Dr inż. Tadeusz Tomczak
Katedra Informatyki Technicznej
Wydział Elektroniki Politechniki Wrocławskiej

Załącznik 2.
do Wniosku o przeprowadzenie postępowania habilitacyjnego

AUTOREFERAT

przedstawiający opis osiągnięć naukowych,
w szczególności określonych w art. 16 ust. 2 ustawy o stopniach naukowych i tytule naukowym
oraz o stopniach i tytule w zakresie sztuki

Wrocław, luty 2019

Spis treści

1 Imię i nazwisko 3

2 Posiadane dyplomy i stopnie naukowe 3

3 Informacje o dotychczasowym zatrudnieniu w jednostkach naukowych 3

4 Wskazanie osiągnięcia wynikającego z art. 16 ust. 2 ustawy z dnia 14 marca 2003 r. o stopniach naukowych i tytule naukowym oraz o stopniach i tytule w zakresie sztuki (Dz. U. nr 65, poz. 595 ze zm.) 3

4.1 Publikacje wchodzące w skład osiągnięcia naukowego 3

4.2 Omówienie celu naukowego ww. prac i osiągniętych wyników wraz z omówieniem ich ewentualnego wykorzystania 7

4.2.1 Programowe metody zwiększania szybkości przetwarzania 8

4.2.1.1 Metody zwiększania szybkości przetwarzania w obliczeniach szablonowych dla geometrii rzadkich 10

4.2.1.2 Metody zwiększania szybkości przetwarzania w algorytmach algebry liniowej dla macierzy rzadkich 20

4.2.2 Sprzętowe metody zwiększania szybkości przetwarzania 31

4.2.2.1 Arytmetyka resztowa w szybkich cyfrowych układach arytmetycznych 31

4.3 Podsumowanie wyników zawartych w pracach [A1]-[A6] 36

5 Omówienie pozostałych osiągnięć naukowo-badawczych 38

5.1 Algorytmy szybkiego numerycznego całkowania równań różniczkowych z wykorzystaniem bezkwadraturowej nieciągłej metody Galerkina 38

5.2 Algorytmy numerycznych symulacji przepływu płynów w mikrokanałach z wykorzystaniem metody siatkowej Boltzmanna 39

5.3 Współpraca z przemysłem 39

5.3.1 Opracowanie komputera przyspieszającego obliczenia CFD 39

5.3.2 Metody przyspieszania procedur rozwiązywania układów równań liniowych 40

5.3.3 Masowo równoległe wersje algorytmów PISO i SIMPLE 40

5.3.4 Masowo równoległe algorytmy algebry liniowej dla macierzy rzadkich . . . 40

6 Działalność dydaktyczna 40

1 Imię i nazwisko

Tadeusz Tomczak

2 Posiadane dyplomy i stopnie naukowe

- Doktor nauk technicznych, dyscyplina: Informatyka
  Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, 2007 r.

  Tytuł rozprawy doktorskiej: „*Akceleracja sprzętowa działań arytmetycznych w algorytmach oświetlenia globalnego*" (praca obroniona z wyróżnieniem, nagrodzona Nagrodą Rektora Politechniki Wrocławskiej).

  Promotor: dr hab. inż. Janusz Biernat, prof. PWr

- Magister inżynier, kierunek: Informatyka
  Wydział Elektroniki Politechniki Wrocławskiej, 2002 r.

  Tytuł pracy magisterskiej: „*Budowa i zastosowanie procesorów o skalowalnej architekturze*".

  Promotor: dr inż. Jacek Majewski

3 Informacje o dotychczasowym zatrudnieniu w jednostkach naukowych

- 2011 – obecnie: adiunkt, Wydział Elektroniki Politechniki Wrocławskiej
- 2007 – 2011: asystent, Wydział Elektroniki Politechniki Wrocławskiej

4 Wskazanie osiągnięcia wynikającego z art. 16 ust. 2 ustawy z dnia 14 marca 2003 r. o stopniach naukowych i tytule naukowym oraz o stopniach i tytule w zakresie sztuki (Dz. U. nr 65, poz. 595 ze zm.)

Osiągnięciem naukowym wynikającym z art. 16 ust. 2 ustawy z dnia 14 marca 2003 roku o stopniach naukowych i tytule naukowym oraz o stopniach i tytule w zakresie sztuki (Dz. U. nr 65, poz. 595 ze zm.) jest jednotematyczny cykl publikacji naukowych pt.:

Opracowanie metodyki zwiększania szybkości przetwarzania z wykorzystaniem równoległości drobnoziarnistej w obliczeniach wysokiej wydajności

4.1 Publikacje wchodzące w skład osiągnięcia naukowego

Cykl publikacji stanowiący podstawę ubiegania się o stopień doktora habilitowanego składa się z 6 pozycji o sumarycznym współczynniku wpływu IF = 9.318. Sumaryczna liczba punktów według Ministerstwa Nauki i Szkolnictwa Wyższego (MNiSW), zgodnie z wykazem czasopism naukowych oraz przepisami dotyczącymi przyznawania kategorii naukowej jednostkom naukowym, jest równa 143 pkt.

Wartości współczynników IF oraz pkt. MNiSW podano zgodnie z rokiem publikacji z wyjątkiem prac najnowszych ([A2] i [A3]), dla których przyjęto dane z okresu bezpośrednio poprzedzającego publikację. W nawiasach podano procentowy wkład poszczególnych współautorów w przygotowanie publikacji na podstawie załączonych oświadczeń.

[A1] **T. Tomczak** (75%), R. Szafran (25%), sierpień 2018 r., Sparse geometries handling in lattice Boltzmann method implementation for graphic processors, *IEEE Transactions on Parallel and Distributed Systems*, t. 29, nr 8, s. 1865–1878.
IF: 3.971, MNiSW: 40 pkt.

Mój wkład w powstanie tej pracy polegał na:

- Opracowaniu dedykowanych dla procesorów masowo równoległych, w szczególności akceleratorów graficznych, struktur danych opartych na podziale geometrii na niewielkie, kwadratowe lub sześcienne podobszary o jednakowym rozmiarze (kafelki, *ang. tiles*). W porównaniu z dotychczas stosowanymi rozwiązaniami dla geometrii rzadkich, opracowane struktury danych umożliwiają znaczne ograniczenie rozmiaru przesyłanych danych dzięki ich regularnemu ułożeniu w pamięci oraz wyeliminowaniu większości dostępów do danych z węzłów nieobliczeniowych.
- Opracowaniu dla zaproponowanych struktur danych szybkich wersji algorytmów realizujących wszystkie kroki metody siatkowej Boltzmanna. Algorytmy te charakteryzują się najwyższą na świecie wydajnością dla wielu klas geometrii rzadkich dzięki zachowaniu właściwej kolejności dostępu do danych, buforowaniu w pamięci notatnikowej (*ang. scratchpad memory*) danych opisujących położenie kafelków, małej liczbie punktów synchronizacji, oraz konstrukcji umożliwiającej wykorzystanie typowych optymalizacji dla akceleratorów graficznych (np. grupowania operacji z wykorzystaniem równoległości na poziomie instrukcji (*ang. instruction-level parallelism, ILP*) czy przeprowadzania operacji o długim czasie wykonania poza fragmentami kodu dywergentnego).
- Przeprowadzeniu szczegółowej teoretycznej analizy pozwalającej określić minimalne zapotrzebowanie na pamięć i maksymalną wydajność zarówno dla zaproponowanych metod, jaki i dla dotychczas stosowanych technik.
- Porównaniu ograniczeń różnych metod obsługi geometrii rzadkich (rozdz. 2.2 *Performance model*, 2.3 *Sparsity handling overhead*, 3.1 *Tiles overhead*).
- Zaplanowaniu, przygotowaniu (włącznie z zaprojektowaniem i zaimplementowaniem przykładowego rozwiązania w technologii CUDA, które powstało na podstawie poprawnej numerycznie wersji zaimplementowanej przez dr R. Szafrana) i przeprowadzeniu eksperymentów przedstawionych w rozdz. 4 *Results*, opracowaniu i analizie wyników oraz sformułowaniu wniosków. Dla geometrii 2D z rys. 6 przeskalowałem na różne rozmiary przypadki wzorcowe przygotowane przez dr R. Szafrana. Dla geometrii 3D przygotowałem zarówno geometrie, jak i parametry fizyczne symulacji.
- Eksperymentalnej weryfikacji uzyskanych wyników teoretycznych, m.in. wykazaniu, że:
  - dla wielu geometrii rzadkich zaproponowane struktury danych i algorytmy pozwalają na osiągnięcie najwyższej wydajności,
  - rzeczywiste narzuty rozmiaru przesyłanych danych są zgodne z przewidywaniami modelu teoretycznego.

- Postawieniu, i częściowemu potwierdzeniu, hipotezy, że średnia liczba węzłów obliczeniowych w niepustych kafelkach zależy od rozmiarów spójnych obszarów zawierających węzły obliczeniowe, natomiast zależność od porowatości całej geometrii jest wtórna.
- Opiece merytorycznej nad całością artykułu, zdefiniowaniu ogólnej struktury artykułu oraz napisaniu manuskryptu z pominięciem wstępu w zakresie opisu fizyki procesu przedstawionej w rozdz. 2.1 *Basics* i pierwszego zdania „The LBM is a highly parallel alternative to classical Navier-Stokes solvers for computational fluid dynamics (CFD)."

Mój udział procentowy wynosi 75%.

[A2] **T. Tomczak** (75%), R. G. Szafran (25%), luty 2019 r., A new GPU implementation for lattice-Boltzmann simulations on sparse geometries, *Computer Physics Communications*, t. 235, s. 258–278.
IF: 3.748, MNiSW: 45 pkt.

Mój wkład w powstanie tej pracy polegał na:

- Opracowaniu dedykowanej dla procesorów masowo równoległych, w szczególności akceleratorów graficznych, struktury danych wraz z ułożeniem danych w pamięci umożliwiającym ograniczenie liczby transakcji blokowych z pamięcią (*ang. burst memory transactions*) podczas komunikacji pomiędzy sąsiednimi węzłami realizowanej według różnych szablonów (*ang. stencil*) występujących w metodzie siatkowej Boltzmanna. Struktura danych wykorzystuje podział geometrii na niewielkie, sześcienne "kafelki" (*ang. tiles*), natomiast ułożenie danych w pamięci jest zdefiniowane przy pomocy różnych funkcji linearyzujących (*ang. linear mapping functions*) opisanych w rozdz. 3.2 *Tiling memory layout.*
- Opracowaniu dla zaproponowanej struktury danych szybkiej wersji algorytmu realizującego wszystkie kroki metody siatkowej Boltzmanna, w którym wysoka wydajność została osiągnięta m.in. poprzez buforowanie wybranych danych opisujących strukturę geometrii w pamięci notatnikowej (*ang. scratchpad memory*).
- Sformułowaniu uproszczonego modelu złożoności zaproponowanego algorytmu i wykazaniu, że wydajność algorytmu jest liniowo zależna od średniej liczby węzłów obliczeniowych w niepustych kafelkach.
- Zaplanowaniu, przygotowaniu (włącznie z przygotowaniem geometrii i parametrów fizycznych symulacji oraz z zaprojektowaniem i zaimplementowaniem przykładowego rozwiązania w technologii CUDA, które powstało na podstawie poprawnej numerycznie wersji zaimplementowanej przez dr R. Szafrana) i przeprowadzeniu eksperymentów przedstawionych w rozdz. 4 *Results*, opracowaniu i analizie wyników oraz sformułowaniu wniosków.
- Eksperymentalnej weryfikacji uzyskanych wyników teoretycznych, m.in. wykazaniu, że:
  - dla wielu geometrii rzadkich zaproponowane struktury danych i algorytmy pozwalają na osiągnięcie najwyższej wydajności na różnych masowo równoległych procesorach graficznych, zarówno z powodu mniejszego rozmiaru przesyłanych danych, jak i wyższego współczynnika trafień w pamięci podręcznej,
  - wydajność jest wprost proporcjonalna do średniej liczby węzłów obliczeniowych na niepusty kafelek,

  - istnieje niewielka zależność wydajności od liczby krawędzi i ścian wspólnych dla sąsiednich kafelków, położonych na różnych płaszczyznach wyznaczonych przez osie układu współrzędnych, wynikająca z różnych ułożeń danych dla tych ścian/krawędzi, które powodują różne wartości fragmentacji wewnętrznej transakcji blokowych z pamięcią.
- Wykazaniu, na podstawie wstępnej analizy, że jeśli spójne obszary zawierające węzły obliczeniowe mają małe rozmiary w porównaniu z rozmiarem kafelka, to ułożenie kafelków może mieć duży wpływ na średnią liczbę węzłów obliczeniowych w niepustych kafelkach. Jeśli rozmiary spójnych obszarów są duże, wpływ ułożenia kafelków jest znacznie mniejszy.
- Opiece merytorycznej nad całością artykułu, zdefiniowaniu ogólnej struktury artykułu oraz napisaniu manuskryptu z pominięciem wstępu w zakresie opisu fizyki procesu przedstawionej w rozdz. 2.2 *Lattice-Boltzmann Method* i pierwszego zdania "The lattice Boltzmann method (LBM) is a versatile and highly parallel approach where the discrete Boltzmann transport equation is solved in the velocity or moment R^n space to obtain the time-dependent fluid velocity distributions."

Mój udział procentowy wynosi 75%.

[A3] **T. Tomczak** (100%), 2018 r., Lattice Boltzmann Method for Sparse Geometries: Theory and Implementation, rozdział w monografii *Analysis and Applications of Lattice Boltzmann Simulations* pod red. P. Valero-Lara, IGI Global, s. 152–187.
MNiSW: 5 pkt.

Praca zawiera przekrojowy przegląd oraz krytyczną analizę metod obsługi geometrii rzadkich w implementacjach metody siatkowej Boltzmanna na komputerach różnych klas.

Mój udział procentowy wynosi 100%.

[A4] **T. Tomczak** (30%), K. Zadarnowska (20%), Z. Koza (20%), M. Matyka (20%), Ł. Mirosław (10%), 2013 r., Acceleration of iterative Navier-Stokes solvers on graphics processing units, *International Journal of Computational Fluid Dynamics*, t. 27, nr 4/5, s. 201–209.
IF: 0.716, MNiSW: 20 pkt.

Mój wkład w powstanie tej pracy polegał na:

- Opracowaniu formatu danych umożliwiającego reprezentację w pamięci procesora graficznego macierzy rzadkich o nieregularnej strukturze, oraz dedykowanych dla tego formatu, równoległych algorytmów przeglądania macierzy wierszami i kolumnami.
- Opracowaniu, na podstawie zaproponowanego formatu danych, równoległych wersji algorytmów PISO/SIMPLE wykorzystujących równoległość drobnoziarnistą (*ang. fine-grained parallelism*) dla siatek niestrukturalnych.
- Zaprojektowaniu i zaimplementowaniu kompletnej realizacji algorytmów PISO/SIMPLE na akceleratorach graficznych.
- Eksperymentalnej weryfikacji poprawności i wydajności zaproponowanego rozwiązania.
- Przygotowaniu wstępnej wersji rozdz. 3.2 *Data format*.

Mój udział procentowy wynosi 30%.

[A5] Z. Malecha (21%), Ł. Mirosław (15%), **T. Tomczak** (26%), Z. Koza (10%), M. Matyka (21%), W. Tarnawski (5%), D. Szczerba (2%), 2011 r., GPU-based simulation of 3D blood

flow in abdominal aorta using OpenFOAM, *Archives of Mechanics*, t. 63, nr 2, s. 137–161.
IF: 0.396, MNiSW: 13 pkt.

Mój wkład w powstanie tej pracy polegał na:

- Opracowaniu prostej metody akceleracji fragmentu algorytmów PISO/SIMPLE (procedury rozwiązywania układu równań liniowych) przy użyciu masowo równoległych procesorów graficznych. Metoda sprowadzała się do konwersji formatów danych na lepiej dostosowane do procesorów graficznych, wykonaniu obliczeń na akceleratorze i odesłaniu wyników.
- Analizie wydajności obliczeniowej procedury mnożenia macierzy rzadkiej przez wektor przy użyciu procesora graficznego (tab. 1 i rys. 2).
- Analizie wydajności całej procedury rozwiązywania układu równań liniowych przy użyciu procesora graficznego (rys. 3 i tab. 2)

Mój udział procentowy wynosi 26%.

[A6] **T.Tomczak** (100%), 2011 r., Hierarchical residue number systems with small moduli and simple converters, *International Journal of Applied Mathematics and Computer Science*, t. 21, nr 1, s. 173–192.
IF: 0.487, MNiSW: 20 pkt.

Praca zawiera propozycję nowej klasy resztowych systemów liczbowych, które, poprzez użycie modułów będących czynnikami pierwszymi liczb $2^k \pm 1$, pozwalają na połączenie niskiej złożoności cyfrowych układów arytmetycznych wynikającej ze zrównoleglenia operacji składających się na pojedyncze działania arytmetyczne, konwerterów z/na system resztowy, oraz algorytmów wykrywania znaku liczby. Dla zaproponowanych systemów liczbowych podano teoretyczne modele złożoności układów arytmetycznych i konwerterów oraz wyniki eksperymentalnych implementacji.

Mój udział procentowy wynosi 100%.

4.2 Omówienie celu naukowego ww. prac i osiągniętych wyników wraz z omówieniem ich ewentualnego wykorzystania

Przedmiotem moich badań, przedstawionych w ramach powiązanego tematycznie cyklu publikacji [A1]–[A6], jest metodyka zwiększania szybkości przetwarzania obliczeń wysokiej wydajności (*ang. high performance computations*) z wykorzystaniem zrównoleglenia drobnoziarnistego (*ang. fine-grained parallelism*), obejmująca metody operujące zarówno na poziomie oprogramowania, jak i sprzętu. Przez obliczenia wysokiej wydajności należy rozumieć obliczenia, w których celem jest uzyskanie jak największej szybkości przetwarzania (mierzonej np. liczbą wykonanych operacji na jednostkę czasu) powodującej skrócenie czasu (przyspieszenie) potrzebnego do rozwiązania danego problemu (*ang. time-to-solution*).

W zakresie metod dotyczących oprogramowania zajmowałem się opracowywaniem nowych struktur danych, algorytmów i ogólnych koncepcji umożliwiających zwiększanie szybkości przetwarzania poprzez użycie procesorów masowo równoległych, oraz analizą tejże szybkości przetwarzania. Przeprowadzone badania skupiały się na dwóch szeroko stosowanych klasach algorytmów: obliczeniach szablonowych (*ang. stencil computations*) dla geometrii rzadkich (*ang. sparse geometries*) i algorytmach algebry liniowej dla macierzy rzadkich (*ang. sparse matrix*), występujących w procedurach numerycznego całkowania równań różniczkowych dla siatek niestrukturalnych (*ang. unstructured grids*). Do metod na poziomie sprzętu należy zaliczyć prace

dotyczące zwiększenia szybkości wykonywania podstawowych działań arytmetycznych (dodawania/mnożenia) poprzez rozbicie ich na zbiór równolegle wykonywanych operacji wykorzystujących arytmetykę resztową (*ang. residue arithmetic*). Najważniejszymi uzyskanymi wynikami naukowymi są:

- Opracowanie, teoretyczna analiza i praktyczna weryfikacja metody, obejmującej m.in. wykorzystanie specjalizowanych struktur danych i algorytmów, i pozwalającej na zwiększenie szybkości przetwarzania w obliczeniach szablonowych dla geometrii rzadkich. Opracowana metoda pozwoliła na uzyskanie najwyższej na świecie wydajności eksperymentalnej implementacji metody siatkowej Boltzmanna (*ang. lattice Boltzmann method, LBM*) dla wielu geometrii rzadkich.

- Opracowanie, teoretyczna analiza i praktyczna weryfikacja metod obejmujących wykorzystanie struktur danych i algorytmów umożliwiających drobnoziarniste zrównoleglenie kluczowych operacji algebry liniowej dla macierzy rzadkich występujących w procedurach numerycznego całkowania równań różniczkowych cząstkowych przy użyciu np. metody objętości skończonych (*ang. finite volume method, FVM*). Opracowane metody umożliwiły powstanie pierwszej na świecie, eksperymentalnej wersji symulatora wykorzystującego algorytmy rozwiązywania równań Naviera-Stokesa dla siatek niestrukturalnych, w którym wszystkie obliczenia przeprowadzane są przy użyciu procesorów masowo równoległych, w szczególności akceleratorów graficznych, skutkując dużą szybkością przetwarzania.

- Opracowanie, teoretyczna analiza i eksperymentalna weryfikacja metody zwiększania szybkości i zmniejszania zajmowanego obszaru układów cyfrowych wykonujących podstawowe operacje arytmetyczne poprzez zastosowanie hierarchicznych resztowych systemów liczbowych (*ang. hierarchical residue number systems, HRNS*) skonstruowanych na bazie modułów postaci $2^n \pm 1$ rozkładalnych na niewielkie czynniki pierwsze. Zaproponowane HRNS umożliwiają połączenie dużej szybkości i niewielkiego rozmiaru układów arytmetycznych z niewielką złożonością operacji "trudnych", m.in. wykrywania znaku liczby oraz konwersji pomiędzy HRNS a systemami wagowymi.

Wyniki te są omawiane poniżej odpowiednio w rozdz. 4.2.1.1, 4.2.1.2 i 4.2.2.

4.2.1 Programowe metody zwiększania szybkości przetwarzania

Z powodów technologicznych i ekonomicznych, szybkość pamięci, określona przez przepustowość magistrali i czas trwania pojedynczych transferów, rośnie wolniej, niż szybkość przetwarzania scalonych, cyfrowych układów obliczeniowych. Stąd, w porównaniu z parametrami dostępnych maszyn obliczeniowych, coraz więcej algorytmów stosowanych w obliczeniach wysokiej wydajności charakteryzuje się dużym stosunkiem liczby przesyłanych danych do liczby wykonywanych obliczeń. Szybkość implementacji takich algorytmów jest więc ograniczona przez przepustowość magistrali pamięci (*ang. bandwidth-bound algorithms*). Dodatkowo, w przypadku algorytmów umożliwiających np. szeroko rozumiane symulacje, m.in. z zakresu obliczeniowej mechaniki płynów (*ang. computational fluid dynamics, CFD*), w rzeczywistych przypadkach konieczne jest stosowanie rozwiązań umożliwiających uzyskanie dużej szybkości przetwarzania dla danych wejściowych o nieregularnej strukturze, opisujących np. skomplikowaną geometrię, dla której przeprowadzane są symulacje. Biorąc pod uwagę ograniczenia pamięci DRAM (*ang. dynamic random-access memory*), zagadnienie to jest zagadnieniem trudnym ze względu na dużą nieregularność używanych struktur danych, która dodatkowo silnie zależy od konkretnego symulowanego przypadku.

Obecnie stosowane maszyny obliczeniowe wysokiej wydajności wykorzystują pamięci dynamiczne DRAM o konstrukcji optymalizowanej pod kątem uzyskania wysokiej przepustowości,

m.in. poprzez wspieranie transferów blokowych (*ang. burst transfer*). Jednocześnie, wysokie czasy trwania pojedynczych transferów powodują, że utrzymanie wysokiej przepustowości na magistrali pamięci możliwe jest jedynie poprzez nakładanie na siebie wielu niezależnych transferów. Z tego powodu efektywna, równoległa realizacja wielu algorytmów spotykanych w obliczeniach wysokiej wydajności wymaga stosowania takich struktur danych i algorytmów równoległych, których połączenie umożliwia zarówno przeprowadzanie wielu niezależnych transferów z/do pamięci, jak i wykorzystanie wszystkich danych przesyłanych w każdym transferze blokowym. Niewłaściwa konstrukcja struktur danych może spowodować, że zależności pomiędzy danymi będą utrudniać równoczesne wykonanie odpowiedniej liczby niezależnych transferów z/do pamięci. W takim przypadku wydajność algorytmu staje się zależna od czasu trwania pojedynczego transferu (*ang. latency-bound algorithms*), co zazwyczaj oznacza znacznie niższą wydajność, niż w sytuacji, gdy jest ona ograniczona przez przepustowość magistrali pamięci.

Problemy związane z niewłaściwą strukturą danych mogą wpływać na zmniejszenie szybkości przetwarzania nie tylko z powodu budowy podsystemu pamięci, ale także ze względu na cechy architektury procesorów stosowanych w obliczeniach wysokiej wydajności. Pomimo wysokiej teoretycznej mocy obliczeniowej dostępnej w obecnie stosowanych procesorach, efektywne jej wykorzystanie może być trudne ze względu na konstrukcję jednostek obliczeniowych. Większość procesorów stosowanych w obliczeniach wysokiej wydajności wykorzystuje jednostki obliczeniowe zbudowane na podstawie koncepcji opracowanych na potrzeby procesorów wektorowych. Przykładami mogą być wbudowane koprocesory SIMD (*ang. Single Instruction Multiple Data*), np. rozszerzenia SSE/AVX w procesorach rodziny x86, czy nawet konstruowane na bazie procesorów graficznych (*ang. graphics processing unit, GPU*) masowo równoległe akceleratory obliczeń, które można traktować jako kolejny etap rozwoju klasycznych procesorów wektorowych. Cechą charakterystyczną tych rozwiązań jest model przetwarzania oparty o równoległość na poziomie danych (*ang. data-level parallelism*), którą może być trudno uzyskać dla nieregularnych danych wejściowych zawierających dodatkowo wzajemne zależności. Dodatkowym utrudnieniem mogą być szeroko stosowane sprzętowe techniki zwiększania liczby równocześnie wykonywanych operacji poprzez równoległość na poziomie instrukcji (*ang. instruction-level parallelism, ILP*), np. przetwarzanie potokowe czy superskalarne, a także wykonywanie poza kolejnością (*ang. out-of-order execution*) czy spekulatywne (*ang. speculative execution*), które nakładają ograniczenia na zależności pomiędzy sąsiednimi instrukcjami w wykonywanym programie.

Wymienione powyżej cechy powodują, że przy obecnych ograniczeniach technologicznych osiągnięcie dużej szybkości przetwarzania wymaga opracowania szybkich algorytmów obliczeniowych wykorzystujących równoległość drobnoziarnistą (*ang. fine-grained parallelism*). Oznacza to konieczność podziału problemów na bardzo krótkie, równolegle wykonywane zadania, które w skrajnych przypadkach mogą ograniczać się do pojedynczych instrukcji, lub nawet ich fragmentów.

Prowadzone przez mnie badania w zakresie zwiększania szybkości przetwarzania w algorytmach o wydajności ograniczonej przepustowością magistrali pamięci dotyczyły możliwości drobnoziarnistego zrównoleglenia dedykowanych dla geometrii rzadkich zarówno algorytmów obliczeń szablonowych, jak i algorytmów algebry liniowej dla macierzy rzadkich występujących w procedurach numerycznego całkowania równań różniczkowych. Geometrią rzadką nazywa się taką geometrię, w której obliczenia wykonuje się jedynie dla niewielkiej części symulowanej objętości. Ze względu na występujące dla geometrii rzadkich problemy z jednoczesnym zapewnieniem niskiego zapotrzebowania na pamięć oraz wysokiej lokalności przestrzennej i czasowej podczas dostępu do danych opisujących geometrię, przy jednoczesnej konieczności zastosowania równoległości drobnoziarnistej w celu efektywnego wykorzystania równoległych jednostek obliczeniowych dostępnych w procesorach, tematyka ta jest uznawana na ogół za trudną. Jednocześnie, bardzo wiele rzeczywistych obszarów zastosowań obliczeń wysokiej wydajności wymaga

używania geometrii rzadkich.

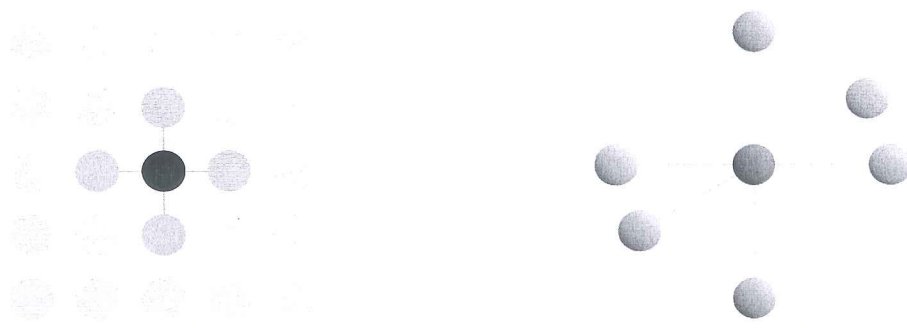
Jako przykładu nietrywialnych obliczeń szablonowych użyto metody siatkowej Boltzmanna (*ang. lattice Boltzmann method, LBM*), natomiast szybkie algorytmy numerycznego całkowania omówiono na przykładzie popularnych metod CFD: PISO (*ang. Pressure-Implicit with Splitting of Operators*) i SIMPLE (*ang. Semi-Implicit Method for Pressure Linked Equations*). Wyniki te opisano poniżej odpowiednio w rozdz. 4.2.1.1 i 4.2.1.2.

4.2.1.1 Metody zwiększania szybkości przetwarzania w obliczeniach szablonowych dla geometrii rzadkich

Do klasy obliczeń szablonowych (*ang. stencil computations*) można zaliczyć wszystkie algorytmy, w których elementy pewnej wielowymiarowej tablicy są iteracyjnie uaktualniane na podstawie wartości elementów sąsiednich wybranych według pewnego szablonu. Typowymi przykładami mogą być algorytmy cyfrowego przetwarzania obrazów, automaty komórkowe, oraz wiele algorytmów numerycznych, szczególnie dla regularnej dyskretyzacji przestrzennej domeny obliczeniowej. Przykłady szablonów obejmujących najbliższe sąsiedztwo przetwarzanego elementu przedstawiono na rys. 1.

Zaletami obliczeń szablonowych, szczególnie dla geometrii gęstych, jest możliwość zagwarantowania bardzo dobrego wykorzystania dostępnej przepustowości pamięci dzięki regularnym wzorcom dostępu do pamięci, ponieważ położenie elementu w pamięci jest ściśle powiązane z jego współrzędnymi w tablicy. Przegląd technik efektywnych implementacji obliczeń szablonowych dla geometrii gęstych można znaleźć np. w pracy [1], znane są także zaawansowane techniki efektywnych implementacji dla procesorów masowo równoległych (por. [2]) czy automatycznej generacji implementacji optymalnych w różnym sensie (np. [3, 4]).

W przypadku geometrii rzadkich, zazwyczaj przechowuje się jedynie wartości elementów biorących udział w obliczeniach, co powoduje, że utracone zostaje powiązanie pomiędzy położeniem elementu w pamięci, a położeniem w tablicy. Uzyskanie wysokiej wydajności równoległych implementacji obliczeń szablonowych dla geometrii rzadkich jest więc dość trudne ze względu na konieczność użycia jakiejś formy adresowania pośredniego (*ang. indirect addressing*) pozwalającej na dostarczenie informacji o położeniu elementów sąsiednich (por. [5–11]).



(a) D2Q5 (b) D3Q7

Rysunek 1: Przykłady szablonów 2- i 3-wymiarowych. Oznaczenia szablonów są zgodne z konwencją stosowaną w metodzie siatkowej Boltzmanna.

4.2.1.1.1 Metoda siatkowa Boltzmanna

Rozpatrywanym przeze mnie przykładem obliczeń szablonowych jest, często stosowana dla geometrii rzadkich, metoda siatkowa Boltzmanna (*ang. lattice Boltzmann method, LBM*), będąca metodą z dziedziny obliczeniowej mechaniki płynów. W metodzie siatkowej Boltzmanna stosuje się podejście oparte na automatach komórkowych. Obliczenia przeprowadza się dla równomiernej (np. kartezjańskiej) siatki „węzłów". Z każdym węzłem związany jest wektor zmiennych opisujących prawdopodobieństwo wystąpienia umownych „cząstek" o zadanych wektorach prędkości. W kolejnych iteracjach algorytmu LBM oblicza się nowe wartości zmiennych w węźle na podstawie wartości z węzłów sąsiednich.

Dotychczas stosowane rozwiązania szybkich algorytmów LBM dla geometrii rzadkich na procesory masowo równoległe, w szczególności akceleratory graficzne, wykorzystywały struktury danych oparte o adresowanie pośrednie: macierz połączeń (*ang. Connectivity Matrix, CM*), zaprezentowaną w pracach [7–10], oraz tablicę indeksów (*ang. Fluid Index Array, FIA*), użytą w pracy [11]. W metodach tych wydajność jest ograniczona zarówno przez konieczność przesyłania dodatkowych danych opisujących strukturę geometrii, jak i przez nieregularne ułożenie danych w pamięci powodujące występowanie wielu transakcji z pamięcią, w których przesyłane jest wiele nieużywanych danych. Dla maszyn zbudowanych w postaci klastrów komputerów, stosowane są także techniki podziału domeny obliczeniowej wykorzystujące zagadnienia teorii grafów, krzywe wypełniające (*ang. space-filling curves*), drzewa ósemkowe oraz różne algorytmy heurystyczne.

W obszarze struktur danych i algorytmów pozwalających na akcelerację metody siatkowej Boltzmanna dla geometrii rzadkich moje badania skupiały się głównie na metodach minimalizacji liczby przesyłanych danych poprzez:

- znaczne (kilkudziesięciokrotne) ograniczenie rozmiaru dodatkowych struktur danych przechowujących informacje o geometrii,
- specjalizowane metody ułożenia danych w pamięci gwarantujące w większości sytuacji minimalną liczbę transakcji z pamięcią w trakcie realizacji algorytmu LBM,
- algorytmy dostępu do danych wykorzystujące buforowanie informacji o strukturze geometrii w pamięci notatnikowej (*ang. scatchpad memory*).

Zostało to osiągnięte dzięki opracowaniu struktur danych wykorzystujących podział dziedziny obliczeniowej na równomierną siatkę niewielkich kwadratów bądź sześcianów nazywanych "kafelkami" (*ang. tiles*) o rozmiarze dostosowanym do parametrów procesora: liczby jednocześnie wykonywanych operacji w jednostkach obliczeniowych SIMD/SIMT (*ang. Single Instruction Multiple Data, Single Instruction Multiple Threads*) oraz rozmiaru pojedynczej transakcji z pamięcią. Ustalony rozmiar kafelka pozwala dodatkowo na optymalizację ułożenia w pamięci danych związanych z węzłami z wnętrza pojedynczego kafelka, co skutkuje zwiększeniem szybkości przetwarzania dzięki lepszemu wykorzystaniu przepustowości magistrali pamięci. Wyniki zostały zaprezentowane w pracach [A1]-[A3].

Zaproponowana przeze mnie struktura danych pozbawiona jest wad spowodowanych adresowaniem pośrednim, umożliwiając dla wielu geometrii osiągnięcie znacznie wyższej wydajności. Wysoka wydajność wynika m.in. z nowej, specjalizowanej metody ułożenia danych opisanej w pracy [A2]. Dodatkowo, praca [A1] zawiera analizy złożoności obliczeniowej i pamięciowej o poziomie szczegółowości nie spotykanym wcześniej w literaturze, które umożliwiają wiarygodne porównywanie różnych metod obsługi geometrii rzadkich. W pracy [A3] przedstawiłem także obszerny, krytyczny przegląd dotychczasowych rozwiązań szybkich algorytmów LBM dla geometrii rzadkich na różnych klasach komputerów.

4.2.1.1.2 Teoretyczny model złożoności

W teoretycznych analizach złożoności algorytmów LBM, np. [12], często stosuje się prosty model maszyny obliczeniowej zdolnej do wykonania podstawowych operacji arytmetycznych, przesyłania pojedynczych bajtów z/do pamięci operacyjnej oraz wyposażonej w szybką pamięć wbudowaną (rejestry, pamięć podręczną i/lub notatnikową). Dla implementacji algorytmu LBM na takiej maszynie można określić, w przeliczeniu na pojedynczy węzeł geometrii, minimalne teoretyczne zapotrzebowanie na pamięć M_{node} i minimalny rozmiar przesyłanych danych B_{node} dla q zmiennych w węźle, z których każda zajmuje s_d bajtów, jako

$$M_{node} = q \cdot s_d, \quad B_{node} = 2 \cdot q \cdot s_d. \tag{1}$$

Używając dodatkowo zaprezentowanego w [13] modelu "zrównoważenia" (*ang. balance*) można wykazać, że dla obecnie dostępnych maszyn wydajność implementacji wielu modeli obliczeniowych stosowanych w LBM jest ograniczona wyłącznie przez rozmiar przesyłanych danych (por. [A1], [A2]).

W pracach [A1] i [A2] wykorzystałem powyższy model rozszerzając go o pojęcie "narzutu" (*ang. overhead*) Δ spowodowanego koniecznością przechowywania i przesyłania dodatkowych danych w celu obsługi geometrii rzadkich. Narzut ten został zdefiniowany jako stosunek rozmiaru danych lub liczby operacji dodatkowych do minimalnego rozmiaru danych lub minimalnej liczby operacji opisanych równaniem (1). Pozwoliło to wyznaczyć teoretyczny stosunek złożoności algorytmu z narzutem do złożoności algorytmu bez narzutu (np. wersji dla geometrii gęstych) jako

$$P = \frac{1}{1+\Delta}. \tag{2}$$

Narzut ten *de facto* określa granicę wydajności implementacji dla geometrii rzadkich w stosunku do wydajności idealnej implementacji dla geometrii gęstych (bez narzutów). Porównanie złożoności różnych wersji algorytmów dla geometrii rzadkich można więc sprowadzić do porównywania narzutów.

Przeprowadzona w pracy [A1] analiza pozwoliła określić narzuty liczby przesyłanych danych dla technik CM i FIA jako

$$\Delta^B_{CM} = \frac{(q-1)\cdot s_{CM\_idx}}{B_{node}} \quad \text{oraz} \quad \Delta^B_{FIA} = \frac{s_{FIA\_idx}}{\phi \cdot B_{node}} + 1, \tag{3}$$

gdzie s_{CM_idx} i s_{FIA_idx} oznaczają rozmiary dodatkowych indeksów używanych w metodach CM/FIA (zazwyczaj 4 bajty), natomiast ϕ oznacza porowatość (*ang. porosity*) geometrii określoną jako stosunek liczby węzłów, dla których należy przeprowadzić obliczenia, do liczby wszystkich węzłów. W podobny sposób określiłem także narzuty złożoności pamięciowej Δ^M. Uzyskane wartości narzutów ($\Delta^B_{CM} \gtrsim 0.25$, $\Delta^B_{FIA} > 1$) pokazały, że dla wielu typowych przypadków możliwe jest zwiększenie wydajności symulacji dla geometrii rzadkich, tym bardziej, że użyty model pomija dodatkowe narzuty spowodowane fragmentacją wewnętrzną blokowych transakcji z pamięcią powstającą wskutek nieoptymalnego rozłożenia danych w pamięci.

4.2.1.1.3 Opracowanie nowej struktury danych

Ponieważ narzuty opisane równaniem (3) są spowodowane m.in. przez dodatkowe indeksy o rozmiarach s_{CM_idx} i s_{FIA_idx}, w celu ich wyeliminowania zaproponowałem strukturę danych wykorzystującą podział geometrii na równomierną siatkę kafelków o jednakowym rozmiarze [A1,A2]. Zaletami tej struktury danych są:

- Wielokrotne (kilkudziesięcio- do ponad dwustukrotne dla przeanalizowanych rozmiarów kafelków) zmniejszenie rozmiaru dodatkowych danych przechowujących informacje o geometrii, ponieważ potrzebna jest jedynie wiedza o wzajemnym ułożeniu kafelków.
- Możliwość przechowywania zmiennych dla węzłów wewnątrz kafelka w zwykłych, wielowymiarowych tablicach, w których indeks elementu jest wyliczany bezpośrednio na podstawie współrzędnych węzła w kafelku (bez dodatkowych narzutów spowodowanych adresowaniem pośrednim).
- Możliwość zastosowania różnych ułożeń danych dla węzłów wewnątrz kafelka, ponieważ wszystkie kafelki mają jednakowy rozmiar dostosowany m.in. do rozmiaru transakcji blokowych z pamięcią. Umożliwia to lepsze wykorzystanie transakcji blokowych z pamięcią poprzez zmniejszenie fragmentacji wewnętrznej tych transakcji.

Do wad tego rozwiązania należy zaliczyć:

- Konieczność dostosowania rozmiaru kafelka do parametrów maszyny, co powoduje, że tylko kilka rozmiarów ma praktyczne zastosowanie, np. 4^3 czy 16^2 węzłów. Rozmiar kafelka powinien umożliwiać:
  - przyporządkowanie węzłów w kafelku do jednostek obliczeniowych pozwalające zminimalizować liczbę nieużywanych jednostek,
  - ułożenie w pamięci danych dla węzłów z wnętrza kafelka umożliwiające zminimalizowanie liczby transakcji z pamięcią poprzez minimalizację fragmentacji wewnętrznej tych transakcji,
  - jednoczesne przetwarzanie przez procesor wystarczająco dużej liczby węzłów pozwalające, zgodnie z prawem Little'a [14], wykorzystać większość dostępnej przepustowości magistrali pamięci.
- Konieczność komunikacji pomiędzy węzłami z sąsiednich kafelków uwzględniającej problemy spowodowane równoległym przetwarzaniem kafelków, np. w maszynach z wykonywaniem poza kolejnością (*ang. out-of-order execution*) i/lub ze słabym modelem uporządkowania pamięci (*weakly-ordered memory model*).
- Szybki wzrost narzutów w funkcji liczby nieobliczeniowych węzłów w kafelku.

4.2.1.1.4 Teoretyczna analiza złożoności zaproponowanej struktury danych

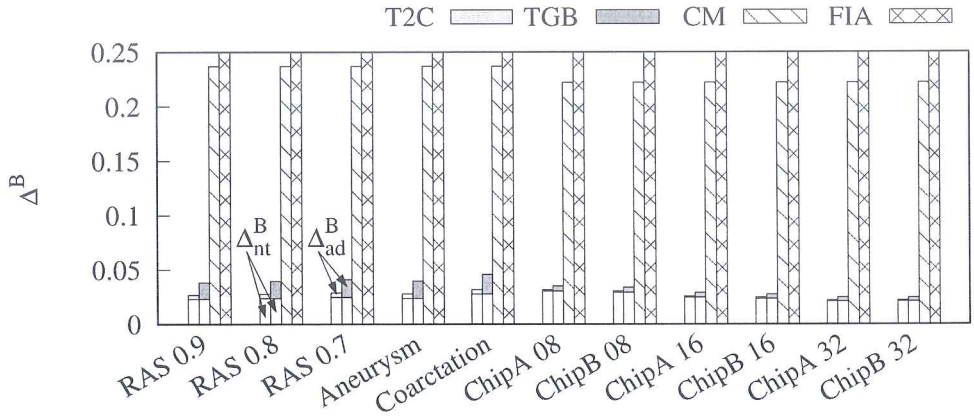
W pracy [A1] przeprowadziłem dla zaproponowanej struktury danych szczegółową, teoretyczną analizę narzutów rozmiaru przesyłanych danych i złożoności pamięciowej dla dwóch sposobów komunikacji pomiędzy węzłami z sąsiednich kafelków: bezpośrednim dostępem do węzłów w sąsiednich kafelkach (T2C) i dostępem z użyciem buforów na krawędziach kafelków (TGB) uzyskując zależności narzutów rozmiaru przesyłanych danych postaci

$$\Delta^B_{T2C} = \frac{(a+2)^d \cdot s_t + (q-1) \cdot s_{ti}}{a^d \cdot \phi_t \cdot 2 \cdot q \cdot s_d} \tag{4}$$

i

$$\Delta^B_{TGB} = \frac{(a+2)^d \cdot s_t + C_{gbi} \cdot s_{gbi}}{a^d \cdot \phi_t \cdot 2 \cdot q \cdot s_d}, \tag{5}$$

gdzie a oznacza długość krawędzi kafelka w węzłach, d oznacza liczbę wymiarów geometrii (2 lub 3), s_t oznacza rozmiar w bajtach dodatkowej zmiennej przechowującej typ węzła, s_{ti} i s_{gbi}



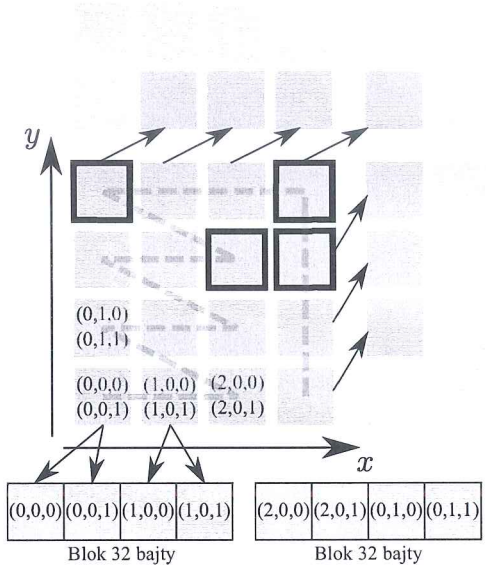
Rysunek 2: Teoretyczne oszacowania narzutów rozmiaru przesyłanych danych dla różnych metod obsługi geometrii rzadkich: kafelków odpowiednio z dostępem bezpośrednim do węzłów w sąsiednim kafelku (T2C) i z komunikacją przy użyciu dodatkowych buforów na krawędziach (TGB), macierzy połączeń (CM) oraz tablicy indeksów (FIA). Narzuty dla FIA przekraczają 1.0 we wszystkich przypadkach. Δ^B_{nt} oznacza narzuty wynikające z przesyłania zmiennych opisujących umowny typ węzła, Δ^B_{ad} oznacza narzuty spowodowane wszystkimi pozostałymi czynnikami.

oznaczają odpowiednio rozmiary w bajtach wskaźników na sąsiednie kafelki lub bufory, C_{gbi} jest stałą zależną od przyjętego szablonu komunikacji i przyjmującą wartości pomiędzy kilkadziesiąt a kilkaset, natomiast ϕ_t jest średnią porowatością kafelka zdefiniowaną jako stosunek liczby węzłów obliczeniowych do liczby wszystkich węzłów w niepustych kafelkach. Ponieważ zazwyczaj większość parametrów (z pominięciem ϕ_t) można potraktować jako stałe zależne od użytej wersji LBM, w typowych sytuacjach wartości $\Delta^B_{T2C}, \Delta^B_{TGB}$ są z zakresu $\left\langle \frac{0.02}{\phi_t}, \frac{0.04}{\phi_t} \right\rangle$, co oznacza, że zaproponowana struktura danych pozwala zmniejszyć narzuty do poziomu pojedynczych procentów umożliwiając osiągnięcie wydajności dla geometrii rzadkich zbliżonej do wydajności idealnych implementacji dla geometrii gęstych. Porównanie narzutów różnych metod obsługi geometrii rzadkich dla rzeczywistych geometrii dwu- i trójwymiarowych zaprezentowano na rys. 2 (por. [A1]).

Niestety, przyjmując dodatkowo bardziej realistyczny model maszyny, w którym odczyty i zapisy pamięci odbywają się wyłącznie poprzez transakcje blokowe o rozmiarze s_b bajtów, należy pesymistycznie założyć, że podczas obliczeń konieczne będzie przesłanie wszystkich danych kafelka, czyli zarówno wszystkich danych wewnątrz kafelka, jak i np. wszystkich buforów na krawędziach, ponieważ nie ma możliwości pominięcia bajtów znajdujących się w tej samej transakcji blokowej, co potrzebna dana. Daje to pesymistyczne oszacowania Δ^B od kilku do kilkudziesięciu razy większe niż wartości wynikające ze wzorów (4) i (5), co pokazuje, że dla osiągnięcia wysokiej wydajności wynikającej z małych narzutów rozmiaru przesyłanych danych konieczne jest zastosowanie takiego ułożenia danych kafelka w pamięci, które zminimalizuje rozmiar przesyłanych danych nadmiarowych.

4.2.1.1.5 Metody ułożenia danych w pamięci

Wymagane metody ułożenia danych w pamięci, umożliwiające ograniczenie rozmiaru przesyłanych danych nadmiarowych poprzez lepsze wykorzystanie blokowych transakcji z pamięcią, opracowałem i opisałem w pracy [A2]. Dla zmiennych przesyłanych pomiędzy węzłami sąsiadującymi w tym samym wierszu/kolumnie zastosowałem standardowe uporządkowanie wiersza-



Rysunek 3: Przykładowe ułożenie danych pojedynczego kafelka w pamięci optymalizowane dla komunikacji pomiędzy węzłami wzdłuż przekątnej zgodnie z wektorem $\langle x, y\rangle = \langle 1, 1\rangle$. W nawiasach umieszczono współrzędne (x_t, y_t, z_t) węzłów wewnątrz kafelka. Szare kwadraty oznaczają dane dla poszczególnych węzłów w jednym plastrze kafelka zawierającego 4^3 węzłów, przerywana linia oznacza uporządkowanie danych w pamięci. Pogrubione, czarne krawędzie oznaczają dane wewnątrz transakcji blokowych wykorzystanych jedynie częściowo.

mi/kolumnami (*ang. row/column-major order*), natomiast dla zmiennych przesyłanych wzdłuż przekątnej zaproponowałem ułożenie pokazane na rys. 3. Istotnymi zaletami tego ułożenia są:

- Niewielki rozmiar nadmiarowych danych przesyłanych podczas dostępu do sąsiadów po przekątnej.
- Stosunkowo proste wyliczanie adresu za pomocą funkcji linearyzującej (*ang. linear mapping function*) postaci np.:

$$L(x_t, y_t, z_t) = 2 \cdot \Big(x_t + 3 \cdot y_t + ((x_t + 1) \cap 4) \cdot (3 - y_t)\Big) + (z_t \cap 1) + 4^2 \cdot (z_t \cap 2), \qquad (6)$$

  w której wykorzystywane są wyłącznie podstawowe operacje arytmetyczne i logiczne, a x_t, y_t i z_t oznaczają współrzędne węzła w kafelku.

4.2.1.1.6 Szybkie wersje algorytmów metody siatkowej Boltzmanna

Dla zaproponowanych formatów danych opracowałem szybkie algorytmy realizujące obliczenia zgodnie z metodą siatkową Boltzmanna, w których wysoka wydajność została osiągnięta dzięki użyciu właściwej kolejności dostępu do danych w pamięci, buforowaniu w pamięci notatnikowej (*ang. scratchpad memory*) danych opisujących położenie kafelków, małej liczbie punktów synchronizacji (szczególnie dla wersji z bezpośrednim dostępem do węzłów w sąsiednich kafelkach), oraz konstrukcji umożliwiającej wykorzystanie typowych optymalizacji dla procesorów masowo-równoległych: grupowania operacji z wykorzystaniem równoległości na poziomie instrukcji czy wykonywania czasochłonnych operacji (np. dostępu do pamięci) poza fragmentami kodu dywergentnego. Pseudokod algorytmów zamieszczono na rys. 4 (por. [A1], [A2]), praca [A2] zawiera dodatkowo schematy dostępu do danych podczas konstruowania lokalnych kopii informacji o położeniu kafelków.

```
1: load copy of tile bitmap {3^3 values}
2: load node types from current tile {4^3 values}
3: load node types from neighbor tiles {WLP}
4: BARRIER
5: if node not solid then
6:    load f_0
7:    for i ∈< 1..18 > do
8:       compute address of neighbor node in direction i
9:       if neighbor node not solid then
10:         gather f_i from neighbor node
11:      end if
12:   end for{all f_i copied to registers}
13:   process boundary {calculate v, ρ}
14:   collide
15:   store all f_i {all coalesced}
16: end if
```

(a) T2C

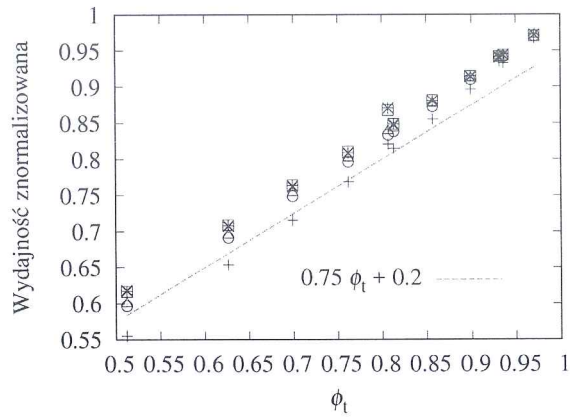
```
1: load node type
2: if node not solid then
3:    load all f_i
4: end if
5: load ghost buffers {2 BARRIERS due to WLP}
6: if node not solid then
7:    process boundary nodes {calculate u, v, ρ}
8:    if node is boundary then
9:       store all f_i
10:   end if
11:   collide
12: end if
13: store ghost buffers {1 BARRIER due to WLP}
14: if node not solid then
15:   scatter all f_i
16: end if
```

(b) TGB

Rysunek 4: Pseudokod szybkich algorytmów LBM dla a) kafelków z dostępem bezpośrednim do węzłów w sąsiednich kafelkach (T2C), b) kafelków z komunikacją poprzez dodatkowe bufory na krawędziach (TGB). Wartości w komentarzach dotyczą kafelków o rozmiarze 4^3 węzłów, WLP oznacza programowanie z wykorzystaniem niejawnej synchronizacji (*ang. implicit synchronization*) wątków w pojedynczym splocie (*ang. Warp-Level Programming*).

4.2.1.1.7 Eksperymentalna analiza wydajności

Wydajność zaproponowanych struktur danych i algorytmów została zbadana zarówno teoretycznie, jak i eksperymentalnie. Dla przykładu, dla trójwymiarowych geometrii obliczona teoretycznie liczba 32-bajtowych transakcji dla zaproponowanych struktur danych jest ok. 6.6% wyższa, niż wynikająca z B_{node} (por. [A2] rozdz. 3.2, gdzie obliczony narzut w stosunku do liczby odczytów z pominięciem zapisów wynosi 13%), natomiast zmierzone rzeczywiste liczby transakcji dla trójwymiarowych geometrii rzadkich odpowiadają narzutom w zakresie od 6.2% do 7.5% (por. [A1] tab. 5). Choć wartości te są ponad 2 razy większe, niż minimum wynikające ze wzoru (4), nadal są znacznie mniejsze, niż dla metod CM i FIA, oraz niż pesymistyczne oszacowania. Dodatkowo, osiągnięta rzeczywista wydajność jest znacznie wyższa, niż dla innych technik: dla geometrii trójwymiarowych i tych samych modeli LBM kafelki umożliwiają osiągnięcie wydajności na poziomie 0.6 maksymalnej teoretycznej wydajności obliczonej przy wykorzystaniu całej dostępnej przepustowości pamięci, natomiast najlepsze implementacje metody CM uzyskują pomiędzy 0.4 a 0.5, a FIA ok. 0.2 maksimum. Co więcej, dla geometrii gęstych rzeczywista, zmierzona liczba transakcji dla kafelków jest jedynie ok. 1% większa, niż wynikająca ze wzoru (4). Wynika to przede wszystkim z wysokiego współczynnika trafień w pamięci podręcznej wskutek lokalności przestrzennej i czasowej odwołań do pamięci. Analiza wpływu ułożenia danych w pamięci na wydajność wykazała, że zastosowanie zaproponowanych struktur danych powoduje, w stosunku do klasycznego ułożenia wszystkich danych wierszami (*ang. row-major order*), zarówno kilkunastoprocentowy wzrost wydajności, jak i ponad dwukrotny wzrost współczynnika trafień w pamięci podręcznej (por. [A2] rozdz. 4.5). Wyniki eksperymentów pokazały także niewielkie zmiany wydajności spowodowane różnym udziałem w globalnym czasie wykonania komunikacji pomiędzy kafelkami stykającymi się krawędziami i ścianami położonymi na różnych płaszczyznach wyznaczonych przez osie układu współrzędnych (por. [A2] rozdz. 4.4). Zmiany te wynikają z różnej liczby transakcji z pamięcią dla ścian/krawędzi położonych na różnych płaszczyznach, ponieważ dla różnych płaszczyzn stosowane są różne ułożenia danych.

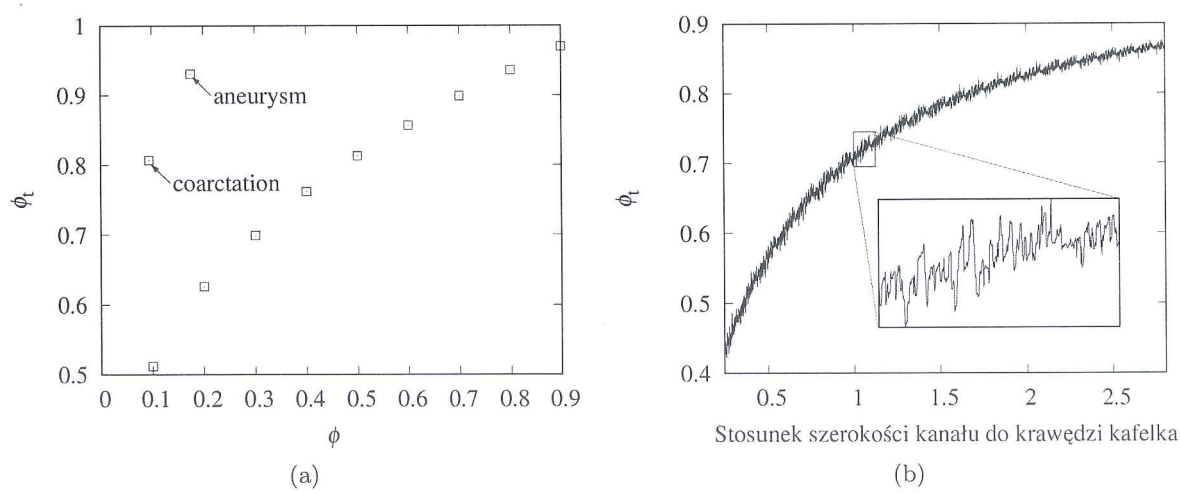


Rysunek 5: Znormalizowana wydajność implementacji różnych modeli LBM (oznaczonych znacznikami o różnych kształtach) w funkcji średniej porowatości kafelka. Dane eksperymentalne dla obliczeń podwójnej precyzji na procesorze Tesla P100 i 11 różnych geometrii o porowatościach $\phi \in [0.09, 0.9]$ i $\phi_t \in [0.51, 0.97]$.

4.2.1.1.8 Uproszczone oszacowanie wydajności

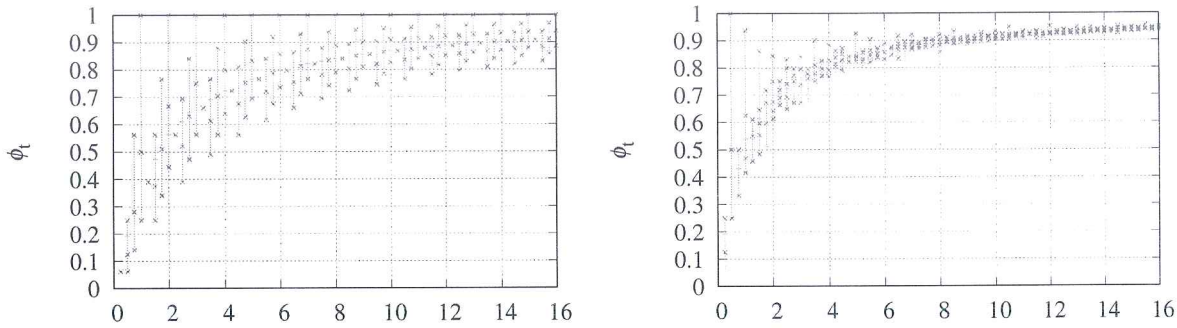
Oprócz zaproponowanych efektywnych ułożeń danych w pamięci oraz szczegółowego modelu opisującego narzuty, w pracy [A2] wykazałem także, że w wielu przypadkach narzuty dla zaproponowanych struktur danych i algorytmów mogą być oszacowane wyłącznie na podstawie wartości średniej porowatości kafelka ϕ_t przy użyciu uproszczonej zależności

$$\Delta_{\phi_t} \approx \frac{1}{\phi_t} - 1. \tag{7}$$



(a) (b)

Rysunek 6: Zależność średniej porowatości kafelka ϕ_t od: a) porowatości geometrii ϕ dla 9 geometrii zawierających losowo rozmieszczone kule oraz dwóch geometrii opisujących układ naczyń krwionośnych ("coarctation" i "aneurysm"), b) szerokości dwuwymiarowych kanałów w geometrii o porowatości $\phi = 0.2$ próbkowanej z rozdzielczościami od 1000×630 do 11500×7300 węzłów i kafelków o rozmiarze 16^2 węzłów.



(a) Kanał o przekroju kwadratowym (b) Kanał o przekroju kolistym

Rysunek 7: Zależność średniej porowatości kafelka ϕ_t od rozmiarów nieskończenie długich kanałów biegnących wzdłuż jednej z osi układu współrzędnych dla wszystkich możliwych ułożeń kafelków o rozmiarze 4^3 węzłów.

Pozwala to określić stosunek wydajności algorytmów dla struktur danych wykorzystujących podział geometrii rzadkiej na kafelki do wydajności idealnej implementacji dla geometrii gęstej jako

$$P = \frac{1}{1 + \Delta_{\phi_t}} \approx \phi_t. \tag{8}$$

Wynik ten został potwierdzony eksperymentalnie w pracy [A2], gdzie wykazano liniową zależność wydajności od średniej porowatości kafelka (rys. 5).

Udowodnienie wiodącego wpływu średniej porowatości kafelka ϕ_t na wydajność umożliwia nie tylko proste oszacowanie wydajności algorytmu dla dowolnej geometrii rzadkiej na postawie wyłącznie wartości ϕ_t, ale też pokazuje, że bardzo istotne dla osiągnięcia wysokiej wydajności jest zapewnienie wysokiej wartości ϕ_t, np. poprzez dobór rozmiaru kafelka czy ułożenia kafelków zapewniającego minimalną ich liczbę. Problem zapewnienia wysokiej wartości ϕ_t pozostaje niestety otwarty, konieczne są dalsze badania w tym kierunku. Dotychczasowe rezultaty wydają się jednak dość obiecujące - nawet dla geometrii o bardzo niskiej porowatości $\phi = 0.09$ możliwe jest uzyskanie wysokiej średniej porowatości kafelka $\phi_t = 0.81$. Przeprowadzone w pracach [A1] i [A2] wstępne analizy wykazały, że co prawda można zaobserwować pewne powiązanie pomiędzy wartościami ϕ_t oraz ϕ (por. rys. 6a), wydaje się jednak, że decydujące znaczenie mają inne czynniki, np. stosunek rozmiarów spójnych obszarów wypełnionych węzłami obliczeniowymi do rozmiarów kafelka (rys. 6b). Wyniki zamieszczone w pracy [A2] pokazują także, że dla niewielkich rozmiarów obszarów wypełnionych węzłami obliczeniowymi duże znaczenie ma też rozmieszczenie kafelków (por. rys. 7 oraz [A2] rozdz. 3.3).

4.2.1.1.9 Wykorzystanie rezultatów

Jako że dla opracowanych struktur danych i algorytmów wydajność zależy praktycznie wyłącznie od średniej liczby węzłów obliczeniowych na kafelek, a nie od porowatości całej geometrii, a jednocześnie dla wielu geometrii rzadkich średnia porowatość kafelka może być utrzymana na wysokim poziomie niezależnie od porowatości samej geometrii, w wielu przypadkach struktury danych wykorzystujące kafelki oferują potencjalnie największą szybkość przetwarzania. Rzeczywiste wyniki eksperymentów potwierdzają, że opracowane metody pozwalają na osiągnięcie najwyższej wydajności dla geometrii rzadkich, a co więcej, jest nawet możliwe uzyskanie wydajności porównywalnej z najszybszymi implementacjami dla geometrii gęstych. Dzięki temu

istnieje możliwość opracowania implementacji uniwersalnych, które równie dobrze będą radziły sobie zarówno z geometriami rzadkimi, jak i gęstymi; cytując [A1]:

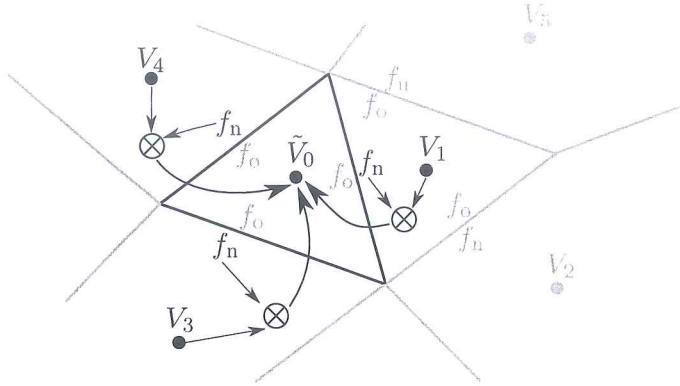
> *„Our implementation achieves a level of performance close to the best dense ones. We hope that its future improvements, especially in memory usage, can make tiles a viable alternative for specialized implementations for dense geometries and remove a slightly artificial distinction between dense and sparse geometries."*

Dodatkowo, wraz z obserwowanym od dawna wzrostem stosunku mocy obliczeniowej procesorów do przepustowości pamięci, metody pozwalające na zmniejszenie liczby przesyłanych danych odgrywają coraz większą rolę w wielu problemach, w których szybkość przetwarzania jest istotna. Zaproponowana idea doskonale wpisuje się w ten trend, gdyż nie tylko umożliwia uzyskanie dużej szybkości przetwarzania dla algorytmów szablonowych, ale najprawdopodobniej może być także zaadaptowana na potrzeby innych algorytmów numerycznych dla geometrii rzadkich, np. jako niskopoziomowa struktura danych w metodach wykorzystujących hierarchiczne siatki o różnej rozdzielczości (*ang. hierarchical multigrid methods*, por. [A1] rozdz. 5. *"Conclusions"*).

Opracowane struktury danych i szybkie algorytmy zostały wykorzystane w implementacji biblioteki *microflow* umożliwiającej symulacje przepływu płynów z użyciem metody siatkowej Boltzmanna. Pierwszą wersję biblioteki, umożliwiającą wyłącznie symulacje geometrii dwuwymiarowych, przygotowano w ramach grantu „Projektowanie, modelowanie i fabrykacja mikroaparatów metodą bezpośredniego grawerowania laserowego" pod kierunkiem dr R. Szafrana z Wydziału Chemicznego Politechniki Wrocławskiej, gdzie opracowanie oprogramowania do symulacji przepływu w mikrokanałach było jednym z zadań. Kolejna wersja, finansowana ze środków statutowych Wydziału Elektroniki Politechniki Wrocławskiej, pozwala na symulacje geometrii trójwymiarowych. Obecnie są to implementacje oferujące najwyższą na świecie wydajność na pojedynczym procesorze graficznym dla wielu geometrii rzadkich. Obie wersje kodu, oraz wyniki uzyskane za ich pomocą, posłużyły do przygotowania artykułów [A1] i [A2].

Mój wkład w przygotowanie implementacji polegał na zaprojektowaniu i zaimplementowaniu kodu o wysokiej wydajności dla geometrii rzadkich na podstawie poprawnej numerycznie, wzorcowej wersji kodu przygotowanej przez dr R. Szafrana. Wzorcowa wersja kodu wykorzystywała proste struktury danych przeznaczone dla regularnych geometrii gęstych (statyczne, globalne wielowymiarowe tablice przechowujące wartości dla wszystkich węzłów geometrii) oraz zawierała wyłącznie bardzo uproszczony interfejs użytkownika. W ramach implementacji wersji GPU wykorzystującej opisane wyżej struktury danych i algorytmy zaprojektowałem, zaimplementowałem, sprofilowałem i zoptymalizowałem kompletny kod pozwalający na przeprowadzenie pełnej symulacji: od przygotowania przypadku na podstawie bitmapy w wersji 2D lub pliku w formacie STL (*ang. stereolitography*) dla wersji 3D, aż do wygenerowania wyników w postaci plików w formacie VTK (*ang. Visualization Toolkit*) zawierających wyliczone pola prędkości, ciśnienia, itd.; przygotowałem zestaw testów wykorzystujących kod dr R. Szafrana do sprawdzania poprawności numerycznej implementacji; a także zaimplementowałem interfejs użytkownika oparty na koncepcji użytej w kodzie przygotowanym na potrzeby niewchodzącej w skład monotematycznego ciągu publikacji pracy [C1], w której mój wkład polegał na zaprojektowaniu i zaimplementowaniu oprogramowania wykorzystanego do przeprowadzenia symulacji.

[C1] R. Szafran, **T. Tomczak**, Wykorzystanie metody lattice-Boltzmann do symulacji mikroprzepływów w kanałach układów lab on a chip, *Inżynieria i Aparatura Chemiczna*, R. 52, nr 6, s. 568-569, 2013.



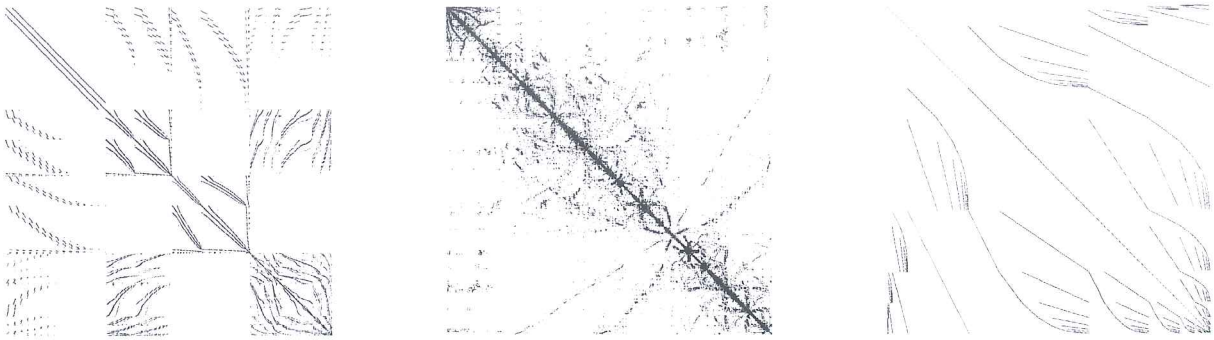
Rysunek 8: Relacja pomiędzy operacją mnożenia macierzy przez wektor a operacjami na wartościach przyporządkowanych do komórek siatki i ich ścian w metodzie objętości skończonych. V_i oznacza wartości wewnątrz komórek zapisane jako elementy wektora, f_n i f_o oznaczają wartości dla ścian zapisane jako elementy macierzy w wierszu (f_n) bądź kolumnie (f_o) o indeksie i. Indeksy kolumn i wierszy w macierzy odpowiadają indeksom komórek V_i (komórki aktualnie rozpatrywanej i sąsiednich), natomiast $\tilde{V}_0$ oznacza nową wartość elementu wektora, odpowiadającą wartości w komórce, równą iloczynowi wiersza macierzy przez wektor zawierający wartości V_i.

4.2.1.2 Metody zwiększania szybkości przetwarzania w algorytmach algebry liniowej dla macierzy rzadkich

Jednym z podstawowych obszarów zastosowań algorytmów algebry liniowej dla macierzy rzadkich są procedury numerycznego całkowania równań różniczkowych, szczególnie dla siatek niestrukturalnych (*ang. unstructured grid*). Z punktu widzenia implementacji, wiele metod numerycznego całkowania równań różniczkowych, np. metodę objętości skończonych [15] czy metodę elementów skończonych [16], [17], można sprowadzić do prostych operacji arytmetycznych wykonywanych na wartościach przyporządkowanych komórkom (*ang. cell*), ich ścianom (*ang. face*) czy wierzchołkom (*ang. vertex*) siatki geometrycznej (*ang. grid*) wypełniającej i dyskretyzującej dziedzinę rozwiązywanego równania. Dla przestrzeni dwu- i trójwymiarowych często stosowane są siatki zbudowane z wielokątów i wielościanów, choć spotyka się także elementy wyższych rzędów, w których krawędzie są krzywymi.

Proces numerycznego całkowania równań różniczkowych można zrealizować w postaci sekwencji etapów składających się z operacji na wartościach przyporządkowanych elementom siatki geometrycznej. Ponieważ zazwyczaj etapy te sprowadzają się do zestawu jednakowych działań wykonywanych dla każdej komórki czy ściany, to naturalną metodą zrównoleglenia procedur całkowania równań różniczkowych jest zrównoleglenie działań w poszczególnych etapach przy zachowaniu sekwencyjnych zależności pomiędzy etapami. Jest to m.in. spowodowane tym, że w praktycznych przypadkach liczba komórek nierzadko osiąga wartości rzędu milionów lub więcej, a komunikacja pomiędzy nimi w ramach pojedynczego etapu procedury całkowania równania różniczkowego jest ograniczona wyłącznie do bliskich sąsiadów.

Poszczególne etapy metod numerycznego całkowania równań różniczkowych można zapisać z wykorzystaniem rachunku macierzowego, co pozwala wyrazić je jako sekwencje operacji na macierzach i wektorach (por. rys. 8) i umożliwia implementację z użyciem standardowych procedur BLAS (*ang. Basic Linear Algebra Subprograms*). Ponieważ w macierzach przechowywane są np. wartości przyporządkowane ścianom współdzielonym przez sąsiednie komórki i opisujące wzajemne oddziaływania pomiędzy tymi komórkami, macierze te są macierzami rzadkimi (*ang.*



Rysunek 9: Rozmieszczenie elementów niezerowych w przykładowych macierzach rzadkich tworzonych w metodach numerycznego całkowania równań różniczkowych dla siatek niestrukturalnych.

sparse matrix), tzn. zawierającymi niewielką liczbę elementów niezerowych. Dodatkowo, większość niezerowych elementów macierzy jest ułożona symetrycznie względem przekątnej macierzy, gdyż elementy te opisują wzajemne relacje pomiędzy sąsiednimi elementami siatki geometrycznej. Ponieważ w wielu sytuacjach dla rzeczywistych, skomplikowanych geometrii stosuje się siatki niestrukturalne, stąd w poszczególnych wierszach macierzy wynikowych liczby elementów niezerowych i ich ułożenie mogą się znacznie różnić (por. rys. 9), np. ze względu na występowanie w siatce geometrycznej wielościanów o różnej liczbie ścian.

Opis procedury całkowania równań różniczkowych w kategoriach rachunku macierzowego umożliwia jej realizację w postaci dwóch naprzemiennych etapów: etapu konstrukcji macierzy i wektora prawej strony układu równań liniowych oraz następującego po nim etapu rozwiązania powstałego układu równań. Podczas konstrukcji układu równań wartości elementów macierzy i wektora prawej strony równania są wyznaczane na podstawie wartości w wybranych punktach siatki geometrycznej. W zależności od operacji występujących w rozwiązywanym równaniu różniczkowym (gradientu, dywergencji, laplasjanu, itd.) struktura kolejnych odwołań do pamięci podczas konstrukcji układu równań liniowych może być taka sama, jak w procedurze mnożenia macierzy przez wektor, lub może odpowiadać mnożeniu przez macierz transponowaną, czyli wymagać dostępu do elementów w kolumnie macierzy, a nie w wierszu. Odpowiada to dostępowi do współczynników f_o lub f_n pokazanych na rys. 8. Niestety, problematyka szybkich metod transponowania macierzy rzadkich jest dość słabo zbadana (por. [18]).

Na etapie rozwiązywania powstałego układu równań liniowych zazwyczaj stosuje się metody iteracyjne ze względu na dużą (rzędu 10^6 lub większą) liczbę niewiadomych. Typowymi przykładami mogą być: metoda gradientu sprzężonego (*ang. conjugate gradient method, CG*) lub metoda BiCGStab (*ang. biconjugate gradient stabilized method*) stosowana, gdy macierz definiująca układ równań liniowych jest niesymetryczna. Z punktu widzenia implementacji (por. [19]) metody te mogą być traktowane jako sekwencje podstawowych operacji na wektorach dostępnych w postaci procedur BLAS 1 poziomu: iloczynów skalarnych, obliczania norm wektora (redukcji), operacji $\vec{y} = \alpha \cdot \vec{x} + \vec{y}$ (operacja nazywana "axpy"), oraz operacji mnożenia macierzy rzadkiej przez wektor (*ang. sparse matrix-vector multiplication, SpMV*). W celu zmniejszenia liczby iteracji potrzebnych do wyznaczenia wystarczająco dokładnego rozwiązania można metody rozwiązywania układu równań liniowych rozszerzyć o procedurę wstępnego polepszenia uwarunkowania macierzy (*ang. preconditioner*), realizowaną także z użyciem prostych operacji na wektorach bądź macierzach.

Wyniki moich badań w zakresie zwiększania szybkości przetwarzania, z wykorzystaniem równoległości drobnoziarnistej wymaganej do efektywnego wykorzystania procesorów masowo rów-

noległych, algorytmów algebry liniowej dla macierzy rzadkich będących fragmentami procedur numerycznego całkowania równań różniczkowych przedstawiono w pracach [A4] i [A5]. W ramach opracowanej metodyki zaproponowano i przeanalizowano dwie metody akceleracji:

- drobnoziarniste zrównoleglenie kompletnych procedur całkowania numerycznego, co wymagało opracowania specjalizowanych struktur danych umożliwiających równoległą realizację wszystkich operacji na macierzach rzadkich i pozwoliło na wyeliminowanie kosztownej komunikacji pomiędzy etapami realizowanymi szeregowo (na procesorze ogólnego przeznaczenia) i równolegle (na procesorze masowo równoległym),
- drobnoziarniste zrównoleglenie jedynie wybranych etapów całkowania (procedur rozwiązywania rzadkich układów równań liniowych).

Jako przykładów algorytmów numerycznego całkowania równań różniczkowych użyto popularnych metod obliczeniowej mechaniki płynów: PISO (*ang. Pressure-Implicit with Splitting of Operators*) i SIMPLE (*ang. Semi-Implicit Method for Pressure Linked Equations*), szczegółowo omówionych np. w [20]. Metody PISO i SIMPLE można potraktować jako iteracyjne rozwiązanie równań różniczkowych opisujących zachowanie poruszającego się płynu (równań Naviera-Stokesa) przy użyciu np. metody objętości skończonych. Ze względu na złożoność metod PISO/SIMPLE oraz rozbudowaną postać rozwiązywanych równań różniczkowych mogą one być traktowane jako reprezentatywny przykład ilustrujący problemy występujące w wielu rzeczywistych zagadnieniach.

4.2.1.2.1 Drobnoziarniste zrównoleglenie kompletnych algorytmów całkowania numerycznego

Pomimo iż znane są równoległe wersje algorytmów całkowania numerycznego, wcześniejsze rozwiązania, w szczególności dla siatek niestrukturalnych, były projektowane z uwzględnieniem przede wszystkim równoległości gruboziarnistej (*ang. coarse-grained parallelism*), gdzie np. fragmenty geometrii są przetwarzane przez poszczególne komputery w klastrze, a każdy z komputerów wykonuje operacje w sposób sekwencyjny [21]. Takie podejście daje dobre wyniki na maszynach obliczeniowych z rozproszoną pamięcią operacyjną (*ang. distributed memory machines*), nie sprawdza się jednak w przypadku procesorów masowo równoległych czy wektorowych. Efektywne wykorzystanie zasobów sprzętowych procesorów masowo równoległych wymaga raczej równoległości średnio- lub nawet drobnoziarnistej (*ang. medium-/fine-grained parallelism*). O ile wiele kroków algorytmów numerycznego całkowania równań różniczkowych daje się naturalnie zrównoleglić drobnoziarniście (np. operacje na wektorach), istotnym problemem są etapy tworzenia macierzy rzadkiej definiującej układ równań liniowych, w których wymagany jest dostęp do elementów z tej samej kolumny macierzy, oraz etap rozwiązywania układu równań liniowych, w którym konieczne jest wielokrotne przeglądanie macierzy wierszami podczas procedury mnożenia macierzy rzadkiej przez wektor (*ang. sparse matrix-vector multiplication, SpMV*). Konieczne więc było opracowanie specjalizowanych struktur danych i szybkich algorytmów pozwalających na uzyskanie wysokiej wydajności obliczeń dla niestrukturalnych siatek, koniecznych dla wydajnych symulacji rzadkich, nieregularnych geometrii.

Wcześniejsze szybkie realizacje algorytmów numerycznego całkowania równań różniczkowych, w szczególności metodami PISO/SIMPLE, na procesorach masowo równoległych w większości dotyczyły wersji dla siatek strukturalnych, dla których stosunkowo łatwo można zagwarantować regularny dostęp do danych w pamięci konieczny do uzyskania wysokiej wydajności [22–26]. Problematyka szybkich, drobnoziarniście równoległych metod całkowania równań różniczkowych dla siatek niestrukturalnych była znacznie słabiej zbadana, a opublikowane wyniki

dotyczyły albo wersji wykorzystujących półautomatyczną generację kodu równoległego [27–29], w których wydajność obliczeniowa była niska, albo metod wysokiego rzędu w których stosunek liczby operacji obliczeniowych do rozmiaru przesłanych danych jest duży [30,31], albo rozwiązań uproszczonych o ograniczonej funkcjonalności [32,33].

Uzyskanie wysokiej wydajności obliczeniowej masowo równoległych algorytmów całkowania szerokiej klasy równań różniczkowych wymagało więc opracowania nowej metody pozwalającej na efektywne, drobnoziarniste zrównoleglenie wszystkich operacji występujących w kompletnych realizacjach procedur całkowania, w tym etapu konstrukcji układu równań liniowych na podstawie rozbudowanych równań różniczkowych zawierających m.in. operatory gradientu, dywergencji i laplasjanu. W ramach zaproponowanej metody opracowano i opisano w pracy [A4] format danych umożliwiający reprezentację w pamięci DRAM macierzy rzadkich o nieregularnej liczbie elementów niezerowych w wierszach, umożliwiający przechowywanie danych dla złożonych, niestrukturalnych siatek geometrycznych, oraz dedykowane, drobnoziarniście równoległe algorytmy przeglądania macierzy wierszami i kolumnami. Mój wkład polegał na opracowaniu formatu danych, algorytmów, oraz metody pozwalającej na ich wykorzystanie w procedurach numerycznego całkowania równań różniczkowych, a także na eksperymentalnej weryfikacji poprawności i wydajności zaproponowanego rozwiązania.

Format danych

Unikalną cechą szczególną zaproponowanego formatu danych jest nie tylko bardzo wysoka wydajność masowo równoległych implementacji przeglądania macierzy wierszami (np. w czasie procedury SpMV), ale także stosunkowo proste przeglądanie macierzy kolumnami, oraz możliwość wykonywania działań na macierzach, w których liczby elementów w poszczególnych wierszach mocno się różnią. Jest to o tyle istotne, że dotychczasowe formaty przechowywania macierzy rzadkich w pamięci operacyjnej (np. CSR, COO, ELL, BSR, por. [34]) były projektowane głównie z myślą o zapewnieniu wysokiej wydajności jedynie procedur SpMV, a dodatkowo część z nich wyłącznie dla macierzy o podobnej liczbie elementów w wierszach.

Zaprojektowany format danych rozszerza opisany w pracy [34] format hybrydowy ELL/COO o dodatkowe informacje konieczne do przeglądania macierzy kolumnami. W formacie hybrydowym zakłada się pewną "typową" liczbę elementów niezerowych w wierszu K, a następnie dla każdego wiersza przechowywane jest K elementów w dwóch gęstych tablicach zawierających wartości elementów (tablica wartości) i ich indeksy kolumn (tablica indeksów). Pozostałe elementy niezerowe przechowywane są w formacie COO. Takie rozwiązanie umożliwia utrzymanie wysokiej wydajności podczas dostępu do pierwszych K elementów z każdego wiersza, pozwalając jednocześnie na przetwarzanie macierzy o różnej liczbie elementów w poszczególnych wierszach. Istotną wadą tej struktury danych w kontekście kompletnych algorytmów numerycznego całkowania równań różniczkowych jest bardzo wysoka złożoność operacji przeglądania macierzy kolumnami, ponieważ konieczna jest analiza całej macierzy w celu wyszukania elementów położonych w zadanej kolumnie.

W przeciwieństwie do formatu hybrydowego ELL/COO, w zaprojektowanym formacie danych tablica indeksów zawiera bezpośrednie wskaźniki do elementów położonych symetrycznie względem przekątnej macierzy. W połączeniu z regularnym ułożeniem danych w pamięci w blokach zawierających ustaloną liczbę sąsiednich elementów, wskaźniki te mogą być przekształcone w indeksy kolumn operacją dzielenia modulo przez rozmiar pojedynczego bloku danych. Dodatkowo, informacja o położeniu elementu odbitego symetrycznie względem przekątnej umożliwia dostęp do elementów w kolumnie i przy użyciu wskaźników dla elementów w wierszu o tym samym numerze i (z wykorzystaniem adresowania pośredniego).

Pozostałe elementy macierzy z wierszy zawierających więcej niż K elementów niezerowych są przechowywane w formacie CSR, aby zagwarantować prosty dostęp do elementów z zadanego

wiersza, co bezpośrednio przekłada się także na łatwość przeglądania macierzy kolumnami z wykorzystaniem adresowania pośredniego. Powiązania pomiędzy tablicami zawierającymi pierwsze K elementów z wiersza i elementami dodatkowymi przechowywanymi w formacie CSR są kodowane przy użyciu pojedynczych bitów w reprezentacjach indeksów elementów. Jeśli elementy położone symetrycznie względem przekątnej znajdują się w różnych zbiorach tablic (np. jeden z elementów znajduje się w tablicach gęstych, a drugi w tablicach CSR), to wskaźniki tych elementów mają ustawiony najbardziej znaczący bit. Takie rozwiązanie powoduje co prawda zmniejszenie maksymalnej liczby elementów możliwych do zapisania w zaproponowanym formacie, nie stanowi to jednak problemu, ponieważ w rzeczywistych implementacjach rozmiary tablic są znacznie bardziej ograniczone przez rozmiar dostępnej pamięci operacyjnej.

Algorytmy przeglądania macierzy

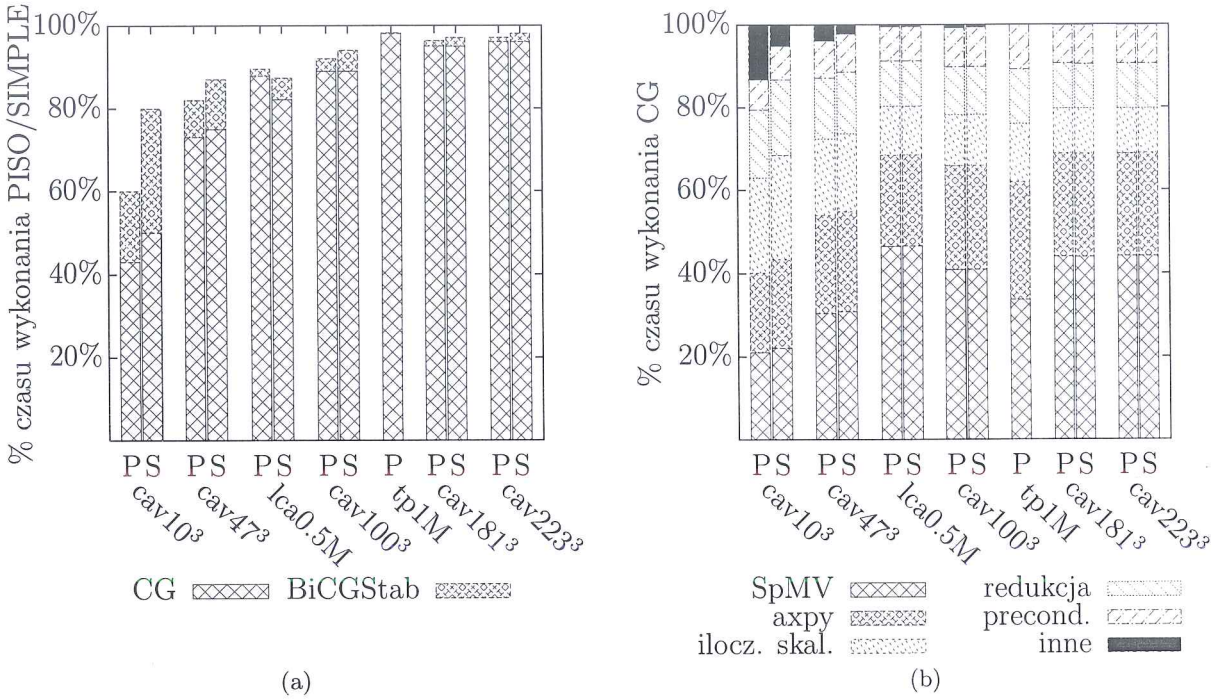
Algorytmy szybkiego, równoległego przeglądania wierszami i kolumnami macierzy rzadkiej w zaproponowanym formacie odpowiedzialne są za:

- uzyskanie, w wyniku operacji logicznych na indeksach, informacji o tablicy, w której przechowywane są wartości konkretnego elementu,
- obliczenie rzeczywistych indeksów poprzez proste operacje logiczne i arytmetyczne na bitach elementów tablic indeksów,
- dostęp do właściwych wartości elementów macierzy.

Podczas przeglądania macierzy kolumnami wykorzystywane jest adresowanie pośrednie powodujące spadek wydajności wynikający zarówno z dodatkowych odczytów indeksów pośrednich, jak i z nieregularnego dostępu do pamięci skutkującego dużą fragmentacją wewnętrzną transakcji blokowych z pamięcią. Przeglądanie macierzy wierszami realizowane jest bez dodatkowych kosztów. Taka konstrukcja struktury danych i algorytmów umożliwia jednoczesne zachowanie wysokiej wydajności często wykonywanych procedur SpMV, wymagających przeglądania macierzy wierszami, jak i wystarczającą wydajność rzadko wykonywanych procedur przeglądania macierzy kolumnami. Duże znaczenie ma też niewielki rozmiar dodatkowych danych (w praktyce zaproponowana struktura danych może zajmować mniejszą liczbę komórek pamięci, niż standardowy format hybrydowy ELL/COO), szczególnie istotny z powodu niewielkiej pojemności pamięci stosowanych w akceleratorach zbudowanych na bazie procesorów graficznych.

Tabela 1: Czas wykonania podstawowych operacji na macierzy w eksperymentalnej implementacji na akceleratorze Tesla C2070. Dane dla dywergencji, gradientu i laplasjanu są znormalizowane w stosunku do czasu wydajnej implementacji procedury SpMV.

Przypadek	czas SpMV [µs]	czas znormalizowany		
		dywergencja	gradient	laplasjan
cav10^3	41	3.6	16.8	2.3
cav47^3	139	4.1	18.3	2.5
cav100^3	1 103	2.9	10.7	2.4
cav181^3	6 763	2.5	9.1	2.3
cav223^3	12 620	2.5	9.0	2.3
lca0.5M	708	2.6	12.1	58.7
tp1M	775	2.8	10.4	2.3



(a) (b)

Rysunek 10: Procentowy udział: a) czasu wykonania procedur rozwiązywania układów równań liniowych metodami CG i BiCGStab w czasie wykonania kompletnych procedur całkowania równania różniczkowego, oraz b) czasu wykonania poszczególnych operacji występujących w implementacji metody CG. Literami P i S oznaczono czasy dla implementacji algorytmów PISO i SIMPLE, SpMV oznacza mnożenie macierzy rzadkiej przez wektor, axpy oznacza operację $\vec{y} = \alpha \cdot \vec{x} + \vec{y}$, natomiast precond. oznacza procedurę wstępnego polepszenia uwarunkowania układu równań (*ang. preconditioner*).

Zrównoleglenie kompletnych algorytmów całkowania

Opracowanie szybkich algorytmów przeglądania macierzy rzadkiej wierszami i kolumnami umożliwiło efektywne, drobnoziarniste zrównoleglenie kompletnych algorytmów numerycznego całkowania równań różniczkowych dla siatek niestrukturalnych. Istotną cechą zrównoleglonych wersji tych algorytmów jest realizacja wszystkich etapów w wersjach masowo równoległych, co pozwoliło na wyeliminowanie większości operacji szeregowych (m.in. komunikacji pomiędzy koprocesorem równoległym a zarządzającym procesorem ogólnego przeznaczenia), znacznie ograniczających wydajność (zgodnie z prawem Amdahla). Według mojej najlepszej wiedzy, opracowana według zaproponowanej metody eksperymentalna, szybka implementacja o rozbudowanej funkcjonalności metod PISO/SIMPLE na masowo równoległych procesorach graficznych była pierwszym na świecie projektem tego typu zakończonym sukcesem. Implementacja ta pozwoliła na praktyczną weryfikację wydajności zaproponowanego rozwiązania oraz jego poprawności. Uzyskane wyniki eksperymentalne potwierdziły niskie narzuty zaproponowanych formatów danych i algorytmów – opracowane równoległe wersje algorytmów na procesory graficzne uzyskały podobny stosunek wydajności obliczeniowej do teoretycznej maksymalnej mocy obliczeniowej sprzętu (*ang. theoretical peak performance*), co optymalizowane dla klasycznych procesorów ogólnego przeznaczenia wersje w rozwijanym od lat oprogramowaniu OpenFOAM. Cytując fragment podsumowania [A4]:

> *„Our results show that a GPU can outperform a dual-socket server-class CPU running algorithmically equivalent implementations of popular CFD solvers, SIMPLE*

or PISO, by approximately four times. This value is consistent with the ratio 4.5 of the theoretical memory bandwidths of the two processors used in our test, 144 GB/s (GPU) and 32 GB/s (CPU).”

Przedstawione na rys. 10 uzyskane wyniki eksperymentalne potwierdziły, że zastosowany format danych pozwolił na praktycznie całkowitą eliminację dodatkowych kosztów spowodowanych koniecznością konstruowania układu równań poza procesorem masowo równoległym. Dla wszystkich przebadanych przypadków czas znalezienia rozwiązania był zdominowany praktycznie wyłącznie czasem potrzebnym na rozwiązanie układów równań liniowych. Oznacza to, że zaproponowana metoda ułożenia danych w pamięci umożliwiła przeprowadzenie z wysoką wydajnością wszystkich dodatkowych operacji wymagających nieregularnego dostępu do pamięci. Szczegółowe dane przedstawiono w tab. 1, gdzie można zauważyć, że dla operacji gradientu wymagającej dostępu do elementów w kolumnie macierzy czasy wykonania są kilkukrotnie wyższe, niż w pozostałych przypadkach. Jednocześnie, na podstawie rozkładu czasu wykonania poszczególnych etapów procedury rozwiązywania układu równań liniowych metodą CG (por. rys. 10) można stwierdzić, że wydajność algorytmu możenia macierzy rzadkiej przez wektor jest wystarczająco wysoka, aby nie wpływać znacząco na wydajność całego algorytmu CG.

4.2.1.2.2 Przyspieszenie procedury rozwiązywania układów równań liniowych

Poza opracowaniem struktur danych i algorytmów dla macierzy rzadkich umożliwiających szybką, drobnoziarniście równoległą realizację kompletnych procedur całkowania równań różniczkowych, w pracy [A5] przeprowadzono także analizę podejścia sprowadzającego się do przyspieszenia jedynie wybranych etapów w/w procedur całkowania, którymi w rozpatrywanym przypadku były etapy iteracyjnego rozwiązywania rzadkich układów równań liniowych. Zaletą takiego rozwiązania była łatwość realizacji wynikająca z dostępności formatów danych i szybkich algorytmów umożliwiających rozwiązanie za pomocą procesorów graficznych dużych, rzadkich układów równań. Nie bez znaczenia jest też fakt, że przyspieszenie jedynie wybranych fragmentów procedur (*ang. partial acceleration*) umożliwia proste zwiększenie wydajności istniejących, rozbudowanych systemów opracowanych z użyciem innych paradygmatów programowania równoległego (np. dla maszyn z rozproszoną pamięcią operacyjną). W eksperymentalnej, prototypowej implementacji przeanalizowano możliwość zwiększenia wydajności biblioteki OpenFOAM [35], będącej rozwijanym od wielu lat, rozbudowanym oprogramowaniem implementującym szereg operacji związanych z całkowaniem równań różniczkowych.

Mój wkład polegał na:

- Opracowaniu prostej metody przyspieszenia procedur rozwiązywania układów równań liniowych przy użyciu procesorów masowo równoległych w gruboziarniście równoległych implementacjach algorytmów rozwiązywania równań różniczkowych.
- Opracowaniu metody pomiarów i porównania wydajności kompletnych procedur rozwiązywania równań liniowych oraz samej procedury SpMV.

Metoda przyspieszenia procedur rozwiązywania układów równań liniowych

Opracowana metoda przyspieszenia sprowadzała się do 3 kroków:

- konwersji formatów danych używanych w gruboziarniście równoległych implementacjach algorytmów rozwiązywania równań różniczkowych do szeroko stosowanego formatu CSR (*ang. Compressed Sparse Row*),

- przesłaniu skonwertowanych danych (macierzy i wektora prawej strony układu równań liniowych) do pamięci procesora graficznego i rozwiązania układu równań przy użyciu istniejących, szybkich procedur dla procesora graficznego,
- przesłaniu wyniku do pamięci operacyjnej komputera sterującego.

Jako formatu danych stosowanego po stronie procesora graficznego do przechowywania macierzy rzadkiej użyto formatu CSR z powodu dostępności i wysokiej wydajności implementacji algorytmów SpMV dla tego formatu oraz łatwości konwersji z formatu stosowanego w istniejącej implementacji w oprogramowaniu OpenFOAM. Format danych wektorów zmiennych i prawej strony równania pozostał bez zmian, co ułatwiło wykorzystanie uzyskanego rozwiązania układu równań liniowych w dalszych krokach procedury numerycznego całkowania.

Metoda analizy wydajności

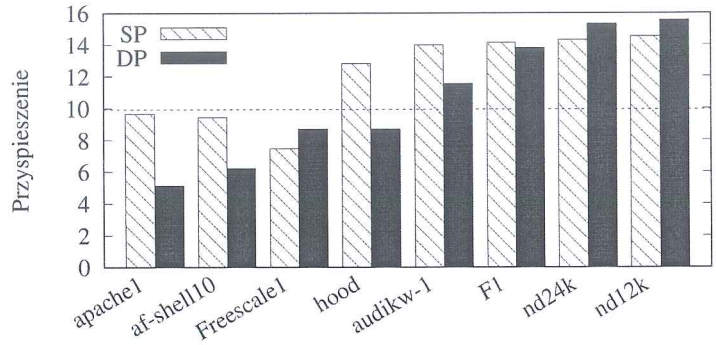
W ramach metody pomiarów i porównania wydajności:

- opracowałem model złożoności obliczeniowej procedur rozwiązywania układów równań liniowych używany do porównywania różnych implementacji oraz obliczenia wydajności, wyrażonej jako liczba operacji arytmetycznych bądź liczba przesłanych bajtów w ciągu sekundy, na podstawie zmierzonego czasu wykonania,
- wybrałem i dostarczyłem zbiór macierzy, spośród których część została użyta w artykule [A5],
- wytypowałem szybką, optymalizowaną dla procesorów ogólnego przeznaczenia bibliotekę procedur Intel Math Kernel Library zawierającą m.in. implementacje procedur BLAS i traktowaną w artykule jako wzorcowa, szybka implementacja dla procesorów ogólnego przeznaczenia,
- dostarczyłem ogólne wytyczne ułatwiające porównanie wydajności różnych implementacji procedur rozwiązywania układów równań liniowych, m.in. wymuszenie ustalonej liczby kroków w porównywanych procedurach.

Opracowana metoda posłużyła do wygenerowania wyników zamieszczonych w pracy [A5] rozdz. 2.3 *Iterative linear solvers* (z pominięciem części dotyczącej wpływu procedury wstępnego polepszenia uwarunkowania macierzy na wydajność) oraz [A5] rozdz. 3.1 *Acceleration measurement* na rys. 6.

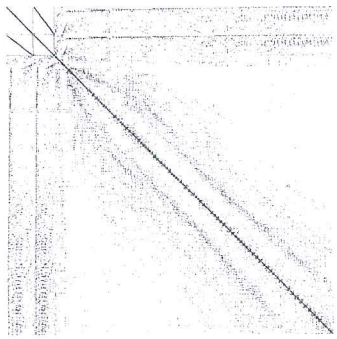
Analiza wydajności procedury mnożenia macierzy rzadkiej

Analiza wydajności wydzielonej procedury mnożenia macierzy rzadkiej przez wektor wykazała, że zaobserwowany wzrost wydajności implementacji drobnoziarniście równoległej uruchamianej na procesorze masowo równoległym w stosunku do implementacji gruboziarniście równoległej uruchomionej na procesorze ogólnego przeznaczenia wynika ze zwiększonej przepustowości pamięci operacyjnej DRAM masowo równoległego procesora graficznego. Zaobserwowany wzrost wydajności obliczeniowej procedury SpMV, pokazany na rys. 11, dla przeanalizowanych przypadków był nieco niższy lub nieco wyższy, niż stosunek przepustowości pamięci DRAM dla porównywanych maszyn. Dane z rys. 11 dotyczą kilku wybranych macierzy rzadkich o różnej strukturze używanych w różnych klasach problemów. Macierze w zestawie zawierały od ok. 0.5×10^6 do ok. 80×10^6 elementów niezerowych, a średnia liczba elementów w wierszu zawierała się pomiędzy 5 a 400. Odchylenia zmierzonego przyspieszenia od stosunku teoretycznych przepustowości

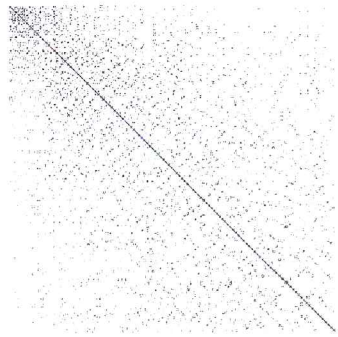


(a) Stosunek wydajności procedur SpMV

(b) apache1



(c) hood



(d) nd12k

Rysunek 11: Stosunek wydajności procedury SpMV wykonywanej na procesorze graficznym do wydajności procedury SpMV z biblioteki Intel Math Kernel Library dla obliczeń z użyciem reprezentacji zmiennoprzecinkowych pojedynczej (SP) i podwójnej (DP) precyzji i 8 różnych macierzy rzadkich a); oraz rozmieszczenie elementów niezerowych w wybranych macierzach b), c), d). Na rys. a) przerywaną linią oznaczono stosunek maksymalnych, teoretycznych przepustowości pamięci operacyjnych DRAM porównywanych platform sprzętowych: Nvidia GeForce GTX 275 i Intel Core 2 Quad Q8400.

pamięci DRAM wynikały z różnic w strukturze macierzy. Dla macierzy, w których elementy niezerowe były bardziej "skupione", tzn. umieszczone w bliskim sąsiedztwie, wydajność procedury SpMV na procesorze ogólnego przeznaczenia była wyższa w stosunku do przepustowości pamięci DRAM, niż dla procesora graficznego. Gdy układ elementów niezerowych w macierzy był nieregularny, lepsze wykorzystanie przepustowości pamięci DRAM zaobserwowano dla implementacji na procesorze graficznym. Przyczyną takiego zachowania był różny rozmiar pamięci podręcznej, w przeliczeniu na liczbę aktualnie przetwarzanych elementów macierzy, dla porównywanych platform sprzętowych. Ponieważ dla procesora graficznego rozmiar ten był znacznie mniejszy, niż dla procesora ogólnego przeznaczenia, to w pamięci podręcznej procesora graficznego mogła być przechowywana znacznie mniejsza część zbioru roboczego procesu, niż w przypadku procesora ogólnego przeznaczenia. Stąd, dla macierzy, w których elementy niezerowe były umieszczone w bliskim sąsiedztwie, wersja dla procesora ogólnego przeznaczenia mogła korzystać z danych przechowywanych w pamięci podręcznej, np. z kopii elementów wektora, przez który macierz jest mnożona. Dla macierzy z elementami niezerowymi rozmieszczonymi bardziej nieregularnie, dane potrzebne do obliczeń nie były wcześniej używane, co powodowało, że w obu porównywanych implementacjach musiały być przesyłane z pamięci operacyjnej DRAM. W tych przypadkach wyższą wydajność w stosunku do przepustowości pamięci DRAM osiągała implementacja dla

procesora graficznego.

Analiza wydajności procedury rozwiązywania układów równań liniowych

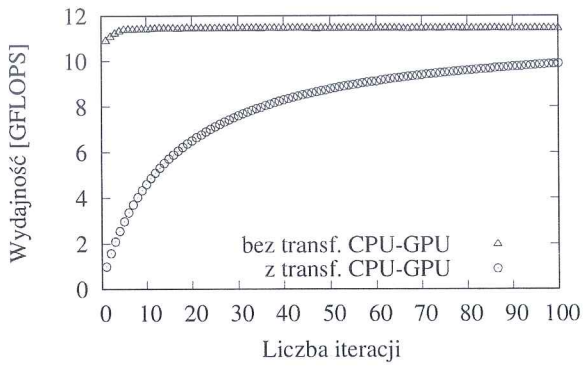
Analiza wydajności procedury rozwiązywania układów równań liniowych wykazała, że w rzeczywistych sytuacjach wydajność procedury może być istotnie ograniczona przez czas potrzebny do komunikacji pomiędzy sterującym procesorem ogólnego przeznaczenia, a procesorem masowo równoległym. O ile pokazana na rys. 12a) wydajność mierzona z pominięciem komunikacji praktycznie nie zależy od liczby iteracji procedury BiCGStab, to uwzględnienie w czasie wykonania BiCGStab czasu komunikacji powoduje, że nawet dla dość dużej liczby iteracji (rzędu kilkudziesięciu) czas komunikacji znacznie ogranicza wydajność procedury BiCGStab uruchamianej na procesorze masowo równoległym. Wykorzystując prawo Amdahla można wykazać, że zależność wydajności obliczeniowej z uwzględnieniem komunikacji od liczby iteracji procedury BiCGStab może być opisana wzorem

$$W = \frac{N_{FLOP}}{t_{transf} + n_{iter} \cdot t_{comp}}, \tag{9}$$

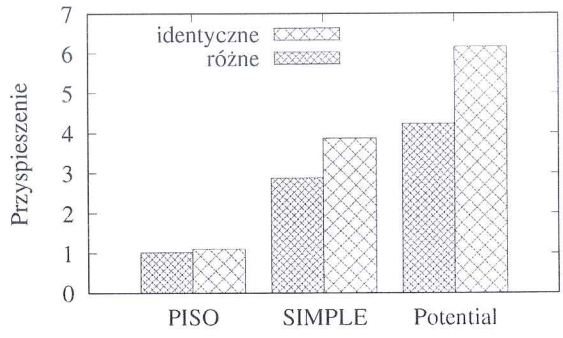
gdzie N_{FLOP} oznacza liczbę operacji zmiennoprzecinkowych w procedurze BiCGStab, n_{iter} oznacza liczbę iteracji procedury BiCGStab, t_{transf} oznacza czas potrzebny na jednokrotne przesłanie macierzy i wektora prawej strony układu równań do procesora graficznego oraz odesłanie wyniku (wektora obliczonych niewiadomych), a t_{comp} oznacza czas potrzebny na wykonanie pojedynczej iteracji procedury BiCGStab. Stąd, wydajność mierzona z uwzględnieniem komunikacji będzie asymptotycznie dążyła do wartości wydajności mierzonej z pominięciem komunikacji. Dla danych pokazanych na rys. 12a) wartości poszczególnych parametrów wynoszą: $N_{FLOP} = 0.057 + n_{iter} \cdot 0.116$ GFLOP, $t_{transf} = 0.158$ s, a $t_{comp} = 10.5$ ms. Duży stosunek t_{transf}/t_{comp} jest spowodowany komunikacją poprzez magistralę PCI-Express, której przepustowość jest znacznie niższa, niż przepustowość pamięci procesora graficznego (wydajność operacji składających się na pojedynczą iterację algorytmu BiCGStab jest ograniczona przepustowością pamięci).

Analiza wydajności procedur całkowania numerycznego

Zależność przyspieszenia kompletnych procedur numerycznego całkowania równań różniczkowych od stosunku czasu operacji rozwiązywania układów równań liniowych do pozostałego czasu można również zaobserwować podczas analizy przyspieszenia różnych algorytmów całkowania równań różniczkowych. Wyniki zaprezentowane na rys. 12b) pokazują, że dla prostych algorytmów całkowania zastosowanie zrównoleglenia jedynie procedury rozwiązywania układów równań liniowych pozwala na uzyskanie kilkukrotnego przyspieszenia w stosunku do wersji w całości uruchamianej na procesorze ogólnego przeznaczenia. W przypadku bardziej złożonych algorytmów całkowania, w których rozwiązanie jest iteracyjnie poprawiane wymuszając komunikację pomiędzy fragmentami kodu wykonywanymi zarówno po stronie procesora masowo równoległego, jak i procesora ogólnego przeznaczenia, uzyskanie przyspieszenia może być trudne lub nawet niemożliwe. Dodatkowym problemem utrudniającym uzyskanie zadowalającego przyspieszenia poprzez przeniesienie części obliczeń na wydajny procesor masowo równoległy może być trudność zrównoleglenia szybkozbieżnych, zaawansowanych algorytmów numerycznych, dla których znane są jedynie wersje szeregowe. W przypadkach pokazanych na rys. 12b) zastosowanie szybkozbieżnych, szeregowych algorytmów na procesorze ogólnego przeznaczenia spowodowało zauważalny spadek przyspieszenia spowodowany zmniejszeniem czasu wykonania wersji szeregowej. Należy jednak zauważyć, że wersje ze zrównoleglonym, prostym algorytmem uruchamianym na procesorze masowo równoległym były co najmniej tak samo szybkie, jak zaawansowane algorytmy uruchamiane w wersjach szeregowych.



(a) Wydajność BiCGStab dla macierzy nd24k



(b) Przyspieszenie kompletnych algorytmów całkowania

Rysunek 12: Zależność wydajności procedury BiCGStab zaimplementowanej na procesorze graficznym od liczby iteracji a); stosunek wydajności obliczeniowej kompletnych algorytmów numerycznego całkowania równań różniczkowych zrealizowanych w wersji masowo równoległej na procesorze graficznym i w wersji gruboziarniście równoległej na procesorze ogólnego przeznaczenia b). Wykres z rys. a) pokazuje wydajność z uwzględnieniem czasu potrzebnego na przesłanie rozwiązywanego równania i uzyskanego wektora rozwiązania pomiędzy pamięciami operacyjnymi procesora graficznego i procesora ogólnego przeznaczenia (opisaną jako "z transf. CPU-GPU"), oraz z pominięciem tego czasu (opis "bez transf. CPU-GPU"). Dane dla rys. b) pokazują przyspieszenie dla identycznych algorytmów na obu porównywanych maszynach (dane opisane jako "identyczne") oraz dla szybszej wersji procedury wstępnego polepszenia uwarunkowania macierzy w wersji CPU ("różne").

Wyniki zaprezentowane w pracy [A5] pokazały więc, że przyspieszenie przy użyciu procesorów masowo równoległych wyłącznie procedur rozwiązywania układu równań liniowych pozwoliło dla części przypadków na uzyskanie kilkukrotnego przyspieszenia kompletnych algorytmów całkowania w stosunku do rozwiązania opartego o szybkie wersje procedur dostępnych w wydajnych implementacjach dostępnych w bibliotece optymalizowanej przez producenta procesora ogólnego przeznaczenia (Intel Math Kernel Library). Niestety, istniały także sytuacje, w których konieczność przesyłania dużych zbiorów danych przy użyciu stosunkowo wolnej magistrali PCI-Express, wykorzystywanej do komunikacji pomiędzy procesorem masowo równoległym a procesorem ogólnego przeznaczenia, powodowała znaczne ograniczenie wydajności. Cytując fragment podsumowania [A5]:

> *„Our investigations revealed that acceleration is limited by the amount of data transfers between the CPU and the GPU, relative to the amount of computations done in the GPU. The acceleration was quite good for relatively simple CFD methods. For example, potential flow calculations were approximately four to six times faster, depending on the preconditioner used in the CPU implementation. (...) However, for more demanding problems, such as in transient flow solvers, this simple strategy is no longer sufficient, since when we linked our GPU-accelerated linear solvers with the transient PISO method, the acceleration was hardly noticeable. This leads to the conclusion that in order to obtain reasonably good acceleration for time-dependent problems, the whole computational algorithm must be ported to the GPU."*

W celu poszerzenia zakresu problemów wykorzystujących równania różniczkowe, w których zastosowanie procesorów masowo równoległych może umożliwić zwiększenie wydajności procedur całkowania, konieczne więc było opracowanie metody umożliwiającej realizację wszystkich kro-

ków algorytmów całkowania równań różniczkowych w wersji drobnoziarniście równoległej na procesorze masowo równoległym. Pozwoliło to na znaczne zmniejszenie kosztownej komunikacji pomiędzy komputerem nadrzędnym a masowo równoległym procesorem graficznym, powodującej, zgodnie z prawem Amdahla, ograniczenie przyspieszenia obliczeń w części przypadków.

4.2.1.2.3 Wykorzystanie rezultatów

Omówione w pracy [A4] struktury danych i szybkie wersje algorytmów PISO/SIMPLE, w których wszystkie obliczenia są przeprowadzane za pomocą procesorów masowo równoległych, stanowią podstawę komercyjnej biblioteki SpeedIT Flow (http://www.vratis.com/what-is-speedit-flow/). Mój wkład polegał na zaprojektowaniu ogólnej struktury biblioteki oraz na zaimplementowaniu kodu realizującego szybkie, drobnoziarniście równoległe wersje algorytmów PISO/SIMPLE wykorzystujących struktury danych i algorytmy przedstawione w pracy [A4]. Poprawność założeń i przydatność zaprojektowanego formatu danych została także zweryfikowana w trakcie późniejszego rozwoju biblioteki, gdy niezależny zespół specjalistów z zakresu fizyki rozbudował zakres rozwiązywanych równań różniczkowych.

Przedstawiona w pracy [A5] metoda akceleracji części obliczeń została zaimplementowana jako fragment komercyjnej biblioteki SpeedIT (http://www.vratis.com/what-is-speedit/). Mój wkład obejmował m.in. zaprojektowanie i zaimplementowanie łącza wykorzystującego koncepcję omówioną w pracy [A5] i pozwalającego na użycie biblioteki do akceleracji obliczeń w szeroko stosowanym oprogramowaniu OpenFOAM.

Poza omówionymi powyżej zastosowaniami związanymi z numerycznym całkowaniem równań różniczkowych, opracowany format danych i algorytmy dedykowane dla macierzy rzadkich mogą także znaleźć zastosowanie w innych obszarach wymagających przeglądania macierzy wierszami i kolumnami. Przykładami mogą być zagadnienia wymagające operacji na transponowanych macierzach rzadkich, np. wymienione w pracy [18] problemy teorii grafów [36, 37], czy zastosowania metody najmniejszych kwadratów (*ang. least squares problem*) dla macierzy rzadkich [38], np. SLAM (*ang. simultaneous localization and mapping*) [39].

4.2.2 Sprzętowe metody zwiększania szybkości przetwarzania

Jedną z metod zwiększania szybkości przetwarzania cyfrowych maszyn obliczeniowych jest zmniejszenie czasu wykonania T podstawowych operacji arytmetycznych, oraz zmniejszenie obszaru A zajmowanego przez układy cyfrowe realizujące te operacje, co umożliwia umieszczenie większej liczby jednostek obliczeniowych w układzie scalonym. Niestety, dla obecnie stosowanych technologii, istotne zmniejszenie wartości parametrów A, T wydaje się już dość trudne: dla n-pozycyjnych argumentów szybkie układy sumatorów prefiksowych (*ang. parallel prefix adder, PPA*) pozwalają na wytworzenie wyniku po czasie $T(n) = O(log(n))$ przy użyciu układów o obszarze $A(n) = O(n \cdot log(n))$, natomiast złożoność szybkich układów mnożących wynosi $T(n) = O(log(n))$, $A(n) = O(n^2)$ przy zastosowaniu wielooperandowych sumatorów z przechowywaniem przeniesień (*ang. carry-save adder, CSA*) do sumowania iloczynów częściowych (por. [40, 41]).

4.2.2.1 Arytmetyka resztowa w szybkich cyfrowych układach arytmetycznych

Metodą potencjalnie umożliwiającą polepszenie parametrów A, T cyfrowych układów dodających i mnożących wydaje się wykorzystanie resztowych systemów liczbowych (*ang. Residue Number Systems, RNS*) i arytmetyki resztowej (*ang. residue arithmetic*). Resztowe systemy

liczbowe umożliwiają reprezentowanie liczby X w postaci wektora $X = \langle X_1, X_2, \ldots, X_r \rangle$, gdzie

$$X_i = X \bmod M_i = |X|_{M_i} \tag{10}$$

oznaczają reszty z dzielenia X przez wzajemnie pierwsze *moduły* M_i tworzące *bazę systemu*. Zakres dynamiczny systemu jest określony przez iloczyn modułów z bazy. Dla tak skonstruowanej reprezentacji, niektóre działania arytmetyczne (mnożenie i dodawanie) można wykonywać niezależnie na każdej z reszt, co pozwala na zrównoleglenie operacji składających się na pojedyncze działanie arytmetyczne:

$$X \odot Y = \left\langle |X_1 \odot Y_1|_{M_1}, |X_2 \odot Y_2|_{M_2}, \ldots, |X_r \odot Y_r|_{M_r} \right\rangle, \tag{11}$$

gdzie $\odot \in \{+, -, \cdot\}$. Ponieważ każda z reszt może być zapisana za pomocą znacznie mniejszej liczby pozycji, niż pełna reprezentacja liczby X w systemie naturalnym bądź uzupełnieniowym pełnym, możliwe jest istotne zmniejszenie zajmowanego obszaru układów, szczególnie mnożących, przy jednoczesnym zwiększeniu ich szybkości.

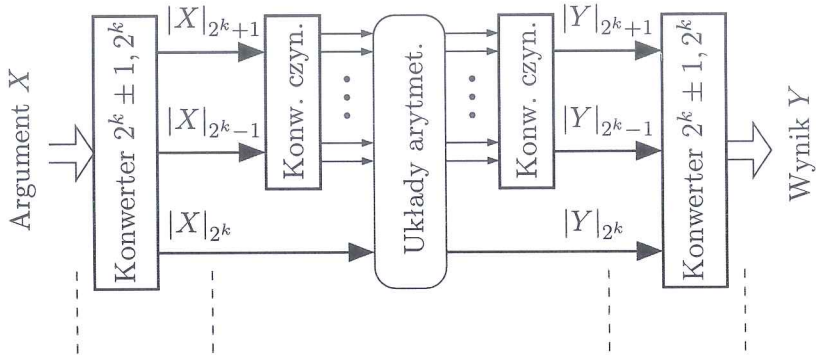
Niestety, pomimo potencjalnych zalet, szerokie wykorzystanie arytmetyki resztowej w maszynach obliczeniowych ogólnego przeznaczenia jest znacznie utrudnione poprzez szereg problemów. Do najbardziej istotnych należy zaliczyć: skomplikowane algorytmy pozostałych działań arytmetycznych (dzielenia, skalowania, porównywania liczb, a nawet wykrywania znaku liczby), konieczność stosowania kosztownych układów konwersji pomiędzy reprezentacjami w systemach naturalnych bądź uzupełnieniowych a reprezentacjami resztowymi, a także złożone metody zmiany zakresu systemu. Z powodu tych problemów, użycie resztowych systemów liczbowych w wielu typowych zastosowaniach, a w szczególności w układach obliczeniowych dla ustandaryzowanej arytmetyki zmiennoprzecinkowej, koniecznej w obliczeniach wysokiej wydajności, jest dość trudne (por. [42–48]).

W trakcie swoich badań nad szybkimi, równoległymi układami arytmetycznymi wykorzystującymi arytmetykę resztową zajmowałem się m.in. metodami zmniejszania złożoności zarówno konwerterów pomiędzy reprezentacjami resztowymi a wagowymi, jak i układów detekcji znaku liczby zapisanej w resztowym systemie liczbowym. Rozwiązanie tych dwóch problemów może także być podstawą opracowania szybkich metod dzielenia i porównywania reprezentacji resztowych, co w dalszej perspektywie może umożliwić szersze wykorzystanie arytmetyki resztowej. Konieczne są jednak dalsze prace tym kierunku.

W okresie bezpośrednio po uzyskaniu stopnia doktora prowadziłem badania w zakresie metod zmniejszania złożoności konwerterów pomiędzy reprezentacjami resztowymi a wagowymi, przy jednoczesnym utrzymaniu dużej szybkości operacji dodawania i mnożenia. Zostało to zrealizowane poprzez zastosowanie nowej klasy hierarchicznych resztowych systemów liczbowych (*ang. Hierarchical Residue Number System, HRNS*), zaprezentowanych w pracy [A6]. Wyniki opublikowane w pracy [A6] są istotnym rozwinięciem wstępnych prac wykonanych na potrzeby rozprawy doktorskiej i obejmują:

- Teoretyczną i eksperymentalną analizę złożoności cyfrowych układów arytmetycznych dedykowanych dla specjalizowanych układów scalonych (*ang. application-specific integrated circuit, ASIC*) – w rozprawie analizowano wyłącznie aplikacje dla układów reprogramowalnych FPGA (*ang. field-programmable gate array*).

- Równania konwersji i dedykowane dla układów ASIC struktury układów arytmetycznych dla wszystkich zbiorów modułów – w rozprawie skupiłem się na 3 zbiorach modułów obejmujących czynniki liczb $2^k \pm 1$ dla $k \in \{18, 24, 30\}$.



Rysunek 13: Struktura toru przetwarzania dla zaproponowanych HRNS.

4.2.2.1.1 Nowa klasa hierarchicznych resztowych systemów liczbowych

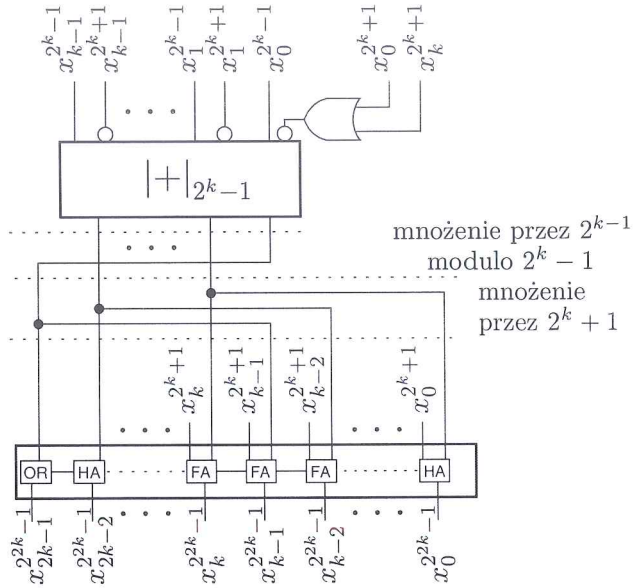
Na rys. 13 zaprezentowano uproszczony schemat toru obliczeniowego zbudowanego z wykorzystaniem zaproponowanej nowej klasy HRNS. Cechą szczególną zaproponowanych HRNS jest to, że występujące podczas konwersji operacje wyznaczania reszt dla szerokich wektorów bitowych mogą być przeprowadzane według uproszczonych algorytmów dla modułów 2^n oraz $2^n \pm 1$, dla których istnieją bardzo wydajne implementacje. Jednocześnie, operacje arytmetyczne mogą być wykonywane na stosunkowo niewielkich resztach z dzielenia przez czynniki modułów $2^n \pm 1$. Dla przykładu, możliwe jest zbudowanie HRNS o zakresie dynamicznym ok. 90 bitów, w którym najszerszym modułem różnym od 2^k jest moduł ok. 10-bitowy (1321, por. dolny wiersz tab. 2) i w którym ponad 1/3 zakresu dynamicznego (ok. 37 bitów) jest obsługiwane za pomocą modułów o szerokości od 3 do 6 bitów (moduły 7 do 61). Umożliwia to połączenie prostych układów konwersji z szybkimi i niewielkimi układami arytmetycznymi. Co więcej, ponieważ opracowana klasa HRNS jest rozszerzeniem 3-modułowego RNS o bazie $(2^n - 1, 2^n, 2^n + 1)$, jej dodatkową zaletą jest możliwość prostej detekcji znaku z wykorzystaniem opracowanej przeze mnie w czasie studiów doktoranckich szybkiej metody detekcji znaku dla RNS $(2^n - 1, 2^n, 2^n + 1)$, która, według mojej najlepszej wiedzy, była pierwszym na świecie rozwiązaniem wykorzystującym wyłącznie proste operacje arytmetyczne i nie wymagającym stosowania pamięci ROM, co przełożyło się na dużą szybkość i niską złożoność sprzętową. Metoda ta, wielokrotnie wykorzystywana przez różnych autorów, została przedstawiona w pracy

[B1] **T. Tomczak**, Fast sign detection for RNS $(2^n - 1, 2^n, 2^n + 1)$, *IEEE Transactions on Circuits and Systems*. 1, Regular Papers. 2008, t. 55, nr 6, s. 1502-1511.

4.2.2.1.2 Konwersja pomiędzy HRNS a systemem pozycyjnym

Dla zaproponowanej klasy HRNS, w pracy [A6] przedstawiono efektywne konwertery wyjściowe oparte na koncepcji określanej przez jej autorów jako nowe chińskie twierdzenie o resztach II (por. [49]). Zgodnie z tym twierdzeniem, konwersja wyjściowa może być zrealizowana poprzez pogrupowanie modułów w pary i przeprowadzenie obliczeń w strukturze drzewiastej o głębokości $log_2(r)$, gdzie r oznacza liczbę modułów w bazie. Pojedyncza operacja w dowolnym węźle drzewa (dla pary modułów M_i, M_j) jest opisana wzorem

$$|X|_{M_i \cdot M_j} = X_j + \left| \left| M_j^{-1} \right|_{M_i} \cdot (X_i - X_j) \right|_{M_i} \cdot M_j, \tag{12}$$



Rysunek 14: Schemat układu wyznaczającego resztę modulo $2^{2k}-1$ na podstawie reszt modulo $2^k \pm 1$. x_i^a oznacza bit na pozycji i binarnej reprezentacji $|X|_a$.

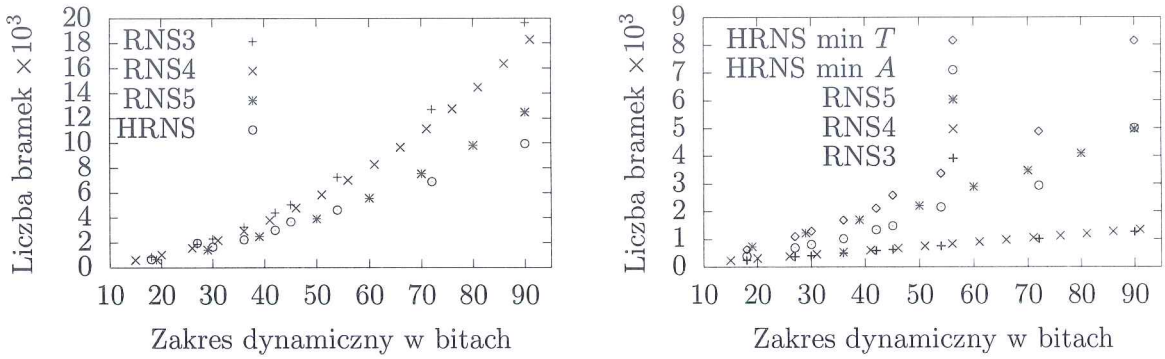
gdzie $\left|M_j^{-1}\right|_{M_i}$ oznacza odwrotność multiplikatywną M_j modulo M_i.

W zaproponowanej w pracy [A6] nowej klasie HRNS, dzięki użyciu modułów będących czynnikami liczb $2^n \pm 1$, możliwe jest zastosowanie szeregu optymalizacji operacji występujących w równaniu (12):

- Dla wielu modułów M_i, obliczenie różnicy $|X_i - X_j|_{M_i}$ można wykonać wykorzystując, wynikającą z tw. Eulera, właściwość okresowości reszt (por. [50]).
- Pary modułów można dobrać w taki sposób, aby wartości $\left|M_j^{-1}\right|_{M_i}$ były sumą niewielu potęg 2, co znacznie zmniejsza złożoność układów mnożenia przez te stałe. W niektórych przypadkach możliwe jest nawet uzyskanie wartości równych potędze dwójki, np. dla modułów będących czynnikami liczby $2^{30}+1$ w trakcie późniejszych etapów konwersji można wybrać parę $2501 = 41 \cdot 61$ i 13, co daje $\left|2501^{-1}\right|_{13} = 8$.
- Dla modułów postaci $2^{2k}-1 = (2^k-1)\cdot(2^k+1)$ konwersję można przeprowadzić w uproszonym układzie, w którym mnożenia przez stałe są zrealizowane jako przesunięcia i rotacje,

Tabela 2: Moduły $2^k \pm 1$ rozkładalne na niewielkie czynniki.

Moduł	Czynniki	Moduł	Czynniki
2^6-1	7, 9	2^6+1	5, 13
2^9-1	7, 73	2^9+1	19, 27
$2^{10}-1$	3, 11, 31	$2^{10}+1$	25, 41
$2^{12}-1$	5, 7, 9, 13	$2^{12}+1$	17, 241
$2^{14}-1$	3, 43, 127	$2^{14}+1$	5, 29, 113
$2^{15}-1$	7, 31, 151	$2^{15}+1$	9, 11, 331
$2^{18}-1$	7, 19, 27, 73	$2^{18}+1$	5, 13, 37, 109
$2^{24}-1$	5, 7, 9, 13, 17, 241	$2^{24}+1$	97, 257, 673
$2^{30}-1$	7, 9, 11, 31, 151, 331	$2^{30}+1$	13, 25, 41, 61, 1321



(a) Obszar układu mnożącego

(b) Obszar konwertera wyjściowego

Rysunek 15: Porównanie szacowanych obszarów układów mnożących i konwerterów wyjściowych dla różnych resztowych systemów liczbowych: RNS3 o bazie $(2^k-1, 2^k, 2^k+1)$, RNS4 o bazie $(2^k-1, 2^k, 2^k+1, 2^{2k+1}-1)$, RNS5 o bazie $(2^k-1, 2^k, 2^k+1, 2^{k-1}-1, 2^{k+1}-1)$, HRNS – proponowana nowa klasa HRNS. Dla konwertera wyjściowego HRNS podano wyniki na podstawie syntezy z użyciem oprogramowania Cadence Encounter RTL Compiler dla układów o najmniejszym obszarze (HRNS min A) oraz najszybszych (HRNS min T).

zgodnie ze schematem przedstawionym na rys. 14.

- W sytuacjach, w których zastosowanie powyższych optymalizacji jest trudne, zadowalające rezultaty można uzyskać grupując moduły w pary, w których jeden z modułów jest znacznie mniejszy od drugiego. Pozwala to na wykonywanie operacji modulo niewielkie liczby, np. przy użyciu małych pamięci ROM.

Szczegółowe opisy zastosowanych przekształceń i odpowiadających im struktur układów dla HRNS o zakresie dynamicznym $\leqslant 90$ bitów zamieszczono w pracy [A6] rozdz. 3.2 *Conversion from the HRNS* i 4.1 *Conversion circuits.*

Zaproponowana klasa HRNS umożliwia także uproszczenie układów konwersji z systemu pozycyjnego do systemu resztowego. Ponieważ dla części modułów w bazie HRNS wartości okresów bądź półokresów potęg dwójki modulo są sobie równe, lub są swoimi wielokrotnościami, niektóre fragmenty układów wyznaczania reszt mogą być współdzielone. Dla przykładu, w HRNS zbudowanym na bazie modułów $(2^{18}\pm1)$ układy wytwarzania reszt modulo 5, 7 i 13 mogą korzystać ze wspólnej części wyznaczającej resztę modulo $2^{12}-1$. Formalny opis reguł wydzielana wspólnych podukładów przedstawiono w pracy [A6] rozdz. 3.3 *Conversion to HRNS.*

4.2.2.1.3 Złożoność układów

Dla zaproponowanej klasy HRNS przeprowadzono analizę złożoności zarówno układów arytmetycznych (mnożących), jak i konwerterów, oraz porównano parametry układów z najnowszymi propozycjami resztowych systemów liczbowych [51, 52]. Parametry układów określono zarówno na podstawie ogólnych, teoretycznych modeli złożoności, jak i eksperymentalnie dla prototypowej implementacji wykorzystującej bibliotekę komórek standardowych FreePDK45nm [53]. Uzyskane wyniki potwierdziły, że w przypadku układów mnożących zastosowanie HRNS pozwala na podział pojedynczego mnożenia na większą liczbę niezależnych operacji, co daje prawie dwukrotne zmniejszenie obszaru w porównaniu z podstawowym RNS o bazie $(2^n-1, 2^n, 2^n+1)$ (por. rys. 15a). Dla systemów o zakresie dynamicznym większym niż 70 bitów, układy mnożące

wykorzystujące HRNS są mniejsze także niż 4- i 5-modułowe RNS. W przypadku zakresów dynamicznych 30-70 bitów nowe HRNS stanowią dobre uzupełnienie 5-modułowego RNS, ponieważ przy podobnej złożoności układu pozwalają na przeprowadzanie obliczeń w zakresach dynamicznych, dla których nie jest możliwe skonstruowanie RNS 5-modułowego. Można więc stwierdzić, że dla zakresów dynamicznych większych od 30 bitów zaproponowane HRNS umożliwiają zbudowanie układów arytmetycznych o lepszych parametrach, niż pozostałe RNS przebadane w pracy [A6].

W przypadku układów konwersji, dane z rys. 15b) pokazują, że dla HRNS możliwe jest skonstruowanie układów o obszarze porównywalnym lub mniejszym niż dla 5-modułowego RNS. Co więcej, mimo że konwertery dla HRNS wymagają większego obszaru niż wersje dla np. 3-modułowego RNS, to w większości przypadków różnice pomiędzy obszarem konwerterów dla HRNS i RNS o bazie $(2^n - 1, 2^n, 2^n + 1)$ są mniejsze, niż różnica obszarów dla pojedynczego układu mnożącego. Stąd, całkowity obszar zajęty przez kompletny tor obliczeniowy HRNS może być mniejszy nawet dla bardzo prostych układów zawierających pojedyncze mnożenie. W rzeczywistych przypadkach oszczędności mogą być znacznie większe, gdyż zazwyczaj resztowy tor obliczeniowy konstruuje się tak, aby zmaksymalizować liczbę operacji arytmetycznych w stosunku do liczby konwersji.

4.2.2.1.4 Wykorzystanie rezultatów

Ze względu na wymienione wyżej, nierozwiązane dotychczas w zadowalającym stopniu problemy z wykonywaniem operacji "trudnych", zakres zastosowań resztowych systemów liczbowych jest obecnie ograniczony głównie do aplikacji związanych z cyfrowym przetwarzaniem sygnałów (sztandarową aplikacją RNS są, wymagające dużej liczby mnożeń akumulacyjnych, wieloetapowe filtry o skończonej odpowiedzi impulsowej), kryptografią i tolerowaniem uszkodzeń (por. [54]). Zaproponowana klasa HRNS może oczywiście znaleźć zastosowanie w typowych aplikacjach resztowych systemów liczbowych, umożliwiając zwiększenie szybkości przetwarzania poprzez lepsze zrównoleglenie operacji składających się na pojedyncze działania arytmetyczne. Przykładem może być tutaj HRNS o zakresie 90 bitów, dla którego złożoność konwertera jest podobna, jak dla RNS 5-modułowego, a jednocześnie w HRNS działania arytmetyczne są wykonywane w 12 niezależnych kanałach, co skutkuje znacznie lepszym zrównolegleniem i parametrami układów. Dodatkowo, ze względu na konstrukcję pozwalającą na zmniejszenie złożoności wybranych operacji "trudnych", nowa klasa HRNS stanowi krok w kierunku znacznego poszerzenia zakresu zastosowań RNS. Konieczne są jednak dalsze badania, szczególnie w kontekście potencjalnej akceleracji obliczeń z użyciem reprezentacji zmiennoprzecinkowych, które są standardem w obliczeniach wysokiej wydajności.

4.3 Podsumowanie wyników zawartych w pracach [A1]-[A6]

W ramach omówionego powyżej cyklu powiązanych tematycznie publikacji [A1]-[A6] uzyskano następujące, oryginalne rezultaty:

1. Rozszerzono standardowy model złożoności pamięciowej i obliczeniowej algorytmów metody siatkowej Boltzmanna (LBM), będącej przykładem obliczeń szablonowych, dla implementacji, w których wydajność jest ograniczona przepustowością magistrali pamięci, wprowadzając pojęcie *narzutu* spowodowanego koniecznością przechowywania i przesyłania dodatkowych danych w celu obsługi geometrii rzadkich.

2. Przeprowadzono szczegółową analizę złożoności pamięciowej i obliczeniowej metod obsługi geometrii rzadkich w szybkich algorytmach metody siatkowej Boltzmanna dla procesorów

masowo równoległych, w szczególności akceleratorów graficznych. Wykazano, że w typowych sytuacjach narzuty dla dotychczasowych metod: macierzy połączeń CM i tablicy indeksów FIA, wynoszą ok. 25% dla CM i ok. 100% dla FIA.

3. Opracowano struktury danych, przeznaczone dla szybkich implementacji LBM dla geometrii rzadkich na procesorach masowo równoległych, w których to strukturach danych wykorzystano podział geometrii na niewielkie kafelki o rozmiarze dostosowanym do parametrów maszyny, a wewnątrz kafelków zastosowano dedykowane ułożenia danych umożliwiające znaczne ograniczenie liczby przesyłanych danych dzięki małej fragmentacji wewnętrznej transakcji blokowych z pamięcią.

4. Dla opracowanych struktur danych zaproponowano dwie wersje szybkich, drobnoziarniście równoległych algorytmów LBM dla geometrii rzadkich różniące się sposobem komunikacji pomiędzy węzłami z sąsiednich kafelków.

5. Przeprowadzono szczegółową analizę złożoności obliczeniowej i pamięciowej zaproponowanych algorytmów wykazując, że możliwe jest uzyskanie wydajności jedynie kilka procent mniejszej od maksymalnej wydajności idealnej implementacji dla geometrii gęstych.

6. Sformułowano uproszczone oszacowanie wydajności zaproponowanych algorytmów wykazując, że jest ona wprost proporcjonalna do średniej liczby węzłów obliczeniowych w niepustych kafelkach ϕ_t.

7. Uzyskane wyniki teoretyczne potwierdzono eksperymentalnie osiągając zarówno najwyższą wydajność dla algorytmów wykorzystujących zaproponowane struktury danych (w porównaniu z metodami CM i FIA), jak i liniową zależność wydajności od średniej liczby węzłów obliczeniowych w niepustych kafelkach.

8. Postawiono, i częściowo potwierdzono, hipotezę, że średnia liczba węzłów obliczeniowych w niepustych kafelkach zależy od rozmiarów spójnych obszarów zawierających węzły obliczeniowe, natomiast zależność od porowatości całej geometrii jest wtórna.

9. Opracowano format danych umożliwiający reprezentację macierzy rzadkich o nieregularnej liczbie elementów niezerowych w wierszach, dedykowany dla szybkich implementacji na procesorach masowo równoległych algorytmów numerycznego całkowania równań różniczkowych. Dla opracowanego formatu podano także szybkie algorytmy przeglądania macierzy wierszami i kolumnami.

10. Na podstawie zaproponowanego formatu danych opracowano wykorzystujące równoległość drobnoziarnistą wersje algorytmów PISO/SIMPLE dla siatek niestrukturalnych.

11. Zweryfikowano eksperymentalnie poprawność i wydajność zaproponowanego formatu danych i algorytmów w kompletnych realizacjach algorytmów PISO/SIMPLE dla masowo równoległych procesorów graficznych.

12. Opracowano prostą metodę przyspieszenia fragmentów gruboziarniście równoległych algorytmów numerycznego całkowania równań różniczkowych (zaimplementowanych w oprogramowaniu OpenFOAM) polegającą na przeniesieniu procedur rozwiązywania układów równań liniowych na masowo równoległy procesor graficzny wraz z wymaganymi konwersjami formatów macierzy rzadkiej.

13. Przeanalizowano wydajność procedury mnożenia macierzy rzadkiej przez wektor i procedury rozwiązywania układów równań liniowych realizowanych na masowo równoległym procesorze graficznym.

14. Zaproponowano nową, dedykowaną dla specjalizowanych układów scalonych ASIC, klasę hierarchicznych resztowych systemów liczbowych (HRNS), w których możliwe jest zrównoleglenie pojedynczych działań arytmetycznych (dodawania, mnożenia) poprzez ich podział na kilka-kilkanaście niezależnych, równocześnie wykonywanych operacji.

15. Dla zaproponowanej klasy HRNS wyprowadzono i zoptymalizowano równania umożliwiające efektywną implementację układów konwersji pomiędzy systemem resztowym a pozycyjnym.

16. Przeanalizowano teoretycznie i eksperymentalnie złożoność układów cyfrowych dla zaproponowanej klasy HRNS wykazując, że dla zakresów dynamicznych większych niż 30 bitów HRNS, w porównaniu z nowymi, 3-5 modułowymi RNS, umożliwia zmniejszenie obszaru zajmowanego przez kompletny tor obliczeniowy.

5 Omówienie pozostałych osiągnięć naukowo-badawczych

Głównym nurtem moich badań były metody przyspieszania obliczeń zarówno na poziomie oprogramowania, jak i sprzętu. Oprócz omówionych wyżej wyników wchodzących w zakres jednotematycznego cyklu publikacji, zajmowałem się również:

1. Algorytmami szybkiego numerycznego całkowania równań różniczkowych z wykorzystaniem bezkwadraturowej nieciągłej metody Galerkina.

2. Opracowaniem oprogramowania do numerycznych symulacji przepływu płynów w mikrokanałach z wykorzystaniem metody siatkowej Boltzmanna.

3. W ramach współpracy z przemysłem:

   (a) obecnie prowadzę prace badawczo-rozwojowe obejmujące opracowanie dedykowanego komputera obliczeniowego przyspieszającego obliczenia numerycznej mechaniki płynów,

   (b) brałem udział w pracach związanych ze zwiększaniem wydajności dedykowanych dla procesorów graficznych procedur rozwiązywania rzadkich układów równań liniowych poprzez użycie metody AMG (*ang. algebraic multigrid method*) w algorytmie wstępnego polepszenia uwarunkowania układu równań (*ang. preconditioner*),

   (c) zaprojektowałem i zaimplementowałem pierwszą wersję biblioteki SpeedIT Flow będącej kompletną, równoległą implementacją algorytmów PISO i SIMPLE na procesorach graficznych,

   (d) brałem udział w przygotowaniu pierwszej wersji biblioteki SpeedIT zawierającej implementacje na procesorach graficznych algorytmów algebry liniowej dla macierzy rzadkich.

Zagadnienia 1 i 2 opisano pokrótce odpowiednio w rozdz. 5.1 i 5.2, w których podano także wybrane publikacje. Zagadnienia 3.a, 3.b, 3.c i 3.d omówiono w rozdz. 5.3.1, 5.3.2, 5.3.3 i 5.3.4.

5.1 Algorytmy szybkiego numerycznego całkowania równań różniczkowych z wykorzystaniem bezkwadraturowej nieciągłej metody Galerkina

W 2018 r. rozpocząłem prace nad zwiększaniem wydajności algorytmów całkowania równań różniczkowych z wykorzystaniem nieciągłej metody Galerkina. Metoda ta wydaje się potencjalnie

oferować możliwość istotnego skrócenia czasu obliczeń, wymaga jednak podejścia interdyscyplinarnego koniecznego do optymalnego doboru wielu parametrów zarówno na poziomach sprzętu i oprogramowania, jaki i na etapie formułowania i przekształcania całkowanych równań. Wstępne wyniki zostały przedstawione w pracy

[D1] **T. Tomczak**, Bezkwadraturowa implementacja nieciągłej metody Galerkina dla geometrii jednowymiarowych, *Raporty Katedry Informatyki Technicznej Politechniki Wrocławskiej*, 2018, Ser. PRE nr 67, 53 s.

5.2 Algorytmy numerycznych symulacji przepływu płynów w mikrokanałach z wykorzystaniem metody siatkowej Boltzmanna

W latach 2011-2013, w ramach wstępnych prac na potrzeby grantu badawczego NCN nr N N501 042140 pod kierownictwem dr Romana Szafrana, „Projektowanie, modelowanie i fabrykacja mikroaparatów metodą bezpośredniego grawerowania laserowego", zaprojektowałem i zaimplementowałem oprogramowanie wykorzystujące bibliotekę Palabos do symulacji przepływu płynów w strukturze mikrokanałów. Istotną cechą przygotowanej implementacji była możliwość analizy, na podstawie szczegółowych raportów, podziału geometrii na prostokątne poddomeny, co docelowo pozwoliło na opracowanie algorytmów przedstawionych w pracach [A1] i [A2]. Użyteczną cechą zaprojektowanego rozwiązania była także prosta metoda zadawania parametrów symulacji, która pozwalała na szczegółową kontrolę układu węzłów w geometrii. Implementacja ta, oprócz podstawowego zadania polegającego na wygenerowaniu użytecznych wyników przedstawionych w pracy [C1], pozwoliła mi na zapoznanie się ze specyfiką algorytmu LBM, wytypowanie kierunków optymalizacji, oraz dostarczyła podstawowy szkielet interfejsu użytkownika wykorzystywany w późniejszych pracach.

[C1] R. Szafran, **T. Tomczak**, Wykorzystanie metody lattice-Boltzmann do symulacji mikroprzepływów w kanałach układów lab on a chip, *Inżynieria i Aparatura Chemiczna*, R. 52, nr 6, s. 568-569, 2013.

5.3 Współpraca z przemysłem

W ramach współpracy z przemysłem obecnie prowadzę badania w zakresie sprzętowego wspomagania obliczeń numerycznej mechaniki płynów. Wcześniej zajmowałem się przede wszystkim zagadnieniami związanymi z wdrożeniami wyników naukowych wchodzących w zakres publikacji [A4] i [A5], oraz z rozwojem powstałych aplikacji.

5.3.1 Opracowanie komputera przyspieszającego obliczenia CFD

Od stycznia 2019 współpracuję z firmą OKANE Hanke spółka jawna w zakresie realizacji projektu badawczego pt. „FlowOne - Innowacyjny komputer obliczeniowy przyspieszający obliczenia numerycznej mechaniki płynów oparty o dedykowany procesor obliczeniowy CFD." Celem prac jest opracowanie dedykowanego procesora obliczeniowego i zbudowanego na jego bazie komputera obliczeniowego przyspieszającego obliczenia numerycznej mechaniki płynów z wykorzystaniem nieciągłej metody Galerkina. Do moich zadań należy m.in. przeprowadzenie prac badawczo-rozwojowych obejmujących opracowanie koncepcji zrównoleglenia algorytmów nieciągłej metody Galerkina i ich adaptacja do implementacji sprzętowej, oraz opracowanie koncepcji dedykowanego procesora i całego systemu.

5.3.2 Metody przyspieszania procedur rozwiązywania układów równań liniowych

W 2013 realizowałem projekt finansowany z programu "Dolnośląski Bon na Innowacje" pt. "Opracowanie koncepcji technicznej wdrożenia nowego preconditionera w bibliotece CUSP w celu akceleracji symulacji biomedycznych na kartach GPU".

Głównym celem moich badań było opracowanie metody pozwalającej na efektywne użycie algorytmu AMG (*ang. algebraic multigrid method*) w procedurze wstępnego polepszenia uwarunkowania układu równań (*ang. preconditioner*) będącej częścią algorytmów rozwiązywania układów równań liniowych zaimplementowanych w bibliotece SpeedIT Flow (http://www.vratis.com/what-is-speedit-flow/). Do moich zadań należało także przeprowadzanie konsultacji z programistami odpowiedzialnymi za implementację zaprojektowanego rozwiązania oraz kierowanie całokształtem prac. W ramach realizacji projektu zaprojektowano struktury danych i algorytmy umożliwiające ograniczenie zapotrzebowania na pamięć przy utrzymaniu wysokiej wydajności, opracowano metody testowania pozwalające zweryfikować poprawność algorytmów i implementacji, oraz przeprowadzono analizę wydajności zaimplementowanego rozwiązania. Projekt był realizowany we współpracy z firmą Vratis sp. z o.o.

5.3.3 Masowo równoległe wersje algorytmów PISO i SIMPLE

W latach 2011–2012, w ramach realizacji projektu Zielony Transfer, „Efektywne metody akceleracji symulacji biomedycznych na platformie GPU", finansowanego ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego, Działanie 8.2, Program Operacyjny Kapitał Ludzki, byłem odpowiedzialny za projekt i implementację pierwszej wersji biblioteki SpeedIT Flow (http://www.vratis.com/what-is-speedit-flow/) będącej szybką implementacją na procesorach graficznych algorytmów symulacji przepływu płynów wykorzystujących metody PISO/SIMPLE do rozwiązywania równań Naviera-Stokesa. Poza implementacją algorytmów omówionych w pracy [A4] i rozdz. 4.2.1.2.1 niniejszego autoreferatu, wymagało to także przeniesienia na GPU obszernego zestawu procedur numerycznych, ich analizy wydajności oraz weryfikacji poprawności. W momencie publikacji było to oprogramowanie o funkcjonalności nie spotykanej dotychczas w literaturze – kompletne algorytmy PISO/SIMPLE dla siatek niestrukturalnych. Łączna liczba powstałych linii kodu była rzędu 100 000. Projekt był realizowany we współpracy z firmą Vratis sp. z o.o.

5.3.4 Masowo równoległe algorytmy algebry liniowej dla macierzy rzadkich

W 2010 r. brałem udział w przygotowaniu implementacji pierwszej wersji komercyjnej biblioteki SpeedIT (http://www.vratis.com/what-is-speedit/). Poza implementacją łącza do oprogramowania OpenFOAM wykorzystującego metodę opisaną w pracy [A5] i rozdz. 4.2.1.2.2 autoreferatu, do moich zadań należało zaprojektowanie i zaimplementowanie interfejsu programisty (ang. application programming interface, API), a także analiza wydajności biblioteki i przeprowadzenie testów poprawności. Projekt był realizowany we współpracy z firmą Vratis sp. z o.o.

6 Działalność dydaktyczna

W toku mojej pracy zawodowej, działalność dydaktyczna stanowiła istotne uzupełnienie prac naukowo-badawczych. Od 2013 r. systematycznie prowadzę wykład z arytmetyki komputerów, którego tematyka jest ściśle związana z moimi zainteresowaniami naukowymi, i do którego przygotowałem ogólnodostępny zestaw pomocy dydaktycznych w formie elektronicznej wersji prezentacji do wykładu i zbioru przykładowych zadań. Od początku studiów doktoranckich nieprzerwanie prowadzę także zajęcia pomocnicze (ćwiczenia, laboratoria i projekty) z zakresu arytmetyki

i architektury komputerów. W razie potrzeby zajmowałem się także prowadzeniem form pomocniczych z kursów związanych z zagadnieniami niskopoziomowymi: układami FPGA, mikrosterownikami, programowaniem współbieżnym i systemami operacyjnymi. Mój średni współczynnik wykonania pensum za lata 2012-2018 wynosi 1.315.

Wiele uwagi poświęciłem także indywidualnej pracy ze studentami: byłem opiekunem 18 prac magisterskich, z których 2 zostały rekomendowane do nagród, i 9 prac inżynierskich (dla jednego z dyplomantów przygotowałem list rekomendujący do KTH Royal Institute of Technology), a także opiekunem indywidualnego toku studiów.

Bibliografia

[1] A. Nguyen, N. Satish, J. Chhugani, C. Kim, P. Dubey, "3.5-D blocking optimization for stencil computations on modern CPUs and GPUs," *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, (Washington, DC, USA), s. 1–13, IEEE Computer Society, 2010.

[2] M. Krotkiewski, M. Dabrowski, "Efficient 3D stencil computations using CUDA," *Parallel Computing*, t. 39, nr 10, s. 533–548, 2013.

[3] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, (Piscataway, NJ, USA), s. 4:1–4:12, IEEE Press, 2008.

[4] Y. Zou, S. Rajopadhye, "A code generator for energy-efficient wavefront parallelization of uniform dependence computations," *IEEE Transactions on Parallel and Distributed Systems*, t. 29, s. 1923–1936, wrzesień 2018.

[5] M. Schulz, M. Krafczyk, J. Tölke, E. Rank, "Parallelization strategies and efficiency of CFD computations in complex geometries using lattice Boltzmann methods on high-performance computers," *High Performance Scientific And Engineering Computing* (M. Breuer, F. Durst, C. Zenger, eds.), (Berlin, Heidelberg), s. 115–122, Springer Berlin Heidelberg, 2002.

[6] A. Schäfer, J. Hammer, D. Fey, "Parallel simulation of dendritic growth on unstructured grids," *Proceedings of the 1st Workshop on Irregular Applications: Architectures and Algorithms*, IA3 '11, (New York, NY, USA), s. 15–22, ACM, 2011.

[7] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, E. Kaxiras, "A flexible high-performance lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries," *Concurr. Comput. : Pract. Exper.*, t. 22, s. 1–14, styczeń 2010.

[8] M. Bernaschi, S. Matsuoka, M. Bisson, M. Fatica, T. Endo, S. Melchionna, "Petaflop biofluidics simulations on a two million-core system," *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, s. 1–12, listopad 2011.

[9] M. Bernaschi, M. Bisson, M. Fatica, S. Melchionna, "20 petaflops simulation of proteins suspensions in crowding conditions," *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, s. 1–11, listopad 2013.

[10] C. Huang, B. Shi, Z. Guo, Z. Chai, "Multi-GPU Based Lattice Boltzmann Method for Hemodynamic Simulation in Patient-Specific Cerebral Aneurysm," *Communications in Computational Physics*, t. 17, nr 4, s. 960–974, 2015.

[11] C. Nita, L. Itu, C. Suciu, "GPU accelerated blood flow computation using the lattice Boltzmann method," *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, s. 1–6, wrzesień 2013.

[12] G. Wellein, T. Zeiser, G. Hager, S. Donath, "On the single processor performance of simple lattice Boltzmann kernels," *Comput. Fluids*, t. 35, s. 910–919, wrzesień 2006.

[13] G. Hager, G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 2010.

[14] J. D. C. Little, "A proof for the queuing formula: $L = \lambda W$," *Operations Research*, t. 9, nr 3, s. 383–387, 1961.

[15] R. LeVeque, D. Crighton, *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics, Cambridge University Press, 2002.

[16] C. Johnson, *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Dover Books on Mathematics Series, Dover Publications, Incorporated, 2012.

[17] M. G. Larson, F. Bengzon, *The Finite Element Method: Theory, Implementation, and Applications*, t. 10 *Texts in Computational Science and Engineering*. Springer-Verlag Berlin Heidelberg, 01 2013.

[18] H. Wang, W. Liu, K. Hou, W.-C. Feng, "Parallel transposition of sparse data structures," *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, (New York, NY, USA), s. 33:1–33:13, ACM, 2016.

[19] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.

[20] J. Ferziger, M. Peric, *Computational Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2001.

[21] A. M. Bruaset, A. Tveito, *Numerical Solution of Partial Differential Equations on Parallel Computers*. Berlin, Heidelberg: Springer, 2006.

[22] J. M. Cohen, M. J. Molemaker, *Parallel Computational Fluid Dynamics. Recent Advances and Future Directions*, rozdz. A Fast Double Precision CFD Code using CUDA, s. 414–429. DEStech Publications, 2010.

[23] M. Lefebvre, P. Guillen, J.-M. L. Gouez, C. Basdevant, "Optimizing 2D and 3D structured Euler CFD slovers on Graphical Processing Units," *Computers & Fluids*, s. 136–147, 2012.

[24] F. Salvadore, M. Bernardini, M. Botti, "GPU accelerated flow solver for direct numerical simulation of turbulent flows," *Journal of Computational Physics*, t. 235, s. 129–142, 2013.

[25] J. Tölke, M. Krafczyk, "TeraFLOP computing on a desktop PC with GPUs for 3D CFD," *International Journal of Computational Fluid Dynamics*, t. 22, nr 7, s. 443–456, 2008.

[26] E. Phillips, Y. Zhang, R. Davis, J. Owens, "Rapid aerodynamic performance prediction on a cluster of graphics processing units," *47th AIAA Aerospace Sciences Meeting*, nr AIAA 2009–565, 2009.

[27] G. R. Markall, D. A. Ham, P. H. Kelly, "Towards generating optimised finite element solvers for GPUs from high-level specifications," *Procedia Computer Science*, t. 1, nr 1, s. 1815–1823, 2010. ICCS 2010.

[28] A. Corrigan, F. Camelli, R. Löhner, F. Mut, "Semi-automatic porting of a large-scale fortran CFD code to GPUs," *International Journal for Numerical Methods in Fluids*, t. 69, nr 2, s. 314–331, 2012.

[29] J. Waltz, "Performance of a three-dimensional unstructured mesh compressible flow solver on Nvidia Fermi-class graphics processing unit hardware," *International Journal for Numerical Methods in Fluids*, 2012.

[30] A. Klöckner, T. Warburton, J. Bridge, J. Hesthaven, "Nodal discontinuous Galerkin methods on graphics processors," *Journal of Computational Physics*, t. 228, nr 21, s. 7863–7882, 2009.

[31] P. Castonguay, D. M Williams, P. E Vincent, M. López Morales, A. Jameson, "On the development of a high-order, multi-GPU enabled, compressible viscous flow solver for mixed unstructured grids," *20th AIAA Computational Fluid Dynamics Conference*, (Honolulu, Hawaii), czerwiec 2011.

[32] V. G. Asouti, X. S. Trompoukis, I. C. Kampolis, K. C. Giannakoglou, "Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on graphics processing units," *International Journal for Numerical Methods in Fluids*, t. 67, nr 2, s. 232–246, 2011.

[33] I. Kampolis, X. Trompoukis, V. Asouti, K. Giannakoglou, "CFD-based analysis and two-level aerodynamic optimization on graphics processing units," *Computer Methods in Applied Mechanics and Engineering*, t. 199, nr 9, s. 712–722, 2010.

[34] N. Bell, M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, s. 1–11, listopad 2009.

[35] H. G. Weller, G. Tabor, H. Jasak, C. Fureby, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in Physics*, t. 12, nr 6, s. 620–631, 1998.

[36] W. McLendon III, B. Hendrickson, S. J. Plimpton, L. Rauchwerger, "Finding strongly connected components in distributed graphs," *Journal of Parallel and Distributed Computing*, t. 65, nr 8, s. 901–910, 2005.

[37] S. Rajamanickam, E. G. Boman, "Parallel partitioning with Zoltan: Is hypergraph partitioning worth it?," *Graph Partitioning and Graph Clustering*, 2012.

[38] A. Björck, *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996.

[39] F. Dellaert, M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research, IJRR*, t. 25, s. 1181–1204, grudzień 2006.

[40] S. Winograd, "On the time required to perform addition," *J. ACM*, t. 12, s. 277–285, kwiecień 1965.

[41] S. Winograd, "On the time required to perform multiplication," *J. ACM*, t. 14, nr 4, s. 793–802, 1967.

[42] R. Dhanabal, V. Barathi, S. K. Sahoo, N. R. Samhitha, N. A. Cherian, P. M. Jacob, "Implementation of floating point MAC using Residue Number System," *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, s. 461–465, luty 2014.

[43] N. R. Samhitha, N. A. Cherian, P. M. Jacob, P. Jayakrishnan, "Implementation of 16-bit floating point multiplier using Residue Number system," *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, s. 195–198, grudzień 2013.

[44] A. Ghosh, S. Singha, A. Sinha, ""Floating Point RNS": A new concept for designing the MAC unit of digital signal processor," *SIGARCH Comput. Archit. News*, t. 40, s. 39–43, maj 2012.

[45] E. Kinoshita, K.-J. Lee, "A residue arithmetic extension for reliable scientific computation," *IEEE Transactions on Computers*, t. 46, s. 129–138, luty 1997.

[46] J.-S. Chiang, M. Lu, "Floating-point numbers in residue number systems," *Computers & Mathematics with Applications*, t. 22, nr 10, s. 127–140, 1991.

[47] F. J. Taylor, C. H. Huang, "A floating-point residue arithmetic unit," *Journal of the Franklin Institute*, t. 311, nr 1, s. 33–53, 1981.

[48] E. Kinoshita, H. Kosako, Y. Kojima, "Floating-point arithmetic algorithms in the symmetric residue number system," *IEEE Transactions on Computers*, t. C-23, s. 9–20, styczeń 1974.

[49] W. Wang, M. N. S. Swamy, M. O. Ahmad, Y. Wang, "A high-speed residue-to-binary converter for three-moduli $\left(2^k, 2^k - 1, 2^{k-1} - 1\right)$ RNS and a scheme for its VLSI implementation," t. 47, s. 1576–1581, grudzień 2000.

[50] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," t. 423, s. 68–77, styczeń 1994.

[51] B. Cao, C.-H. Chang, T. Srikanthan, "A residue-to-binary converter for a new five-moduli set," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, t. 54, s. 1041–1049, maj 2007.

[52] A. Molahosseini, K. Navi, C. Dadkhah, O. Kavehei, S. Timarchi, "Efficient reverse converter designs for the new 4-moduli sets $2^n - 1, 2^n, 2^n + 1, 2^{2n+1} - 1$ and $2^n - 1, 2^n + 1, 2^{2n}, 2^{2n} + 1$ based on new CRTs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, t. 57, s. 823–835, kwiecień 2010.

[53] J. E. Stine, J. Grad, I. Castellanos, J. Blank, V. Dave, M. Prakash, N. Iliev, N. Jachimiec, "A framework for high-level synthesis of system-on-chip designs," *International Conference on Microelectronic Systems Education*, s. 11–12, IEEE Computer Society, 2005.

[54] W. K. Jenkins, M. A. Soderstrand, C. Radhakrishnan, "Historical patterns of emerging residue number system technologies during the evolution of computer engineering and digital signal processing," *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, s. 1–5, maj 2018.