS incurred an overhead of 3%, while for small data sets (10^7 records) the execution time with HDFS was even twice as long.

Once the TeraSort algorithm implementation using PCJ was ready, a performance study of the Hadoop-based solution was conducted. Due to the need to use the SLURM job scheduling system, it was natural that one job would serve as the Hadoop cluster with so-called *datanodes* storing the data, while other jobs would be responsible for processing the data. Additionally, in most SLURM configurations, the launched jobs receive exclusive access to the nodes on which they are executed. As a result, the data-processing jobs, i.e., in the case of Hadoop, the components with *nodemanagers*, were executed on separate compute nodes (*external mode*). Moreover, this mechanism allowed setting a limit on the number of nodes that process the data. Figure 12a shows the execution times of the algorithm for different numbers of records. The results demonstrated that the performance of the PCJ-based solution was much higher than the Hadoop-based solution. This was because this mode is *unnatural* for Hadoop, which cannot take advantage of the benefits related to data locality.

I therefore decided to test a different approach, where when creating the Hadoop cluster, *datanodes* and *nodemanagers* were launched together on the same compute nodes within the same SLURM job (*internal mode*). This resulted in a constant number of *nodemanagers* throughout all

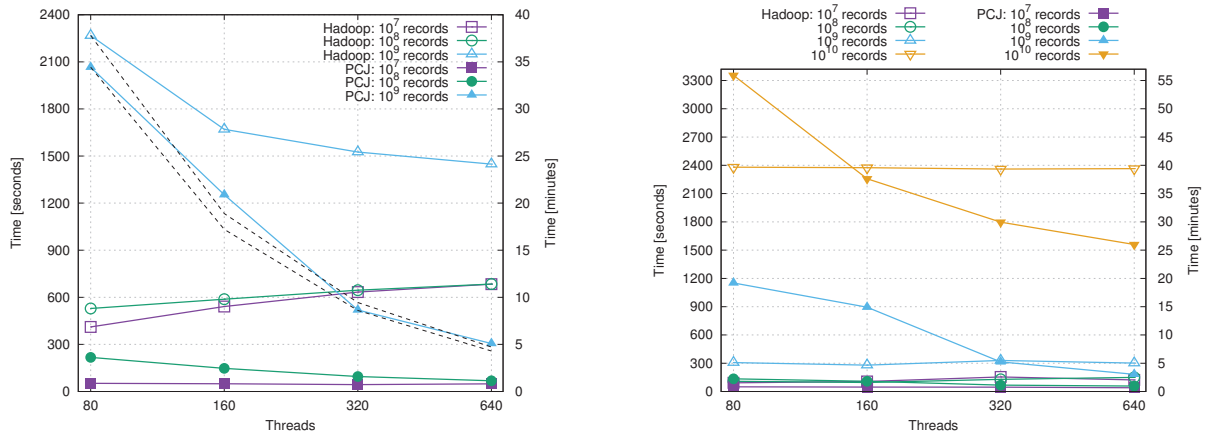


Figure 12: Total execution time of the TeraSort algorithm implementation in Hadoop and PCJ. Lower values indicate better performance. (from [3])

tests, meaning that jobs could be placed on any node without limits. In this case, the PCJ-related jobs were executed in an unnatural way for SLURM, by being attached to the Hadoop cluster job. The results of this approach are shown in Figure 12b. The Hadoop-based solution shows approximately constant execution time, suggesting that there is no limit on the number of containers running within the cluster. The cluster chose to execute the mapping and reduction tasks on all nodes to minimize the data that must be transmitted between the data nodes and the compute nodes. The PCJ-based solution also benefits from being executed on the same nodes as the data, and ultimately, using 640 threads, this solution is more efficient than the Hadoop-based solution.

The large gap in total execution time between PCJ and Hadoop required verification of the Hadoop cluster configuration, in particular the determination of an appropriate value for the *maximum allocated memory* for map and reduce tasks. Previously, this value had been set to 40 GB. Considering that each node has at least 820 GB of free RAM and 40 physical cores (80 logical cores using *hyper-threading*), the maximum number of map and reduce tasks was set to 80 for each type. Taking these parameters into account, performance tests were conducted for different values of *maximum allocated memory* for the tasks: 5 GB, 10 GB, 20 GB, and 40 GB. The shortest execution times were obtained when this value was set to 10 GB or 20 GB, but for the largest input data (10^{10} records) the highest performance was achieved with 10 GB, and this value was used in the subsequent performance tests. This setting allows full utilization of the available memory without excessive overallocation.

In this work, I included many other performance results. The final result, shown in Figure 13, is the most important, as it summarizes the obtained outcomes. It presents the strong scaling of the TeraSort algorithm implementation created using both Apache Hadoop and PCJ for different numbers of records (i.e. 10^7 , 10^8 , 10^9 , and 10^{10} records). For the tests, the execution environment was prepared to ensure that Hadoop ran as optimally as possible. In the same environment, under the same conditions, the PCJ-based solution was also executed.

The total execution time of the PCJ-based application largely depended on the number of nodes and threads used – with a fixed number of threads, running the application on more nodes was more efficient. For smaller input data sizes (10^7 and 10^8 records), the PCJ implementation is more efficient. The sorting time for 10^9 records using Hadoop and PCJ is similar. For the largest data set (10^{10} records), the Hadoop-based solution is more efficient for a smaller number of threads, while the difference is almost negligible for the largest number of threads tested (640).

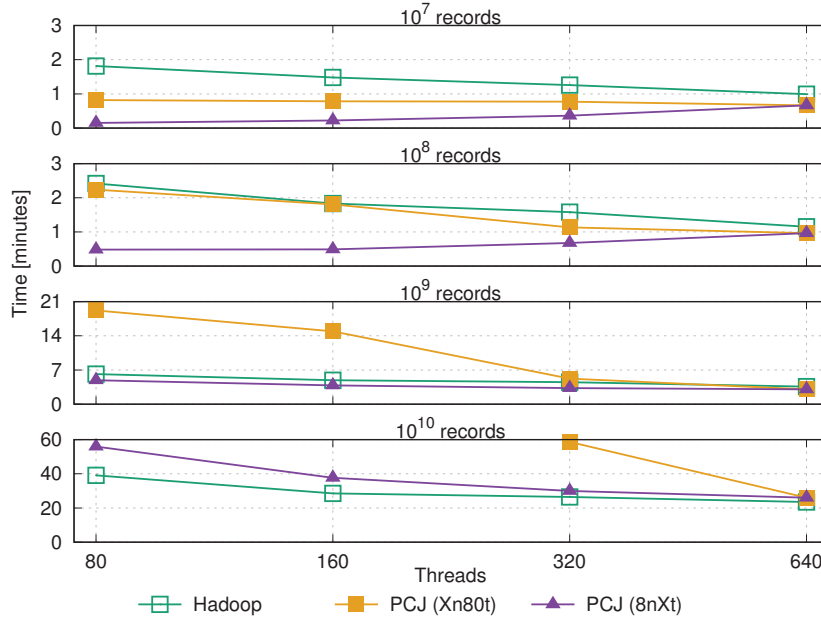


Figure 13: Comparison of the performance of the TeraSort algorithm implementation in Hadoop and PCJ for different numbers of records in the strong scaling regime. Each map and reduce task in Hadoop was executed using 10 GB of memory. Compute nodes were placed on the same nodes as the data nodes. PCJ results were collected using 1, 2, 4, and 8 nodes (with 80 threads per node; Xn80t) and on 8 nodes (with 10, 20, 40, and 80 threads per node; 8nXt). Lower values indicate better performance. (from [3])

In summary, the studies presented in this work demonstrated similar performance for Hadoop and PCJ. The PGAS programming model, represented by the PCJ library, can be a highly competitive alternative to the map-reduce model – both in terms of potential performance benefits and software development efficiency. This model provides a clear abstraction for constructing parallel applications based on a global address space, and additionally offers control over data layout and its distribution among processors. Importantly, PGAS is a general-purpose model that can be applied to a wide range of computational problems, unlike the map-reduce model, in which the formulated problem must be adapted to the constraints and characteristics of the model itself.

PCJ 5.2.0

Version 5.2.0 of the PCJ library introduced significant changes to the thread group creation mechanism.

Previously, to join a group, one had to call the `PCJ.joinGroup("groupName")` operation, providing the name of the group to which the thread should be assigned. This operation could be performed at any time. However, to ensure correct execution and consistency of group information among threads, it was necessary to make sure that all threads recorded the execution of this operation, for example, by invoking a barrier on the parent group (e.g., the global one).

An additional limitation of the earlier solution was the lack of an easy way to determine the order of threads within the created group – to achieve this, each thread had to be joined individually, requiring a barrier between consecutive joins. This approach was inconvenient to use and sometimes led to hard-to-diagnose errors.

Version 5.2.0 introduced a new group creation mechanism that resolved the described issues at the cost of limiting the ability to extend groups after their creation. However, with this

new approach, the order of threads within a group can be specified directly at creation time. All threads of the parent group (e.g., the global one) execute the `PCJ.splitGroup` operation together, passing the *color* (or `null` if the thread is not joining any group) and a *key* as parameters. Threads with the same color are placed in the same group, and their order is determined based on the key value. This way, with a single collective operation, multiple groups can be created, where the order of threads is clearly defined and consistent with the user's intent.

This version also introduced a new collective operation: `PCJ.scatter`. A detailed discussion of collective operations is presented in the following subsection.

4.1.6 Collectives [6] – PCJ 5.3.0

Collective operations are communication operations in parallel environments that simultaneously involve a group of processes. Unlike *point-to-point* communication, which occurs between two specific processes, collective operations coordinate the exchange or synchronization of data in an organized, group-based manner. The standardization of collective operations took place in 1994 with the publication of the MPI-1 specification [O24, O25]. Examples of collective operations include *barrier* (a mechanism ensuring that all processes reach the same execution point before continuing computations), *broadcast* (sending data from one process to all others), *scatter* and *gather* (distributing and collecting data among processes), and *reduce* (combining partial results from multiple processes into a single global result).

Most PGAS languages provide *barrier* and *broadcast* operations, while other collective operations are rarely available as built-in methods or procedures. In practice, this means that users must implement such mechanisms on their own. Optimized and scalable collective operations are a key factor influencing the performance of developed solutions.

Up to version 5.1.0, the only collective operations available in the PCJ library were `barrier()` and `broadcast(T newValue, Enum<?> name)` (and their asynchronous versions, `asyncBarrier()` and `asyncBroadcast(...)`). The development of collective operations in the PCJ library is presented in the next paper from the cycle [6]. In this work, I was responsible for preparing the research concept, implementing collective mechanisms in the PCJ library, and performing performance tests for the *FFT* and *WordCount* problems. I extended the previous implementations of *FFT* and *WordCount* by introducing the use of collective operations available in the new version of the library. I was also responsible for preparing nearly the entire manuscript text.

PCJ 5.3.0

Work on extending the PCJ library interface with additional collective operations had been underway for quite some time. In version 5.1.0, I introduced the operations `reduce(ReduceOperation<T> function, Enum<?> name)` and `collect(Enum<?> name)`, while version 5.2.0 added the `scatter(...)` operation. Version 5.3.0 brought substantial functional improvements in this area. In particular, I modified the signature of the *scatter* method, which now accepts a map as its parameter: `scatter(Map<Integer, T> newValueMap, Enum<?> name)`. Additionally, I renamed the `collect` method to `gather` to avoid confusion among Java developers – in *Java Streams*, introduced in Java 8, there is already a `collect` method with broader semantics and capabilities. At the same time, I introduced a new method, `collect(SerializableSupplier<Collector<T, ?, R>> collectorSupplier, Enum<?> variable)`,

whose behavior closely resembles that of the `collect` method in *Java Streams*. Each of these operations also has its asynchronous counterpart.

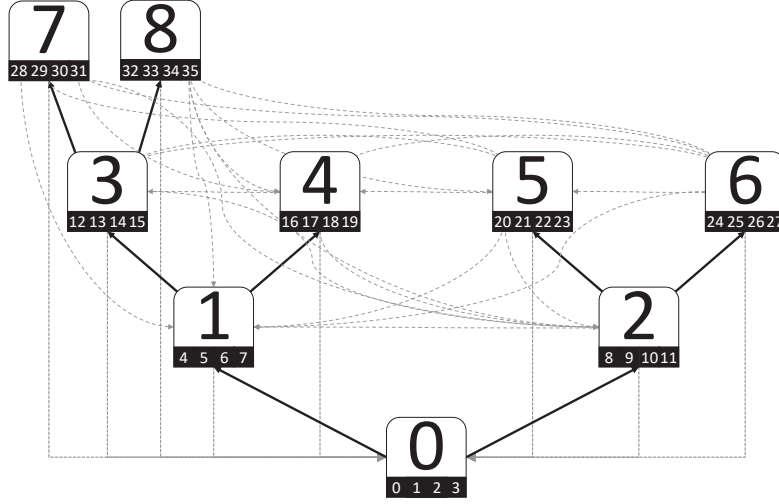


Figure 14: Communication scheme of nodes in PCJ. Node is represented by round single corner rectangles - large number represents node id, tiny numbers represent thread ids. Various types of arrows mean established TCP/IP connections during the startup phase: the elbow arrow represents a connection made to node #0; the bold, straight arrow represents a connection to a child node that creates a communication binary tree; the curved arrows represent other connections between each pair of nodes. (from [6])

The implemented collective operations utilize the most efficient communication via a balanced binary tree (Figure 14). Collective operations are staged – performed first at the thread level and then at the node level. For example, during a reduction, values are first summed across all threads within a node, and then the aggregated value is passed to the parent node. As a result, collective operations using n nodes and t threads per node are executed in $O(\log n + t)$ time and are significantly faster than simple iteration over threads, which has a time complexity of $O(nt)$. Additionally, it should be noted that communication operations, especially inter-node data transfers, are time-consuming, and introducing staged execution of collective operations allows faster completion of the requested operation.

In version 5.1.0, the communication scheme for executing collective operations was simple. The node that initiated the collective operation (the *requester*) sent a request to node #0, and then communication proceeded through the tree – from the root, through intermediate nodes, to the leaves, and then back: from the leaves, through the intermediate nodes, to the root. The resulting value or a notification of completion was collected at node #0 and then sent back to the *requester* node. However, when many threads attempted to perform a collective operation simultaneously, performance decreased because all threads relied on node #0 as the root, which became a bottleneck.

In version 5.2.0, I introduced an optimization based on *shifting* nodes in the communication tree. Thanks to this optimization, not only does the *requester* node become the root of the tree, but the entire tree is also reorganized, which significantly improves communication performance. Figure 15 shows both the original and the *shifted* trees, in which the *requester* node is node #5.

Figure 16 presents the performance results of the *FFT* and *WordCount* applications. The results were collected using the *Okeanos* supercomputer (ICM UW).

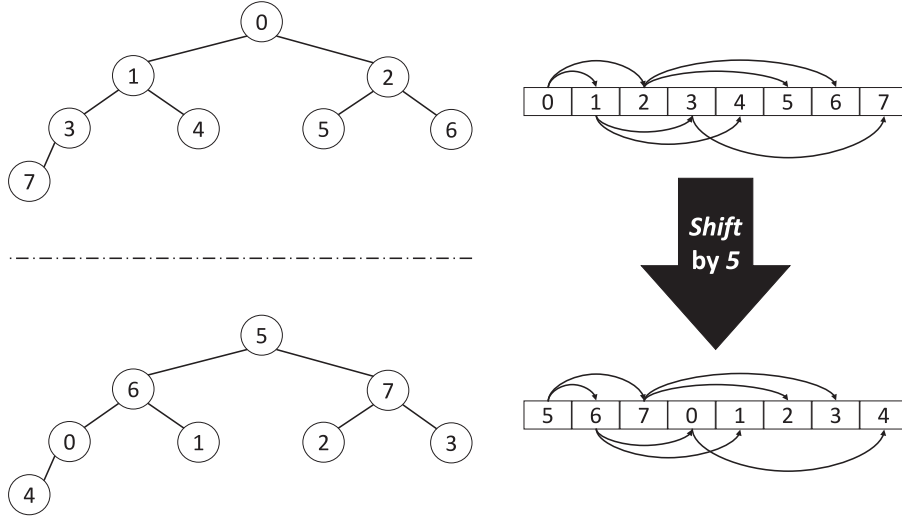
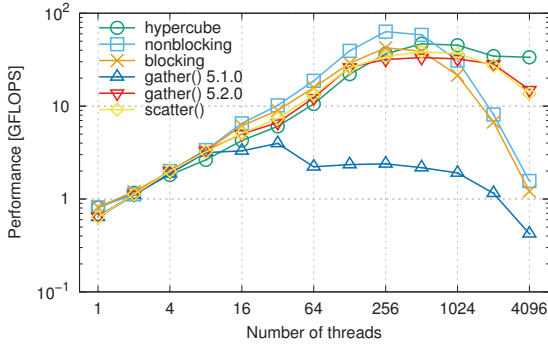
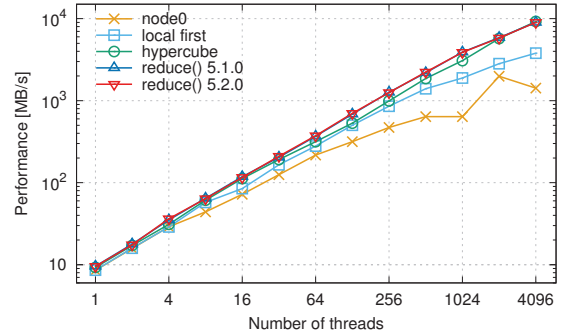


Figure 15: Communication scheme for collective operations using the *shift* of nodes in the communication tree. (from [6])



(a) *FFT* on 2^{27} elements



(b) *WordCount* for a 10 MB file in the weak scalability regime.

Figure 16: Performance results with different implementations of collective operations. (from [6])

Figure 16a shows the performance of the one-dimensional *FFT* on 2^{27} elements. I compared solutions using three *all-to-all* communication schemes independently implemented by the co-author with the newly introduced collective operations in the PCJ library: `gather(...)` and `scatter(...)`. It is easy to observe the performance improvement of the *gather* operation resulting from the library upgrade from version 5.1.0 to 5.2.0. The version using the *shifted* tree achieves performance almost comparable to the carefully crafted *hypercube*-based implementation. The poor performance of *gather* in version 5.1.0 is caused by the bottleneck at node #0. In version 5.2.0, the performance of *gather* and *scatter* methods is slightly lower than in the *hypercube* version because the *hypercube* implementation is optimized for a specific task and sends data directly to the target threads, whereas *gather* and *scatter* involve all nodes in communication. On the other hand, the code using *gather* and *scatter* is more readable and easier to maintain.

Figure 16b presents the performance results for the *WordCount* problem. For the analysis, I used the text of the novel *Artamène ou le Grand Cyrus* by Georges de Scudéry, encoded in ISO-8859-1, with a size of 10 MB. Measurements were conducted in the weak scaling regime, in which each thread reads the entire file, and a reduction operation is performed at the end of the computation. As in the *FFT* test, I compared solutions using three reduction schemes independently implemented by the co-author with the newly introduced collective operations: `gather(...)` and `scatter(...)`.

The results indicate good scalability of the PCJ implementation, although performance gradually decreases with an increasing number of threads due to the growing volume of data exchanged during communication. Scalability is almost linear for the *hypercube* and *reduce()* implementations, confirming the high efficiency of the reduction phase of the algorithm. In this case, no performance differences were observed between PCJ versions 5.1.0 and 5.2.0, as the reduction operation is executed only once by thread #0. The other two implementations exhibit lower performance.

The collective operations introduced in PCJ version 5.1.0 and further refined in versions 5.2.0 and 5.3.0 reduce programming effort and simplify the development of efficient and scalable code.

4.1.7 HPV-KITE [7] – PCJ 5.3.3

The development of the PCJ library enables the creation of applications that can run on multi-core and multi-node systems in a flexible and intuitive manner.

The final paper in the cycle [7], which describes HPV-KITE (a solution I developed), was produced as part of the HEAP (Human Exposome Assessment Platform) project, where I served as the leader of work package WP9 during the final year of the project.

HPV-KITE is an application that enables rapid searching of large genomic datasets to identify sequences of the *Human papillomavirus* (HPV). It was designed for fast and accurate detection of HPV viruses in virtually any data obtained from *next-generation sequencing* (NGS) technologies. The parallel processing core of the application was implemented using the PCJ library.

In earlier approaches to sequence processing, after obtaining sequencing data from the device, the data had to be preprocessed, including the removal of sequence fragments corresponding to human DNA. Depending on the size of the dataset (for example, 260 files totaling 302 GB), preprocessing could take up to a week. In the developed solution, preprocessing is no longer required, and the analysis of the same data took less than 16 minutes using eight nodes of the Puhti supercomputer (CSC, Finland), producing comparable qualitative results.

The main idea behind HPV-KITE is to compute the Tversky index [O26] for sets containing *shingles* (n-element sequences; also known as *k-mers* or *n-grams*) derived from the processing of both the reference database (in FASTA format) and the input sequences (*samples*, in FASTQ format). The process of creating a set of *shingles* involves scanning the sequence with a sliding window (step size of 1) of a fixed width (by default, 31 characters) and adding the observed substrings to the set. In the implementation described in the paper, built-in classes implementing the `java.util.Set` interface are used to store the *shingle* sets and compute the size of their intersection, although work is underway to apply dictionary-based data structures for this purpose [A9].

Given these sets, the Tversky index can be computed as follows:

$$S(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X \setminus Y| + \beta|Y \setminus X|}$$

where X is the set derived from the input data (FASTQ files), Y is the set from the reference database containing sequences of the HPV virus family, and α (alpha) and β (beta) are the parameters (weights) of the Tversky index. In this work, the parameters α and β were set to 0 and 1, respectively, because the goal is to identify sequences in the FASTQ file (X) that show high similarity to known HPV sequences in the reference database (Y), where only elements unique to the reference database (Y) affect the similarity score. Accordingly, the equation can be simplified to the following form:

$$C(X, Y) = \frac{|X \cap Y|}{|Y|}$$

By calculating the value according to the formula above, it is possible to determine the *coverage*, understood as the number of unique *shingles* belonging to a given HPV type present in the database, for the analyzed input sequence. To classify a sample as infected, we set a threshold of 10% viral coverage (corresponding to 10% coverage of the HPV genome). Since the values are calculated separately for each virus, the results also provide information about co-infections with multiple HPV types. The exact number of covered k-mers depends on the specific HPV virus and the width of the sliding window. In the default configuration, with a window width of 31, in the reference HPV database used (PaVE [O27] as of February 23, 2024), HPV213 has the smallest number of unique *shingles* (7065), while HPV83 has the largest (8073); thus, the 10% threshold corresponds to values above 706 and 807 *shingles*. On average, viruses have about 7484 *shingles*, and the median number of *shingles* is 7367.

An advantage of the developed solution, in addition to its speed and accuracy, is the ability to utilize multiple computation nodes through the use of the PCJ library. In this implementation, PCJ version 5.3.3, the latest available to date, was used. The code is based on Java 17, and the tests were conducted using the OpenJDK Runtime Environment version 21.0.2.

PCJ 5.3.3

The changes introduced from version 5.3.0 to the current version 5.3.3 of the PCJ library were primarily organizational. In version 5.3.1, I improved the mechanism for adding nodes in `ExecutionBuilder`, making it more robust to user-provided data (e.g., trimming whitespace from the beginning and end of node names, validating names, preventing empty names, and ignoring blank lines in the node list file). Version 5.3.2 simplified the use of the `@Storage` and `@RegisterStorage` annotations by allowing their parameters to be omitted – the values are then automatically inferred from the context. In version 5.3.3, I standardized the naming of methods for local operations on shared variables; for example, the `getLocal` method was renamed `localGet` (analogously to `asyncGet`).

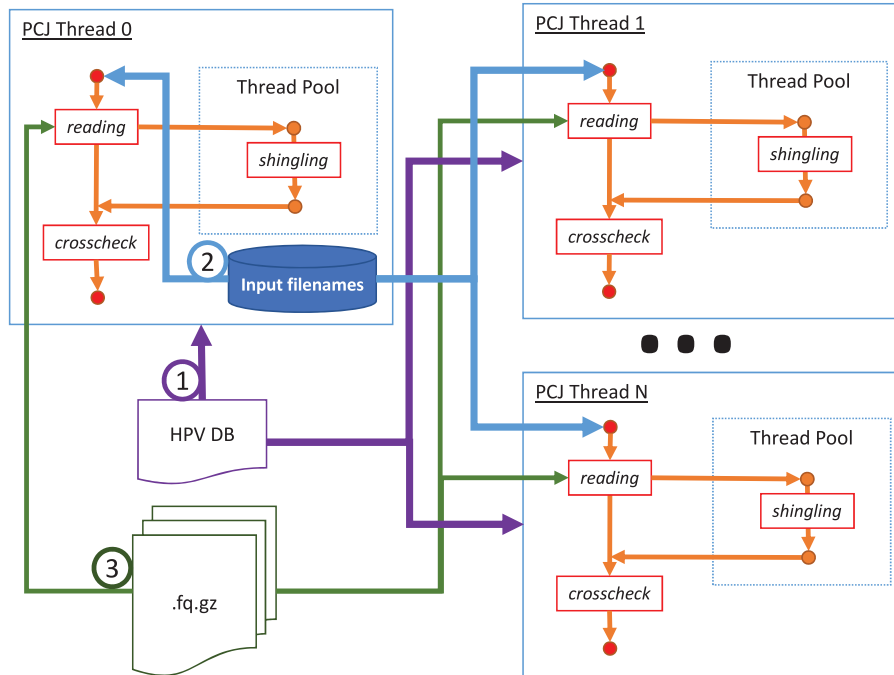


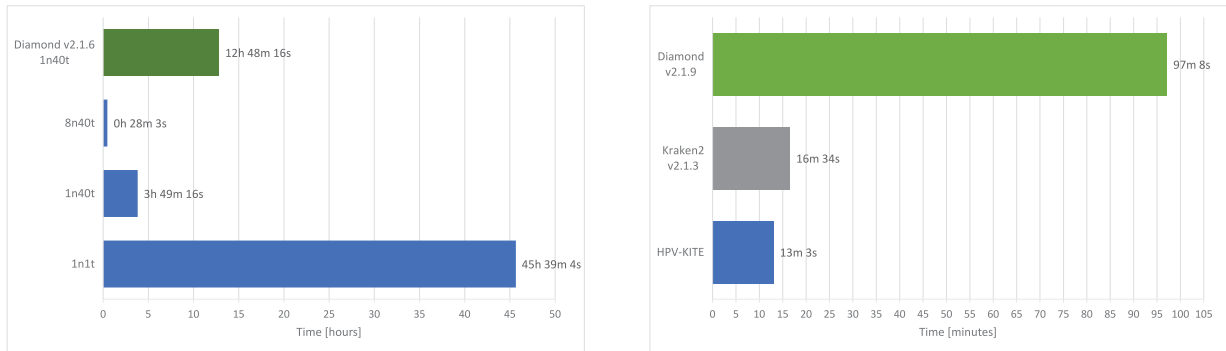
Figure 17: Schematic view of the HPV-KITE execution. (from [7])

Figure 17 presents a schematic overview of the application execution. For optimal parallel processing, I applied a *load-balancing* algorithm. Each node runs a single PCJ thread responsible for

the parallel processing of input files using local threads. Execution parameters, such as the name of the HPV database file, the *shingle* length, or the list of files, are passed to the main process. The supervising thread (*PCJ thread #0*) creates a shared object that stores the list of files to be processed. This object is an instance of `java.util.concurrent.ConcurrentLinkedQueue` due to its properties ensuring safe concurrent access. Next, all threads simultaneously read the reference database (initially, it was planned that the supervising thread would process the reference database and pass it to the other threads, but performance tests showed that this approach is slower). Once the reference database is available, each PCJ thread retrieves the path to the next file to process from the supervising thread. Fetching paths to subsequent files is performed using a mechanism that allows tasks to be delegated in the context of another PCJ thread. This mechanism is available via the `at` method (or in the asynchronous version: `asyncAt`). After completing the processing of a given file, a thread requests the path to the next file, repeating this process until all files are processed, and then terminates.

In some of the datasets provided to me, a single *sample* consisted of multiple files (typically 2, 4, or 8). For example, 8 files for one sample occur when four *lanes* are used during sequencing, with each *lane* containing reads in both directions (*paired-end*). All these files can, however, be unambiguously associated with the same sample, as they share a common name prefix.

In the HPV-KITE solution, I also designed and implemented support for parallel processing of samples consisting of multiple files, allowing a single result to be obtained for each sample. For example, in the case of 8 files belonging to one sample and 8 PCJ threads processing the data, each PCJ thread processes a separate file, after which a combined result is generated for the entire sample.



(a) Total processing time of the entire Finnish dataset for HPV-KITE and Diamond on the Puhti supercomputer. (b) Total processing time of the TCGA subset for HPV-KITE, Diamond, and Kraken2 on the Mahti supercomputer.

Figure 18: Processing times for sequential data using HPV-KITE. Lower values indicate better results. (from [7])

Figure 18a presents the processing times for the sequence dataset from the Finnish HPV vaccination trial (96 samples) on the Puhti supercomputer (CSC, Finland). A single node consisted of two 20-core Intel Xeon Gold 6230 CPU @ 2.10 GHz (40 cores in total) and 384 GB of RAM. The results indicated that relying on a single node and a single thread leads to suboptimal performance, highlighting the need for parallelization. Parallelizing the *shingle* generation significantly improved performance – the dataset was processed in approximately 4 hours (11 times faster), and comparing this time with the execution time of Diamond showed more than a threefold improvement. Moreover, extending this solution to multi-node execution reduced the computation time to less than half an hour.

Figure 18b shows the processing times for a subset of sequences from *The Cancer Genome Atlas* (TCGA) database. TCGA contains fully sequenced genomes of over 20 000 cancer samples. From

the entire database, we selected only samples with primary tumors from cervix uteri (304 samples) and endometrium (590 samples), totaling 894 samples. Processing was performed on a single node of the Mahti supercomputer (CSC, Finland) equipped with two 64-core AMD EPYC 7H12 processors @ 2.6 GHz (128 cores in total) and 256 GB of RAM. The most efficient solution was HPV-KITE, completing the processing in 13 minutes, followed by Kraken2 (16 minutes), while Diamond was the slowest, taking 97 minutes.

The study also presents results evaluating the accuracy of the outputs generated by HPV-KITE. The experimental results show that HPV-KITE provides accurate detection (sensitivity > 96%) of the most prevalent HPV types in DNA and RNA metagenomes, compared to standard, computationally intensive direct mapping algorithms such as Diamond. Furthermore, compared to PCR-based genotyping using the Luminex assay, HPV-KITE achieved a sensitivity of 89% and specificity of 91% in detecting both single and multiple HPV infections (18 samples).

Summary

In the presented cycle of publications, I have described the development and analysis of the PCJ library, an implementation of the PGAS paradigm for the Java language, in the context of its applications across various areas of parallel computing in multi-node environments. The works in this cycle cover, among others, bioinformatics applications (PCJ-BLAST, HPV-KITE), studies on the use of cloud environments, integration of HPC, Big Data, and Artificial Intelligence workloads, the implementation and analysis of the TeraSort algorithm in the PGAS model, as well as the development of collective operation mechanisms in the PCJ library. These achievements constitute a significant contribution to the advancement of research on parallel programming, with a particular focus on applications in the natural sciences.

The successive versions of the PCJ library, along with the corresponding scientific publications included in this cycle, were shown in Figure 19.

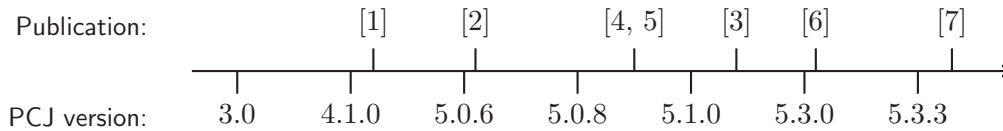


Figure 19: Diagram of subsequent versions of the PCJ library and their corresponding publications included in the cycle.

4.2 Overview of Other Publications

In this subsection, I provide a brief, thematically organized overview of my other publications.

4.2.1 Other Publications

Among other publications related to the development of the PCJ library, not covered in the main discussion of the results, there are publications and abstracts released before I obtained my PhD [B1, B2, B3, B4, B5, B6] as well as those published afterward [A10, A11]. Additionally, in [A12, A13, A14], we presented implementations of graph algorithms using the PCJ library. In paper [A15], we described version 4.1 of the library and discussed, among other things, parts of the implementation of the *MolDyn* test from the *Java Grande Benchmark Suite*, used for simulating

particle motion in the *N-body* problem. In paper [A16], we demonstrated the feasibility of implementing fault tolerance mechanisms within the PCJ library, while in [A17], we presented performance studies of the library on specialized high-core-count processors designed for massive parallelism, specifically the *Intel Xeon Phi Knights Landing (KNL)*.

The paper [A18] is a brief technical report from a grant awarded to Benson Muite, from the University of Tartu, at the *Future SOC Lab HPI*, in which I executed *PingPong* tasks using the PCJ library to evaluate the feasibility of running PCJ applications and their communication performance on a cluster with 10 Gb/s network bandwidth.

The paper [A19] is an abstract from an *e-poster* session organized during the *55th European Society of Human Genetics (ESHG) Conference*, briefly presenting the status of research conducted by work package WP9 within the HEAP project as of 2022.

4.2.2 Education

ZawodyWeb The papers [E1, E2], published before I obtained my doctoral degree (during my master’s studies), present the *ZawodyWeb* system, which I developed. This system enables automatic verification of programming solutions. In the described version, it supported languages such as Pascal, C, C++, Java, PHP, and Python, and also allowed for manual evaluation of solutions. The *ZawodyWeb* system facilitated the checking of algorithmic tasks developed using web technologies for the *Informatics Contest* (in Polish: *Konkurs Informatyczny*). It was also used, among others, during classes of the *University Computer Science Course* (in Polish: *Uniwersyteckie Koło Informatyczne*) for high school and middle school students.

The paper [E3], also published before I obtained my doctoral degree, describes the further development of the *ZawodyWeb* system and its application in the educational process for both full-time and part-time Computer Science students at the Faculty of Mathematics and Computer Science, Nicolaus Copernicus University.

The paper [E4], published after I obtained my doctoral degree, presents the development of the *ZawodyWeb* system in the context of teaching parallel programming and its integration with the UNICORE software, enabling task submission on a multi-node infrastructure. It demonstrates that the *ZawodyWeb* system allows for the verification of solutions using OpenMP and MPI technologies in C and C++, as well as solutions developed in Java using the PCJ library.

School The papers [E5, E6], published before I obtained my doctoral degree, describe the *Regional Computer Science Course* (in Polish: *Regionalne Koło Informatyczne*) project – the first initiative of its kind in the country aimed at popularizing programming and teaching it to high school students. The courses, held at the Faculty of Mathematics and Computer Science at UMK, were conducted in an innovative remote learning format. For this purpose, we also used the *ZawodyWeb* system.

The paper [E7], published after I obtained my doctoral degree, provides a brief description of hands-on workshops held during the *Informatyka w Edukacji XV* conference in 2018, dedicated to teaching the basics of programming in Java.

The paper [E8], also published after obtaining my doctoral degree, discusses the teaching of parallel programming in high schools and universities. It presents the PCJ library and a mini-cluster built from Raspberry Pi minicomputers as a platform enabling effective introduction of parallel programming at the high school level.

Scratch The paper [E9], published before I obtained my doctoral degree, presents the visual programming environment *Scratch*. It introduces the basic principles of creating applications in this environment and provides examples of its use in teaching programming in primary and middle schools.

The paper [E10], published after I obtained my doctoral degree, contains a brief description of hands-on workshops conducted during the *Informatyka w Edukacji XV* conference in 2018, dedicated to the new version of the *Scratch* environment.

The paper [E11], also published after obtaining my doctoral degree, presents the scenario and experiences from conducting preschool activities as part of *Scratch Day* in 2019 and 2022.

The paper [E12], published after I obtained my doctoral degree, provides a full description of hands-on workshops I conducted during the *Informatyka w Edukacji XX* conference in 2024. Workshop participants were introduced to the visual programming language *Scratch* and guided step-by-step to create a *Snake*-type game within this environment.

Other works published after obtaining the PhD

- [A1] Marek Nowicki and Piotr Bała. Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ. In Łukasz Koźmiński, Jacek Duskocz, and Piotr Kardasz, editors, *Innowacje w polskiej nauce w obszarze matematyki i informatyki : przegląd aktualnej tematyki badawczej / red. nauk. Łukasz Koźmiński, Jacek Duskocz, Piotr Kardasz*, pages 130–140. 2016.
- [A2] Marek Nowicki, Davit Bzhalava, and Piotr Bała. Massively parallel sequence alignment with BLAST through work distribution implemented using PCJ library. In Shadi Ibrahim, Kim-Kwang Raymond Choo, Zheng Yan, and Witold Pedrycz, editors, *Algorithms and architectures for parallel processing : 17th International Conference, ICA3PP 2017, Helsinki, Finland, August 21-23, 2017 : proceedings / eds. Shadi Ibrahim, Kim-Kwang Raymond Choo, Zheng Yan, Witold Pedrycz*, volume 10393 of *Lecture Notes In Computer Science*, pages 503–512. Springer International Publishing, 2017.
- [A3] Marek Nowicki, Łukasz Górski, and Piotr Bała. Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud. In Bartosz Balis, Dora B. Heras, Laura Antonelli, Andrea Bracciali, Thomas Gruber, Jin Hyun-Wook, Michael Kuhn, Stephen L. Scott, Didem Unat, and Roman Wyrzykowski, editors, *Euro-Par 2020 : Parallel Processing Workshops : Euro-Par 2020 International Workshops : Warsaw, Poland, August 24-25, 2020 : revised selected papers / ed. by Bartosz Balis, Dora B. Heras, Laura Antonelli, Andrea Bracciali, Thomas Gruber, Jin Hyun-Wook, Michael Kuhn, Stephen L. Scott, Didem Unat, Roman Wyrzykowski*, number 12480 in *Lecture Notes In Computer Science*, pages 213–224, 2021.
- [A4] Marek Nowicki, Łukasz Górski, and Piotr Bała. Performance evaluation of parallel computing and Big Data processing with Java and PCJ library. In *Visionary Computing: CUG 2018, Stockholm, Sweden, 20-24 May 2018.*, pages 1–14. Cray User Group, 2018.
- [A5] Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, and Piotr Bała. Big Data analytics in Java with PCJ library : performance comparison with Hadoop. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman*,

Konrad Karczewski, volume 10778 of *Lecture Notes In Computer Science*, pages 318–327. Springer International Publishing, 2018.

- [A6] Marek Nowicki. Benchmarking Java on Ethernet Cluster: benchmarking PCJ – a library for parallel computing in Java. In Christoph Meinel, Andreas Polze, Karsten Beins, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, and Jürgen Müller, editors, *HPI Future SOC Lab: proceedings 2019*, volume 158 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 263–271. Universitätsverlag Potsdam, 2024.
- [A7] Marek Nowicki. Benchmarking the sort algorithm on Ethernet Cluster : Benchmarking PCJ : a library for parallel computing in Java. In Christoph Meinel, Andreas Polze, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, Jürgen Müller, and Jürgen Sommer, editors, *HPI Future SOC Lab : proceedings 2020*, volume 159 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 53–63. Universitätsverlag Potsdam, 2024.
- [A8] Marek Nowicki. Benchmarking the TeraSort algorithm on Ethernet Cluster : Benchmarking Apache Hadoop based implementation. In Christoph Meinel, Andreas Polze, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, Jürgen Müller, and Jürgen Sommer, editors, *HPI Future SOC Lab : proceedings 2020*, volume 159 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 97–108. Universitätsverlag Potsdam, 2024.
- [A9] Marcin Piątkowski, Magdalena Mroczek, and Marek Nowicki. Słownikowe struktury danych i ich zastosowania. In Anna Beata Kwiatkowska, editor, *Informatyka w edukacji : edukacja informatyczna a kompetencje cyfrowe XXI wieku*, pages 80–90, 2025.
- [A10] Marek Nowicki and Piotr Bała. Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ. In *Materiały konferencyjne - Innowacyjne projekty badawcze: [II Ogólnopolska e-Konferencja Innowacyjnych Projektów Badawczych IPB]*, Wrocław, 2 września 2016., pages 30–30. Dolnośląski Akcelerator Technologii Innowacji Sp. z o.o., 2016.
- [A11] Marek Nowicki, Łukasz Górski, Magdalena Ryczkowska, and Piotr Bała. PCJ as a tool for massively parallel data processing. In *CGW Workshop’17, Kraków, Poland, October 23-25, 2017: proceedings / eds. Marian Bubak, Michał Turała, Kazimierz Wiatr.*, pages 39–40. Academic Computer Centre Cyfronet. AGH, 2017.
- [A12] Magdalena Ryczkowska, Marek Nowicki, and Piotr Bała. Level-synchronous BFS algorithm implemented in Java using PCJ Library. In Hamid R. Arabnia, Leonidas Deligiannidis, and Mary Yang, editors, *2016 International Conference on Computational Science and Computational Intelligence (CSCI 2016), 15-17 December 2016, Las Vegas, Nevada, USA : proceedings / ed. by Hamid R. Arabnia, Leonidas Deligiannidis, Mary Yang*, pages 596–601, 2016.
- [A13] Magdalena Ryczkowska, Marek Nowicki, and Piotr Bała. The performance evaluation of the Java implementation of Graph500. *Lecture Notes In Computer Science*, 9574:221–230, 2016.
- [A14] Magdalena Ryczkowska and Marek Nowicki. Performance comparison of graph BFS implemented in MapReduce and PGAS programming models. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski*, volume 10778 of *Lecture Notes In Computer Science*, pages 328–337. Springer International Publishing, 2018.

- [A15] Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, Michał Szynekiewicz, and Piotr Bała. PCJ : a Java library for heterogenous parallel computing. In *Recent advances in information science: proceedings of the 15th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS '16) ; proceedings of the 10th International Conference on Computer Engineering and Applications (CEA '16), Barcelona, Spain, February 13-15, 2016 / ed.: Xiaodong Zhuang.*, pages 66–72. World Scientific and Engineering Academy and Society Press, 2016.
- [A16] Michał Szynekiewicz and Marek Nowicki. Fault-tolerance mechanisms for the Java parallel codes implemented with the PCJ library. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski*, volume 10778 of *Lecture Notes In Computer Science*, pages 298–307. Springer International Publishing, 2018.
- [A17] Marek Nowicki, Łukasz Górski, and Piotr Bała. Evaluation of the parallel performance of the Java and PCJ on the Intel KNL based systems. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics : 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017 : revised selected papers. Part 2 / eds.: Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, Konrad Karczewski*, volume 10778 of *Lecture Notes In Computer Science*, pages 288–297. Springer International Publishing, 2018.
- [A18] Marek Nowicki, Benson K. Muite, and Mehmet Can Boysan. Benchmarking FFT on an Ethernet Cluster: Latency, Bandwidth and MPI benchmarks. In Christoph Meinel, Andreas Polze, Karsten Beins, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, and Jürgen Müller, editors, *HPI Future SOC Lab: proceedings 2019*, volume 158 of *Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam*, pages 93–98. Universitätsverlag Potsdam, 2024.
- [A19] Magdalena Mroczek, Marek Nowicki, Łukasz Górski, Sara Arroyo Mühr, Alex Ormenisan, and Piotr Bała. High performance AI based tool for mining and classification of viral genomes in human exposome metadata. *European Journal of Human Genetics*, 31(suppl. 1):292–292, 2023.

Other works published before obtaining the PhD

- [B1] Marek Nowicki and Piotr Bała. Parallel computations in Java with PCJ library. In *Proceedings of the 2012 International Conference on High Performance Computing & Simulation : HPCS 2012, July 2 - July 6 2012, Madrid, Spain / ed. Waleed W. Smari.*, pages 381–387. IEEE, 2012.
- [B2] Marek Nowicki and Piotr Bała. PCJ - new approach for parallel computations in Java. *Lecture Notes In Computer Science*, 7782:115–125, 2013.
- [B3] Łukasz Górski, Marek Nowicki, and Piotr Bała. Komunikacja wątków w bibliotece PCJ. In *XXVII Forum Informatyki Teoretycznej, Toruń, 11-14 kwietnia 2013.*, pages 28–28. [s.n.], 2013.
- [B4] Marek Nowicki and Piotr Bała. The performance of the PCJ library for massively parallel computing. In *Proceedings of the 7th International Conference on PGAS Programming Models, 3-4th October 2013, Edinburgh, UK / ed. by M. Weiland, A. Jackson and N. Johnson.*, pages 254–254. The University of Edinburgh, 2013.

- [B5] Rafał Kluszczyński, Marcin Stolarek, Grzegorz Marczak, Łukasz Górski, Marek Nowicki, and Piotr Bała. Development of parallel codes using PL-Grid infrastructure. In *Cracow '14 Grid Workshop, October 27-29 2014, Krakow, Poland.*, pages 81–82. Academic Computer Centre Cyfronet. AGH, 2014.
- [B6] Marek Nowicki, Łukasz Górski, Patryk Grabarczyk, and Piotr Bała. PCJ - Java library for high performance computing in PGAS model. In Waleed W. Smari and Vesna Zeljkovic, editors, *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014) / ed. by W.W. Smari, V. Zeljkovic*, pages 202–209. 2014.

Educational-related works

- [E1] Marek Nowicki, Rafał Kluszczyński, and Piotr Bała. Internetowe narzędzia do sprawdzania zadań programistycznych i ich zastosowania. In *V Konferencja "Informatyka w Edukacji" - IwE 2008, Toruń, 3-6 lipca 2008 / red. nauk. Anna Beata Kwiatkowska, Maciej M. Sysło.*, pages 88–94. Agencja Wydawnicza Argi, 2008.
- [E2] Rafał Kluszczyński, Łukasz Mikulski, Marek Nowicki, and Piotr Bała. Contests Hosting Service as a tool to teach programming. *Annales Universitatis Mariae Curie-Skłodowska. Sectio AI: Informatica*, 9:213–224, 2009.
- [E3] Marek Nowicki, Łukasz Mikulski, Marcin Piątkowski, Rafał Kluszczyński, and Piotr Bała. On-line contests hosting service as a tool to teach computer science students programming. In Nicholas Reynolds, Mary Webb, Valentina Dagiene, and Maciej Sysło, editors, *WCCE 2013 : 10th IFIP World Conference on Computers in Education, Toruń, Poland; July 1-7, 2013. Vol. 2 / eds. Nicholas Reynolds, Mary Webb, in coop. with Valentina Dagiene, Maciej M. Sysło*, pages 61–71. 2013.
- [E4] Marek Nowicki, Maciej Marchwiany, Maciej Szpindler, and Piotr Bała. On-line service for teaching parallel programming. *Lecture Notes In Computer Science*, 9523:78–89, 2015.
- [E5] Piotr Bała, Anna Kwiatkowska, Marek Nowicki, Mariusz Strzelecki, and Maciej Sysło. Regionalne Koło Informatyczne. In *Informatyka w edukacji: materiały z warsztatów VII Konferencji IwE 2010 / red. Anna Beata Kwiatkowska.*, pages 61–74. Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011.
- [E6] Marek Nowicki, Michał Matuszak, Anna Kwiatkowska, and Maciej Sysło. Teaching secondary school students programming using distance learning : a case study. In Nicholas Reynolds, Mary Webb, Valentina Dagiene, and Maciej Sysło, editors, *WCCE 2013 : 10th IFIP World Conference on Computers in Education, Toruń, Poland; July 1-7, 2013. Vol. 2 / eds. Nicholas Reynolds, Mary Webb, in coop. with Valentina Dagiene, Maciej M. Sysło*, pages 246–254. 2013.
- [E7] Piotr Bała, Łukasz Górski, and Marek Nowicki. Java w szkole. In Anna Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : myśl komputacyjnie! / red. Anna Beata Kwiatkowska, Maciej M. Sysło*, pages 171–172. 2018.
- [E8] Piotr Bała and Marek Nowicki. Czy jesteśmy gotowi na naukę programowania równoległego w szkole? In Anna Beata Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : edukacja informatyczna : wyzwania współczesnego świata*, pages 127–136, 2023.

- [E9] Piotr Bała and Marek Nowicki. Scratch - środowisko do nauki programowania. In *Informatyka w edukacji: materiały z warsztatów VII Konferencji IwE 2010* / red. Anna Beata Kwiatkowska., pages 33–40. Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011.
- [E10] Piotr Bała and Marek Nowicki. Scratch 3.0 : nowe możliwości. In Anna Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : myśl komputacyjnie!* / red. Anna Beata Kwiatkowska, Maciej M. Sysło, pages 173–174. 2018.
- [E11] Marek Nowicki. Scratch w przedszkolu. In Anna Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : uczniowie i nauczyciele w szkole przyszłości*, pages 414–424, 2022.
- [E12] Marek Nowicki. Scratch : podstawy na przykładzie gry Wąż. In Anna Beata Kwiatkowska and Maciej Sysło, editors, *Informatyka w edukacji : kształcenie informatyczne drogą do cyfrowej transformacji społeczeństwa*, pages 284–294, 2024.

5 Scientific activity

Presentation of significant scientific or artistic activity carried out at more than one university, scientific or cultural institution, especially at foreign institutions.

Apart from my employment at the **Nicolaus Copernicus University in Toruń** and the **University of Warsaw**, I have also completed research internships at the following national and international research institutions, where I demonstrated significant scientific activity:

- research visit at the **High Performance Computing Center Stuttgart (HLRS)**, University of Stuttgart, Germany (June–July 2017);
- six-month research internship at **IBM Zurich Research Laboratory**, Zurich, Switzerland (Mathematical & Computational Science Department) (May–November 2013);
- semester-long internship at the Laboratory of Biomolecular Machines, University of Warsaw (October 2011–March 2012).

In addition to employment at more than one university and participation in research internships, my scientific activity has also involved collaboration with, among others, researchers from Karolinska Institutet (Sweden), where I held the status of an *Affiliated Researcher* at the *Center for Cervical Cancer Elimination*, part of *Department of Clinical Science, Intervention and Technology*, as well as with researchers from Oulun Yliopisto (*University of Oulu*, Finland).

I have participated in international research projects:

- HEAP (Human Exposome Assessment Platform; funded by the European Union under the Horizon 2020 programme – Grant Agreement ID: 874662; DOI: 10.3030/874662):
since 02/2021 – **researcher**, 05/2024–06/2025 – **principal investigator** (ICM UW; WP9)
- HPDCJ (Heterogeneous Parallel and Distributed Computing with Java; funded by the CHIST-ERA consortium – in Poland funded by the National Science Centre, grant no. 2014/14/Z/ST6/0007):
10/2014–09/2017 – *technical coordinator of the work package* (**researcher**)
- APOS (Application Performance Optimization and Scalability; EU 7th Framework Programme):
2011–2013 – software developer (before obtaining the PhD degree)

- Kardionet (EEA Funds):
2008–2011 – software developer (during master’s studies, before obtaining the PhD degree)

I have participated in national and international conferences and summer schools, where I delivered talks, lectures, and conducted *tutorials*, including:

- *HPV-KITE for high-performance metagenomics data analysis* (talk and poster), BioInformatics in Toruń 2025; 25-28/06/2025, Toruń
- *HPV-KITE for high-performance metagenomics data analysis*, EHEN Scientific Meeting 2024; 26-27/11/2024, Brussels, Belgium
- *Parallel Programming in Java in the PCJ Library* (**invited lecture**), TSSP ExSci – Toruń Students Summer Program in Exact Sciences, 14/07/2022, Toruń
- *Big Data, processing and storage* (**invited lecture**), 4EU+ Against Cancer Summer School, 6/07/2022, Warsaw
- *PCJ - Java Library that integrates HPC, Big Data and AI workloads*, International Symposium on Grids & Clouds 2021 (ISGC2021); 22-26/03/2021; Academia Sinica Computing Centre (ASGC), Taipei, Taiwan (Online due to COVID-19)
- *Performance Evaluation of Java/PCJ Implementation of Parallel Algorithms on the Cloud*, **26th International European Conference on Parallel and Distributed Computing (Euro-Par 2020)**, 24-28/08/2020, Warsaw, Poland (Online due to COVID-19)
- *Benchmarking Java on Ethernet Cluster / Benchmarking PCJ - a library for parallel computing in Java*, 19th HPI Future SOC Lab Day - Fall 2019, 12/11/2019, Potsdam, Germany
- *High Performance Computing And Big Data Analytics With PCJ And Java* (**tutorial**), Supercomputing Frontiers Europe 2018; 12-15/03/2018; Warsaw, Poland
- *Evaluation of the parallel performance of the Java and PCJ on the Intel KNL based systems*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM’2017), 10-13/09/2017, Lublin, Poland
- *Parallel Computing in Java* (**tutorial**), 12th International Conference On Parallel Processing And Applied Mathematics (PPAM’2017), 10/09/2017, Lublin, Poland
- *Massively Parallel Sequence Alignment with BLAST Through Work Distribution Implemented Using PCJ Library*, **17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2017)**, 21-23/08/2017, Helsinki, Finland
- *Level-synchronous BFS Algorithm Implemented in Java Using PCJ Library*, The International Conference on Computational Science and Computational Intelligence (CSCI’16); 15-17/12/2016, Las Vegas, Nevada, USA
- *PCJ5* (invited lecture), Team seminar, Cognitive Computing & Computational Sciences Department, IBM Research Lab – Zurich, 14/10/2016, Zurich, Switzerland
- *Parallel Computing in Java* (**tutorial**), 11th International Conference On Parallel Processing And Applied Mathematics (PPAM’2015), 6-9/09/2015, Kraków

- *The performance of the PCJ library for massively parallel computing* (poster), 7th International Conference on PGAS Programming Models, 3-4/10/2013, Edinburgh, Scotland, Great Britain (before obtaining the PhD degree)
- *PCJ – a Partioned Global Address Space Approach for Parallel Computations in Java*, Exascale Applications and Software Conference (EASC2013): Solving Software Challenges for Exascale, 9-11/04/2013, Edinburgh, Scotland, Great Britain (before obtaining the PhD degree)
- *Parallel computations in Java with PCJ library*, The 2012 International Conference on High Performance Computing & Simulation (HPCS 2012), 2-6/07/2012, Madrid, Spain (before obtaining the PhD degree)
- *PCJ - New Approach for Parallel Computations in Java*, PARA 2012: Workshop on State-of-the-Art in Scientific and Parallel Computing, 11-13/06/2012, Helsinki, Finland (before obtaining the PhD degree)

I was also a member of the organizing and scientific committees of international conferences, including:

- **scientific committee member** of the workshop: Workshop on PGAS Programming, PPAM 2019 Conference: 13th International Conference on Parallel Processing and Applied Mathematics, 8-11/09/2019, Białystok
- **organizing committee member** of the international conference: Petri Nets 2016 // ACSD 2016: 37th International Conference On Applications And Theory Of Petri Nets And Concurrency, 16th International Conference On Application Of Concurrency To System Design, 19-24/06/2016, Toruń

I was an **assistant supervisor** for PhD student MSc. Magdalena Ryczkowska, whose dissertation topic was *Development of selected parallel graph algorithms in the PGAS model and their implementation using the Java language* (in Polish: *Opracowanie w modelu PGAS wybranych, równoległych algorytmów grafowych i ich implementacja przy użyciu języka Java*; defense date: 5 November 2018; degree awarded: 8 November 2018; supervisor: Prof. Piotr Bala; field and discipline: technical sciences in computer science; institution: Institute of Computer Science, Polish Academy of Sciences).

My scientific activity is also related to **reviewing research papers** submitted to journals (including *Bioinformatics*, *BMC Bioinformatics*, *Journal of Big Data*, *SoftwareX*) and conferences (*Petri Nets 2020*, *PPAM 2022*). I have also **reviewed grant applications** as a member of the evaluation committee of the *Grants4NCUStudents* programme, 4th edition (2022), for research projects and international trips carried out by students and PhD candidates of Nicolaus Copernicus University in Toruń – ST panel.

During my master's and doctoral studies (2009-2015), I was a member of the scientific society *Association for Computing Machinery (ACM)* (ACM Student Member).

A full list of my achievements is available in the *List of Scientific or Artistic Achievements* document.

6 Presentation of teaching, organizational and science popularization achievements

Presentation of teaching and organizational achievements as well as achievements in popularization of science or art.

- Courses taught:
 - for MSc students in Computer Science: *Software Reverse Engineering* (in Polish: *Inżynieria wsteczna oprogramowania*) (laboratory classes), *Research and Development Project* (in Polish: *Projekt badawczo-rozwojowy*) (team supervisor), *Introduction to Parallel Computing* (in Polish: *Wprowadzenie do obliczeń równoległych*) (lectures and laboratory classes), *Parallel and Distributed Programming* (in Polish: *Programowanie równoległe i rozproszone*) (lectures and laboratory classes), *Team Programming* (in Polish: *Programowanie Zespołowe*) (team supervisor), *Selected Applications of Computer Science* (in Polish: *Wybrane Zastosowania Informatyki*) (laboratory classes)
 - for BSc and Engineer students in Computer Science: *Parallel and Distributed Programming* (in Polish: *Programowanie równoległe i rozproszone*) (lectures and laboratory classes), *Programming III (Java)* (in Polish: *Programowanie III (Java)*) (laboratory classes), *Programming II (C,C++)* (in Polish: *Programowanie II (C,C++)*) (laboratory classes), *Introduction to Programming* (in Polish: *Podstawy Programowania*) (laboratory classes), *Preparation for Olympiads* (in Polish: *Przygotowanie do Olimpiad*) (laboratory classes)
 - for BSc students in Mathematics, specialization in Mathematics and Computer Science Teaching: *Programming II (Java)* (in Polish: *Programowanie II (Java)*) (laboratory classes)
 - in the postgraduate course: practical classes on *Applications of Big Data Tools in Omics Analyses* (in Polish: *zastosowania narzędzi Big Data w analizach omicznych*) in the postgraduate course *OMICS Data Science – Bioinformatics and Large-scale Biomedical Data Analysis* (in Polish: *Bioinformatyka i analiza wielkoskalowych danych biomedycznych*)
 - for children from the Children’s University of NCU: *Game Programming in Scratch* (in Polish: *zajęcia z programowanie gier w języku Scratch*), *Programming in Scratch* (in Polish: *Programowanie w języku Scratch*),
 - preparation of exercises and assistance in running the *Regional Computer Science Course* (in Polish: *Regionalne Koło Informatyczne*)
- Lectures, workshops, and other activities at science festivals:
 - *Who Can Solve the Problem Fastest with a Thousand Processors?* (in Polish: *Kto najszybciej rozwiąże problem, mając tysiąc procesorów?*) (together with Prof. Piotr Bala), lecture at the 19th Warsaw Science Festival 2015, 26 September 2015, Warsaw
 - *Even if a Thousand Athletes Came... or, on Parallel Programming* (in Polish: *I choćby przyszło tysiąc atletów... czyli o programowaniu równoległym*) (together with Prof. Piotr Bala), workshop at the 19th Warsaw Science Festival 2015, 26 September 2015, Warsaw
 - assistance in conducting a *mini-programming contest* (in Polish: *mini-konkurs programistyczny*) as part of the *24th Toruń Science and Arts Festival* (in Polish: *24. Toruński Festiwal Nauki i Sztuki*), April 2006
- Supervisor of numerous defended Engineer’s theses (15) and Master’s theses (7)

- participation in the Polish Academic Championships in Collegiate Programming, Wrocław, 26-27 October 2018, as co-coach of three teams of Computer Science students from the Faculty of Mathematics and Computer Science, NCU
- administration, maintenance, and development of the *ZawodyWeb* system for automatic testing of students' programming solutions (since 10/2009)
- participation in competition committees:
 - member of the Regional Committee in Toruń of *Informatics Olympiad* (in Polish: *Olimpiada Informatyczna*) (since 01/2014)
 - member of the jury of the hackathon *BrainCode 2019* organized by Allegro, 12-13 April 2019, Toruń
 - **chair** of the competition committee: *Great Programming Challenges Hackathon* (in Polish: *Hackaton Wielkie Wyzwania Programistyczne*) – edition III: *Switch to Smart (Switch to Smart)*, ICM UW, 23-24 November 2018, Warsaw
 - member of the jury of the hackathon *BrainCode 2018* organized by Allegro, 20-21 April 2018, Toruń
 - member of the judging committee of the competition *FRI: III Kuyavian-Pomeranian Programming Contest* (in Polish: *III Kujawsko-Pomorskie Zawody Programistyczne*), 20 April 2018, Toruń
 - **chair** of the competition committee: *Great Programming Challenges Hackathon* (in Polish: *Hackaton Wielkie Wyzwania Programistyczne*) – edition II: *Smart Charge (Smart Charge)*, ICM UW, 15-16 February 2018, Warsaw
 - member of the jury of the hackathon *BrainCode 2017* organized by Allegro, 23-24 March 2017, Toruń
 - member of the judging committee of the competition *FRI: II Kuyavian-Pomeranian Programming Contest* (in Polish: *II Kujawsko-Pomorskie Zawody Programistyczne*), 24 March 2017, Toruń
 - member of the competition committee: *Contest Coding with Scratch 2017* (in Polish: *Kodowanie ze Scratchem 2017*) – for primary school students, 26 May 2017, Toruń
 - **chair** of the competition committee: *Great Programming Challenges Hackathon* (in Polish: *Hackaton Wielkie Wyzwania Programistyczne*) – edition I: *Map the Gap (Map the Gap)*, ICM UW, 16-17 February 2017, Warsaw
 - member of the competition committee: *Contest Coding with Scratch 2016* (in Polish: *Kodowanie ze Scratchem 2016*) – for primary school students, 17 June 2016, Toruń
 - member of the competition committee: *Contest Coding with Scratch 2015* (in Polish: *Kodowanie ze Scratchem 2015*) – for primary school students, 19 June 2015, Toruń
 - assistance in conducting the *Informatics Contest* (in Polish: *Konkurs Informatyczny*) (editions: 2006/2007, 2007/2008, 2008/2009) at the Faculty of Mathematics and Computer Science, NCU
- articles presenting the results of scientific studies on websites
 - article describing HPV-KITE on **Forum Akademickie**
<https://forumakademickie.pl/naukowcy-z-uw-autorami-metody-szybkiego-wykrywania-sekwencji-wirusow-hpv/>

- article describing HPV-KITE on **Medonet**
<https://www.medonet.pl/biznes-system-i-zdrowie/digital-health-innovators,polscy-naukowcy-opracowali-hpv-kite--przelom-w-diagnostyce-wirusow-hpv,artykul,27980435.html>
- article describing HPV-KITE on **Serwis Naukowy UW**
<https://serwisnaukowy.uw.edu.pl/na-tropie-hpv-matematyka-przychodzi-z-pomoca-medycynie/>
- article in a popular science magazine
 - article describing HPV-KITE in **the magazine *Wiedza i Życie***, July 2025, no. 7 (1087), ISSN 0137-8929: *HPV-KITE – to win against time* (in Polish: *HPV-KITE – wygrać z czasem*; p. 21)
- server administration and support in organizing the Polish edition of the International *Math Kangaroo* Contest (in Polish: Międzynarodowy Konkurs *Kangur Matematyczny*), development of software for participant registration, results, and reports (since 09/2009)
- creation of the Polish translation of the visual programming environment *Scratch* developed by the Lifelong Kindergarten Group at MIT Media Lab (08-10/2008)

7 Other important information

Apart from information set out in 1-6 above, the applicant may include other information about his/her professional career, which he/she deems important.

- Implemented and deployed technologies
 - *ZawodyWeb* system – deployed, among others, at the Faculty of Mathematics and Computer Science, Nicolaus Copernicus University in Toruń
 - *PCJ* library – available in the central Maven repository, deployed, among others, in the *International ImMunoGenetics Information System (IMGT)*
 - *PCJ-BLAST* application – binary version available in the *GitHub* repository
 - *HPV-KITE* application – available, among others, in the *Conda-Forge* repository
 - *KITE* application – available, among others, in the *Conda-Forge* repository
 - the development and deployment of an internal **IBM** wide *Sales Decision Support* system that was adopted by more than 5000 sellers in IBM worldwide
- Awards and distinctions
 - **Individual distinction from the Rector** of Nicolaus Copernicus University in Toruń for scientific and research achievements in 2020, 25 June 2021
 - **Individual 3rd-degree award from the Rector** of Nicolaus Copernicus University in Toruń for scientific and research achievements in 2018, 26 September 2019
 - **Distinction for outstanding scientific achievements** in the university competition for priority research teams, Nicolaus Copernicus University, 28 March 2019 – team award; team *Formal Languages and Concurrency (FoLCo)*

- *HPC Challenge Class 2 Award: **Best Productivity – Most Elegant Solution*** at the international conference *SC14 – the International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing 2014)* – 16-21 November 2014, New Orleans, LA, USA – for the *PCJ* library
- Winner of the *IBM Great Minds student internships 2013*
- Sun Certified Programmer for the Java Platform, SE 6 (certification date: 16 December 2009)



applicant's signature

Bibliography

- [O1] Karl Feind. Shared memory access (SHMEM) routines. *Cray Research*, 1995.
- [O2] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global Arrays: A Non-Uniform-Memory-Access Programming Model For High-Performance Computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.
- [O3] Laxmikant V. Kale and Gengbin Zheng. *Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects*, chapter 13, pages 265–282. John Wiley & Sons, Ltd, 2009. doi:10.1002/9780470558027.ch13.
- [O4] William W. Carlson, Jesse M. Draper, David E. Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. Introduction to UPC and language specification. Technical report, Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, 1999.
- [O5] John Reid. The new features of Fortran 2008. In *ACM SIGPLAN Fortran Forum*, volume 27, pages 8–21. ACM, 2008.
- [O6] B.L. Chamberlain, D. Callahan, and H.P. Zima. Parallel Programmability and the Chapel Language. *The International Journal of High Performance Computing Applications*, 21(3):291–312, 2007. doi:10.1177/1094342007078442.
- [O7] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph Von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *Acm Sigplan Notices*, 40(10):519–538, 2005.

- [O8] Kathy Yelick, Luigi Semenzato, Geoff Pike, Carleton Miyamoto, Ben Liblit, Arvind Krishnamurthy, Paul Hilfinger, Susan Graham, David Gay, Phil Colella, et al. Titanium: a high-performance java dialect. *Concurrency and Computation: Practice and Experience*, 10(11-13):825–836, 1998.
- [O9] Olivier Tardieu. The apgas library: resilient parallel and distributed programming in java 8. In *Proceedings of the ACM SIGPLAN Workshop on X10*, pages 25–26. ACM, 2015.
- [O10] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.
- [O11] John Mellor-Crummey, Laksono Adhianto, Guohua Jin, Mark Krentel, Karthik Murthy, William Scherer, and Chaoran Yang. Class II submission to the HPC Challenge award competition Coarray Fortran 2.0. URL: http://www.hpcchallenge.org/presentations/sc2011/hpcc11_report_caf2_0.pdf.
- [O12] Łukasz Górski. *Adaptacja wybranych równoległych algorytmów do modelu PGAS i ich implementacja w języku Java z wykorzystaniem biblioteki PCJ*. PhD thesis, Instytut Podstaw Informatyki Polskiej Akademii Nauk, 2017.
- [O13] Leo Tolstoy. War and Peace. Accessed: 12.05.2021. URL: <http://www.gutenberg.org/files/2600/2600-0.txt>.
- [O14] Madeleine de Scudéry. Artamène ou le grand Cyrus. Accessed: 12.05.2021. URL: <https://github.com/xbsd/cyrusfiles/blob/master/cyrusprint.txt>.
- [O15] TOP500. Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect. Accessed: 30.10.2025. URL: <https://top500.org/system/178446/>.
- [O16] Martin Gardener. Mathematical Games: The fantastic combinations of John Conway’s new solitaire game ”Life”. *Scientific American*, 223(4):120–123, 1970.
- [O17] Jonas Posner, Lukas Reitz, and Claudia Fohry. Comparison of the HPC and Big Data Java Libraries Spark, PCJ and APGAS. In *2018 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM)*, pages 11–22. IEEE, 2018.
- [O18] Franciszek Rakowski and Jan Karbowski. Optimal synaptic signaling connectome for locomotory behavior in *Caenorhabditis elegans*: Design minimizing energy cost. *PLoS computational biology*, 13(11):e1005834, 2017.
- [O19] Ananth Grama, Vipin Kumar, Anshul Gupta, and George Karypis. *Introduction to parallel computing*. Pearson Education, 2003.
- [O20] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST Database of handwritten digits. Accessed: 17.03.2021. URL: <http://yann.lecun.com/exdb/mnist/>.
- [O21] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, Inc., 2017.
- [O22] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow, 2018. [arXiv:1802.05799](https://arxiv.org/abs/1802.05799).

- [O23] Janis Keuper and Franz-Josef Pfreundt. Asynchronous parallel stochastic gradient descent: A numeric core for scalable distributed machine learning algorithms. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 1. ACM, 2015.
- [O24] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI message passing interface standard. In *Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994*, pages 213–218. Springer, 1994.
- [O25] Message Passing Interface Forum. MPI: A message-passing interface standard, 1994.
- [O26] Amos Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.
- [O27] The PapillomaVirus Episteme (PaVE). <https://pave.niaid.nih.gov/>, Accessed: 2024-02-23.

Wykaz osiągnięć naukowych albo artystycznych, stanowiących znaczny wkład w rozwój określonej dyscypliny

Marek Nowicki

26 listopada 2025

I. WYKAZ OSIĄGNIĘĆ NAUKOWYCH ALBO ARTYSTYCZNYCH, O KTÓRYCH MOWA W ART. 219 UST. 1. PKT 2 USTAWY

2. Cykl powiązanych tematycznie artykułów naukowych, zgodnie z art. 219 ust. 1. pkt 2b ustawy

Wszystkie prace wymienione w tej sekcji zostały opublikowane po uzyskaniu stopnia doktora (po 18 czerwca 2015 r.).

- [1] *Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *Journal of Computational Biology*, Mary Ann Liebert, Inc., 2018, ISSN: 1066-5277, Vol. 25 no. 8, pp. 871-881 [**Punktacja ministerialna (poprzednia): 25 / aktualnie: 70**]
- [2] *PCJ - Java Library for Highly Scalable HPC and Big Data Processing (Invited Tutorial Paper)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *2018 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2018, ISBN: 978-1-5386-7878-7, pp. 12-20 [**Punktacja ministerialna: 70**]
- [3] *Comparison of sort algorithms in Hadoop and PCJ*, Marek Nowicki, *Journal of Big Data*, Springer, 2020, ISSN: 2196-1115, Vol. 7 no. 1, Art no. 101, pp. 1-28, DOI: 10.1186/s40537-020-00376-9 [**Punktacja ministerialna: 140**]
- [4] *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Concurrency and Computation: Practice and Experience*, Wiley, 2023, ISSN: 1532-0626, eISSN: 1532-0634, Art. no. 6536 pp. 1-15, DOI: 10.1002/cpe.6536 [**Punktacja ministerialna: 100**]
- [5] *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Journal of Big Data*, Springer, 2021, ISSN: 2196-1115, Vol. 8 no. 1, Art no. 62, pp. 1-21, DOI: 10.1186/s40537-021-00454-6 [**Punktacja ministerialna: 140**]
- [6] *Scalable computing in Java with PCJ library: improved collective operations*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Proceedings of Science – Proceedings of International Symposium on Grids and Clouds 2021 (ISGC2021) - Converging High Performance infrastructures: Supercomputers, clouds, accelerators*, ISSN: 1824-8039, Vol. 378, 2021, Art. no. 007, pp. 1-19, DOI: 10.22323/1.378.0007 [**Punktacja ministerialna: 20**]

- [7] *HPV-KITE: sequence analysis software for rapid HPV genotype detection*, Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, Laila Sara Arroyo Mühr, *Briefings in Bioinformatics*, Volume 26, Issue 2, March 2025, bbaf155, DOI: 10.1093/bib/bbaf155 [**Punktacja ministerialna: 140**]

Opis wkładu merytorycznego w publikacjach wieloautorskich znajduje się w autoreferacie oraz w oświadczeniach autora i współautorów.

II. WYKAZ AKTYWNOŚCI NAUKOWEJ ALBO ARTYSTYCZNEJ

1. Wykaz członkostwa w redakcjach naukowych monografii.

Nie dotyczy.

2. Wykaz wystąpień na krajowych lub międzynarodowych konferencjach naukowych lub artystycznych, z wyszczególnieniem przedstawionych wykładów na zaproszenie i wykładów plenarnych.

- Po uzyskaniu stopnia doktora (po 18 czerwca 2015 r.):
 1. *Scratch – podstawy na przykładzie gry ping-pong* (warsztaty), XXI Informatyka w Edukacji, 1-3/07/2025, Toruń
 2. *HPV-KITE for high-performance metagenomics data analysis* (referat oraz poster), BioInformatics in Toruń 2025; 25-28/06/2025, Toruń
 3. *HPV-KITE for high-performance metagenomics data analysis*, HEAP BYOD#5 workshop; 23-25/06/2025, Graz/Ehrenhausen, Austria
 4. *HPV-KITE for high-performance metagenomics data analysis*, EHEN Scientific Meeting 2024; 26-27/11/2024, Bruksela, Belgia
 5. *HPV-KITE for high-performance metagenomics data analysis*, HEAP BYOD#4 workshop; 19-22/11/2024, Helsinki, Finlandia
 6. *Scratch – podstawy na przykładzie gry wąż* (warsztaty), XX Informatyka w Edukacji, 25-27/06/2024, Toruń
 7. *Czy jesteśmy gotowi na naukę programowania równoległego w szkole?* (referat wspólnie z prof. Piotrem Bałą), XIX Konferencja Informatyka w edukacji (IwE 2023), 22-24/09/2023, Toruń
 8. *Parallel Programming in Java in the PCJ Library* (**wykład na zaproszenie**), TSSP ExSci – Toruń Students Summer Program in Exact Sciences, 14/07/2022, Toruń
 9. *Big Data, processing and storage* (**wykład na zaproszenie**), 4EU+ Against Cancer Summer School, 6/07/2022, Warszawa
 10. *Scratch w przedszkolu* (warsztaty), XVIII Konferencja Informatyka w Edukacji (IwE 2022), 28-30/06/2022, Toruń
 11. *Benchmarking Java on Ethernet Cluster / Running Java/APGAS application on Ethernet Cluster*, 23th HPI Future SOC Lab Day – Fall 2021, 23/11/2021, Poczdam, Niemcy (Online due to COVID-19)

12. *PCJ – Java Library that integrates HPC, Big Data and AI workloads*, International Symposium on Grids & Clouds 2021 (ISGC2021); 22-26/03/2021; Academia Sinica Computing Centre (ASGC), Taipei, Taiwan (Online due to COVID-19)
 13. *Performance Evaluation of Java/PCJ Implementation of Parallel Algorithms on the Cloud*, 26th International European Conference on Parallel and Distributed Computing (Euro-Par 2020), 24-28/08/2020, Warsaw, Poland (Online due to COVID-19)
 14. *Benchmarking Java on Ethernet Cluster / Benchmarking PCJ – a library for parallel computing in Java*, 19th HPI Future SOC Lab Day – Fall 2019, 12/11/2019, Poczdam, Niemcy
 15. *Java i PCJ w HPC*, XX Sesja Sprawozdawcza KDM ICM; 15-16/03/2018; Warszawa, Polska
 16. *High Performance Computing And Big Data Analytics With PCJ And Java (tutorial)*, Supercomputing Frontiers Europe 2018; 12-15/03/2018; Warszawa, Polska
 17. *Evaluation of the parallel performance of the Java and PCJ on the Intel KNL based systems*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM'2017), 10-13/09/2017, Lublin, Poland
 18. *Parallel Computing in Java (tutorial)*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM'2017), 10/09/2017, Lublin, Poland
 19. *Massively Parallel Sequence Alignment with BLAST Through Work Distribution Implemented Using PCJ Library*, 17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2017), 21-23/08/2017, Helsinki, Finlandia
 20. *Level-synchronous BFS Algorithm Implemented in Java Using PCJ Library*, The International Conference on Computational Science and Computational Intelligence (CSCI'16); 15-17/12/2016, Las Vegas, Nevada, USA
 21. *Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ*, II Ogólnopolska Konferencja Innowacyjnych Projektów Badawczych, 2/09/2016
 22. *Tutorial PCJ*, 11th International Conference On Parallel Processing And Applied Mathematics (PPAM'2015), 6-9/09/2015, Kraków
- Przed uzyskaniem stopnia doktora (przed 18 czerwca 2015 r.):
 1. *The performance of the PCJ library for massively parallel computing* (poster), 7th International Conference on PGAS Programming Models, 3-4/10/2013, Edinburgh, Scotland, Great Britain
 2. *PCJ – a Partitioned Global Address Space Approach for Parallel Computations in Java*, Exascale Applications and Software Conference (EASC2013): Solving Software Challenges for Exascale, 9-11/04/2013 r., Edynburg, Szkocja, Wielka Brytania
 3. *Parallel computations in Java with PCJ library*, The 2012 International Conference on High Performance Computing & Simulation (HPCS 2012), 2-6/07/2012, Madrid, Hiszpania
 4. *PCJ – New Approach for Parallel Computations in Java*, PARA 2012: Workshop on State-of-the-Art in Scientific and Parallel Computing, 11-13/06/2012, Helsinki, Finlandia
 5. *PCJ for Parallel Computation in Java* (poster), PRACE Spring School 2012, School for Developers of Petascale Applications, 16-18/05/2012, Kraków
 6. *Short Talk about communication efficiency in ProActive*, Facing the Multicore-Challenge II, 28-30/09/2011, Karlsruhe, Niemcy

- Inne wybrane wykłady i wystąpienia:

1. *The PCJ library as an illustration of the PGAS paradigm of parallel programming in Java*, Seminarium Uniwersyteckiego Centrum Doskonałości: *Dynamika, analiza matematyczna i sztuczna inteligencja*, 22/10/2025, Toruń
2. *Biblioteka PCJ – aplikacje równoległe w Javie?*, 37. spotkanie Toruń JUG, 20/12/2017, Toruń
3. *Biblioteka PCJ – aplikacje równoległe w Javie*, 209. spotkanie Warszawa Java User Group, 23/05/2017, Warszawa
4. *PCJ5* (wykład na zaproszenie), Seminarium zespołu, Cognitive Computing & Computational Sciences Department, IBM Research Lab – Zurich, 14/10/2016, Zürich, Szwajcaria,
5. *Kto najszybciej rozwiąże problem, mając tysiąc procesorów?* (wspólnie z prof. Piotrem Bałą), wykład na XIX Festiwal Nauki Warszawa 2015, 26/09/2015 r., Warszawa
6. *I choćby przyszło tysiąc atletów... czyli o programowaniu równoległym* (wspólnie z prof. Piotrem Bałą), warsztaty na XIX Festiwalu Nauki Warszawa 2015, 26/09/2015, Warszawa
7. *Biblioteka PCJ do naukowych obliczeń równoległych*, 12. spotkanie Toruń JUG, 29/04/2015, Toruń
8. *ZawodyWeb – Serwis do nauki programowania równoległego* (wykład na zaproszenie), ICM UW, 12/02/2015, Warszawa
9. *PCJ – Parallel Computing in Java* (wykład na zaproszenie), Seminarium zespołu, Cognitive Computing & Computational Sciences Department, IBM Research Lab – Zurich, 5/02/2015, Zürich, Szwajcaria,
10. *PCJ – Parallel Computing in Java* (wykład na zaproszenie), Seminarium PJWSTK, Polsko-Japońska Wyższa Szkoła Technik Komputerowych w Warszawie, 7/07/2014, Warszawa
11. *PCJ – Parallel Computing in Java* (wykład na zaproszenie), Seminarium HPC, ICM UW, 22/05/2014, Warszawa
12. *Distributed Computing in Java*, Java Specialists Symposium: JCrete 2014, 25-29/08/2014, Kolimbari, Kreta, Grecja
13. *Distributed Computing in Java – The PCJ library*, Java Specialists Symposium in Crete: JCrete 2013, 19-22/08/2013, Chania, Grecja

3. Wykaz udziału w komitetach organizacyjnych i naukowych konferencji krajowych lub międzynarodowych, z podaniem pełnionej funkcji.

- członek komitetu naukowego warsztatów: Workshop on PGAS Programming, Konferencja PPAM 2019: 13th International Conference on Parallel Processing and Applied Mathematics, 8-11/09/2019, Białystok
- członek komitetu organizacyjnego międzynarodowej konferencji: Petri Nets 2016 // ACS D 2016: 37th International Conference On Applications And Theory Of Petri Nets And Concurrency, 16th International Conference On Application Of Concurrency To System Design, 19-24/06/2016 r., Toruń
- session chairperson: The 2016 International Conference on Computational Science and Computational Intelligence (CSCI'16), 15-17/12/2016, Las Vegas, Nevada, USA

4. Wykaz uczestnictwa w pracach zespołów badawczych realizujących projekty finansowane w drodze konkursów krajowych lub zagranicznych, z podziałem na projekty zrealizowane i będące w toku realizacji, oraz z uwzględnieniem informacji o pełnionej funkcji w ramach prac zespołów.

- Po uzyskaniu stopnia doktora:

1. HEAP (Human Exposome Assessment Platform; finansowany ze środków Unii Europejskiej w ramach programu Horyzont 2020 – Grant agreement ID: 874662; DOI: 10.3030/874662): od 02/2021 – **wykonawca**, 05/2024-06/2025 – **kierownik** (ICM UW; WP9)
 - Budżet projektu: 11 996 515 EUR
 - Partnerzy:
 - * Karolinska Institutet (KI) (Sweden)
 - * Statens Serum Institut (Denmark)
 - * CSC – IT Center for Science (Finland)
 - * Hopsworks (Sweden)
 - * The Medical University of Graz (MUG) (Austria)
 - * University of Innsbruck (UIBK) (Austria)
 - * Lygature (Netherlands)
 - * Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw (Poland)
 - * The Pirkanmaa Hospital District (TAUH) (Finland)
 - * The International Agency for Research on Cancer (IARC) (International Organization)
 - * The University of Oulu (Finland)
2. HPDCJ (Heterogenous Parallel and Distributed Computing with Java; finansowany przez konsorcjum CHIST-ERA – w Polsce finansowane przez NCN: 2014/14/Z/ST6/0007): 10/2014-09/2017 – *technical coordinator of the work package* (**wykonawca**)
 - Budżet projektu: 970 389 EUR
 - Partnerzy:
 - * Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw (Poland)
 - * IBM Research Lab (Zurich, Switzerland)
 - * Queen’s University of Belfast (Belfast, UK)
 - * Bilkent Üniversitesi (Ankara, Turkey)

- Przed uzyskaniem stopnia doktora:

1. udział w projekcie APOS (Application Performance Optimization and Scalability; EU 7th Project Framework): 2011-2013 – programista
2. udział w projekcie Kardionet (EEA Funds): 2008-2011 – programista

5. Wykaz członkostwa w międzynarodowych lub krajowych organizacjach i towarzystwach naukowych wraz z informacją o pełnionych funkcjach.

- Association for Computing Machinery (ACM) – członek w latach 2009-2015 (ACM Student Member).

6. Wykaz staży w instytucjach naukowych lub artystycznych, w tym zagranicznych, z podaniem miejsca, terminu, czasu trwania stażu i jego charakteru.

- Po uzyskaniu stopnia doktora:

1. 06/2017-07/2017: ośrodek superkomputerowy HLRS – High Performance Computing Center Stuttgart, University of Stuttgart, Niemcy; w ramach Cross-Site Collaboration Grant projektu EuroLab-4-HPC

- Przed uzyskaniem stopnia doktora:

1. 05/2013-11/2013: laboratorium badawcze IBM Zurich Research Laboratory w Zurychu, w Szwajcarii (Mathematical & Computational Science Department) - student internship
2. 10/2011-03/2012: Laboratorium Maszyn Biomolekularnych Uniwersytetu Warszawskiego - doktorant

7. Wykaz członkostwa w komitetach redakcyjnych i radach naukowych czasopism wraz z informacją o pełnionych funkcjach (np. redaktora naczelnego, przewodniczącego rady naukowej, itp.).

- recenzowanie prac naukowych dla czasopism, m.in.: *Bioinformatics* (Oxford University Press), *BMC Bioinformatics* (Springer Nature), *Journal of Big Data* (Springer Nature), *SoftwareX* (Elsevier)
- recenzowanie prac naukowych na konferencje: *Petri Nets 2020*, *PPAM 2022*

8. Wykaz recenzowanych prac naukowych lub artystycznych, w szczególności publikowanych w czasopismach międzynarodowych.

- Prace opublikowane po uzyskaniu stopnia doktora (wymienione w sekcji I.2):

1. *HPV-KITE: sequence analysis software for rapid HPV genotype detection*, Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, Laila Sara Arroyo Mühr, *Briefings in Bioinformatics*, Volume 26, Issue 2, March 2025, bbaf155, DOI: 10.1093/bib/bbaf155 [**Punktacja ministerialna: 140**]
2. *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Concurrency and Computation: Practice and Experience*, Wiley, 2021/2023, ISSN: 1532-0626, eISSN: 1532-0634, Art. no. 6536 pp. 1-15, DOI: 10.1002/cpe.6536 [**Punktacja ministerialna: 100**]
3. *Scalable computing in Java with PCJ library: improved collective operations*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Proceedings of Science – Proceedings of International Symposium on Grids and Clouds 2021 (ISGC2021) - Converging High Performance infrastructures: Supercomputers, clouds, accelerators*, ISSN: 1824-8039, Vol. 378, 2021, Art. no. 007, pp. 1-19, DOI: 10.22323/1.378.0007 [Punktacja ministerialna: 20]
4. *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Journal of Big Data*, Springer, 2021, ISSN: 2196-1115, Vol. 8 no. 1, Art no. 62, pp. 1-21, DOI: 10.1186/s40537-021-00454-6 [**Punktacja ministerialna: 140**]

5. *Comparison of sort algorithms in Hadoop and PCJ*, Marek Nowicki, *Journal of Big Data*, Springer, 2020, ISSN: 2196-1115, Vol. 7 no. 1, Art no. 101, pp. 1-28, DOI: 10.1186/s40537-020-00376-9 [**Punktacja ministerialna: 140**]
 6. *PCJ – Java Library for Highly Scalable HPC and Big Data Processing*, Marek Nowicki, Łukasz Górski, Piotr Bała, *2018 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2018, ISBN: 978-1-5386-7878-7, pp. 12-20 [**Punktacja ministerialna: 70**]
 7. *Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *Journal of Computational Biology*, Mary Ann Liebert, Inc., 2018, ISSN: 1066-5277, Vol. 25 no. 8, pp. 871-881 [**Punktacja ministerialna (poprzednia): 25 / aktualnie: 70**]
- Prace opublikowane po uzyskaniu stopnia doktora (niewymienione w sekcji I.2):
8. *Słownikowe struktury danych i ich zastosowania*, Marcin Piątkowski, Magdalena Mroczek, Marek Nowicki, *Informatyka w edukacji : Edukacja informatyczna a kompetencje cyfrowe XXI wieku* / Kwiatkowska Anna Beata (red.), 2025, Wydawnictwo Adam Marszałek, ISBN: 978-83-8180-347-2, pp. 80-90 [Punktacja ministerialna: 20]
 9. *Scratch : podstawy na przykładzie gry Wąż*, Marek Nowicki, *Informatyka w edukacji : kształcenie informatyczne drogą do cyfrowej transformacji społeczeństwa* / Kwiatkowska Anna Beata, Sysło Maciej (red.), 2024, Wydawnictwo Adam Marszałek, ISBN: 978-83-8180-904-7, pp. 284-294 [Punktacja ministerialna: 20]
 10. *Czy jesteśmy gotowi na naukę programowania równoległego w szkole?*, Piotr Bała, Marek Nowicki, *Informatyka w edukacji : edukacja informatyczna : wyzwania współczesnego świata*, Wydawnictwo Adam Marszałek, 2023, Toruń, pp. 127-136, ISBN: 978-83-8180-810-1 [Punktacja ministerialna: 20]
 11. *Scratch w przedszkolu*, Marek Nowicki, *Informatyka w edukacji : uczniowie i nauczyciele w szkole przyszłości*, Marek Nowicki, Wydawnictwo Adam Marszałek, Toruń, 2022, s.414-424, ISBN: 978-83-8180-645-9 [Punktacja ministerialna: 20]
 12. *Performance Evaluation of Java/PCJ Implementation of Parallel Algorithms on the Cloud*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Euro-Par 2020: Parallel Processing Workshops. Euro-Par 2020*, Springer, Cham, 2021, ISBN: 978-3-030-71592-2, pp. 213-224 [**Punktacja ministerialna: 140**] – rozszerzona wersja pracy została wymieniona w sekcji I.2.
 13. *Performance evaluation of parallel computing and Big Data processing with Java and PCJ library*, Marek Nowicki, Łukasz Górski, Piotr Bała, *CUG2018 - Visionary Computing*, Cray User Group 2018, 21-24 May 2018, Stockholm, Sweden, 2018
 14. *Java w szkole*, Piotr Bała, Łukasz Górski, Marek Nowicki, *Informatyka w edukacji, myśl komputacyjnie!*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, Toruń, 2018, ISBN: 978-83-231-4049-8, pp. 171-172
 15. *Scratch 3.0: nowe możliwości*, Piotr Bała, Marek Nowicki, *Informatyka w edukacji, myśl komputacyjnie!*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, Toruń, 2018, ISBN: 978-83-231-4049-8, pp. 173-174
 16. *Fault-Tolerance Mechanisms for the Java Parallel Codes Implemented with the PCJ Library*, Michał Szynkiewicz, Marek Nowicki, *12th International Conference on Parallel Processing*

- and *Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 298-307 [Punktacja ministerialna: 20]
17. *Big Data Analytics in Java with PCJ Library: Performance Comparison with Hadoop*, Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, Piotr Bała, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 318-327 [Punktacja ministerialna: 20]
 18. *Evaluation of the Parallel Performance of the Java and PCJ on the Intel KNL Based Systems*, Marek Nowicki, Łukasz Górski, Piotr Bała, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 288-297 [Punktacja ministerialna: 20]
 19. *Performance Comparison of Graph BFS Implemented in MapReduce and PGAS Programming Models*, Magdalena Ryczkowska, Marek Nowicki, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 328-337 [Punktacja ministerialna: 20]
 20. *Massively Parallel Sequence Alignment with BLAST Through Work Distribution Implemented Using PCJ Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2017)*, Springer, Cham, ISBN: 978-3-319-65481-2, pp. 503-512 [Punktacja ministerialna: 70]
– rozszerzona wersja pracy została wymieniona w sekcji I.2.
 21. *Level-synchronous BFS algorithm implemented in Java using PCJ Library*, Magdalena Ryczkowska, Marek Nowicki, Piotr Bała, *2016 International Conference on Computational Science and Computational Intelligence (CSCI 2016)*, 15-17 December 2016, Las Vegas, Nevada, USA: proceedings, Piscataway, NJ : IEEE, 2016, ISBN: 978-1-5090-5510-4, pp. 596-601 [Punktacja ministerialna (poprzednia): 15]
 22. *Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ*, Marek Nowicki, Piotr Bała, *Innowacje w polskiej nauce w obszarze matematyki i informatyki. Przegląd aktualnej tematyki badawczej*, 2016, Wydawnictwo Nauka i Biznes, 2016, ISBN: 978-83-947095-2-5, pp. 130-140
 23. *PCJ – a Java library for heterogenous parallel computing*, Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, Michał Szynkiewicz, Piotr Bała, *15th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS16)*, Barcelona, Spain, February 13-15, 2016, ISBN: 978-1-61804-365-8, pp. 66-72
 24. *The performance evaluation of the Java implementation of Graph500*, Magdalena Ryczkowska, Marek Nowicki, Piotr Bała, *PPAM2015: 11th International Conference On Parallel Processing And Applied Mathematics*, September 6-9, 2015, Kraków, Poland, LNCS Series, ISBN: 978-3-319-32151-6, pp. 221-230 [Punktacja ministerialna (poprzednia): 15]
 25. *On-line service for teaching parallel programming*, Marek Nowicki, Maciej Marchwiany, Maciej Szpindler, Piotr Bała, *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops*, Vienna, Austria, August 24-25, 2015: revised selected papers, ISBN: 978-3-319-27307-5, pp. 78-89 [Punktacja ministerialna (poprzednia): 15]
- Prace opublikowane przed uzyskaniem stopnia doktora (niewymienione w sekcji I.2):
26. *PCJ – Java library for high performance computing in PGAS model*, Marek Nowicki, Łukasz Górski, Patryk Grabarczyk, Piotr Bała, *Proceedings of the 2014 International Conference*

- on *High Performance Computing & Simulation (HPCS 2014)*, ISBN: 978-1-4799-5311-0, pp. 202-209
27. *The performance of the PCJ library for massively parallel computing*, Marek Nowicki, Piotr Bała, *7th International Conference on PGAS Programming Models*, The University of Edinburgh, 2013, ISBN: 978-0-9926615-0-2, pp. 254
 28. *On-line contests hosting service as a tool to teach computer science students programming*, Marek Nowicki, Łukasz Mikulski, Marcin Piątkowski, Rafał Kluszczyński, Piotr Bała, *10th World Conference on Computers in Education (WCCE 2013)*, 2-5 lipca 2013 r., Toruń, ISBN: 978-83-231-3093-2, pp. 61-71
 29. *Teaching secondary school students programming using distance learning. A case study*, Marek Nowicki, Michał Matuszak, Anna B. Kwiatkowska, Maciej M. Sysło, Piotr Bała, *10th World Conference on Computers in Education (WCCE 2013)*, 2-5 lipca 2013 r., Toruń, ISBN: 978-83-231-3093-2, pp. 246-254
 30. *PCJ – New Approach for Parallel Computations in Java*, Marek Nowicki, Piotr Bała, *PARA 2012 Proceedings*, Springer Lecture Notes in Computer Science, Vol. 7782, ISBN: 978-3-642-36802-8, pp. 115-125
 31. *Parallel computations in Java with PCJ library*, Marek Nowicki, Piotr Bała, *Proceedings of the 2012 International Conference on High Performance Computing & Simulation: HPCS 2012*, July 2 - July 6 2012, Madrid, Spain, ISBN: 978-1-4673-2359-8, pp. 381-387
 32. *Scratch – środowisko do nauki programowania*, Piotr Bała, Marek Nowicki, *Informatyka w edukacji: materiały z warsztatów VII Konferencji – IwE 2010*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011, ISBN: 978-83-23126-77-5, pp. 33-40
 33. *Regionalne Koło Informatyczne*, Piotr Bała, Anna Beata Kwiatkowska, Marek Nowicki, Mariusz Strzelecki, Maciej M. Sysło, *Informatyka w edukacji: materiały z warsztatów VII Konferencji – IwE 2010*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011, ISBN: 978-83-23126-77-5, pp. 61-75
 34. *Contests Hosting Service as a tool to teach programming*, Rafał Kluszczyński, Łukasz Mikulski, Marek Nowicki, Piotr Bała, *Annales UMCS Informatica*, Vol. 9/2009, ISSN:1732-1360, pp. 213-224, DOI: 10.2478/v10065-009-0017-9
 35. *Internetowe narzędzia do sprawdzania zadań programistycznych i ich zastosowania*, Marek Nowicki, Rafał Kluszczyński, Piotr Bała, *V Konferencja "Informatyka w Edukacji" – IwE 2008*, Toruń, 3-6 lipca 2008, Agencja Wydawnicza Argi, Wrocław, 2008, ISBN: 978-83-60425-30-5, pp. 88-94
- Inne publikacje w formie raportów technicznych i streszczeń:
36. *Benchmarking the TeraSort algorithm on Ethernet Cluster : Benchmarking Apache Hadoop based implementation*, Marek Nowicki, *HPI Future SOC Lab : proceedings 2020 / Meinel Christoph [i in.] (red.)*, Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 159, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-565-1, pp. 97-108
 37. *Benchmarking the sort algorithm on Ethernet Cluster : Benchmarking PCJ : a library for parallel computing in Java*, Marek Nowicki, *HPI Future SOC Lab : proceedings 2020 / Meinel Christoph [i in.] (red.)*, Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 159, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-565-1, pp. 53-63

38. *Benchmarking Java on Ethernet Cluster: benchmarking PCJ – a library for parallel computing in Java*, Marek Nowicki, *HPI Future SOC Lab: proceedings 2019 / Meinel Christoph* [i in.] (red.), Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 158, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-564-4, pp. 263-271
39. *Benchmarking FFT on an Ethernet Cluster: Latency, Bandwidth and MPI benchmarks*, Marek Nowicki, Benson K. Muite, Boysan Mehmet Can, *HPI Future SOC Lab: proceedings 2019 / Meinel Christoph* [i in.] (red.), Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 158, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-564-4, pp. 93-98
40. *High performance AI based tool for mining and classification of viral genomes in human exposome metadata (Abstracts from the 55th European Society of Human Genetics (ESHG) Conference: e-Posters)*, Magdalena Mroczek, Marek Nowicki, Łukasz Górski, Sara Arroyo Mühr, Alex Ormenisan, Piotr Bała, *European Journal of Human Genetics*, 31, 292 (2023)
41. *PCJ as a tool for massively parallel data processing*, Marek Nowicki, Łukasz Górski, Magdalena Ryczkowska, Piotr Bała, *CGW Workshop'17*, Kraków, Poland, October 23-25, 2017, proceedings, Academic Computer Centre Cyfronet. AGH, Kraków, 2017, ISBN: 978-83-61433-24-8, pp. 39-40
42. *Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ*, Marek Nowicki, Piotr Bała, *Materiały konferencyjne - Innowacyjne projekty badawcze: II Ogólnopolska e-Konferencja Innowacyjnych Projektów Badawczych IPB*, Wrocław, 2 września 2016., Wrocław: Dolnośląski Akcelerator Technologii Innowacji Sp. z o.o., 2016, ISBN: 978-83-945037-2-7, pp. 30
43. *Development of parallel codes using PL-Grid infrastructure*, Rafał Kluszczyński, Marcin Stolarek, Grzegorz Marczak, Łukasz Górski, Marek Nowicki, Piotr Bała, *Cracow '14 Grid Workshop*, October 27-29 2014, Krakow, Poland, pp. 81-82
44. *Komunikacja wątków w bibliotece PCJ*, Łukasz Górski, Marek Nowicki, Piotr Bała, *XXVII Forum Informatyki Teoretycznej*, Toruń, 11-14 kwietnia 2013., Toruń, 2013, pp. 28

9. Wykaz uczestnictwa w programach europejskich lub innych programach międzynarodowych.

- *The PCJ library on HLRS*, Cross-Site Collaboration Grant, EuroLab-4-HPC: Foundations of a European Research Center of Excellence in High Performance Computing Systems (2017)

10. Wykaz udziału w zespołach badawczych, realizujących projekty inne niż określone w pkt. II.4.

- członek priorytetowego zespołu Informatyka – języki formalne i współbieżność (FoLCo) w ramach Uniwersyteckiego Centrum Doskonałości *Dynamika, analiza matematyczna i sztuczna inteligencja* (Inicjatywa Doskonałości – Uczelnia Badawcza, Uniwersytet Mikołaja Kopernika w Toruniu) - od 01/2020
- **kierownik** grantów obliczeniowych *Benchmarking Java on Ethernet Cluster*, HPI Future SOC Lab, Hasso Plattner Institut, Potsdam, Niemcy (04/2019-05/2024)

- **kierownik** grantu obliczeniowego *Dalszy rozwój metod programowania w modelu PGAS w języku Java (biblioteka PCJ)*, Trzeci konkurs na dostęp do superkomputera Okeanos (OKEANOS 3), GA69-19, Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego, Uniwersytet Warszawski (07/2017-07/2019)
- **kierownik** grantu obliczeniowego *Scalability of PCJ applications on HLRS*, Preparatory type A – Code scalability testing, PRACE - Partnership for Advanced Computing in Europe / HLRS, University of Stuttgart, Germany (06/2017-07/2017)
- **kierownik** grantu obliczeniowego *Rozwój metod programowania w modelu PGAS w języku Java (biblioteka PCJ)*, Pierwszy konkurs na dostęp do superkomputera Okeanos, GB65-15, Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego, Uniwersytet Warszawski (06/2016-06/2017)

11. Wykaz uczestnictwa w zespołach oceniających wnioski o finansowanie badań, wnioski o przyznanie nagród naukowych, wnioski w innych konkursach mających charakter naukowy lub dydaktyczny.

- członek komisji konkursowej *Grants4NCUStudents* IV edycja (2022 r.) na zadania badawcze i wyjazdy zagraniczne realizowane przez studentów i doktorantów Uniwersytetu Mikołaja Kopernika w Toruniu – panel ST

III. WSPÓŁPRACA Z OTOCZENIEM SPOŁECZNYM I GOSPODARCZYM

1. Wykaz dorobku technologicznego.

Nie dotyczy.

2. Współpraca z sektorem gospodarczym.

- **współzałożyciel** i jeden z liderów *Toruń JUG* – toruńskiej grupy zrzeszającej pasjonatów technologii opartych o *Java Virtual Machine* (02/2014-12/2024)

3. Wykaz uzyskanych praw własności przemysłowej, w tym uzyskanych patentów krajowych lub międzynarodowych.

Nie dotyczy.

4. Wykaz wdrożonych technologii.

- system *ZawodyWeb* – wdrożony m.in. na *Wydziale Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika w Toruniu*
- biblioteka *PCJ* – dostępna w centralnym repozytorium Maven, wdrażana m.in. w *international ImMunoGenetics information system*
- aplikacja *PCJ-BLAST* – dostępna w repozytorium *GitHub*
- aplikacja *HPV-KITE* – dostępna m.in. w repozytorium *Conda-Forge*

- aplikacja *KITE* – dostępna m.in. w repozytorium *Conda-Forge*
- wewnętrzny system wsparcia decyzji sprzedażowych **IBM** (IBM wide Sales Decision Support) wykorzystywany był w 2013 roku przez ponad 5000 sprzedawców IBM na całym świecie

5. Wykaz wykonanych ekspertyz lub innych opracowań wykonanych na zamówienie instytucji publicznych lub przedsiębiorców.

Nie dotyczy.

6. Wykaz udziału w zespołach eksperckich lub konkursowych.

- członek Komitetu Okręgowego w Toruniu *Olimpiady Informatycznej* (od 01/2014)
- członek jury hackathonu *BrainCode 2019* organizowanego przez Allegro, 12-13/04/2019, Toruń
- **przewodniczący** komisji konkursowej: *Hackaton Wielkie Wyzwania Programistyczne* – edycja III: *Switch to Smart*, ICM UW, 23-24/11/2018, Warszawa
- członek Jury hackathonu *BrainCode 2018* organizowanego przez Allegro, 20-21/04/2018, Toruń
- członek komisji sędziowskiej konkursu *FRI: III Kujawsko-Pomorskich Zawodów Programistycznych*, 20/04/2018, Toruń
- **przewodniczący** komisji konkursowej: *Hackaton Wielkie Wyzwania Programistyczne* – edycja II: *Smart Charge*, ICM UW, 15-16/02/2018, Warszawa
- członek jury hackathonu *BrainCode 2017* organizowanego przez Allegro, 23-24/03/2017, Toruń
- członek komisji sędziowskiej konkursu *FRI: II Kujawsko-Pomorskich Zawodów Programistycznych*, 24/03/2017, Toruń
- członek komisji konkursowej: Konkurs *Kodowanie ze Scratchem 2017* – dla uczniów szkół podstawowych, 26/05/2017, Toruń
- **przewodniczący** komisji konkursowej: *Hackaton Wielkie Wyzwania Programistyczne* – edycja I: *Map the Gap*, ICM UW, 16-17/02/2017, Warszawa
- członek komisji konkursowej: Konkurs *Kodowanie ze Scratchem 2016* – dla uczniów szkół podstawowych, 17/06/2016, Toruń
- członek komisji konkursowej: Konkurs *Kodowanie ze Scratchem 2015* – dla uczniów szkół podstawowych, 19/06/2015, Toruń
- pomoc w przeprowadzaniu *Konkursu Informatycznego* (edycje: 2006/2007, 2007/2008, 2008/2009) na Wydziale Matematyki i Informatyki UMK
- pomoc w przeprowadzeniu *mini konkursu programistycznego* w ramach *24. Toruńskiego Festiwalu Nauki i Sztuki* w kwietniu 2006 r.

7. Wykaz projektów artystycznych realizowanych ze środowiskami pozaartystycznymi.

Nie dotyczy.

IV. DANE NAUKOMETRYCZNE

Stan danych naukometrycznych na: 26 listopada 2025 r.

1. Impact Factor (w dziedzinach i dyscyplinach, w których parametr ten jest powszechnie używany jako wskaźnik naukometryczny)

Dane z *Bazy Wiedzy Omega* prowadzonej przez Uniwersytet Mikołaja Kopernika w Toruniu:

- Sumaryczny IF: 20,914
- Sumaryczna punktacja ministerialna: 1100

2. Liczba cytowań publikacji wnioskodawcy, z oddzielnym uwzględnieniem autocytowań

- Scopus: 124 (62 bez autocytowań)
- Web of Science: 85 (46 bez autocytowań)
- Google Scholar: 249

3. Indeks Hirscha

- Scopus: 7
- Web of Science: 5
- Google Scholar: 10



podpis wnioskodawcy

List of scientific achievements constituting a significant contribution to the
development of a specific discipline

Marek Nowicki

November 26, 2025

**I. LIST OF SCIENTIFIC OR ARTISTIC ACHIEVEMENTS, REFERRED TO IN
ARTICLE 219, SECTION 1. POINT 2 OF THE ACT**

**2. A cycle of thematically related scientific articles, in accordance with Article 219,
Section 1. Point 2b of the Act**

All works listed in this section were published after the doctoral degree had been awarded (after 18 June 2015).

- [1] *Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *Journal of Computational Biology*, Mary Ann Liebert, Inc., 2018, ISSN: 1066-5277, Vol. 25 no. 8, pp. 871-881 [Ministerial score (previous): 25 / current: 70]
- [2] *PCJ - Java Library for Highly Scalable HPC and Big Data Processing (Invited Tutorial Paper)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *2018 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2018, ISBN: 978-1-5386-7878-7, pp. 12-20 [Ministerial score: 70]
- [3] *Comparison of sort algorithms in Hadoop and PCJ*, Marek Nowicki, *Journal of Big Data*, Springer, 2020, ISSN: 2196-1115, Vol. 7 no. 1, Art no. 101, pp. 1-28, DOI: 10.1186/s40537-020-00376-9 [Ministerial score: 140]
- [4] *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Concurrency and Computation: Practice and Experience*, Wiley, 2023, ISSN: 1532-0626, eISSN: 1532-0634, Art. no. 6536 pp. 1-15, DOI: 10.1002/cpe.6536 [Ministerial score: 100]
- [5] *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Journal of Big Data*, Springer, 2021, ISSN: 2196-1115, Vol. 8 no. 1, Art no. 62, pp. 1-21, DOI: 10.1186/s40537-021-00454-6 [Ministerial score: 140]
- [6] *Scalable computing in Java with PCJ library: improved collective operations*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Proceedings of Science – Proceedings of International Symposium on Grids and Clouds 2021 (ISGC2021) - Converging High Performance infrastructures: Supercomputers, clouds, accelerators*, ISSN: 1824-8039, Vol. 378, 2021, Art. no. 007, pp. 1-19, DOI: 10.22323/1.378.0007 [Ministerial score: 20]

- [7] *HPV-KITE: sequence analysis software for rapid HPV genotype detection*, Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, Laila Sara Arroyo Mühr, *Briefings in Bioinformatics*, Volume 26, Issue 2, March 2025, bbaf155, DOI: 10.1093/bib/bbaf155 [Ministerial score: 140]

A description of the substantive contribution to the co-authored publications is provided in the *Summary of Professional Accomplishments* and in the statements of the author and co-authors.

II. LIST OF SCIENTIFIC OR ARTISTIC ACTIVITY

1. List of membership in scientific editorial boards of monographs.

Not applicable.

2. List of speeches at national or international scientific or artistic conferences, with specification of invited lectures and plenary lectures.

- After the doctoral degree had been awarded (after 18 June 2015):
 1. *Scratch – podstawy na przykładzie gry ping-pong* (hands-on workshop), XXI Informatyka w Edukacji (IwE 2025), 1-3/07/2025, Toruń, Poland
 2. *HPV-KITE for high-performance metagenomics data analysis* (talk and poster), BioInformatics in Toruń 2025; 25-28/06/2025, Toruń, Poland
 3. *HPV-KITE for high-performance metagenomics data analysis*, HEAP BYOD#5 workshop; 23-25/06/2025, Graz/Ehrenhausen, Austria
 4. *HPV-KITE for high-performance metagenomics data analysis*, EHEN Scientific Meeting 2024; 26-27/11/2024, Brussels, Belgium
 5. *HPV-KITE for high-performance metagenomics data analysis*, HEAP BYOD#4 workshop; 19-22/11/2024, Helsinki, Finland
 6. *Scratch – podstawy na przykładzie gry wąż* (hands-on workshop), XX Informatyka w Edukacji (IwE 2024), 25-27/06/2024, Toruń, Poland
 7. *Czy jesteśmy gotowi na naukę programowania równoległego w szkole?* (joint presentation with prof. Piotr Bała), XIX Konferencja Informatyka w edukacji (IwE 2023), 22-24/09/2023, Toruń, Poland
 8. *Parallel Programming in Java in the PCJ Library (invited lecture)*, TSSP ExSci – Toruń Students Summer Program in Exact Sciences, 14/07/2022, Toruń, Poland
 9. *Big Data, processing and storage (invited lecture)*, 4EU+ Against Cancer Summer School, 6/07/2022, Warsaw, Poland
 10. *Scratch w przedszkolu* (hands-on workshop), XVIII Konferencja Informatyka w Edukacji (IwE 2022), 28-30/06/2022, Toruń, Poland
 11. *Benchmarking Java on Ethernet Cluster / Running Java/APGAS application on Ethernet Cluster*, 23th HPI Future SOC Lab Day – Fall 2021, 23/11/2021, Potsdam, Germany (Online due to COVID-19)
 12. *PCJ – Java Library that integrates HPC, Big Data and AI workloads*, International Symposium on Grids & Clouds 2021 (ISGC2021); 22-26/03/2021; Academia Sinica Computing Centre (ASGC), Taipei, Taiwan (Online due to COVID-19)

13. *Performance Evaluation of Java/PCJ Implementation of Parallel Algorithms on the Cloud*, 26th International European Conference on Parallel and Distributed Computing (Euro-Par 2020), 24-28/08/2020, Warsaw, Poland (Online due to COVID-19)
 14. *Benchmarking Java on Ethernet Cluster / Benchmarking PCJ – a library for parallel computing in Java*, 19th HPI Future SOC Lab Day – Fall 2019, 12/11/2019, Potsdam, Germany
 15. *Java i PCJ w HPC*, XX Sesja Sprawozdawcza KDM ICM; 15-16/03/2018; Warsaw, Poland
 16. *High Performance Computing And Big Data Analytics With PCJ And Java (tutorial)*, Supercomputing Frontiers Europe 2018; 12-15/03/2018; Warsaw, Poland
 17. *Evaluation of the parallel performance of the Java and PCJ on the Intel KNL based systems*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM'2017), 10-13/09/2017, Lublin, Poland
 18. *Parallel Computing in Java (tutorial)*, 12th International Conference On Parallel Processing And Applied Mathematics (PPAM'2017), 10/09/2017, Lublin, Poland
 19. *Massively Parallel Sequence Alignment with BLAST Through Work Distribution Implemented Using PCJ Library*, 17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2017), 21-23/08/2017, Helsinki, Finland
 20. *Level-synchronous BFS Algorithm Implemented in Java Using PCJ Library*, The International Conference on Computational Science and Computational Intelligence (CSCI'16); 15-17/12/2016, Las Vegas, Nevada, USA
 21. *Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ*, II Ogólnopolska Konferencja Innowacyjnych Projektów Badawczych, 2/09/2016
 22. *Tutorial PCJ*, 11th International Conference On Parallel Processing And Applied Mathematics (PPAM'2015), 6-9/09/2015, Cracow, Poland
- Before the doctoral degree had been awarded (before 18 June 2015):
 1. *The performance of the PCJ library for massively parallel computing* (poster), 7th International Conference on PGAS Programming Models, 3-4/10/2013, Edinburgh, Scotland, Great Britain
 2. *PCJ – a Partitioned Global Address Space Approach for Parallel Computations in Java*, Exascale Applications and Software Conference (EASC2013): Solving Software Challenges for Exascale, 9-11/04/2013 r., Edinburgh, Scotland, Great Britain
 3. *Parallel computations in Java with PCJ library*, The 2012 International Conference on High Performance Computing & Simulation (HPCS 2012), 2-6/07/2012, Madrid, Spain
 4. *PCJ – New Approach for Parallel Computations in Java*, PARA 2012: Workshop on State-of-the-Art in Scientific and Parallel Computing, 11-13/06/2012, Helsinki, Finland
 5. *PCJ for Parallel Computation in Java* (poster), PRACE Spring School 2012, School for Developers of Petascale Applications, 16-18/05/2012, Cracow, Poland
 6. *Short Talk about communication efficiency in ProActive*, Facing the Multicore-Challenge II, 28-30/09/2011, Karlsruhe, Germany
 - Other selected lectures and presentations:
 1. *The PCJ library as an illustration of the PGAS paradigm of parallel programming in Java*, Seminar of University Centre of Excellence: *Dynamics, Mathematical Analysis and Artificial Intelligence*, 22/10/2025, Toruń, Poland

2. *Biblioteka PCJ – aplikacje równoległe w Javie?*, 37. Toruń JUG meeting, 20/12/2017, Toruń, Poland
3. *Biblioteka PCJ – aplikacje równoległe w Javie*, 209. Warszawa Java User Group meeting, 23/05/2017, Warsaw, Poland
4. *PCJ5* (invited lecture), Team seminar, Cognitive Computing & Computational Sciences Department, IBM Research Lab – Zurich, 14/10/2016, Zürich, Switzerland
5. *Kto najszybciej rozwiąże problem, mając tysiąc procesorów?* (together with Prof. Piotr Bala), lecture at the 19th Warsaw Science Festival 2015, 26 September 2015, Warsaw, Poland
6. *I choćby przyszło tysiąc atletów... czyli o programowaniu równoległym* (together with Prof. Piotr Bala), workshop at the 19th Warsaw Science Festival 2015, 26 September 2015, Warsaw, Poland
7. *Biblioteka PCJ do naukowych obliczeń równoległych*, 12. Toruń JUG meeting, 29/04/2015, Toruń, Poland
8. *ZawodyWeb – Serwis do nauki programowania równoległego* (invited lecture), ICM UW, 12/02/2015, Warsaw, Poland
9. *PCJ – Parallel Computing in Java* (invited lecture), Team seminar, Cognitive Computing & Computational Sciences Department, IBM Research Lab – Zurich, 5/02/2015, Zürich, Switzerland,
10. *PCJ – Parallel Computing in Java* (invited lecture), PJWSTK Seminar, Polsko-Japońska Wyższa Szkoła Technik Komputerowych w Warszawie, 7/07/2014, Warsaw, Poland
11. *PCJ – Parallel Computing in Java* (invited lecture), HPC Seminar, ICM UW, 22/05/2014, Warsaw, Poland
12. *Distributed Computing in Java*, Java Specialists Symposium: JCrete 2014, 25-29/08/2014, Kolimbari, Crete, Greece
13. *Distributed Computing in Java – The PCJ library*, Java Specialists Symposium in Crete: JCrete 2013, 19-22/08/2013, Chania, Greece

3. List of participation in organizational and scientific committees of national or international conferences, with specifying the function performed.

- member of the scientific committee of the workshop: Workshop on PGAS Programming, 13th International Conference on Parallel Processing and Applied Mathematics (PPAM 2019), 8-11/09/2019, Białystok, Poland
- member of the organizing committee of the international conference: Petri Nets 2016 // ACSN 2016: 37th International Conference On Applications And Theory Of Petri Nets And Concurrency, 16th International Conference On Application Of Concurrency To System Design, 19-24/06/2016 r., Toruń, Poland
- session chairperson: The 2016 International Conference on Computational Science and Computational Intelligence (CSCI'16), 15-17/12/2016, Las Vegas, Nevada, USA

4. List of participation in works of research teams implementing projects financed by way of national or international competitions, with division into completed projects and projects in the course of performance and with taking into account the information on the function performed within the framework of works of the teams.

- After the doctoral degree had been awarded:

1. HEAP (Human Exposome Assessment Platform; funded by the European Union under Horizon 2020 program – Grant agreement ID: 874662; DOI: 10.3030/874662): from 02/2021 – **researcher**, 05/2024-06/2025 – **work package leader** (ICM UW; WP9)

- Project budget: 11 996 515 EUR

- Partners:

- * Karolinska Institutet (KI) (Sweden)
- * Statens Serum Institut (Denmark)
- * CSC – IT Center for Science (Finland)
- * Hopsworks (Sweden)
- * The Medical University of Graz (MUG) (Austria)
- * University of Innsbruck (UIBK) (Austria)
- * Lygature (Netherlands)
- * Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw (Poland)
- * The Pirkanmaa Hospital District (TAUH) (Finland)
- * The International Agency for Research on Cancer (IARC) (International Organization)
- * The University of Oulu (Finland)

2. HPDCJ (Heterogenous Parallel and Distributed Computing with Java; funded by the CHIST-ERA consortium – in Poland funded by the National Science Centre (NCN): 2014/14/Z/ST6/0007): 10/2014-09/2017 – *technical coordinator of the work package (researcher)*

- Project budget: 970 389 EUR

- Partners:

- * Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw (Poland)
- * IBM Research Lab (Zurich, Switzerland)
- * Queen’s University of Belfast (Belfast, UK)
- * Bilkent Üniversitesi (Ankara, Turkey)

- Before the doctoral degree had been awarded:

1. involvement in the project APOS (Application Performance Optimization and Scalability; EU 7th Project Framework): 2011-2013 – programmer
2. involvement in the project Kardionet (EEA Funds): 2008-2011 – programmer

5. **List of membership in international or national organizations and scientific societies, together with information on the functions performed.**
 - Association for Computing Machinery (ACM) – member from 2009 to 2015 (ACM Student Member).
6. **List of internships at scientific or artistic institutions, including foreign ones, with specifying the place, date, duration of the internship and its nature.**
 - After the doctoral degree had been awarded:
 1. 06/2017–07/2017: HLRS – High Performance Computing Center Stuttgart, University of Stuttgart, Germany; as part of the Cross-Site Collaboration Grant of the EuroLab-4-HPC project
 - Before the doctoral degree had been awarded:
 1. 05/2013–11/2013: IBM Zurich Research Laboratory, Zurich, Switzerland (Mathematical & Computational Science Department) – student internship
 2. 10/2011–03/2012: Biomolecular Machines Laboratory, University of Warsaw – PhD student
7. **List of membership in editorial committees and scientific councils of journals, along with information on the functions performed (e.g. editor-in-chief, chairman of the scientific council, etc.).**
 - peer reviewing of scientific papers for journals, i.a.: *Bioinformatics* (Oxford University Press), *BMC Bioinformatics* (Springer Nature), *Journal of Big Data* (Springer Nature), *SoftwareX* (Elsevier)
 - peer reviewing of scientific papers for conferences: *Petri Nets 2020*, *PPAM 2022*
8. **List of peer-reviewed scientific or artistic works, particularly those published in international journals.**
 - Works published after the doctoral degree had been awarded (listed in Section I.2):
 1. *HPV-KITE: sequence analysis software for rapid HPV genotype detection*, Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, Laila Sara Arroyo Mühr, *Briefings in Bioinformatics*, Volume 26, Issue 2, March 2025, bbaf155, DOI: 10.1093/bib/bbaf155 [**Ministerial score: 140**]
 2. *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Concurrency and Computation: Practice and Experience*, Wiley, 2021/2023, ISSN: 1532-0626, eISSN: 1532-0634, Art. no. 6536 pp. 1-15, DOI: 10.1002/cpe.6536 [**Ministerial score: 100**]
 3. *Scalable computing in Java with PCJ library: improved collective operations*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Proceedings of Science – Proceedings of International Symposium on Grids and Clouds 2021 (ISGC2021) - Converging High Performance infrastructures: Supercomputers, clouds, accelerators*, ISSN: 1824-8039, Vol. 378, 2021, Art. no. 007, pp. 1-19, DOI: 10.22323/1.378.0007 [Ministerial score: 20]

4. *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Journal of Big Data*, Springer, 2021, ISSN: 2196-1115, Vol. 8 no. 1, Art no. 62, pp. 1-21, DOI: 10.1186/s40537-021-00454-6 [Ministerial score: 140]
 5. *Comparison of sort algorithms in Hadoop and PCJ*, Marek Nowicki, *Journal of Big Data*, Springer, 2020, ISSN: 2196-1115, Vol. 7 no. 1, Art no. 101, pp. 1-28, DOI: 10.1186/s40537-020-00376-9 [Ministerial score: 140]
 6. *PCJ – Java Library for Highly Scalable HPC and Big Data Processing*, Marek Nowicki, Łukasz Górski, Piotr Bała, *2018 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2018, ISBN: 978-1-5386-7878-7, pp. 12-20 [Ministerial score: 70]
 7. *Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *Journal of Computational Biology*, Mary Ann Liebert, Inc., 2018, ISSN: 1066-5277, Vol. 25 no. 8, pp. 871-881 [Ministerial score (previous): 25 / current: 70]
- Works published after the doctoral degree had been awarded (not listed in Section I.2):
8. *Słownikowe struktury danych i ich zastosowania*, Marcin Piątkowski, Magdalena Mroczek, Marek Nowicki, *Informatyka w edukacji : Edukacja informatyczna a kompetencje cyfrowe XXI wieku / Kwiatkowska Anna Beata (red.)*, 2025, Wydawnictwo Adam Marszałek, ISBN: 978-83-8180-347-2, pp. 80-90 [Ministerial score: 20]
 9. *Scratch : podstawy na przykładzie gry Wąż*, Marek Nowicki, *Informatyka w edukacji : kształcenie informatyczne drogą do cyfrowej transformacji społeczeństwa / Kwiatkowska Anna Beata, Sysło Maciej (red.)*, 2024, Wydawnictwo Adam Marszałek, ISBN: 978-83-8180-904-7, pp. 284-294 [Ministerial score: 20]
 10. *Czy jesteśmy gotowi na naukę programowania równoległego w szkole?*, Piotr Bała, Marek Nowicki, *Informatyka w edukacji : edukacja informatyczna : wyzwania współczesnego świata*, Wydawnictwo Adam Marszałek, 2023, Toruń, pp. 127-136, ISBN: 978-83-8180-810-1 [Ministerial score: 20]
 11. *Scratch w przedszkolu*, Marek Nowicki, *Informatyka w edukacji : uczniowie i nauczyciele w szkole przyszłości*, Marek Nowicki, Wydawnictwo Adam Marszałek, Toruń, 2022, s.414-424, ISBN: 978-83-8180-645-9 [Ministerial score: 20]
 12. *Performance Evaluation of Java/PCJ Implementation of Parallel Algorithms on the Cloud*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Euro-Par 2020: Parallel Processing Workshops. Euro-Par 2020*, Springer, Cham, 2021, ISBN: 978-3-030-71592-2, pp. 213-224 [Ministerial score: 140] – an extended version of the work is listed in the section I.2.
 13. *Performance evaluation of parallel computing and Big Data processing with Java and PCJ library*, Marek Nowicki, Łukasz Górski, Piotr Bała, *CUG2018 - Visionary Computing*, Cray User Group 2018, 21-24 May 2018, Stockholm, Sweden, 2018
 14. *Java w szkole*, Piotr Bała, Łukasz Górski, Marek Nowicki, *Informatyka w edukacji, myśl komputacyjnie!*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, Toruń, 2018, ISBN: 978-83-231-4049-8, pp. 171-172

15. *Scratch 3.0: nowe możliwości*, Piotr Bała, Marek Nowicki, *Informatyka w edukacji, myśl komputacyjnie!*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, Toruń, 2018, ISBN: 978-83-231-4049-8, pp. 173-174
16. *Fault-Tolerance Mechanisms for the Java Parallel Codes Implemented with the PCJ Library*, Michał Szynkiewicz, Marek Nowicki, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 298-307 [Ministerial score: 20]
17. *Big Data Analytics in Java with PCJ Library: Performance Comparison with Hadoop*, Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, Piotr Bała, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 318-327 [Ministerial score: 20]
18. *Evaluation of the Parallel Performance of the Java and PCJ on the Intel KNL Based Systems*, Marek Nowicki, Łukasz Górski, Piotr Bała, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 288-297 [Ministerial score: 20]
19. *Performance Comparison of Graph BFS Implemented in MapReduce and PGAS Programming Models*, Magdalena Ryczkowska, Marek Nowicki, *12th International Conference on Parallel Processing and Applied Mathematics (PPAM'2017)*, Springer, Cham, 2018, ISBN: 978-3-319-78053-5, pp. 328-337 [Ministerial score: 20]
20. *Massively Parallel Sequence Alignment with BLAST Through Work Distribution Implemented Using PCJ Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *17th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2017)*, Springer, Cham, ISBN: 978-3-319-65481-2, pp. 503-512 [Ministerial score: 70] – an extended version of the work is listed in the section I.2.
21. *Level-synchronous BFS algorithm implemented in Java using PCJ Library*, Magdalena Ryczkowska, Marek Nowicki, Piotr Bała, *2016 International Conference on Computational Science and Computational Intelligence (CSCI 2016)*, 15-17 December 2016, Las Vegas, Nevada, USA: proceedings, Piscataway, NJ : IEEE, 2016, ISBN: 978-1-5090-5510-4, pp. 596-601 [Ministerial score (previous): 15]
22. *Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ*, Marek Nowicki, Piotr Bała, *Innowacje w polskiej nauce w obszarze matematyki i informatyki. Przegląd aktualnej tematyki badawczej*, 2016, Wydawnictwo Nauka i Biznes, 2016, ISBN: 978-83-947095-2-5, pp. 130-140
23. *PCJ – a Java library for heterogenous parallel computing*, Marek Nowicki, Magdalena Ryczkowska, Łukasz Górski, Michał Szynkiewicz, Piotr Bała, *15th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS16)*, Barcelona, Spain, February 13-15, 2016, ISBN: 978-1-61804-365-8, pp. 66-72
24. *The performance evaluation of the Java implementation of Graph500*, Magdalena Ryczkowska, Marek Nowicki, Piotr Bała, *PPAM2015: 11th International Conference On Parallel Processing And Applied Mathematics*, September 6-9, 2015, Kraków, Poland, LNCS Series, ISBN: 978-3-319-32151-6, pp. 221-230 [Ministerial score (previous): 15]
25. *On-line service for teaching parallel programming*, Marek Nowicki, Maciej Marchwiany, Maciej Szpindler, Piotr Bała, *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops*, Vienna, Austria, August 24-25, 2015: revised selected papers, ISBN: 978-3-319-27307-5, pp. 78-89 [Ministerial score (previous): 15]

- Works published before the doctoral degree had been awarded (not listed in Section I.2):
 26. *PCJ – Java library for high performance computing in PGAS model*, Marek Nowicki, Łukasz Górski, Patryk Grabarczyk, Piotr Bała, *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014)*, ISBN: 978-1-4799-5311-0, pp. 202-209
 27. *The performance of the PCJ library for massively parallel computing*, Marek Nowicki, Piotr Bała, *7th International Conference on PGAS Programming Models*, The University of Edinburgh, 2013, ISBN: 978-0-9926615-0-2, pp. 254
 28. *On-line contests hosting service as a tool to teach computer science students programming*, Marek Nowicki, Łukasz Mikulski, Marcin Piątkowski, Rafał Kluszczyński, Piotr Bała, *10th World Conference on Computers in Education (WCCE 2013)*, 2-5 lipca 2013 r., Toruń, ISBN: 978-83-231-3093-2, pp. 61-71
 29. *Teaching secondary school students programming using distance learning. A case study*, Marek Nowicki, Michał Matuszak, Anna B. Kwiatkowska, Maciej M. Sysło, Piotr Bała, *10th World Conference on Computers in Education (WCCE 2013)*, 2-5 lipca 2013 r., Toruń, ISBN: 978-83-231-3093-2, pp. 246-254
 30. *PCJ – New Approach for Parallel Computations in Java*, Marek Nowicki, Piotr Bała, *PARA 2012 Proceedings*, Springer Lecture Notes in Computer Science, Vol. 7782, ISBN: 978-3-642-36802-8, pp. 115-125
 31. *Parallel computations in Java with PCJ library*, Marek Nowicki, Piotr Bała, *Proceedings of the 2012 International Conference on High Performance Computing & Simulation: HPCS 2012*, July 2 - July 6 2012, Madrid, Spain, ISBN: 978-1-4673-2359-8, pp. 381-387
 32. *Scratch – środowisko do nauki programowania*, Piotr Bała, Marek Nowicki, *Informatyka w edukacji: materiały z warsztatów VII Konferencji – IwE 2010*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011, ISBN: 978-83-23126-77-5, pp. 33-40
 33. *Regionalne Koło Informatyczne*, Piotr Bała, Anna Beata Kwiatkowska, Marek Nowicki, Mariusz Strzelecki, Maciej M. Sysło, *Informatyka w edukacji: materiały z warsztatów VII Konferencji – IwE 2010*, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2011, ISBN: 978-83-23126-77-5, pp. 61-75
 34. *Contests Hosting Service as a tool to teach programming*, Rafał Kluszczyński, Łukasz Mikulski, Marek Nowicki, Piotr Bała, *Annales UMCS Informatica*, Vol. 9/2009, ISSN:1732-1360, pp. 213-224, DOI: 10.2478/v10065-009-0017-9
 35. *Internetowe narzędzia do sprawdzania zadań programistycznych i ich zastosowania*, Marek Nowicki, Rafał Kluszczyński, Piotr Bała, *V Konferencja "Informatyka w Edukacji" – IwE 2008*, Toruń, 3-6 lipca 2008, Agencja Wydawnicza Argi, Wrocław, 2008, ISBN: 978-83-60425-30-5, pp. 88-94
- Other publications in the form of technical reports and abstracts:
 36. *Benchmarking the TeraSort algorithm on Ethernet Cluster : Benchmarking Apache Hadoop based implementation*, Marek Nowicki, *HPI Future SOC Lab : proceedings 2020 / Meinel Christoph [i in.] (red.)*, Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 159, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-565-1, pp. 97-108

37. *Benchmarking the sort algorithm on Ethernet Cluster : Benchmarking PCJ : a library for parallel computing in Java*, Marek Nowicki, *HPI Future SOC Lab : proceedings 2020* / Meinel Christoph [i in.] (red.), Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 159, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-565-1, pp. 53-63
38. *Benchmarking Java on Ethernet Cluster: benchmarking PCJ – a library for parallel computing in Java*, Marek Nowicki, *HPI Future SOC Lab: proceedings 2019* / Meinel Christoph [i in.] (red.), Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 158, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-564-4, pp. 263-271
39. *Benchmarking FFT on an Ethernet Cluster: Latency, Bandwidth and MPI benchmarks*, Marek Nowicki, Benson K. Muite, Boysan Mehmet Can, *HPI Future SOC Lab: proceedings 2019* / Meinel Christoph [i in.] (red.), Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam, vol. 158, 2024, Universitätsverlag Potsdam, ISBN: 978-3-86956-564-4, pp. 93-98
40. *High performance AI based tool for mining and classification of viral genomes in human exposome metadata (Abstracts from the 55th European Society of Human Genetics (ESHG) Conference: e-Posters)*, Magdalena Mroczek, Marek Nowicki, Łukasz Górski, Sara Arroyo Mühr, Alex Ormenisan, Piotr Bała, *European Journal of Human Genetics*, 31, 292 (2023)
41. *PCJ as a tool for massively parallel data processing*, Marek Nowicki, Łukasz Górski, Magdalena Ryczkowska, Piotr Bała, *CGW Workshop'17*, Kraków, Poland, October 23-25, 2017, proceedings, Academic Computer Centre Cyfronet. AGH, Kraków, 2017, ISBN: 978-83-61433-24-8, pp. 39-40
42. *Programowanie równoległe w języku Java z wykorzystaniem biblioteki PCJ*, Marek Nowicki, Piotr Bała, *Materiały konferencyjne - Innowacyjne projekty badawcze: II Ogólnopolska e-Konferencja Innowacyjnych Projektów Badawczych IPB*, Wrocław, 2 września 2016., Wrocław: Dolnośląski Akcelerator Technologii Innowacji Sp. z o.o., 2016, ISBN: 978-83-945037-2-7, pp. 30
43. *Development of parallel codes using PL-Grid infrastructure*, Rafał Kluszczyński, Marcin Stolarek, Grzegorz Marczak, Łukasz Górski, Marek Nowicki, Piotr Bała, *Cracow '14 Grid Workshop*, October 27-29 2014, Krakow, Poland, pp. 81-82
44. *Komunikacja wątków w bibliotece PCJ*, Łukasz Górski, Marek Nowicki, Piotr Bała, *XXVII Forum Informatyki Teoretycznej*, Toruń, 11-14 kwietnia 2013., Toruń, 2013, pp. 28

9. List of participation in European or other international programs.

- *The PCJ library on HLRS*, Cross-Site Collaboration Grant, EuroLab-4-HPC: Foundations of a European Research Center of Excellence in High Performance Computing Systems (2017)

10. List of participation in research teams implementing projects other than those specified in point II.4.

- member of the priority team Informatics – Formal Languages and Concurrency (FoLCo) within the University Center of Excellence *Dynamics, Mathematical Analysis, and Artificial Intelligence* (Excellence Initiative – Research University, Nicolaus Copernicus University in Toruń) – since 01/2020

- **principal investigator** of the computational grant *Benchmarking Java on Ethernet Cluster*, HPI Future SOC Lab, Hasso Plattner Institute, Potsdam, Germany (04/2019–05/2024)
- **principal investigator** of the computational grant *Further Development of PGAS Programming Methods in Java (PCJ Library)*, Third Call for Access to the Okeanos Supercomputer (OKEANOS 3), GA69-19, Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw (07/2017–07/2019)
- **principal investigator** of the computational grant *Scalability of PCJ Applications on HLRS*, Preparatory Type A – Code Scalability Testing, PRACE – Partnership for Advanced Computing in Europe / HLRS, University of Stuttgart, Germany (06/2017–07/2017)
- **principal investigator** of the computational grant *Development of PGAS Programming Methods in Java (PCJ Library)*, First Call for Access to the Okeanos Supercomputer, GB65-15, Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw (06/2016–06/2017)

11. List of participation in teams assessing applications for research funding, applications for granting scientific awards, applications in other competitions having a scientific or didactic nature.

- Member of the evaluation committee for *Grants4NCUStudents*, 4th edition (2022), for research projects and international travel undertaken by students and PhD candidates of Nicolaus Copernicus University in Toruń – ST panel

III. COOPERATION WITH THE SOCIAL AND ECONOMIC ENVIRONMENT

1. List of technological achievements.

Not applicable.

2. Cooperation with the economic sector.

- **co-founder** and one of the leaders of *Toruń JUG* – the Toruń-based group for enthusiasts of Java Virtual Machine technologies (02/2014–12/2024)

3. List of obtained industrial property rights, including obtained national or international patents.

Not applicable.

4. List of implemented technologies.

- *ZawodyWeb* system – deployed, among others, at the Faculty of Mathematics and Computer Science, Nicolaus Copernicus University in Toruń
- *PCJ* library – available in the central Maven repository, deployed, among others, in the *International ImMunoGenetics Information System (IMGT)*

- *PCJ-BLAST* application – available in the *GitHub* repository
- *HPV-KITE* application – available, among others, in the *Conda-Forge* repository
- *KITE* application – available, among others, in the *Conda-Forge* repository
- the development and deployment of an internal **IBM** wide *Sales Decision Support* system that was adopted by more than 5000 sellers in IBM worldwide

5. List of performed expert opinions or other studies made to order for public institutions or entrepreneurs.

Not applicable.

6. List of participation in expert or competition teams.

- member of the Regional Committee in Toruń of *Polish Olympiad in Informatics* (in Polish: *Olimpiada Informatyczna*) (since 01/2014)
- member of the jury of the hackathon *BrainCode 2019* organized by Allegro, 12-13 April 2019, Toruń, Poland
- **chair** of the competition committee: *Great Programming Challenges Hackathon* (in Polish: *Hackaton Wielkie Wyzwania Programistyczne*) – edition III: *Switch to Smart (Switch to Smart)*, ICM UW, 23-24 November 2018, Warsaw, Poland
- member of the jury of the hackathon *BrainCode 2018* organized by Allegro, 20-21 April 2018, Toruń, Poland
- member of the judging committee of the competition *FRI: III Kuyavian-Pomeranian Programming Contest* (in Polish: *III Kujawsko-Pomorskie Zawody Programistyczne*), 20 April 2018, Toruń, Poland
- **chair** of the competition committee: *Great Programming Challenges Hackathon* (in Polish: *Hackaton Wielkie Wyzwania Programistyczne*) – edition II: *Smart Charge (Smart Charge)*, ICM UW, 15-16 February 2018, Warsaw, Poland
- member of the jury of the hackathon *BrainCode 2017* organized by Allegro, 23-24 March 2017, Toruń, Poland
- member of the judging committee of the competition *FRI: II Kuyavian-Pomeranian Programming Contest* (in Polish: *II Kujawsko-Pomorskie Zawody Programistyczne*), 24 March 2017, Toruń, Poland
- member of the competition committee: *Contest Coding with Scratch 2017* (in Polish: *Kodowanie ze Scratchem 2017*) – for primary school students, 26 May 2017, Toruń, Poland
- **chair** of the competition committee: *Great Programming Challenges Hackathon* (in Polish: *Hackaton Wielkie Wyzwania Programistyczne*) – edition I: *Map the Gap (Map the Gap)*, ICM UW, 16-17 February 2017, Warsaw, Poland
- member of the competition committee: *Contest Coding with Scratch 2016* (in Polish: *Kodowanie ze Scratchem 2016*) – for primary school students, 17 June 2016, Toruń, Poland

- member of the competition committee: Contest *Coding with Scratch 2015* (in Polish: *Kodowanie ze Scratchem 2015*) – for primary school students, 19 June 2015, Toruń, Poland
- assistance in conducting the *Informatics Contest* (in Polish: *Konkurs Informatyczny*) (editions: 2006/2007, 2007/2008, 2008/2009) at the Faculty of Mathematics and Computer Science, Nicolaus Copernicus University in Toruń, Poland
- assistance in conducting a *mini-programming contest* (in Polish: *mini-konkurs programistyczny*) as part of the *24th Toruń Science and Arts Festival* (in Polish: *24. Toruński Festiwal Nauki i Sztuki*), April 2006

7. List of artistic projects implemented with non-artistic environments.

Not applicable.

IV. SCIENTOMETRIC DATA

Bibliometric data as of 26 November 2025

1. Impact Factor (in fields and disciplines where this parameter is commonly used as a scientometric indicator).

Data from the *Omega Knowledge Base*, Nicolaus Copernicus University in Toruń:

- Total Impact Factor (IF): 20.914
- Total ministerial score: 1100

2. Number of citations of the applicant's publications, with separate consideration of self-citations.

- Scopus: 124 (62 without self-citations)
- Web of Science: 85 (46 without self-citations)
- Google Scholar: 249

3. Hirsch index.

- Scopus: 7
- Web of Science: 5
- Google Scholar: 10



applicant's signature

Davit Bzhalava

Stockholm, 31 October 2025 r.

Department of Laboratory Medicine,
Karolinska Institutet,
Stockholm, Sweden

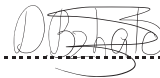
Declaration of the substantive contribution of the work

I declare that I am a co-author of the work entitled:

Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library, Marek Nowicki, Davit Bzhalava, Piotr Bała, *Journal of Computational Biology*, Mary Ann Liebert, Inc., 2018, ISSN: 1066-5277, Vol. 25 no. 8, pp. 871-881

I also confirm that my substantive contribution to the work was as follows:

I was responsible for providing the datasets, verifying the results, and describing the biological knowledge presented in the work.



(signature)

Interdyscyplinarne Centrum Modelowania
Matematycznego i Komputerowego
Uniwersytet Warszawski

Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika w Toruniu

Oświadczenie o wkładzie merytorycznym

Oświadczam, że jestem współautorem poniższych prac i przedstawiam mój wkład merytoryczny włożony w ich powstanie:

1. *PCJ - Java Library for Highly Scalable HPC and Big Data Processing* (Invited Tutorial Paper), Marek Nowicki, Łukasz Górski, Piotr Bała, *2018 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2018, ISBN: 978-1-5386-7878-7, pp. 12-20

Wkład merytoryczny:

Moim wkładem merytorycznym w pracę była implementacja, przeprowadzone testy wydajnościowe i opis rozwiązania *WordCount*. Ode mnie pochodzi również część treści pracy (w szczególności część tekstu motywacji). Brałem również udział w poprawianiu tekstu pracy (m.in. poprawianie literówek i stylu zdań).

2. *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Concurrency and Computation: Practice and Experience*, Wiley, 2023, ISSN: 1532-0626, eISSN: 1532-0634, Art. no. 6536 s. 1-15, DOI: 10.1002/cpe.6536

Wkład merytoryczny:

Moim wkładem merytorycznym w pracę było przygotowanie implementacji, metodologii testowania i opisu rozwiązań: *Fast Fourier Transform (FFT)* oraz *WordCount*. Wykonałem również kilka drobnych poprawek w treści pracy.

3. *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Journal of Big Data*, Springer, 2021, ISSN: 2196-1115, Vol. 8 no. 1, Art no. 62, s. 1-21, DOI: 10.1186/s40537-021-00454-6

Wkład merytoryczny:

Moim wkładem w pracę była implementacja, testy wydajnościowe, opis i analiza uzyskanych wyników następujących rozwiązań: *FFT*, *WordCount*, ewolucji różnicowej i rozproszonego treningu sieci neuronowej poprzez integrację biblioteki PCJ z TensorFlow. Przetestowałem i opisałem tę integrację na przykładzie rozpoznawania cyfr zapisanych ręcznie korzystając z zbioru danych MNIST i porównałem uzyskaną wydajność z wydajnością uzyskaną w przeprowadzonych przeze mnie testach wydajnościowych przy użyciu

rozwiązania Horovod. Brałem udział w poprawianiu tekstu pracy w sekcjach nieodnoszących się do moich badań (drobne poprawki językowo-redakcyjne).

4. *Scalable computing in Java with PCJ library: improved collective operations*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Proceedings of Science – Proceedings of International Symposium on Grids and Clouds 2021 (ISGC2021) - Converging High Performance infrastructures: Supercomputers, clouds, accelerators*, ISSN: 1824-8039, Vol. 378, 2021, Art. no. 007, s. 1-19, DOI: 10.22323/1.378.0007

Wkład merytoryczny:

Moim wkładem była bazowa implementacja rozwiązania *Fast Fourier Transform (FFT)* oraz *WordCount* wraz z metodologią testowania; w szczególności byłem twórcą implementacji algorytmów komunikacji *wszyscy-do-wszystkich*: *hypercube*, *nonblocking* i *blocking* dla *FFT* oraz operacji redukcji *node0*, *local first* i *hypercube* dla *WordCount*. Przygotowałem również implementację i wykonałem ewaluację wydajności rozproszonego treningu sieci neuronowej korzystając z biblioteki PCJ w wersji 5.2.0 a także z użyciem rozwiązania Horovod. Opisałem ten problem i przeanalizowałem uzyskane rezultaty.


.....
(podpis)

dr Marek Nowicki

Toruń, 26 listopada 2025 r.

Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika w Toruniu

Interdyscyplinarne Centrum Modelowania
Matematycznego i Komputerowego
Uniwersytet Warszawski

Oświadczenie o wkładzie merytorycznym

Ze względu na śmierć **prof. Piotra Bały** dnia 31 sierpnia 2024 r., nie posiadam jego oświadczenia o wkładzie merytorycznym do przedstawionych w cyklu pracach współautorskich.

Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika w Toruniu

Interdyscyplinarne Centrum Modelowania
Matematycznego i Komputerowego
Uniwersytet Warszawski

Oświadczenie o wkładzie merytorycznym

Oświadczam, że mam następujący wkład merytoryczny w powstanie poniższych publikacji:

1. *Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library*, Marek Nowicki, Davit Bzhalava, Piotr Bała, *Journal of Computational Biology, Mary Ann Liebert, Inc.*, 2018, ISSN: 1066-5277, Vol. 25 no. 8, pp. 871-881

W ramach badań będących podstawą niniejszej pracy zaprojektowałem i zaimplementowałem mechanizm przygotowywania i równoległego przetwarzania danych wejściowych i zapisu danych wyjściowych. Ponadto zaimplementowałem mechanizm wyrównujący obciążenie poszczególnych składników systemu. Wykonałem również testy wydajnościowe swojego rozwiązania i innych rozwiązań porównywanych w pracy. Byłem odpowiedzialny za przygotowanie opisu przedstawianego rozwiązania i wykonanych badań a także wyników wydajnościowych wraz z dyskusją osiągniętych rezultatów. Jako autor wiodący i korespondencyjny byłem odpowiedzialny za kontakt z czasopismem i przygotowanie odpowiedniej wersji pracy wraz z korektami edycyjnymi.

2. *PCJ - Java Library for Highly Scalable HPC and Big Data Processing (Invited Tutorial Paper)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *2018 International Conference on High Performance Computing & Simulation (HPCS), IEEE*, 2018, ISBN: 978-1-5386-7878-7, pp. 12-20

Byłem współodpowiedzialny za przygotowanie koncepcji warsztatów na konferencje *PPAM'15*, *PPAM'17* i *Supercomputing Frontiers Europe 2018*, po których zostaliśmy zaproszeni do przedstawienia tutoriala na konferencji *The International Conference on High Performance Computing & Simulation (HPCS 2018)*.

Byłem odpowiedzialny za przygotowanie prawie całego opisu biblioteki PCJ – w tym ilustracji i większości przykładowych kodów źródłowych.

3. *Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud (extended version)*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Concurrency and Computation: Practice and Experience, Wiley*, 2023, ISSN: 1532-0626, eISSN: 1532-0634, Art. no. 6536 s. 1-15, DOI: 10.1002/cpe.6536

Niniejsza praca to rozszerzona wersja pracy z warsztatów *2nd International Workshop on Parallel Programming Models in High-Performance Cloud (ParaMo 2020)* konferencji *Euro-Par 2020: 26th International Conference on Parallel and Distributed Computing*.

W ramach konferencji przedstawiłem referat o uzyskanych wynikach i zostałem poproszony o przygotowanie wersji rozszerzonej artykułu.

W ramach badań będących podstawą pracy zaprojektowałem i zaimplementowałem mechanizm uruchamiania zadań równoległych wykorzystujących bibliotekę PCJ na chmurach obliczeniowych. Ponadto byłem odpowiedzialny za kontakt z firmami Amazon i Linaro by uzyskać zasoby obliczeniowe do celów naukowych w ilościach wystarczających do przeprowadzenia miarodajnych badań. Dodatkowo zajmowałem się obsługą maszyn chmurowych i wykonywaniem obliczeń z wykorzystaniem tych zasobów. Opisałem mechanizm przygotowania środowiska chmurowego do wykonania obliczeń z wykorzystaniem biblioteki PCJ. Wykonałem wszystkie przedstawione w publikacji testy wydajnościowe aplikacji równoległych wykorzystujących bibliotekę PCJ na obu chmurach jak również na superkomputerze Okeanos w ramach grantu obliczeniowego, którego byłem kierownikiem. Byłem odpowiedzialny za przygotowanie opisu konfiguracji eksperymentu a także wyników wydajnościowych wraz z dyskusją osiągniętych rezultatów. Jako autor wiodący i korespondencyjny byłem odpowiedzialny za kontakt z czasopismem i przygotowanie odpowiedniej wersji pracy wraz z korektami edycyjnymi.

4. *PCJ Java library as a solution to integrate HPC, Big Data and Artificial Intelligence workloads*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Journal of Big Data*, Springer, 2021, ISSN: 2196-1115, Vol. 8 no. 1, Art no. 62, s. 1-21, DOI: 10.1186/s40537-021-00454-6

Byłem odpowiedzialny za szczegółowy opis biblioteki PCJ oraz przygotowanie i uruchomienie testów wydajnościowych badających wydajność biblioteki w problemie typu *2D stencil (Game of Life)* na wybranych superkomputerach. W ramach badań zdobyłem grant na uruchomienie programów wykorzystujących bibliotekę PCJ na superkomputerze *HazelHen* w HLRS w Niemczech, będącym w tamtym czasie na 17. miejscu listy TOP500. Dzięki temu możliwe było wykonanie testów wydajnościowych na ponad 4000 węzłach obliczeniowych tego superkomputera, co pozwoliło na uruchomienie aplikacji na blisko 200000 wątkach obliczeniowych. Ponadto opisałem uzyskane wyniki wraz z dyskusją osiągniętych rezultatów. Byłem odpowiedzialny za całość pracy. Jako autor wiodący i korespondencyjny byłem odpowiedzialny za kontakt z czasopismem i przygotowanie odpowiedniej wersji pracy wraz z korektami edycyjnymi.

5. *Scalable computing in Java with PCJ library: improved collective operations*, Marek Nowicki, Łukasz Górski, Piotr Bała, *Proceedings of Science – Proceedings of International Symposium on Grids and Clouds 2021 (ISGC2021) - Converging High Performance infrastructures: Supercomputers, clouds, accelerators*, ISSN: 1824-8039, Vol. 378, 2021, Art. no. 007, s. 1-19, DOI: 10.22323/1.378.0007

Niniejsza praca ukazała się w materiałach pokonferencyjnych konferencji *International Symposium on Grids and Clouds 2021 (ISGC2021)*, na której prezentowałem uzyskane rezultaty.

Byłem odpowiedzialny za przygotowanie koncepcji badań, implementacji mechanizmów kolektywnych w bibliotece PCJ, przeprowadzenie testów wydajnościowych i prawie całego tekstu manuskryptu. Jako autor wiodący byłem odpowiedzialny za kontakt z organizatorami konferencji i przygotowanie odpowiedniej wersji pracy wraz z korektami edycyjnymi.

6. *HPV-KITE: sequence analysis software for rapid HPV genotype detection*, Marek Nowicki, Magdalena Mroczek, Dhananjay Mukhedkar, Piotr Bała, Ville Nikolai Pimenoff, Laila Sara Arroyo Mühr, *Briefings in Bioinformatics*, Volume 26, Issue 2, March 2025, bbaf155, DOI: 10.1093/bib/bbaf155

Byłem pomysłodawcą powstania tego artykułu. Opis wkładu merytorycznego poszczególnych autorów zawarty jest w samej pracy.

W ramach niniejszej pracy razem z L.S.A.M, V.N.P, M.M, i P.B byłem współodpowiedzialny za koncepcję i projekt przeprowadzonych badań. L.S.A.M, V.N.P i D.M byli odpowiedzialni za przygotowanie zbiorów danych. D.M przeanalizował wybrany zbiór danych za pomocą Diamond. M.M zasugerowała użycie istniejącej bazy wirusów HPV – PaVE. Opracowałem i zaimplementowałem prototypową wersję algorytmu zaprezentowanego w pracy. Po zweryfikowaniu poprawności działania metody na małych przykładowych danych, zaimplementowałem algorytm w języku Java i zrównolegliłem jego działanie przy użyciu biblioteki PCJ. Następnie testowałem i optymalizowałem działanie aplikacji na dostępnych zbiorach danych. Instalowałem, konfigurowałem i uruchamiałem różne inne narzędzia i testowałem ich wydajność. Wraz z M.M, L.S.A.M i V.N.P brałem udział w weryfikacji wyników i uzyskanej jakości rezultatów na dostępnych zbiorach danych. Razem z L.S.A.M, V.N.P, i P.B nadzorowałem pracę członków zespołu. Wraz z M.M brałem udział w pisaniu pierwszego szkicu manuskryptu. Wraz z M.M, L.S.A.M i V.N.P wykonywałem potrzebne korekty manuskryptu w odpowiedzi na otrzymane recenzje. Jako autor wiodący i korespondencyjny dostosowałem styl manuskryptu do wymagań czasopisma oraz byłem odpowiedzialny za kontakt z czasopismem i przygotowanie kolejnych wersji pracy wraz z korektami edycyjnymi.



.....
(podpis)



UNIwersytet Warszawski

Wydział Matematyki, Informatyki i Mechaniki
(nazwa jednostki organizacyjnej)

DYPLOM

WYDANY W RZECZYPOSPOLITEJ POLSKIEJ

Marek Nowicki
(imię i nazwisko)

urodzony dnia

r. w

na podstawie przedstawionej rozprawy doktorskiej *Opracowanie nowych metod programowania równoległego w Java w oparciu o paradygmat PGAS (Partitioned Global Address Space)*

oraz po złożeniu wymaganych egzaminów uzyskał stopień naukowy

DOKTORA

nauk *matematycznych w zakresie informatyki*

(bliższe określenie nazwy stopnia)

nadany uchwałą Rady *Wydziału Matematyki, Informatyki i Mechaniki*

(nazwa rady jednostki organizacyjnej)

z dnia *18 czerwca* 2015 r.

Promotor w przewodzie doktorskim: *dr hab. Piotr Bala*

dr Michał Chlebicz

Recenzenci w przewodzie doktorskim: *prof. dr hab. inż. Marek Suchan*

dr hab. inż. Paweł Wojciechowski

Warszawa, 29 czerwca 2015 r.
(miejscowość, data)

Prof. dr hab. Andrzej Tarlecki
Przewodniczący Rady jednostki
organizacyjnej/Dziekan
(pieczęć imienna i podpis)

Piotr Bala
Promotor
(podpis)

dr hab. Marcin Pałys, prof. UW
Rektor
(pieczęć imienna i podpis)

**Karolinska
Institutet**

Sida: 1 (2)

Certificate

2025-11-06

**Center for Cervical Cancer
Elimination
Department of
Clinical Science,
Intervention and Technology –
CLINTEC**

Joakim Dillner,
Professor

To whom it may concern

This is to certify that Dr. Marek Nowicki has been an *Affiliated Researcher* at the *Center for Cervical Cancer Elimination*, part of the *Department of Clinical Science, Intervention and Technology* of *Karolinska Institutet*, from 1st February 2023 to 31st October 2025.

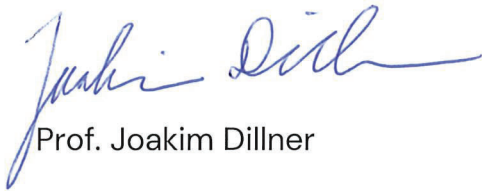
Our Center focuses on increasing the knowledge base for the design and evaluation of the most effective cervical cancer elimination strategies possible. We conduct both basic and translational studies of the tumor virus Human Papillomavirus (HPV) and HPV-based prevention (screening and vaccination), including an international reference laboratory function where we sequence, document, and store all new Papillomaviruses. The Center also operates the Swedish National Cervical Screening Registry that evaluates and optimises the cervical cancer prevention in Sweden.

One of the larger EU projects at the Center was HEAP (Human Exposome Assessment Platform; H2020 Grant agreement ID: 874662), where we have been developing an AI-driven data platform to evaluate the impact of the internal and external exposome on health.

Dr. Marek Nowicki's affiliation was connected to the HEAP project, enabling him to conduct research and access our datasets and computing resources. One of his key contributions during this affiliation was the creation and development of the HPV-

KITE application, which can be used to rapidly detect HPV genomes in metagenomic data.

Dr. Marek Nowicki's research contributions during the affiliation definitely supported the successful completion of the HEAP project.



Prof. Joakim Dillner

Head of Center

Professor Joakim Dillner
Karolinska Institutet
Department for Clinical Science
Intervention and Technology (CLINTEC)

**ZAŁ. 3** do zgody kierownika jednostki na udział w projekcie

Human Exposome Assessment Platform, HEAP, H2020
Tytuł projektu, akronim, program

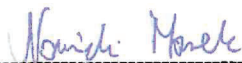
dr Marek Nowicki, ICM UW

Imię i nazwisko kierownika projektu, nazwa jednostki UW

Obowiązki kierownika projektu

1. Realizacja zadań UW w Projekcie zgodnie ich opisem w umowie z instytucją finansującą 874662.
2. Określenie zakresu obowiązków członków zespołu realizujących zadania w projekcie,
3. Określenie wymiaru czasu pracy oraz okresu zatrudnienia członków zespołu w projekcie zgodnie z zaangażowaniem czasowym, przewidzianym w umowie grantowej (nieprzekraczanie liczby osobo-miesięcy, dni lub godzin bez zgody instytucji finansującej);
4. Nadzorowanie realizacji powierzonych obowiązków członkom zespołu,
5. Zatwierdzanie wydatków pod względem merytorycznym,
6. Wydatkowanie powierzonych środków w projekcie zgodnie z zasadami jego finansowania, w szczególności w sposób zgodny z obowiązującymi przepisami, racjonalny, oszczędny i uzasadniony,
7. Zatwierdzanie kart czasu pracy, poprzez złożenie własnoręcznego podpisu z częstotliwością wynikającą z zasad realizacji projektu, jeśli wymagają tego zasady kwalifikowalności kosztów osobowych w projekcie,
8. Sporządzanie i składanie do instytucji finansujących, w tym za pośrednictwem koordynatora, tam, gdzie to wymagane, raportów merytorycznych zgodnie z harmonogramem sprawozdań, określonym w projekcie,
9. Współpraca z administracją centralną, sekcjami finansowymi i sekcjami obsługi badań w jednostkach z zakresie administrowania projektem, w szczególności w odniesieniu do terminowego przygotowania wniosków o dofinansowanie projektu, raportów okresowych i raportu końcowego z wydatkowania środków finansowych, oraz przygotowania szczegółowych wyjaśnień dotyczących wydatkowania środków, w szczególności na żądanie instytucji finansującej lub innego podmiotu kontrolującego wydatki w projekcie.

Przyjąłem/przyjęłam do wiadomości i zobowiązuje się realizować ww. obowiązki kierownika projektu z należytą starannością.


Podpis kierownika projektu

Wyrażam zgodę na zmianę kierownika projektu od maja 2024 r.

V2.0_1.08.2019 r.


p.o. Dyrektor ICM
dr Robert Sot



IBM Research GmbH
IBM Research – Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
Tel +41 (0)44 724 81 11
Fax +41 (0)44 724 89 11
www.zurich.ibm.com

To whom it may concern

Mr. Marek Nowicki was employed in IBM Research - Zurich (Switzerland) from May 5, 2013 to November 6, 2013. He worked as an intern in our "Information Analytics" group.

The IBM Research - Zurich Laboratory is the European branch of IBM's research division, which employs some 3000 individuals at nine research laboratories worldwide. It is the largest industrial IT research organization in the world. Cutting-edge research and outstanding scientific achievements - most notably two Nobel Prizes - are associated with the Zurich Lab.

Mr. Nowicki was a recipient of the Great Minds internship in our Lab which is awarded yearly to very highly qualified students in EMEA after a very tough competition among students from the best research and education institutions in the region.

During his internship, Mr. Nowicki worked on the development and deployment of an IBM wide Sales Decision Support system that is now adopted by more than 5000 sellers in IBM worldwide. His contributions to the project were mainly at the Middleware level where he successfully designed and implemented in Java several data and analytics services that are required by the application. His work also included several system integration challenges as he worked on integrating third-party APIs, internal data sources, and other middleware services. He consistently delivered valuable contributions to the project all along his working period in our Lab.

Beside his technical and scientific skills, Mr. Nowicki is a very pleasant person to work with, and was very well appreciated and respected by the whole project team.

We thank Mr. Nowicki for his great contributions, and wish him the very best for his new venture as a next step in an outstanding career.


Dr. Abdel Labbi
Manager Information Analytics


Oliver Ottow
Human Resources Manager

Rüschlikon, August 12, 2014/cle



HPC CHALLENGE

CLASS2AWARD

PRESENTED AT



SC14
New Orleans, hpc
LA | mellers.

NOVEMBER 18, 2014

PODPIS ZAUFANY

MAREK
NOWICKI

26.11.2025 16:20:52 GMT+1
Dokument podpisany elektronicznie
podpisem zaufanym

WINNER

Most Elegant

PRESENTER

Piotr Bała

ICM University of Warsaw

LANGUAGE

PCJ

DAVID BAILEY
JACK DONGARRA

JEREMY KEPNER
BOB LUCAS

EWING LUSK
PIOTR LUSZCZEK

JOHN MCCALPIN
ROLF RABENSEIFNER

DAISUKE TAKAHASHI
JEFFREY VETTER





**IMGT®, the international ImMunoGeneTics
information system®**

Institut de Génétique Humaine
UMR 9002 CNRS-Université de Montpellier



PODPIS ZAUFANY

MAREK

NOWICKI

26.11.2025 16:22:55 GMT+1

Dokument podpisany elektronicznie
podpisem zaufanym

Tics information
system®
<http://www.imgt.org>

Prof. Sofia KOSSIDA
Head of the IMGT team
IMGT®, the international ImMunoGeneTics information system®
Institut de Génétique Humaine, IGH, UMR 9002 CNRS-UM
141, rue de la Cardonille
34396 Montpellier Cedex 5 - France

Montpellier, May 22nd, 2025

Dear Dr Marek Nowicki,

IMGT® is a pioneering international information system in immunogenetics and immunoinformatics. It provides databases and tools based on the IMGT-ONTOLOGY to the scientific community for in-depth exploration of the adaptive immune response. IMGT® is dedicated to advancing scientific knowledge through its three main axes of research and development.

IMGT/V-QUEST (V-QUERy and STandardization) is an integrated alignment tool for the immunoglobulin (IG) and T cell receptor (TR) nucleotide sequences. IMGT/V-QUEST compares germline or rearranged IG or TR variable sequences with the IMGT/V-QUEST reference directory sets.

IMGT/HighV-QUEST is the multithreaded, single node version of IMGT/V-QUEST.

In this context, we are conducting integration and performance tests of the PCJ library to enhance the throughput of IMGT/HighV-QUEST when deployed on clusters; depending on the outcome of these tests, we may integrate PCJ as a full part of IMGT/HighV-QUEST.

Sincerely yours,

Sofia KOSSIDA