Mnożenie macierzy rzadkich

Rafał Wawszczak
1828016@student.pwr.wroc.pl

Tymon Tobolski
181037@student.pwr.wroc.pl

15 czerwca 2011

Spis treści

Streszczenie

Projekt dotyczył mnożenia macierzy rzadkich. Naszym zadaniem było napisania kodu mnożącego macierze rzadkie z plików MatrixMarket (*.mtx) w formacie CSR w języku C++. Wybraną przez naszą grupę platformą był Intel Cluster, więc naszym zadaniem było przy użyciu oprogramowania Intela przyśpieszenie wykonywania naszego kodu a następnie docelowo wykorzystanie do obliczeń kilku maszyn pracujących równolegle.

Słowa kluczowe: *architektura komputerów, mnożenie macierzy, MPI, klaster*

1 Opis teoretyczny

Niezbędną wiedzą teoretyczną potrzebną do wykonania tego projektu była po pierwsze znajomość języka programowania C++, w którym powstawał kod. Następnie potrzebna była znajomość struktury zapisu macierzy w plikach MatrixMarket oraz w formacie CSR. Do napisania kodu mnożącego macierze rzadkie niezbędna była wiedza o działaniu algorytmu mnożenia zwykłych macierzy. Kolejną ważną kwestią jest niezależność danych obliczeniowych. Bez świadomości powstawania tak zwanego zjawiska wyścigów oraz wiedzy o jego konsekwencjach podczas używania wielu wątków do wykonywania operacji niemożliwym by było napisanie zrównoleglonego kodu wykonującego obliczenia na tej samej macierzy.

Do projektu wykorzystywane było oprogramowanie Intela. W związku z tym koniecznym było zapoznanie się z dostępnym softwarem, możliwościami jego wykorzystania oraz przydatności do naszego projektu. Po wybraniu oprogramowania jakim było Thread Building Blocks oraz MPI niezbędną była cała wiedza odnośnie wykorzystania oraz używania go w kodzie programu.

2 Rozwiązania problemów oraz implementacja

Pierwszym problemem było napisanie samego algorytmu mnożenia macierzy rzadkich w formacie CSR. Związany on był z samą strukturą tego formatu, która pozwala na dość swobodne poruszanie się po elementach wzdłuż wierszy macierzy w przeciwieństwie do wybierania elementów z kolumn. Obie formy dostępu do elementów były niezbędne ze względu na działanie algorytmu mnożenia macierzy, który wymaga pobierania elementów z wiersza macierzy pierwszej oraz elementów z kolumny macierzy drugiej aby obliczyć wartość danej komórki w macierzy wynikowej. Problem ten został rozwiązany poprzez transpozycję drugiej macierzy do formatu CSC działającego analogicznie do CSR lecz pozwalającego na swobodny dostęp do elementów wzdłuż kolumn. Po wykonaniu transpozycji algorytm mnożenia w prosty sposób wyszukuje odpowiadające sobie nie zerowe elementy macierzy a następnie wymnaża je i dodaje obliczając kolejne wartości wyniku.

Kolejnym problemem było zrównoleglenie wykonywania kodu przy pomocy Thread Building Blocks. Dotychczasowy algorytm nie pozwalał na równoległe wyko-

nywanie obliczeń ponieważ aby ustalić zapis każdego elementu konieczna była znajomość liczby wszystkich wcześniejszych elementów. Aby rozwiązać ten problem trzeba było opracować metodę uzupełniania zapisu CSR już po wykonaniu wszystkich obliczeń. Każdy wątek działający równolegle oblicza wartości jednego wiersza a następnie zwraca je wraz z ilością nie zerowych elementów. Gdy wszystkie wyniki zostają otrzymane program przystępuje do scalenia ich w całość oraz wyliczenia wartości tabeli rows z formatu CSR na podstawie danych otrzymanych od wszystkich wątków.

Gdy program bezproblemowo wykonywał obliczenia równolegle przy użyciu wątków problemem okazało się uruchomienie kodu z użyciem MPI na wielu maszynach. Problematyczne były nie tyle zmiany w kodzie co przygotowanie używanych maszyn oraz przesłanie na nie danych. Pojawiły się trudności z licencją, brakiem dostępu do Roota. Dodatkowo sposób dystrybucji danych obliczeniowych oraz programu wykonywalnego pomiędzy hostami wymagał przemyślenia. Finalnie każda maszyna otrzymuje całe macierze do pomnożenia oraz skompilowany program a następnie informacje o przydzielonym zakresie wierszy do obliczeń. Po wykonaniu operacji odesłane zostają obliczone wartości wraz z ich ilością oraz indeksami. Gdy master host otrzyma wszystkie dane scala je w sposób identyczny jak miało to miejsce przy użyciu wątków z TBB.

3 Przebieg badań

Początkowo testy zostały przeprowadzone na dwóch hostach: tywin i nic. Wykorzystane zostały kolejno macierze 3948 x 3948 z 60882 niezerowymi elementami (bcsstk15.mtx), 17281 x 17281 z 553956 niezerowymi elementami (e40r5000.mtx) oraz 90449 x 90449 z 1921955 niezerowymi elementami (sedkt3m2.mtx).

Podczas eksperymentu macierze były przemnażane przez siebie same. Uzyskane pomiary przedstawiały czasy wykonania obliczeń kolejno metodą przetwarzania szeregowego i dwoma metodami przetwarzania równoległego na tywinie oraz z użyciem MPI na obu maszynach. Wyniki nie były do końca satysfakcjonujące. Mimo, iż wyraźnie widać było przewagę pozostałych metod nad metodą szeregową to jedna z równoległych za każdym razem wykonywała obliczenia w czasie bardzo podobnym do MPI. Dało się zauważyć jednak iż przewaga metody równoległej malała wraz ze wzrostem rozmiaru analizowanej macierzy. Bardzo prawdopodob-

ną hipotezą było więc to, że większość czasu dla niewielkich macierzy podczas przetwarzania z wykorzystaniem MPI tracona była na dystrybucję oraz zbieranie danych przez co czas zyskany na wykonywaniu obliczeń równolegle na dwóch maszynach stawał się mało istotny. Z tego też powodu postanowiliśmy powtórzyć eksperyment ponownie na większą skalę wykorzystując tywina jako master hosta oraz komputery z uczelnianego laboratorium.

Tabela 1: Parametry symulowanych konfiguracji mikroprocesora

Host	System operacyjny	Ilość rdzeni	Procesor
tywin	Linux 2.6.32-29-generic 58-Ubuntu x86_64 GNU/Linux	4	Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GH
pump1	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump2	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump4	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump5	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump6	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump7	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump8	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump9	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump10	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump11	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	AMD Athlon(tm) Dual Core Processor 4850e
pump12	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	Intel(R) Pentium(R) D CPU 3.00GHz
pump14	Linux 2.6.32-28-generic 55-Ubuntu x86_64 GNU/Linux	2	Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GH
nic	Linux nic 2.6.32-25-generic 44-Ubuntu x86_64 GNU/Linux	4	AMD Phenom(tm) 9550 Quad-Core Processor