AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Artur Banul*, Konrad Wala*

Porównanie algorytmów konstrukcyjnych statycznego harmonogramowania dla wybranych architektur wieloprocesorowych

1. Wprowadzenie

NP-trudny problem harmonogramowania zadań obliczeniowych polega na takim przydziale zadań do równolegle pracujących procesorów, który minimalizuje terminy zakończenia zadań. Prezentowana praca dotyczy problemu harmonogramowania statycznego, kiedy to harmonogram jest wyznaczony przed wykonaniem aplikacji na jednostce wieloprocesorowej za pomocą algorytmów klasy APN (*Arbitrary Processors Network*), tj. algorytmów przydzielających zadania dla dowolnej architektury sieci procesorów.

W punkcie 2 przedstawiono model formalny optymalizowanego harmonogramu, w punkcie 3 opisano badane algorytmy konstruujące dopuszczalne harmonogramy, a punkt 4 zawiera wybrane wyniki badań komputerowych. W zakończeniu podano krótkie podsumowanie badań komputerowych.

2. Sformułowanie problemu

Rozważany jest wieloprocesorowy system komputerowy złożony ze zbioru $M = \{1, 2, ..., m\}$ identycznych procesorów, z których każdy posiada własną pamięć RAM oraz taką samą moc obliczeniową, tj. czas wykonania danego zadania na dowolnym procesorze jest jednakowy. Procesory systemu komunikują się między sobą przez dwukierunkowe połączenia o jednakowej przepustowości. Struktura sieci procesorów jest jednym z parametrów problemu i w szczególnym przypadku może być typu „każdy z każdym". Przyjęto, że czas tracony na komunikację zadań przydzielonych do tego samego procesora jest pomijany, gdyż operacje te wykonywane są w jego pamięci RAM. Komunikaty od zadań nadawczych

* Wyższa Szkoła Biznesu w Dąbrowie Górniczej

do zadań odbiorczych przydzielonych do różnych procesorów są przesyłane kanałami komunikacyjnymi, przy czym czasy przesyłu są zależne od rozmiaru komunikatu i długości kanału estymowanej liczbą procesorów pośredniczących w procesie przesyłu komunikatu, stąd czas przesyłu komunikatu pomiędzy zadaniami alokowanymi do różnych procesorów jest zwielokrotniany o liczbę procesorów pośredniczących w procesie przesyłania danych (por. [2, 4]).

Równoległy program komputerowy jest podzielony na elementarne, niepodlegające dalszemu podziałowi zadania obliczeniowe wykonywane na pojedynczych procesorach, bez przerwań podczas procesu realizacji. Program jest modelowany za pomocą ważonego acyklicznego digrafu $G = (V, E)$, gdzie zbiór wierzchołków V reprezentuje zbiór zadań, a zbiór łuków E relację poprzedzania w zbiorze V. Zakładamy znajomość *a priori* funkcji $p: V \rightarrow \boldsymbol{R}^{+}$ określającej średni czas wykonania zadań oraz funkcji $a: E \rightarrow \boldsymbol{R}^{+}$ określającej wagę łuków definiującą średni jednostkowy czas przesłania komunikatu pomiędzy zadaniami przydzielonymi do różnych procesorów, gdzie czas przesłania komunikatu pomiędzy konkretnymi procesorami badanej architektury jest równy czasowi jednostkowemu pomnożonemu przez liczbę procesorów pośredniczących w procesie przesyłania wzdłuż najkrótszej ścieżki (por. [2, 5]).

3. Konstrukcyjne algorytmy harmonogramowania

Daną wejściową dla algorytmu konstrukcyjnego jest digraf ponumerowanych topologicznie zadań obliczeniowych. Algorytm, na podstawie heurystycznych reguł, przydziela do każdego procesora podzbiór uporządkowanych liniowo w podciąg zadań, gdzie kolejność zadań w podciągu jest zgodna z porządkiem częściowym wprowadzonym w zbiorze zadań V przez digraf G. Na tej podstawie wyznaczane są terminy zakończenia zadań C_j, $j \in V$, przy czym uwzględniane są czasy przesyłania komunikatów między procesorami, charakterystyczne dla badanej architektury połączeń procesorów w systemie wieloprocesorowym.

W pracy podano wyniki badań następujących algorytmów konstrukcyjnych:

1. Algorytm BSA (*Bubble Scheduling and Allocation*) Ahmada i Kwoka działa na zasadzie przydziału zadań do procesora centralnego – będącego procesorem o największej liczbie połączeń, a następnie w kolejnych krokach algorytmu – na migracji zadań do procesorów sąsiednich, o ile nie zwiększy to całkowitego czasu zakończenia harmonogramu zadań (por. [5]). Zadania zostają uporządkowane za pomocą procedury Dominant Sequence, która dzieli zadania w digrafie G na trzy podzbiory: CPN – należące do ścieżki krytycznej, IBN – nienależące do ścieżki krytycznej, lecz mające z nią połączenie, oraz pozostałe OBN. Przydzielanie zadań do listy rozpoczyna się od zadania wejściowego. Kolejno przydzielone zostaje najbliższe zadanie z grupy CPN, a następnie

rekurencyjnie wszystkie jego zadania poprzedzające z grupy IBN. Na końcu działania procedury przydzielane są zadania z grupy OBN.

2. Algorytm RL (*Ready List*) działa na zasadzie przydziału podzbiorów zadań do procesorów. W pierwszej kolejności do procesora centralnego zostaje przydzielone zadanie wejściowe. W kolejnych iteracjach algorytmu generowany jest podzbiór zadań, których wszystkie zadania poprzedzające zostały wcześniej przydzielone do procesorów. Zadania z podzbioru mogą być przydzielane według dowolnie wyznaczanych funkcji priorytetów (por. [2]). W badaniach zastosowana została reguła nadająca najwyższy priorytet zadaniom o najniższej rezerwie czasowej. Zadania przydzielane są do procesorów gwarantujących najwcześniejszy termin rozpoczęcia.
3. Algorytm PL (*Priority List*) jest oparty na koncepcji listy zadań uporządkowanej według odpowiedniej funkcji priorytetu. W algorytmie można zastosować dowolną funkcję priorytetu, której działanie uwzględnia ograniczenia kolejnościowe digrafu G. W badaniach zastosowano funkcję $t\_level(i)$ określającą najdłuższą ścieżkę od wierzchołka początkowego do wierzchołka i (z wyłączeniem i). Zadanie wejściowe przydzielone zostaje do procesora centralnego, a kolejne zadania z listy przydzielane zostają do procesorów gwarantujących najwcześniejszy termin rozpoczęcia.
4. Zmodyfikowany algorytm LC (*Linear Clustering*, [3, 2]) Kima i Browne'a, oryginalnie należący do klasy UNC algorytmów harmonogramowania, którego działanie polega na grupowaniu w kolejnych iteracjach zadań leżących na nowo wyznaczanej ścieżce krytycznej w podgrafy i przydzielaniu ich do dostępnych procesorów. Działanie zmodyfikowanego LC polega na przydzieleniu zadań należących do pierwotnej ścieżki krytycznej do procesora centralnego, a następnie na ponownym wyznaczeniu ścieżki krytycznej z pominięciem przesyłu komunikatów pomiędzy zadaniami przydzielonymi. Zadania należące do kolejnej ścieżki krytycznej przydzielane są do procesorów gwarantujących najwcześniejszy termin rozpoczęcia. Jeżeli wyszukany procesor jest już obciążony zadaniami z poprzednich iteracji, to uruchamiana jest procedura pomocnicza PRECEDE szeregująca zadania na procesorze zgodnie z porządkiem częściowym określonym przez digraf G. Algorytm zaopatrzony został w dodatkową procedurę, która po przydzieleniu zadań ze ścieżki krytycznej próbuje przydzielić do aktualnego procesora zadania poprzedzające a nie należące do ścieżki krytycznej.
5. Zmodyfikowany algorytm EZ (*Edge Zeroing*, [4, 2]) Sarkara, oryginalnie należący do klasy UNC algorytmów harmonogramowania zadań, którego działanie polega na grupowaniu zadań w podgrafy, kolejno według nierosnących kosztów przesyłu komunikatów pomiędzy zadaniami. Grupowanie trwa do momentu, gdy całkowity termin zakończenia zadań zaczyna wzrastać. Działanie zmodyfikowanego algorytmu rozpoczyna się od przydzielenia zadań, pomiędzy którymi występuje największy koszt przesyłu komunikatów, do procesora centralnego. W kolejnych iteracjach zadania przydzielane są do procesorów gwarantujących najwcześniejszy termin rozpoczęcia, według

podobnej zasady jak w oryginalnym algorytmie LC. Zastosowanie algorytmu do klasy APN wymaga zastosowania dodatkowej procedury PRECEDE szeregującej zadania na procesorach.

6. Algorytm LCEZ (*Linear Clustering & Edge Zeroing*) jest hybrydowym połączeniem dwóch powyższych algorytmów. W pierwszej kolejności do procesora centralnego przydzielane są zadania należące do ścieżki krytycznej w grafie zadań. Po zaktualizowaniu czasów zakończenia zadań częściowo wyznaczonego harmonogramu uruchamiany jest algorytm EZ, który przydziela zadania o maksymalnym koszcie przesyłu danych do procesora gwarantującego najwcześniejszy czas zakończenia. Celem działania algorytmu jest w pierwszej kolejności zgrubny przydział zadań, a następnie optymalizacja przydziału zadań algorytmem EZ.

4. Badania komputerowe

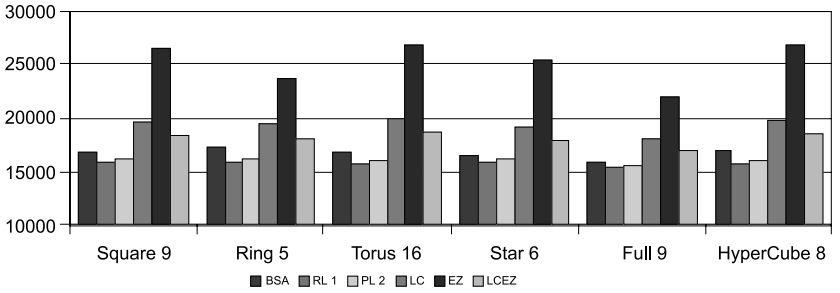
Do badań komputerowych wygenerowane zostały 72 klasy digrafów zadań po 100 egzemplarzy w każdej z klas. Grafy zadań sklasyfikowane zostały według: liczby zadań w grafie: {50; 75; 100}, współczynników gęstości łuków: {0,1; 0,3; 0,5; 0,7}, zakresu wag wierzchołków: [100, 500] oraz współczynnika *CCR*: {0,1; 0,2; 1; 2; 5; 10}; w sumie 7 200 digrafów dla każdej z 6. badanych architektur, łącznie przeprowadzono 43 200 eksperymentów obliczeniowych.

Do badań zastosowano następujące architektury wieloprocesorowe: 9-procesorowe połączenie typu „każdy z każdym", pierścień 5 procesorów, torus 16 procesorów, 6-procesorowa struktura gwiazdy, 9-procesorowa sieć 3×3, 27-procesorowa hiperkostka 3×3×3.

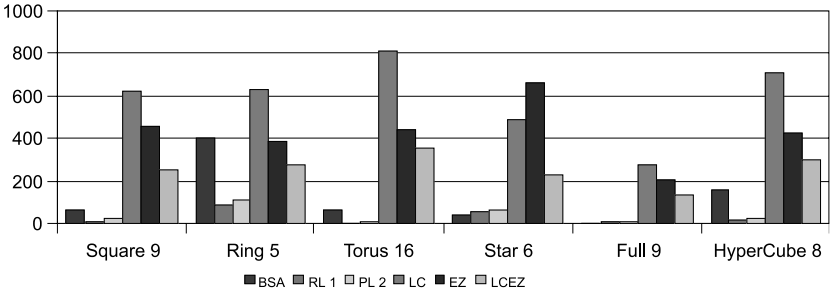
Jednym z kryterium oceny rozwiązań wyznaczonych przez algorytmy konstrukcyjne, dalej nazywane krótko rozwiązaniami, jest procentowy stopień poprawy rozwiązań przez klasyczny algorytm TS (*Tabu Serach*) realizujący 100 iteracji poprawy dla czasu tabu 10.

Badania (rys. 1–9) objęły następujące wskaźniki jakości:

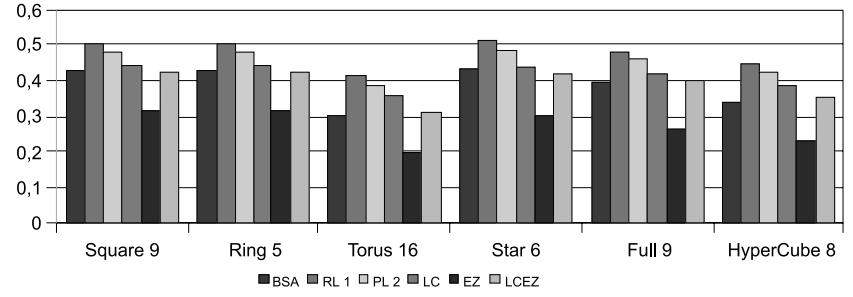
- *avg Cmax* – średni całkowity czas zakończenia zadań,
- *avg Lmax* – średnie maksymalne opóźnienie zadań,
- *avg efficiency* – średnia efektywność przydziału zadań do procesorów definiowana jako stosunek przyspieszenia programu do ilości obciążonych procesorów,
- *avg speed-up* – średnie przyspieszenie programu, definiowane jako stosunek całkowitego czasu zakończenia programu równoległego do całkowitego czasu zakończenia na jednostce wieloprocesorowej,
- *sum best* – liczba najlepszych rozwiązań na tle pozostałych algorytmów,
- *sum worst* – liczba najgorszych rozwiązań na tle pozostałych algorytmów,
- *sum t* – suma czasu działania algorytmu dla zdefiniowanych digrafów,
- stopień poprawy za pomocą algorytmu TS [%].



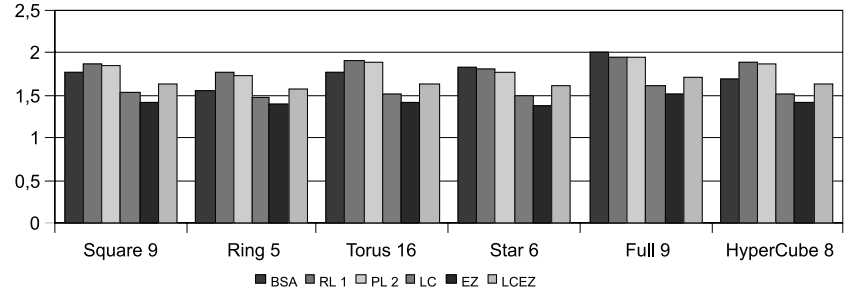
Rys. 1. Kryterium *avg Cmax* dla algorytmów z podziałem na architektury



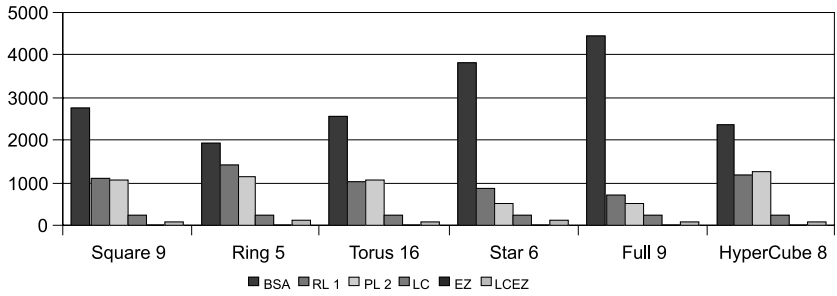
Rys. 2. Kryterium *avg Lmax* dla algorytmów z podziałem na architektury



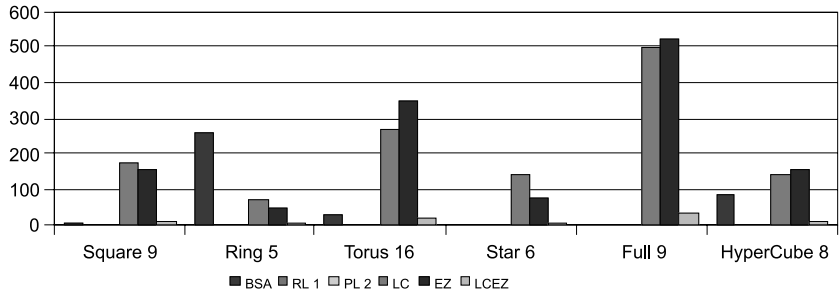
Rys. 3. *Avg efficiency* dla algorytmów z podziałem na architektury



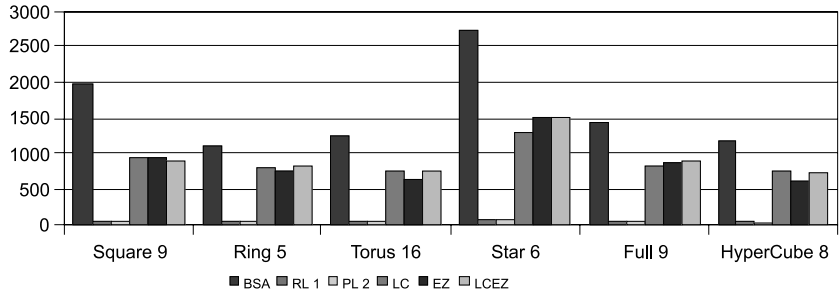
Rys. 4. *Avg speed-up* dla algorytmów z podziałem na architektury



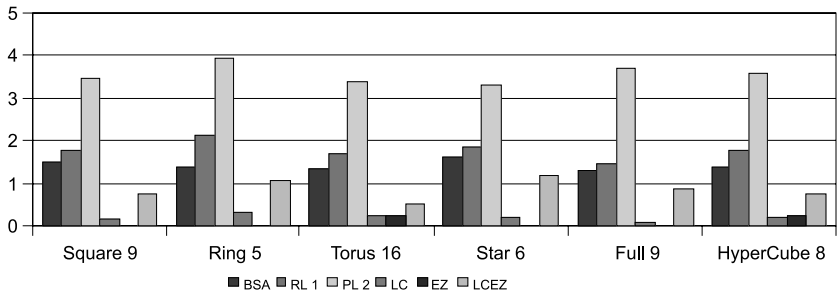
Rys. 5. *Sum best* dla algorytmów z podziałem na architektury



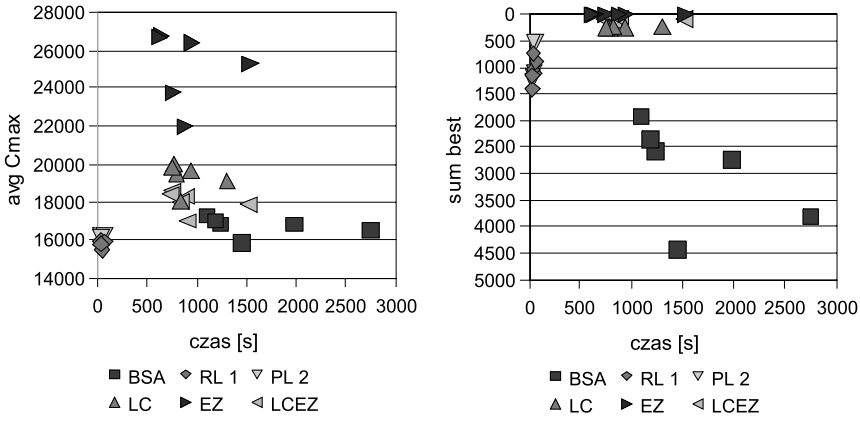
Rys. 6. *Sum worst* dla algorytmów z podziałem na architektury



Rys. 7. *Sum t* dla algorytmów z podziałem na architektury



Rys. 8. Stopień poprawy za pomocą TS[%] dla algorytmów z podziałem na architektury



Rys. 9. Kryterium *avg Cmax* oraz *sum best* w zależności od czasu działania algorytmu

5. Podsumowanie

Z przeprowadzonych badań wynika, że zwiększenie liczby procesorów nie ma tak wielkiego wpływu na jakość wyznaczonych rozwiązań, jak można by się było tego spodziewać. Zaobserwować to można na wykresach wskaźników *Cmax*, *speed-up* oraz *avg efficiency*, które dla różnych architektur połączeń i ilości procesorów nie mają, względnie, dużych odchyleń.

Wyznaczanie rozwiązań dla systemów czasu rzeczywistego jest optymalizacją dwukryterialną – pożądane są rozwiązania o najmniejszych całkowitych czasach zakończenia przy jednoczesnym minimalnym czasie wyznaczania rozwiązań. W badaniach najlepsze okazały się pod tym względem algorytmy RL oraz PL, charakteryzujące się bardzo krótkim czasem wyznaczania rozwiązań i co najważniejsze, niezależnym od architektury. Konkurencyjnym jest algorytm BSA, który w badaniach wyznaczył najwięcej najlepszych rozwiązań, co skutkuje jednak drastycznym zwiększeniem czasu wyznaczania rozwiązań. Zauważyć można również, że algorytm BSA pod względem ilości najlepszych rozwiązań oraz czasu wyznaczania jest najbardziej czuły na zmiany architektury systemu. Czas wyznaczania zwiększa się wraz ze wzrostem liczby połączeń z procesorami sąsiednimi, jednak z drugiej strony, wzrasta też wskaźnik *sum best* i zwiększa się przyspieszenie programu. Liczba najgorszych rozwiązań wzrasta dla algorytmu BSA dla architektur o małej liczbie połączeń.

Zauważyć można, że rozwiązania wyznaczone przez algorytmy BSA, RL, a przede wszystkim przez PL, charakteryzują się największym stopniem poprawy przez algorytm TS, podczas gdy rozwiązania pozostałych algorytmów raczej utknęły w minimach lokalnych.

Literatura

[1] Banul A., Wala K., *Harmonogramowanie statyczne zadań obliczeniowych w homogenicznym systemie wieloprocesorowym,* Zeszyty Naukowe Politechniki Śląskiej, seria Automatyka 2008.
[2] Banul A., *Komputerowy system do badania efektywności przybliżonych algorytmów harmonogramowania zadań obliczeniowych w wieloprocesorowym systemie czasu rzeczywistego*. Dyplomowa praca magisterska, promotor: K. Wala, Akademia Górniczo-Hutnicza, Kraków 2007.
[3] Banul A., Wala K., *Harmonogramowanie zadań obliczeniowych w wieloprocesorowym systemie czasu rzeczywistego*. Automatyka (półrocznik AGH), t. 7, z. 3, 2003, 627–635.
[4] Kim S.J., Brown J.C., *A general approach to mapping of parallel computation upon multiprocessor architectures*. International Conference on Parallel Processing, vol. 3, 1988, 1–8.
[5] Sarkara V., *Partitioning and scheduling parallel programs for executing on multiprocessors*. The MIT Press, Cambridge, MA, 1989.
[6] Kwok Y.K., Ahmad I., *Static scheduling algorithms for allocating directed tasks graphs to multiprocessors*. AMC Computing Surveys, vol. 31, No. 4, 1999, 406–471.