POLITECHNIKA OPOLSKA

WYDZIAŁ

ELEKTROTECHNIKI AUTOMATYKI I INFORMATYKI

MGR INŻ. WALDEMAR POKUTA

RÓWNOLEGŁE APLIKACJE MES W SYSTEMACH KLASTROWYCH DO ROZWIĄZYWANIA ZAGADNIEŃ ELEKTROSTATYKI

PRACA DOKTORSKA

PROMOTOR: DR HAB. INŻ. JAN SADECKI

PROF. POLITECHNIKI OPOLSKIEJ

OPOLE 2007

Prof. Janowi Sadeckiemu pragnę złożyć gorące podziękowania za poświęcony czas, wszystkie cenne uwagi oraz cały wkład wniesiony w realizację niniejszej pracy.

Spis treści

1 Wprowadzenie

Pojęcie pole używane jest w fizyce do określenia obszaru, w którym występuje zależna od miejsca i czasu pewna wielkość fizyczna, np.: siła, potencjał elektryczny, temperatura, prędkość. Problemy polowe występują wszędzie tam, gdzie przestrzeń (płaszczyzna) składa się z mniejszych elementów o zróżnicowanych kształtach i własnościach fizycznych, mających wpływ na inną wartość fizyczną, której nie znamy. Zagadnienia takie pojawiają się zarówno w życiu codziennym, jak też w złożonych zadaniach naukowo-technicznych. Obliczenie dowolnego zadania fizyki (matematycznego lub technicznego) na ogół prowadzi do rozwiązywania układów algebraicznych równań liniowych o takiej czy innej strukturze. Z tego powodu w analizie numerycznej najwięcej uwagi poświęca się metodom redukcji różnorodnych zadań do układów algebraicznych równań liniowych i systematycznemu ich rozwiązywaniu [Mar].

Do znaczących osiągnięć w zakresie metodologii rozwiązywania tego typu problemów można zaliczyć m. in. prace Harold'a C. Martin'a [Har]. Rozwój stosowanych w lotnictwie technologii związanych z wprowadzaniem nowych materiałów konstrukcyjnych, osiąganiem przez samoloty coraz większych prędkości wymusiły na konstruktorach konieczność zmiany metod stosowanych do obliczania naprężeń powstających w strukturze materiałów konstrukcyjnych, czy też projektowania nowych, bardziej dostosowanych do zmieniających się warunków kształtów skrzydeł. W latach pięćdziesiątych minionego stulecia, H.C. Martin, prowadząc badania na uniwersytecie w Princeton we współpracy z dwoma koncernami lotniczymi opracował nową metodę rozwiązywania problemów polowych, później nazwaną *metodą elementów skończonych,* wnosząc tym samym. znaczący wkład do rozwoju nauki w omawianej dziedzinie.

W chwili obecnej złożone obliczenia wielkiej skali realizowane są przy wykorzystaniu superkomputerów. Przykładowo, zbudowany przez firmę NEC komputer równoległy znany pod nazwą *Earth Simulator* (zlokalizowany w pobliżu Jokohamy w Japonii) może wykonywać $35{,}6*10^{12}$ operacji zmiennopozycyjnych w ciągu jednej sekundy (TFLOPS). Składa się on z 5120 procesorów skonfigurowanych w 256 ośmioprocesorowych węzłów połączonych przełącznicą krzyżową. Jest on używany między innymi do tworzenia wirtualnego modelu ziemi [EaS] służącego do przewidywania różnorodnych zjawisk atmosferycznych oraz śledzenia zmian klimatycznych. Jednym z najszybszych na świecie

komputerów jest aktualnie[1] superkomputer BGW (Blue Gene Watson), zainstalowany w Yorktown Heights. Na dwukrotnie aktualizowanej w ciągu roku liście pięciuset najszybszych komputerów na świecie (Top500) znajduje się on na trzeciej pozycji, osiągając w benchmarku linpack moc obliczeniową 91,29 TFLOPS. Komputer ten wykorzystywany jest głównie do symulacji syntezy protein i innych procesów biologicznych. Pierwszym superkomputerem, którego moc obliczeniowa przekroczyła wartość $100*10^{12}$ operacji zmiennopozycyjnych na sekundę był komputer BlueGene/L. W czerwcu 2006 osiągnął on 207,3 TFLOPS, Komputer ten wykorzystywano m. in. do symulacji procesu zachowania się molibdenu pod wysokim ciśnieniem. Inne uruchamiane na tym komputerze aplikacje umożliwiają symulację wybuchów jądrowych bez konieczności przeprowadzania ich w rzeczywistości. Aktualnie planowane jest rozwijanie serii BlueGene. Jego kolejnymi implementacjami mają być: Cyclops64, BlueGene/P, BlueGene/Q. Ostatni z tych komputerów ma osiągnąć 3 PETAFLOPS. Do symulacji wybuchów jądrowych wykorzystywany jest również ASC Purple znajdujący się obecnie na czwartej pozycji najszybszych komputerów świata. Zainstalowano na nim między innymi oprogramowanie Linux SUSE oraz MPI (implementacja firmy IBM).

W Polsce na uwagę zasługuje projekt Clusterix [Clu, ClW]. Zakłada on budowę Krajowego rozproszonego klastra linuksowego, utworzonego z lokalnych klastrów PC o architekturze 64- i 32-bitowej, zlokalizowanych w wielu odległych geograficznie niezależnych ośrodkach. Wydajność tego klastra ocenia się obecnie na 4,4 TFLOPS. Clusterix wykorzystywany będzie m. in. do badań naukowych w zakresie: modelowania zjawisk termomechanicznych, złożonych symulacji (np. przepływu krwi), przewidywania struktur białek czy projektowania układów elektroniki molekularnej itd. Moc, jaką posiadał klaster w chwili uruchomienia, uplasowałaby go na około 130 miejscu najszybszych komputerów na świecie. Projekt łączy dwanaście polskich ośrodków akademickich, oto one: Politechnika Białostocka, Politechnika Częstochowska (koordynator), Centrum Informatyczne Trójmiejskiej Akademickiej Sieci Komputerowej TASK, Politechnika Łódzka, Uniwersytet Marii Curie-Skłodowskiej w Lublinie, Akademickie Centrum Komputerowe CYFRONET AGH, Uniwersytet Opolski, Poznańskie Centrum Superkomputerowo-Sieciowe, Politechnika Szczecińska, Politechnika Warszawska, Wrocławskie Centrum Sieciowo-Superkomputerowe, Politechnika Wrocławska, Uniwersytet Zielonogórski.

Publikacje opisujące wydajność równoległych aplikacji MES to często prace japońskich naukowców. K.Garatani, H. Okuda i G. Yagawa badają wpływ dekompozycji macierzy na efektywność aplikacji [GOY], a publikacje K. Nakajima [Na1, Na2, Na3, Na4, Na5] opisują głównie wpływ architektury komputera na wydajność algorytmu. Badane są przez niego

[1] lista Top500 z czerwca 2006

głównie superkomputery, przede wszystkim Earth Simulator. Równoległe implementacje MES rozważane są również w pracy A. Kaceniauskas, P. Rutschmann [KaR]. Na tym tle bardzo ciekawie prezentują się również opracowania polskich autorów (T. Olas, R. Wyrzykowski, A. Tomaś, K. Karczewski) [OKT, OLK, OWT]; wyprowadzają oni model matematyczny obliczeń MES dla aplikacji równoległych. Bazując na powyższym modelu autor niniejszej rozprawy zamierza wyprowadzić model, którego parametry będą uwzględniały również pewne niedoskonałości klastrów o węzłach wieloprocesorowych. Czas obliczeń MES otrzymany z modelu będzie mógł być dzięki temu bardziej zbliżony do czasów otrzymanych doświadczalnie.

Równoległy sposób realizacji aplikacji, polegający na wykorzystaniu do jej implementacji nie jednego, lecz wielu niezależnych jednostek procesorowych, prowadzi z reguły do skrócenia czasu jej wykonywana. Wielkość uzyskiwanego w tym zakresie przyspieszenia zależy jednak od bardzo wielu czynników związanych zarówno z oprogramowaniem jak i sprzętem systemów równoległych [Fos]. Jednym z czynników negatywnie wpływających na ostateczną efektywność realizacji aplikacji równoległych są tzw. wymagania komunikacyjne, polegające na konieczności zapewnienia wymiany informacji (komunikatów) pomiędzy realizowanymi równolegle jej fragmentami. Wymagania te mogą być w pewnych przypadkach tak wysokie, że rozpraszanie zadań może okazać się nieopłacalne. Ponadto, w zależności od rodzaju aplikacji, sam proces jej zrównoleglenia może okazać się zadaniem bardzo złożonym, szczególnie w przypadku, gdy dąży się do zbudowania efektywnej aplikacji, wykorzystującej w sposób możliwie optymalny możliwości sprzętu konkretnego systemu równoległego. W zasobnych finansowo ośrodkach problemy te rozwiązuje się przy wykorzystaniu dużych (nie rozproszonych geograficznie) systemów równoległych, składających się niejednokrotnie z bardzo wielu elementów procesorowych. Clusterix jest przykładem jak można ominąć ograniczenia finansowe poprzez integrację istniejących systemów. Pociąga to jednak za sobą zwiększenie narzutów komunikacyjnych na wykonujące się równolegle aplikacje. Architektura rozproszona powoduje, że należy rozważnie projektować same aplikacje, gdzie optymalizacja wymiany danych jest zadaniem priorytetowym. Potrzebne jest zbudowanie modelu aplikacji wykonującej się równolegle, gdyż jej nieprzemyślane wdrożenie może pociągnąć za sobą niepotrzebne koszty.

1.1 Trendy w technologii komputerowej

Gdy komputery stawały się coraz szybsze, przypuszczano, że w końcu ich moc obliczeniowa będzie już wystarczająca, aby wykonać dowolną znaną aplikację. W latach 1940-tych rząd brytyjski stwierdził, że zapotrzebowania Wielkiej Brytanii to dwa, może trzy

komputery [Fos]. W praktyce jednak nowe aplikacje wymagały wprowadzania coraz większych mocy obliczeniowych. Od 1945 roku możliwości najszybszych komputerów wzrastają dziesięciokrotnie co pięć lat. Ta tendencja utrzymuje się również dla komputerów osobistych. Możliwości procesorów również wzrastają wykładniczo. Tendencja ta jednak ma swoje ograniczenia, takie jak prędkość światła [Fos]. Aby informacja na wejściach układu scalonego mogła zostać przetworzona i pojawiła się na wyjściach, musi przebyć określoną drogę wyznaczoną przez przekątną układu scalonego. Czas ten ogranicza możliwość zwiększania częstotliwości taktowania układu a w rezultacie wyznacza górną granicę możliwości procesorów wykonujących zadania sekwencyjnie. Projektanci komputerów używają różnych technik, aby obejść te ograniczenia. Stosuje się przetwarzanie potokowe, wiele jednostek arytmetyczno-logicznych, tworzone są komputery oparte na wielu procesorach i wreszcie wykorzystuje się wiele osobnych komputerów posiadających własne procesory i pamięć, połączonych ze sobą typową bądź dedykowaną siecią połączeń.

Trendy te powodują, że zmienia się też sposób projektowania aplikacji. W przyszłości nie tylko aplikacje uruchamiane na superkomputerach, ale również na zwykłych stacjach roboczych będą musiały być projektowane tak, aby wykorzystać równoległy potencjał tych komputerów. Ponieważ zdecydowana większość algorytmów tworzona jest dla pojedynczych procesorów, istnieje potrzeba ponownego ich zaprojektowania tak, aby były one efektywne dla większej liczby procesorów. Programy muszą być skalowalne, tzn. uruchamialne na dowolnej liczbie procesorów (komputerów).

Foster [Fos] omawiając zagadnienia związane z przekształcaniem oprogramowania z realizacji sekwencyjnej na równoległą stwierdza, że dobrze zaprojektowany program równoległy powinien mieć takie cechy jak:

- Równoległość - czyli możliwość wykonywania wielu akcji jednocześnie; jest to podstawowa właściwość, jeżeli program ma być wykonywany na wielu procesorach,
- Skalowalność - oznacza, że można zwiększać lub zmniejszać liczbę procesorów, na których program będzie wykonywany,
- Lokalność - oznacza, że odwołania przez procesy do pamięci zdalnej (komunikacja) powinny być niewielkie w stosunku do odwołań lokalnych; jest to klucz do osiągnięcia wysokiej wydajności w architekturze wielokomputerowej,
- Modularność - dekompozycja złożonych jednostek na proste komponenty; jest to cecha ważna zarówno w programowaniu równoległym jak też w sekwencyjnym.

Jak dalej podaje autor, większość problemów programistycznych posiada wiele równoległych rozwiązań. Najlepsze rozwiązanie może różnić się od tego, które sugeruje konstrukcja programu sekwencyjnego. Trudno jest podać ogólną receptę projektowania takiego programu, każdy algorytm wymaga indywidualnego, "kreatywnego" podejścia. Niemniej jednak Foster

podaje cztery etapy budowania takiej aplikacji:

1. Partycjonowanie - czyli rozdzielenie na małe części obliczeń i związanych z nimi danych. Można rozróżnić tu dwa podejścia: najpierw dzielimy na partycje (części) dane programu a następnie danym przyporządkowujemy obliczenie, które powinny się wykonać dla tych danych. Podejście to nazywamy *dekompozycją dziedziny*. Drugim podejściem jest *dekompozycja funkcjonalna*, czyli rozdział czynności a następnie przyporządkowanie im określonych danych. Chociaż są to techniki komplementarne, jednak częściej wykorzystywany jest pierwszy sposób.
2. Komunikacja - zadania wygenerowane przez partycjonowanie z założenia są wykonywane równolegle, ale najczęściej nie niezależnie od siebie. Obliczenia, które muszą zostać wykonane w jednym zadaniu, potrzebują danych skojarzonych z innym zadaniem. Dane muszą zostać przesłane pomiędzy zadaniami tak, aby umożliwić wykonanie tych obliczeń. Ten przepływ danych pomiędzy zadaniami musi zostać zaprojektowany w fazie projektowania komunikacji. Pierwszym etapem projektowania komunikacji jest wyznaczenie kanałów pomiędzy nadawcami i odbiorcami komunikatów, natomiast drugim określenie struktury komunikatów przesyłanych przez te łącza. Rozwiązania wymaga również problem komunikacji globalnej, w której muszą brać udział wszystkie, lub duża liczba procesów; przykładem może być sumowanie wartości znajdujących się w różnych procesach aplikacji. Nieodpowiednie zaprojektowanie tej czynności spowoduje, że dla dużej liczby procesów nastąpi znaczące wstrzymanie obliczeń całej aplikacji.
3. Aglomeracja - w tej fazie przechodzimy od abstrakcyjnej struktury do konkretu. Rewidujemy decyzje podjęte w czasie projektowania partycji i komunikacji tak, aby program mógł być efektywnie uruchomiony na pewnej klasie komputerów równoległych. W szczególności rozważamy możliwość połączenia zadań utworzonych w czasie partycjonowania tak, aby zmniejszyć narzuty komunikacyjne.
4. Mapowanie - to decydowanie, na którym procesorze powinno się wykonać każde z zadań. Problem mapowania nie istnieje w komputerach jednoprocesorowych i ze współdzieloną pamięcią, w których zachodzi automatyczny przydział zadań do procesorów. Zadanie może być różnie rozwiązywane dla różnych algorytmów. Niekiedy zadanie jest trywialne i jest jedynie uzupełnieniem aglomeracji, niekiedy jednak wykorzystuje się takie metody balansowania obciążeń procesorów jak: rekursywna bisekcja, algorytmy lokalne, metody obliczające prawdopodobieństwo, cykliczne mapowanie oraz kierownik/pracownik.

1.2 Cel, teza i zakres pracy

Mimo poprawnego zaprojektowania aplikacji równoległej nie można z całą pewnością

odpowiedzieć czy poprawi się jej wydajność dla danego algorytmu i dla dowolnego sprzętu (klastra), w którym został uruchomiony program. Aby odpowiedzieć na to pytanie autor niniejszej pracy zamierza utworzyć model czasowy równoległych obliczeń MES opartych o metodę CG (gradientu sprzężonego). Umożliwi to realizację głównego celu pracy, jakim jest::

Optymalizacja aplikacji równoległych związanych z rozwiązywaniem zagadnień polowych, polegająca na dostosowaniu tych aplikacji - algorytmów (rozdział zadań obliczeniowych, dystrybucja danych, komunikacja) do struktury oraz możliwości klastrów obliczeniowych.

Jako podzadania można wymienić:

- *Zbudowanie modelu pewnej klasy klastrów w celu optymalizacji obliczeń dla MES.*
- *Zbadanie możliwości przyspieszenia komunikacji w klastrze poprzez wykorzystanie wspólnej pamięci i nakładkowanie (overlapping).*
- *Opracowanie prostej, efektywnej, równoległej aplikacji MES.*

Do zadań symulacji zjawisk opisanych równaniami różniczkowymi cząstkowymi często wykorzystywana jest metoda elementów skończonych (MES). Narzędzia ułatwiające równoległe implementacje tej metody rozwijane są przez ośrodki naukowe na całym świecie [Azt, Blo, Pet, PSp]. Znając parametry klastra oraz model możemy oszacować korzyści płynące z rozbudowy klastra oraz podjąć odpowiednie decyzje wcześniej. Mając to na uwadze teza pracy brzmi następująco:

„Uwzględnienie specyfiki konfiguracji oraz oprogramowania klastrów obliczeniowych stwarza realną podstawę do poprawy efektywności (szybkości) równoległego rozwiązywania złożonych zagadnień polowych przy wykorzystaniu MES, w porównaniu do standardowych implementacji.”

Układ pracy jest następujący. Dalsza część rozdziału pierwszego omawia skrótowo podstawowe metody obliczania rozkładu pól, definiuje pojęcie klastra obliczeniowego i umieszcza go w klasyfikacji systemów komputerowych, wyjaśnia znaczenie macierzy rzadkich podczas obliczeń MES oraz podaje parametry klastra wykorzystywanego w badaniach. W rozdziale drugim zwrócono uwagę na wszystkie istotne, zdaniem autora, aspekty tworzenia efektywnych aplikacji klastrowych: konfiguracja połączeń sieciowych, ilość pamięci operacyjnej, rola systemu operacyjnego i oprogramowania klastrowego uwzględnienie architektury wieloprocesorowej oraz sposób podziału zadania na części. Podano przykład równoległej aplikacji mnożenia macierzy. W rozdziale trzecim dokonano przeglądu metod rozwiązywania układów równań liniowych. Przedstawiono podstawowe metody dokładne z uwzględnieniem technik pasmowych i frontalnych, metody iteracyjne stacjonarne i niestacjonarne oraz dokonano porównania tych metod w rozwiązywaniu układów pełnych oraz rzadkich. Czwarty rozdział dotyczy z kolei problematyki związanej z

implementacją macierzy rzadkich. Omówiono wpływ sposobu zapisu macierzy na efektywność operacji dokonywanych na tych macierzach, opisano czynności wykonywane na macierzy podczas obliczeń MES, przeprowadzono badania nad zwiększeniem efektywności tych operacji poprzez redukcję odwołań do pamięci, wykorzystanie techniki SSE2 oraz optymalizację całych wyrażeń wektorowych. W rozdziale piątym przedstawiono równoległe implementacje metod Bi-CGSTAB i CG oraz metody optymalizacji komunikacji pomiędzy procesami programu. Rozdział szósty podaje dalsze możliwości zwiększenia efektywności komunikacji w klastrze, omówiono rolę rozgłoszeń, przedstawiono algorytm numeracji procesów, ustalający ścieżkę rozprzestrzeniających się wiadomości w klastrze. W rozdziale siódmym przedstawiono realizację równoległej implementacji metody elementów skończonych oraz przykładowe zadanie elektrostatyki i dokonano pomiaru czasu wykonywania obliczeń. W ósmym rozdziale wyprowadzono model równoległych obliczeń MES przy wykorzystaniu metody CG. W modelu uwzględniono czas obliczeń, czas komunikacji oraz możliwość występowania overlapping'u (nakładkowania) obliczeń i komunikacji. Zbudowany model zweryfikowano z danymi doświadczalnymi oraz wykorzystano do zasymulowania czasów obliczeń dla zmienionych parametrów klastra.

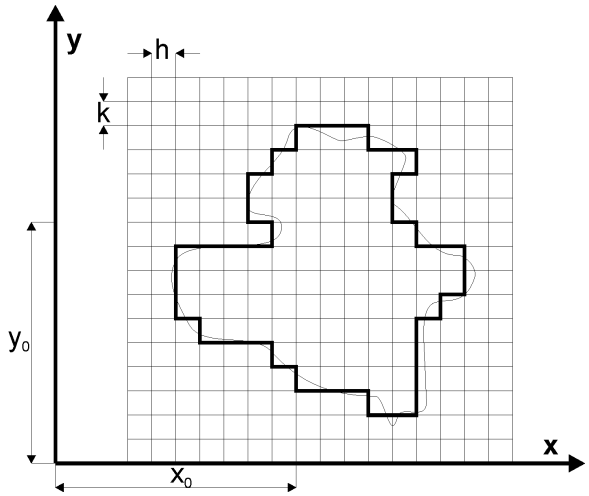
1.3 Numeryczne metody obliczania rozkładu pól

Problemy polowe opisywane są równaniami różniczkowymi, cząstkowymi. Ze względu na często nieregularne kształty elementów, które należy uwzględnić podczas obliczania rozkładu pola, rozwiązanie analityczne problemu może stać się niemożliwe - pozostają wówczas jedynie metody numeryczne, które pozwalają rozwiązać problem z pewną, ograniczoną dokładnością. Dokładność rozwiązania można poprawić stosując zagęszczenie siatki oraz odpowiedni podział obszaru, dopasowany do kształtu elementów. Zmiany takie skutkują często zwiększeniem rozmiaru układu równań algebraicznych, który należy rozwiązać. Pociąga to za sobą z kolei konieczność zwiększenia mocy obliczeniowej jak też wykorzystania wydajniejszych algorytmów uwzględniających zarówno sprzętowe możliwości zaimplementowane w nowoczesnych procesorach jak też optymalizację komunikacji pomiędzy procesami znajdującymi się na tym samym, bądź na różnych komputerach.

1.3.1 Metoda różnic skończonych

W metodzie różnic skończonych [Sik] równanie różniczkowe cząstkowe sprowadzane jest do równania różnicowego. Tworzona jest siatka (na płaszczyźnie (rys. 1.1) lub w przestrzeni), której elementy mogą mieć kształt kwadratowy (sześcienny) lub prostokątny (prostopadłościenny). Tworzony jest układ równań algebraicznych równoważny równaniu

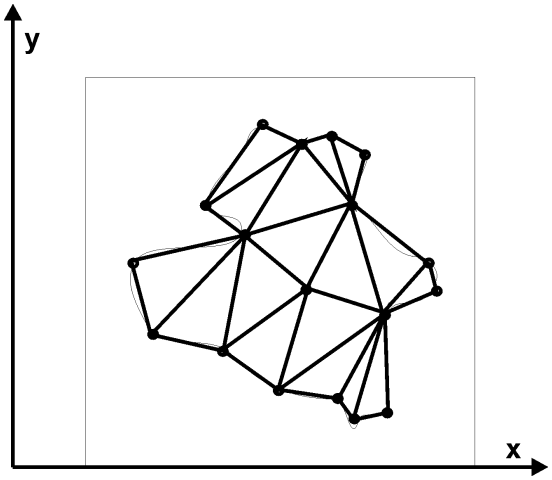
różnicowemu. Macierz tak utworzonego układu jest macierzą rzadką (większość elementów równa zero). Układ ten rozwiązywany jest najczęściej przy wykorzystaniu metod przybliżonych (iteracyjnych).



Rys. 1.1. Podział przestrzeni (płaszczyzny) na elementy w metodzie różnic skończonych.

1.3.2 Metoda elementów skończonych

W metodzie elementów skończonych [Zie] przestrzeń dzielona jest na proste elementy różnego kształtu i wielkości (rys. 1.2). Jest to wyraźną zaletą tej metody w porównaniu do metody różnic skończonych, gdyż można w niej stosować w jednym obszarze podział na elementy o różnej gęstości (nazywane elementami skończonymi). Dalsze postępowanie jest równoważne minimalizacji całkowitej energii potencjalnej układu (funkcjonału). Podobnie jak w metodzie różnic skończonych tworzony jest duży układ równań będący również układem rzadkim. Podobnie też można go rozwiązać metodami iteracyjnymi [OWT].



Rys. 1.2. Podział przestrzeni (płaszczyzny) na elementy w metodzie elementów skończonych.

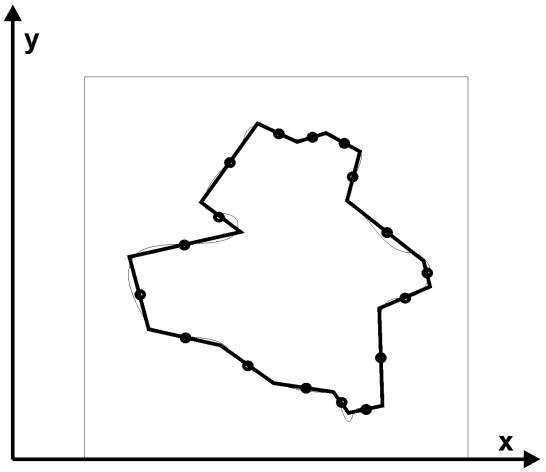
Główne etapy rozwiązania konkretnego problemu przy wykorzystaniu MES to [Zie]:

- Generacja siatki (podział na elementy skończone).
- Matematyczne sformułowanie metody Galerkina lub wariacyjnej.
- Wybór funkcji kształtu (interpolujących rozkład szukanych wartości wewnątrz elementu).
- Wyznaczenie układu równań algebraicznych dla każdego elementu skończonego.
- Składanie układów równań dla poszczególnych elementów w celu utworzenia jednego układu globalnego.
- Uwzględnienie warunków brzegowych w globalnym układzie równań.
- Rozwiązanie układu równań algebraicznych.
- Obliczenie wartości wtórnych i prezentacja wyników

Problemy automatycznego generowania siatek omawiane są m. in. w publikacjach takich autorów jak B. Głut, T. Jurczyk i M. Pietrzyk [GJ1, GJ2, GJ3, GJ4]

1.3.3 Metoda elementów brzegowych

W metodzie elementów brzegowych [BSS] analizowane są jedynie brzegi obszaru (rys. 1.3). Powoduje to zmniejszenie się liczby równań w tworzonym układzie. Układ równań powstający w tej metodzie jest jednak układem gęstym. Układ najczęściej jest rozwiązywany metodami dokładnymi (np. eliminacji Gaussa).



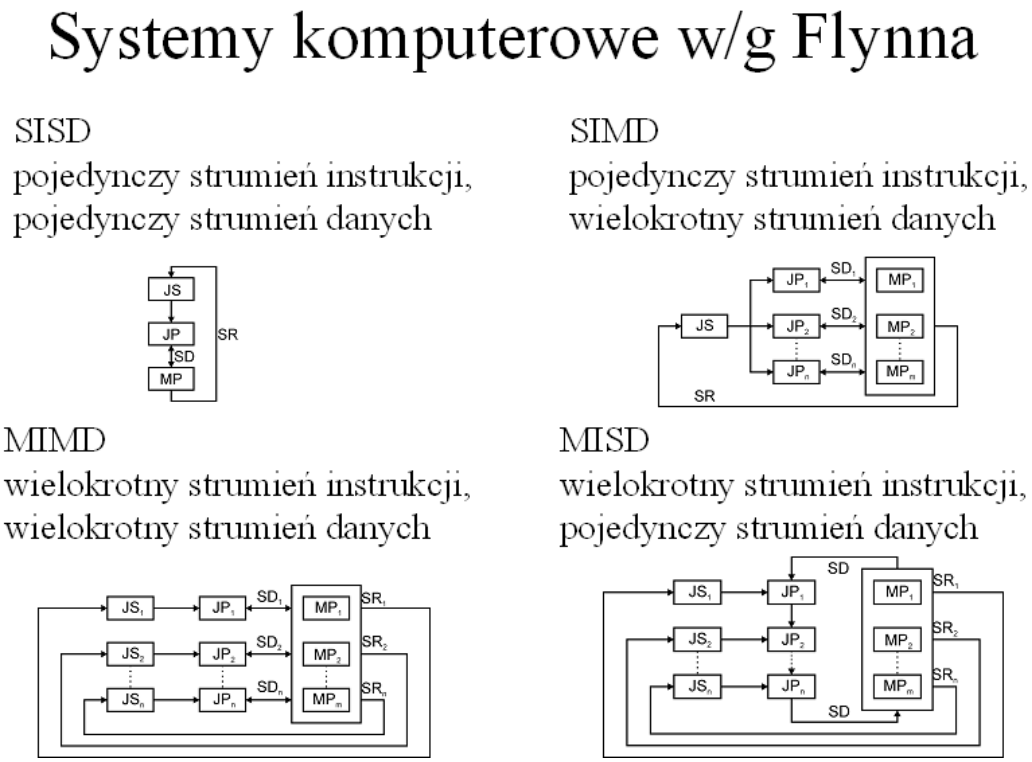
Rys. 1.3. Podział na elementy w metodzie elementów brzegowych.

1.4 Klastry obliczeniowe

1.4.1 Klasyfikacja klastrów

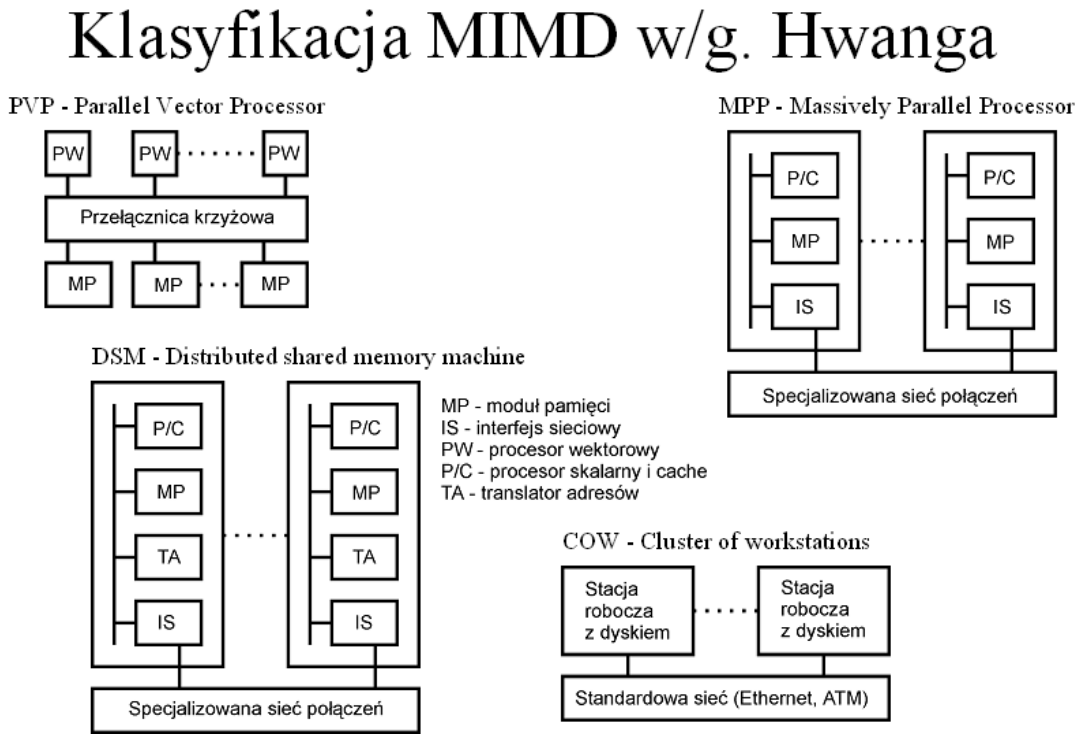
Według klasycznego podziału [Fl2] systemy równoległe możemy w zależności od ilości strumieni danych i liczby strumieni instrukcji podzielić na systemy typy SISD (pojedynczy strumień instrukcji, pojedynczy strumień danych), SIMD (pojedynczy strumień instrukcji,

wielokrotny strumień danych), MISD (wielokrotny strumień instrukcji, pojedynczy strumień danych) oraz MIMD (wielokrotny strumień instrukcji, wielokrotny strumień danych).



Rys. 1.4. Klasyfikacja systemów równoległych według Flynna [KSz].

Zdecydowana większość systemów równoległych to systemy typu MIMD. W tej kategorii zostało wyróżnionych przez Hwanga jeszcze pięć podkategorii (rys. 1.5).



Rys. 1.5. Klasyfikacja systemów MIMD według Hwanga.

Systemy klastrowe możemy zaklasyfikować jako systemy typu MIMD, ponieważ są to systemy wieloprocesorowe mogące jednocześnie wykonywać różne programy z różnymi danymi. W klasyfikacji Hwanga są to oczywiście COW (Cluster of Workstations). Pewne standardy jak MPI (Message Passing Interface) narzucają jednoczesne wykonywanie tych samych programów na wszystkich procesorach, co powoduje zawężenie standardu MIMD ze względu na oprogramowanie jako SPMD (jeden program, wiele strumieni danych). Można tak zaprojektować kod, aby każdy uruchomiony proces mógł wykonywać całkiem różne operacje. Analizując budowę pojedynczego węzła klastra, można dojść do wniosku że sam mikroprocesor może działać na zasadzie SISD, SIMD, MISD lub MIMD. Nowoczesne klastry możemy zatem klasyfikować różnie na różnych poziomach. Ogólnie można je zaklasyfikować jako MIMD, jeżeli jednak pominie się równoległość na poziomie procesora lub pojedynczego węzła, to można zbudować aplikację równoległą o relatywnie mniejszej efektywności.

Systemy MIMD można podzielić na systemy ze wspólną pamięcią i systemy oparte na przesyle komunikatów. Klastry w tej klasyfikacji będą z pewnością oparte na przesyle komunikatów. Jednakże system wieloprocesorowy znajdujący się w obrębie jednego węzła może być oparty o wspólną pamięć, co w rezultacie daje rozwiązanie hybrydowe, stające się aktualnie pewnym standardem w zakresie konfiguracji klastrów.

1.4.2 Co to jest klaster obliczeniowy?

Dokładna definicja klastra nie jest sprecyzowana. Jednakże jest kilka cech, którymi charakteryzują się klastry [CIn].

- Składają się z wielu takich samych lub podobnych maszyn (komputerów), nazywanych węzłami,
- Posiadają w zależności od przeznaczenia typową lub dedykowaną sieć połączeń,
- Wszystkie węzły posiadają wspólne zasoby takie jak katalog domowy,
- Węzły klastra „ufają” sobie nawzajem tak, aby dane pomiędzy nimi nie musiały być szyfrowane ani nie było potrzeby podawania hasła,
- Węzły muszą mieć zainstalowane odpowiednie oprogramowanie klastrowe, np. bibliotekę MPI tak, by programy mogły być uruchamiane wspólnie przez wszystkie węzły.

1.4.3 Budowa prostego klastra

Prosty klaster można zbudować na bazie istniejącej sieci komputerowej Ethernet. Jeżeli istniejące połączenia sieciowe są niskiej przepustowości, warto zainwestować w szybsze

urządzenia, ponieważ może się to okazać kluczową sprawą w osiągnięciu odpowiedniego przyspieszenia wykonywanych aplikacji. Stosowanie typowych sieci jest dosyć tanim rozwiązaniem ze względu na brak lub niewielkie zmiany konfiguracji istniejącego sprzętu.

Drugim krokiem jest skonfigurowanie interfejsów sieciowych i wspólnego katalogu domowego. Ta operacja może się różnić dla różnych systemów operacyjnych. Każdy węzeł klastra powinien mieć nadaną własną nazwę (identyfikator).

Następnie należy tak skonfigurować zabezpieczenia, aby procesy uruchamiane w różnych węzłach mogły się ze sobą komunikować bez podawania hasła i szyfrowania danych.

Ostatnim krokiem jest zainstalowanie oprogramowania umożliwiającego uruchamianie aplikacji komunikujących się poprzez przesył komunikatów np. jedną z implementacji MPI.

1.4.4 Efektywność klastrów

Technologia klastrowa w stosunku do superkomputerów daje znaczne obniżenie kosztów sprzętu z zachowaniem podobnej mocy obliczeniowej. Jedną z poważniejszych niedogodności w systemach klastrowych jest mała szybkość wymiany informacji w klastrze w stosunku do szybkości wykonywanych operacji wykonywanych na jednym procesorze. W skrajnych przypadkach bardziej opłaca się wykonać zadanie w jednym procesie niż dzielić je na kilka podzadań. Poza tym, aby podział zadania był optymalny, powinien być silnie związany z rodzajem algorytmu. Ostatnio zauważyć można coraz wyraźniej technologie mieszane, gdzie każdy z węzłów klastra ma coraz bardziej zaawansowaną strukturę. Wykorzystanie tych nowych możliwości wiąże się z potrzebą zastosowania rozszerzeń do istniejących standardów.

Do zalet klastrów można zaliczyć:

- korzystny stosunek ceny do wydajności,
- odporność na awarie,
- wysoka dostępność,
- możliwość stopniowej rozbudowy (skalowalność),
- wysoka wydajność dla pewnej klasy zadań.

Jako wady klastrów można wymienić:

- niedostosowane oprogramowanie (nie tworzone z myślą o uruchamianiu w klastrach),
- brak możliwości efektywnego rozpraszania zadań na wiele węzłów przez systemy operacyjne,
- powolna komunikacja międzywęzłowa.

1.4.5 Oprogramowanie

O ile sprzęt komputerowy często jest narzucony poprzez istniejącą sieć komputerową, o tyle oprogramowanie zarządzające klastrem można wybrać spośród wielu tanich lub darmowych, dostępnych przez internet wersji. System operacyjny powinien być przede wszystkim wielozadaniowy, powinien wykorzystywać technologie wieloprocesorowe, HyperThread, i in. Wybór niewłaściwego systemu może spowodować, że mimo posiadanego zaawansowanego sprzętu nie wykorzysta się w pełni jego mocy obliczeniowej. Może się też okazać (dla mniej popularnych systemów), że nie będzie dla danego systemu operacyjnego odpowiedniego oprogramowania klastrowego. Należy więc wybrać jeden z popularniejszych systemów (np. Windows, Linux). Oprogramowanie klastrowe rozwijane jest przez różne ośrodki naukowe i dostępne jest często w wersjach darmowych. Jako popularniejsze można wymienić: MPICH, LAM MPI, MP_Lite, PVM. Oprogramowanie klastrowe wykorzystuje dostępne dla danego systemu kompilatory (np. GCC – darmowy, Visual Studio – komercyjny). Mając już oprogramowanie podstawowe można się zastanowić nad instalacją bibliotek obliczeniowych specjalizowanych do wykonywania zadań na macierzach rzadkich. Wśród nich, jako najbardziej kompletne można wymienić takie pakiety jak: P-SPARSLIB [PSp], AZTEC [Azt], PETSc [Pet], BLOCKSOLVE [Blo].

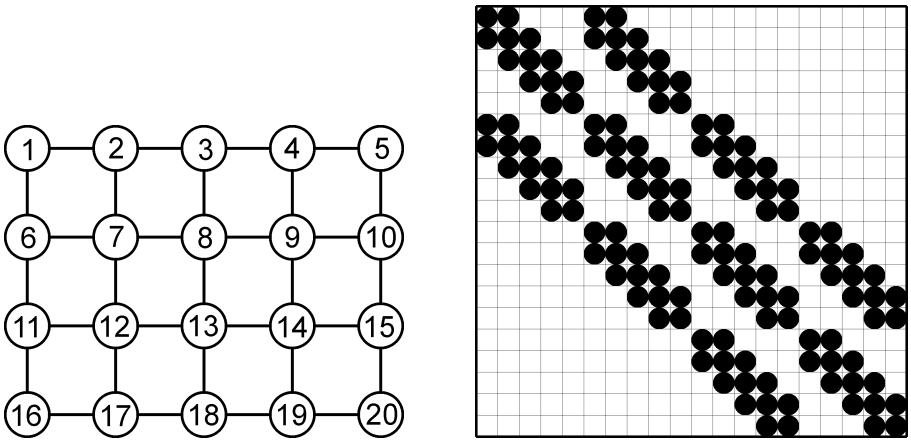
1.5 Macierze rzadkie

Wiele metod rozwiązywania zadań z różnorodnych dziedzin, (genetyka, teoria grafów, socjologia, elektrotechnika, mechanika, itd.) prowadzi w ostateczności do konieczności wykonywania różnego typu operacji na macierzach rzadkich. Macierz rzadka to taka, której większość elementów ma tą samą wartość (w dalszej części rozważana będzie tylko taka sytuacja, gdy wartość ta będzie zerem). Macierze takie można zapisywać w pamięci komputera w podobny sposób jak macierze pełne, jednak jest to często nieopłacalne z co najmniej dwóch powodów:

Zajmują one wiele więcej pamięci operacyjnej niż jest to w rzeczywistości konieczne. W rzeczywistych zadaniach często liczba elementów niezerowych rośnie liniowo wraz ze wzrostem rozmiaru zadania (im większa macierz tym procentowo mniej elementów istotnych), pełny zapis macierzy wymaga jednak zapamiętania N^2 współczynników. W wyniku tego większe zadania mają zapotrzebowanie na ogromne ilości pamięci.

Pełny zapis macierzy wymaga uwzględnienia podczas operacji arytmetycznych działań na wszystkich elementach macierzy (również zerowych). Powoduje to zazwyczaj zwiększenie złożoności obliczeniowej zadań w stosunku do analogicznych zadań wykorzystujących skrócony opis macierzy. Przykładową strukturę macierzy rzadkiej wygenerowanej na

przykład w trakcie obliczeń MES przedstawia rys. 1.6.



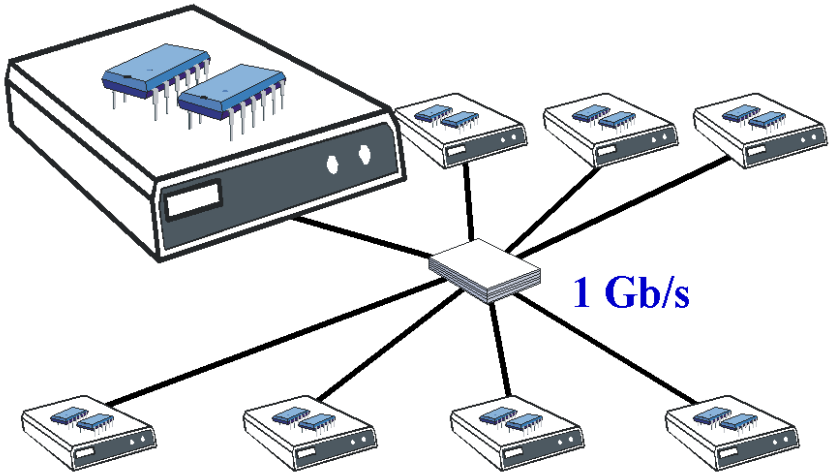
Rys. 1.6. Struktura macierzy rzadkiej generowana przez program MES z siatki o dwudziestu węzłach.

1.6 Standardy MPI i PVM

MPI i PVM umożliwiają tworzenie aplikacji klastrowych wykorzystujących przesył komunikatów do przekazywania informacji pomiędzy procesami. Aplikacje można tworzyć w językach C, C++ i Fortran. Każdy z tych standardów udostępnia zestaw funkcji umożliwiających uzyskanie pewnych informacji o klastrze i aktualnym procesie, przesyłanie komunikatów, realizację rozgłoszeń oraz synchronizację procesów. Każdy proces posiada swój identyfikator zadania i może być przyporządkowany do pewnych grup procesów.

Jak widać założenia standardów są podobne, więc porównywanie ich funkcjonalności musiałoby się odbyć poprzez porównanie wszystkich istniejących implementacji tych standardów.

1.7 Klaster wykorzystywany do obliczeń



Rys. 1.7. Struktura klastra badawczego Politechniki Opolskiej.

Większość pomiarów i badań wykonywanych w trakcie realizacji niniejszej pracy (gdy tak nie było - zaznaczono to w pracy) przeprowadzono przy użyciu klastra badawczego Politechniki Opolskiej. Parametry klastra są następujące (rys. 1.7):

- Klaster składa się z ośmiu węzłów,
- Każdy z węzłów posiada dwa procesory Intel Pentium IV Xeon, 2.8 GHz (technologia HyperThread),
- Parametry węzłów: 2 GB RAM, 80 GB HDD, 2 x Intel 1 Gigabit Ethernet,
- Wyposażenie dodatkowe: FDD, CDROM, USB, RAID, IDE.

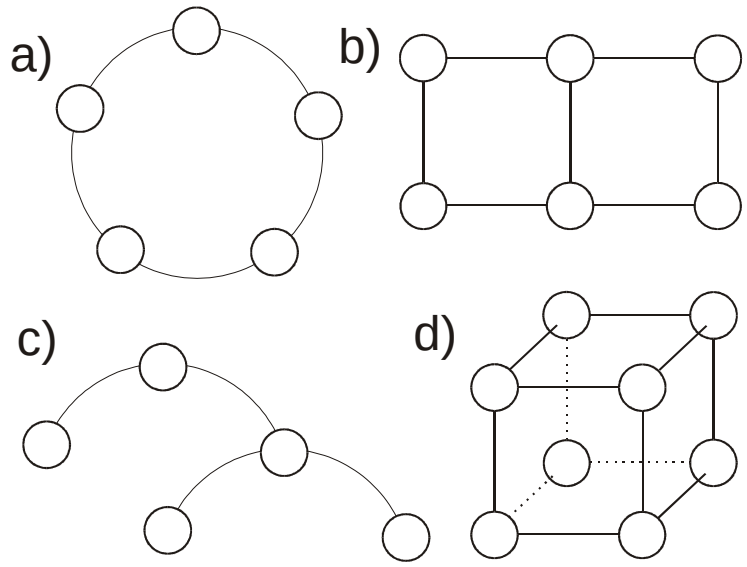
2 Efektywna realizacja obliczeń w klastrach

Efektywność aplikacji obliczeniowych działających w klastrach można analizować na kilku poziomach. Odpowiednia konfiguracja sprzętowa i wybór właściwego oprogramowania stanowi platformę do uruchamiania programów, natomiast modyfikacja algorytmu może zmniejszyć obciążenie klastra komunikacją i wykorzystać efektywniej jego wieloprocesorowe węzły.

2.1 Optymalizacja na poziomie sprzętu

2.1.1 Topologie fizyczne i logiczne

Topologie połączeń sieciowych we współczesnych klastrach mogą mieć różnorodne struktury. Do popularniejszych należą: pierścień, tablica, drzewo i hipersześcian (rys. 2.1).



Rys. 2.1. Różnorodne topologie sieci: a) pierścień, b) tablica, c) drzewo i d) hipersześcian.

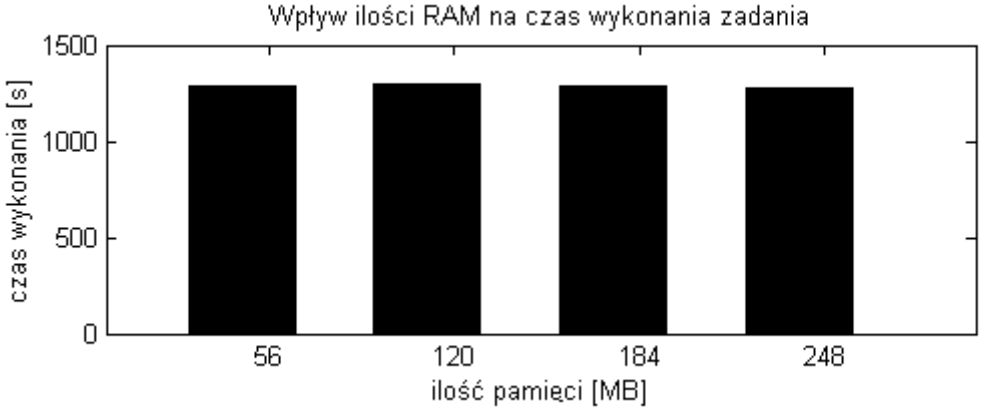
Fizyczne połączenie determinuje w dużym stopniu efektywność wykonywanych na klastrze zadań. Połączenie klastra w pierścień powoduje, że najdłuższa droga, jaką musi przebyć wiadomość z jednego węzła do innego jest równa *N/2* (gdzie *N* to liczba węzłów). W strukturze tablicy każdy węzeł może mieć do czterech sąsiadów – wymusza to posiadanie czterech interfejsów (kart sieciowych) przez niektóre węzły. Najdłuższa droga wiadomości wynosi w tym przypadku $2\left(\sqrt{N}-1\right)$. Topologia drzewa wymusza posiadanie do trzech interfejsów przez każdy z węzłów. Droga wiadomości jest w przybliżeniu proporcjonalna do

$\log_2(N)$. Wydaje się, że jest to dosyć wydajne połączenie, jednak powoduje nadmierne obciążenie węzła, który znajduje się na szczycie drzewa. Droga wiadomości przy połączeniu w hipersześcian jest podobna jak w topologii drzewa jednakże nie posiada węzłów, które będą przeciążane podczas transmisji. Istnieje tu jednak inne ograniczenie. Wraz ze wzrostem liczby węzłów, każdy węzeł musi posiadać większą liczbę interfejsów ($\log_2(N)$).

Popularne jest też inne rozwiązanie (dla małych klastrów) – połączenie wszystkich węzłów poprzez szybką przełącznicę. Połączenie takie umożliwia często komunikację z maksymalną prędkością przez wszystkie węzły jednocześnie przy jednoczesnym wymaganiu posiadania zaledwie jednego interfejsu przez każdy węzeł. Ograniczenie możliwości stosowania tego rozwiązania polega na skończonej liczbie wejść przełącznicy, do których można podłączyć węzły klastra. Niedogodność tą można niwelować łącząc ze sobą większą liczbę przełącznic. Rozwiązanie to warto zastosować w klastrach przystosowanych z istniejących lokalnych sieci komputerowych (zmiana przełącznicy na szybszą i pozostawienie istniejącej konfiguracji wydaje się być bardziej uniwersalnym i tanim podejściem dla małych klastrów niż modyfikacja fizycznych połączeń).

2.1.2 Pamięć operacyjna

Interesującym zagadnieniem jest wpływ ilości pamięci RAM na czas wykonywania określonego zadania. Obserwując zachowanie się komputerów z małą ilością pamięci, wpływ wydaje się oczywisty. Zostało to jednak sprawdzone dla zadania mnożenia macierzy wielkości $2*10^3 x 2*10^3$. Testy przeprowadzono na komputerze *Celeron 500* w systemie Windows XP obniżając ilość fizycznej pamięci RAM nawet poniżej minimalnej zalecanej dla Windows XP, mimo to nie dało się zauważyć żadnych istotnych różnic dla tych przypadków (rys. 2.2). Wnioskiem może być to, że algorytm ten ma stosunkowo małe zapotrzebowanie na pamięć. Dla bardzo dużych macierzy testy są utrudnione z powodu bardzo długiego czasu obliczeń.

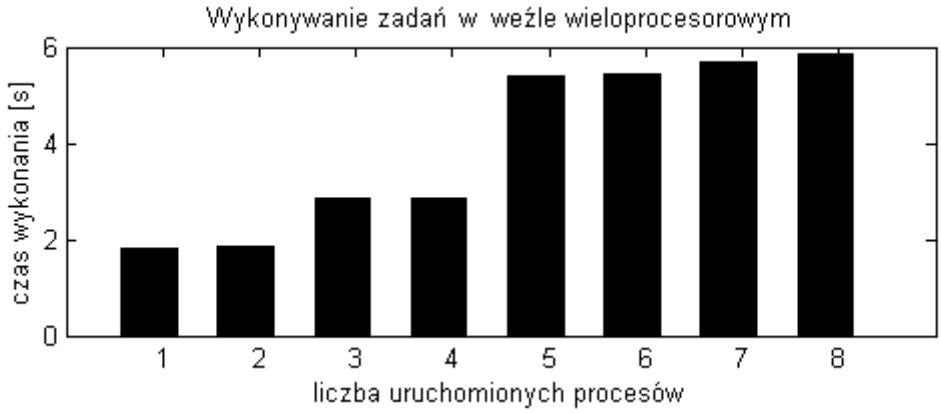


Rys. 2.2. Czas mnożenia macierzy w zależności od dostępnej ilości pamięci operacyjnej.

Można stąd wysnuć wniosek, że dla niektórych zadań wpływ ilości pamięci RAM nie jest zbyt istotny.

2.1.3 Węzły wieloprocesorowe

Następny test przeprowadzono w węźle dwuprocesorowym. Teoretycznie moc obliczeniowa wynikająca z dwuprocesorowości będzie wykorzystana wtedy, gdy w każdym węźle uruchomione będą przynajmniej dwa procesy obliczeniowe. Aby to zbadać, uruchomiono w jednym węźle kilka procesów mnożenia macierzy.



Rys. 2.3. Wykonywanie jednoczesnych obliczeń w dwuprocesorowym węźle.

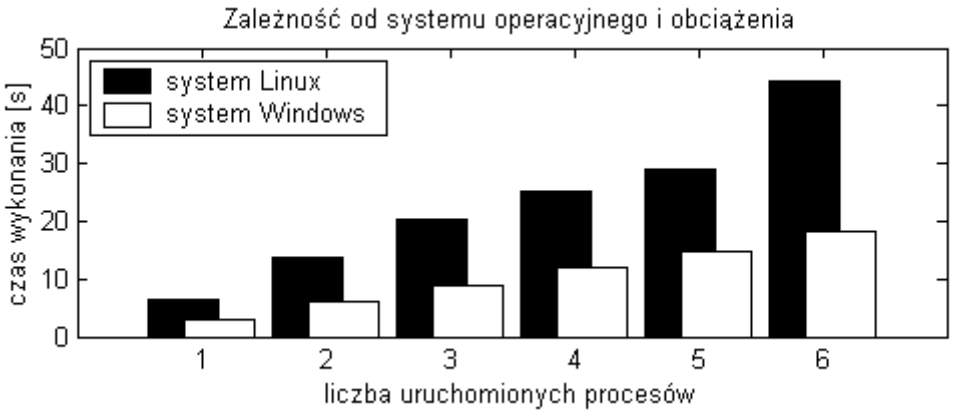
Jak widać z wykresu (rys. 2.3), wykonanie jednego i dwóch zadań zajmuje tyle samo czasu, węzeł jest więc wykorzystywany efektywniej, gdy zostanie na nim jednocześnie wykonane dwa lub więcej procesów (jest to zgodne z oczekiwaniami, ponieważ węzeł jest dwuprocesorowy). Programy powinny być więc tak projektowane aby wykorzystać w pełni możliwości wieloprocesorowych węzłów klastra.

2.2 Rola wykorzystywanego oprogramowania

2.2.1 System operacyjny i obciążenie

W systemach wielozadaniowych można sprawdzić jak na czas wykonania algorytmu wpływa obciążenie procesora kilkoma jednocześnie działającymi procesami. Prezentowane poniżej wartości czasów, dotyczące również problemu mnożenia macierzy, zmierzono dla systemu Linux (kompilator gcc) i Windows XP (kompilowane w Delphi) obciążając je wykonującymi się jednocześnie do *6* procesami mnożenia macierzy. Test wypadł na korzyść

systemu Windows – zmierzone czasy są o ok. połowę krótsze. W systemie Windows jednak czas wykonania zadania obliczeniowego bardzo mocno zależał od tego czy proces był pierwszoplanowy. Niedogodnością w systemie Windows jest fakt, że po uruchomieniu zadania obliczeniowego bardzo trudno wykonać inne czynności - problemu tego nie dało się zauważyć w systemie Linux. W przypadku obydwu systemów operacyjnych przy obciążeniu sześcioma procesami otrzymano czasy wykonania algorytmu ok. *6*-krotnie dłuższe (rys. 2.4), czego można się było intuicyjnie spodziewać.



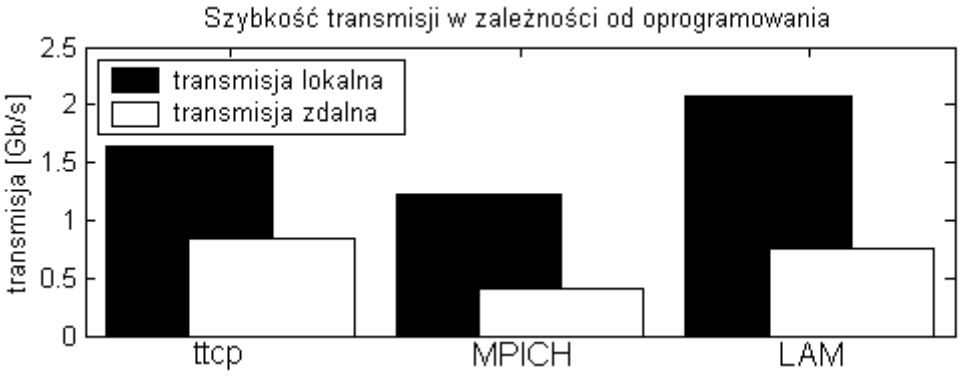
Rys. 2.4. Zależność czasu obliczeń od systemu operacyjnego i obciążenia procesora.

2.2.2 Oprogramowanie klastrowe

Istnieje co najmniej kilka standardów oprogramowania klastrowego. Do popularniejszych należą MPI (Message Passing Interface) i PVM (Parallel Virtual Machine). Standardy te opisują jedynie interfejs tzn. zestaw dostępnych funkcji, efektywność natomiast zależna jest od konkretnych implementacji. Standard MPI promowany jest przez wiele ośrodków naukowych, więc w dalszej części omawiane będzie właśnie to oprogramowanie.

Aby porównać różne implementacje tego standardu zbadana została prędkość przesyłania danych pomiędzy procesami znajdującymi się w tym samym i w różnych węzłach klastra. Wartość ta była badana przy pomocy programu ttcp oraz oprogramowania MPICH i LAM. Program ttcp bada przepustowość sieci mierząc czas kopiowania plików pomiędzy węzłami poprzez gniazda. MPICH i LAM to implementacje MPI, które umożliwiają tworzenie programów składających się z wielu procesów uruchamianych w różnych węzłach klastra. Komunikacja między tymi węzłami odbywa się poprzez przesył komunikatów.

Program ttcp uzyskał prędkość kopiowania danych *0,84 Gb/s* (rys. 2.5). Osiągnięto więc ok. *84%* teoretycznych możliwości sprzętu (*1 Gb/s*).

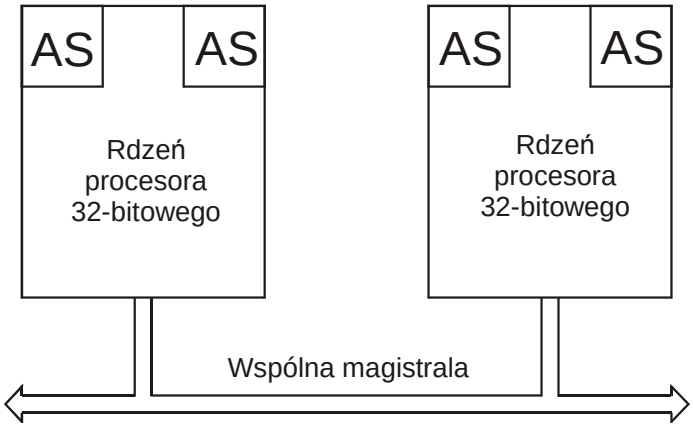


Rys. 2.5. Porównanie szybkości transmisji lokalnej i zdalnej zmierzonej za pomocą programu ttcp oraz oprogramowania MPICH i LAM.

Przesyłając dane pomiędzy procesami lokalnymi (realizowanymi na tym samym węźle) otrzymano szybkość ok. dwukrotnie wyższą (brak ograniczeń mediów transmisyjnych). Transmisje między procesami zdalnymi (realizowanymi na różnych węzłach) wykonane przy pomocy oprogramowania MPICH okazały się ok. dwukrotnie wolniejsze, natomiast w przypadku transmisji lokalnej prędkość spadła o ok. *25%*. Oprogramowanie LAM dla transmisji zdalnej uzyskało prędkość *0,75Gb/s*, co jest wynikiem prawie dwukrotnie lepszym niż w MPICH. Dla transmisji lokalnej LAM osiągnął nawet lepsze wyniki niż program ttcp.

2.3 Uwzględnienie architektury wieloprocesorowej

W technologiach wieloprocesorowych oraz HyperThread zyski czasowe w aplikacjach obliczeniowych są widoczne podczas uruchamiania kilku procesów w jednym węźle. Eksperymenty wykonywane były na węzłach dwuprocesorowych, każdy o architekturze HyperThread, tzn. składający się z dwóch logicznych procesorów podłączonych do wspólnej magistrali (rys. 2.6). Architektura taka pozwala teoretycznie na uzyskanie około czterokrotnego przyspieszenia algorytmów. Systemy operacyjne wykorzystują taką architekturę, aby różne procesy wykonywały się na różnych procesorach. Zyski będą widoczne, gdy algorytm zostanie podzielony pomiędzy wykonujące się procesy. Dla różnych algorytmów uzyskane przyspieszenie może być różne. W skrajnych przypadkach programy mogą zachowywać się tak jakby były uruchamiane w architekturze jednoprocesorowej. Technologia HyperThread polega na umieszczeniu wewnątrz jednego fizycznego procesora dwu lub więcej procesorów logicznych. Każdy logiczny procesor posiada prawie kompletny zestaw rejestrów umożliwiający zapamiętanie stanu procesora oraz własny kontroler przerwań. Wszystkie logiczne procesory mają natomiast wspólny rdzeń, czyli część wykonawczą oraz interfejs magistrali systemowej. W rezultacie technologia HT może nie dawać tak dobrych rezultatów jak systemy wieloprocesorowe.



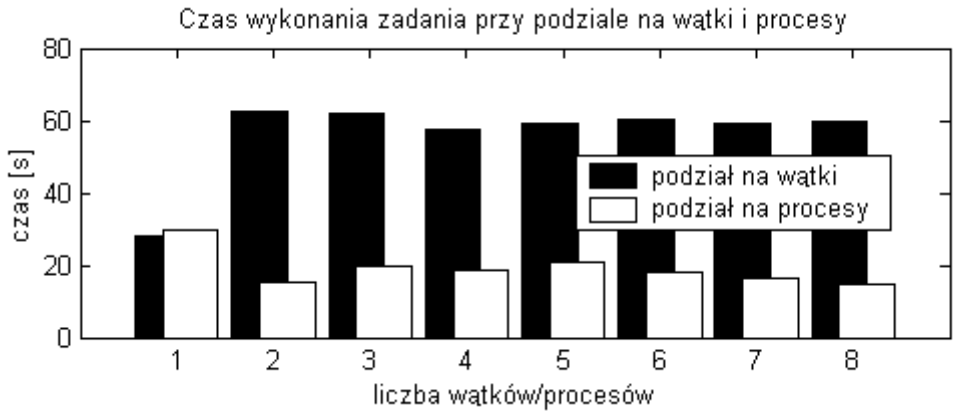
Rys. 2.6. Dwa procesory o architekturze HyperThread podłączone do wspólnej magistrali systemowej. AS-zapamiętany stan procesora logicznego.

Duże znaczenie ma zarówno sposób, w jaki system operacyjny przydziela czas zadaniom jak i liczba odwołań procesorów do wspólnej pamięci. Na system operacyjny można wpłynąć np. poprzez zmianę priorytetów procesów, jednak powoduje to raczej przywłaszczanie sobie czasu innych procesów niż zwiększenie efektywności. Modyfikując algorytm można w znacznym stopniu zredukować liczbę odwołań do pamięci oraz wykorzystać specyficzne możliwości procesorów.

2.3.1 Podział zadania na procesy i wątki

Systemy operacyjne udostępniają mechanizmy wspomagające obliczenia w systemach wieloprocesorowych ze wspólną pamięcią (jeden węzeł klastra). Poprzez tworzenie nowych procesów lub wątków można zrównoleglić obliczenia. Każdy z tych mechanizmów jest możliwy do wykorzystania również w MPI. Możliwość podziału procesu na wątki została wprowadzona z powodu mniejszego obciążenia systemu wątkami niż procesami, poza tym czas zainicjowania nowego wątku jest wielokrotnie krótszy niż nowego procesu. Wątki w obrębie jednego procesu współdzielą ten sam obszar danych globalnych, natomiast przy podziale na procesy dane muszą zostać skopiowane.

Aby porównać zysk czasowy osiągany przy podziale zadania na wątki i procesy, zostało uruchomione zadanie równoległego mnożenia macierzy w jednym węźle klastra. Węzeł posiadał dwa procesory w technologii HyperThread. Rozmiar mnożonych macierzy wynosił *1000x1000*. Zadanie było dzielone na *1* do *8* wątków/procesów. Wyniki pomiarów przedstawione są na rysunku 2.7.

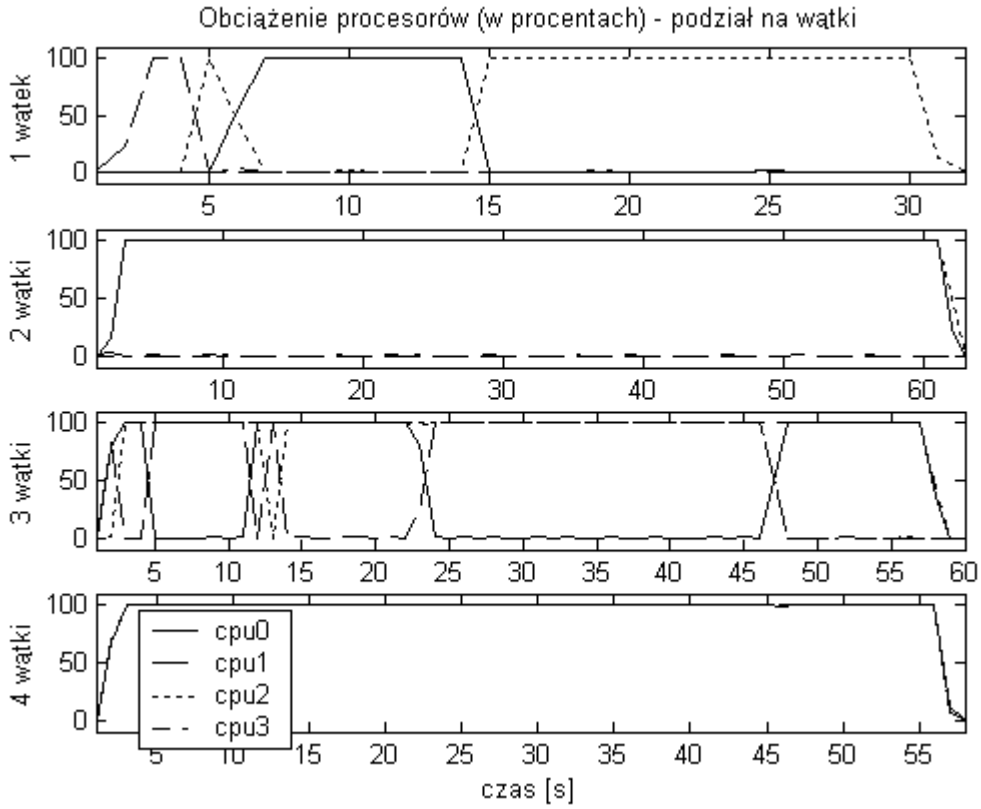


Rys. 2.7. Wyniki pomiarów czasu równoległego mnożenia macierzy w jednym węźle klastra przy podziale na wątki i procesy.

Wydawać by się mogło, że wykorzystanie wspólnej przestrzeni adresowej wątków znacznie przyspieszy zadanie. Wyniki jednak zaprzeczają tym dywagacjom. Okazuje się, że skopiowanie potrzebnych danych w tego typu zadaniach jest zaniedbywalnie małe w porównaniu ze stratami czasowymi wynikającymi z wykorzystywania wspólnego obszaru danych. Konflikty procesorów w odwołaniach do pamięci okazują się na tyle duże, że zwiększają czas obliczeń przeszło dwukrotnie. Natomiast podział na procesy przyspiesza zadanie prawie dwukrotnie.

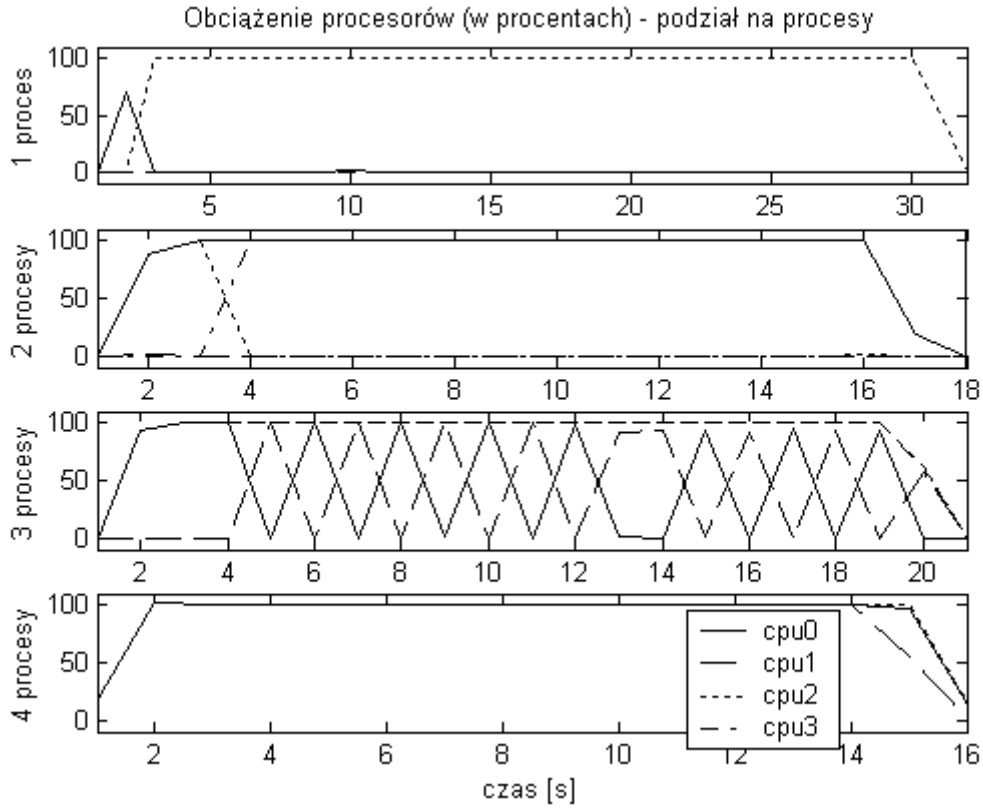
2.3.2 Śledzenie obciążenia procesorów

Węzeł w wykorzystywanym klastrze jest dwuprocesorowy, jednak w z powodu architektury HyperThread w systemie widoczne są aż cztery procesory. W czasie wykonywania zadania włączono monitoring obciążenia poszczególnych procesorów. Wyniki dla podziału zadania na *1* do czterech wątków pokazane są na rysunku 2.8. Obsługa wątków przekazywana jest pomiędzy procesorami nawet kilkakrotnie w trakcie działania programu. Najmniej zmian odnotowano przy podziale programu na dwa i cztery wątki.



Rys. 2.8. Procentowe obciążenie procesorów przy podziale zadania na wątki.

Przy podziale zadania na trzy procesy (rys. 2.9) można zauważyć jakby zjawisko migotania - proces przerzucany jest pomiędzy procesorem `cpu0` i `cpu1`. Nie wpływa to jednak w widoczny sposób na czas wykonania tego procesu.



Rys. 2.9. Procentowe obciążenie procesorów przy podziale zadania na procesy.

Wniosek, jaki można wyciągnąć z tych pomiarów dla tego typu zadania uruchamianego w tego rodzaju węzłach: należy unikać odwołań do wspólnej pamięci wykonywanych równolegle przez różne procesory. Zysk uzyskany z braku kopiowania danych jest niewspółmiernie mały w porównaniu ze stratami wynikającymi z konfliktów podczas odwołań do wspólnych obszarów pamięci.

2.4 Efektywne algorytmy rozproszone

Efektywność obliczeń rozproszonych zależna jest nie tylko od używanego sprzętu i oprogramowania, ale też od sposobu realizacji zadania. W poniższych przykładach przedstawione zostaną sposoby na zoptymalizowanie obliczeń na poziomie algorytmu.

2.4.1 Przykład równoległego mnożenia macierzy

W tym podpunkcie przeanalizowany będzie problem efektywnej realizacji zadania równoległego mnożenia macierzy w klastrze.

W przypadku podziału na dwa równe podzadania połowę wyników otrzymać można w lokalnym węźle, połowę natomiast w zdalnym. Jeżeli macierz zostanie podzielona linią poziomą na dwie części, wówczas wystąpi konieczność przesłania do węzła zdalnego $\frac{3}{2}N^2$

współczynników oraz ze zdalnego do lokalnego $\frac{1}{2}N^2$ wyników (rys. 2.10).

$$P_1 = \frac{3}{2}N^2 + \frac{1}{2}N^2 = 2N^2 \tag{2.1}$$

Rys. 2.10. Przesył i wykonanie zdalnych obliczeń przy podziale na dwa podzadania. Podział pierwszą metodą.

Jeżeli w zdalnym węźle zamierza się przeprowadzić bądą obliczenia dla kwadratowego wycinka macierzy, to należy wysłać do niego $\sqrt{2}N^2$ współczynników a ze zdalnego do lokalnego $\frac{1}{2}N^2$ wyników (rys. 2.11).

$$P_2 = \sqrt{2}N^2 + \frac{1}{2}N^2 \approx 1{,}91N^2 \tag{2.2}$$

Rys. 2.11. Przesył i wykonanie zdalnych obliczeń przy podziale na dwa podzadania. Podział drugą metodą.

Druga metoda podziału jest więc bardziej opłacalna, chociaż pierwsza wydaje się bardziej intuicyjna. Czas obliczeń na poszczególnych węzłach zależny jest głównie od ilości wykonywanych operacji mnożenia i jest on równy dla obydwu metod podziału zadania (2.3).

$$M = \frac{1}{2}N^3 \tag{2.3}$$

Ostatecznie czas równoległego mnożenia macierzy będzie równy (2.4):

$$C = P \cdot C_P + M \cdot C_M \tag{2.4}$$

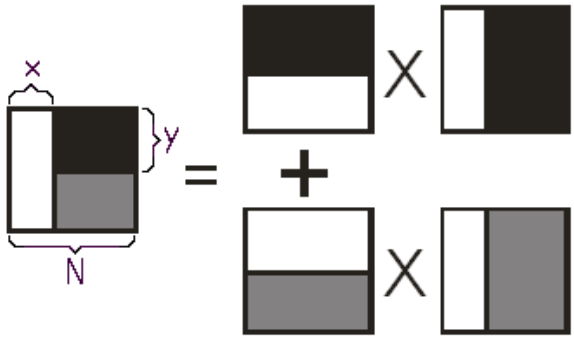
gdzie:

C_P - czas przesyłu jednego współczynnika,

C_M - czas wykonania jednej operacji mnożenia skalarnego.

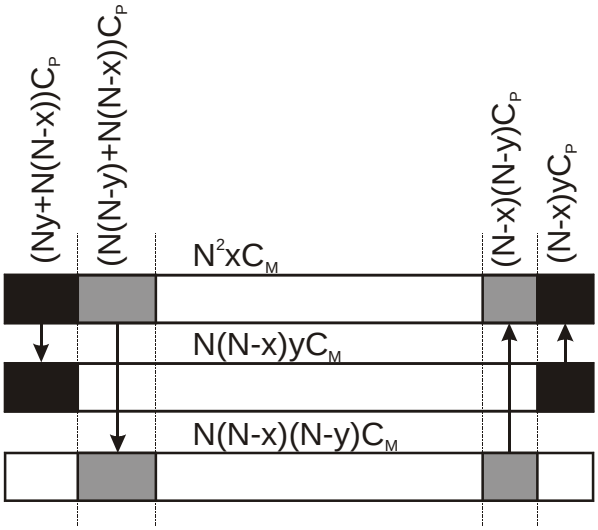
W przypadku podziału zadania na trzy podzadania węzeł lokalny będzie w większym stopniu obciążony transmisją danych niż węzły zdalne. Dlatego wydaje się logiczne obciążenie węzłów zdalnych większą ilością obliczeń. Jednak pociąga to za sobą zwiększenie ilości przesyłanej informacji i może zwiększyć całkowity czas obliczeń.

Proponowany przez autora niniejszej pracy sposób podziału zadania na węzły zdalne przedstawiony jest na rysunku 2.12



Rys. 2.12. Przesył i wykonanie zdalnych obliczeń przy podziale na trzy podzadania.

Biała część macierzy obliczana jest lokalnie. Reszta dzielona jest na dwa węzły zdalne.



Rys. 2.13. Przebieg czasowy transmisji i obliczeń przy podziale zadania na trzy podzadania.

Jeżeli liczbę kolumn macierzy obliczanych lokalnie oznaczy się przez *x* a liczba wierszy obliczanych w pierwszym węźle zdalnym przez *y* to czas potrzebny na wykonanie wszystkich zadań w węźle lokalnym jest równy (2.5):

$$C_L = \left(4N^2 - 3Nx\right)C_P + N^2 x C_M \tag{2.5}$$

Czas potrzebny na wykonanie zadań przesyłu i obliczeń w węzłach zdalnych to (2.6):

$$\begin{aligned} C_{Z1} &= \left(Ny + \left(N - x\right)\left(N + y\right)\right)C_P + \left(N - x\right)NyC_M \\ C_{Z2} &= \left(4N^2 - 3Nx\right)C_P + N\left(N - x\right)\left(N - y\right)C_M \end{aligned} \tag{2.6}$$

Całkowity czas mnożenia macierzy dla ustalonych wartości *N*, C_M i C_P wynosi (2.7):

$$C_C\left(x, y\right) = Max\left(C_L, C_{Z1}, C_{Z2}\right) \tag{2.7}$$

Należy jeszcze tylko znaleźć *x* i *y*, dla których powyższa funkcja przyjmuje minimum. Przy podziale na większą ilość podzadań znalezienie optymalnej metody podziału staje się coraz bardziej utrudnione a ilość niewiadomych w równaniach rośnie.

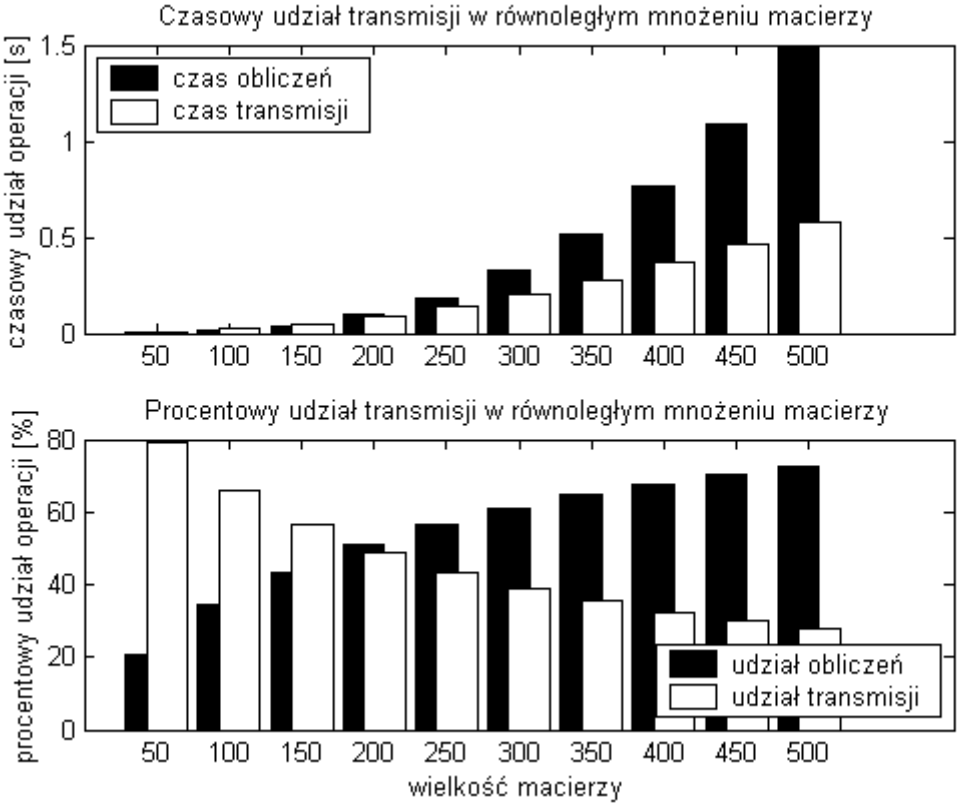
Przykładowe średnie czasy zmierzone w klastrze:

Czas mnożenia dwóch zmiennych o podwójnej precyzji (*double*): $C_M = 24ns$

Czas przesyłu przez sieć jednej zmiennej typu *double* (*8* bajtów): $C_P = 1{,}2\mu s$

Dla takiej proporcji czasów nasuwa się pytanie, czy w ogóle warto dzielić zadanie na części i przesyłać siecią.

Ilość danych, które należy przesłać przez sieć rośnie proporcjonalnie do kwadratu wielkości macierzy. Jak wiadomo złożoność czasowa samego mnożenia macierzy jest rzędu N^3. Można więc przypuszczać, że nawet w przypadku wolnej sieci i szybkich komputerów opłaca się dla pewnych wielkości macierzy zastosować podział na części.

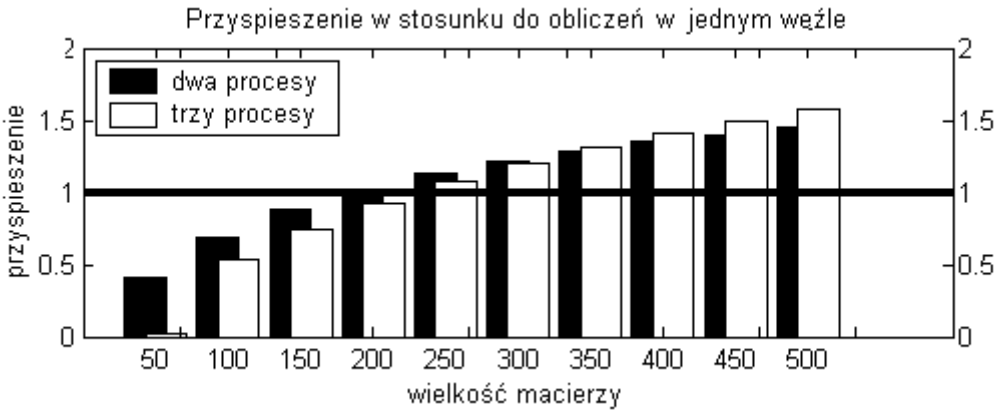


Rys. 2.14. Czasowy i procentowy udział obliczeń i transmisji podczas mnożenia macierzy przy podziale na dwa podzadania dla różnych wielkości macierzy.

Na wykresie (rys. 2.14) można zauważyć, że o ile przy wielkości macierzy *200x200* czas przesyłu danych jest podobny jak czas obliczeń, to dla wielkości *500x500* stanowi już tylko około *25%*.

Na tej podstawie wyciągnąć można wnioski dotyczące podziału zadania na części.

Wykres (rys. 2.15) przedstawia przyspieszenie, jakie zostanie uzyskane przy podziale na podzadania w stosunku do wykonywania zadania bez podziału. Dla podanych wyżej przykładowych parametrów sieci i prędkości procesorów otrzymano następujące wyniki: dla macierzy o wielkości do *200x200* nie ma sensu dzielenia zadania na podzadania. Dla macierzy z przedziału od *200x200* do *350x350* warto zadanie podzielić na dwa podzadania. Dla rozmiarów większych należy podzielić na trzy podzadania. Analogicznie można sprawdzić podział na większą liczbę podzadań.



Rys. 2.15. Przyspieszenie uzyskane przy podziale zadania na dwa i trzy podzadania.

3 Rozwiązywanie układów równań liniowych

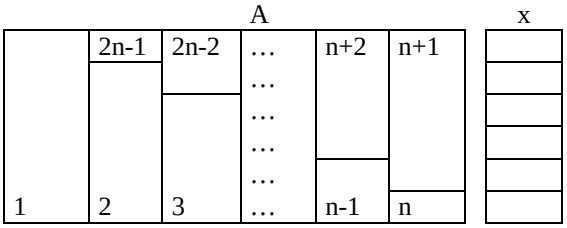
Analizowane w tym rozdziale macierze, to macierze pełne. Dla porównania przetestowane zostały wygenerowane losowo macierze o wszystkich współczynnikach z przedziału *(0, 1)*, oraz takie, które na przekątnej posiadają współczynniki większe niż suma elementów w wierszu. Szybkość zbieżności algorytmów rozwiązywania układów równań liniowych w znacznym stopniu różni się dla różnych typów macierzy. Uzyskane wyniki stanowią podstawę do wyboru metod, które będą najlepiej spełniały oczekiwania stawiane dla konkretnych zadań.

3.1 Metody dokładne

Metody dokładne (czasami nazywane skończonymi) są najczęściej używanymi metodami w przypadku rozwiązywania pełnych układów równań i układów pasmowych [BSS].

3.1.1 Metoda eliminacji Gaussa

Jest to najprostsza z metod dokładnych. Polega ona na eliminowaniu (zerowaniu) kolejnych współczynników macierzy tak, aby otrzymać macierz jednostkową (w realizacji Gaussa-Jordana). Początkowa wartość wektora *x* jest równa wektorowi wyrazów wolnych *b*. Wektor *x* jest przekształcany wraz z wierszami macierzy *A*. Po przekształceniu macierzy *A* w macierz jednostkową, w wektorze *x* otrzymujemy rozwiązanie układu równań (rys. 3.1).



Rys. 3.1. Kolejność eliminacji w metodzie Gaussa.

W przypadku macierzy rzadkich ignorowana jest i burzona ich korzystna (z punktu widzenia złożoności obliczeniowej) struktura (rys. 3.2).

$$\left[\begin{array}{cccc|c} a & 0 & b & c & d \\ e & f & 0 & 0 & g \\ 0 & 0 & h & 0 & i \\ j & 0 & 0 & k & l \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} 1 & 0 & m & n & o \\ 0 & 1 & p & q & r \\ 0 & 0 & 1 & 0 & s \\ 0 & 0 & 0 & 1 & t \end{array}\right]$$

Rys. 3.2. Po wyzerowaniu niezerowych elementów pod przekątną inne, zerowe elementy zmieniają wartość.

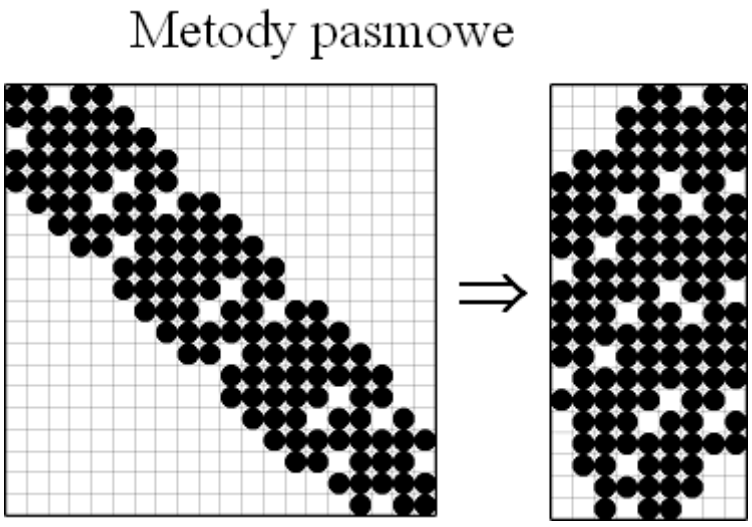
Chociaż metoda jest nazywana dokładną, to w przypadku obliczeń numerycznych na liczbach o skończonej dokładności dochodzi do błędów zaokrągleń. Aby zminimalizować powstające w ten sposób błędy należy zmieniać kolejność wierszy i kolumn macierzy przed każdym krokiem eliminacji tak, aby na pozycji diagonalnej znalazł się element o największej wartości bezwzględnej.

3.1.2 Metoda Cholesky'ego

Metodę tą można stosować, gdy macierz współczynników A układu równań jest symetryczna i dodatnio określona. Należy dokonać dekompozycji macierzy A, tzn. wyznaczyć taką dolną trójkątną macierz L, aby zachodziło $A=LL^T$. Po dokonaniu dekompozycji otrzymujesię równanie postaci $LL^Tx=b$. Najpierw rozwiązywany jest układ $Lv=b$, a następnie $L^Tx=v$. Metoda algebraicznie podobna jest do metody eliminacji Gaussa. Wymaga mniejszej ilości obliczeń w czasie eliminacji kosztem dokonanej wcześniej dekompozycji. Na uwagę zasługuje również metoda Crouta, również podobna do metody eliminacji Gaussa, w której występuje jednak inna kolejność obliczania elementów macierzy.

3.1.3 Techniki pasmowe

Techniki pasmowe zapisu macierzy rzadkich stosuje się przy do rozwiązywania układów równań, dla których macierz współczynników ma strukturę pasmową, tzn. $A_{ij}=0$ dla $j>i+m$ oraz $j<i-m$. Do rozwiązywania takich układów równań stosuje się wymienione wcześniej metody dokładne. Różnica polega na oszczędniejszym wykorzystaniu pamięci operacyjnej oraz na pomijaniu podczas obliczeń elementów spoza pasma (rys. 3.3).

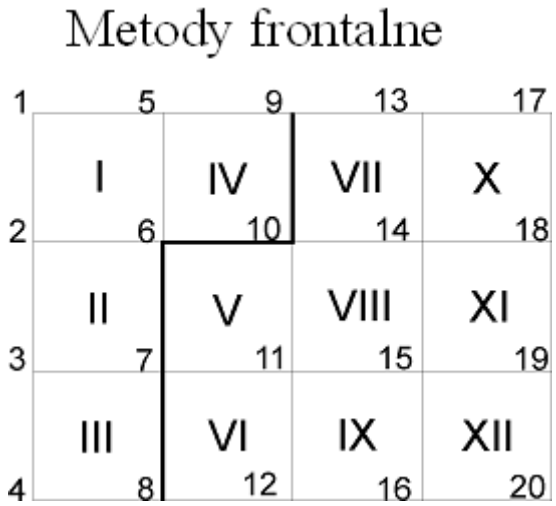


Rys. 3.3. Implementacja macierzy pasmowych.

Dla macierzy pasmowych nie stosuje się przestawiania wierszy i kolumn macierzy. Ilość zajmowanej pamięci jest bardzo istotna, gdyż wielkości macierzy dla złożonych zadań (np. MES) mogą przekraczać rozmiary dostępnej pamięci operacyjnej. Wykorzystanie specyficznej struktury macierzy (macierze rzadkie, pasmowe) pozwala zmniejszyć złożoność pamięciową i obliczeniową algorytmu. Stosowane są również metody zmiennopasmowe, gdzie parametr *m* zmienia się dla różnych wierszy macierzy lub pomijane są w ten sposób zerowe „dziury” wewnątrz pasma.

3.1.4 Techniki typu frontalnego

Techniki frontalne również opierają się na metodach dokładnych, zazwyczaj na metodzie eliminacji Gaussa. Organizacja obliczeń w metodach frontalnych związana jest ściśle z numeracją elementów w metodzie elementów skończonych.



Rys. 3.4. Przykładowa numeracja elementów i węzłów w MES

Po wygenerowaniu macierzy dla elementu *I* (rys. 3.4) otrzymuje się równania dla węzłów siatki *1, 2, 5* i *6*. Następnie można zauważyć, że węzeł pierwszy nie występuje już w żadnym innym elemencie skończonym. Można więc dokonać eliminacji zmiennej x_1 zgodnie z zasadami eliminacji Gaussa. W tym kroku algorytmu front obejmuje węzły *2, 5* i *6*, natomiast węzeł *1* znajduje się już poza frontem. W następnym kroku analizuje się element *II*. Współczynniki macierzy dla tego elementu dodaje się w odpowiednie pozycje utworzonej wcześniej macierzy. Zmienne *x2* oraz *x3* są już w pełni określone, można więc dokonać ich eliminacji a front przesunąć na węzły *3, 5, 6 i 7*. Algorytm ma mniejsze zapotrzebowanie na pamięć operacyjną niż metody pasmowe. Bardziej szczegółowe omówienie metod frontalnych można znaleźć np. w pracach [AMJ, Hoo, Iro].

3.2 Metody iteracyjne - stacjonarne

Metody stacjonarne mogą być przedstawione w postaci następującego wzoru (3.1):

$$x^{(k)} = Bx^{(k-1)} + c \tag{3.1}$$

gdzie *B* i *c* są stałymi i nie zależą od aktualnej iteracji *k*. Opisane zostaną cztery metody stacjonarne: Jacobiego, Gaussa-Seidla, SOR i SSOR.

3.2.1 Metoda Jacobi'ego

Rozważa się następujący układ równań:

$$Ax=b,$$

gdzie:

A jest macierzą o wymiarze nxn natomiast x i b są n-wymiarowymi wektorami.

Metoda Jacobiego bazuje na wyznaczaniu wartości osobno każdej niewiadomej w zależności od aktualnych wartości pozostałych niewiadomych. Jedna iteracja metody, to wyznaczenie wszystkich niewiadomych układu. Metoda jest łatwa do zrozumienia, więc podane zostanie jej wyprowadzenie. Badając osobno każde równanie układu $Ax=b$, można je zapisać w postaci wzoru (3.2):

$$\sum_{j=1}^{n} a_{i,j} x_j = b_i \tag{3.2}$$

można rozwiązać niewiadomą x_i znając pozostałe niewiadome za pomocą zależności (3.3):

$$x_i = (b_i - \sum_{j \neq i} a_{i,j} x_j) / a_{i,i} \tag{3.3}$$

Sugeruje to zastosowanie następującej iteracji (3.4):

$$x_i^{(k)} = (b_i - \sum_{j \neq i} a_{i,j} x_j^{(k-1)}) / a_{i,i} \tag{3.4}$$

gdzie: k - kolejny numer iteracji

Jest to właśnie metoda Jacobi'ego. Kolejność, w której równania są badane, jest dowolna, gdyż są one niezależne od siebie. Zapis macierzowy tej metody wygląda następująco (3.5):

$$x^{(k)} = D^{-1}(L+U)x^{(k-1)} + D^{-1}b \tag{3.5}$$

gdzie *D*, *-L*, *-U* to nxn wymiarowe macierze: diagonalna, trójkątna-dolna i trójkątna-górna utworzone bezpośrednio z macierzy *A*. Metoda wymaga, aby w każdej iteracji dokonane zostało mnożenie macierzy przez wektor.

3.2.2 Metoda Gaussa-Seidla

Metoda ta wprowadza zależność następującej postaci (3.6):

$$x_i^{(k)} = (b_i - \sum_{j<i} a_{i,j} x_j^{(k)} - \sum_{j>i} a_{i,j} x_j^{(k-1)}) / a_{i,i} \tag{3.6}$$

Kolejność rozwiązywania równań w tej metodzie jest istotna, gdyż analizowane są wartości wyliczone już w bieżącej iteracji. Wynik iteracji $x^{(k)}$ zależny jest od kolejności, w której obliczane są niewiadome. Jeżeli kolejność będzie inna, obliczone wartości również będą inne. Te własności mogą się okazać istotne w równoległym przetwarzaniu macierzy rzadkich. Wykorzystując istnienie zer w macierzach można grupować równania tak, aby zminimalizować wpływ wyników jednych równań na inne. W zapisie macierzowym zależność (3.6) przyjmuje postać (3.7):

$$x^{(k)} = (D-L)^{-1}(Ux^{(k-1)} + b) \tag{3.7}$$

Podobnie jak wcześniej: *D*, *-L*, *-U* to macierze: diagonalna, trójkątna-dolna i trójkątna-górna utworzone bezpośrednio z macierzy *A*.

3.2.3 Metoda SOR

Metoda SOR (Successive Overrelaxation) może być wyprowadzona z metody Gaussa-Seidla przez wprowadzenie dodatkowego parametru ω. Przy optymalnym wyborze tego parametru zbieżność może być szybsza niż dla metody Gaussa-Seidla. Zapis macierzowy tej metody jest następujący (3.8):

$$x^{(k)} = (D-\omega L)^{-1}(\omega U + (1-\omega)D)x^{(k-1)} + \omega(D-\omega L)^{-1}b \tag{3.8}$$

Dla $\omega=1$ równanie upraszcza się do postaci równania w metodzie Gaussa-Seidla. Jak podaje się w Literaturze [Kah] metoda SOR dla ω spoza przedziału $(0, 2)$ jest rozbieżna. W praktyce stosowane są metody heurystyczne określenia ω, dla którego metoda będzie działała efektywnie, jednakże znalezienie optymalnej wartości ω może się okazać zbyt kosztowne w

porównaniu z osiąganym zyskiem. W innych realizacjach do określania tego parametru wykorzystywane są też metody adaptacyjne.

3.2.4 Metoda SSOR

Metoda SSOR (Symmetric Successive Overrelaxation) wykorzystuje metodę SOR w ten sposób, że obliczana jest dwukrotnie SOR, a następnie wyliczana jest ich kombinacja. Równanie macierzowe opisujące tą metodę jest następujące (3.9):

$$x^{(k)} = B_1 B_2 x^{(k-1)} + \omega(2-\omega)(D-\omega U)^{-1} D (D-\omega L)^{-1} b \tag{3.9}$$

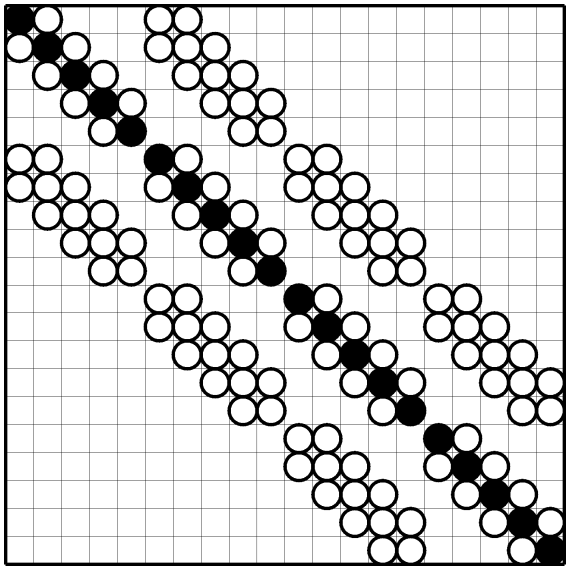
gdzie:

$$B_1 = (D-\omega U)^{-1}(\omega L + (1-\omega)D),$$

$$B_2 = (D-\omega L)^{-1}(\omega U + (1-\omega)D)$$

3.2.5 Porównanie metod stacjonarnych

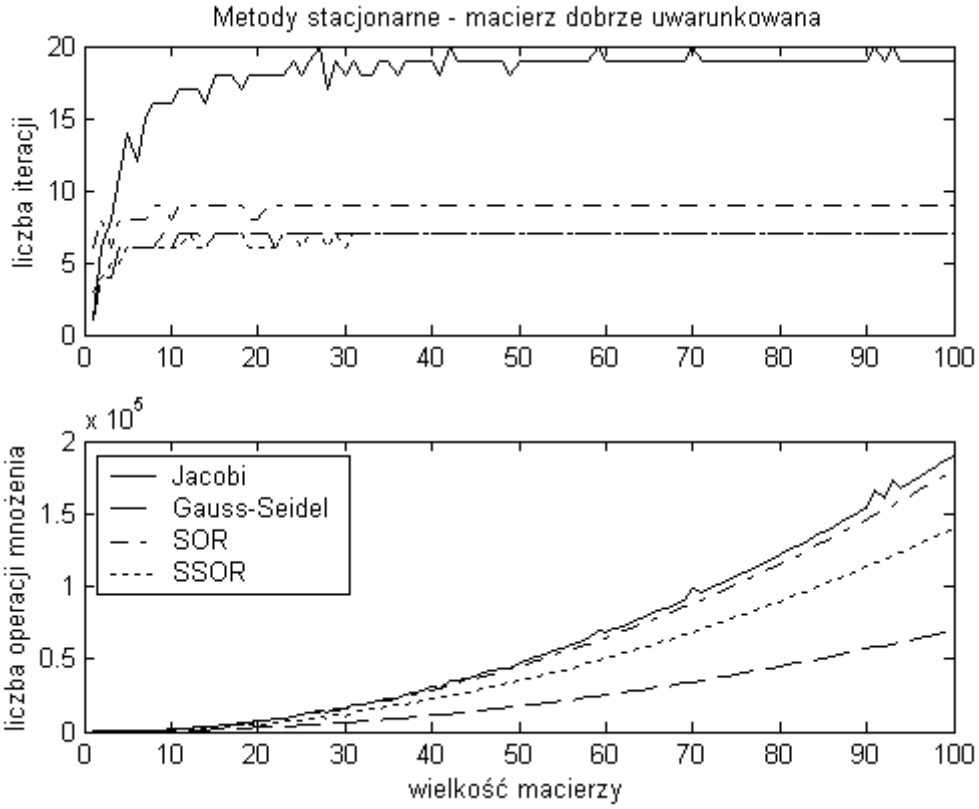
Testy metod prowadzone były dla macierzy dobrze uwarunkowanych (tzn. wartości elementów na przekątnej były większe niż suma elementów w wierszu - rys. 3.5) oraz słabo uwarunkowanych (tzn. wszystkie elementy macierzy były losowe z przedziału *(0, 1)*.



Rys. 3.5. Struktura macierzy rzadkiej, dobrze uwarunkowanej. Puste pola to wartości zerowe; białe i czarne koła oznaczają odpowiednio niskie i wysokie wartości współczynników.

Obliczenia były prowadzone dla macierzy o rozmiarze od *1* do *100*. Algorytm zatrzymywał się po osiągnięciu oczekiwanej dokładności wyników *eps* równej $max\_eps=10^{-6}$ lub po przekroczeniu maksymalnej liczby iteracji $max\_step=2*10^4$. Dla parametru $\omega=1$

algorytm Gaussa-Seidla i SOR są tożsame. Dla rozróżnienia tych metod przyjęto więc $\omega=1,1$ zarówno dla SOR jak i dla SSOR. Najszybszy okazał się algorytm Gaussa-Seidla, następnie kolejno SSOR, SOR i Jacobiego (rys. 3.6).



Rys. 3.6. Porównanie metod stacjonarnych dla macierzy dobrze uwarunkowanych.

Dla macierzy słabo uwarunkowanych powyższe algorytmy okazały się rozbieżne. Dla wielu zadań potrzebne okazuje się wykorzystanie innej klasy metod iteracyjnych.

3.3 Metody iteracyjne - niestacjonarne

Metody niestacjonarne różnią się od stacjonarnych tym, że wykorzystują one informację, która zmienia się w trakcie iteracji.

3.3.1 Metoda CG

Metoda CG (Conjugate Gradient) wykorzystywana jest do obliczania układów symetrycznych, dodatnio określonych. Algorytm metody zaimplementowanej w języku Matlab jest następujący:

```
function x = cg(A, b, max_eps, max_step)
x = zeros(size(b)); r = b; p = r;
```

```
eps = max_eps+1; step = 0;
while ((eps>max_eps)&(step<max_step))
    ro_1 = r'*r;
    if (step>0)
        beta = ro_1/ro_2;
        p = r+beta*p;
    end
    q = A*p;
    alfa = ro_1/(p'*q);
    x = x+alfa*p;
    r = r-alfa*q;
    ro_2 = ro_1;
    eps = max(abs(A*x-b));
    step = step+1;
end
```

Wektor przybliżonych rozwiązań uaktualniany jest za pomocą wektora kierunku *p* i współczynnika kroku α (3.10):

$$x^{(k)} = x^{(k-1)} + \alpha p^{(k)} \tag{3.10}$$

Uaktualniany jest też wektor kierunku *p* oraz ortogonalny do niego wektor residuów *r* (3.11):

$$\begin{aligned} p^{(k)} &= r^{(k-1)} + \beta p^{(k-1)} \\ r^{(k)} &= r^{(k-1)} - \alpha A p^{(k)} \end{aligned} \tag{3.11}$$

Obliczanie warunku stopu `eps = max(abs(A*x-b))` jest dosyć złożone obliczeniowo i można je zastąpić inną zależnością np. `eps = norm(r)/norm(x)`.

3.3.2 Rozwinięcia metody CG

Metody MINRES i SYMMLQ to warianty metody CG dla symetrycznych, ale nie koniecznie dodatnio określonych macierzy. Metody bliżej zostały opisane przez Paige, Parlett i Van der Vorst [PPV] (MINRES) oraz Paige i Saunders [PS1, PS2]. GMRES to rozszerzenie MINRES na układy niesymetryczne. Metoda ta posiada wiele odmian. Cechą charakterystyczną metody jest potrzeba gromadzenia dodatkowych wektorów w każdej iteracji, co powoduje, że zajętość pamięci i nakład czasowy obliczeń zwiększają się liniowo wraz z kolejnymi iteracjami. Aby ominąć ten problem stosuje się restarty algorytmu po pewnej liczbie iteracji. Zgromadzone dane są usuwane a obliczone wyniki służą jako dane

startowe po restarcie. Więcej na temat GMRES znaleźć można w publikacjach [DDS] oraz [DHV]. Metody CGNE i CGNR to rozwinięcia CG działające na prostej zależności $A^TAx=A^Tb$. Iloczyn A^TA jest zawsze symetryczny i dodatnio określony, można więc ten układ obliczyć przy wykorzystaniu metody CG. Dodatkowy nakład obliczeń potrzebny jest na otrzymanie iloczynu A^TA. Inne rozwinięcie metody CG na układy niesymetryczne jest omawiane w literaturze przez Paige i Saunders [PS2]. Wymaga ono zbudowania układu równań postaci (3.12):

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{3.12}$$

Omijane w ten sposób zostaje kosztowne obliczenie iloczynu A^TA kosztem obliczeń dokonywanych na macierzy o dwukrotnie większych rozmiarach. Inną metodą, posiadającą cechy wspólne z GMRES jest QMR, posiadająca wiele odmian. Cechą charakterystyczną niektórych implementacji są zabezpieczenia przed załamaniem się algorytmu poprzez analizowanie w przód. Odmiany tego algorytmu omawiane są w pracach Freund i Nachtigal [FN1, FN2].

3.3.3 Metoda BiCG

Metoda BiCG (BiConjugate Gradient) wykorzystywana jest do obliczania układów niesymetrycznych. Algorytm metody zaimplementowanej w języku Matlab jest następujący:

```
function x = bicg(A, b, max_eps, max_step)
x = zeros(size(b));
r1 = b; r2 = r1; p1 = r1; p2 = r2;
eps = max_eps+1; step = 0;
while ((eps>max_eps)&(step<max_step))
    ro_1 = r1'*r2;
    if (step>0)
        beta = ro_1/ro_2;
        p1 = r1+beta*p1; p2 = r2+beta*p2;
    end
    q1 = A*p1; q2 = A'*p2;
    alfa = ro_1/(p2'*q1);
    x = x+alfa*p1;
    r1 = r1-alfa*q1; r2 = r2-alfa*q2;
    ro_2 = ro_1;
    eps = max(abs(A*x-b));
```

```
    step = step+1;
end
```

Dla `ro_1=0` metoda zawodzi. Wektor przybliżonych rozwiązań uaktualniany jest za pomocą wektora kierunku *p* i współczynnika kroku α (3.13):

$$x^{(k)} = x^{(k-1)} + \alpha p1^{(k)} \tag{3.13}$$

Uaktualniane tutaj muszą być dwie sekwencje wektorów kierunku *p1* i *p2* oraz ortogonalne do nich wektory residuów *r1* i *r2* (3.14):

$$\begin{aligned} p1^{(k)} &= r1^{(k-1)} + \beta p1^{(k-1)} & p2^{(k)} &= r2^{(k-1)} + \beta p2^{(k-1)} \\ r1^{(k)} &= r1^{(k-1)} - \alpha A p1^{(k)} & r2^{(k)} &= r2^{(k-1)} - \alpha A^T p2^{(k)} \end{aligned} \tag{3.14}$$

Dla macierzy symetrycznych i dodatnio określonych układów metoda sprowadza się do metody CG. Wykonywana jest jednak dwukrotnie wolniej z powodu potrzeby dwukrotnego obliczania iloczynu macierzy przez wektor.

3.3.4 Metoda CGS

CGS (Conjugate Gradient Squared) opisywana jest jako metoda działająca zwykle około dwukrotnie szybciej niż BiCG. Algorytm metody zaimplementowanej w języku Matlab jest następujący:

```
function x = cgs(A, b, max_eps, max_step)
x = zeros(size(b));
r1 = b; r2 = r1; u1 = r1; p = u1;
eps = max_eps+1; step = 0;
while ((eps>max_eps)&(step<max_step))
    ro_1 = r2'*r1;
    if (step>0)
        beta = ro_1/ro_2;
        u1 = r1+beta*q1;
        p = u1+beta*(q1+beta*p);
    end
    v = A*p;
    alfa = ro_1/(r2'*v);
    q1 = u1-alfa*v;
    u2 = u1+q1;
    x = x+alfa*u2;
    q2 = A*u2;
```

```
    r1 = r1-alfa*q2;
    ro_2 = ro_1;
    eps = max(abs(A*x-b));
    step = step+1;
end
```

Metoda nie wymaga mnożenia przez A^T, więc w przypadkach, gdy wykorzystywanie A^T jest niepraktyczne (np. w macierzach rzadkich lub obliczeniach równoległych) metoda CGS wydaje się atrakcyjna. Więcej informacji na temat tej metody można znaleźć w publikacjach Sonneveld [Son] oraz Van der Vorst [Van].

3.3.5 Metoda Bi-CGSTAB

Metoda Bi-CGSTAB (BiConjugate Gradient Stabilized) została wprowadzona, aby rozwiązywać układy niesymetryczne, jednakże eliminując nieregularną zbieżność metody CGS. Algorytm metody w języku Matlab został podany poniżej:

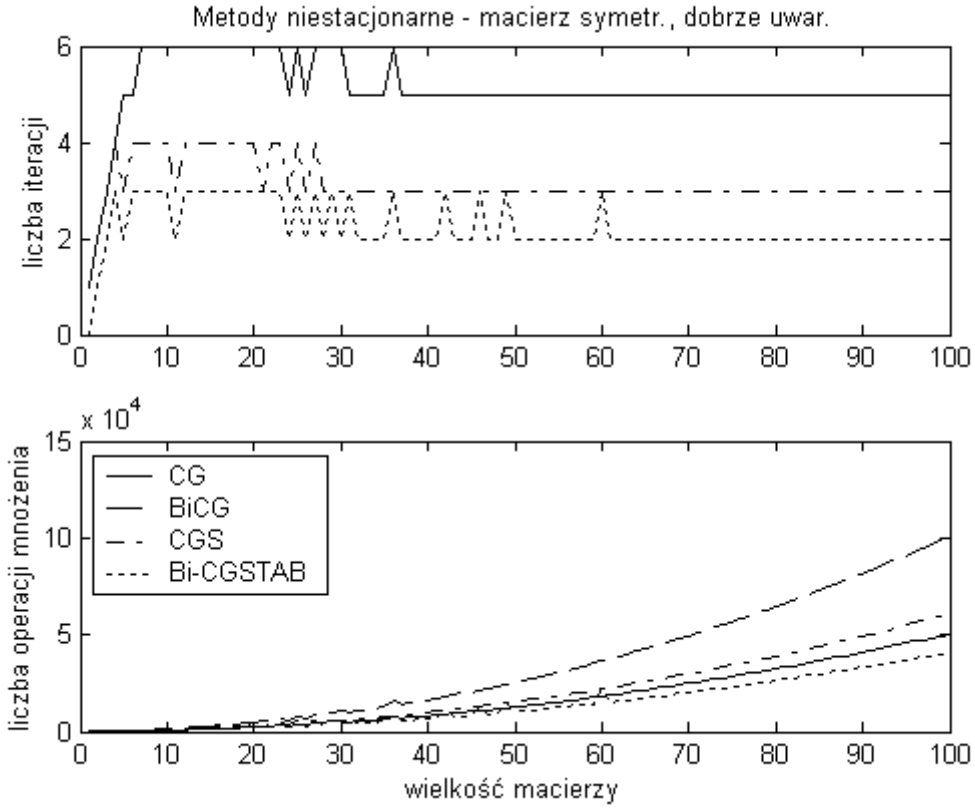
```
function x = bicgstab(A, b, max_eps, max_step)
x = zeros(size(b));
r1 = b; r2 = r1; ro_1 = 1; p = r1;
eps = max_eps+1; step = 0;
while ((eps>max_eps)&(step<max_step))
    ro_1 = r2'*r1;
    if (step>0)
        beta = (ro_1/ro_2)*(alfa/w);
        p = r1+beta*(p-w*v);
    end
    v = A*p;
    alfa = ro_1/(r2'*v);
    s = r1-alfa*v;
    if (norm(s)<max_eps)
        x = x+alfa*p;
        break;
    end;
    l = A*s;
    w = (l'*s)/(l'*l);
    x = x+alfa*p+w*s;
    r1 = s-w*l;
```

```
        ro_2 = ro_1;
        eps = max(abs(A*x-b));
        step = step+1;
    end
```

Jak można zauważyć algorytm posiada dwa wyjścia. Jeżeli wyjście nastąpi poprzez sprawdzenie normy wektora *s* wynik może numerycznie różnić się dokładnością od wyniku gdzie zatrzymanie nastąpiło podczas sprawdzenia drugiego warunku.

3.3.6 Porównanie metod niestacjonarnych

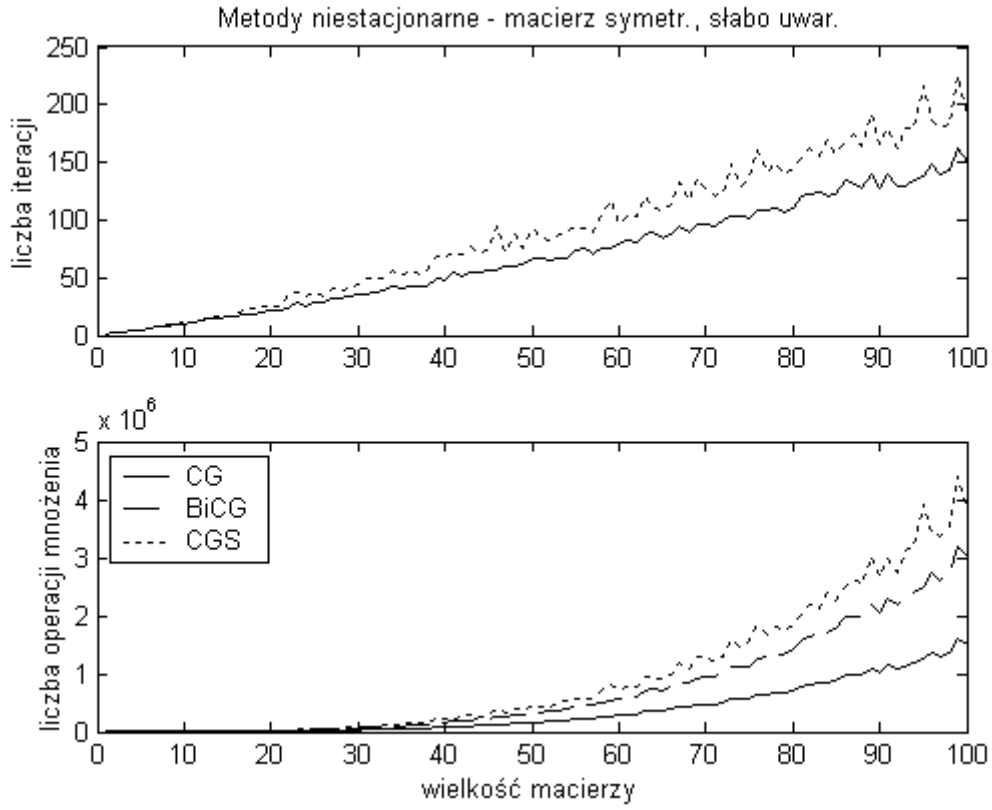
Testy metod, podobnie jak dla metod stacjonarnych, prowadzone były dla macierzy dobrze uwarunkowanych (tzn. wartości elementów na przekątnej były większe niż suma elementów w wierszu.) oraz słabo uwarunkowanych (tzn. wszystkie elementy macierzy były losowe z przedziału *(0, 1)*. Obliczenia były prowadzone dla macierzy o rozmiarze od *1* do *100*. Algorytm zatrzymywał się po osiągnięciu oczekiwanej dokładności wyników $eps<10^{-6}$ lub po przekroczeniu maksymalnej liczby iteracji $max\_step>2*10^{4}$. Wyniki testów dla macierzy symetrycznych i dobrze uwarunkowanych przedstawione są wykresie (rys. 3.7):



Rys. 3.7. Porównanie metod CG, BiCG, CGS i Bi-CGSTAB dla macierzy symetrycznych, dobrze uwarunkowanych.

Najlepsza w tej kategorii okazała się metoda Bi-CGSTAB, mimo że wymaga ona dwukrotnego mnożenia macierzy przez wektor w każdej iteracji. Fakt, że zadanie było zazwyczaj rozwiązywane w dwóch iteracjach, spowodował, że metoda ta okazała się najefektywniejsza. BiCG jest około dwukrotnie mniej efektywna od CG, co wynika z faktu, że dla macierzy symetrycznych BiCG jest tożsama algebraicznie z CG, wymaga jednak dwukrotnego mnożenia macierzy przez wektor w każdej iteracji. CGS jedynie nieznacznie ustąpiło miejsca metodzie CG, mimo dwukrotnej operacji mnożenia macierzy przez wektor.

Dla macierzy słabo uwarunkowanych Bi-CGSTAB okazywała się często rozbieżna, dlatego nie została ona uwzględniona w następnym teście

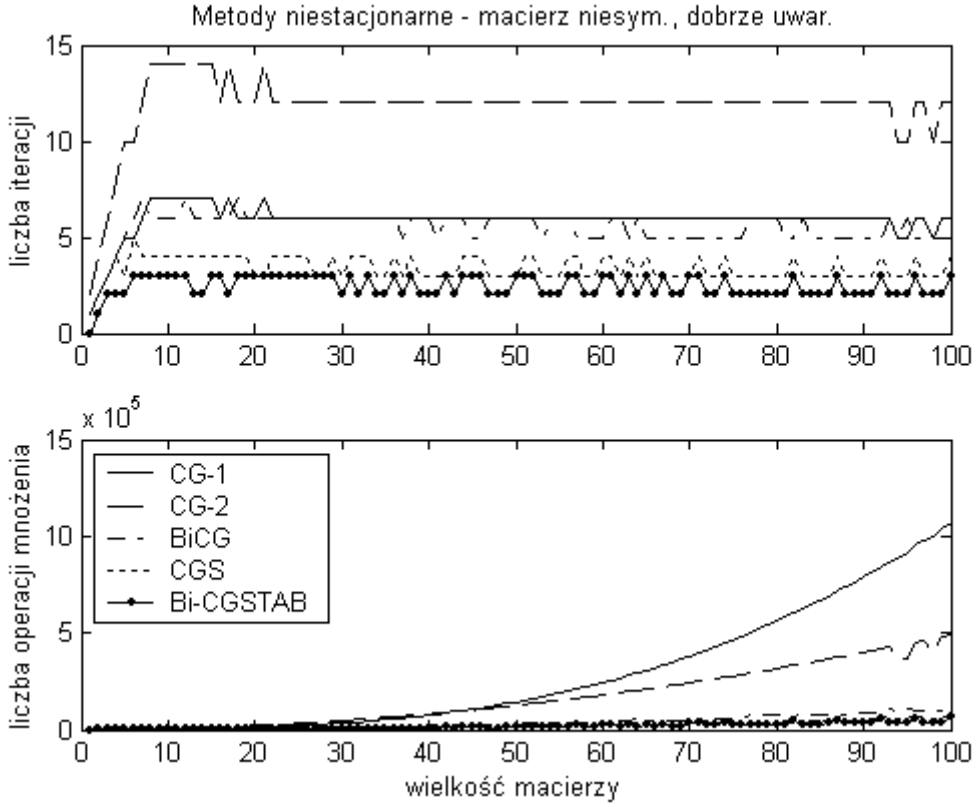


Rys. 3.8. Porównanie metod CG, BiCG i CGS dla macierzy symetrycznych, słabo uwarunkowanych.

Porównanie pozostałych trzech metod (rys. 3.8) wypadło na korzyść CG. BiCG był dwukrotnie wolniejszy z powodu dwukrotnego mnożenia. Metoda CGS okazała się najmniej efektywna dla tego typu zadań.

W następnej klasie zadań nie została uwzględniona metoda CG z powodu niesymetrii badanego układu. Przetestowane zostały następujące metody: CG-1 (Rozwinięcie CG polegające na zastosowaniu zależności $A^TAx=A^Tb$), CG-2 (Rozwinięcie CG polegające na

zastosowaniu zależności $\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$), BiCG, CGS oraz Bi-CGSTAB.

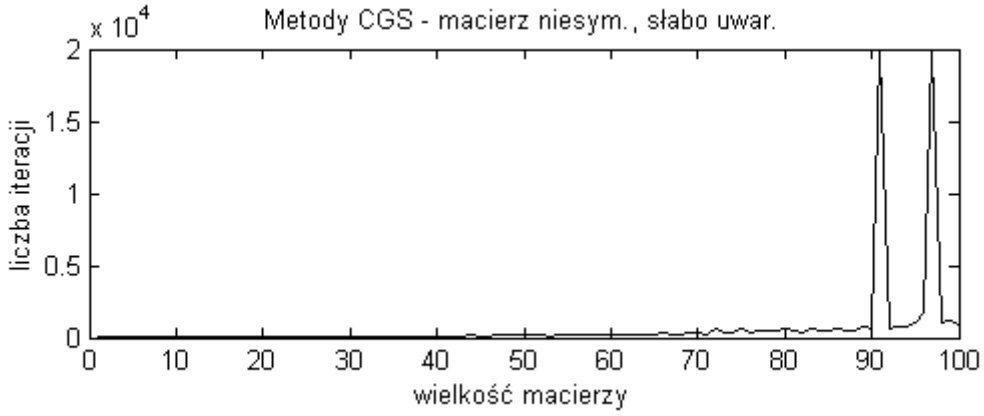


Rys. 3.9. Porównanie metod CG-1, CG-2, BiCG, CGS i Bi-CGSTAB dla macierzy niesymetrycznych, dobrze uwarunkowanych.

W tej klasie zadań (rys. 3.9) najlepsza okazała się Bi-CGSTAB, niewiele jej ustępowała metoda CGS. Należy jednak pamiętać, że algorytm metody Bi-CGSTAB posiada dwa warunki zatrzymania iteracji - w jednym z nich otrzymana dokładność wyników jest wątpliwa. Najmniej efektywna okazała się CG-1 z powodu konieczności wykonania operacji mnożenia macierzy. Metodę CG-2 zastosowano w najprostszej postaci. W przypadku dekompozycji operacji mnożenia macierzy przez wektor (3.15) można algorytm przyspieszyć około dwukrotnie. Jednakże będzie ona nawet w tym przypadku wolniej zbieżna niż BiCG.

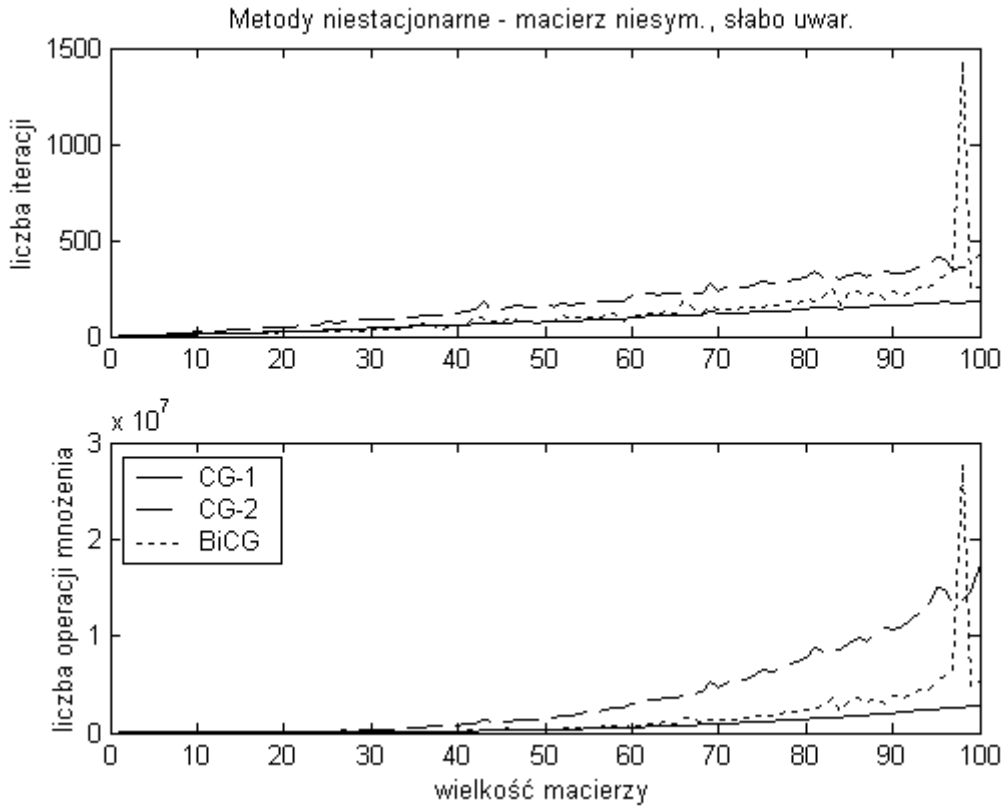
$$\begin{bmatrix} d1 \\ d2 \end{bmatrix} = \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} c1 \\ c2 \end{bmatrix} \Rightarrow \begin{array}{l} d1 = c1 + Ac2 \\ d2 = A^T c1 \end{array} \qquad (3.15)$$

Następną grupę testów przeprowadzono dla macierzy niesymetrycznych i słabo uwarunkowanych. Rysunek 3.10 przedstawia błędne działanie metody CGS, która w niektórych przypadkach przekraczała maksymalną liczbę iteracji algorytmu. Z tego powodu nie została ona uwzględniona (podobnie jak Bi-CGSTAB) w testowanej grupie metod.



Rys. 3.10. Metoda CGS okazała się niestabilna dla macierzy niesymetrycznych, słabo uwarunkowanych.

Przetestowane zostały Metody CG-1, CG-2 oraz BiCG (rys. 3.11).



Rys. 3.11. Porównanie metod CG-1, CG-2 i BiCG dla macierzy niesymetrycznych, słabo uwarunkowanych.

Z powodu dużej liczby iteracji wymaganej do uzyskania zbieżności metod, operacja mnożenia macierzy w metodzie CG-1, która jest wykonywana jednokrotnie, nie odgrywa tak ważnej roli jak w poprzednich testach. W związku z tym metoda ta okazała się najlepsza w tej grupie. CG-2 była najmniej efektywna z powodu podwójnego rozmiaru macierzy, która

musiała być analizowana. Mimo dobrej zbieżności algorytmu BiCG, przejawia on cechy niestabilności dla pewnych danych wejściowych.

3.4 Podsumowanie

Złożoność metod dokładnych można ustalić bez dokonywania testów. Mają one zastosowanie głównie dla układów pełnych, słabo uwarunkowanych, dla których metody iteracyjne okazały się mało efektywne lub nieskuteczne. W układach dobrze uwarunkowanych z metod stacjonarnych warta polecenia jest metoda Gaussa-Seidla z powodu szybkiej zbieżności i braku dodatkowych parametrów uruchomienia algorytmu. W metodach niestacjonarnych wyróżnić można układy symetryczne gdzie najlepsze okazały się Bi-CGSTAB i CG oraz układy niesymetryczne, w których najszybciej zbieżne są Bi-CGSTAB i CGS oraz niewiele im ustępująca metoda BiCG. Metoda Bi_CGSTAB okazała się jedną z najlepszych metod dla tej klasy zadań. Pamiętać jednak należy o wątpliwej dokładności wyników uzyskiwanych tą metodą oraz o tym, że nie udało się tą metodą uzyskać zbieżności dla układów słabo uwarunkowanych. Chociaż metoda CGS okazała się niestabilna w układach niesymetrycznych, słabo uwarunkowanych, to jednak dla układów dobrze uwarunkowanych okazała się jedną z najlepszych. Cechą charakterystyczną metod Bi-CGSTAB oraz CGS jest to, że nie ma potrzeby mnożenia macierzy transponowanej przez wektor. Może to być istotne przy podziale zadania na części w obliczeniach równoległych oraz w macierzach rzadkich, gdzie uzyskanie macierzy transponowanej może stanowić problem.

Każdy problem obliczeniowy może generować macierze o innych cechach charakterystycznych. Aby wybrać odpowiednią metodę obliczeń, należy przetestować ją dla tego typu zadań.

4 Implementacja macierzy rzadkich

4.1 Wybór struktury macierzy

Macierze rzadkie mogą być zapamiętywane na wiele różnych sposobów [MaS]. Popularniejsze z nich to zapis wierszowy, kolumnowy, trójkowy oraz tablice rozproszone. Wybór jednej z nich powinien być dokonany na podstawie operacji, które zamierzamy na nich przeprowadzać.

Zapis wierszowy (kolumnowy) wymaga zapamiętania trzech ciągów:

- kolejne elementy niezerowe czytanych wierszami (kolumnami),
- numery kolumn (wierszy) kolejnych elementów niezerowych,
- ciąg „wskaźników" informujących który niezerowy element jest pierwszym w kolejnym wierszu (kolumnie)[Sad] (rys. 4.1).

W przypadku takiego zapisu liczba zapamiętywanych elementów równa jest $2N+K$ gdzie N to liczba elementów niezerowych a K to liczba wierszy (kolumn) macierzy. W przypadku mnożenia macierzy wygodne jest takie zaprojektowanie algorytmu, aby macierz w zapisie wierszowym mnożona była przez macierz w zapisie kolumnowym. Mnożenie przez wektor wymaga rozpatrywania jedynie elementów niezerowych macierzy, co znacznie obniża złożoność operacji. Słabymi stronami tego zapisu jest złożony sposób wstawiania kolejnego niezerowego elementu, jego usunięcie lub odnalezienie w macierzy.

$$A = \begin{bmatrix} a & b & 0 & 0 & c & 0 \\ 0 & d & 0 & 0 & e & 0 \\ f & 0 & g & 0 & 0 & 0 \\ 0 & h & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & j & 0 \\ 0 & k & l & 0 & 0 & m \end{bmatrix}$$

$$\begin{aligned} elem &= (a,\ b,\ c,\ d,\ e,\ f,\ g,\ h,\ i,\ j,\ k,\ l,\ m), \\ ind &= (1,\ 2,\ 5,\ 2,\ 5,\ 1,\ 3,\ 2,\ 4,\ 5,\ 2,\ 3,\ 6), \\ ptr &= (1,\ 4,\ 6,\ 8,\ 10,\ 11,\ 14). \end{aligned}$$

Rys. 4.1. Zapis przykładowej macierzy rzadkiej w postaci wierszowej.

W zapisie trójkowym zapamiętywane są dla każdego niezerowego elementu jego

współrzędne i wartość. Liczba zapamiętywanych elementów jest więc równa *3N*. Zaletą jest prostota zapisu i łatwość dodawania kolejnych elementów macierzy (jeżeli jesteśmy pewni, że takie elementy jeszcze nie istnieją). Usunięcie elementu wymaga odnalezienia go w liście (w przypadku nieposortowanej listy wymagane jest do *N* sprawdzeń). Mnożenie dwóch macierzy w tym formacie oraz mnożenie macierzy przez wektor jest bardzo złożone obliczeniowo, ponieważ wymaga każdorazowego odnalezienia kolejnego elementu macierzy.

Zapis macierzy rzadkiej jako tablicy rozproszonej polega na upakowaniu elementów niezerowych tak, że współrzędne elementu są kluczem do odnalezienia go w pamięci (tablicy): *ADR=f(i,j)*. Zapis ten wymaga rozwiązywania konfliktów w przypadku, gdy kilka elementów niezerowych według algorytmu powinno być zapisane pod tym samym adresem. Zapis ten wymaga również pewnej nadmiarowości pamięci tak, aby konflikty nie zdarzały się zbyt często. W zapisie tym odnalezienie elementu jest dosyć szybkie, jednakże algorytm mnożenia macierzy wymaga sprawdzenia wszystkich elementów macierzy, nawet zerowych. Powoduje to nieefektywność algorytmów wykonujących operacje arytmetyczne na macierzach w postaci tablic rozproszonych.

Oprócz wymienionych, istnieje wiele innych sposobów zapisu macierzy rzadkich w pamięci. W specjalny sposób opisuje się macierze pasmowe, trójkątne, symetryczne. Czasami w zależności od implementacji warto macierze opisywać w postaci tablic a czasami w postaci list.

4.2 Operacje na macierzach rzadkich

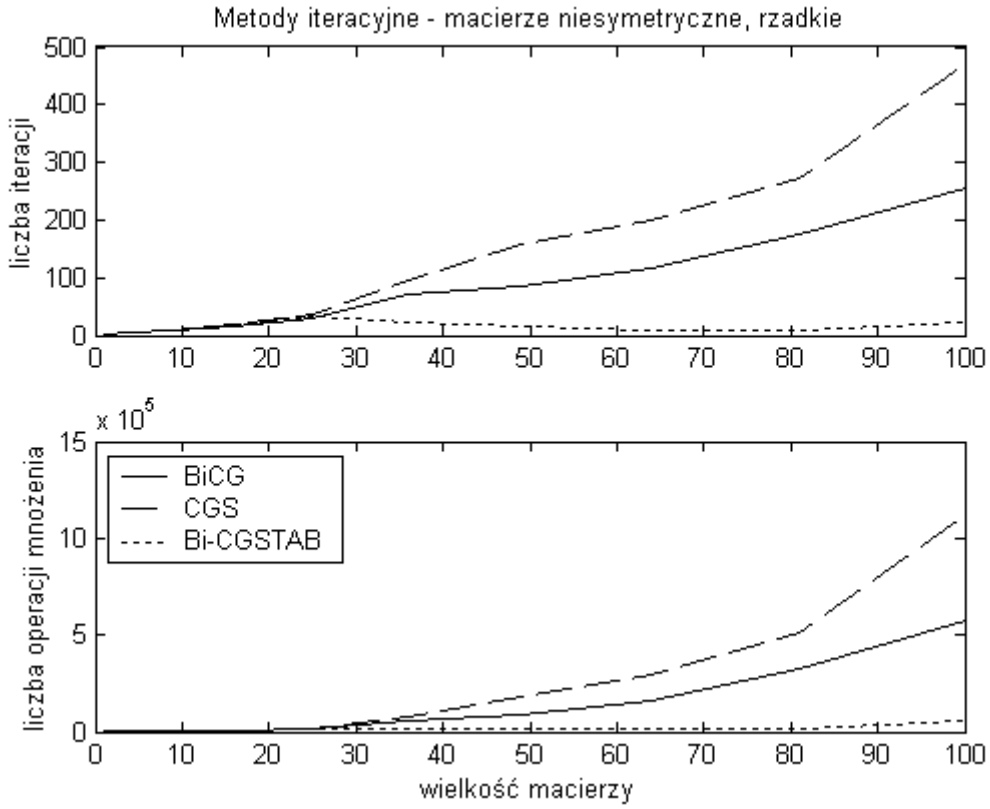
Operacje wykonywane na macierzach rzadkich są specyficzne dla typu zadania, które zamierza się za ich pomocą wykonać. Macierze, które służą jedynie do przechowywania danych powinny umożliwiać proste operacje wstawiania, wyszukiwania i usuwania elementów niezerowych. W przypadku wykonywania operacji równoległych, powinny umożliwić łatwą dekompozycję i przesłanie części macierzy do różnych procesów. Format macierzy może być inny w zależności od tego czy znajduje się w pamięci operacyjnej, czy na dysku. Macierze znajdujące się na dysku nie muszą być optymalizowane pod kątem wyszukiwania elementów i operacji arytmetycznych, ale powinny umożliwiać wczytywanie dużych podmacierzy wprost do pamięci operacyjnej.

Jako przykład można rozważyć operacje na macierzach w metodzie elementów skończonych. W metodzie tej pewien obszar dzielony jest na podobszary. Wygenerowana zostaje w ten sposób siatka zawierająca węzły. Węzły siatki zostają ponumerowane. Węzły, które są sąsiednie w siatce wyznaczają elementy niezerowe macierzy kwadratowej. Jeżeli macierz jest symetryczna, możemy wstawiać do macierzy po dwa symetryczne względem

głównej przekątnej elementy. W przypadku zapisu do macierzy pełnej nie stanowi to problemu, jednak nie każda macierz rzadka umożliwia łatwy dostęp do dowolnego jej elementu. W tym wypadku uzasadnione może być zapamiętanie macierzy w postaci trójkowej. Wygenerowana w ten sposób macierz może zostać umieszczona na dysku. Kolejność trójek reprezentujących węzły siatki może nie być zgodna z kolejnością elementów w macierzy. Poza macierzą rzadką w MES otrzymuje się też kolumnę wyrazów wolnych, która nie jest już jednak macierzą rzadką; zostaje ona zapisana na dysku w pełnej postaci. Tak przygotowane dane powinny być wejściem do aplikacji, która rozwiąże układ równań z macierzą rzadkiej. W przypadku, gdy aplikacja uruchamiana jest w klastrze obliczeniowym, proces odczytujący musi przegrupować dane zapisane na dysku tak, aby odpowiednie elementy macierzy trafiły do właściwych procesów obliczeniowych. Po skompletowaniu wszystkich danych, każdy z procesów musi posortować elementy względem ich położenia w macierzy a następnie dokonać na niej obliczeń. Przed przystąpieniem do rozwiązania zadania format macierzy warto zmienić na inny, który lepiej nadaje się do operacji arytmetycznych – np. format wierszowy. Jeżeli dane na dysku byłyby już przygotowane w odpowiedniej kolejności, niepotrzebne byłyby operacje przegrupowania i sortowania danych. Uprościłoby to znacznie obliczenia, warto więc skonstruować algorytm generacji siatki tak aby otrzymać dane już posortowane względem położenia w macierzy.

4.2.1 Wybór metody rozwiązywania układu równań

Następnym krokiem jest wybór metody rozwiązywania układu równań. Można to uczynić metodą dokładną, (np. metoda Eliminacji Gaussa i metody pochodne) lub jedną z metod iteracyjnych. Metody dokładne nie znalazły dużego zastosowania w rozwiązywaniu układów, w których macierz jest rzadka. Opłaca się je stosować w przypadku, gdy macierz rzadka ma regularną budowę, np. pasmową lub trójkątną. W metodach iteracyjnych najistotniejszymi (z punktu widzenia złożoności obliczeń) operacjami są mnożenie macierzy rzadkiej przez wektor, iloczyn skalarny wektorów oraz mnożenie liczby przez wektor. Zastosowana została wierszowa struktura macierzy, dla której mnożenie macierzy przez wektor jest najprostsze w implementacji. Testy porównawcze przeprowadzone zostały dla macierzy rzadkich o wypełnieniu podobnym do tych, które uzyskiwane są w MES. Wyniki testów przedstawione są na wykresie (rys. 4.2).



Rys. 4.2. Porównanie metod iteracyjnych dla macierzy rzadkich, niesymetrycznych.

Wykorzystana w testach metoda Bi-CGSTAB została zmodyfikowana tak aby mieć pewność uzyskiwania poprawnych wyników tzn. z kodu algorytmu została usunięta możliwość zakończenia działania poprzez sprawdzenie normy wektora *s* (zob. *Rozwiązywanie układów równań liniowych*). Testy wykazały przewagę metody Bi-CGSTAB dla tego typu zadań.

4.3 Redukcja odwołań do pamięci

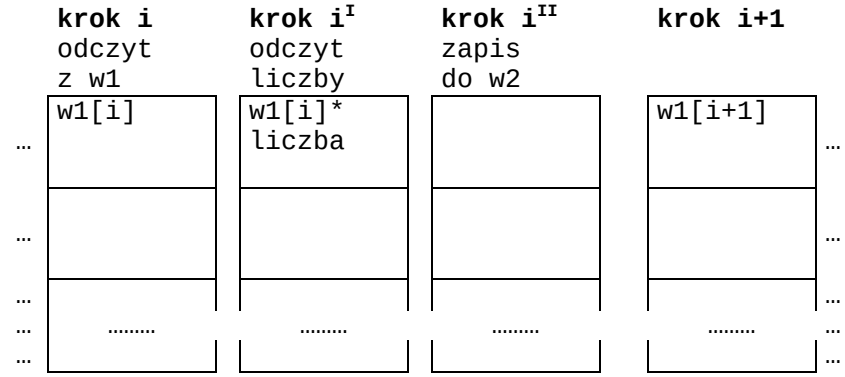
W programach obliczeniowych odwołania do pamięci operacyjnej komputera mogą być „wąskim gardłem". Szybkość procesorów jest stosunkowo wysoka. Warto więc zadbać o to by kluczowe operacje obliczeniowe wykonywane były przy jak najmniejszej liczbie odwołań do pamięci. Kompilatory nie zawsze potrafią zoptymalizować kod programu, aby był efektywny. Potrzebna jest ingerencja na poziomie asemblera. Testy przeprowadzone były w systemie Linux, na klastrze badawczym Politechniki Opolskiej (po 2 procesory Xeon w węźle).

4.3.1 Mnożenie liczby przez wektor

Kod operacji mnożenia liczby przez wektor w języku C wyglądał następująco:

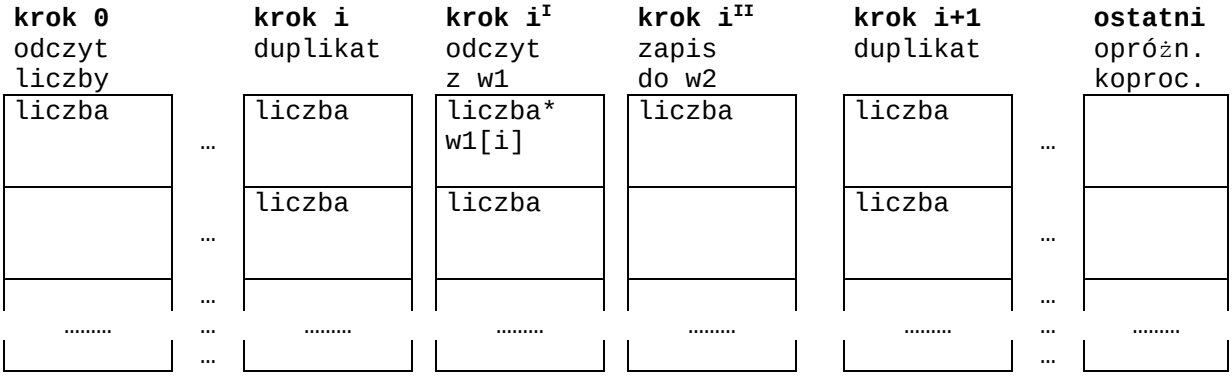
```
for(i=0; i<max; i++)
    w2[i] = liczba*w1[i];
```

Utworzony przez kompilator kod wynikowy dokonuje w każdym kroku dwóch operacji odczytu z pamięci do rejestrów koprocesora oraz jednej operacji zapisu do pamięci (rys. 4.3).



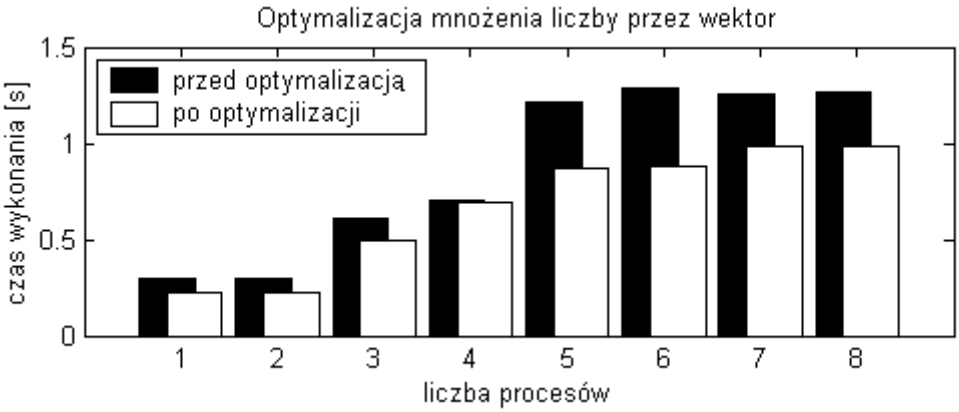
Rys. 4.3. Operacje wykonywane w koprocesorze podczas mnożenia liczby przez wektor przed optymalizacją

Po zmianie kodu na równoważny arytmetycznie, w każdym kroku dokonywana jest jedynie jedna operacja odczytu i jedna zapisu do pamięci. Dodatkowo w kroku zerowym dodano jeden odczyt a w ostatnim opróżnienie koprocesora (rys. 4.4).



Rys. 4.4. Operacje wykonywane w koprocesorze podczas mnożenia liczby przez wektor po optymalizacji

Testy przeprowadzone były dla wektorów o długości 10^4 liczb zmiennoprzecinkowych zapisanych w podwójnej precyzji. Mnożenie przeprowadzone było 10^4 razy. Zastąpienie w każdym kroku jednej operacji odczytu z pamięci przez operację duplikowania rejestru koprocesora spowodowało zmniejszenie czasu wykonywania operacji mnożenia liczby przez wektor (rys. 4.5).



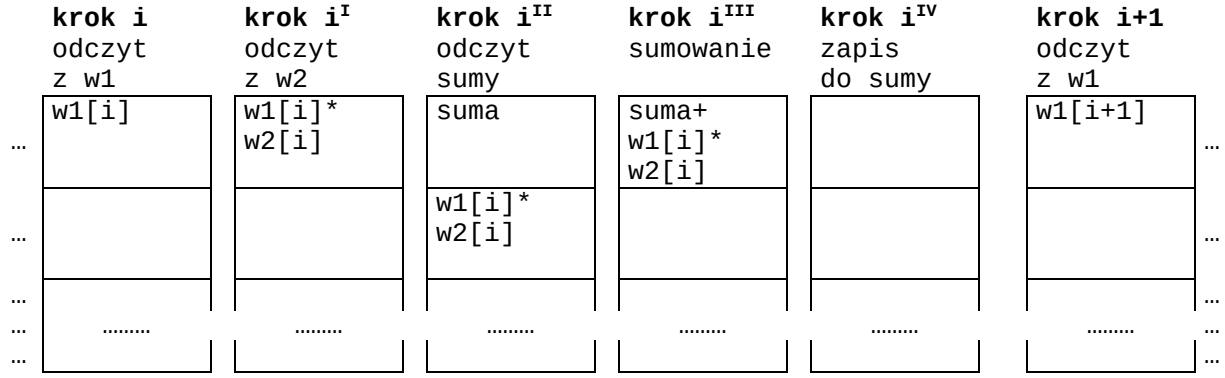
Rys. 4.5. Czas wykonania operacji mnożenia liczby przez wektorów dla różnej liczby procesów uruchamianych w jednym węźle.

4.3.2 Iloczyn skalarny wektorów

Testowany był następujący kod programu:

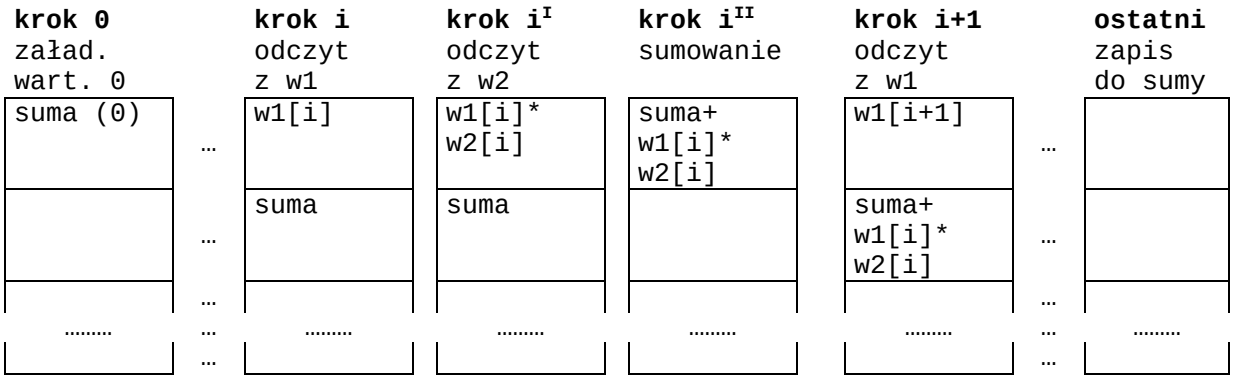
```
suma = 0;
for(i=0; i<max; i++)
    suma+=w1[i]*w2[i];
```

Utworzony przez kompilator kod wynikowy dokonuje w każdym kroku trzech operacji odczytu z pamięci do rejestrów koprocesora oraz jednej operacji zapisu do pamięci (rys. 4.6).



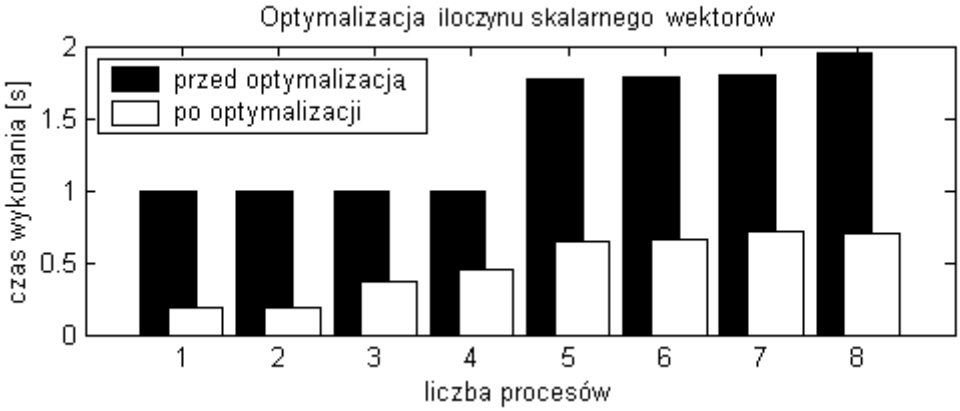
Rys. 4.6. Operacje wykonywane w koprocesorze podczas iloczynu skalarnego wektorów przed optymalizacją

Po zmianie kodu na równoważny arytmetycznie, w każdym kroku dokonywane są jedynie dwie operacje odczytu z pamięci. Dodatkowo w kroku zerowym dodano załadowanie koprocesora wartością *0* a w ostatnim dodano jeden zapis (rys. 4.7).



Rys. 4.7. Operacje wykonywane w koprocesorze podczas iloczynu skalarnego wektorów po optymalizacji

Testy przeprowadzone były dla wektorów długości 10^4 liczb zmiennoprzecinkowych o podwójnej dokładności. Mnożenie przeprowadzone było 10^4 razy. Zmniejszenie liczby operacji związanych z pamięcią spowodowało znaczne zmniejszenie czasu wykonywania operacji iloczynu skalarnego wektorów (rys. 4.8).



Rys. 4.8. Czas wykonania operacji iloczynu skalarnego wektorów dla różnej liczby procesów uruchamianych w jednym węźle.

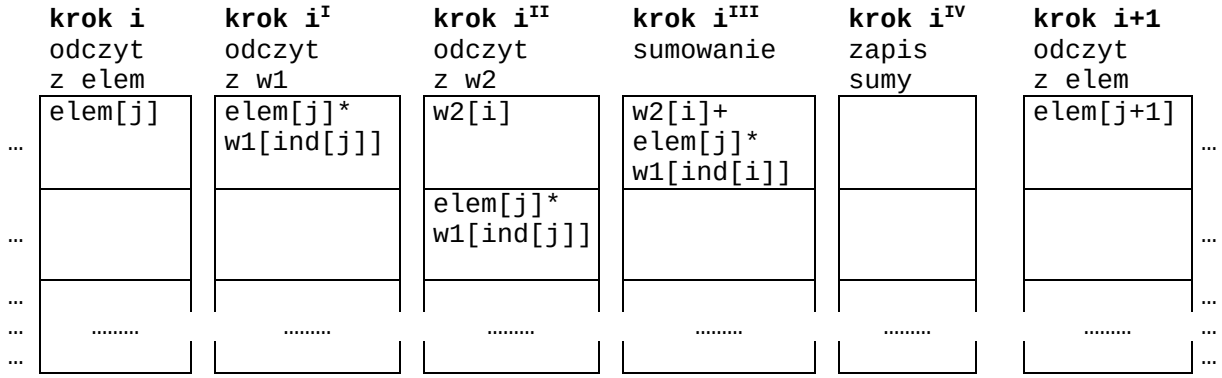
Na rysunku przed optymalizacją można zaobserwować efekty czasowe związane z architekturą dwuprocesorową oraz technologią HyperThread. Po optymalizacji zacierają się efekty HyperThread (nie ma grupowania w czwórki) ale widać że wykonanie jednego procesu i dwu procesów zajmuje tyle samo czasu procesorów. W przypadku więc uruchomienia przynajmniej dwu procesów w jednym węźle, moc obliczeniowa procesorów jest wykorzystywana znacznie efektywniej.

4.3.3 Mnożenie macierzy przez wektor

Przetestowany został następujący kod programu:

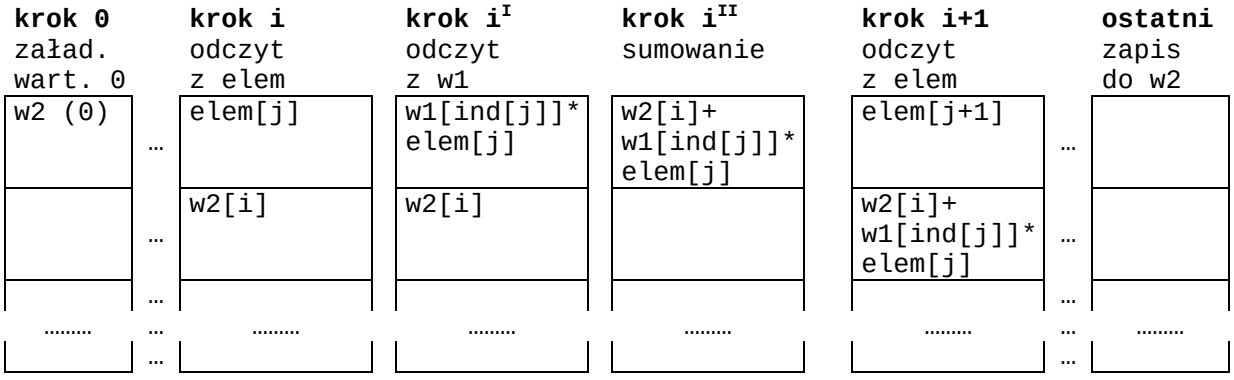
```
for(i=0; i<l_ptr-1; i++){
  w2[i]=0;
  for(j=ptr[i]; j<ptr[i+1]; j++)
       w2[i]+=elem[j]*w1[ind[j]]
}
```

Utworzony przez kompilator kod wynikowy dokonuje w każdym kroku trzech operacji odczytu z pamięci do rejestrów koprocesora oraz jednej operacji zapisu do pamięci (rys. 4.9).



Rys. 4.9. Operacje wykonywane w koprocesorze podczas mnożenia wektorów przed optymalizacją

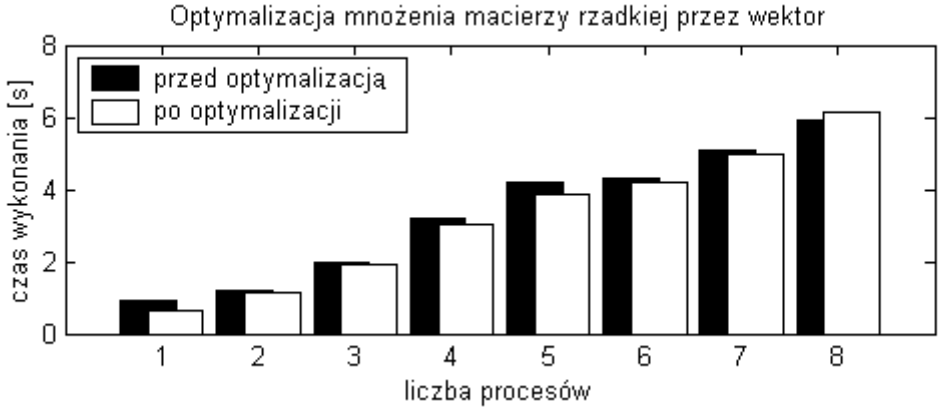
Po zmianie kodu na równoważny arytmetycznie, w każdym kroku dokonywane są jedynie dwie operacje odczytu z pamięci. Dodatkowo w kroku zerowym dodano załadowanie koprocesora wartością *0* a w ostatnim dodano jeden zapis (rys 4.10).



Rys. 4.10. Operacje wykonywane w koprocesorze podczas mnożenia macierzy rzadkiej przez wektor po optymalizacji.

Testy przeprowadzone były dla siatki elementów MES wielkości *100x100* co spowodowało utworzenie macierzy rzadkiej wielkości $10^4 x 10^4$. Mnożenie przeprowadzone było 10^3 razy. Zmniejszenie liczby operacji związanych z pamięcią spowodowało nieznaczne

zmniejszenie czasu wykonywania operacji mnożenia macierzy przez wektor (rys. 4.11).

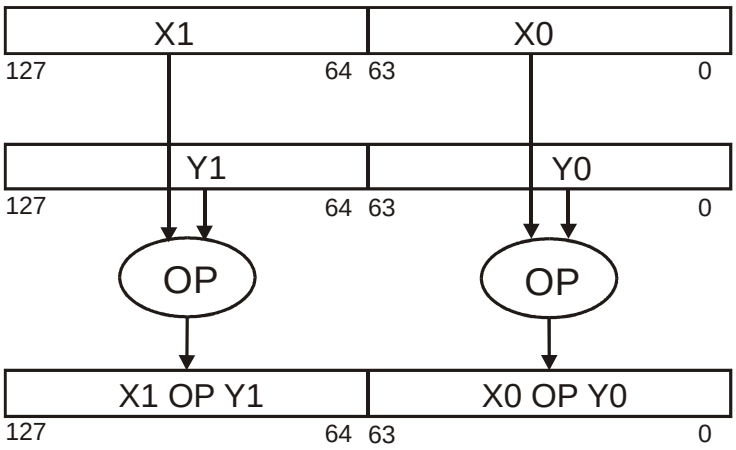


Rys. 4.11. Czas wykonania operacji mnożenia macierzy rzadkiej przez wektor dla różnej liczby procesów uruchamianych w jednym węźle.

Optymalizacja operacji mnożenia macierzy rzadkiej przez wektor przebiega podobnie jak w przypadku iloczynu skalarnego wektorów, gdzie kolejne wiersze macierzy można traktować jako osobne wektory. Stosunkowo mały zysk optymalizacji mnożenia macierzy rzadkiej przez wektor można tłumaczyć tym, że zysk z optymalizacji uzyskiwany jest przy dłuższych wektorach, natomiast w macierzach rzadkich otrzymywanych w MES każdy wiersz ma zaledwie kilka elementów niezerowych. Poza tym duża część obliczeń procesora podczas wykonywania algorytmu koncentruje się nie na obliczeniach właściwych tzn. mnożenia i sumowania liczb, ale na obliczaniu adresów tych liczb w tablicach.

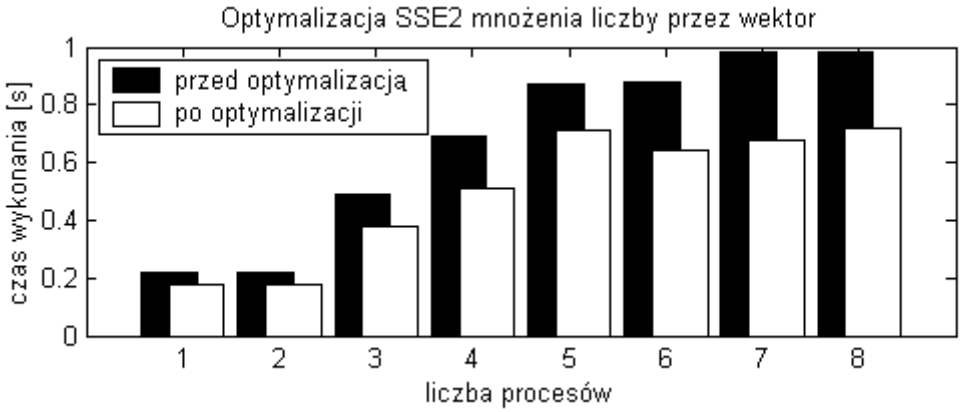
4.4 Wykorzystanie techniki SSE2

SSE2 to rozszerzenie MMX umożliwiające dokonywanie operacji SIMD (Single Instruction Multiple Data) na liczbach zmiennoprzecinkowych o podwójnej precyzji. Rejestry XMM o długości *128* bitów umożliwiają za pomocą jednego rozkazu załadowanie dwóch liczb *64*-bitowych a następnie wykonanie na nich jednoczesnych operacji arytmetycznych (rys. 4.12). Wykorzystanie techniki SSE2 może do dwóch razy zwiększyć szybkość algorytmów obliczeniowych dokonujących operacji na liczbach zmiennoprzecinkowych o podwójnej precyzji.



Rys. 4.12. Wykorzystanie operacji SIMD dla liczb zmiennoprzecinkowych o podwójnej precyzji.

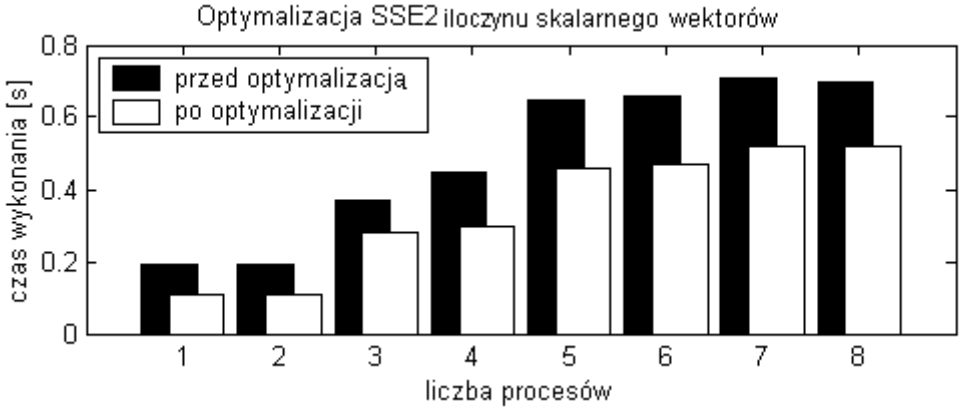
4.4.1 Mnożenie liczby przez wektor



Rys. 4.13. Zastosowanie operacji SIMD dla operacji mnożenia liczby przez wektor.

Zysk czasowy uzyskany dzięki operacjom SIMD w algorytmie mnożenia liczby przez wektor jest rzędu kilkunastu procent (rys. 4.13).

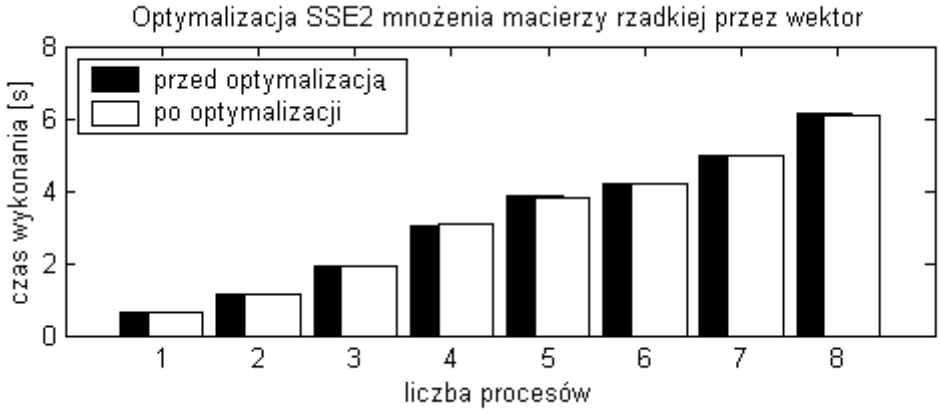
4.4.2 Iloczyn skalarny wektorów



Rys. 4.14. Zastosowanie operacji SIMD dla operacji iloczynu skalarnego wektorów.

Wykorzystanie operacji SIMD podczas iloczynu skalarnego wektorów dało zysk do *40%* (rys. 4.14).

4.4.3 Mnożenie macierzy przez wektor



Rys. 4.15. Zastosowanie operacji SIMD dla operacji mnożenia macierzy rzadkiej przez wektor.

W algorytmie mnożenia macierzy rzadkiej przez wektor nie uzyskano oczekiwanego przyspieszenia (rys. 4.15). Wynik ten można tłumaczyć podobnie jak w przypadku optymalizacji odwołań do pamięci: krótkie optymalizowane ciągi liczb oraz złożone odwoływanie się do elementów tablic.

4.5 Optymalizacja całych wyrażeń

Optymalizacja kodu na poziomie każdego działania z osobna może się okazać o wiele mniej efektywne niż optymalizacja całego wyrażenia. Zmienne tymczasowe, będące wynikiem jednego z działań a danymi dla drugiego, nie muszą być zapisywane do pamięci operacyjnej, ale mogą być zatrzymane w rejestrach do ponownego wykorzystania. W

algorytmie CG trzykrotnie w iteracji wykonywane jest wyrażenie typu (zapis w języku Matlab):

```
w1 = w2+wsp*w1;
```

lub

```
w1 = w1+wsp*w2;
```

gdzie:

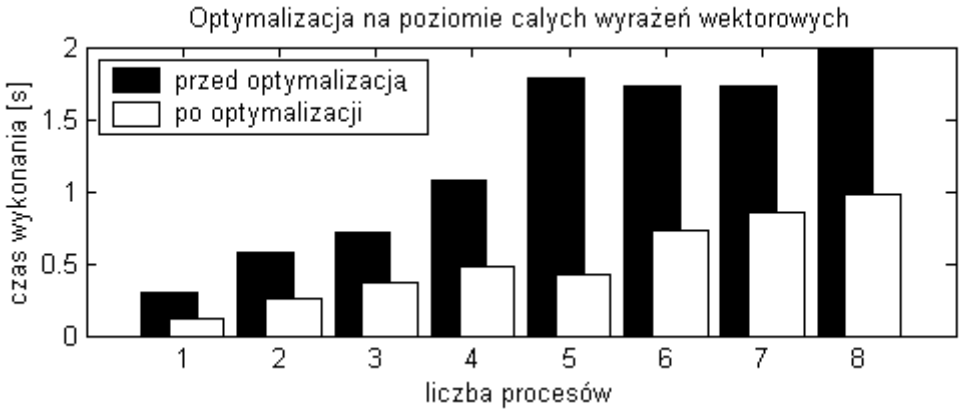
`w1`, `w2` - wektory,

`wsp` - wartość skalarna.

W języku C zapis pierwszego wyrażenia wygląda następująco:

```
for(i=0; i<max; i++)
    w1[i]=w2[i]+wsp*w1[i];
```

Optymalizując kod na poziomie całego wyrażenia otrzymano dodatkowe skrócenie czasu wykonywania zadania. Na rysunku 4.16 porównano czas wykonania działań na wektorach zawierających 10^7 liczb zmiennoprzecinkowych o podwójnej precyzji. Pierwsze pomiary dotyczą czasów otrzymanych po redukcji odwołań do pamięci i wykorzystaniu SSE2. Drugie pomiary otrzymano po dodatkowej optymalizacji całych wyrażeń.

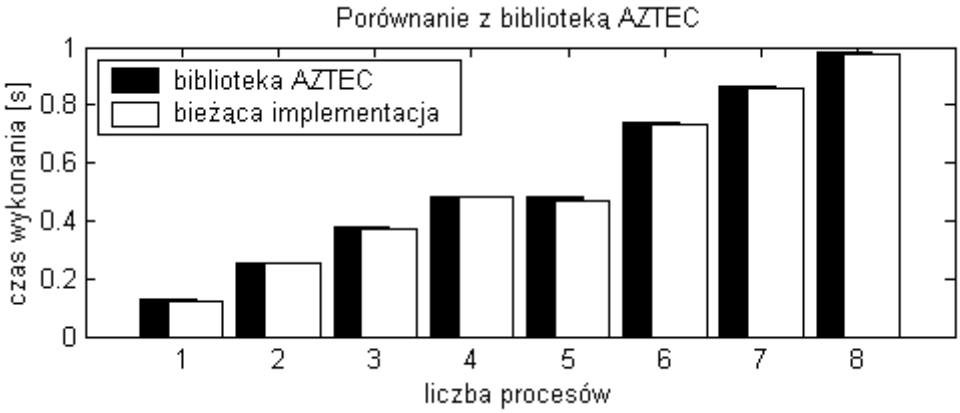


Rys. 4.16. Dodatkowe skrócenie czasu wykonywania kodu poprzez zastosowanie optymalizacji całych wyrażeń wektorowych.

4.6 Porównanie wydajności algorytmu z inną implementacją

Po dokonaniu wszystkich wymienionych optymalizacji, czas wykonania powyższego wyrażenia został porównany z czasem wykonywania równoważnego kodu zaimplementowanego w bibliotece AZTEC. Biblioteka AZTEC została zaprojektowana przez korporację *Sandia* na zlecenie rządu Stanów Zjednoczonych. Biblioteka zawiera szereg

funkcji w języku C i Fortran służących do wykonywania operacji na macierzach rzadkich przy wykorzystaniu klastrów obliczeniowych. Po dokonaniu wszystkich optymalizacji udało się uzyskać ok. *0,55%* przyspieszenia w stosunku do biblioteki AZTEC (rys 4.17).



Rys. 4.17. Porównanie optymalizacji wprowadzonych do bieżącej implementacji z czasami otrzymanymi przy użyciu biblioteki AZTEC.

4.7 Podsumowanie

Macierze rzadkie są specyficzne zarówno z punktu widzenia struktury jak i możliwości optymalizacji wykonywanych na nich operacji. Wyniki obliczeń uzyskane w jednym węźle stanowią podstawę do prawidłowego rozdziału zadań w klastrze obliczeniowym.

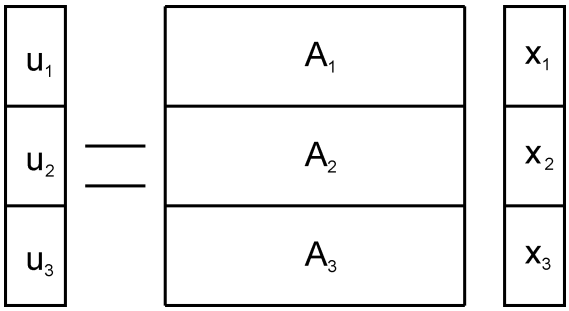
5 Rozwiązywanie układów równań liniowych w klastrze

Analizowane w tym rozdziale macierze, to macierze rzadkie o rozkładzie elementów niezerowych podobnym jak w MES. Macierze są dobrze uwarunkowane tzn. współczynniki na przekątnej są większe niż suma elementów w wierszu. Zaimplementowana metoda rozwiązywania to Bi-CGSTAB ze względu na bardzo szybką zbieżność dla układów o takiej strukturze.

5.1 Równoległe implementacje metod

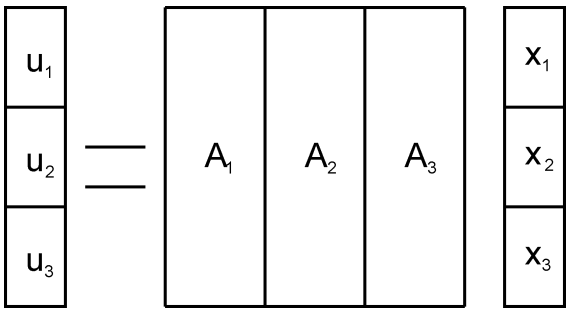
5.1.1 Dekompozycja macierzy

Aby zaimplementować metodę równoległą, należy się zastanowić nad sposobem zdekomponowania macierzy rzadkiej. W zależności od wybranej metody zmieni się też sposób równoległego wykonania operacji mnożenia macierzy przez wektor. W dekompozycji wierszowej (rys. 5.1) podczas operacji mnożenia macierzy przez wektor należy najpierw skompletować całość wektora w każdym procesie, a następnie wykonać obliczenie iloczynu lokalnej części macierzy przez wektor.



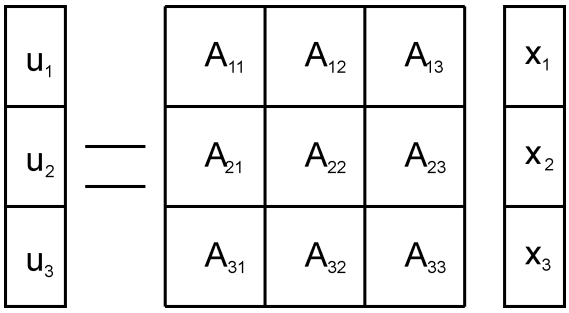
Rys. 5.1. Dekompozycja wierszowa macierzy.

Dekompozycja kolumnowa (rys. 5.2) umożliwia bez transmisji wykonanie iloczynu lokalnej części macierzy z lokalną częścią wektora i uzyskanie wszystkich, ale częściowych wyników mnożenia. Uzyskanie całkowitych wyników możliwe jest po dokonaniu redukcji (sumowania sieciowego) wyników uzyskanych przez każdy z procesów.



Rys. 5.2. Dekompozycja kolumnowa macierzy.

Dekompozycja blokowa (rys. 5.3) jest podobna do dekompozycji wierszowej, ale pamiętane są tylko niezerowe bloki. Elementy w blokach mogą być pamiętane w postaci wierszowej, kolumnowej, trójkowej i in.



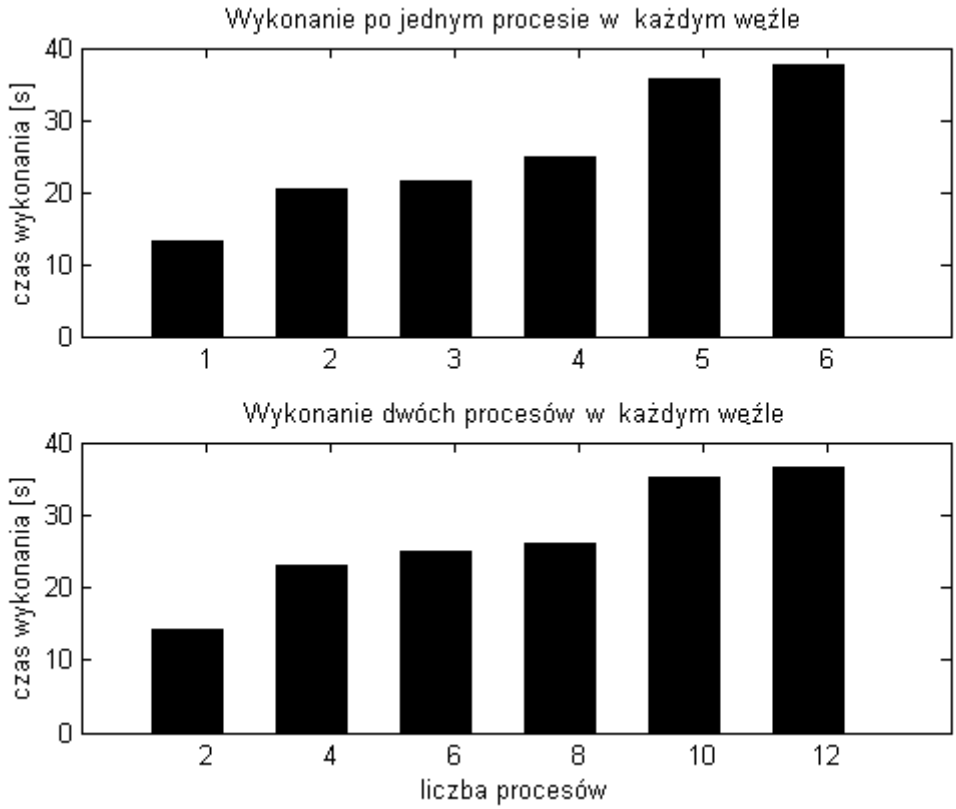
Rys. 5.3. Dekompozycja blokowa macierzy.

5.1.2 Implementacja Bi-CGSTAB

Metoda Bi-CGSTAB wymaga wykonania w każdym kroku trzech operacji mnożenia macierzy przez wektor, trzech operacji iloczynu skalarnego wektorów, sześciokrotnego mnożenia wektora przez liczbę oraz siedmiokrotnego wykonania dodawania lub odejmowania wektorów. W równoległej implementacji metody (rys. 5.2) potrzebne są również operacje komunikacyjne: sumowania i znajdowanie maksimum liczb z różnych procesów, oraz kompletowanie wektorów z danymi z innych procesów. O ile sumowanie i znajdowanie maksimum dotyczy liczb skalarnych i nie wnosi zbytniego obciążenia komunikacyjnego, o tyle kompletowanie wektorów wymaga przesyłania dużej liczby danych i może spowodować, że równoległe wykonanie algorytmu stanie się nieopłacalne.

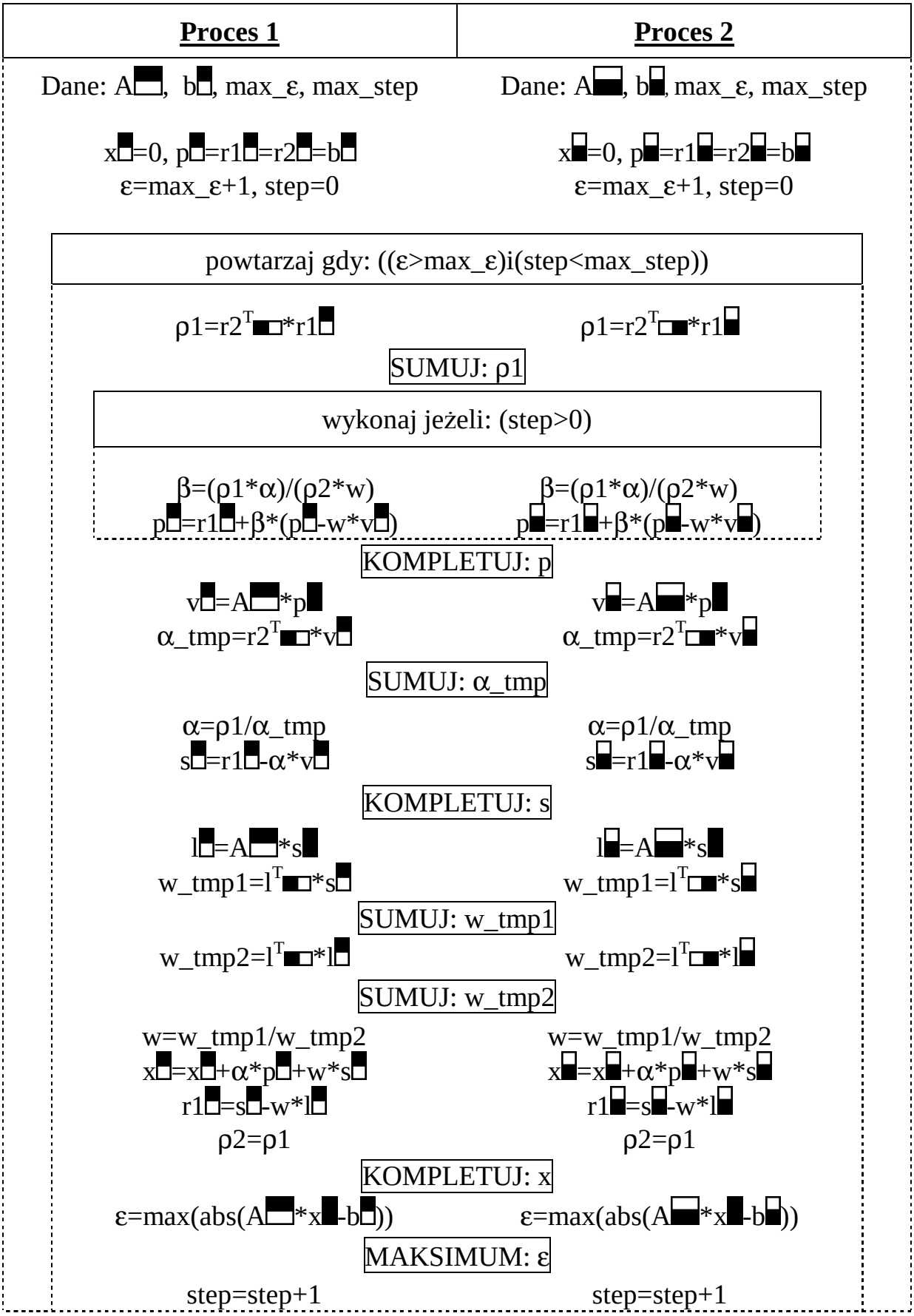
Algorytm wykonany sekwencyjnie w jednym węźle klastra obliczył układ równań z $4*10^6$ niewiadomymi w *17,51* sekundy. Wprowadzenie optymalizacji związanych z redukcją odwołań do pamięci i operacjami wektorowymi zmniejszyło ten czas do *13,21* sekundy, czyli uzyskano ok. *22,8%* przyspieszenie. Chociaż wyniki otrzymane w jednym węźle klastra wydają się obiecujące, to wykonanie algorytmu równolegle nie dało oczekiwanego przyspieszenia (rys. 5.4). Podział zadania na części dał w rezultacie zwiększenie czasu

obliczeń. Spowodowane jest to dużym czasowym nakładem komunikacji międzyprocesowej.



Rys. 5.4. Czas równoległego rozwiązywania rzadkiego układu równań metodą Bi-CGSTAB.

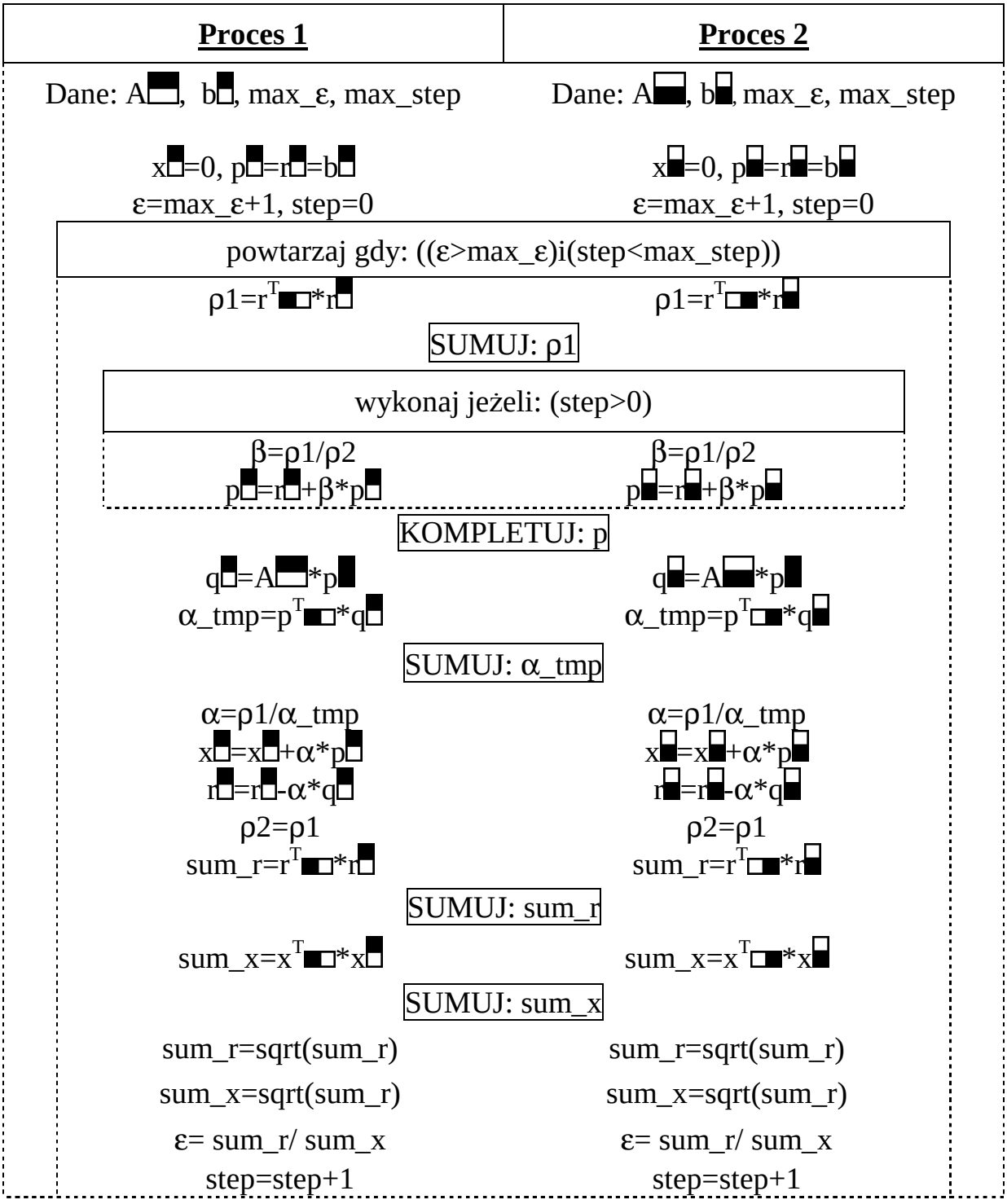
Bazując na istniejącym sprzęcie i oprogramowaniu, efektywne wykorzystanie całego klastra możliwe jest tylko poprzez zminimalizowanie wymagań komunikacyjnych algorytmu.



Rys. 5.5. Algorytm równoległej implementacji metody Bi-CGSTAB.

5.1.3 Implementacja CG

Rozwiązując zagadnienia statyczne otrzymuje się układ równań z symetryczną macierzą, który można rozwiązać metodą CG. Po dokonaniu uproszczenia w warunku zakończenia iteracji (patrz listing funkcji CG w rozdziale 8) algorytm wymaga ona wykonania jednej operacji mnożenia macierzy przez wektor, czterech operacji iloczynu skalarnego wektorów, trzykrotnego mnożenia wektora przez liczbę oraz trzykrotnego wykonania dodawania lub odejmowania wektorów w każdym kroku iteracji.

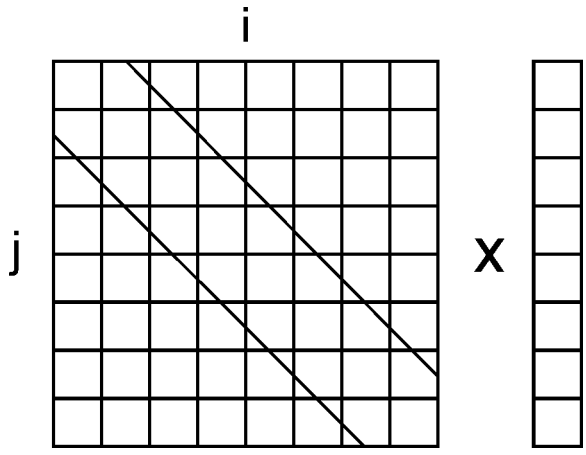


Rys. 5.6. Algorytm równoległej implementacji metody CG.

Na rysunku 5.6 przedstawiającym równoległy algorytm metody CG można zaobserwować, że w pojedynczej iteracji algorytmu niezbędne jest wykonanie jednej operacji kompletowania wektorów oraz czterech operacji redukcji (sieciowego sumowania).

5.2 Optymalizacja równoległej implementacji metod

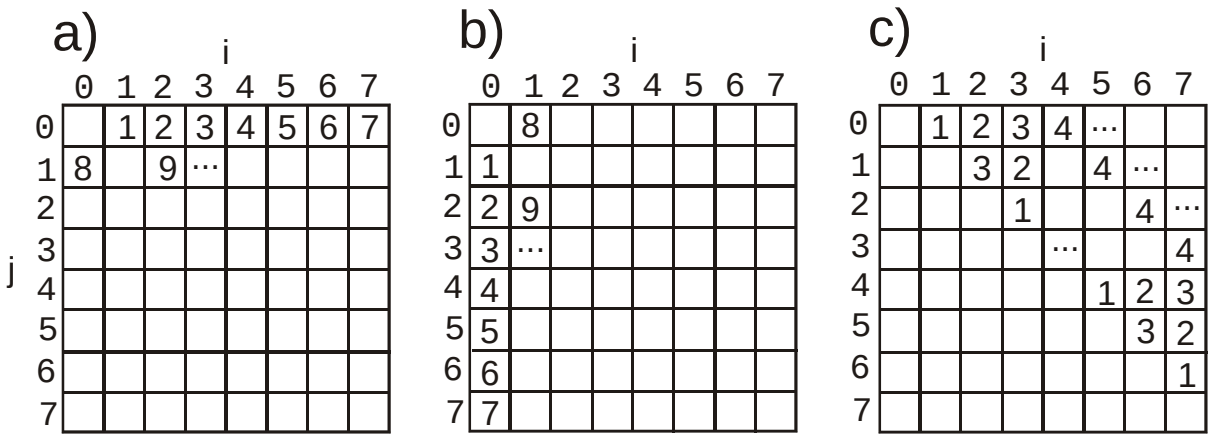
Przyglądając się dokładniej operacji równoległego mnożenia macierzy rzadkiej przez wektor można dojść do wniosku, że żaden z procesów nie potrzebuje do tego celu całego kompletowanego wektora. Dla dużych zadań elementy niezerowe w macierzy zajmują jedynie stosunkowo wąskie pasmo wzdłuż przekątnej. Elementy wektora, które będą mnożone jedynie przez elementy zerowe macierzy nie wnoszą żadnego wkładu w obliczenia wykonywane w danym procesie. Na rysunku 5.7 zaznaczono pasmo elementów niezerowych macierzy. Indeksem *j* oznaczono poziome części macierzy, dla których obliczenia będą wykonywane w pojedynczym procesie. Indeksem *i* oznaczono pionowe części macierzy, dla których podczas mnożenia potrzebne są elementy wektora z procesu *i*. Dla części macierzy o indeksach (1, 1), (2, 2)… elementy wektora już znajdują się w bieżącym procesie, natomiast dla części macierzy oddalonych od przekątnej nie ma potrzeby przesyłania elementów wektora.



Rys. 5.7. Organizacja danych podczas równoległego mnożenia macierzy rzadkiej przez wektor.

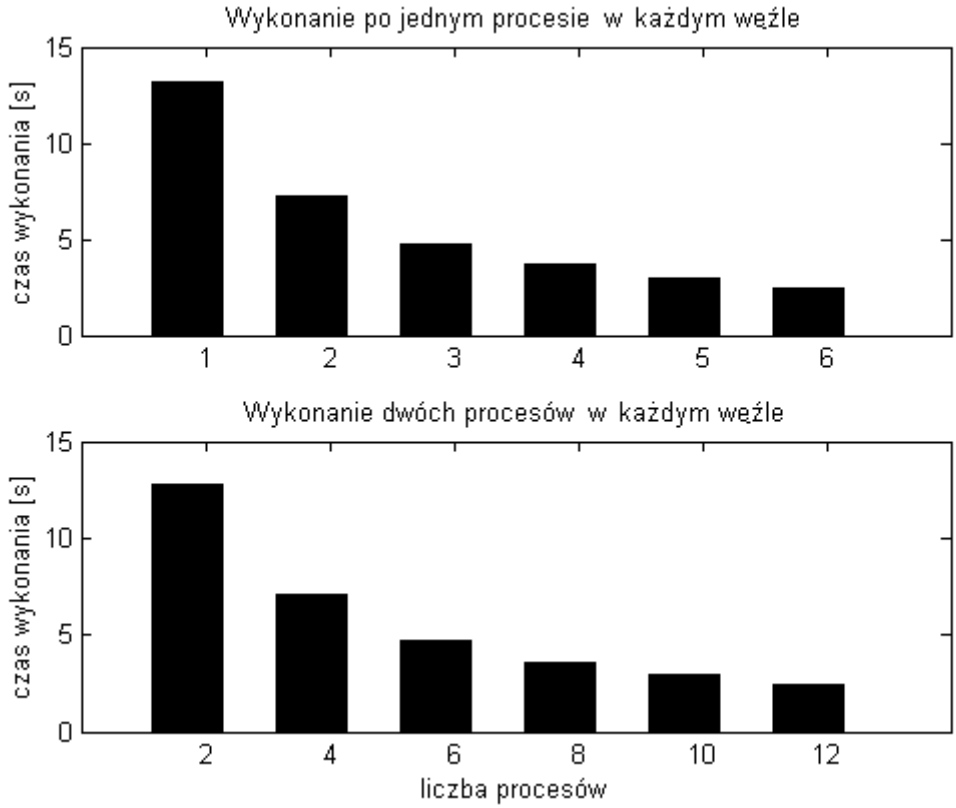
Przesyłanie jedynie istotnych elementów wektora z procesu *i* do procesu *j* wymaga wcześniejszego, jednokrotnego przesłania tablicy z numerami potrzebnych elementów z procesu *j* do procesu *i*. Powoduje to dodatkową transmisję. Poza tym przesyłane elementy będą musiały być pakowane w procesie *i* i rozpakowywane w procesie *j*, co powoduje dodatkowe obciążenie procesora.

Innym sposobem zmniejszenia nakładu czasu potrzebnego na komunikację jest odpowiednia kolejność transmisji. Jeżeli program jest wykonywany przez *8* procesów, każdy proces musi uzyskać brakujące *7* części wektora od pozostałych procesów. W sumie w systemie będzie musiało być wykonanych do *56* transmisji (w przypadku macierzy pełnej). W macierzy rzadkiej liczba ta będzie mniejsza ze względu na przesyłanie jedynie elementów istotnych. Organizując przesył kolejnymi wierszami (rys. 5.8a) kompletujemy na początku cały wektor dla procesu *0*, pozostałe procesy oczekują kolejno na dostęp do procesu *0* aby przekazać dane. Następnie kompletowany jest wektor w procesie *1*, pozostałe procesy czekają na dostęp do tego procesu. Jeżeli zorganizujemy transmisję kolejno kolumnami (rys. 5.8b), proces *0* wysyła najpierw dane do wszystkich procesów, które oczekują kolejno na potrzebną paczkę informacji z procesu *0*. Tą samą czynność wykonuje następnie proces *1* itd. Jak widać w każdym z tych sposobów w kolejnych fazach transmisji, jeden z procesów jest przeciążony, natomiast reszta - prawie bezczynna. Aby zmienić tą sytuację, można zorganizować transmisję w fazach (rys. 5.8c), gdzie kilka jednoczesnych transmisji nie koliduje ze sobą (problem może pojawić się na przełącznicy, ale ma ona zazwyczaj wielokrotnie większą przepustowość niż pojedyncza karta sieciowa).



Rys. 5.8. Trzy warianty podziału operacji kompletowania wektorów na fazy transmisyjne.

Po opisanych zabiegach dokonano ponownego podziału algorytmu na procesy. Tym razem zrównoleglenie dało pożądane skutki (rys 5.9) a nakład komunikacyjny okazał się na tyle niski, że możliwe było dla sześciu procesów przeszło pięciokrotne zmniejszenie czasu obliczeń.



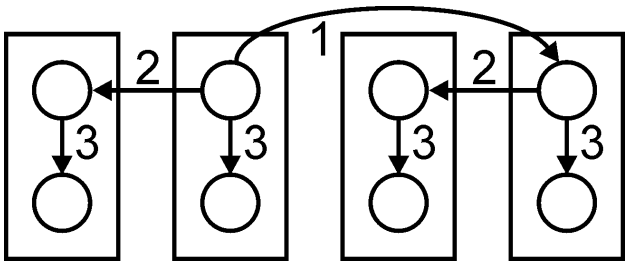
Rys. 5.9. Czas równoległego rozwiązywania rzadkiego układu równań metodą Bi-CGSTAB po optymalizacji kompletowania wektorów.

6 Efektywna komunikacja w klastrze

Połączenia komunikacyjne w klastrze można rozumieć jako fizyczne (pomiędzy procesorami w węźle, pomiędzy węzłami w sieci, pomiędzy sieciami) i logiczne (pomiędzy procesami, pomiędzy wątkami w procesach). Koszt (czas) przesłania wiadomości pomiędzy dwoma węzłami zależny jest od ich fizycznego położenia w klastrze. Połączenia logiczne informują algorytm o tym, które zadania powinny się komunikować ze sobą. Odpowiednie zaprojektowanie połączeń logicznych w zależności od istniejących połączeń fizycznych umożliwia implementację efektywnych algorytmów obliczeniowych, podczas gdy inna struktura połączeń może się okazać bardzo niekorzystna.

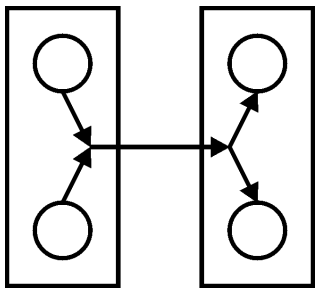
6.1 Rozwiązania komunikacyjne

W zależności od rodzaju algorytmu, struktury klastra i zainstalowanego oprogramowania możliwe są różne metody optymalizacji czasu komunikacji międzyprocesowej. Dla klastrów o węzłach wieloprocesorowych, często korzystne jest uruchamianie wielu procesów w jednym węźle. Rozgłaszanie komunikatów powinno być tak zaprojektowane, aby najpierw wiadomość dotarła do każdego węzła klastra a następnie była rozgłaszana pomiędzy procesami znajdującymi się w każdym, pojedynczym węźle (rys. 6.1).



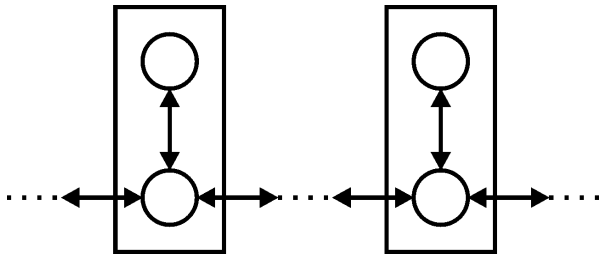
Rys. 6.1. Projektowanie ścieżki rozgłoszeń w klastrach o węzłach wieloprocesorowych.

Klasycznie do budowania modeli komunikacyjnych przyjmuje się, że przesłanie komunikatu pomiędzy procesami charakteryzują dwa czasy: inicjalizacja przesyłu i czas transmisji jednej danej. Inicjalizacja przesyłu może być wielokrotnie bardziej czasochłonna niż sama transmisja, dlatego warto budować algorytmy tak, aby komunikaty, które są jednocześnie przesyłane przez różne procesy znajdujące się w tym samym węźle klastra były w miarę możliwości grupowane (rys. 6.2).



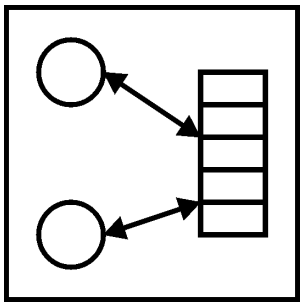
Rys. 6.2. Grupowanie komunikatów w transmisji międzywęzłowej.

Często poruszanym zagadnieniem jest też nakładkowanie obliczeń i komunikacji (overlapping). Wyodrębniane są w każdym węźle procesy (lub wątki) których zadaniem jest jedynie komunikacja międzywęzłowa. Niezależnie od nich wykonywane są procesy zajmujące się obliczeniami, które inicjują zadania w procesach komunikacyjnych oraz korzystają z uzyskanych w ten sposób danych (rys. 6.3).



Rys. 6.3. Nakładkowanie obliczeń i komunikacji.

W wielu przypadkach korzystne może się okazać wyodrębnienie pamięci wspólnej dla procesów znajdujących się w tym samym węźle klastra. Proces nadawczy nie musi wtedy oczekiwać na gotowość procesu odbiorczego, nieważna jest też kolejność odbioru wiadomości przez procesy podczas rozgłoszenia. Zmniejsza się też ilość zajętej pamięci gdyż nie jest wymagane alokowanie pamięci dla tych samych danych przez każdy proces osobno (rys. 6.4).



Rys. 6.4. Wykorzystanie pamięci wspólnej procesów.

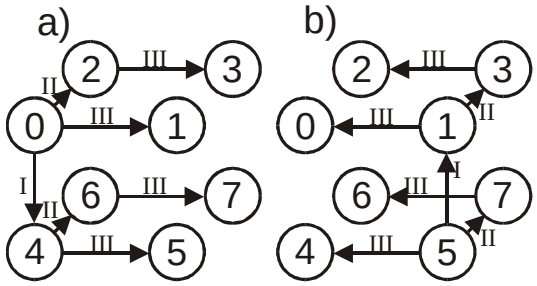
Oczywiście nie każde z tych rozwiązań będzie możliwe dla dowolnego algorytmu i nie

każde z nich da oczekiwane przyspieszenie dla dowolnej konfiguracji sprzętu i oprogramowania.

6.2 Rola rozgłoszeń

Klastrowe algorytmy obliczeniowe często składają się z kolejnych faz, gdzie następują po sobie naprzemiennie faza obliczeń i faza transmisji. W fazie transmisji często potrzebna jest wymiana informacji „jeden do wszystkich", „wszyscy do jednego" lub „wszyscy do wszystkich. W czasie rozgłoszenia wszystkie procesy otrzymują kopię danych przekazywanych przez jeden z procesów. W czasie redukcji dane ze wszystkich procesów kumulowane są w jednym z procesów za pomocą pewnych operatorów (sumowanie, iloczyn, itp.). Rozgłoszenie i redukcja są podobnymi zadaniami, ale wykonywanymi w przeciwnych kierunkach. Dlatego można je rozpatrywać jako analogiczne problemy.

Dosyć naiwnym sposobem rozgłaszania wiadomości jest wysyłanie danych z jednego procesu kolejno do wszystkich innych. Spowodowałoby to przeciążenie jednego procesu całą transmisją w klastrze, podczas gdy można rozłożyć zadania bardziej równomiernie. Dobrą drogą do osiągnięcia wysokiej efektywności rozgłoszenia wydaje się być wykorzystanie struktury hipersześcianu (rys. 6.5).



Rys. 6.5. Droga rozgłoszenia wiadomości w hipersześcianie a) z węzła nr *0* b) z węzła nr *5*.

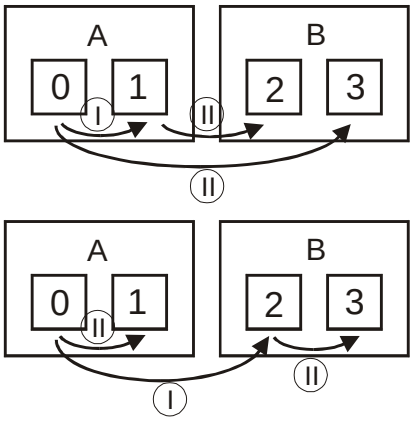
W oprogramowaniu MPI każdy proces posiada swój numer z zakresu od *0* do *N-1*. Podobnie wierzchołki hipersześcianu są również odpowiednio ponumerowane. Na rysunku 6.1 przedstawiono osiem ponumerowanych od *0* do *7* procesów ułożonych w sześcian (odmiana hipersześcianu). Aby przeanalizować kolejność procesów, którym przekazywana jest wiadomość, należy zapisać numery procesów binarnie. Dla procesu *5* będzie to *101*. Proces ten przekazuje wiadomość do procesu o takim samym numerze ale z zanegowanym najwyższym bitem. Będzie to więc *001* (dziesiętnie: *1*). Następnie obydwa procesy przez zanegowanie kolejnego bitu otrzymują numery kolejnych dwóch procesów, które otrzymają wiadomość: *111* i *011* (dziesiętnie: *7* i *3*). W trzeciej i ostatniej fazie każdy z czterech procesów, które otrzymały już wiadomość, poprzez negację swojego najniższego bitu

otrzymuje „adres” pod który należy teraz przesłać wiadomość: *100*, *000*, *110* i *010* (dziesiętnie: *4*, *0*, *6* i *2*).

Teoretycznie złożoność czasowa rozgłoszenia w hipersześcianie jest rzędu $log_2(n)$. Aby rozgłosić informację w ośmiu węzłach, potrzebne są tylko trzy fazy przesyłu danych. W praktyce może się okazać, że transmisje, które powinny się były wykonywać niezależnie, wykorzystują te same zasoby (kartę sieciową, przełącznicę, itp.), gdyż struktura hipersześcianu dotyczy tu połączeń logicznych a nie fizycznych. Może to spowodować że czas wykonania rozgłoszenia jest trudny do przewidzenia, gdy nie znamy fizycznej struktury klastra.

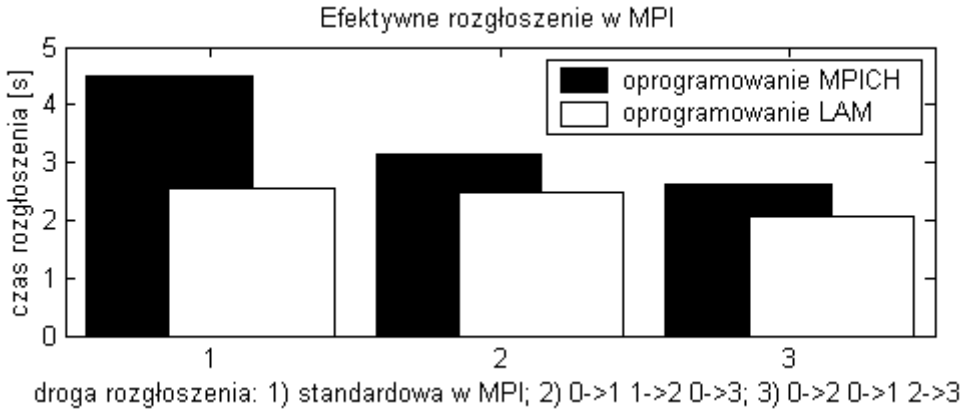
6.2.1 Optymalizacja rozgłoszeń

MPI posiada standardową funkcję rozgłoszenia. Aby zbadać jej efektywność, zbadano czas rozgłoszenia wiadomości wielkości *100MB* w klastrze o dwuprocesorowych węzłach. Już po wstępnej analizie, znając możliwości transmisyjne klastra, okazało się że można skrócić ten czas ustalając własną drogę rozgłoszenia wiadomości. Schemat (rys. 6.6) demonstruje przykładowe zadanie, gdzie uruchomione zostały *4* procesy, po dwa na każdym z węzłów. Proces *0* rozsyła rozgłoszenie do pozostałych. Można to zrobić na kilka różnych sposobów, ustalając kolejne fazy rozprzestrzeniania się informacji.



Rys. 6.6. Różne warianty drzewa rozgłoszenia.

Dla większej liczby węzłów ustalenie efektywnego drzewa rozgłoszenia może się okazać o wiele bardziej złożone.



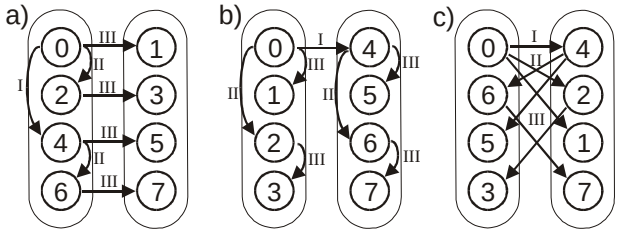
Rys. 6.7. Pomiar czasów dla standardowego rozgłoszenia oraz ustalonej kolejności rozgłoszenia w MPICH i LAM.

Wykres (rys. 6.7) przedstawia porównanie czasu działania standardowego rozgłoszenia MPI oraz czasów rozgłoszeń według zadanego drzewa. Widać, że udało się otrzymać czasy krótsze od standardowych, jednakże nie według uniwersalnego algorytmu a tylko ręcznie zadanego schematu.

6.2.2 Numeracja procesów

Aby zbudować algorytm rozgłoszenia dla liczby procesów będącej dowolną potęgą dwójki, można potraktować numery procesów MPI jako odpowiednie wierzchołki hipersześcianu, w którym będzie się rozgłaszać informację. Pozostawienie istniejącej numeracji może się jednak okazać mało efektywne. W jednym węźle może wykonywać się jednocześnie kilka procesów. Czas przesyłu wiadomości pomiędzy nimi może być kilkakrotnie krótszy niż przesył do innego węzeła. Procesy, które sąsiadują fizycznie, powinny stać się również sąsiednie logicznie w hipersześcianie.

Rozpatrzono kilka przykładów rozmieszczenia ośmiu procesów w klastrze składającym się z dwóch węzłów (rys. 6.8) analizując efektywność rozgłoszenia w tych przypadkach.



Rys. 6.8. Trzy przypadki rozmieszczenia procesów w dwóch węzłach: a) ułożenie naprzemienne, b) wariant optymistyczny, c) wariant pesymistyczny.

W ułożeniu naprzemiennym (rys. 6.8a) w trzeciej fazie będą wykonywane jednocześnie

aż cztery transmisje przez to samo fizyczne łącze, co prawdopodobnie znacząco wpłynie na czas wykonania rozgłoszenia. Rysunek 6.8b pokazuje wariant optymistyczny, gdzie odbywa się tylko jedna transmisja pomiędzy węzłami – pozostałe wykonują się już wewnątrz węzłów. Wariant c) jest pesymistyczny i wymaga każdorazowej transmisji poprzez interfejs fizyczny łączący obydwa węzły.

Najbardziej krytyczna jest ostatnia faza rozgłoszenia, gdy wykonywana jest równolegle największa liczba transmisji. W tej fazie powinny się wykonać transmisje, które będą najsłabiej obciążały interfejsy komunikacyjne klastra.

Aby tego dokonać można wykonać próbne transmisje pomiędzy każdą parą procesów. Czas przesyłu ustalonej wielkości paczki określi koszt drogi pomiędzy dwoma procesami, który będzie zapisywany w odpowiedniej tablicy. Czasy muszą być mierzone wtedy, gdy w klastrze nie odbywają się inne transmisje, aby mieć pewność, że wyniki pomiarów są niezakłócone.

Aby określić złożoność czasową rozgłoszenia w klastrze, należy przyjąć szereg założeń. Przyjęte tutaj założenia są następujące:

- klaster składa się z $N=2^n$ węzłów,
- każdy węzeł posiada $P=2^p$ procesorów i jest na nim uruchomione P procesów,
- każdy węzeł posiada jeden interfejs działający w trybie półdupleksowym, wszystkie węzły klastra połączone są za pomocą przełącznicy, która nie powoduje dodatkowych opóźnień,
- przesył określonej wielkości paczki w obrębie jednego węzła zajmuje czas T_P a pomiędzy różnymi węzłami T_N, przy czym $T_P<T_N$,
- jednoczesne transmisje w obrębie jednego węzła wykonywane przez niezależne procesy wykonują się równolegle nie powodując dodatkowych opóźnień,
- jednoczesne transmisje wykonywane przez niezależne węzły wykonują się równolegle bez dodatkowych opóźnień na przełącznicy,
- jednoczesne k transmisji wykorzystujących ten sam interfejs opóźnia transmisję k-krotnie.

W wersji optymistycznej wiadomość jest najpierw przesyłana do wszystkich węzłów klastra a później rozgłaszana w obrębie każdego węzła. Czas takiego rozgłoszenia wynosi (6.1):

$$T_{opt} = T_N \cdot \log_2 N + T_P \cdot \log_2 P \qquad (6.1)$$

Wersja pesymistyczna pomiędzy dwoma węzłami przedstawiona jest na rysunku 3c. Czas takiego rozgłoszenia wynosi $T_N \cdot (2P-1)$. Dla czterech węzłów w kolejnej fazie wszystkie procesy węzła *0* przesyłają jednocześnie wiadomość do węzła *2*, a wszystkie procesy węzła *1*

do węzła *3*. Czas tej operacji wynosi $T_N \cdot P$. Ostateczny czas rozgłoszenia w wersji pesymistycznej wynosi (6.2):

$$T_{pes} = T_N \left(2 \cdot P - 1 + P \cdot \left(\log_2 N - 1\right)\right) \qquad (6.2)$$

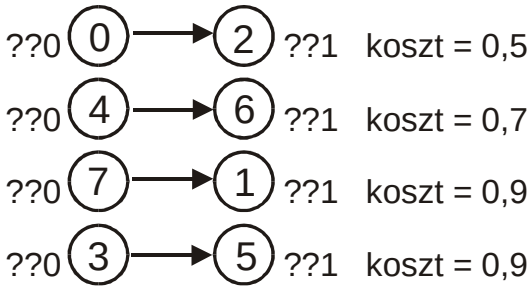
We wzorze na czas pesymistyczny w ogóle nie występuje krótszy czas T_P a jedynie dłuższy T_N. Czas T_{pes} jest prawie liniowo zależny od liczby procesów uruchomionych w jednym węźle *P*.

Warto więc spróbować zmienić numerację procesów tak aby przybliżyć się do wariantu optymistycznego.

6.3 Realizacja algorytmu numeracji procesów

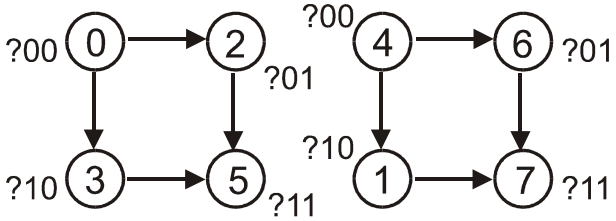
6.3.1 Opis działania algorytmu

Koszty ustalone za pomocą pomiaru czasów przesyłu wiadomości pomiędzy procesami posłużą nam do pogrupowania procesów w pary gdzie komunikacja będzie najszybsza (rys. 6.9).



Rys. 6.9. Przykładowe numery procesów, które będą połączone w pary (obok wypisano nową numerację w zapisie bitowym - na razie ustalono najniższy bit).

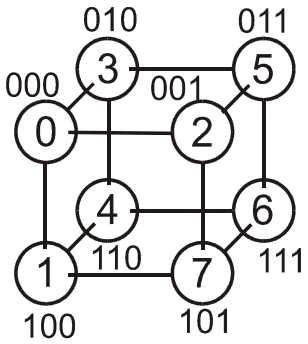
Następnym krokiem jest łączenie utworzonych już par w czwórki. Szukane są połączenia o najmniejszym koszcie, między procesami, które nie należą do tej samej pary.



Rys. 6.10. Numery procesów, które będą połączone w czwórki (obok wypisano nową numerację w zapisie bitowym - na razie ustalono dwa najniższe bity).

W ten sposób otrzymuje się dwie czwórki, które jeszcze należy połączyć w jedną ósemkę

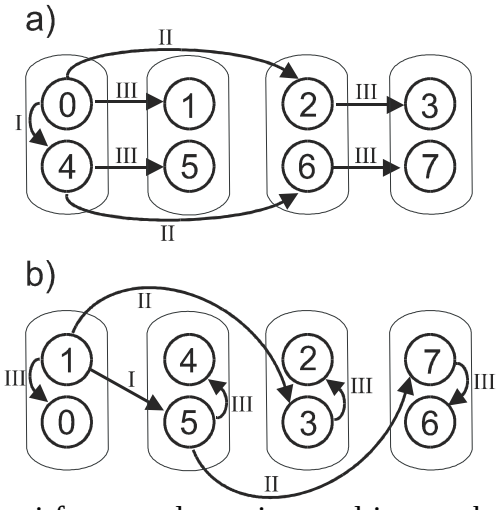
(rys. 6.10). Wynik działania algorytmu przedstawiony jest na rysunku 6.11.



Rys. 6.11. Procesy połączone w hipersześcian (bitowo zapisano pozycje w hipersześcianie).

Procesy między którymi odbywa się najszybsza komunikacja, umieszczone są w pozycjach hipersześcianu różniących się tylko najniższym bitem. Będą więc odpowiadały za ostatnią fazę rozgłoszenia, która może się okazać krytyczna dla efektywności algorytmu.

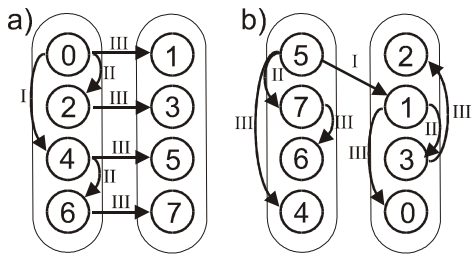
Algorytm został przetestowany w klastrze badawczym Politechniki Opolskiej. Do pierwszego testu wykorzystano cztery węzły i uruchomiono na nich osiem procesów po czym zmierzono czas rozgłoszenia paczki wielkości *100 MB*. Rysunek 6.12a przedstawia kolejne fazy rozgłoszenia przed zmianą numeracji, natomiast 6.12b po wykonaniu algorytmu numeracji procesów.



Rys. 6.12. Numery procesów i fazy rozgłoszenia przed i po wykonaniu algorytmu numeracji dla ośmiu procesów i czterech węzłów.

Czas wykonania rozgłoszenia zmniejszył się z *5,15 s* do *4,82 s*. Uzyskano więc stosunkowo nieduże *6%* przyspieszenie.

Drugi test wykonano dla ośmiu procesów uruchomionych w dwóch węzłach. Rysunki 6.13a i 6.13b przedstawiają kolejne fazy rozgłoszenia przed i po zmianie numeracji.



Rys. 6.13. Numery procesów i fazy rozgłoszenia przed i po wykonaniu algorytmu numeracji dla ośmiu procesów i dwóch węzłów.

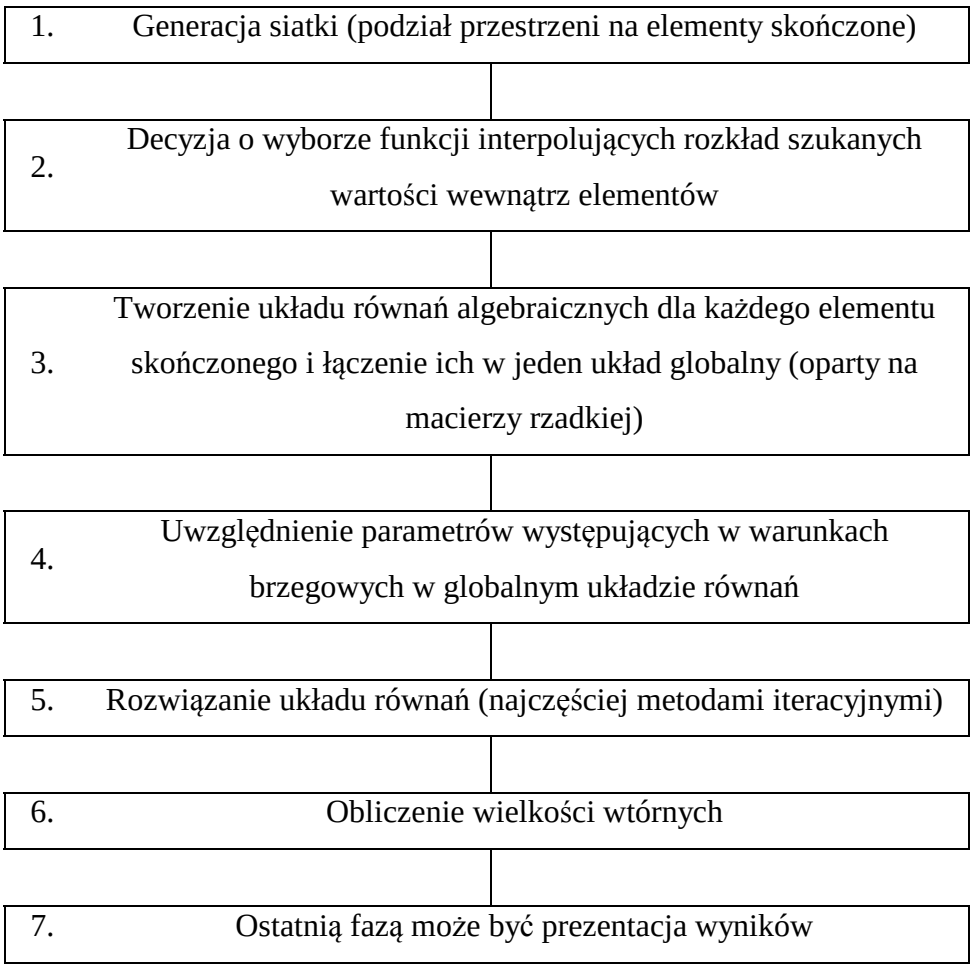
Czas wykonania rozgłoszenia zmniejszył się z *5,35 s* do *3,69 s*. Uzyskano więc tym razem większe, ok. *31%* przyśpieszenie.

6.4 Podsumowanie

Pełne wykorzystanie klastra obliczeniowego wiąże się z zastosowaniem efektywnej komunikacji. Można ją uzyskać na różne sposoby w zależności od rodzaju wykonywanych obliczeń i możliwości wykorzystywanego sprzętu. Przed zaimplementowaniem danej metody należy zbadać jej efektywność dla konkretnego zastosowania.

7 Równoległa implementacja programu

Program rozwiązujący problemy za pomocą MES działa kilkuetapowo (rys. 7.1).



Rys. 7.1. Etapy działania aplikacji MES.

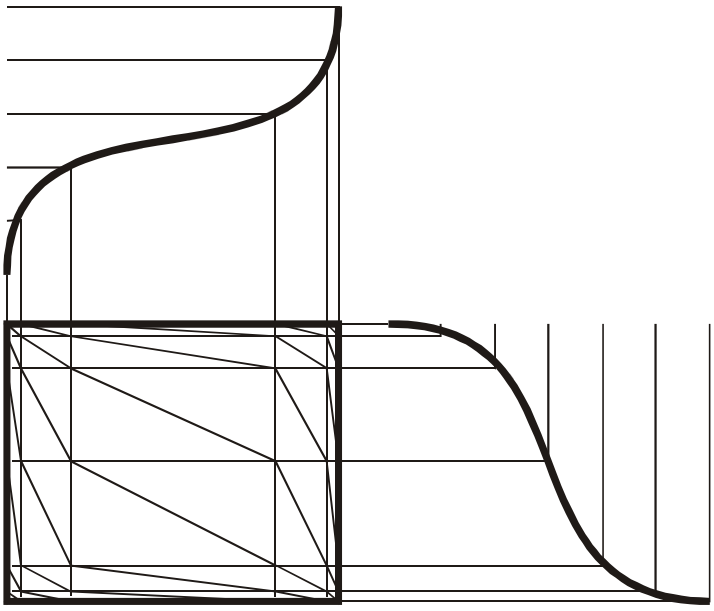
Etapy 1, 3, 4, 5, 6 mogą zostać wykonane równolegle lub prawie równolegle (potrzebne są momenty wymiany danych pomiędzy procesami).

Opracowana aplikacja równoległa została wykonana w systemie Linux przy wykorzystaniu oprogramowania klastrowego LAM MPI w języku programowania C++. Realizuje ona punkty 1 - 5 przy czym decyzja o wyborze funkcji interpolujących (etap 2) podjęta była już na etapie projektowania programu. Pozostałe etapy zostały zrealizowane równolegle, przy czym szczególną uwagę poświęcono efektywnej implementacji etapu 5 (rozwiązanie układu równań). Etap ten rozwiązuje równoległa implementacja algorytmu CG (można nią rozwiązywać układy symetryczne, dodatnio określone). Jedna iteracja pętli tego algorytmu składa się z operacji skalarnych i wektorowych. Każda operacja wektorowa

wykonywana jest przez lokalny procesor, poczym następuje (oprócz operacji sumowania wektorów i mnożenia liczby przez wektor) komunikacja. Każda iteracja (oprócz pierwszej) zawiera następujące operacje wektorowe: sumowanie (bądź odejmowanie) wektorów (3-krotnie), mnożenie liczby przez wektor (3-krotnie), iloczyn skalarny wektorów (4-krotnie), sumowanie wszystkich elementów wektora (2-krotnie) i mnożenie macierzy rzadkiej przez wektor. Operacja, która pochłania najwięcej czasu procesorów i wymaga najwięcej operacji komunikacyjnych to mnożenie macierzy rzadkiej przez wektor. Teoretycznie obliczenia i transmisja danych mogą się wykonywać jednocześnie podczas tej operacji (tzw. overlapping), jednak nie zostało to zaimplementowane w aplikacji, uwzględniono jednak tę własność w utworzonym modelu czasowym (rozdział 8).

7.1 Generowanie siatki

Programy komercyjne posiadają złożone algorytmy generacji siatek. W tym przypadku zdecydowano się zastosować prostą, jednak nie trywialną metodę, która będzie uwzględniała prostokątne kształty podobszarów, zagęszczenie przy krawędziach oraz trójkątny kształt elementu skończonego. Zagęszczenie elementów wzdłuż brzegów obszarów dokonuje się automatycznie z wykorzystaniem funkcji kosinus (rys. 7.2).



Rys. 7.2. Podział podobszaru na trójkątne elementy skończone

7.2 Tworzenie macierzy elementów

Funkcje interpolujące powinny być obliczone dla każdego elementu i spełniać następujące założenia:

- z każdym węzłem elementu związana jest jedna funkcja kształtu,
- funkcja przejmuje wartość *1* w pozycji węzła z którym jest związana a *0* dla pozostałych węzłów,
- suma wszystkich funkcji powinna przyjąć wartość *1* na obszarze całego elementu.

Dla elementów trójkątnych będzie to zazwyczaj funkcja postaci (7.1):

$$N_i = \frac{(a_i + b_i x + c_i y)}{2\Delta} \tag{7.1}$$

Do opisu pola elektrycznego obszaru wypełnionego ładunkiem, stosujemy równanie Poissona (dla uproszczenia dwuwymiarowe, bez anizotropii) (7.2):

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = -\frac{\rho}{\varepsilon_0 \varepsilon_r} \tag{7.2}$$

Funkcjonał podlegający minimalizacji to (7.3):

$$\chi = \iint \left[\frac{1}{2}\left(\frac{\partial V}{\partial x}\right)^2 + \frac{1}{2}\left(\frac{\partial V}{\partial y}\right)^2 - \frac{\rho}{\varepsilon_0 \varepsilon_r} \right] dxdy \tag{7.3}$$

Macierz dla każdego elementu można wyznaczyć ze wzoru (7.4):

$$h_{ij}^e = \int_{V^e} \left(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} \right) dxdy \tag{7.4}$$

Dla elementu trójkątnego będzie to (7.5):

$$[h]^e = \frac{1}{4\Delta} \begin{bmatrix} b_i b_i + c_i c_i & b_i b_j + c_i c_j & b_i b_k + c_i c_k \\ b_j b_i + c_j c_i & b_j b_j + c_j c_j & b_j b_k + c_j c_k \\ b_k b_i + c_k c_i & b_k b_j + c_k c_j & b_k b_k + c_k c_k \end{bmatrix} \tag{7.5}$$

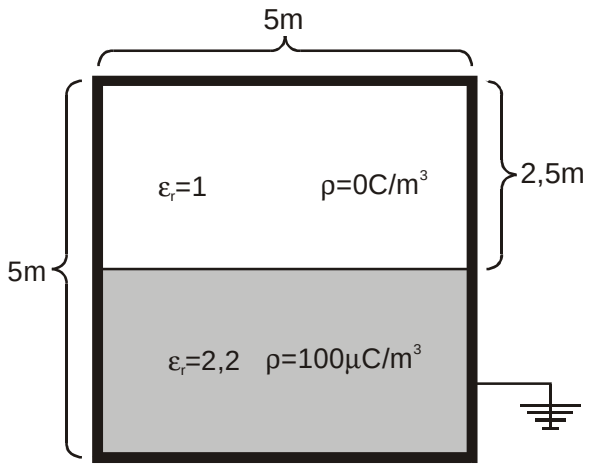
Wyprowadzenia powyższych wzorów znajdują się m. in. w książce O. C. Zienkiewicza „Metoda elementów skończonych”[Zie]. Wszystkie macierze elementów są następnie łączone w jedną macierz globalną

7.3 Przykładowe zadanie

Przepływ cieczy w rurociągach może w określonych warunkach generować ładunki elektrostatyczne. Powoduje to liczne zagrożenia w zakładach przemysłowych, a szczególnie w przemyśle petrochemicznym. Ciecze wytwarzane z ropy naftowej posiadają małą konduktywność, co sprzyja generacji ładunków, natomiast energia zapłonu oparów tych cieczy są rzędu ułamka mJ.

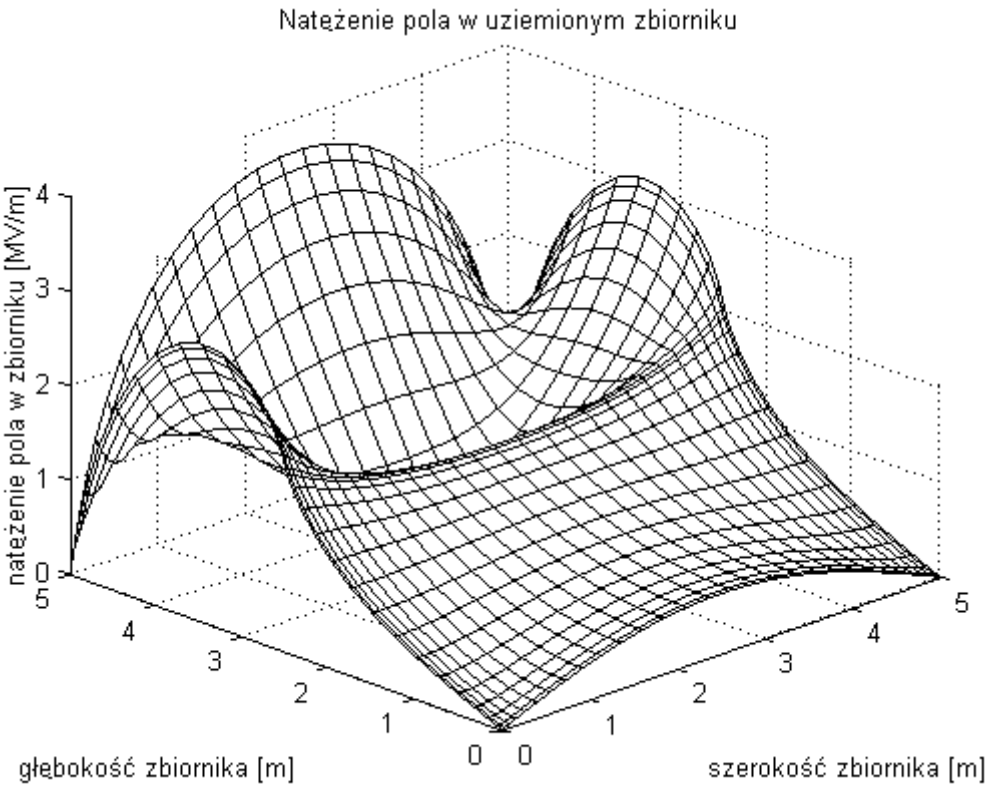
Zadanie polegało na obliczeniu rozkładu potencjałów wewnątrz zbiornika o przekroju kwadratowym *5x5 m* (rys. 7.3), do połowy wypełnionego cieczą o względnym współczynniku przenikalności elektrycznej $\varepsilon_r=2{,}2$ oraz gęstości ładunku $\rho=100\mu C/m^3$. Pozostała część

zbiornika to powietrze o względnym współczynniku przenikalności elektrycznej $\varepsilon_r=1$ i gęstości ładunku $\rho=0C/m^3$. Obudowa zbiornika jest uziemiona.



Rys. 7.3. Przykładowe zadanie - uziemiony zbiornik częściowo wypełniony naładowaną cieczą.

W obrębie cieczy obowiązuje równanie Poissona natomiast w pozostałym obszarze upraszcza się ono do równania Laplace'a. Po obliczeniu rozkładu potencjału elektrycznego, przeprowadzono analizę rozkładu natężenia pola elektrycznego (rys. 7.4).



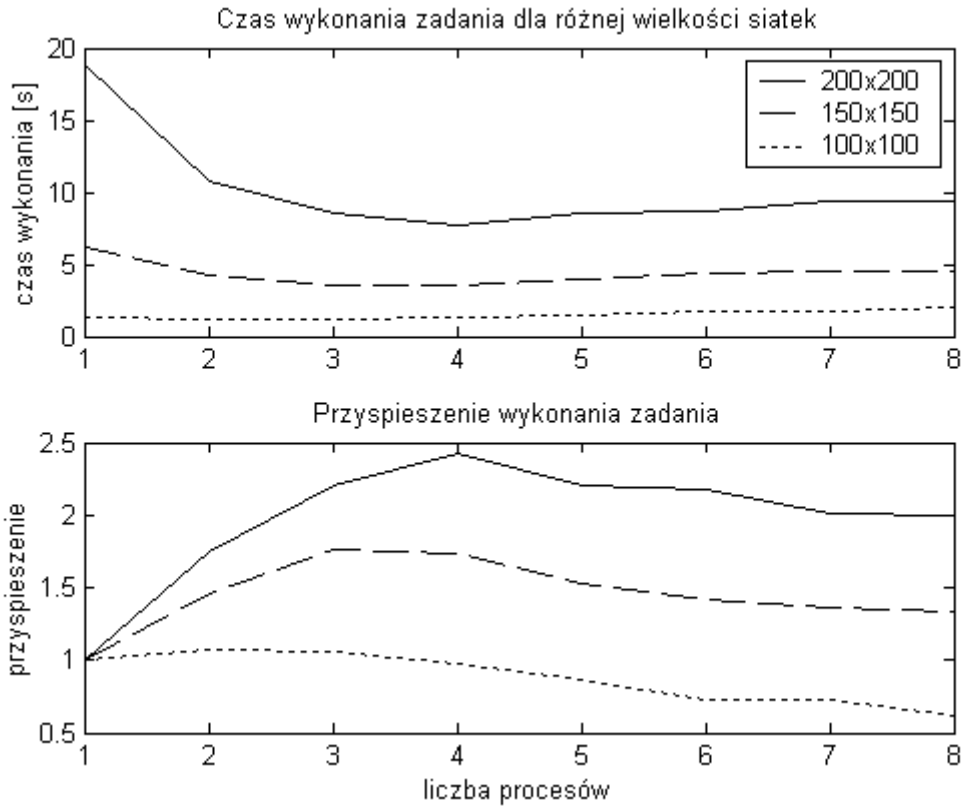
Rys. 7.4. Rozkład natężenia pola w uziemionym zbiorniku - analiza dwuwymiarowa (przekrój pionowy).

Na granicy ośrodków widoczna jest zmiana charakterystyki rozkładu natężenia. Można zauważyć, że wartości natężeń niebezpiecznie zbliżają się do wartości krytycznej (wytrzymałości dielektrycznej), która dla powietrza wynosi około *3[MV/m]*. Po przekroczeniu tej wartości może nastąpić wyładowanie i zapłon par cieczy.

Wyniki uzyskane za pomocą zaimplementowanej równolegle MES są zgodne z obliczeniami wykonanymi w programie Matlab metodą różnic skończonych.

7.4 Pomiar szybkości obliczeń

Na rysunku 7.5 pokazano czasy wykonania zadania dla trzech wielkości siatek, po uruchomieniu w *1, 2...8* węzłach klastra. Układ równań obliczono metodą CG.



Rys. 7.5. Czas wykonania zadania dla różnych wielkości siatek, przy podziale zadania na różną liczbę węzłów.

Dla każdej wielkości zadania istnieje optymalny podział zadania na części. Dla siatki *200x200* najlepsze okazało się wykonanie zadania przez *4* procesy, dla siatki *150x150* były to *3* procesy, natomiast zadanie o siatce wielkości *100x100* najszybciej zostało obliczone przy

podziale na *2* procesy. Widać na tym przykładzie następującą zasadę: większe zadania dają się efektywniej zrównoleglić, natomiast zadania małe należy wykonywać w mniejszej liczbie węzłów lub na pojedynczym komputerze.

8 Model obliczeń równoległych MES

Centralną częścią obliczeń MES jest rozwiązanie dużego rzadkiego układu równań. W zależności od rodzaju problemu (statyczny, dynamiczny), wygenerowana macierz może być symetryczna bądź niesymetryczna. Dla przeanalizowanego problemu zbiornika wypełnionego płynem macierz jest symetryczna. Dla tego typu zadań najprostszą metodą rozwiązania jest algorytm iteracyjny CG (metoda sprzężonych gradientów). Implementacja algorytmu w języku Matlab znajduje się na listingu.

```
function x = cg2(A, b, max_eps, max_step)
x = zeros(size(b)); r = b; p = r;
eps = max_eps+1; step = 0;
while ((eps>max_eps)&(step<max_step))
    ro_1 = r'*r;
    if (step>0)
        beta = ro_1/ro_2;
        p = r+beta*p;
    end
    q = A*p;
    alfa = ro_1/(p'*q);
    x = x+alfa*p;
    r = r-alfa*q;
    ro_2 = ro_1;
    sum_r = sqrt(sum(r.^2));
    sum_x = sqrt(sum(x.^2));
    eps = sum_r/sum_x;
    step = step+1;
end
```

8.1 Schemat ogólny modelu

Schemat ogólny modelu przeniesiony został z publikacji Olas T., Wyrzykowski R., Tomaś A. i Karczewski K opisującego model równoległych aplikacji MES działających w klastrach [OWT]. Czas jednej iteracji algorytmu można rozpisać jako (8.1):

$$t = t_{comp\_initial} + t_{comm} + t_{mul} + t_{comp\_final} = t_{comp}^{initial+final} + t_{comm} + t_{mul} \quad (8.1)$$

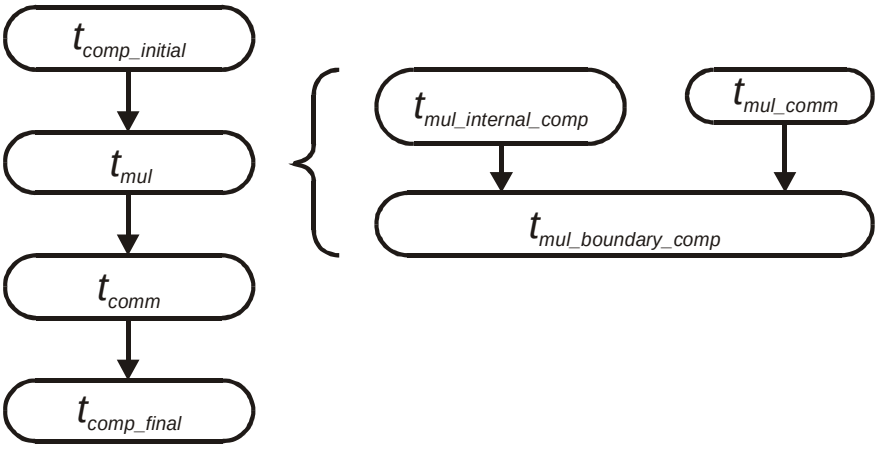
gdzie:

t_{mul} - czas mnożenia macierzy przez wektor,

t_{comm} - czas komunikacji podczas jednej iteracji algorytmu,

$t_{comp\_initial}$ + $t_{comp\_initial}$ - czas pozostałych obliczeń podczas jednej iteracji algorytmu.

Rysunek 8.1 przedstawia schemat czasowy jednej iteracji. Na początku rozpatrzono klaster złożony z komputerów jednoprocesorowych. W komputerach takich nie występuje problem jednoczesnego dostępu do pamięci przez różne procesy. Czas obliczeń nie zmniejsza się przy podziale zadania na podzadania w obrębie jednego komputera. Możliwe jest jednak często częściowe nakładkowanie (overlapping) obliczeń i komunikacji, ponieważ transmisja danych jest zazwyczaj jedynie inicjowana przez procesor, a jej dalszy ciąg jest wykonywany niezależnie.



Rys. 8.1. Schemat czasowy wykonania jednej iteracji algorytmu CG [OWT].

8.2 Czas obliczeń koprocesora

Na zamieszczonym wyżej listingu podkreślono operacje które będą wykonywane przez koprocesor arytmetyczny w kolejnych iteracjach algorytmu bez uwzględnienia operacji mnożenia macierzy przez wektor. Czas ten będzie równy (8.2):

$$t_{comp}^{initial+final} = (10n_w + 4(n_w - 1) + 10)t_f \quad (8.2)$$

gdzie:

t_f - czas wykonania jednej operacji zmiennoprzecinkowej.

n_w - liczba węzłów siatki w których wartości obliczane są przez bieżący proces

8.3 Czas komunikacji

Do przybliżenia czasu komunikacji potrzebnego do przesłania wiadomości w klastrze użyto standardowego modelu (8.3):

$$T_{comm} = \alpha + \beta w \tag{8.3}$$

gdzie:

α - czas inicjowania transmisji,

β - czas przesłania jednego współczynnika zmiennoprzecinkowego,

w - liczba współczynników do przesłania.

Wymiana informacji (oprócz operacji mnożenia macierzy przez wektor) występuje w algorytmie po iloczynie skalarnym wektorów oraz po operacji sumowania. Następuje wtedy redukcja sum częściowych (poprzez ich sumowanie sieciowe), oraz umieszczenie otrzymanego wyniku we wszystkich procesach programu. Operacja taka w MPI (*MPI_Allreduce*) zajmuje zazwyczaj czas równy sumie czasów redukcji (MPI_Reduce) i rozgłoszenia (MPI_Bcast) [Ger]. Dobre implementacje rozgłoszenia i redukcji dają złożoność logarytmiczną (patrz rozdz. 6) tych operacji. Ponieważ podczas wyżej wymienionych operacji przesyłany jest tylko jeden współczynnik zmiennoprzecinkowy, więc formułę opisującą czas komunikacji podczas jednej iteracji algorytmu można zapisać w postaci równania (8.4). Równanie (8.4) stanowi rozwinięcie modelu [OWT], uwzględniające logarytmiczną złożoność rozgłoszenia.

$$t_{comm} = 8 \cdot (\alpha + \beta) \cdot ceil(\log_2(wk)) \tag{8.4}$$

gdzie:

ceil - operacja zaokrąglania w górę,

wk - liczba węzłów klastra.

8.4 Mnożenie macierzy przez wektor

Do wyprowadzenia pozostał jeszcze czas wykonania operacji mnożenia macierzy przez wektor. W czasie mnożenia, część współczynników potrzebna do tej operacji jest dostępna lokalnie w każdym z procesów a część z nich należy uzyskać z innych procesów. Dla dużych płaskich siatek o elementach trójkątnych można liczbę współczynników niezerowych macierzy przypadających na lokalny proces zapisać wzorem (8.5):

$$nz_w = 7 \cdot n_w \tag{8.5}$$

Dla procesów analizujących krawędzie siatki, liczba ta będzie mniejsza, jednakże z punktu widzenia złożoności obliczeniowej istotne są czasy najdłuższe. Siedem elementów niezerowych w każdym wierszu macierzy wymusza podczas mnożenia macierzy przez wektor

siedem operacji mnożenia i sześć dodawania. Czas wykonywania wszystkich obliczeń w danym procesie będzie równy (8.6):

$$t_{mul\_comp} = 13 \cdot n_w \cdot t_f \tag{8.6}$$

Większość współczynników dostępna jest lokalnie, jednakże niektóre muszą zostać przesłane z procesów analizujących sąsiednie części siatki. Niektóre zaś współczynniki lokalne muszą zostać przesłane do procesów sąsiadujących. Czas potrzebny do skompletowania wszystkich niezbędnych współczynników wektorów można przybliżyć sumując (8.7) obydwa te czasy (wysyłanie i przyjmowanie danych) wzór (8.7):

$$t_{mul\_comm} = 2s\alpha + (n_e + n_b)\beta \tag{8.7}$$

gdzie:

s - liczba sąsiadujących (na siatce) procesów,

n_b - liczba węzłów siatki obliczanych przez dany proces, przylegających do innego obszaru,

n_e - liczba węzłów siatki obliczanych przez inny proces, przylegających do bieżącego obszaru.

Jeżeli założy się, że w systemie możliwy jest overlapping komunikacji wykonywany całkowicie w tle, to podczas wykonywania obliczeń mnożenia macierzy przez wektor, które nie wymagają transferu danych można wykonać przesył współczynników, które będą potrzebne w dalszych obliczeniach (patrz rys. 7.5). Należy jednak zauważyć, że overlapping może być częściowy, tzn. czas mnożenia macierzy przez wektor będzie większy niż czas samych operacji arytmetycznych, jednak mniejszy niż suma operacji arytmetycznych i transferu danych. Należałoby więc wprowadzić dodatkowy współczynnik charakteryzujący dany system. Jest to kolejna modyfikacją modelu w stosunku do wersji publikowanej w [OWT]. Dla dużych układów równań czas transmisji jest mniejszy niż czas lokalnych obliczeń i możliwe jest wykonanie całej niezbędnej transmisji podczas wykonywania lokalnych obliczeń [OWT]. Przy takim założeniu możemy wyprowadzić następującą zależność na czas mnożenia macierzy rzadkiej przez wektor (8.8):

$$t_{mul} = 13 \cdot n_w \cdot t_f + (1 - wo) \cdot (2s\alpha + (n_e + n_b)\beta) \tag{8.8}$$

gdzie:

wo - współczynnik charakteryzujący overlapping w systemie (*0* - brak, *1* - maksymalny)

Ostatecznie, po uwzględnieniu wszystkich składników czas wykonania jednej iteracji algorytmu CG można zapisać jako (8.9):

$$\begin{aligned} t = {} & (10n_w + 4(n_w - 1) + 10)t_f + 8 \cdot (\alpha + \beta) \cdot ceil(\log_2(wk)) + \\ & + 13 \cdot n_w \cdot t_f + (1 - wo) \cdot (2s\alpha + (n_e + n_b)\beta) \end{aligned} \tag{8.9}$$

$$+13 \cdot n_w \cdot t_f + (1 - wo) \cdot (2s\alpha + (n_e + n_b)\beta)$$

8.5 Obliczenia w węzłach wieloprocesorowych

Otrzymane równanie można zastosować w przypadkach, gdy uruchamia się po jednym procesie w każdym węźle klastra. W przypadku węzłów wieloprocesorowych zasadne wydaje się uruchamianie większej ilości procesów w każdym z węzłów. Teoretycznie można by założyć, że obecność czterech procesorów umożliwia jednoczesne, niezależne uruchomienie czterech procesów i uzyskanie w ten sposób czterokrotnego przyspieszenia algorytmu. W praktyce przyspieszenie może być mniejsze ze względu na specyfikę węzła klastra. Sposób zrównoleglenia dostępu do pamięci (wspólna magistrala, przełącznica krzyżowa), architektura równoległa procesorów (fizycznie rozdzielone, HyperThread, MultiCore), dostęp równoległy procesorów do pamięci Cache, możliwości systemu operacyjnego i kompilatora do wykorzystania tych technologii, itp. mają wpływ na to w jakim stopniu zadanie będzie można zrównoleglić. Uwzględnienie tych wszystkich czynników w równaniu, może spowodować małą elastyczność modelu, gdyż bardzo szybko pojawiają się nowe rozwiązania w zakresie technologii równoległych, które spowodowałyby przedawnienie się założeń. Racjonalnym sposobem uwzględnienia tych wszystkich niuansów wydaje się być wprowadzenie współczynnika aproksymującego stopień zrównoleglenia dla danej konfiguracji sprzętu i oprogramowania w zadaniach MES. Spowoduje to zmianę we wzorze zmiennej t_f na t_f* (8.10) (kolejna zmiana modelu w stosunku do [OWT]).

$$t_f * = t_f \cdot p^{1-wz} \tag{8.10}$$

gdzie:

wz - współczynnik zrównoleglenia,

p - liczba procesów uruchomiona w jednym węźle.

Współczynnik *wz=1* oznacza idealne zrównoleglenie bez straty szybkości obliczeń, *wz=0* oznacza, że zadanie nie daje się zrównoleglić w tym systemie. Należy zauważyć, że wzór może przybliżać czas, gdy liczba procesów jest mniejsza bądź równa liczbie procesorów, gdyż nie można otrzymać idealnego zrównoleglenia dla dowolnej liczby procesów, gdy liczba procesorów w systemie jest ograniczona.

Czas przesłania informacji pomiędzy procesami znajdującymi się w tym samym węźle może być wielokrotnie krótszy niż wymiana danych w sieci, dlatego należy wyznaczyć dodatkowe parametry α i β które będą charakteryzować lokalny transfer danych. Efektywne rozgłoszenie informacji pomiędzy procesami powinno najpierw rozesłać dane do każdego węzła klastra a następnie pomiędzy procesami w obrębie jednego węzła (patrz rozdz. 6). Czas

t_{comm} zmieni się następująco (8.11):

$$t_{comm} = 8 \cdot \left((\alpha_z + \beta_z) \cdot ceil(\log_2(wk)) + (\alpha_l + \beta_l) \cdot ceil(\log_2(p)) \right) \tag{8.11}$$

gdzie:

α_l i β_l - współczynniki charakteryzujące transmisję lokalną,

α_z i β_z - współczynniki charakteryzujące transmisję zdalną,

Zmianie ulegnie również czas przesyłu informacji podczas mnożenia macierzy przez wektor. W tym wypadku muszą zostać rozróżnione węzły siatki, które sąsiadują z obszarami analizowanymi przez proces znajdujący się na lokalnym i zdalnym komputerze (8.12).

$$t_{mull\_comm} = 2s_l\alpha_l + (n_{el} + n_{bl})\beta_l + 2s_z\alpha_z + (n_{ez} + n_{bz})\beta_z \tag{8.12}$$

Ostatecznie czas wykonania jednej iteracji algorytmu może być przybliżony w następujący sposób (8.13):

$$t = t_{comp}^{initial+final} + t_{comm} + t_{mul} \tag{8.13}$$

gdzie:

$$t_{comp}^{initial+final} = (10n_w + 4(n_w - 1) + 10)t_f *$$

$$t_{comm} = 8 \cdot \left((\alpha_l + \beta_l) \cdot ceil(\log_2(p)) + (\alpha_z + \beta_z) \cdot ceil(\log_2(wk)) \right)$$

$$t_{mul} = 13 \cdot n_w \cdot t_f * + (1 - wo) \cdot (2s_l\alpha_l + (n_{el} + n_{bl})\beta_l + 2s_z\alpha_z + (n_{ez} + n_{bz})\beta_z)$$

$$t_f * = t_f \cdot p^{1-wz}$$

n_w - liczba węzłów siatki analizowana przez dany proces,

α_l, β_l, α_z, β_z - współczynniki transmisji lokalnej i zdalnej,

p, wk - liczba uruchomionych procesów w węźle klastra, liczba węzłów klastra,

t_f - czas wykonania jednej operacji przez koprocesor,

n_{el}, n_{bl}, n_{ez}, n_{bz} - liczby węzłów siatki sąsiadujących z bieżącym lub innym procesem położonym lokalnie bądź zdalnie,

wo, wz - współczynniki overlappingu i zrównoleglenia.

8.6 Weryfikacja modelu

Aby zweryfikować model należało określić wartości współczynników charakterystycznych dla badanego klastra. Współczynniki α i β otrzymano mierząc przesył małych i dużych paczek współczynników. Układ równań postaci $\begin{cases} t_1 = \alpha + \beta w_1 \\ t_2 = \alpha + \beta w_2 \end{cases}$ pozwala na obliczenie ich wartości. Współczynnik t_f otrzymano przez odpowiednie podstawienie do modelu bez zrównoleglenia i overlappingu. Wartość wo ustalono wykonując operację mnożenia macierzy. Najpierw zmierzono czas samego mnożenia macierzy (t_1) i wyznaczono

paczkę informacji, której przesył zajmował dokładnie taki sam czas. Następnie zmierzono czas jednoczesnego wykonania mnożenia macierzy i transmisji (t_2). Współczynnik overlappingu otrzymano podstawiając: $wo = \left(\frac{t_1}{t_2} - \frac{1}{2} \right) * 2$. Ponieważ współczynnik zrównoleglenia może się okazać charakterystyczny dla danego typu algorytmu, należało go obliczyć mierząc wykonanie jednej iteracji algorytmu dla jednego i większej liczby (mniejszej od liczby procesorów) procesów uruchamianych na jednym węźle, a następnie dobierając współczynnik *wz* tak, aby wyniki doświadczalne najlepiej pasowały do modelu. Współczynniki dla badanego klastra wynosiły:

α_l - *0,82μs*

β_l - *0,029μs*

α_z - *50μs*

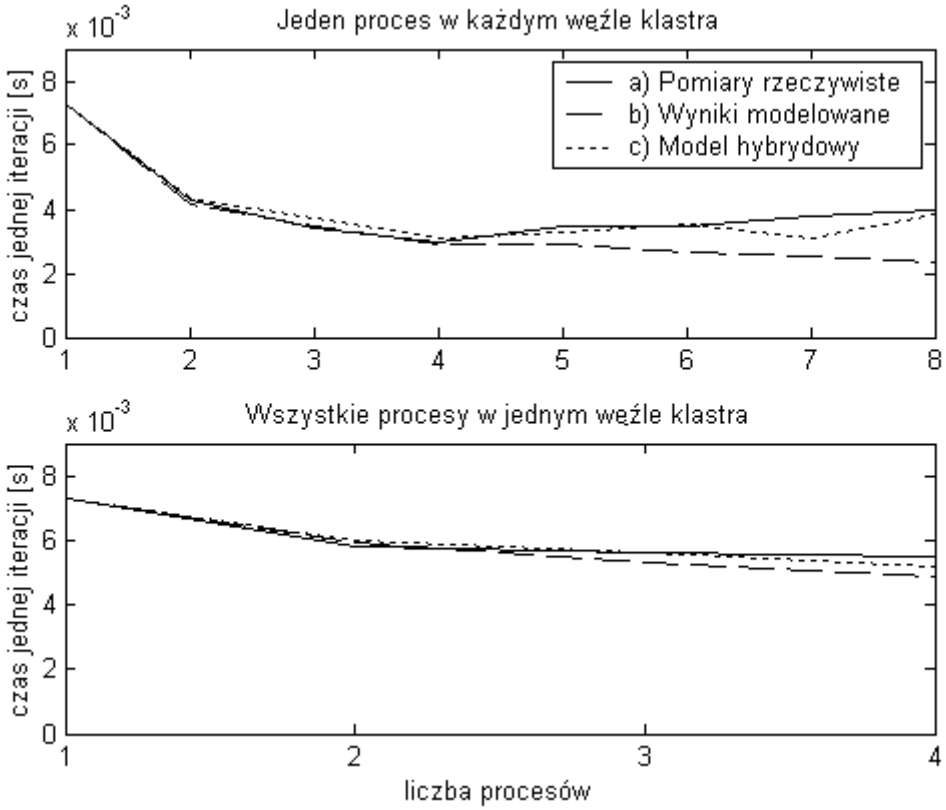
β_z - *0,07μs*

t_f - *0,0067μs*

wo - *1*

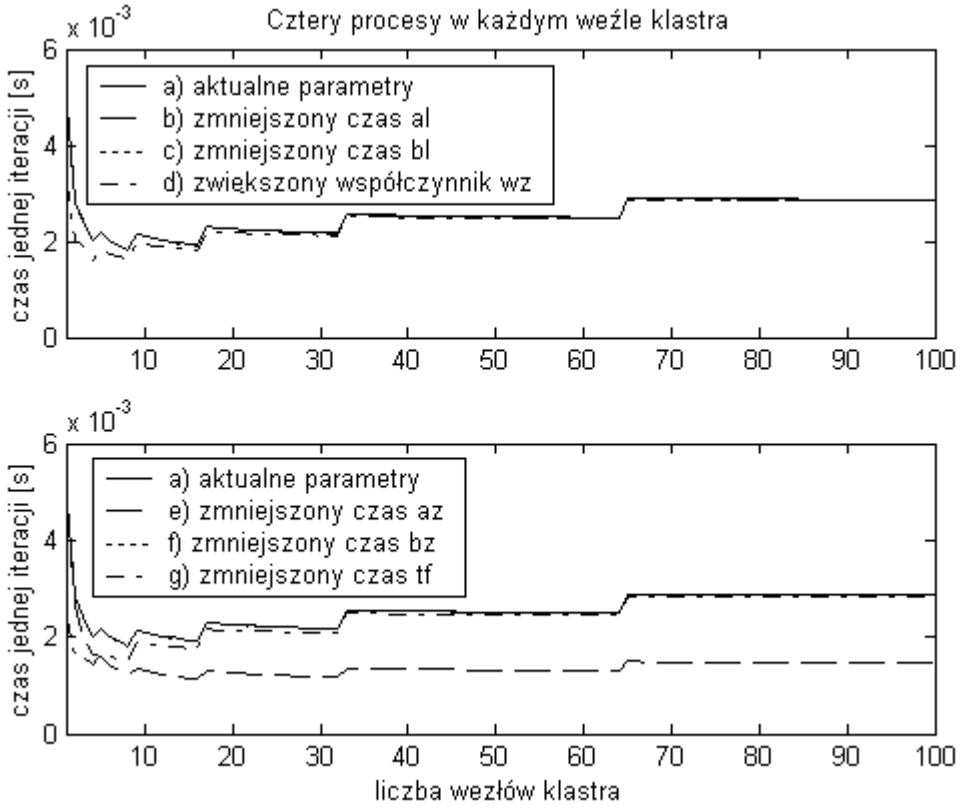
wz - *0,3*

Dużą rolę w wymianie danych odgrywa czas inicjalizacji transmisji, który dla komunikacji zdalnej jest około *60*-krotnie większy niż dla lokalnej. Długi czas inicjalizacji jest charakterystyczny dla sieci Ethernet. Aby to zmienić należałoby wprowadzić sieć Myrinet o wielokrotnie krótszym czasie inicjalizacji [OWT]. Współczynnik overlappingu równy *1* oznacza, że jednoczesna transmisja danych i obliczenia nie kolidują ze sobą w zauważalny sposób. Niski natomiast okazał się współczynnik zrównoleglenia - mimo, że każdy węzeł klastra ma dwa procesory, obliczenia nie przebiegają niezależnie. Przyczyną są najprawdopodobniej konflikty w dostępie do pamięci przez różne procesy. Rysunek 8.2 porównuje wyniki uzyskane za pomocą modelu i otrzymane doświadczalnie. Wyniki uzyskane dla więcej niż czterech węzłów klastra zaczynają się znacząco różnić od siebie. Jest to spowodowane tym, że model uwzględnia optymistyczną (logarytmiczną) złożoność operacji redukcji i rozgłoszenia (MPI_Allreduce). Model hybrydowy uwzględnia rzeczywiście zmierzone czasy tej operacji w klastrze.



Rys. 8.2. Weryfikacja modelu: a) czas jednej iteracji algorytmu CG zmierzony w klastrze, b) wyniki uzyskane z modelu przy uwzględnieniu optymalnego rozgłoszenia i redukcji, c) wyniki uzyskane z modelu przy uwzględnieniu rzeczywistych czasów rozgłoszenia i redukcji.

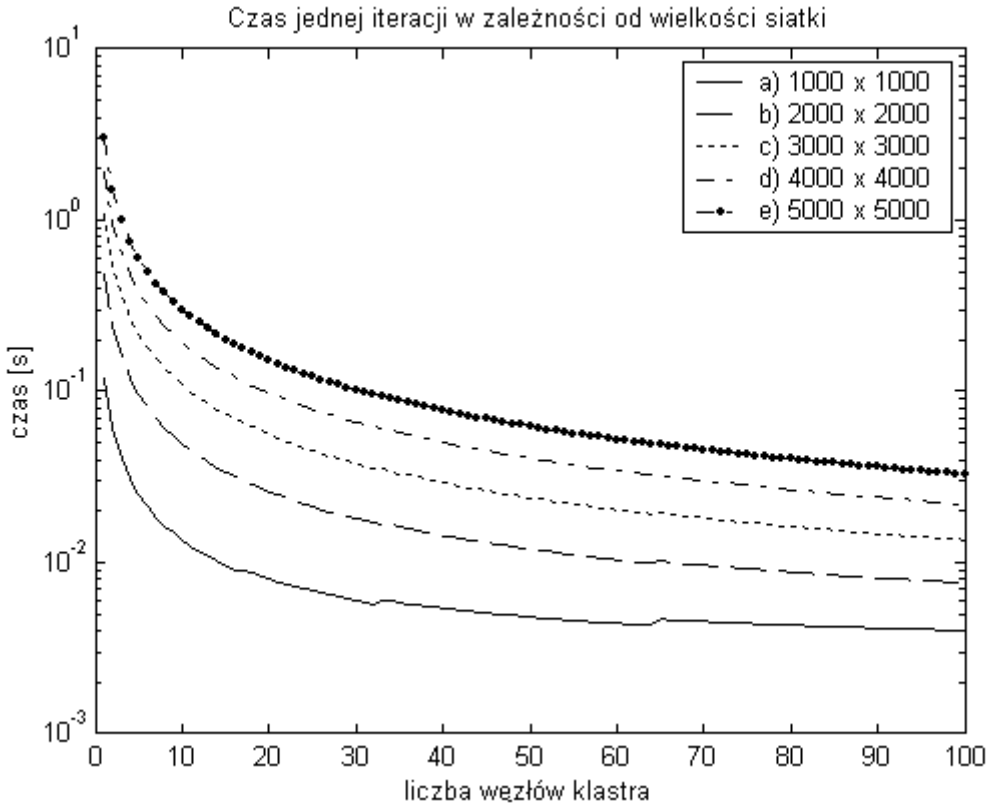
Mając do dyspozycji klastrowy model obliczeń MES, można zmodernizować istniejący już sprzęt tak, aby nasze obliczenia wykonywały się szybciej. Można to zrobić na kilka sposobów. Aby obliczenia w jednym węźle wykonywały się szybciej należy wymienić płyty główne wraz z procesorami. Ulegnie wtedy na pewno zmianie czas wykonania operacji arytmetycznych (wsp. *tf*). Może też zmienić się sposób dostępu procesorów do pamięci a co za tym idzie współczynnik zrównoleglenia (*wz*). Współczynniki α_l, β_l i *wo* mogą być zależne w dużej mierze od systemu operacyjnego. Zmieniając istniejącą sieć komputerową uzyskuje się wpływ na współczynniki α_z i β_z. Aby podjąć właściwą decyzję, należy wiedzieć jak każdy współczynnik ma wpływ na czas obliczeń. W tym celu przeprowadzono symulację czasów obliczeń dla siatki wielkości *200x200* elementów. Liczbę procesów uruchamianych na jednym węźle ustalono na *4*, a liczbę węzłów zmieniano w zakresie *1* do *100*. Wyniki symulacji prezentuje rysunek 8.3.



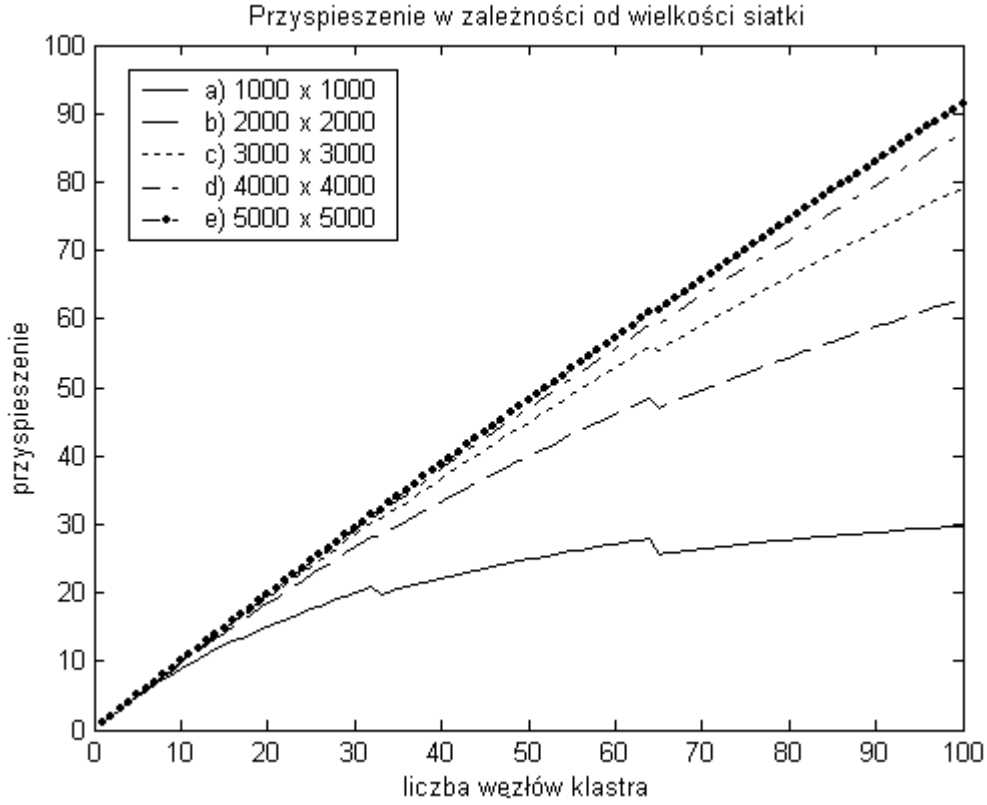
Rys. 8.3. Wynik działania modelu dla zmodyfikowanych współczynników: a) obliczenia z aktualnymi współczynnikami badanego klastra, b) zmniejszony dwukrotnie czas α_l, c) zmniejszony dwukrotnie czas β_l, d) zwiększony dwukrotnie współczynnik *wz*, e) zmniejszony dwukrotnie czas α_z, f) zmniejszony dwukrotnie czas β_z, g) zmniejszony dwukrotnie czas t_f.

Z przeprowadzonej symulacji wynika, że dla małej liczby węzłów najkorzystniejsze jest zmniejszenie czasu wykonywania operacji arytmetycznych t_f (odpowiedzialne elementy to głównie procesory, pamięć operacyjna i podręczna). Dla większej liczby węzłów, konieczna jest zmiana parametrów sieci, szczególnie czasu inicjowania transmisji danych (α_z), który w badanym klastrze przekraczał ok. 700-krotnie czas przesyłu jednego współczynnika. Długie czasy inicjowania połączeń są charakterystyczne dla połączeń Ethernet. Zakup dodatkowych węzłów klastra bez zmiany istniejącej sieci może nie dać oczekiwanego przyspieszenia obliczeń MES.

Powyższa symulacja wykonywana była dla stałej wielkości siatki. Aby przekonać się jak wielkość zadania wpływa na przyspieszenie uzyskiwane przy zrównolegleniu na większą liczbę węzłów, uruchomiono symulację dla różnych rozmiarów siatki przy uwzględnieniu aktualnych współczynników badanego klastra (rys. 8.4, 8.5).



Rys. 8.4. Czasy jednej iteracji uzyskane przez zrównoleglenie na wiele węzłów klastra dla różnych wielkości siatek MES.



Rys. 8.5. Przyspieszenie uzyskane przez zrównoleglenie na wiele węzłów klastra dla różnych

wielkości siatek MES.

Wynik obliczeń wskazuje na to, że w klastrze najlepiej zrównoleglają się duże zadania MES. Dla małych zadań narzut czasowy związany z komunikacją zmniejsza zyski wynikające z jednoczesnego wykonywania zadania na wielu węzłach.

8.7 Podsumowanie

W przypadku MES dane wejściowe ustalone są jeszcze przed wykonaniem programu. Przy ustalonym algorytmie i danych wejściowych program wykonuje zawsze te same operacje z tą samą liczbą kroków iteracji. Utworzony model jest pewnym uproszczeniem operacji zachodzących w klastrze komputerowym podczas obliczania zadania MES. Współczynnik t_f wykorzystywany w modelu należy rozumieć nie jako rzeczywisty czas wykonania operacji zmiennoprzecinkowej, ale jako średni, charakterystyczny dla tego algorytmu czas operacji zmiennoprzecinkowej wraz z przygotowaniem danych (określenie adresu w tablicy, pamięci, załadowanie koprocesora, zapis danych do pamięci). Należy również pamiętać o pamięci podręcznej (cache), która w systemach komputerowych może być wielopoziomowa, częściowo wspólna dla kilku procesorów. Efektywność wykorzystania tej pamięci zależy również od wielkości tablic wykorzystywanych w programie jak również kolejności odczytu danych. Zamodelowanie algorytmów wykorzystywanych w pamięci podręcznej wymaga pewnych uproszczeń lub (jak to zostało ustalone w bieżącym modelu) uznać je jako stały element dostępu do pamięci podczas operacji zmiennoprzecinkowych. Uproszczenia dotyczą również struktury klastra jako sieci komputerowej z jedną przełącznicą, która nie powoduje dodatkowych opóźnień w transmisji.

Stosowane uproszczenia modelu powodują, że nie odwzorowuje on idealnie rzeczywistych czasów obliczeń. Złożoność systemu komputerowego (związana ze sprzętem i oprogramowaniem) powoduje, że nawet doświadczalne kolejne pomiary czasów różnią się od siebie. Uproszczenia modelu jednak są konieczne, aby wraz ze zmianą stosowanych technologii nie było konieczności zmiany całego modelu a jedynie wartości charakterystycznych współczynników. Utworzony model nie jest też zdominowany aktualną implementacją bibliotek do programowania równoległego (MPI, PVM, i in.), ale uwzględnia optymistyczne (optymalne) charakterystyki rozgłoszeń i redukcji informacji w klastrze. Mimo wszystkich uproszczeń, można uznać, że model dobrze odwzorowuje czas procesów obliczeniowych podczas wykonywania równoległego algorytmu CG - głównej części programu MES.

9 Wnioski

Założenia, które zostały wymienione we wstępie pracy, zostały osiągnięte. Został zbudowany model pewnej klasy klastrów, według którego przeprowadzono symulację mającą na celu optymalizację obliczeń równoległych MES.

Zaimplementowano prostą, równoległą aplikację MES, w której wprowadzono optymalizację odwołań programu do pamięci, wykorzystano technikę SSE2, wprowadzono poprawiony schemat komunikacji ograniczający liczbę kolizji w czasie faz transmisyjnych algorytmu. Następnie porównano jej efektywność z kodem zaimplementowanym w specjalizowanej do tego celu bibliotece AZTEC. Wykorzystując tą aplikację równoległą dokonano obliczeń rozkładu potencjałów wewnątrz zbiornika uziemionego zbiornika, częściowo wypełnionego cieczą. Zbadano też możliwości przyspieszenia komunikacji w klastrze poprzez wykorzystanie pamięci wspólnej, nakładkowanie oraz numerację procesów. Elementy te nie znalazły się w implementacji aplikacji, jednakże nakładkowanie zostało uwzględnione w zbudowanym modelu. Jako słabość modelu można uznać fakt, że uwzględnia on jedynie rozwiązanie zadania metodą CG. Zbudowanie modeli dla innych metod wymagać będzie zazwyczaj jedynie zmiany niektórych wartości liczbowych podanego modelu.

W pakiecie Matlab zaimplementowano szereg funkcji obliczających pełne oraz rzadkie układy równań. Wykorzystując te funkcje dokonano analizy porównawczej metod dla różnych uwarunkowań macierzy. W programie Matlab zaimplementowano również funkcję modelującą czas wykonania równoległej aplikacji MES.

Wprowadzony model obliczeń równoległych MES umożliwił symulację efektywności obliczeń dla zmienionych parametrów klastra, oraz wybór najbardziej efektywnego podziału zadania na klastrze o zadanych parametrach, co było podstawą do realizacji (udowodnienia) tezy pracy.

Oryginalny dorobek naukowy rozprawy obejmuje zatem następujące osiągnięcia:

- Opracowanie - w oparciu o istniejący model czasowy - poprawionego modelu wykonania jednej iteracji algorytmu CG w klastrze obliczeniowym, a w tym:
    a. zmiana złożoności czasowej operacji globalnych komunikacji (uwzględnienie logarytmicznych charakterystyk rozgłoszenia),
    b. uwzględnienie faktu kolizji w dostępie do wspólnej pamięci przez różne procesy węzła wieloprocesorowego,
    c. uwzględnienie współczynnika overlappingu charakteryzującego węzeł klastra do wykonywania równolegle obliczeń oraz transmisji danych,

- Opracowano algorytm numeracji procesów umożliwiający implementację efektywnego rozgłoszenia informacji w klastrze obliczeniowym o węzłach wieloprocesorowych i nieznanym rozłożeniu procesów w węzłach,
- Dokonanie optymalizacji czasu obliczeń poprzez redukcję odwołań do pamięci, wykorzystanie techniki SSE2, uwzględnianie obliczeń całych wyrażeń wektorowych,
- Dokonanie testów porównawczych stacjonarnych i niestacjonarnych metod iteracyjnego rozwiązywania rzadkich układów równań liniowych,
- Analiza obciążeń węzłów klastra podczas równoległego mnożenia macierzy,
- Uzyskanie parametrów analizowanego klastra, badanie obciążenia przy podziale na wątki i procesy, analiza efektywności architektury HyperThread dla wybranego algorytmu równoległego.

Przeprowadzone i zaprezentowane w niniejszej rozprawie badania wskazują na celowość i konieczność szczególnie starannego podejścia w procesie budowania aplikacji równoległych, gdyż to determinuje w dużym stopniu ostateczną efektowność prowadzonych obliczeń równoległych. Otrzymane rezultaty będą stanowiły w najbliższej przyszłości podstawę do tworzenia przez autora rozprawy efektywnej biblioteki funkcji pozwalającej na w miarę proste budowanie klastrowych aplikacji dla rozważanych typów problemów i zagadnień numerycznych, uwzględniających specyfikę architektury klastrów obliczeniowych.

Literatura

[AMJ] Agrawal A.B., Mufti A.A., Jaoger L.G., *Band-schemes vs frontal-routines in nonlinear structural analysis*, Int. J. Num. Meth. Engng., 1980, vol. 15, s. 753-766,

[AnF] Anily S., Federgruen A., *Structured Partitioning Problems*, Operations Research,1991.

[BBC] Barett 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*, PA, http://www.netlib.org/templates/Templates.html, Philadelphia 1994,

[BeH] Beers G., Haas W., *A partitioned frontal solver for finite element analysis*. Int. J. Num. Meth. Engng., 1982, vol. 18, nr 11, s. 1623-1654,

[BBK] Bermond J.C., Bonnecaze A., Kodate T., Perennes S. and Sole P., *Broadcasting in hypercubes under the circuit-switched model*. In Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2000.

[BSS] Bolkowski S., Stabrowski M., Skoczylas J., Sroka J., Sikora J., Wincenciak S., *Komputerowe metody analizy pola elektromagnetycznego*, WNT, Warszawa 1993,

[Bor] Borowik B.E., *Programowanie równoległe w zastosowaniach*, MIKOM, Warszawa 2001,

[BDR] Bruck J., Dolev D., Ho C.T., Rosu M.C. and Strong R., *Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations*, Journal Of Parallel And Distributed Computing, No.40, pp.19–34, 1997,

[BuA] Bultan T., Aykanat C., *A New Mapping Heuristic Based on Mean Field Annealing*, J. Parallel and Distributed Comp., 1992,

[CaW] Cang S. and Wu J., *Time-Step OptimalBroadcasting in 3-D Meshes with Minimum TotalCommunication Distance*, Journal of Parallel and Distributed Computing, No.60, pp.966-997, 2000,

[CaA] Catalyurek U.V., Aykanat C.: *Hypergraph-PartitioningBased Decompo-sition for Parallel Sparse-Matrix Vector Multiplication*, submitted for publication in J. Parallel and Distributed Computing,

[Ch1]Charn B.: *Matrix Partitioning on a Virtually Shared Memory Parallel Machine*, IEEE Trans. Parallel and Distributed Systems, 1996,

[Ch2] Chawathe Y., *Scattercast: an adaptable broadcast distribution framework*, Multimedia Systems No.9, pp.104–118, 2003,

[ChK] Cheung Y.K., Khatua T.P., *A finite element solution program for large structure*, Int. J. Num. Meth. Engng., 1976, vol. 10, nr 2, s. 401-412,

[CFM] Cohen J., Fraigniaud P. and Mitjana M., *Polynomial-Time Algorithms for Minimum-Time Broadcast in Trees*, Theory Comput. Systems, No.35, pp.641–665, 2002,

[Cro] Crotty J.M, *A block equation solver for large unsymetric matrices arising in the boundary integral equation method*., Int. J. Num. Met. Engng., 1982, vol. 18, nr 7, s. 997-1017,

[DaB] Dahlquist G., Bjorck A., *Metody Numeryczne*, PWN, Warszawa 1983,

[DHV] Demmel J., Heath M., Van der Vorst H., *Parallel numerical linear algebra*, in Acta Numerica, vol. 2, Cambridge Press, New York 1993.

[Dij] Dijkstra E.W., *Cooperating sequential processes*, Academic Press, London 1988,

[DDS] Dongarra J., Duff I., Sorensen D., Van der Vorst H., *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelhia 1991,

[DrJ] Dryja M., Jankowscy J. i M., *Przegląd metod i algorytmów numerycznych. Cz. 2*, WNT, Warszawa 1982,

[Fl1] Fleury E., *Strategies for Path-Based Multicasting in Wormhole-Routed Meshes*, Journal of parallel and distributed computing, No.53, pp.26- 62, 1998,

[Fl2] Flynn M.J., *Some Computer Organisations and Their Effectiveness*, IEEE Trans. on Comp. vol C-21, No 9, 1972, pp. 948-960.

[Fl3] Flynn M.J., *Very high-speed computing systems*, proc. IEEE, 1986, vol. 54,

[FMW] Fortuna Z., Macukow B., Wąsowski J., *Metody numeryczne*, WNT, Warszawa 1982,

[FHP] Franke H., Hochschild P., Pattnaik P. and Snir M., *MPI-F: An efficient implementation of MPI on IBM-SP1*. International Conference on Parallel Processing, Vol. III, pp.197–201, 1994,

[Fre] Frederickson G.N., *Optimal Algorithms for Partitioning Trees*, Proc. Second ACM-SIAM Symp. Discrete Algorithms, 1991,

[FN1] Freund R., Nachtigal N., *An implementation of the QMR method based on coupled two-term recurrences*, SIAM J. Sci. Statist. Comput., 15 (1994), pp. 313-337.

[FN2] Freund R., Nachtigal N., QMR*: A quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315-339.

[Fos] Foster I., *Designing and Building Parallel Programs*, 1995, online version http://www-unix.mcs.anl.gov/dbpp/

[GBD] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek B. and Sunderam V., *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Network Parallel Computing*. Cambridge: MIT Press, 1994,

[GeL] George J.A., Lin J.W.H., *Some results on fill for sparse matrices*, SIAM J. Num. Anal. 1975, vol. 12, s. 452-455,

[Ger] Gergel V., *Optimizing Performance of MPI open-source implementations for Linux on POWER processor clusters*, Nizhni Novgorod, Russia, 2005,

[GGK] Grama A., Gupta A., Karypis G., Kumar V., *Design and Analysis of Algorithms*, in: Introduction to Parallel Computing (second edition), Pearson Education Ltd., 2003,

[GJ1] Głut B., Jurczyk T, Pietrzyk M., *Adaptacja siatek w modelowaniu metodą elementów skończonych procesów przepływu ciepła* — Mesh adaptation in finite element modelling of heat transport processes, Informatyka w Technologii Materiałów : kwartalnik AGH. — S. 90–103,

[GJ2] Głut B., Jurczyk T, Pietrzyk M., *Adaptacja siatki w symulacji procesów w metalurgii* — Mesh adaptation in the simulation of metallurgic processes KomPlasTech 2002 : zastosowanie komputerów w zakładach przetwórstwa metali : materiały 9. [dziewiątej] konferencji : Szczawnica 13–16 stycznia 2002 Kraków : Wydawnictwo Naukowe ,,Akapit", 2002, S. 63–70,

[GJ3] Głut B., Jurczyk T, Pietrzyk M., *Adaptacja siatki w symulacji procesu przemiany fazowej* — Mesh adaptation for simulation of phase transition process, PTSK : X Warsztaty Naukowe PTSK : Symulacja w badaniach i rozwoju : Kraków–Zakopane 10–12 września 2003,

[GJ4] Głut B., Jurczyk T, Pietrzyk M., *Adaptive mesh generation for non-steady state heat transport problems* WCCM V : fifth World Congress on Computational Mechanics : July 7–12, 2002 Vienna,

[Gre] Gręboszc J., *Symfonia C++*, Kallimach, Kraków,

[GLS] Gropp W., Lusk E., Skjellum A.*: Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge MA, 1995,

[GOY] Garatani K., Okuda H. and Yankawa G., *A Study of Solution Method for Large-scaled Parallel FEM (Performance Evaluation of Two Methods Based on DDM)*, Japan Society for Computational Engineering and Science, 1999,

[HGS] Haviland K., Gray D., Salama B., *Unix - programowanie systemowe*, RM, Warszawa 1999,

[Hoo] Hood P., *Frontal solution program for unsymmetric matrices*, Int. J. Num. Meth. Engng. 1976, vol. 10, s.379-399,

[HwB] Hwang K. Briggs F.A., *Computer architecture and parallel processing*, McGraw-Hill, New York 1984,

[Iro] Irons B.M. *A frontal solution program for finite element analysis*, Int. J. Num. Meth. Engng. 1970, vol. 2, nr 1, s. 5-32,

[Jag] Jagielski R., *Tablice Rozproszone* - WNT, Warszawa 1982,

[JoH] Johnsson S.L. and Ho C.T., *Optimum broadcasting and personalized communication in hypercubes*, IEEE Trans. Comput. No.38 (9), pp.1249–1268, 1989,

[Kah] Kahan W., *Gauss-Seidel methods of solving large systems of linear equations*, PhD thesis, University of Toronto, 1958,

[KaR] A. Kaceniauskas, P. Rutschmann, *Parallel FEM Software for CFD Problems*, INFORMATICA, 2004, Vol. 15, No. 3, 363-378,

[KL1] Kędzia J., Łukaniszyn M., *Influence of the tank shape on the electrostatic field generated by a space charge*, Fire Safety Journal,

[KL2] Kędzia J., Łukaniszyn M., *Sposób zmniejszenia zagrożenia wyładowaniami elektryczności statycznej w zbiornikach cieczy i materiałów sypkich,* Zgłoszenie patentowe, P-279302, 1989,

[KlV] Klinkenberg A., Van Der Minne I., *Electrostatic in the Petroleum Industry*, Elsevier, 1958,

[KoS] Kornatowski T., Styś T., *Wybrane metody analizy numerycznej*, WPW, Warszawa 1979,

[KSz] Kozielski S., Szczerbiński Z., *Komputery równoległe: architektura, elementy programowania,* WNT, Warszawa,1993,

[Kra] Krakowski M., *Elektrotechnika Teoretyczna, Tom III*, PWN, Warszawa, 1979,

[LaR] Lal. K., Rak T., *Budowa i strojenie klastra komputerowego Mosix-Linux*, Pomiary Automatyka Kontrola 1/2005,

[LEN] Lan Y., Esfahanian A-H. and Ni L., *Multicast in hypercube multiprocessor,* J. Parallel and Distrib. Computing No.8, pp.30-41, 1990,

[Lei] Leighton F. T., *Arrays-Trees-Hypercubes,* In: Introduction to Parallel Algorithms and Architectures, , Los Altos, CA, Morgan-Kaufmann, 1992,

[Leo] Leonard J., *Static Electricity in Hydrocarbon Liquids and Fuels,* 1981, J. Electrostatics, 10 s. 17-30,

[LEM] Lin X., Esfahanian A-H., McKinley P. K. and Burago A., *Adaptive multicast wormhole routing in hypercube multicomputers*, Fifth IEEE Symposium on Parallel and Distributed Computing, Dallas, TX, 1993,

[Loe] Loeb B., *Static electrification,* Gosenergoizdat, Moskwa, 1983,

[LoO] Loucif S. and Ould-Khaoua M., *Support for broadcast communication in multicomputer networks,* Microprocessors and Microsystems No.26, pp.151-159, 2002,

[LK1] Łukaniszyn M., Kędzia J., *3-D Analysis of the Electrostatics Field in the Parallepided Tank,* Archiwum Elektrotechniki,

[LK2] Łukaniszyn M., Kędzia J., *Trójwymiarowa analiza pola elektrycznego w zbiornikach kulistych i elipsoidalnych*, Zeszyty Naukowe WSI Opole, seria Elektryka, 1990,

[Luk] Łukaniszyn M., *Trójwymiarowa analiza pola elektrycznego w zbiornikach walcowych*, Archiwum Elektrotechniki,

[Mar] Marczuk G. I., *Analiza numeryczna zagadnień fizyki matematycznej*, PWN 1983,

[MoS] Mostow G.D., Sampson J.H., *Linear algebra*, McGraw-Hill, New York 1969,

[Na1] Nakajima K., *Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator*, ACM/IEEE Proceedings of SC2003, 2003,

[Na2] Nakajima, K., *Preconditioned Iterative Linear Solvers for Unstructured Grids on the Earth Simulator*, IEEE Proceedings of HPC Asia 2004, 150-169, 2004

[Na3] K. Nakajima, *The Impact of Parallel Programming Models on the Linear Algebra Performance for Finite Element Simulations*, Proceedings of VECPAR 2006 Conference (to appear in *Lecture Notes in Computer Science*)

[Na4] K. Nakajima, *Parallel programming models for finite-element method using preconditioned iterative solvers with multicolor ordering on various types of SMP cluster supercomputers*, IEEE Proceedings of HPC Asia 2005, 83-90, 2005

[Na5] K. Nakajima, *Performance of Parallel FEM Applications on Various Architectures K.*,First International Symposium for "Integrated Predictive Simulation System for Earthquake and Tsunami Disaster" Koshiba Hall, The University of Tokyo, Japan Fall,

[Nas] Nash J.C., *Compact numerical methods for computers*, Adam Hilger Ltd., Bristol 1979,

[Ola] Olas T,: *Solving Application Finite Element Modeling Problems in Parallel using Nuscas Software*, Proc. Int. Workshop on Parallel Numerics - ParNum 2000, Bratislave, Slovakia, pp. 125-143,

[OKT] Olas T., Karczewski K, Tomas A., Wyrzykowski R., *FEM Computations on Clusters Using Different Models of Parallel Programming*, Lect. Notes in Comp. Sci. 2328 (2002) 170-182,

[OLK] Olas T., Lacinski L, Karczewski K, Tomas A., Wyrzykowski R., *Performance of Different Communication Mechanisms for FEM Computations on PC-Based Clusters with SMP Nodes*, Proc. Int. Conf. on Parallel Computing in Electrical Engineering - PARALEC 2002, Warsaw, IEEE Computer Society, 2002, pp. 305-311,

[OWT] Olas T., Wyrzykowski R., Tomas A., Karczewski K, *Performance Modeling of Parallel FEM Computations on Clusters*, Częstochowa, 2003,

[PPV] Paige C., Parlett B., Van der Vorst H., *Approximate solutions and eigenvalue bounds from Krylov subspaces*, Numer. Lin. Alg. Appls., 29 (1995), pp. 115-134.

[PS1] Paige C., Saunders M., *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Soft., 8 (1982), pp. 43-71.

[PS2] Paige C., Saunders M., *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617-629.

[PeS] Peters J. and Syska M., *Circuit-switched broadcasting in torus networks*. IEEE Transactions on Parallel and Distributed Systems, No.7(3) pp.246–255, 1996,

[Po1] Pokuta W., *Badanie złożoności czasowej algorytmu równoległego mnożenia macierzy*, VI Międzynarodowe Warsztaty Doktoranckie, OWD 2004, s.344-348,

[Po2] Pokuta W., *Porównanie efektywności zadań komunikacyjnych realizowanych w klastrze obliczeniowym przy wykorzystaniu bibliotek MPI - MPICH i LAM*, Zeszyty Naukowe Politechniki Opolskiej, seria: informatyka, Opole 2005,

[Po3] Pokuta W., *Efektywna realizacja komunikacji międzyprocesowej w klastrach obliczeniowych*, X Konferencja Metrologia-Ekologia-Dydaktyka, MED'05.

[Ral] Ralston A., *Wstęp do analizy numerycznej*, PWN, Warszawa 1975,

[Roc] Rochkind M.J., *Programowanie w systemie Unix dla zaawansowanych*, WNT, Warszawa 1993,

[RuM] Rumelhart D., McClelland J. and the PDP Research Group, *Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1986,

[Saa] Saad Y., *Iterative Methods for Sparse Linear Systems*, PWS Publishing, New York, 1995,

[Sad] Sadecki J., *Algorytmy równoległe optymalizacji i badanie ich efektywności; systemy równoległe z rozproszoną pamięcią*, OWPO, Opole 2001,

[SaT] Salinger P. and Tvrdik P., *Broadcasting in All-Output-Port Meshes of Trees with Distance-Insensitive Switching*, Journal of Parallel and Distributed Computing, No.62, pp.1272–1294, 2002,

[SaS] Sanders P. and Sibeyn J. F., *A bandwidth latency tradeoff for broadcast and reduction*, Information Processing Letters No.86, pp.33–38, 2003,

[Sa1] Santos E. E., *Optimal and Efficient Algorithms for Summing and Prefix Summing on Parallel Machines*, Journal of Parallel and Distributed Computing, No.62, pp.517-543, 2002,

[Sa2] Santos E. E., *Optimal and Near-Optimal Algorithms for k-Item Broadcast*, Journal of Parallel and Distributed Computing, No.57, pp.121-139, 1999,

[SMO] Shahrabi A., Mackenzie L.M. and Ould-Khaoua M., *An analytical model of wormhole-routed hypercubes under broadcast traffic*, Performance Evaluation, No.53, pp.23–42, 2003,

[SOM] Shahrabi A., Ould-Khaoua M. and Mackenzie L.M., *Analytical modelling of broadcast in adaptive wormhole-routed hypercubes*, Microprocessors and Microsystems, No.25, pp.389-398, 2001,

[Sik] Sikora R., *Teoria pola elektromagnetycznego*, WNT, Warszawa 1977,

[SSS] Smykowski J., Stępień, S., Szymański G., *Metoda podwójnie sprzężonych gradientów rozwiązywania wielkich układów równań liniowych uzyskiwanych w MES*, IC-SPETO 2005, s. 31-34,

[SOH] Snir M., Otto S.W., Huss-Lederman S., Walker D.W. and Dongarra J., *MPI—The Complete Reference*, Cambridge: MIT Press, 1996,

[Son] Sonneveld P., *CGS, a fast Lanczos-type solver for non-symmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36-52.

[St1] Stabrowski M.M., *An algorithm for the solution of very large banded unsymmetric linear equation systems*, Int. J. Num. Meth. Engng. 1981. vol. 17, nr 7, s. 1103-1117,

[St2] Stabrowski M.M., *Block-indexed solution very large linear equation systems with symmetric DBBF coefficient matrix*, Int. J. Num. Meth. Engng, 1982, vol. 18, nr 10, s. 1529-1546,

[StB] Stoer J., Bulirsch R., *Wstęp do metod numerycznych*, PWN, Warszawa 1979 T.1, 1980 T.2,

[TKB] Tanenbaum A. S., Kaashoek M. F. and Bal H. E., *Parallel programming using shared objects and broadcasting.* IEEE Computer No.25, 1992,

[ToK] Topping B.H., Khan A.I., *Parallel Finite Element Computations*, Saxe-Coburg Publications, 1996,

[Van] Van der Vorst H., *The convergence behavior of preconditioned CG and CG-S in the presence of rounding errors*, in Preconditioned Conjugate Gradient Methods, O. Axelsson and L.Y.Kolotilina, eds., vol. 1457 of Lecture Notes in Mathematics, Berlin, New York, 1990, Springer-Verlag.

[Wil] Wilkinson J.H., *Błędy zaokrągleń w procesach algebraicznych*, PWN, Warszawa 1967,

[WBD] Wilson E.L., Bathe K.J., Doherty W.P., *Direct solution of large systems of linear equations*, Comp. and Struct. 1974, vol. 4, s. 363-372,

[XMN] Xu H., McKinley P. K. and Ni L. M., *Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers*, Parallel and Distrib. Comput. No.16, pp.172-184, 1992,

[Zim] Ziavras S. G. and Mukherjee A., *Data broadcasting and reduction, prefix computation, and sorting on reduced hypercube parallel computers*, Parallel Comput. No.22, 1996,

[Zie] Zienkiewicz O.C., *Metoda elementów skończonych*, Arkady, Warszawa 1972,

Inne materiały

[Azt] Obliczenia równoległe dla macierzy rzadkich - biblioteka AZTEC [http://www.sandia.cs.gov/CRF/aztec1.html],

[Blo] Obliczenia równoległe dla macierzy rzadkich - biblioteka BLOCKSOLVE [http://www.mcsl.anl.gov/home/freitag/SC94demo/software].

[CIn] Podstawowe informacje na temat klastrów, http://cmp.ameslab.gov/cmp/cluster_computers/cluster_intro.html,

[Clu] Informacje Politechniki Częstochowskiej na temat projektu CLUSTERIX http://clusterix.pcz.pl,

[ClW] Informacje w Wikipedii na temat projektu CLUSTERIX http://pl.wikipedia.org/wiki/CLUSTERIX,

[EaS] Informacje na temat komputera Earth Simulator http://pl.wikipedia.org/wiki/Earth_Simulator, http:// www.es.jamstec.go.jp/,

[GaH] Gahvari H., Hoemmen M., *A benchmark for register-blocked sparse matrix-vector multiply*, http://www.cs.berkeley.edu/~kubitron/courses/cs252-F03/projects/reports/project14_report_ver4.pdf,

[Har] Prace Harold C. Martin http://www.washington.edu/research/pathbreakers/1950b.html,

[IA8] Rao B.B., *Inline assembly for x86 in Linux*, http://www-106.ibm.com/developerworks/library/l-ia.html, 2001,

[Luc] Kody źródłowe biblioteki funkcji wykorzystujących technikę SSE2 w arytmetyce macierzy http://cscience.org/~lucasvr/Patches/libSIMD-lucasvr-25082004.patch,

[MaS] Malinowski W., Skoczylas A., *Macierze rzadkie: metody zapisu, wybrane operacje*, materiały dydaktyczne Politechniki Rzeszowskiej.

[Pet] Obliczenia równoległe dla macierzy rzadkich - Biblioteka PETSc [http://www.mcsl.anl.gov/petsc/petsc.html],

[PSp] Obliczenia równoległe dla macierzy rzadkich - biblioteka P-SPARSLIB [http://www.cs.umn.edu/Research/arpa/p-sparslib/psp-abs.html],

[SSE] Dokumentacja procesora Xeon, *Programing with streaming SIMD extension 2 (SSE2)*, http://www.intel.com,