Branch and bound

Jakub Sawicki

6 stycznia 2016

1 Analiza PCAM

Analiza rozbita została na bloki: partition, communication, agglomeration oraz mapping. [1]

1.1 Podział

Obliczenia dla każdego wierzchołka drzewa stanowić będą podstawowe zadanie. Takie zadanie nie zawiera w sobie obliczeń dla dzieci danego wierzchołka.

1.2 Komunikacja

Każde zadanie będzie musiało otrzymać informację o swojej pozycji w drzewie obliczeń oraz informację o najlepszym dotychczas rozwiązaniu. Będzie też musiało być świadome tablicy z czasami wykonania poszczególnych tasków i globalnego limitu czasu.

Liście drzewa obliczeń będą też musiały przesłać informację o swoim rozwiązaniu, jeżeli jest ono lepsze niż poprzednie najlepsze znalezione.

1.3 Aglomeracja

Drzewo możemy podzielić na poddrzewa o głębokości m. m powinno zostać tak dobrane, żeby narzut komunikacyjny nie był zbyt duży w stosunku do obliczeń. Każdy liść takiego poddrzewa będzie też korzeniem kolejnego poddrzewa.

Na ostatnim poziomie, jeżeli liczba tasków n nie będzie wielokrotnością m, ilość poziomów poddrzewa będzie odpowiednio zredukowana.

1.4 Mapowanie

Będzie istnieć centralna kolejka, która rozdzielać będzie zadania (poddrzewa). Początkowo w kolejce umieszczone zostanie tylko jedno początkowe zadanie. Po jego zakończeniu do kolejki trafi już większa pula zadań, które zostaną rozdzielone pomiędzy procesorami.

Kolejka musi być priorytetowa. Jako pierwsze wysyłane do obliczeń muszą być zadania będące na głębszym poziomie, tak aby obliczenia wykonywały się mniej więcej w porządku DFS. Na tym samym poziomie, zadania powinny być szeregowane wg id ostatniego taska.

Kolejka taka zbierała by też od zadań informacje o zmianie najlepszego rozwiązania. Przy przydzielaniu zadań warunek ten byłby od razu weryfikowany przez kolejkę (ocena każdego rozwiązania

musiała by być przechowywana wraz z informacją o rozwiązaniu). Wtedy nawet jeśli aktualnie wykonywane by były zadania, które powinny zostać odcięte to po ich zakończeniu zajęte procesory będą już mogły zająć się lepszymi obszarami poszukiwań.

Można by też postarać się zbudować z tych kolejek kolejną strukturę drzewiastą, tak aby nie tworzyć jednego wąskiego gardła. Wszystkie procesory były by początkowo podłączone do kolejki w korzeniu. Gdy powstawałyby kolejki dla poddrzew, procesory były by kierowane do podległych kolejek z zachowaniem zrównoważenia obciążenia. Informacje o zmianie najlepszego rozwiązania były by propagowane po drzewie od liści do korzenia.

2 Implementacja

Program zaimplementowany został w pythonie. Do obsługi MPI wykorzystany został moduł mpi4py. Ze względu na niedostępność tego modułu na zeusie musiał on zostać skompilowany dodatkowo pociągając za sobą kompilację MPICH. MPICH dostępne na zeusie nie zostało skompilowane z opcją -fPIC co uniemożliwiało dynamicznego dołączenia mpi4py. Oddzielna instalacja MPICH nie jest jednak w stanie nawiązywać połączeń pomiędzy węzłami Zeusa - możliwa jest praca jedynie na jednym węźle (12 rdzeni).

Powstały dwie wersje, sekwencyjna — nie używa MPI i szuka najlepszego rozwiązania wykorzystując sekwencyjny algorytm branch-and-bound.

Równoległa — korzysta z modułu mpi4py. Proces root'a zarządza jedynie komunikacją co znacząco wpływa na efektywność. Pozostałe procesy wykonują zadania.

Obie wersje sortują na początku zadania malejąco według czasu trwania. W średnim przypadku prowadzi to do szybszego znalezienia rozwiązania.

3 Wyniki

Wykonane zostały testy dla dwóch problemów.

3.1 Problem 1

Limit czasu: 4. 39 zadań o długościach od 0,1 do 3,9.

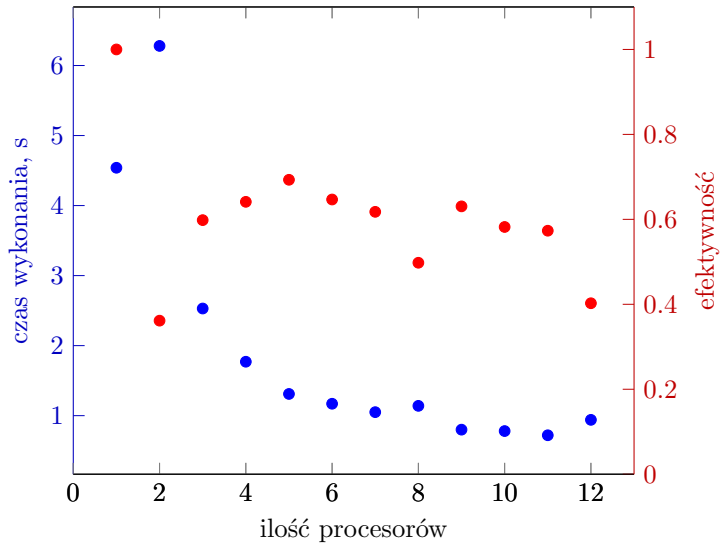
Wykres czasu wykonania i efektywności dla różnych ilości procesorów przedstawiony jest na Rys. 1. Jako czas dla jednego procesora wzięty został czas wykonania wersji sekwencyjnej algorytmu. Czas wykonania wersji sekwencyjnej jest też podstawą dla efektywności innych problemów.

Efektywność dla tych ilości procesorów utrzymuje się w okolicach 60 %. Wykazuje też dość znaczne spadki dla 8 oraz 12 procesorów. Dla 2 procesorów efektywność jest bardzo mała, jako że jeden procesor pełni funkcję zarządcy.

3.2 Problem 2

Został dobrany tak, aby lepiej obciążyć obliczeniowo procesory. Długości zadań to liczby losowe z przedziału $(0, 10)$.

Limit czasu: 10. Ilość zadań: 26.

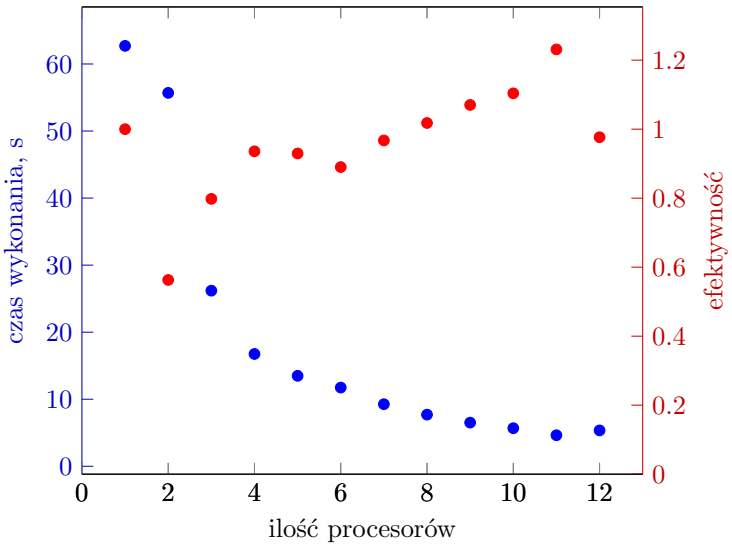


Rys. 1: Wykres czasu wykonania i efektywności dla problemu 1.

Wyniki pokazane są na Rys. 2. Widoczny jest tu również spadek wydajności dla dwóch procesorów. Poza tym, problem jest na tyle intensywny obliczeniowo, że efektywność rośnie wraz z ilością procesorów, osiągając nawet ponad 120 %. Może to wynikać, z innej kolejności przeszukiwania drzewa niż w przypadku sekwencyjnym.

Literatura

[1] I. Foster: *Designing and Building Parallel Programs* http://www.mcs.anl.gov/dbpp/ dostęp 2015.10.14



Rys. 2: Wykres czasu wykonania i efektywności dla problemu 2.