AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PROJEKT INŻYNIERSKI

pt.„Opracowanie modelu urządzenie-program dla wybranych solwerów układów równań liniowych wykonywanych na heterogenicznych architekturach sprzętowych”

Imię i nazwisko dyplomanta: **Mateusz Szewczyk**

Kierunek studiów: **Informatyka Stosowana**

Profil dyplomowania:  **Modelowanie i Technologie Informacyjne**

Nr albumu: **240197**

Opiekun: dr inż. Łukasz Rauch

Podpis dyplomanta: Podpis opiekuna:

Kraków 2013

***Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszy projekt inżynierski wykonałem(-am) osobiście i samodzielnie i że nie korzystałem (-am) ze źródeł innych niż wymienione w pracy.***

Kraków, dnia ………….… Podpis dyplomanta…………….

Spis treści

[1. Wstęp 4](#_Toc376712173)

[2. Heterogeniczne platformy obliczeniowe. 5](#_Toc376712174)

[2.1. Procesory graficzne GPU 5](#_Toc376712175)

[2.2. Porównanie wydajności oraz energochłonności procesorów CPU z GPU. 8](#_Toc376712176)

[2.3. OpenCL 11](#_Toc376712177)

[3. Przegląd solwerów układów równań liniowych 12](#_Toc376712178)

[3.1. Solwery iteracyjne 12](#_Toc376712179)

[3.2. Solwery bezpośrednie 14](#_Toc376712180)

[4. Solwery wybrane do badań. 16](#_Toc376712181)

[4.1. ViennaCl 16](#_Toc376712182)

[4.2. Charakterystyka wybranych solwerów. 16](#_Toc376712183)

[4.2.1. Metoda Gradientu Sprzężonego 16](#_Toc376712184)

[4.2.2. Ustabilizowana metoda Gradientów BiSprzężonych 18](#_Toc376712185)

[4.2.3. Uogólniona metoda najmniejszego residuum GMRES 20](#_Toc376712186)

[4.2.4. Metoda LU bez pivotingu 21](#_Toc376712187)

[4.2.5. Metoda multifrontalna. 22](#_Toc376712188)

[4.2.6. Zmodyfikowany solver eliminacji Gaussa. 23](#_Toc376712189)

[5. Testy solwerów 23](#_Toc376712190)

[6. Opracowanie modelu urządzenie-program 24](#_Toc376712191)

[7. Podsumowanie 25](#_Toc376712192)

[8. Bibliografia 26](#_Toc376712193)

# Wstęp

# Heterogeniczne platformy obliczeniowe.

Heterogeniczne platformy obliczeniowe to systemy, które wykorzystują więcej niż jeden rodzaj procesora. Są to wielordzeniowe systemy, których wydajność wzrasta nie tylko poprzez dodawanie kolejnych rdzeni, lecz także poprzez wprowadzenie wyspecjalizowanego przetwarzania i obsługi poszczególnych zadań. Architektura heterogeniczna składa się z systemu zawierającego kilka rodzajów procesorów. Zazwyczaj jest to procesor (CPU) oraz procesor graficzny (GPU) znajdujące się najczęściej na tej samej płycie głównej w celu uzyskania najlepszej wydajności. Procesory graficzne poza przetwarzaniem i renderowaniem grafik 3D, mogą także wykonywać skomplikowane obliczenia matematyczne na dużych zbiorach danych, podczas gdy procesor obsługuje system operacyjny oraz wykonuje tradycyjne zadania szeregowe.

Do końca 2010 roku niemalże każdy komputer stacjonarny posiadał wielordzeniowe procesory. Przetwarzanie wielordzeniowe jest jednak problematyczne. Dodatkowe rdzenie oraz pamięć podręczna powodują wzrost rozmiarów oraz wymagają większej ilości energii.

W tym samym czasie jakość i złożoność rozwiązań na procesorach graficznych znacznie wzrosła, co spowodowało, że zaczęto się nimi w większej mierze interesować. Dużą zaletą procesorów graficznych jest możliwość wykonywania przetworzeń wektorowych. Oznacza to, że mogą one wykonywać równoległe operacje na bardzo dużych zbiorach danych, zarazem pobierając znacznie mniej energii niż podobny tradycyjny procesor. To jest powodem, dla którego procesory graficzne są w stanie wyświetlać tak realistyczne grafiki w wielu grach. Początkowo były wykorzystywane jedynie do odciążenia procesora oraz zwiększenia wydajności grafiki 3D. Teraz stały się one jeszcze bardziej atrakcyjne ze względu na wykorzystanie ich do bardziej ogólnych celów, takich jak równoległe obliczenia i zadania programistyczne.[[1]](#footnote-1)

## Procesory graficzne GPU

Procesory graficzne oryginalnie były projektowane do wykorzystania w grach komputerowych, gdzie różne obiekty geometryczne są renderowane. Każdy element wyjściowej grafiki składa się z pikseli, a GPU wykorzystuje swój procesor, aby wyliczyć równolegle każdy z kolorów pojedynczychn pikseli. W tej chwili teoretyczny szczyt możliwości procesora graficznego wynosi ok. 3TFlops [6 Brodtkorb A.R., Dyken C., Hagen T.R., Hjelmervik J.M, Storaasli O.O., State of the art in heterogeneous computing, Scientific Programming, 18, 1-33, 2010.

7 Butrylo B., Musy F., Nicolas L., Perrussel R., Scorretti R., Vollaire C., A Survey of Parallel Solvers for the Finite Element Method in Computional Electromagnetics, The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, 23, 531 – 546, 2004.

8 Malkowski K. Lee I, Raghavan P., Irwin M. J., Conjugate Gradient Sparse Solvers: Performance-Power Characteristics, IPDPS,

9 Bahadir A.R., Ellerby F.B., On the performance of certain direct and iterative methods on equations arising on a two dimensional in situ combustion simulator, Applied Mathematics and Computations, 125, 347 – 359, 2002.

10 Babaoglu B., Application of Biconjugate Stabilized Method with spectral acceleration for propagation over terrain profiles, 2003, Praca magisterska http://www.thesis.bilkent.edu.tr/0002436.pdf

11 Barret R., Berry M., Chan T.F., Demmel J., Donato J. M., Dongarra J., Eijkhout V., Pozo R., Romine C., Van der Vorst H., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Society for Industrial and Applied Mathematics, 1st edition, 1987.

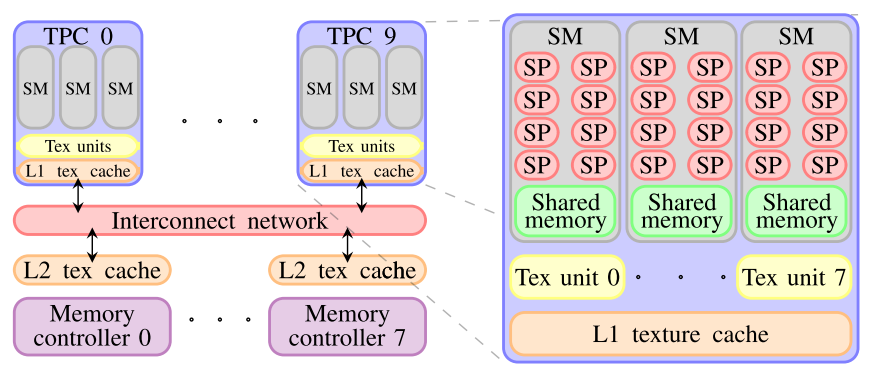
12 Galoppo N., Govindaraju N.K., Henson M., Manocha D., Lu-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphic Hardware, SC ’05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 3, 2005.

13 Guermouche A., L’Excellent J.Y., Utard G., Impact of reordering on the memory of a multifrontal solver, Parallel Computing 29, 1191-1218, 2003.

14 Wal P., Realizacja frontalnego solwera MES z wykorzystaniem technologii OpenCL, 2014

]. Ograniczeniem w przypadku GPU jest jednak to, że karty graficzne zazwyczaj zamontowane są na szynie PCI Express. Druga generacja PCI Express pozwala na transfer danych z prędkością 8 Gb/s pomiędzy pamięcią CPU i GPU, natomiast realna osiągalna prędkość wynosi ok. 5.2 Gb/s. GPU jest symetrycznym procesorem wielordzeniowym, który jest dostępny i sterowany wyłącznie poprzez procesor CPU, przez co stają się one systemem heterogenicznym. Procesor graficzny pracuje asynchronicznie z CPU, umożliwiając jednoczesne wykonanie i przeniesienie pamięci. W tej chwili głównymi producentami na rynku są AMD, Intel oraz Nvidia.

Przykładem procesorów GPU wykorzystywanej w architekturach heterogenicznych może być seria NVIDIA GT200 programowana zazwyczaj przy pomocy CUDA, która eksponuje model programowania SPMD przy użyciu dużej liczby wątków zorganizowanych w bloki. Wszystkie bloki obsługują ten sam program, a wątki w jednym bloku można synchronizować. Mogą one także komunikować się przy pomocy pamięci współdzielonej. Komunikacja między samymi blokami jest jednak ograniczona do operacji atomowych wykonywanych w pamięci globalnej. Bloki są automatycznie dzielone na osnowy, składające się z 32 nitek. Bloki są zaprojektowane tak, aby prowadziły strumieniowanie wieloprocesorowe (SM) w czasie wykonywania, a każda osnowa jest wykonywana w modelu SIMD. Jest to realizowane poprzez wykonywanie tych samych instrukcji przez cztery kolejne działania na ośmiu procesorach skalarnych. Rozbieżność w przepływie kodu pomiędzy wątkami obsługiwana jest sprzętowo poprzez automatyczne maskowanie wątków oraz serializację, przez co różnice w osnowach zmniejszają wydajność, a pomiędzy poszczególnymi osnowami nie ma wpływu na szybkość. Rysunek 1 przedstawia schemat architektury NVIDIA GT200.



*Rys. 1 Architektura NVIDIA GT200 GPU. Skróty mają następujące znaczenie: TPC – klastr do przetwarzania tekstur; SM – multiprocesor strumieniowy; Tex unit – jednostka tekstury, Tex cache – pamięć podręczna tekstur; SP – procesor skalarny. Źródło [6 Brodtkorb* A.R., Dyken C., Hagen T.R., Hjelmervik J.M, Storaasli O.O., State of the art in heterogeneous computing, Scientific Programming, 18, 1-33, 2010.

7 Butrylo B., Musy F., Nicolas L., Perrussel R., Scorretti R., Vollaire C., A Survey of Parallel Solvers for the Finite Element Method in Computional Electromagnetics, The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, 23, 531 – 546, 2004.

8 Malkowski K. Lee I, Raghavan P., Irwin M. J., Conjugate Gradient Sparse Solvers: Performance-Power Characteristics, IPDPS,

9 Bahadir A.R., Ellerby F.B., On the performance of certain direct and iterative methods on equations arising on a two dimensional in situ combustion simulator, Applied Mathematics and Computations, 125, 347 – 359, 2002.

10 Babaoglu B., Application of Biconjugate Stabilized Method with spectral acceleration for propagation over terrain profiles, 2003, Praca magisterska http://www.thesis.bilkent.edu.tr/0002436.pdf

11 Barret R., Berry M., Chan T.F., Demmel J., Donato J. M., Dongarra J., Eijkhout V., Pozo R., Romine C., Van der Vorst H., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Society for Industrial and Applied Mathematics, 1st edition, 1987.

12 Galoppo N., Govindaraju N.K., Henson M., Manocha D., Lu-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphic Hardware, SC ’05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 3, 2005.

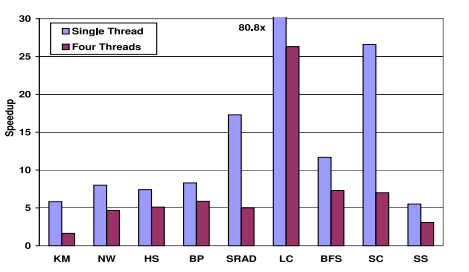
13 Guermouche A., L’Excellent J.Y., Utard G., Impact of reordering on the memory of a multifrontal solver, Parallel Computing 29, 1191-1218, 2003.

14 Wal P., Realizacja frontalnego solwera MES z wykorzystaniem technologii OpenCL, 2014

## 

## Porównanie wydajności oraz energochłonności procesorów CPU z GPU.

W 2009 roku zespół naukowców wykonał badania dotyczące wydajności procesorów graficznych [2]. Do badań użyto karty nVidia GeForce GTX 280 GPU z zegarem ustawionym na 1.3 Ghz oraz czterordzeniowego procesora Inter Core 2 Extreme CPU z zegarem ustawionym na 3.2 Ghz. Badania te wykazały, że w porównaniu z jednowątkową implementacją te same obliczenia wykonane na procesorze graficznym są szybsze od 5.5 do 80.8 razy, natomiast w przypadku czterowątkowej implementacji sięgają od 1.6 do 26.3 razy. Rysunek 2 przedstawia wyniki tych badań.

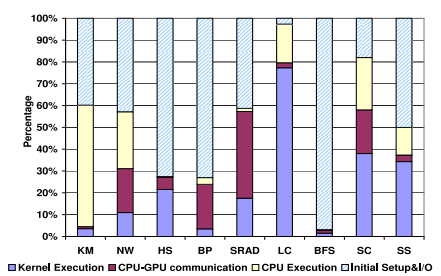


Rys. 2 Przyśpieszenie procesora graficznego w porównaniu z odpowiednikami obliczeń da jedno (Single Thread) i czterowątkowej (Four Threads) implementacji dla tradycyjnego procesora.

Powyższa analiza przeprowadzona została z wykorzystaniem różnych algorytmów. Skróty oznaczają kolejno:

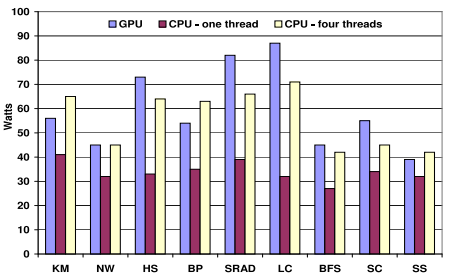
* KM – algorytm centroidów, który jest wykorzystywany w analizie skupień[[2]](#footnote-2).
* NW – algorytm Needlemana-Wunscha umożliwiający znalezienie optymalnego globalnego dopasowania dwóch sekwencji[[3]](#footnote-3).
* HS – HotSpot czyli dokładny i szybki model termiczny znajdujący zastosowanie w architekturze[[4]](#footnote-4) .
* BP – propagacja wsteczna, czyli jeden z podstawowych algorytmów uczenia nadzorowanego dla sieci neuronowych[[5]](#footnote-5).
* SRAD – algorytm dyfuzji oparty na cząstowych równaniach różniczkowych stosowany do usuwania plamek w obrazach [2].
* LC – wykrywa i śledzi białe krwinki naczyń krwionośnych w wideo mikroskopii wykorzystując GICOV [2]. [3]
* BFS – algorytm przeszukiwania grafu wszerz.
* SC – algorytm, który dla strumienia punktów wejściowych, znajduje mediany w taki sposób, aby każdy punkt był przypisany do najbliższego centrum2.
* SS – Similarity Score – algorytm stosowany w celu grupowania dokumentów WWW do obliczenia podobieństwa między nimi **Error! Reference source not found.**.

Badania wykazały wyższość rozwiązania wykorzystującego procesor graficzny. W każdym przypadku przyśpieszenie osiągnięte poprzez wykorzystanie GPU było znaczne. Powyższy wykres został opracowany nie biorąc pod uwagę operacji wejścia/wyjścia, a także początkowych ustawień, które ograniczają znacznie wydajność rozwiązań opartych na procesorach graficznych. Ilustruje to wykres przedstawiony na rysunku 3.



Rys. 3 Procentowy podział czasu na kolejne zadania.

Teoretyczną granicę wydajności dla zrównoleglonych obliczeń wyznacza prawo Amdahla. W rzeczywistości granica ta jest niższa od teoretycznego maximum. Spowodowane jest to czasem wymaganym przez procesor do wykonania odpowiednich rozkazów, a także komunikacji pomiędzy CPU i GPU. Warto zwrócić uwagę, że w przypadku algorytmu LC, gdzie komunikacja ta została ograniczona do minimum, wydajność była największa.



Rys. 4 Energochłonność GPU, CPU dla jednowątkowego oraz czterowątkowego rozwiązania

Ze względu na dużą ilość dostępnych rozwiązań związanych z wysoką wydajnością obliczeń, ważnym zagadnieniem jest również energochłonność poszczególnych rozwiązań. Z przeprowadzonych badań wynika, że jednordzeniowe obliczenia na CPU pobrały w każdym przypadku mniej mocy niż GPU. Porównując jednak GPU z czterowątkowymi obliczeniami proporcje te się zmieniają i CPU pobiera w niektórych przypadkach więcej mocy niż procesor graficzny. Ostateczne wyniki przeprowadzonych badań wykazały jednak, że biorąc pod uwagę stosunek wydajności do energochłonności zawsze lepiej wypada GPU.

## OpenCL

OpenCL jest pierwszym w pełni darmowym językiem programowania, którego celem jest wykonywanie obliczeń zrównoleglonych na architekturach heterogenicznych i którego popularność szybko rośnie. OpenCL pozwala programistom skupić się na pisaniu aplikacji, a nie architekturze, za pośrednictwem jednej przenośnej bazy kodu źródłowego. Podczas korzystania z OpenCL programista może korzystać z jednolitych narzędzi i języka, które będą działać na zrównoleglonych procesorach. API jest niskopoziomowe natomiast składnia przypomina w dużej mierze język C. [[6]](#footnote-6) Program napisany w OpenCL może być uruchomiony na wielu platformach, nie wymagając przy tym zmian w kodzie źródłowym.

# Przegląd solwerów układów równań liniowych

Solwery wykorzystywane do rozwiązania układów równań liniowych można podzielić na iteracyjne i bezpośrednie.

## Solwery iteracyjne

Solwery iteracyjne były używane w dużym zakresie we wczesnych latach 60’tych ubiegłego wieku, jednak z czasem zaprzestano wykorzystywania ich do obliczeń. Było to spowodowane odkryciem, że liczba operacji, które były przeprowadzane przez metody iteracyjne przekraczała w znacznym stopniu limity wyznaczone w założeniach teoretycznych. Zastąpione zostały one o wiele bardziej efektywnymi solwerami bezpośrednimi, opartymi o trójkątny rozkład macierzy. Wzrastająca moc obliczeniowa oraz możliwość zrównoleglania obliczeń sprawiła jednak ponowne ożywienie w świecie metod iteracyjnych. Możliwości nowych architektur spowodowały, że solwery iteracyjne były wydajniejsze niż bezpośrednie metody dla bardzo dużych problemów tj. większych niż 100 000 elementów [1]. Kolejną zaletą było o wiele mniejsze zużycie pamięci, ponieważ metody iteracyjne wymagają mniej więcej takiej samej ilości pamięci co oryginalne dane [4]. Jest to spowodowane tym, że współczynnik macierzy pozostawiany jest w oryginalnej formie i traktowany jako liniowy operator do wyliczania produktu wynikowego macierzy i wektora. Dodatkowo w przypadku solwera iteracyjnego użytkownik może wykorzystać zrównoleglanie obliczeń .

Iteracyjne metody rozwiązywania układów równań liniowych polegają na krokowym wyznaczaniu rozwiązania. Z każdą kolejną iteracją wynik jest poprawiany, aż do momentu, gdy wynik zbliży się wystarczająco do założonego przybliżenia. W związku z tym końcowy wynik solwera iteracyjnego jest zawsze obarczony pewnego stopnia błędem. Teoretycznie używając solwera iteracyjnego można uzyskać idealny wynik przy nieskończonej ilości kroków, jednak w rzeczywistych obliczeniach ilość iteracji jest zawsze wartością skończoną. Złe warunki początkowe mogą jednak spowodować duży błąd w ostatecznych wynikach[[7]](#footnote-7). Kiedy solwer iteracyjny rozwiązuje źle uwarunkowany problem, wyniki mogą być wolne lub nie mieć zbieżności. Głównymi operacjami metod iteracyjnych są mnożenia macierzy i wektorów oraz rozwiązywanie liniowych równań algebraicznych dotyczących uwarunkowań wstępnych [3]. Metody iteracyjne można podzielić na stacjonarne i niestacjonarne:

* Stacjonarne:
  + Jacobiego – metoda ta jest oparta na rozwiązaniu dla każdej zmiennej lokalnej w odniesieniu do pozostałych. Każda pojedyncza iteracja wprowadza przybliżenie dla każdej zmiennej. Metoda jest bardzo prosta do zrozumienia i zaimplementowania, jednak zbieżność jest bardzo wolna [11].
  + Gaussa-Seidel’a – jest bardzo podobna do metody Jacobiego jednak uaktualnia wartości zmiennych, kiedy tylko są dostępne. Sprawia to, że zbieżność jest szybsza jednak nadal wolna w porównaniu do metod niestacjonarych [11].
  + SOR – może być wyprowadzona z metody Gaussa-Seidel’a poprzez wprowadzenie parametru ekstrapolacji ω. Jeżeli ω zostanie dobrana optymalnie, metoda SOR osiąga zbieżność szybciej od metody Gaussa-Seidel’a o rząd wielkości [11].
  + SSOR – nie ma przewag na SOR jako samodzielna metoda iteracyjna, może jednak zostać zastosowania przy uwarunkowaniu wstępnym dla metod niestacjonarnych [11].
* Niestacjonarne
  + Gradientu Sprzężonego – Jest ona jedną z najlepiej znanych metod iteracyjnych wykorzystywaną do rozwiązywania liniowych układów równań z rzadką, symetryczną, dodatnio określoną macierzą[[8]](#footnote-8). W każdym kolejnym kroku wykonywane jest mnożenie macierzy i wektora [4]. Metoda gradientu sprzężonego jest bardzo skuteczną metodą, gdy macierz jest symetryczna, a współczynnik dodatnio określony [11].
  + MINRES – czyli metoda najmniejszego residuum, która jest alternatywą dla metody gradientu sprzężonego dla macierzy symetrycznej, ale bez określonego współczynnika [11].
  + GMRES – czyli uogólniona metoda najmniejszego residuum, wylicza sekwencję prostopadłych wektorów, podobnie do metody MINRES, łączy je dzięki metodzie najmniejszych kwadratów i aktualizuje. Jednak w przeciwieństwie do metod MINRES oraz Gradientów Sprzężonych wymaga przechowywania całej sekwencji w pamięci, także potrzebna jest duża ilość miejsca. Z tego powodu stosuje się ponowne uruchomienie tej metody. W ponownie uruchomionej wersji koszty obliczeniowe i przechowywania są ograniczone poprzez określenie liczby wektorów do wygenerowania. Sposób ten jest przydatny dla niesymetrycznych macierzy [11]. Badania dowiodły, że w metodzie GMRES stopień zbieżności może poprawiać się wraz z każdą kolejną iteracją [4,5].
  + BiCG – czyli metoda gradientu BiSprzężonego generuje dwie sekwencje wektorów podobnych do tych z metody Gradientów Sprzężonych. Jedna z nich oparta na oryginalnej macierzy wejściowej A, natomiast druga na transponowanej AT. Metoda ta, podobnie jak Gradientów Sprzężonych wykorzystuje ograniczoną ilość pamięci. Jest przydatna, kiedy macierz jest niesymetryczna i nieosobliwa, jednak zbieżność może być nieregularna i istnieje możliwość, że metoda nie skończy się. BiCG wymaga mnożenia macierzy ze swoim transponowanym odpowiednikiem przy każdej iteracji [11].
  + BiCGSTAB – czyli ustabilizowana metoda gradientu BiSprzężonego służy do rozwiązywania liniowych układów równań liniowych. Jest stabilniejsza od metody BiCG [11].
  + QMR – czyli metoda kwasi-najmniejszego residuum stosuje metodę najmniejszych kwadratów do rozwiązania i aktualizacji wektora błędów resztkowych BiCG, dzięki czemu wygładzone zostają problemy z nieregularną zbieżnością BiCG. QMR unika także w dużym stopniu załamań, które występują w BiCG. Z drugiej strony, nie wpływa znacząco na minimalizację błędu i, mimo że osiąga dobrą zbieżność, nie poprawia istotnie metody BiCG [11].

## Solwery bezpośrednie

Decydującym czynnikiem przy wyborze solwera bezpośredniego może być chęć użycia metody, która znacząco zmniejszy ilość niezerowych przejść podczas faktoryzacji macierzy. Dodatkowym atutem jest także możliwość wykorzystania pamięci dyskowej podczas dużych obliczeń na komputerach z małą ilością pamięci RAM i zachowanie przy tym wysokiej wydajności i prędkości podczas przetwarzania danych przechowywanych w pamięci RAM. Bezpośrednie metody pozwalają również na wykrywanie geometrycznej niestabilności modelu. Do tej pory najbardziej rozpowszechnionym solwerem bezpośrednim był solwer multifrontalny. Posiada on wszystkie wymienione powyżej zalety, jednak posiada także nadmierną ilość transferów pamięć-pamięć oraz pamięć-dysk-pamięć. Na wielordzeniowym komputerze w architekturze SMP stanowi to problem uniemożliwiający osiągnięcie maksymalnych osiągnięć i przyśpieszenia z wykorzystaniem dodatkowych procesorów. Powszechnym problemem z bezpośrednimi solwerami jest kwadratowa zależność liczby operacji w wymiarze problemu, przekraczanie ilości dostępnej pamięci RAM oraz zbyt duży czas faktoryzacji macierzy [3]. Podczas, gdy solwery są bardzo wydajne wymagają dużej ilości pamięci. Do solwerów bezpośrednich zalicza się:

* Metodę eliminacji Gaussa – bezpośredni solwer rozwiązujący układy równań pierwszego stopnia.
* Metoda eliminacji Gaussa- Jordana – ta metoda obliczająca odwrotność macierzy jest jedną z najstarszych. Jest mocna, dokładna w zakresie danej macierzy i nie wymaga wielokrotnej kontroli w zależności od typu macierzy biorącej udział w obliczeniach. Prace nad zrównolegleniem tego algorytmu zostały ograniczone ze względu na ograniczenia sprzętowe **Error! Reference source not found.Error! Reference source not found.Error! Reference source not found.**[**Error! Reference source not found.**].
* Dekompozycja LU – metoda ta sprowadza się do stworzenia dwóch macierzy trójkątnych dolnej oraz górnej (z ang. Lower – dolna Upper – górna) oraz rozwiązania odpowiedniego równania.

# Solwery wybrane do badań.

## ViennaCl

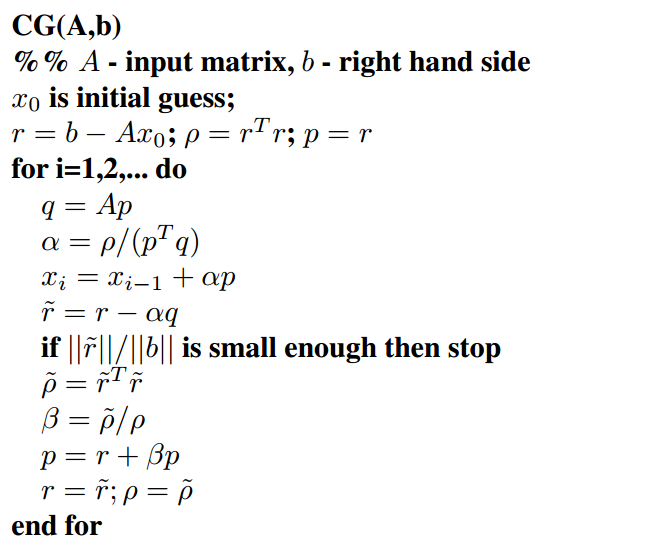
Do przeprowadzonych badań użyto solwerów wchodzących w skład otwartej biblioteki ViennaCL. Jest to biblioteka zaimplementowana w języku C++ umożliwiająca użytkownikowi wykonywanie obliczeń naukowych, zapewniając przy tym wsparcie dla CUDA, OpenCL oraz OpenMP. Zapewnia prosty dostęp do obliczeń wielowątkowych na architekturach takich jak procesory graficzne. ViennaCL wykonuje głównie obliczenia algebry liniowej, a także rozwiązuje równania za pomocą iteracyjnych i bezpośrednich metod[[9]](#footnote-9). Główne cechy biblioteki ViennaCL to:

* wygodne API napisane w języku C++ dla rzadkich i gęstych operacji algebry liniowej,
* wsparcie dla BLAS czyli wysokiej jakości procedur służących do przeprowadzania podstawowych operacji na macierzach i wektorach[[10]](#footnote-10). ViennaCL za pomocą BLAS wspiera takie operacje jak: wektor\*wektor, macierz\*wektor oraz macierz\*macierz,
* wsparcie dla obliczeń w technologii CUDA, OpenCL i OpenMP,
* trzy solwery iteracyjne: Gradientów Sprzężonych, Metodę Gradientów BI-Sprzężonych oraz GMRES,
* solwer bezpośredni: Metoda LU.

## Charakterystyka wybranych solwerów.

### Metoda Gradientu Sprzężonego

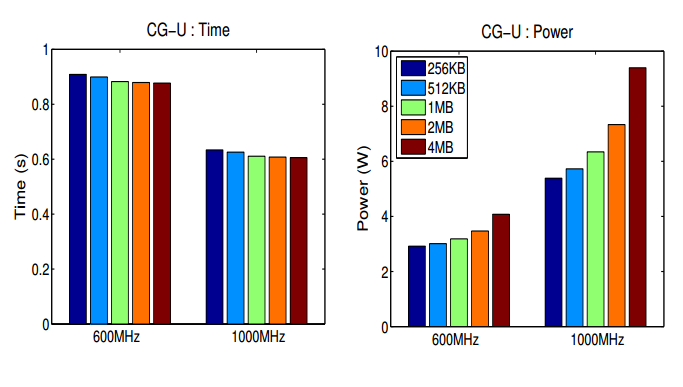
Jest to jedna z najpopularniejszych metod iteracyjnych ,ponieważ jest dobrze przystosowana do rozwiązywania układów macierzy rzadkich. W celu poprawy rozwiązań stosuje się różne techniki uwarunkowań wstępnych [7777]. Algorytm metody gradientów sprzężonych dla równania **A\*x = b\*** przedstawiony został na rysunku 5.



Rys. 5 Schemat algorytmu dla metody gradientów sprzężonych. A – macierz wejściowa b –wektor wejściowy x0- przypuszczany wynik. Źródło [8]

Powyższy schemat używa standardowych struktur danych do przechowywania macierzy rzadkiej A oraz wektory p, q, r, które przechowywane są w jednowymiarowych tablicach w pamięci. Pojedyncza iteracja wymaga jednego przemnożenia wektora z macierzą, dwóch wektorów, trzech operacji dodawania wektorów oraz dwóch operacji dzielenia liczb zmiennoprzecinkowych. Pośród tych operacji najbardziej czasochłonne jest przemnożenie wektora i macierzy, co zabiera 90% całego czasu operacji. Ponieważ macierz A jest macierzą rzadką, liczba zmiennych zmiennoprzecinkowych nie jest duża, co znacznie zmniejsza liczbę operacji na pamięci podczas przemnożenia. Dodatkowo pobieranie z pamięci elementów wektora *p* zależy od rzadkości macierzy A [8].

Rysunek 6 przedstawia wykonane badania na maszynie z procesorem PowerPC 440 z częstotliwością ustawioną na 600 MHZ, a podczas drugiego badania 1000 MHZ. Pamięć wykorzystana w maszynie to DDR2. Badania sprawdziły czas rozwiązania macierzy o rozmiarze 35588 x 35588 oraz moc, jaka została wykorzystana w trakcie badań dla różnych wielkości pamięci podręcznej.



Rys. 6 Czas obliczeń (po lewej) oraz moc wykorzystana w trakcie obliczeń (po prawej) dla metody gradientów sprzężonych. Źródło [8]

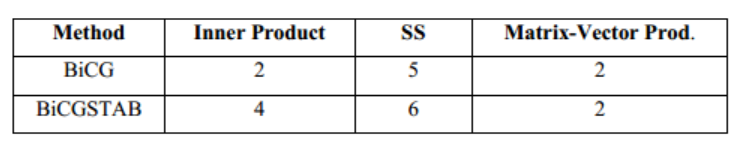
Każda iteracja metody gradientów sprzężonych wymaga:

* Jednego mnożenia macierz-wektor,
* Dwóch produktów wynikowych iloczynu skalarnego,
* Kilku mnożeń przez skalar oraz sumowań wektorów.

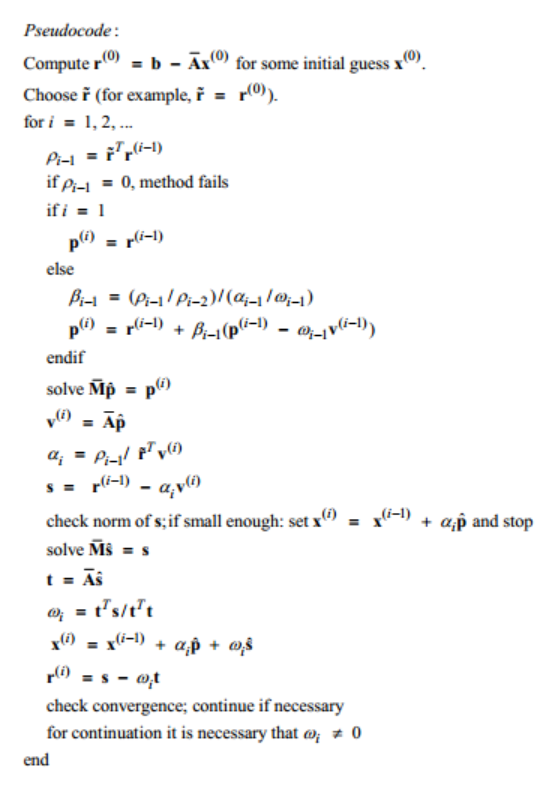
Jeżeli *A* jest macierzą rzadką z *O(N)* ilością miejsc niezerowych wtedy złożoność obliczeniowa każdej iteracji jest równa także *O(N)*. Dodatkowo jeżeli wartości własne są skupione, całkowity koszt obliczenia dobrego przybliżenia dla metody Gradientów sprzężonych będzie niski[[11]](#footnote-11).

### Ustabilizowana metoda Gradientów BiSprzężonych

Ustabilizowana metoda Gradientów BiSprzężonych (znana równierz jako BiCGSTAB) jest bardziej wydajną, dla niektórych przypadków, wersją BiCG. Zbieżność metody BiCGSTAB jest często nawet dwukrotnie szybsza niż w przypadku BiCG. Metoda ta ma dwa testy kończące. Działanie algorytmu BiCGSTAB zostało przedstawione na rysunku 8. Zbieżność może osiągnąć przy pierwszym teście na normę **s**,gdzie **s** jest relacją między bieżącym i poprzednim wektorem błędów resztkowych. Kiedy kończy się pracę metody na pierwszym teście, unika się kilku zbędnych operacji. BiCGSTAB tworzy o dwa więcej produkty wewnętrzne niż BiCG. Dokładne porównanie operacji w metodach BiCGSTAV oraz BiCG prezentuje rysunek 7. Są to dwa produkty wynikowe operacji mnożenia macierzy i wektora [10].



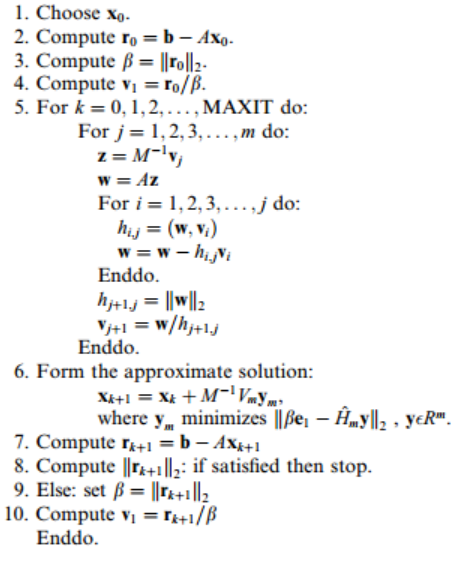
Rys. 7 Liczba operacji wykonywanych w każdej iteracji w metodzie BiCG oraz BiCGSTAB. Inner Product – produkt wewnętrzny. SS – wektor sum i różnic. Matrix-Vector Prod. – produkt operacji macierz-wektor. Źródło [10]



Rys. 8 Pseudokod przedstawiający algorytm BiGCSTAB. Źródło [10]

### Uogólniona metoda najmniejszego residuum GMRES

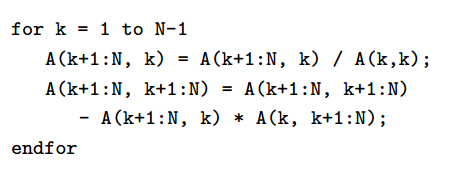
GMRES jest bardziej uniwersalna od metody Gradientów Sprzężonych ze względu na to, że macierz, którą rozwiązuje nie musi być symetryczna, ani też określona dodatnio[[12]](#footnote-12). Metoda polega na wstępnej aproksymacji x0, zdefiniowania w *n*-tej iteracji jako funkcja wektora szczątkowego *r0 = b – Ax0* oraz przybliżenia *zn* w przestrzeni Kryłowa Km(r0,A), które rozwiązywane jest za pomocą metody najmniejszych kwadratów [9]. Algorytm metody GMRES przedstawiony został na rysunku 9. Podobnie do metody MINRES generuje ciąg prostopadłych wektorów, jednak w przypadku braku symetrii nie można ich szybko odtworzyc. Zamiast tego wszystkie wcześniej wyliczone wektory z ciągu są przechowywane. Najbardziej popularna wersja GMRES jest oparta na zmodyfikowanej ortogonalizacji Grama-Schmidta oraz wykorzystuje restarty, aby kontrolować pamięć. Jeżeli restarty nie są wykorzystywane, GMRES osiągnie zbieżność w ilości kroków nie większych niż *n* (zakładając dokładną arytmetykę). Oczywiście w praktyce może być inaczej, jeżeli *n* jest duże, a ponadto wymagania dotyczące pamięci i mocy obliczeniowej są wygórowane [11].



Rys. 9 Algorytm GMRES. Źródło [9]

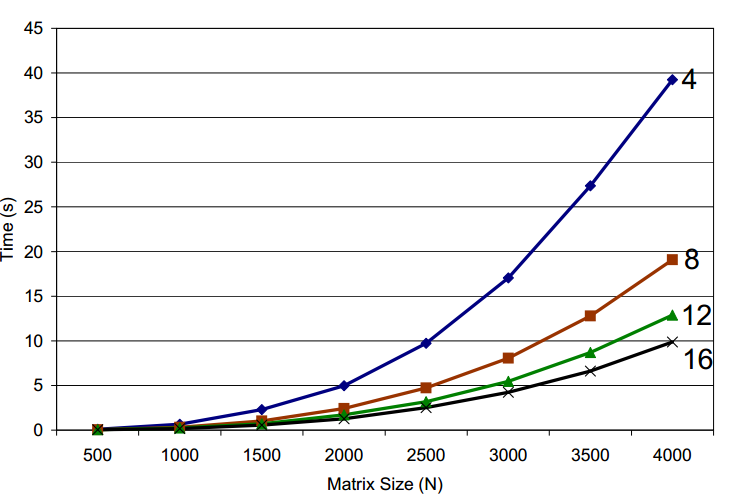
### Metoda LU bez pivotingu

Chcąc uzyskać macierz A bez odwracania możemy skorzystać z dwóch metod tj. metody Gaussa-Jordana lub dekompozycji LU. Biorąc pod uwagę wydajność obu metod oczywistym wyborem staje się ta druga.



Rys. 10 Rozkład LU. Źródło [12]

Rysunek 9 przedstawia algorytm dekompozycji LU. W *k*-tym kroku tylko dolna prawa część macierzy jest aktualizowana. Tak jak w przypadku eliminacji Gaussa-Jordana, odniesienia do pamięci w tym algorytmie są bardzo spójne, niezależnie od rasteryzacji w kolumnach lub rzędach. Powszechnie wiadomo, że złożoność obliczeniowa rozkładu LU jest trzy razy mniejsza niż Gaussa-Jordana [12].

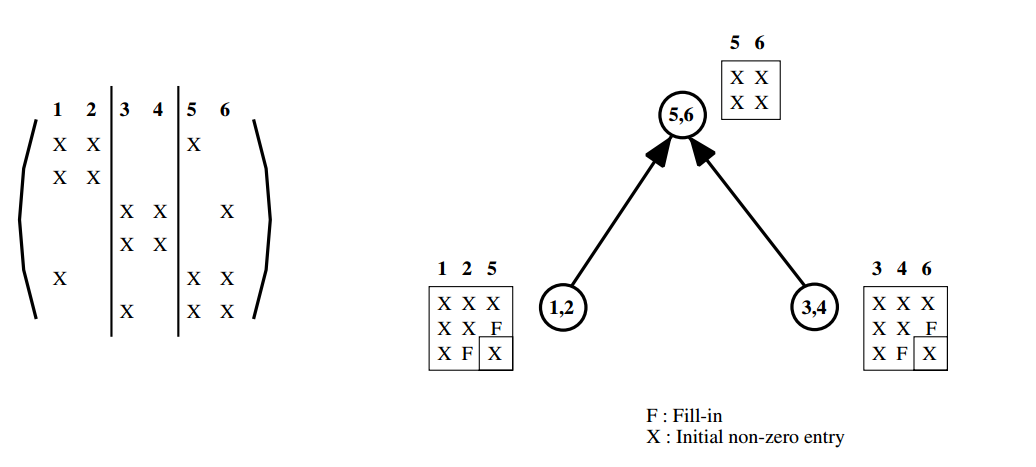


Rys. 11 Średni czas trwania dekompozycji LU w zależności od ilości włączonych rdzeni. Źródło [12]

Rysunek 10 przedstawia zbadaną prędkość metody LU bez pivotingu w zależności od rozmiaru macierzy oraz ilości włączonych rdzeni na procesorze graficznym Nvidia 6800 Ultra. Dane wskazują, że ilość rdzeni ma duży wpływ na czas trwania metody. Różnica między czterema a ośmioma rdzeniami jest duża, praktycznie podwojona zgodnie z ilością zwiększonych rdzeni. Taki sam wynik można zauważyć w przypadku ponownego zwiększenia rdzeni do szesnastu.

## Metoda multifrontalna.

Tak jak inne metody bezpośrednie metoda multifrontalna bazuje na drzewie eliminacji, które jest najmniejszą strukturą danych reprezentującą zależności pomiędzy operacjami w macierzy. W praktyce wykorzystywana jest struktura będąca zespołem drzew, uzyskana poprzez połączenie węzłów z drzewa eliminacji, które należą do tego samego superwęzła. Superwęzeł jest ciągłym zakresem kolumn posiadającym takie same dolne diagonalne niezerowe struktury. Rysunek 11 przedstawia przykładową strukturę macierzy wraz z drzewem. Z macierzy powstał zespół drzew złożony z super węzłów. Dwa węzły {1,2} oraz {3,4} są niezależne podczas, gdy {5,6} jest zależny od wyników w pozostałych.

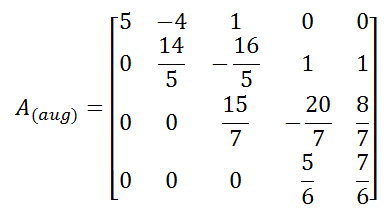


Rys. 12 Macierz z 3 superwęzłami ({1,2}, {3,4}, {5,6}) oraz powiązaniy zespół drzew. Źródło [13]

W multifrontalnym podejściu faktoryzacja macierzy wykonywana jest poprzez przeprowadzenie sukcesji częściową faktoryzacją małych gęstych macierzy nazywanych macierzami frontalnymi, które są połączone z węzłami drzewa. Kolejność we frontalnej macierzy jest dana przez numer niezerowych elementów pod diagonalną osią symetrii w pierwszej kolumnie superwęzła powiązanego z drzewem. Każda macierz frontalna jest podzielona na dwie części. Pierwsza odpowiada za zmienne, które podlegają faktoryzacji podczas działania algorytmu eliminacji w macierzy frontalnej. Druga odpowiada za zmienne aktualizowane podczas przetwarzania macierzy frontalnej. Kiedy częściowa faktoryzacja zostaje ukończona druga część zostaje przesłana do węzła-rodzica. Kiedy węzeł-rodzic otrzyma wyniki od wszystkich dzieci, mogą one zostać połączone (to znaczy zsumowane zawartymi w macierzy frontalnej rodziców). Algorytm eliminacji przechodzi wtedy po drzewie (węzeł rodzic nie jest przetwarzany przed dziećmi). Wykorzystywane są trzy przestrzenie pamięciowe. Jedna przechowuje współczynniki, druga stos z elementami z drugiej części macierzy frontalnej oraz ostatnia przechowywująca obecnie wykorzystywaną macierz frontalną. Podczas przechodzenia drzewa miejsce w pamięci wymagane przez czynniki stale rośnie, podczas gdy stos pamięci (zachowujący drugą część macierzy frontalnych) zmienia się w zależności od wykonywanych operacji.

## Zmodyfikowany solver eliminacji Gaussa.

Podstawowa metoda eliminacji Gaussa polega na przeprowadzeniu dwóch kroków. Pierwszym z nich jest eliminacja w przód, która polega na dodawaniu i odejmowaniu poszczególnych wierszy w taki sposób, aby powstała macierz trójkątna górna. Jest to macierz, która pod pod diagonalą ma jedynie zerowe wyrazy. Macierz tego typu została przedstawiona na rysunku 13



Rys. 13 Układ równań liniowych zapisany w macierzowej postaci. Źródło [14]

Macierz pod tą postacią może zostać przetworzona w kolejnym kroku tj. Podstawianiu wstecz. Ten etap polega na podstawieniu poszczególnych niewiadomych równania w taki sposób, aby uzyskać pełne rozwiązanie. Całość przeprowadza się od ostatniego wyrazu macierz, w tym wypadku jest to wyraz [4,4]. W ten sposób niewiadoma zostaje wyliczony dzięki wykorzystaniu podstawowych operacji arytmetycznych. Reszta podstawiania wstecz polega na wyliczaniu następnych niewiadomych, które dzięki znajomości poprzednich wyrazów wyliczane są w równie prosty sposób.

Solwer, który został wykorzystany w testach został przygotowany z myślą o rozwiązaniu układów liniowych z zakresu metody elementów skończonych, zaimplementowany w języku OpenCL. Do fazy eliminacji w przód wprowadzony został, w celu uzyskania najlepszej wydajności wektor przechowujący macierz oraz wektor prawej strony. Pomaga on w porównaniu miejsca wiersza w oryginalnej macierzy z miejscem w macierzy schodkowej. Potrzebny jest on ze względu na brak gwarancji wykonywania w danej kolejności grup roboczych w zrównoleglonym kodzie. Podstawienie wstecz także zostało w pewnym stopniu zmodyfikowane w porównaniu do oryginalnej metody. Przechodzenie po kolejnych indeksach niewiadomych nie jest przeprowadzane po macierzy od dołu do góry lecz po wcześniej utworzonym wektorze [14].

# Testy solwerów

# Opracowanie modelu urządzenie-program

# Podsumowanie

# Bibliografia

1 verschoorJouglard C.E., Coutinho A.L.G.A, A comparison of iterative multi-level finite element Solvers, Computers & Structures, 69, 655-670, 1998.

2 Shuai C., Boyer M., Jiayuan M., Tarjan D., Sheaffer J.W., Sang-Ha L., Skadron K., Rodinia: A Benchmark Suite for Heterogeneous Computing, Proc. of IEEE International Symposium on Workload Characterization, 44-54, 2009.

3 Fialko S.Y., [Iterative methods for solving large-scale problems of structural mechanics using multi-core computers](http://www.sciencedirect.com/science/article/pii/S1644966513000666), Archives of Civil and Mechanical Engineering, 14, 190-203, 2014.

4 Verschoor M, Jalba A.C.,Analysis and performance estimation of the Conjugate Gradient method on multiple GPUs, Parallel Computing, 38, 552-575, 2012.

5 Van der Vorst H.A., Vuik C., The superlinear convergence behaviour of GMRES, Journal of Computional and Applied Mathematics, 48, 327-341, 1993.

6 Brodtkorb A.R., Dyken C., Hagen T.R., Hjelmervik J.M, Storaasli O.O., State of the art in heterogeneous computing, Scientific Programming, 18, 1-33, 2010.

7 Butrylo B., Musy F., Nicolas L., Perrussel R., Scorretti R., Vollaire C., A Survey of Parallel Solvers for the Finite Element Method in Computional Electromagnetics, The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, 23, 531 – 546, 2004.

8 Malkowski K. Lee I, Raghavan P., Irwin M. J., Conjugate Gradient Sparse Solvers: Performance-Power Characteristics, IPDPS,

9 Bahadir A.R., Ellerby F.B., On the performance of certain direct and iterative methods on equations arising on a two dimensional in situ combustion simulator, Applied Mathematics and Computations, 125, 347 – 359, 2002.

10 Babaoglu B., Application of Biconjugate Stabilized Method with spectral acceleration for propagation over terrain profiles, 2003, Praca magisterska <http://www.thesis.bilkent.edu.tr/0002436.pdf>

11 Barret R., Berry M., Chan T.F., Demmel J., Donato J. M., Dongarra J., Eijkhout V., Pozo R., Romine C., Van der Vorst H., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Society for Industrial and Applied Mathematics, 1st edition, 1987.

12 Galoppo N., Govindaraju N.K., Henson M., Manocha D., Lu-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphic Hardware, SC ’05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 3, 2005.

13 Guermouche A., L’Excellent J.Y., Utard G., Impact of reordering on the memory of a multifrontal solver, Parallel Computing 29, 1191-1218, 2003.

14 Wal P., Realizacja frontalnego solwera MES z wykorzystaniem technologii OpenCL, 2014

1. <http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-computing/> [↑](#footnote-ref-1)
2. [http://pl.wikipedia.org/wiki/Algorytm\_centroidów](http://pl.wikipedia.org/wiki/Algorytm_centroid%C3%B3w) [↑](#footnote-ref-2)
3. <http://pl.wikipedia.org/wiki/Algorytm_Needlemana-Wunscha> [↑](#footnote-ref-3)
4. <http://lava.cs.virginia.edu/HotSpot/> [↑](#footnote-ref-4)
5. <http://pl.wikipedia.org/wiki/Propagacja_wsteczna> [↑](#footnote-ref-5)
6. <http://www.techdarting.com/2013/06/what-is-opencl.html> [↑](#footnote-ref-6)
7. <http://th-www.if.uj.edu.pl/zfs/gora/metnum12/wyklad05.pdf> [↑](#footnote-ref-7)
8. <http://www.icm.edu.pl/kdm/Metoda_gradient%C3%B3w_sprz%C4%99%C5%BConych_CG> [↑](#footnote-ref-8)
9. <http://viennacl.sourceforge.net/viennacl-about.html> [↑](#footnote-ref-9)
10. <http://pl.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms> [↑](#footnote-ref-10)
11. <http://www.math.montana.edu/~vogel/Courses/ECE580_2002/cg.pdf> [↑](#footnote-ref-11)
12. <http://mst.mimuw.edu.pl/lecture.php?lecture=mo2&part=Ch6#S2> [↑](#footnote-ref-12)