AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PROJEKT INŻYNIERSKI

pt.

„Realizacja frontalnego solwera MES z wykorzystaniem technologii OpenCL”

Imię i nazwisko dyplomanta: **Paweł Wal**

Kierunek studiów: **Informatyka Stosowana**

Nr albumu: **240202**

Opiekun: dr inż. Łukasz Rauch

Podpis dyplomanta: Podpis opiekuna:

Kraków 2014

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

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

Spis treści

[1 WSTĘP 4](#_Toc375647881)

[2 WPROWADZENIE TEORETYCZNE 5](#_Toc375647882)

[2.1 METODY ROZWIĄZYWANIA UKŁADÓW RÓWNAŃ LINIOWYCH 5](#_Toc375647883)

[2.2 METODA ELIMINACJI GAUSSA 6](#_Toc375647884)

[2.2.1 ELIMINACJA W PRZÓD 7](#_Toc375647885)

[2.2.2 PODSTAWIANIE WSTECZ 9](#_Toc375647886)

[2.3 MACIERZE 11](#_Toc375647887)

[2.3.1 CHARAKTERYSTYKA MACIERZY W METODZIE ELEMENTÓW SKOŃCZONYCH 11](#_Toc375647888)

[2.3.2 METODY PRZECHOWYWANIA MACIERZY RZADKICH 13](#_Toc375647889)

[2.4 ARCHITEKTURA OPENCL 15](#_Toc375647890)

[2.4.1 SKŁADNIKI ŚRODOWISKA OPENCL 15](#_Toc375647891)

[2.4.2 SKŁADNIKI ARCHITEKTURY OPENCL 15](#_Toc375647892)

[2.4.3 PARALELIZM DANYCH 15](#_Toc375647893)

[2.4.4 ZADANIA I GRUPY ROBOCZE 16](#_Toc375647894)

[2.4.5 ZARZĄDZANIE PAMIĘCIĄ 18](#_Toc375647895)

[3 IMPLEMENTACJA SOLWERA 19](#_Toc375647896)

[3.1 RÓWNOLEGŁA METODA GAUSSA 19](#_Toc375647897)

[3.1.1 ELIMINACJA W PRZÓD 19](#_Toc375647898)

[3.1.2 ZAPEWNIENIE FORMY SCHODKOWEJ MACIERZY PO WYKONANIU ELIMINACJI 19](#_Toc375647899)

[3.1.3 PODSTAWIENIE WSTECZ 22](#_Toc375647900)

[3.2 PROBLEMY RÓWNOLEGŁOŚCI MASOWEJ 22](#_Toc375647901)

[3.2.1 PODZIAŁ PROBLEMU NA ZADANIA 23](#_Toc375647902)

[3.2.2 WYKORZYSTANIE PAMIĘCI LOKALNEJ 23](#_Toc375647903)

[3.2.3 UNIKALNOŚĆ LOKALNA A GLOBALNA 25](#_Toc375647904)

[3.2.4 PRZYKŁAD DZIAŁANIA PROPONOWANEGO ALGORYTMU 26](#_Toc375647905)

[3.3 WYKORZYSTANE STRUKTURY DANYCH 28](#_Toc375647906)

[4 BADANIA WYDAJNOŚCI 29](#_Toc375647907)

[5 WNIOSKI 30](#_Toc375647908)

[6 BIBLIOGRAFIA 30](#_Toc375647909)

# WSTĘP

Ponad czterdzieści lat temu, w roku 1970, Bruce Irons opublikował swoją koncepcję programu rozwiązującego układy równań w postaci macierzy rzadkich metodą frontalną. Jego główną motywacją do opracowania tej techniki była nie tylko wydajność, ale również zużycie pamięci[5]. Irons pracował z komputerem ICT 1905, który w najlepszym razie mógł mieć 96 kilobajtów pamięci operacyjnej w postaci 32,768 słów po 24 bity. Nic więc w tym dziwnego, że poszukiwał rozwiązania jak najbardziej optymalnego pamięciowo.

Dziś co prawda pamięć jest znacznie mniej ograniczonym zasobem – przeciętny komputer biurowy dysponuje na ogół kilkoma gigabajtami, większość kart które wykorzystuje się do obliczeń nie będzie posiadała mniej niż gigabajt. Jak jednakże mówi przysłowie – apetyt rośnie w miarę jedzenia. Wraz ze wzrostem pamięci i mocy obliczeniowej wzrastają również rozmiary problemów – w tym tych z zakresu metody elementów skończonych - na które zasadzają się badacze.

Wraz z biegiem lat technologia obliczeniowa ewoluowała, kulminując obecnie w wielordzeniowych procesorach i koprocesorach, oraz – z samej swojej natury przystosowanych do operacji zmiennoprzecinkowych – kartach graficznych wykorzystywanych w zagadnieniach GPGPU. Szczególnie te ostatnie stanowią interesującą materię. Składają się z jednego lub więcej procesorów strumieniujących, z których każdy dysponuje na ogół dużą ilością (powyżej stu) rdzeni obliczeniowych, mogących wykonywać obliczenia już nie tylko równolegle, ale w sposób masowo równoległy, niedostępny dla urządzeń opartych na klasycznych procesorach. Dla kodu obliczeniowego uruchamianego na takich urządzeniach istnieje jednak szereg ograniczeń[4]: transfery z pamięci dostępu ogólnego (pamięci hosta) do pamięci urządzenia są kosztowne czasowo gdyż wymuszają okresy w których ani CPU, ani GPU nie wykonują obliczeń, operacje atomowe na pamięci globalnej są powolne, problem stanowi również nadmierna dywergencja wątków.

Niniejsza praca jest poświęcona implementacji solwera podążającego za ideą solwera frontalnego zaprezentowanego w 1970 roku przez Ironsa, rozumianej jako rozłożenie złożonego problemu na serię częściowo tylko od siebie zależnych, lżejszych pamięciowo i znacznie prostszych obliczeniowo pod-problemów. Celem przyświecającym tej pracy jest również stworzenie oprogramowania wykorzystującego w najlepszy dostępny sposób możliwości masowej równoległości oferowane przez nowoczesne, wysokowydajne urządzenia obliczeniowe.

Mimo iż oprogramowanie podąża za myślą Ironsa, postulowana przez niego metoda rozwiązywania układów równań liniowych nie została dosłownie zastosowana. Zasada matematyczna na której opiera się stworzone oprogramowanie jest wyprowadzona ze zmodyfikowanej metody eliminacji Gaussa. Modyfikacje mają na celu wykorzystanie mocnych stron zastosowanych urządzeń obliczeniowych, jednocześnie omijając ich ograniczenia i potencjalne słabości.

Do realizacji samego oprogramowania została wykorzystana architektura i zestaw bibliotek OpenCL. Każdy z wiodących producentów sprzętu obliczeniowego ma zazwyczaj swoją własną architekturę którą obsługują jego urządzenia – jest to na przykład oprogramowanie CUDA dla urządzeń firmy NVIDIA czy Stream od firmy ATI. W odróżnieniu od nich OpenCL jest otwartym standardem, którego implementacje dla konkretnych urządzeń należą wprawdzie do ich producentów, lecz jego zastosowanie jest możliwe na urządzeniach NVIDIA, ATI, Intel, a nawet na niektórych procesorach w architekturze ARM pod kontrolą systemu Android[12]. Standard OpenCL dostarcza warstwę abstrakcji, dzięki której urządzenie na którym wykonywany jest kod nie ma znaczenia dla twórcy oprogramowania, tak długo jak implementacja OpenCL dla tej platformy jest z nim zgodna.

Stworzony kod został przetestowany pod kątem wydajności dla szeregu macierzy o różnych rozmiarach i charakterystykach. [TODO: wymienić urządzenia, GT550Ti, GT540M, Tesla M2090, Intel(R) Xeon(R) CPU X5650 @ 2.67GHz, xeon phi?]

# WPROWADZENIE TEORETYCZNE

## METODY ROZWIĄZYWANIA UKŁADÓW RÓWNAŃ LINIOWYCH

Koniunkcję pewnej liczby równań liniowych w skład których wchodzi ten sam zestaw zmiennych nazywa się układem równań liniowych. Układ takowy można zapisać w postaci równania macierzowego:

Bądź jego ogólnej reprezentacji macierzowej:

Ze względu na wygodę, w przykładach umieszczanych w literaturze często stosowana jest również reprezentacja w postaci macierzy rozszerzonej:

Ze względu na czytelność, powyższa forma będzie stosowana w pozostałej części tego dokumentu.

Zostało zaproponowanych wiele metod ich rozwiązywania. Oprogramowanie komputerowe służące do ich rozwiązywania nazywane jest solwerami.

Metody służące do rozwiązywania układów równań liniowych można podzielić na dwie główne kategorie[7]:

1. Metody iteracyjne
2. Metody bezpośrednie

Do metod bezpośrednich zaliczana jest między innymi metoda eliminacji Gaussa oraz metoda faktoryzacji (dekompozycji) LU.

## METODA ELIMINACJI GAUSSA

Metoda eliminacji Gaussa składa się z dwóch zasadniczych faz, to jest fazy eliminacji w przód oraz fazy eliminacji wstecz. Uważna analiza obu tych faz pozwala wysnuć kluczowe w modyfikacji algorytmu dla oprogramowania równoległego wnioski.

### ELIMINACJA W PRZÓD

Pierwsza z nich opiera się na liniowej operacji algebraicznej[9], czyli na wiedzy, iż każde równanie w układzie równań liniowych może zostać zastąpione równaniem powstałym z połączenia tego równania z dowolnym innym występującym w tym układzie.

Przedstawiając układ równań w postaci macierzowej, daje to podstawy do stosowania jednej z macierzowych operacji elementarnych, czyli dodawania lub odejmowania od siebie wierszy bądź ich wielokrotności.

Celem fazy eliminacji w przód jest doprowadzenie układu równań w postaci macierzowej do górnej macierzy trójkątnej (znanej również jako macierz schodkowa). Dokonuje się tego eliminując - przy pomocy operacji elementarnych – pewną liczbę niewiadomych z poszczególnych równań, reprezentowanych przez wiersze w macierzy [1,9].

Prześledźmy przykład fazy eliminacji w przód za [1]. Na rysunku 1 dany jest układ równań w postaci macierzy rozszerzonej:

Rysunek 1. Układ równań liniowych w postaci macierzowej

Jest to układ równań wygenerowany przez prosty problem metody elementów skończonych. Na pierwszy rzut oka widać że jest symetryczna oraz pasmowa – mimo że pasmo to jest bardzo szerokie. Są to dwie przydatne właściwości macierzowych postaci równań opisujących problemy metody elementów skończonych.

Pierwszym krokiem eliminacji w przód jest wyeliminowanie zmiennych oraz przy pomocy pierwszego wiersza macierzy pomnożonego przez odpowiednią liczbę. Po przeprowadzeniu tej operacji, macierz wygląda jak na rysunku 2.

Rysunek 2. Układ równań liniowych w postaci macierzowej po pierwszym kroku eliminacji w przód

Następnym krokiem jest postąpienie analogicznie w stosunku do zmiennych oraz , co zostało uwidocznione na rysunku 3.

Rysunek 3. Układ równań liniowych w postaci macierzowej po drugim kroku eliminacji w przód

Ostatnim krokiem w przypadku tej przykładowej macierzy jest zredukowanie wyrazu , jak zostało to uwidocznione na rysunku 4.

Rysunek 4. Układ równań liniowych w postaci macierzowej po zakończeniu eliminacji w przód

Analizując przebieg tego przykładu można wysnuć dwa pomocne wnioski. Dla każdego równania, czyli wiersza macierzy, można – przy założeniu wykonywania operacji w sposób sekwencyjny – wyznaczyć ilość operacji elementarnych które są konieczne do doprowadzenia go do pożądanej postaci. W powyższym przykładzie dla wiersza 1 było to zero operacji elementarnych, wiersz numer 2 wymagał jednej operacji elementarnej, zaś wiersze 3 i 4 – po dwie. Wychodząc z powyższego wniosku postawić można kolejny - iż dla danej macierzy można szybko wyznaczyć maksymalną ilość operacji elementarnych konieczną do sprowadzenia jej do macierzy schodkowej przypadającej na wiersz. Jak łatwo zauważyć, ta ilość operacji jest związana bezpośrednio z szerokością pasma macierzy: .

Drugi wniosek wyprowadzić można z faktu, iż eliminacja w przód sprowadza macierz do postaci macierzy schodkowej. Cechą charakterystyczną tej macierzy jest to, iż pierwszy wyraz niezerowy w danym wierszu musi znajdować się na diagonali macierzy; bezpośrednio oznacza to, że żadne dwa wiersze nie mogą mieć identycznego pierwszego wyrazu niezerowego.

### PODSTAWIANIE WSTECZ

Drugą z faz rozwiązania układu równań liniowych przy pomocy metody Gaussa jest faza podstawiania wstecz (ang. back substitution). Etap ten przeprowadzany jest na macierzy sprowadzonej do górnej macierzy trójkątnej, czyli macierzy w postaci schodkowej utworzonej w fazie eliminacji w przód. Rysunek 5 przedstawia macierz w formie schodkowej gotową do przeprowadzenia fazy podstawiania wstecz.

Rysunek 5. Układ równań liniowych w postaci macierzy schodkowej

Kontynuując analizę tego przykładu za [9], zauważyć można iż wartość wyrazu może zostać wyliczona bez przeprowadzania żadnych dodatkowych operacji, podczas gdy pozostałe wyrazy wymagają operacji odpowiednio więcej. Wyraz zostaje więc wyliczony poprzez obustronne dzielenie jak uwidoczniono na rysunku 6.

Rysunek 6. Macierz schodkowa po przeprowadzeniu pierwszego kroku podstawienia wstecz

Wiedząc iż , możliwe jest teraz wyliczenie za pomocą równania danego trzecim wierszem macierzy: . Po wykonaniu odpowiednich operacji, uzyskujemy wynikową macierz przedstawioną na rysunku 7.

Rysunek 7. Macierz schodkowa po przeprowadzeniu drugiego kroku podstawienia wstecz

W drodze analogicznych działań dla dwóch pozostałych poszukiwanych wyrazów, oraz – danych równaniami opartymi odpowiednio na drugim i pierwszym wierszu macierzy w formie zaprezentowanej na rysunku 7 – uzyskiwana jest ostatecznie macierz przedstawiona na rysunku 8.

Rysunek 8. Finalna postać macierzy – rozwiązany układ równań liniowych

Macierz w tej postaci zawiera wyłącznie rozwiązanie układu równań liniowych, czyli cel zastosowania metody eliminacji Gaussa.

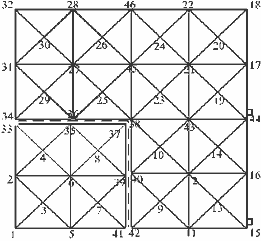
Na podstawie powyższego przykładu można wysnuć kolejny istotny wniosek. W przypadku fazy podstawiania wstecz, każdy kolejny krok – wyliczenie kolejnego wyrazu – wymaga wyliczenia wszystkich wyrazów którymi jest on dany, czyli inaczej wszystkich wyrazów od których jest zależny. Inaczej mówiąc, wyliczenie wyrazu wymaga – co można przyjąć jako ogólną regułę – wyliczenia wyrazów .

Drugim istotnym spostrzeżeniem jest fakt, iż kolejność rozwiązywania równań z dołu do góry, typowa dla górnej macierzy trójkątnej, jest pewną abstrakcją. W istocie nieistotnym jest, którym wierszem macierzy dany jest który wyraz rozwiązywanego układu równań, tak długo jak istnieje metoda identyfikacji właściwego dla danego wyrazu wiersza.

## MACIERZE

Proponowane rozwiązanie ma służyć przede wszystkim jako solwer dla układów równań liniowych przedstawionych w postaci macierzowej wygenerowanych przez oprogramowanie rozwiązujące problemy z zakresu metody elementów skończonych. Wychodząc z tego założenia można wyciągnąć pewne wnioski co do charakteru rozpatrywanych macierzy.

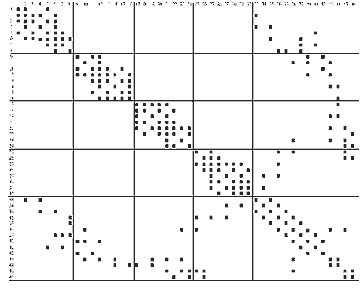
### CHARAKTERYSTYKA MACIERZY W METODZIE ELEMENTÓW SKOŃCZONYCH



Rysunek 8. Siatka elementów skończonych[[1]](#footnote-1)

Analizując siatkę elementów skończonych przedstawioną na rysunku 8, łatwo zauważyć iż każdy element, a nawet cała siatka jako taka, może być postrzegana jako rodzaj grafu nieskierowanego. Globalna macierz sztywności w metodzie elementu skończonego jest konstruowana na podstawie umieszczenia w niej w odpowiednich punktach – wynikających z numeracji węzłów w elementach oraz numeracji samych elementów – wartości z lokalnych macierzy sztywności wyliczonych dla poszczególnych elementów[8].

Wychodząc z powyższych założeń można powiedzieć, że macierz sztywności tworzona w metodzie elementów skończonych jest w swojej strukturze podobna do macierzy sąsiedztwa dla grafu nieskierowanego, które to macierze są inherentnie symetryczne. Intuicyjnie „jeśli wierzchołek 1 jest sąsiadem wierzchołka 2 to wierzchołek 2 jest sąsiadem wierzchołka 1”. Identyczną logikę – zamieniając jedynie słowo „wierzchołek” na „węzeł” – można zastosować w przypadku siatki elementów skończonych. Dzięki temu uzyskujemy pewność iż wygenerowana macierz będzie symetryczna.



Rysunek 9. Macierz sztywności dla metody elementów skończonych[[2]](#footnote-2)

Na rysunku 9 została uwidoczniona macierz sztywności dla przedstawionej powyżej siatki elementów skończonych. Na tym przykładzie dobrze uwidoczniona jest druga z interesujących cech macierzy którymi posługuje się metoda elementów skończonych: macierze te są rzadkie. W praktyce oznacza to iż wiele z ich elementów to zera.

W pracy[10] Reginald Tewarson postulował że macierz rzadka to taka, która przy wymiarach ma około niezerowych elementów, w praktyce – dwa do dziesięciu elementów niezerowych przypadających na każdy wiersz dla dużych . Są to wprawdzie definicje skonstruowane w zupełnie innych czasach – praca Tewarsona została opublikowana w 1973 roku – i niezbyt precyzyjne (nie jest na przykład sprecyzowane co autor miał na myśli przez duże ), lecz pozwalają na wytworzenie definicji intuicyjnej: macierz jest rzadka jeśli więcej niż połowa jej elementów to zera. Praktycznie każda macierz wygenerowana przez metodę elementów skończonych spełnia to założenie.

Specyficzną formą macierzy rzadkich są macierze pasmowe, czyli takie macierze rzadkie w których elementy niezerowe są skupione w paśmie ułożonym na przekątnej, którego środek wyznacza główna przekątna macierzy i zero lub więcej przekątnych po obu jej stronach. Macierze pasmowe bardzo dobrze nadają się do zastosowania metody Gaussa i jej pochodnych, gdyż z samej ich definicji wynika, iż niewiele operacji jest koniecznych by sprowadzić je do postaci macierzy schodkowej – wymagana jest jedynie eliminacja elementów znajdujących się pod główną przekątną macierzy.

W metodzie elementu skończonego uzyskanie macierzy pasmowych wymaga odpowiedniego ponumerowania elementów i węzłów. Niektóre solwery[3] integrują się z rozwiązaniem samego problemu metody elementów skończonych do tego stopnia że same zmieniają numerację węzłów i elementów by uzyskać korzystniejszą dla ich metody działania strukturę macierzy. Ujmuje to jednak takim rozwiązaniom uniwersalności i utrudnia ich implementację w rozwiązaniach zewnętrznych.

### METODY PRZECHOWYWANIA MACIERZY RZADKICH

Duże macierze rzadkie na ogół przechowuje się w pamięci komputera w postaci skompresowanej[10]. Pozwala to przechować większą macierz niż pozwalałaby na to – przy klasycznych metodach składowania – pamięć zastosowanego urządzenia, lub – odwrotnie – przechować daną macierz kosztem mniejszej ilości pamięci.

W kontekście solwera działającego przeważnie na kartach graficznych zużycie pamięci jest istotne z co najmniej dwóch powodów. Po pierwsze, dzięki kompresji można umieścić w pamięci urządzenia obliczeniowego większą macierz na raz. Po drugie, szybkość całkowitego działania solwera będzie związana między innymi z czasem transferu danych na urządzenie za pośrednictwem szyny danych PCI-Express[4]. Z tych dwóch względów konieczne jest rozważenie najpopularniejszych schematów przechowywania macierzy rzadkich.

Jednym z najprostszych schematów jest metoda koordynatowa (Coordinate Format), w której elementy niezerowe macierzy rzadkiej przechowywane są w formie tripletów , gdzie oraz stanowią odpowiednio wiersz i kolumnę w której znajduje się wartość . O ile najprostsza, metoda ta zużywa miejsc w pamięci (dla uproszczenia kalkulacji pominięta jest kwestia stricte implementacyjna, tj. użytych typów danych), gdzie to ilość niezerowych elementów w macierzy.

Wariacją na temat metody koordynatowej i jednym z popularniejszych schematów przechowywania macierzy rzadkich, jest metoda Compressed Sparse Row Format[[3]](#footnote-3) (CSR). W tej metodzie przechowywane w pamięci są trzy wektory.

Pierwszy z nich, o długości , zawiera niezerowe wartości z macierzy ułożone w kolejności od lewej do prawej oraz od góry do dołu macierzy. Drugi wektor zawiera indeksy kolumn w których konkretne wartości w pierwszym wektorze znajdowały się w macierzy (we właściwych sobie wierszach). Trzeci wektor zaś zawiera informację o tym od których indeksów zaczynają się poszczególne wiersze w dwóch poprzednich wektorach.

Przykład metody CSR przechowywania macierzy został przedstawiony na rysunku 10.

AA: 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0

JA: 1 4 1 2 4 1 3 4 5 3 4 5

IA: 1 3 6 10 12 13

Rysunek 10. Przykład macierzy zapisanej w formacie CSR

W tym przypadku ilość zużytej pamięci wynosi – dwa pierwsze wektory muszą mieć długość równą ilości niezerowych wartości w macierzy, trzeci zaś w typowym dla metody elementów skończonych przypadku – gdzie w macierzy nie ma wierszy w których nie ma żadnych danych – ma długość równą wymiarowi macierzy.

Ponieważ z reguły niezerowych elementów w macierzy wygenerowanej w metodzie elementów skończonych będzie więcej niż wynosi wymiar macierzy , metoda ta wygrywa kompresją z metodą koordynatową ( jest mniejsze niż ). Istnieją również wariacje na temat tej metody, najoczywistszym z których jest format Compressed Sparse Column (CSC), różniący się tym, iż drugi wektor przechowuje indeksy wierszy, a trzeci początki kolumn w pozostałych wektorach.

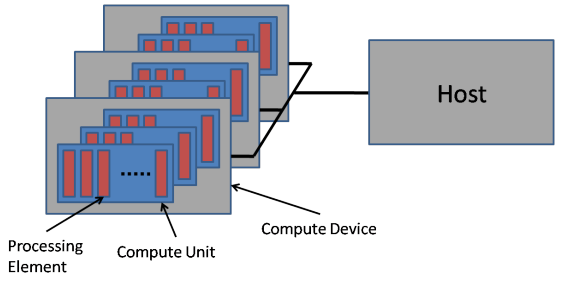
Podczas śledzenia zachowania metody Gaussa oczywistym stało się, iż ze względu na niemożność dynamicznej alokacji pamięci na urządzeniu oraz fakt iż popularne metody kompresji macierzy nie przewidują alokacji pamięci dla elementów zerowych które mogą ulec zmianie, dla solwera będzie konieczne zaprojektowanie uwzględniającej to metody kompresji macierzy stanowiącej kompromis pomiędzy wydajnością pamięciową, transferu a obliczeniową.

## ARCHITEKTURA OPENCL

Jak wspomniano we wstępie do niniejszej pracy, OpenCL (lub Open Computing Language) jest stworzonym przez Khronos Group otwartym standardem tworzenia oprogramowania dla nowoczesnych urządzeń obliczeniowych. Architektura ta posiada kilka specyficznych cech, których zrozumienie jest konieczne dla jej skutecznego wykorzystania.

### SKŁADNIKI ŚRODOWISKA OPENCL

Idea funkcjonowania platformy OpenCL została przedstawiona na rysunku 11. Środowisko składa się z hosta – „gospodarza”, który zleca wykonanie konkretnych obliczeń jednemu lub więcej urządzeniom obliczeniowym. Każde z nich posiada wiele jednostek obliczeniowych, każda z których z kolei posiada więcej niż jeden element przetwarzający.



Rysunek 11. Składniki środowiska dla architektury OpenCL[6]

### SKŁADNIKI ARCHITEKTURY OPENCL

Sama architektura składa się z trzech zasadniczych części[6]: specyfikacji języka, API platformy i API czasu wykonania.

Specyfikacja języka OpenCL definiuje składnię programów i kerneli które zostaną uruchomione z użyciem pozostałych dwóch części architektury. Jest oparty na standardzie ISO C99, lecz ze zmodyfikowanymi, dodanymi lub usuniętymi słowami kluczowymi, a także bez niektórych funkcjonalności.

### PARALELIZM DANYCH

OpenCL – podobnie jak karty graficzne, które są główną grupą urządzeń na których wykorzystuje się tą technologię – działa na zasadzie paralelizmu danych[13]. W praktyce oznacza to iż w przeciwieństwie do paralelizmu zadań, który zakłada wykonanie różnych zadań w tym samym czasie, OpenCL zakłada wielokrotne wykonanie identycznego (lub podobnego) zadania na elementach pewnego zbioru danych.

W tabeli 1 zostało przedstawione porównanie standardowego kodu skalarnego w języku C, oraz kodu paralelnego względem danych w języku OpenCL.

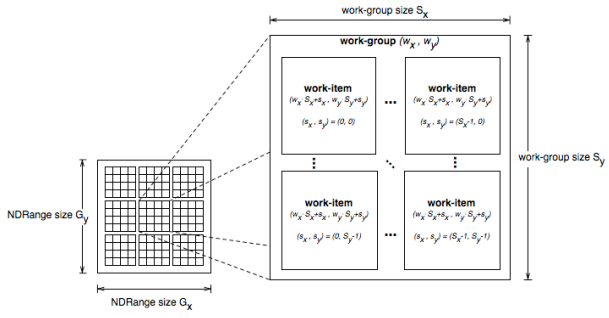
Tabela 1. Porównanie kodu paralelnego względem danych z funkcją skalarną (za [6]).

|  |  |
| --- | --- |
| Skalarna funkcja w języku C | Funkcja paralelna względem danych |
| void sqrt(int n, const float \*a, float \*res)  {  int i;  for(i=0; i<n; i++)  res[i] = a[i]\*a[i];  } | kernel void dp\_square  (global const float \*a, global float \*res)  {  int id = get\_global\_id(0);  res[id] = a[id] \* a[id];  } |

Paralelizm danych jest w OpenCL realizowany za pośrednictwem programów, składających się z jednego lub więcej kerneli. Program może też zawierać dodatkowe funkcje wykorzystywane przez kernele oraz stałe dane. Podczas wykonania danego kernela wiele jego kopii jest równolegle wykonywanych na elementach przetwarzających urządzenia obliczeniowego. Wykonania te zwane są work-itemami[13]. W dalszej części niniejszej pracy będą wykorzystywane terminy „zadanie” by określić work-item i „grupa robocza” by określić work-group.

### ZADANIA I GRUPY ROBOCZE

Zadania są powiązane ze sobą w grupy robocze, jak uwidoczniono na rysunku 12.



Rysunek 12. Zadania i grupy robocze w OpenCL [13].

Przedstawiona po lewej stronie powierzchnia – NDRange – to metoda organizacji pamięci i rozplanowania zadań w architekturze OpenCL. NDRange to N-wymiarowa przestrzeń (OpenCL pozwala na wykorzystanie jedno-, dwu- lub trójwymiarowej przestrzeni). Przestrzeń ta jest dzielona na grupy robocze.

Każda z grup roboczych dysponuje indeksami lokującymi ją w konkretnym punkcie NDRange. Każde zadanie – stanowiące odrębną instancję kernela – dysponuje unikalnym numerem identyfikacyjnym globalnym, oraz unikalnym wewnątrz grupy roboczej identyfikatorem lokalnym.

Z punktu widzenia paralelizmu danych, NDRange jest swoistym mapowaniem konkretnych instancji kernela na konkretne elementy obszaru pamięci który ma zostać przetworzony. Do dodania do siebie dwóch macierzy N x N można wykorzystać kernele uruchomione przy dwuwymiarowym NDRange z Gx = Gy = N. Do wyliczenia iloczynu skalarnego z kolei należałoby wykorzystać kernele uruchomione na jednowymiarowym NDRange.

Podział zadań na grupy robocze służy też do podziału barier na dwa rodzaje. W kernelach OpenCL można wykorzystywać bariery lokalne, które synchronizują tylko wątki wewnątrz grupy roboczej, bądź bariery globalne, które zsynchronizują wszystkie wątki działające na NDRange.

### ZARZĄDZANIE PAMIĘCIĄ

Jednym z istotnych założeń architektury OpenCL o którym należy pamiętać projektując dla niej oprogramowanie jest fakt, iż pojedyncza grupa robocza będzie wykonywana symultanicznie na jednej jednostce obliczeniowej (a wielu elementach przetwarzających). Ta wiedza przydatna jest podczas analizy modelu pamięci OpenCL.

W OpenCL wyróżniamy cztery osobne przestrzenie w pamięci[13]. Są to:

1. **Pamięć stała.** W tej kategorii pamięci przechowywane są wartości niezmienne podczas całego wykonania kernela. Może ona zostać zainicjalizowana przez „gospodarza”.
2. **Pamięć globalna.** Do tej pamięci mają dostęp (zapis i odczyt) wszystkie wątki we wszystkich grupach roboczych. Jeśli urządzenie na to zezwala, operacje na tej pamięci mogą być cache’owane. Jest to zazwyczaj największy dostępny obszar pamięci na urządzeniu obliczeniowym, ale najczęściej oferuje również najwyższy czas dostępu.
3. **Pamięć prywatna.** Do tego rodzaju pamięci ma dostęp tylko pojedyncze zadanie – to, które ją zaalokowało.
4. **Pamięć lokalna.** Pamięć współdzielona przez wszystkie wątki w danej grupie roboczej, niedostępna poza grupą roboczą. Jeśli jest to wspierane przez konkretne urządzenie, może być mapowana na obszar pamięci cache przy konkretnej jednostce obliczeniowej, dzięki czemu dostęp do takiej pamięci jest znacznie szybszy dla danej jednostki i uruchomionych na niej zadań. Jeśli nie ma takiej możliwości, jako pamięć lokalna dla danej grupy roboczej zostanie zamapowany pewien obszar pamięci globalnej, co oczywiście nie pozwoli na uzyskanie takiego przyspieszenia jak prawdziwa pamięć lokalna.

Tabela 2 pokazuje jakie urządzenia i w jaki sposób mogą alokować poszczególne rodzaje pamięci.

Tabela 2. Dostęp do pamięci oraz możliwości jej alokacji w OpenCL (za [13])

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Globalna | Stała | Lokalna | Prywatna |
| Gospodarz | Dynamiczna alokacja  Odczyt i zapis | Dynamiczna alokacja  Odczyt i zapis | Dynamiczna alokacja  Brak dostępu | Brak alokacji  Brak dostępu |
| Kernel | Brak alokacji  Odczyt i zapis | Statyczna alokacja  Odczyt | Statyczna alokacja  Odczyt i zapis | Statyczna alokacja  Odczyt i zapis |

# IMPLEMENTACJA SOLWERA

## RÓWNOLEGŁA METODA GAUSSA

W obliczu wysnutych w rozdziale 2.2 wniosków można pokusić się o analizę potencjalnych możliwości zrównoleglenia metody eliminacji Gaussa.

### ELIMINACJA W PRZÓD

Kluczowe wnioski jeśli chodzi o zrównoleglenie fazy eliminacji w przód zostały wyciągnięte w rozdziale 2.2.1. Cała macierz musi zostać sprowadzona do postaci macierzy schodkowej, co oznacza iż dla każdego wiersza konieczne jest wykonanie pewnej ilości eliminacji wyrazów.

Aby eliminacja pierwszego niezerowego wyrazu w danym wierszu była efektywna i nie spowodowała niepożądanych efektów, eliminację należy przeprowadzić przy pomocy wiersza o identycznej charakterystyce, tj. tym samym pierwszym niezerowym wyrazie. Co więcej, eliminacje takie muszą trwać, dopóki dla żadnego wiersza w macierzy nie będzie takiego innego wiersza, który miałby ten sam pierwszy niezerowy wyraz (zgodnie z faktem iż w macierzy schodkowej pierwszy niezerowy wyraz w wierszu jest unikalną charakterystyką każdego wiersza).

Można zatem wyznaczyć pewną operację która musi zostać wykonana na wszystkich elementach pewnego zbioru danych – czyli operacja eliminacji w przód może zostać zrównoleglona w zgodzie z paradygmatem paralelizmu względem danych.

### ZAPEWNIENIE FORMY SCHODKOWEJ MACIERZY PO WYKONANIU ELIMINACJI

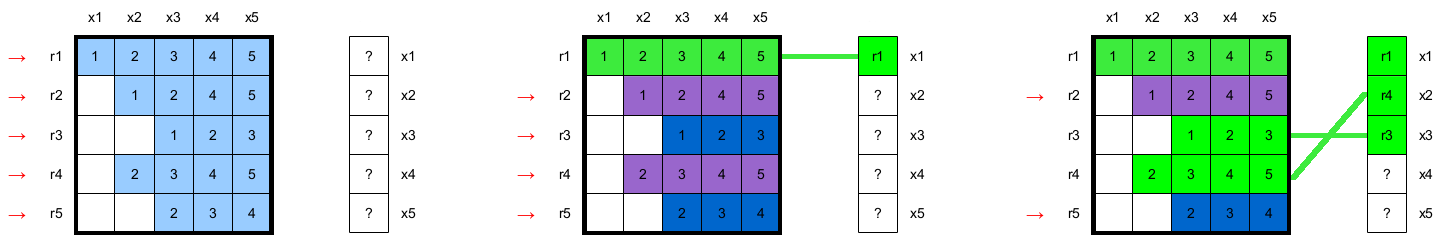
Powstaje wszakże inny problem – ponieważ OpenCL nie gwarantuje ani kolejności wykonania poszczególnych grup roboczych, ani zadań w ich obrębie, może się zdarzyć iż eliminacje – prowadzące skądinąd macierz do formy, którą można sprowadzić do macierzy schodkowej przy pomocy operacji zamiany wierszy – nie dadzą macierzy schodkowej.

Optymalne dla rozwiązania było również uniknięcie całkowicie operacji zamiany wierszy, gdyż ma znacznie większy narzut pamięciowy i obliczeniowy niż operacje dodawania i mnożenia wiersza przez wyraz (jedyne dwie operacje konieczne do przeprowadzenia eliminacji w przód).

Stąd do rozwiązania został wprowadzony dodatkowy wektor, obok pamięci przechowującej macierz oraz wektor prawej strony. Wektor ten stanowi swoiste mapowanie rzeczywistej pozycji wiersza w macierzy do pozycji, którą zajmowałby w macierzy schodkowej (zgodnie z założeniem że wiersz o pierwszym niezerowym wyrazie będzie zajmował -tą pozycję w macierzy).

Mapowanie takie rozwiązuje jeszcze jeden problem, mianowicie szybkiego zweryfikowania czy dany wiersz posiada unikalny pierwszy niezerowy wyraz, a jeżeli nie – to względem jakiego wiersza należy go wyeliminować; informacja ta jest zapisana w tymże dodatkowym wektorze i oszczędza kosztownych przeszukań dużych obszarów pamięci. Jedynym kosztem jest tutaj konieczność blokowania dostępu do wektora mapy na czas odczytu i zapisu informacji przez zadania.

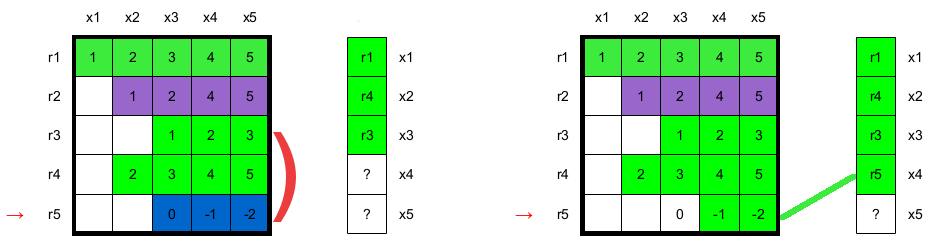
Na rysunku 13 można prześledzić pierwszy etap takiej operacji; przykład ten pokazuje działanie fazy eliminacji w przód proponowanego równoległego wariantu metody Gaussa w obrębie jednej grupy roboczej. Na rysunku przedstawiona została macierz oraz mapa; dla czytelności pominięty został wektor prawej strony.



Rysunek 13. Pierwszy krok równoległego wariantu metody Gaussa

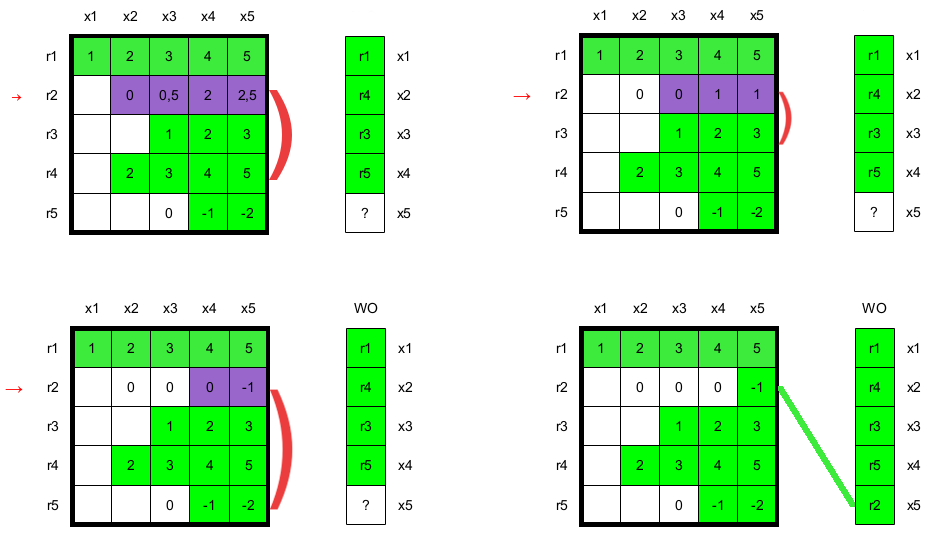
Zadanie operujące na wierszu r1 znajduje pierwszy niezerowy wyraz w kolumnie numer 1. Ponieważ przed nim dostępu do mapy nie uzyskał żaden inny wątek, wpisuje on swoje ID do mapy na odpowiedniej pozycji i zakańcza funkcjonowanie.

Podobnie dzieje się w przypadku wierszy r3 oraz r4; warto nadmienić tutaj, iż wiersze te zostały wybrane dla tego przykładu jako uzyskujące dostęp do mapy przed r2 oraz r5 by odwzorować typową dla OpenCL sytuację niesekwencyjnego przetwarzania zadań.



Rysunek 14. Drugi element przykładu równoległego wariantu metody Gaussa

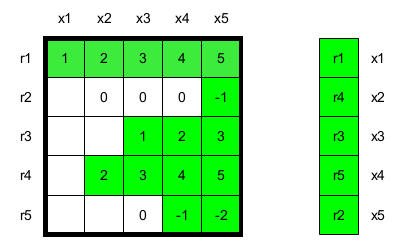
Przykład kontynuowany jest na rysunku 14. Przyjęto tym razem, iż w następnej kolejności dostęp do mapy uzyskał wiersz r5. Zadanie przetwarzające wiersz r5 znajduje pierwszy niezerowy wyraz w kolumnie 3, lecz to miejsce w mapie zajmuje już inny wiersz. Na wierszu r5 jest zatem przeprowadzana eliminacja przy użyciu znalezionego w mapie wiersza r3. Po zakończeniu eliminacji wiersz r5 ma teraz pierwszy niezerowy wyraz w kolumnie 4; zadanie znajduje niezajęte miejsce w wektorze mapy, wpisuje weń swoje ID i zakańcza wykonanie.



Rysunek 15. Trzeci element przykładu równoległego wariantu metody Gaussa

Rysunek 15 pokazuje ostatni element przykładu, w którym ostatnie zadanie, obsługujące wiersz r2, uzyskuje dostęp do wektora zawierającego mapę. Tym razem potrzebne są aż trzy eliminacje by wiersz spełnił warunek unikalnego pierwszego wyrazu niezerowego. Po wykonaniu eliminacji i odnalezieniu wolnego miejsca w wektorze mapy zadanie zakańcza wykonanie. Ze względu na to że wszystkie wątki skończyły pracę, wychodzi również cała grupa robocza.

W pamięci pozostaje macierz przygotowana do fazy podstawiania wstecz, uwidoczniona na rysunku 16.



Rysunek 16. Macierz przygotowana do fazy podstawiania wstecz przez równoległy wariant metody Gaussa.

### PODSTAWIENIE WSTECZ

Sposób przeprowadzenia podstawienia wstecz na macierzy wynikowej przedstawionej powyżej również wykorzystuje wektor mapy. Jedyną różnicą między tym wariantem a klasyczną metodą Gaussa jest to, iż podczas podstawiania wstecz algorytm porusza się od dołu do góry po wektorze mapy, wykorzystując wiersze o indeksach na które kolejno natrafia, zamiast po prostu – w sposób naiwny – poruszać się w górę po macierzy. W powyższym przykładzie kolejno przetworzone zostałyby wiersze r2, r5, r3, r4, r1.

Jeżeli chodzi o zrównoleglenie fazy podstawiania wstecz, wiedza zebrana w 2.2.2 wydaje się intuicyjnie sugerować iż nie będzie to wykonalne. W istocie, z punktu widzenia programowania równoległego, wykonanie podstawiania wstecz dla wiersza jest operacją zależną od wykonania podstawiania wstecz dla . Operacje uzależnione od siebie w ten sposób nie mogą być zrównoleglone w prosty sposób[2], czyli bez przeformułowania problemu tak, by usunąć z niego zależność następnego kroku od poprzednich.

## PROBLEMY RÓWNOLEGŁOŚCI MASOWEJ

Zaprezentowane zostało ogólne rozwiązanie problemu zrównoleglenia fazy eliminacji w przód w metodzie Gaussa. By dostosować to rozwiązanie do uruchomienia w architekturze OpenCL i uzyskać adekwatną wydajność, konieczne jest wykorzystanie dodatkowych wniosków wynikających z analizy funkcjonowania tej platformy przeprowadzonej w rozdziale 2.4.

### PODZIAŁ PROBLEMU NA ZADANIA

Zgodnie z proponowanym powyżej algorytmem, do każdego wiersza macierzy zostało przypisane jedno zadanie – OpenCL operuje więc na jednowymiarowym NDRange o rozmiarze co najmniej N dla macierzy N x N. W kodzie kernela OpenCL zostały wprowadzone rozwiązania zapobiegające przekroczeniu granic pamięci w przypadku wybrania przez użytkownika N większego niż wymiar macierzy.

Zadania są podzielone na grupy robocze. Każda z grup roboczych wykonywana jest na osobnej jednostce obliczeniowej w zakresie tego samego urządzenia obliczeniowego. Ponieważ zazwyczaj – by uzyskać dobrą saturację obliczeniami urządzenia obliczeniowego – grup roboczych jest więcej, niż dostępnych równocześnie jednostek obliczeniowych (dla kart graficznych – procesorów strumieniowych), każdą z tych grup roboczych lub bloków można rozwiązywać jako osobny front na którym prowadzone jest, niezależnie od pozostałych frontów, rozwiązanie układu równań w postaci macierzowej. Jest to podejście inne niż w pracy [5] proponował Bruce Irons – jak zostało wspomniane we wstępie, proponowana metoda jest bardziej intelektualnym spadkobiercą zaprezentowanej przez niego metody frontalnej rozwiązania układu równań niż dosłowną implementacją tejże metody.

### WYKORZYSTANIE PAMIĘCI LOKALNEJ

W proponowanym powyżej rozwiązaniu ogólnym wprowadzona została koncepcja wektora zawierającego mapowanie wierszy macierzy na ich docelowe pozycje w macierzy schodkowej. W praktyce mapa została zrealizowana jako wektor typu integer, o długości odpowiadającej wymiarowi macierzy. Pozycje nieobsadzone, tj. takie dla których jeszcze żaden z wierszy w macierzy nie umieścił informacji w mapie, ustawione są domyślnie na wartość -1.

Podczas analizy problemu okazało się, iż nie jest konieczne blokowanie dostępu do całego wektora mapy podczas wykonywania zadań. Z racji tego, iż dla danego wiersza w danym momencie musi być zapewniony synchroniczny dostęp tylko do jednej pozycji w tymże wierszu, konieczne jest zablokowanie dostępu tylko do tej jednej pozycji. Zarzucony został pomysł wykorzystania wykluczenia wzajemnego (mutexów); zamiast tego w kernelu zastosowano operacje atomowe, konkretnie wbudowaną w OpenCL funkcję atomic\_cmpxchg[[4]](#footnote-4).

Funkcja ta przyjmuje trzy parametry: wskaźnik na miejsce docelowe w pamięci, wartość do porównania z nim i wartość do ewentualnego podstawienia. W momencie więc kiedy zadanie rozpatrujące dany wiersz ustali jaki jest jego pierwszy wyraz niezerowy, wywołuje funkcję atomic\_cmpxchg w następujący (zapisany poglądowo) sposób:

atomic\_cmpxchg(&mapa[pierwszy\_wyraz\_niezerowy], -1, numer\_wiersza)

Dla takich danych funkcja atomic\_cmpxchg porówna pozycję w wektorze mapa podaną pozycję z -1. Jeżeli są takie same, na miejsce to zostanie wstawiona wartość zmiennej numer\_wiersza, a funkcja zwróci -1. Jeżeli nie są, tj. w mapie na podanej pozycji jest już różny od -1 wpis, na mapie nie zostanie wykonana żadna operacja, funkcja natomiast zwróci wartość która została znaleziona w mapie. W ten sposób wywołaniem jednej funkcji w mapie zostaje umieszczona właściwa informacja, bądź do zadania trafia informacja o tym, względem którego wiersza należy wyeliminować pierwszy wyraz niezerowy w aktualnie przetwarzanym wierszu.

Najprostszym i najbardziej intuicyjnym rozwiązaniem byłoby umieszczenie pojedynczej mapy w pamięci globalnej, gdzie mogą ją osiągnąć wszystkie zadania. Jednakże jak zostało nadmienione wcześniej, operacje atomowe – których nie da się tutaj uniknąć – na pamięci globalnej są inherentnie powolne. Znacznie lepszym w kontekście prędkości wykonania jest przechowanie mapy w pamięci lokalnej.

W praktyce w kernelach OpenCL zastosowanych w rozwiązaniu dla każdej grupy roboczej tworzona jest osobna mapa, w identycznej postaci jak ta zaproponowana powyżej w opisie równoległego algorytmu. Pewnym problemem była niemożność dynamicznej alokacji odpowiedniego rozmiaru pamięci z poziomu samego wykonania kernela. Trudność ta została jednakże ominięta przez wykorzystanie w kernelu następującego zapisu:

\_\_local int localMap[--TAG\_LOCAL\_MAP\_SIZE--];

Z punktu widzenia OpenCL, pomijając wpis --TAG\_LOCAL\_MAP\_SIZE--, jest to poprawne żądanie statycznej alokacji pamięci lokalnej. Ponieważ kernele OpenCL są przechowywane w plikach tekstowych lub zmiennych znakowych, a pożądany rozmiar mapy lokalnej jest znany podczas ich kompilacji przez program-gospodarza, przed przekazaniem łańcucha znakowego zawierającego finalny kernel do kompilatora wpis --TAG\_LOCAL\_MAP\_SIZE-- jest zamieniany na pożądaną wartość przy pomocy prostej operacji wyszukania i zamiany w łańcuchu znakowym.

Jeżeli program-host ustali, przykładowo, iż pożądana jest lokalna mapa o długości 1024 wpisy, --TAG\_LOCAL\_MAP\_SIZE-- zostanie zamienione na 1024, w końcowym kernelu który zostanie skompilowany skutkując zapisem:

\_\_local int localMap[1024];

Co jest w pełni poprawnym żądaniem alokacji pamięci lokalnej.

Zadanie ustawienia początkowych wartości w tejże lokalnej mapie zostało – na potrzeby każdej grupy roboczej – scedowane na pierwsze z jej zadań, tj. to o lokalnym numerze identyfikacyjnym zero. Po kodzie wykonującym tą operację została umieszczona lokalna bariera, dzięki czemu żadne z zadań w grupie roboczej nie rozpocznie wykonania dalszego kodu – uzależnionego od początkowego stanu lokalnej mapy zgodnego z oczekiwaniami – zanim mapa nie zostanie wypełniona wpisami -1.

Dalsza część kernela przebiega zgodnie z proponowanym powyżej algorytmem: każde zadanie wykonuje eliminację aż osiągnie stan w którym posiada unikalny – tym razem w zakresie grupy, a nie w zakresie całego problemu – pierwszy wyraz niezerowy. Po zakończeniu tej części kernela umieszczona jest kolejna bariera lokalna. Gdy zostanie minięta, wątek o lokalnym numerze identyfikacyjnym zero kopiuje zawartość lokalnej mapy do mapy globalnej.

Mapę globalną można przedstawić jako macierz o wymiarach N x M, gdzie N to ilość uruchomionych grup roboczych OpenCL, zaś M to długość macierzy. Każda lokalna mapa jest kopiowana 1:1 do kolumny w tej macierzy o indeksie odpowiadającym numerowi grupy roboczej.

### UNIKALNOŚĆ LOKALNA A GLOBALNA

Ze względu na wykorzystanie pamięci lokalnej (i lokalnych kopii mapy), wykonanie wyżej wspomnianego kernela zapewnia spełnienie warunku unikalności tylko wewnątrz pojedynczej grupy roboczej. Jako że zazwyczaj (dla większej ilości grup roboczych niż jedna) istnieje więcej niż jedna kopia mapy, po skopiowaniu map lokalnych do macierzy przechowującej globalny zestaw map, konieczne jest przywrócenie warunku unikalności w zakresie globalnym. Do tego celu został stworzony drugi kernel, rozwiązujący problem konfliktujących ze sobą (posiadających identyczny pierwszy wyraz niezerowy) wierszy w macierzy.

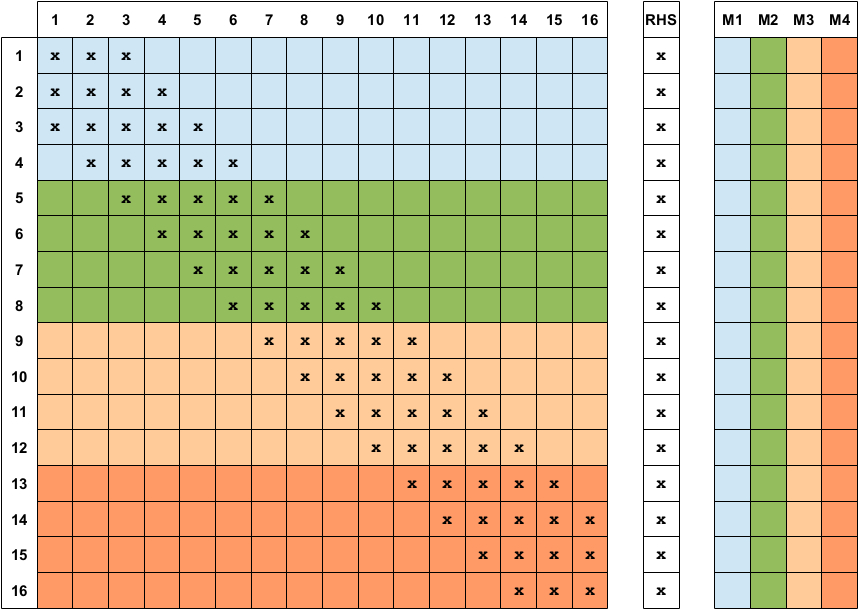
Jego działanie jest mniej skomplikowane niż kernela przywracającego warunek unikalności wewnątrz mapy lokalnej. Ten kernel również uruchamiany jest na jednowymiarowym NDRange o długości co najmniej takiej, ile wynosi wymiar N macierzy zawierającej układ równań. Numer identyfikacyjny zadania ponownie jest traktowany jako numer wiersza, lecz tym razem w macierzy przechowującej mapę globalną.

Ponieważ mapy lokalne wpisane są do mapy globalnej jako kolumny, każde zadanie przegląda przypisany mu wiersz w mapie globalnej. Pierwszy znaleziony wpis różny od -1, oznaczający iż istnieje w macierzy wiersz o takim pierwszym wyrazie niezerowym, traktowane jest jako kanoniczne, tj. ten wiersz jest wykorzystywany do redukcji ewentualnych innych wierszy które posiadają ten sam pierwszy wyraz niezerowy. Jeżeli podczas dalszego przeglądania wiersza w mapie zostaną znalezione numery innych wierszy które mają ten sam pierwszy wyraz niezerowy, na tych dalszych wierszach wykonywana jest eliminacja względem pierwszego znalezionego wiersza.

Ten kernel przywraca jedynie warunek unikalności globalnej nie zwracając z kolei uwagi na unikalność lokalną (wewnątrz bloków, tj. grup roboczych). Uruchomienie pierwszego, a potem drugiego kernela (w sposób blokujący, czyli niedopuszczający uruchomienia ich symultanicznie) jest traktowany jako jeden cykl rozwiązania. Każda wykonana eliminacja w skali globalnej jest zapisywana do odpowiedniej zmiennej jako „operacja”; tylko ta dana jest pobierana do hosta po każdym cyklu rozwiązania, by zaoszczędzić na ilości danych koniecznych do przesłania przez złącze PCI-Express. Jeżeli ilość operacji globalnego przywracania jest niezerowa, jest to informacja iż konieczne jest przeprowadzenie kolejnego cyklu rozwiązania, czyli ponowne uruchomienie obu kerneli.

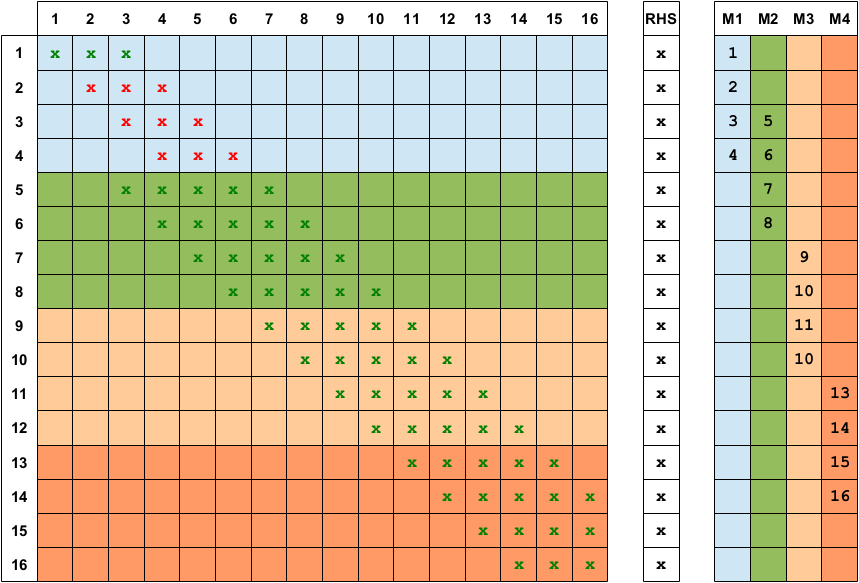
### PRZYKŁAD DZIAŁANIA PROPONOWANEGO ALGORYTMU

Na rysunku 17 została przedstawiona przykładowa macierz o wymiarze N równym 16. Na potrzeby tego przykładu założono, iż zadania rozwiązujące tą macierz zostały podzielone na cztery bloki. Obok macierzy i wektora prawej strony przedstawione zostały cztery mapy lokalne oznaczone M1,…,M4, ułożone tak, jak zostaną wpisane do mapy globalnej; kolor symbolizuje przypisanie map lokalnych do grup roboczych.



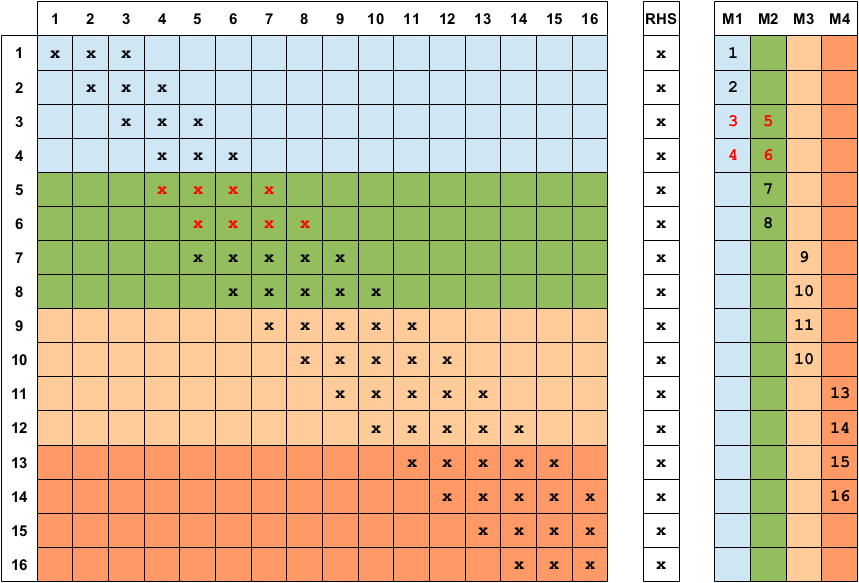
Rysunek 17. Przykładowa macierz do rozwiązania proponowanym algorytmem

Na rysunku 18 został przedstawiony wynik działania pierwszego z opisywanych kerneli, przywracającego warunek unikalności lokalnej. Kolorem zielonym oznaczono wiersze których ID zostały umieszczone w mapach lokalnych bez konieczności przeprowadzenia żadnych dodatkowych eliminacji. Kolorem czerwonym oznaczono wiersze, które uległy eliminacji.



Rysunek 18. Efekt wykonania pierwszego kernela w pierwszym cyklu rozwiązania układu równań liniowych proponowanym algorytmem

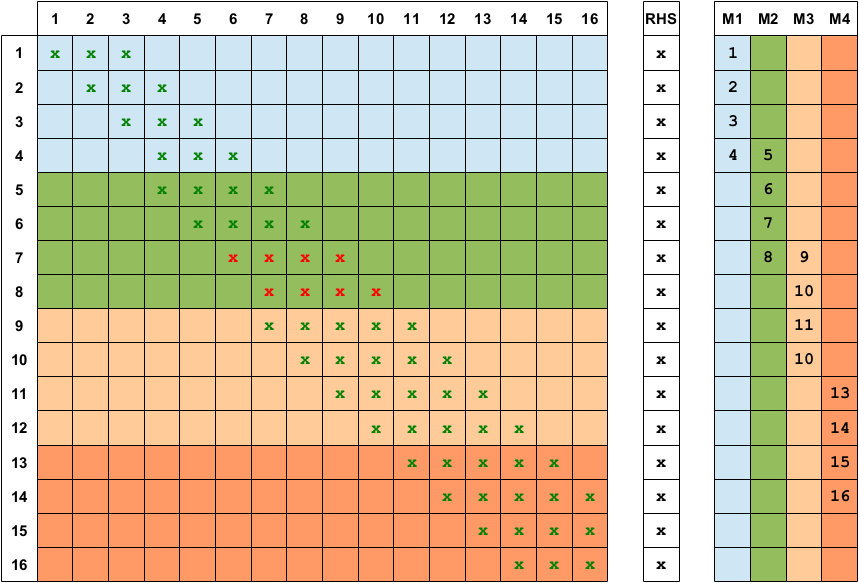
Jak można łatwo zauważyć, warunek unikalności pierwszego wyrazu niezerowego został przywrócony wewnątrz grup roboczych, lecz nie jest spełniony w skali globalnej. Na rysunku 19 zostało przedstawione działanie drugiego kernela, który przywraca jego spełnienie globalnie. Kolorem czerwonym w macierzy zostały zaznaczone wiersze które uległy eliminacji. Kolorem czerwonym w mapach zostały zaznaczone pozycje w których zostały znalezione rozwiązane właśnie konflikty.



Rysunek 19. Efekt wykonania drugiego kernela w pierwszym cyklu rozwiązania układu równań liniowych proponowanym algorytmem

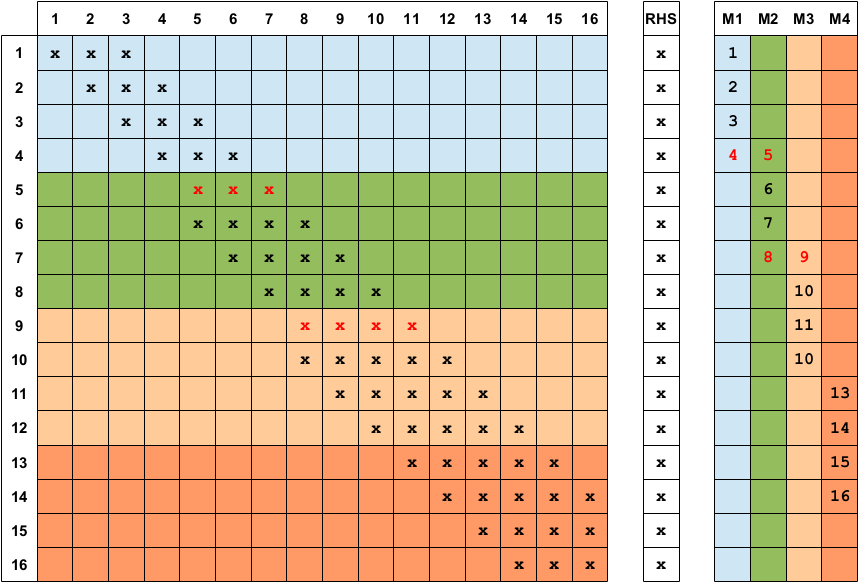
W wyniku działania drugiego kernela zostanie zarejestrowana informacja iż wykonano dwie operacje. Dana ta zostanie pobrana z powrotem z urządzenia OpenCL na hosta, w związku z czym zostanie podjęta decyzja o wykonaniu kolejnego cyklu rozwiązania.

Efekt wykonania pierwszego kernela w drugim cyklu rozwiązania został przedstawiony na rysunku 20. Ponownie kolorem zielony zostały zaznaczone wiersze dla których nie zostały wykonane żadne operacje, zaś kolorem czerwonym wiersze dla których zostały przeprowadzone eliminacje by przywrócić warunek unikalności pierwszych wyrazów niezerowych w obrębie grupy roboczej.



Rysunek 20. Efekt wykonania pierwszego kernela w drugim cyklu rozwiązania układu równań liniowych proponowanym algorytmem

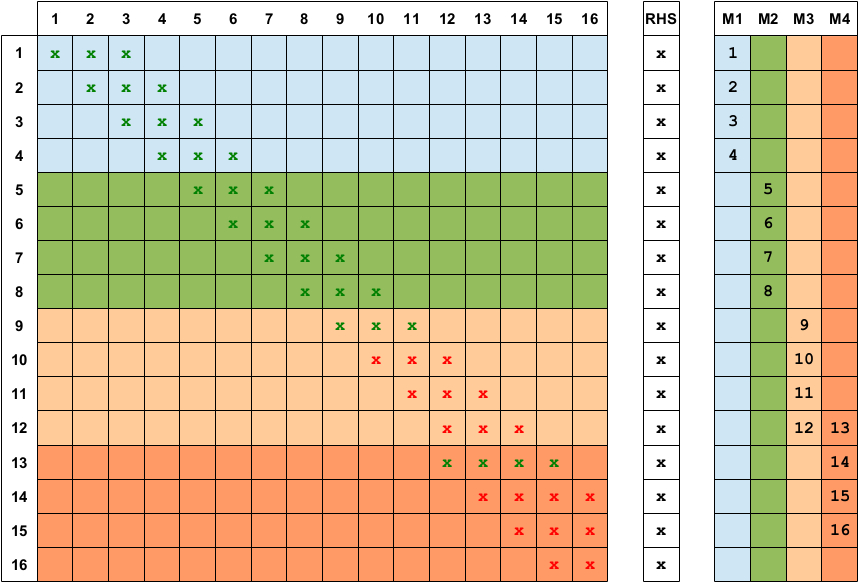
W wyniku wykonania pierwszego kernela powstały nowe globalne konflikty unikalności. Odpowiadające im wiersze w mapie globalnej zostały zaznaczone kolorem czerwonym na rysunku 21.



Rysunek 21. Efekt wykonania drugiego kernela w drugim cyklu rozwiązania układu równań liniowych proponowanym algorytmem

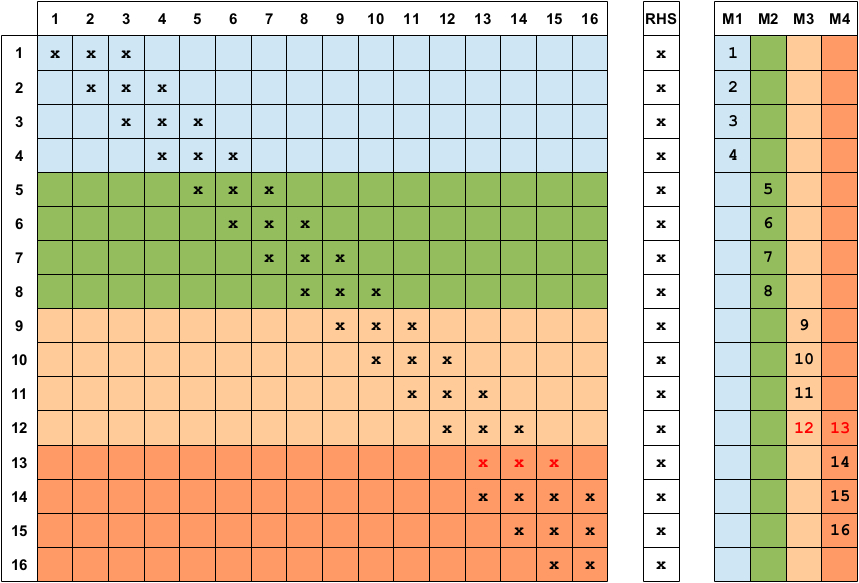
Drugi kernel w drugim cyklu rozwiązania ponownie wykonał dwie operacje eliminacji konfliktów globalnych. Ta informacja zostanie ściągnięta z urządzenia obliczeniowego do programu gospodarza, i w związku z nią zostanie znowu podjęta decyzja o przeprowadzeniu kolejnego cyklu rozwiązania.

Dla zwiększenia czytelności przykładu grafiki reprezentujące dwa następne cykle rozwiązania zostaną pominięte. Zamiast tego przedstawiony zostanie cykl prowadzący do uzyskania finalnego rozwiązania układu równań. Na rysunku 22 przedstawione zostało wykonanie pierwszego kernela w przedostatnim cyklu prowadzącym do rozwiązania układu równań.



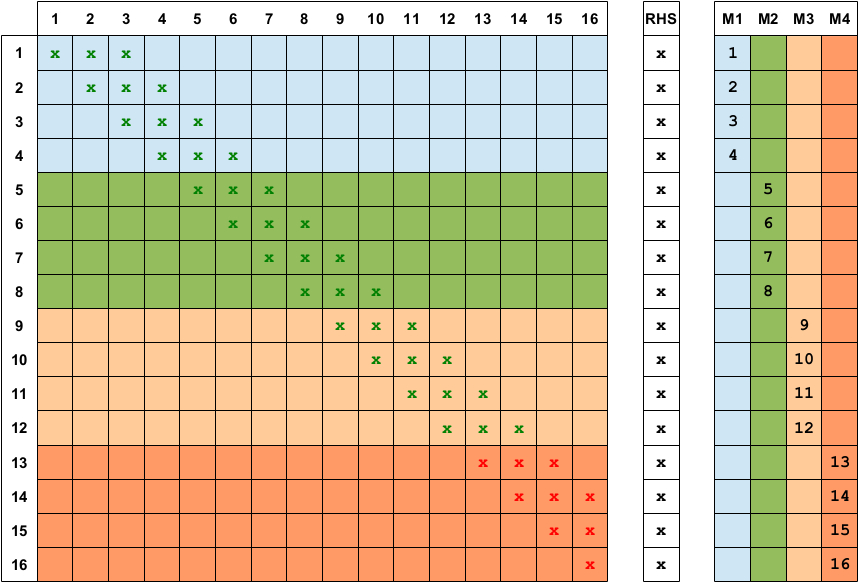
Rysunek 22. Efekt wykonania pierwszego kernela w przedostatnim cyklu rozwiązania układu równań liniowych proponowanym algorytmem

Wygenerowany został już tylko jeden konflikt globalny. Na rysunku 23 zostało uwidocznione jego rozwiązanie przez drugi kernel.



Rysunek 23. Efekt wykonania pierwszego kernela w przedostatnim cyklu rozwiązania układu równań liniowych proponowanym algorytmem

Z racji tego, iż została wykonana operacja eliminacji globalnej, zostanie przeprowadzony jeszcze jeden cykl rozwiązania. Jak jednakże widać na rysunku 24, tym razem kernel pierwszy – przywracający warunek unikalności – nie wytworzy już żadnych konfliktów globalnych, w związku z czym drugi kernel zwróci do gospodarza informację o niewykonaniu żadnych operacji.



Rysunek 24. Efekt wykonania pierwszego kernela w finalnym cyklu rozwiązania układu równań liniowych proponowanym algorytmem

Po otrzymaniu tej informacji oprogramowanie gospodarza zaprzestanie wykonywania kerneli. W następnym kroku rozwiązania macierz schodkowa, wektor prawej strony i ostateczna zawartość mapy globalnej zostanie pobrana z urządzenia obliczeniowego. Macierz zawierająca mapę globalną zostanie sprowadzona do wektora zgodnego z ideą zaprezentowaną w rozdziale 3.1. Następnie na uzyskanych strukturach danych – sekwencyjnie, na CPU – zostanie przeprowadzona operacja podstawienia wstecz, co zaowocuje finalnym wynikiem.

Warto zauważyć tutaj, iż do minimum zostały ograniczone transfery danych po złączu PCI-Express. Duże struktury danych, czyli układ równań, wektor prawej strony i pamięć mapy globalnej zostaje przesłana raz do urządzenia – na początku pętli cykli rozwiązania – i raz z urządzenia do gospodarza. W każdym cyklu rozwiązania z urządzenia transferowana jest zbywalnie mała ilość danych, gdyż informacja o ilości przeprowadzonych operacji eliminacji globalnej jest zapisywana w pojedynczej zmiennej typu unsigned int.

## WYKORZYSTANE STRUKTURY DANYCH

# BADANIA WYDAJNOŚCI

Work in progress…

# WNIOSKI

Work in progress…

# BIBLIOGRAFIA

[1] Bathe, K. J.: Finite element procedures in engineering analysis. *Englewood Cliffs: Prentice-Hall*. 1982.

[2] Bernstein, A.J.: Analysis of Programs for Parallel Processing. *IEEE Transactions on Electronic Computers.* 1966 Vol. EC-15, no. 5, pp. 757-763

[3] Butrylo, B. [et al.]: A Survey of Parallel Solvers for the Finite Element Method in Computational Electromagnetics. *International Journal for Computation and Mathematics in Electrical and Electronic Engineering.* 2004 Vol. 23, no. 2, pp. 531-546.

[4] Cook, S.: CUDA Programming. A Developer’s Guide to Parallel Computing with GPUs. *Waltham: Elsevier.* 2013.

[5] Irons, B.: A Frontal Solution Program For Infinite Element Analysis. *International Journal for Numerical Methods in Engineering.* 1970 Vol. 2, pp. 5-32

[6] Introduction to OpenCL Programming – Training Guide. *AMD*. 2010

[7] Jamil, N.: A Comparison of Direct and Indirect Solvers for Linear Systems of Equations. *International Journal of Emerging Sciences.* 2012 Vol. 2, no. 2, pp. 310-321

[8] Milenin, A.: Podstawy MES. Zagadnienia termomechaniczne. *AGH*. 2010

[9] Rońda J., Oliver G.J., Introduction to numerical methods with Matlab procedures. *AGH*. 2010

[10] Tewarson, R.P.: Sparse matrices. *New York: Academic Press, Inc.* 1973

[11] ICT 1900 Series Central Processors 1904, 1905, *ICT Press release* (ICT), 1964 p. 4. [dostęp: 2013-11-02], Dostępny w Internecie: <<http://bitsavers.trailing-edge.com/pdf/ict_icl/1900/brochures/1904_Central_Processor_Sep64.pdf>>

[12] Conformant Products [online]. *Khronos Group*. [dostęp: 2013-12-11], Dostępny w Internecie: <<http://www.khronos.org/conformance/adopters/conformant-products#opencl>>

[13] The OpenCL Specification [online]. *Khronos Group.* [dostęp: 2013-12-15], Dostępny w Internecie: <<http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf>>

1. Rysunek został zaczerpnięty z <http://icis.pcz.czest.pl/~roman/mat_dyd/prz_rown/mac_rzadkie/4_2.html> [dostęp 23-12-2013]. [↑](#footnote-ref-1)
2. Rysunek został zaczerpnięty z <http://icis.pcz.czest.pl/~roman/mat_dyd/prz_rown/mac_rzadkie/4_2.html> [dostęp 23-12-2013]. [↑](#footnote-ref-2)
3. Opis za <https://www.icm.edu.pl/kdm/Numeryka:_Macierzy#Compressed_Sparse_Row_Format> [↑](#footnote-ref-3)
4. <http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml/atomic_cmpxchg.html> [↑](#footnote-ref-4)