ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 30 Nr kol. 1315

Maciej DROZDOWSKI

SZEREGOWANIE ZADAŃ JEDNORODNYCH W ROZPROSZONYCH SYSTEMACH KOMPUTEROWYCH

Streszczenie. W pracy rozważany jest problem szeregowania zadań, które mogą być dzielone na części o dowolnych rozmiarach, przetwarzane następnie niezależnie od siebie w rozproszonym systemie komputerowym. Zadania o powyższych własnościach nazywać będziemy zadaniami jednorodnymi. Zaproponowany model procesu obliczeniowego uwzględnia czas dystrybucji zadania do rozproszonych komputerów i przetwarzania na nich. W pracy omówione zostaną najważniejsze wyniki uzyskane dla problemu szeregowania zadań jednorodnych.

SCHEDULING DIVISIBLE JOBS IN DISTRIBUTED COMPUTER SYSTEMS

Summary. In this work we analze the problem of scheduling jobs which can be divided into parts of arbitrary size. Such parts can be processed separately and independently by a distributed computer system. Jobs with the above features will be called divisible. The proposed model of the computation process includes both time of distribution to the remote sites as well as the processing time. We present the most important results for the problem of divisible job scheduling.

SCHEDULING VON TEILBAREN TASKS AUF VERTEILTEN SYSTEMEN

Zusammenfassung. In unseren aktuellen Untersuchungen im Bereich des Taskscheduling betrachten wir Tasks, die in eine beliebige Anzahl von unabhaengigen Teiltasks zerlegt werden koennen. Die Teiltasks koennen auf verschiedenen Prozessoren abgearbeitet werden. Solche Tasks nennen wir teilbar. In der Arbeit stellen wir die wichtigsten Ergebnisse zum Scheduling teilbarer Tasks vor.

1. Wstęp

Dzięki rozwojowi technologii sieciowej w ostatnich latach wzrasta praktyczne znaczenie komputerów wykonujących obliczenia w sposób równoległy i rozproszony. Oczekiwana poprawa czasu obliczeń przez stosowanie równoległego przetwarzania w sieci komputerów staje się codzienną praktyką. Środowiska programistyczne, takie jak: PVM, Linda, Express, sprawiają, że moc obliczeniowa rozproszonych komputerów może być wykorzystana w stosunkowo łatwy sposób. Uzyskano współdzielenie obliczeń (ang. *load sharing*). Nie osiągnięto jednak zadowalającego i powszechnie uznanego sposobu równoważenia obciążeń (ang. *load-balancing*). Postęp w technologii procesorów sprawił, że dysponujemy szybkimi jednostkami centralnymi, które jednak muszą łączyć się z wolnymi pamięciami i jeszcze wolniejszymi urządzeniami zewnętrzynymi (takimi jak sieć). Przepustowość kanałów komunikacyjnych jest więc cennym zasobem. Można odnieść wrażenie, że wraz z rozwojem sprzętu nie następuje zadowalający rozwój modeli teoretycznych, przydatnych do tworzenia efektywnych aplikacji rozproszonych. Wiele proponowanych podejść zmierza do optymalizacji jednych aspektów obliczeń równoległych pomijając inne. W niniejszej pracy omawiamy model, który obejmuje zarówno szeroką klasę architektur komputerowych, jak i wiele aspektów komunikacyjnych wpływających na przebieg obliczeń równoległych.

Rozpatrywać będziemy zadnia, które mogą być dzielone na części o dowolnym rozmiarze, rozwiązywane następnie niezależnie przez różne komputery. Obliczenia o powyższych własnościach nazywać będziemy zadaniami jednorodnymi (ang. *divisible job*). Wiele praktycznych aplikacji rozproszonych można uznać za zadania jednorodne. Rozważmy bazę danych składającą się z tysięcy rekordów. W celu rozproszonego wyszukania pożądanych rekordów możemy podzielić plik bazy danych z ziarnistością, która jest mała w stosunku do rozmiaru całego pliku. Można uznać, że zadanie jest jednorodne. Analogiczna sytuacja ma miejsce przy wyszukiwaniu wzorca w tekście lub w pliku graficznym. Podobnie jest przy rozproszonym sortowaniu bazy danych, przetwarzaniu dużych wolumenów danych pomiarowych przez filtrowanie, FFT itp. [14]. Kolejnym przykładem zadania jednorodnego mogą być symulacje zachowania dużej liczby cząsteczek [13], niektóre problemy algebry liniowej z użyciem dużych macierzy [8], rozwiązywanie równań różniczkowych z dekompozycją dziedziny na gęste sieci elementów skończonych [22]. Naszym celem jest określenie takiego rozdziału obliczeń, aby dla znanych prędkości łącz komunikacyjnych i komputerów całkowity czas dystrybucji danych i obliczeń był minimalny.

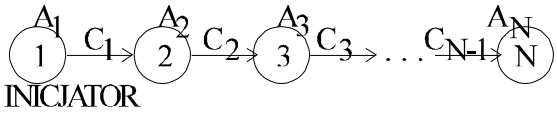
Opiszemy teraz w ogólnym zarysie przebieg procesu obliczeniowego i wprowadzimy oznaczenia stosowane w dalszej części pracy. Rozproszony komputer składa się z N elementów przetwarzających, z których każdy zawiera procesor, lokalną pamięć i jest zdolny do komunikacji w sieci. Początkowo cały wolumen V danych do przetworzenia (całe zadanie) znajduje się w jednym komputerze (elemencie obliczającym), który nazwiemy *inicjatorem*. Inicjator zatrzymuje część α_1 z całego wolumenu danych dla przetworzenia lokalnego, a resztę wysyła do sąsiada(ów) w celu przetwarzania zdalnego. Sąsiad i zatrzymuje i przetwarza część α_i, a resztę danych wysyła do kolejnych, wciąż nieaktywnych sąsiadów. Czas lokalnego przetwarzania jest wprost procporcjonalny do ilości danych i dla części α_i, obliczanej na i-tym

procesorze, wynosi $\alpha_i A_i$, gdzie A_i jest odwrotnością prędkości procesora i. Czas transmisji wolumenu danych x przez łącze j jest równy R_j+xC_j, gdzie R_j jest czasem poświęconym na zapoczątkowanie komunikacji, a C_j jest odwrotnością prędkości łącza.

Dalsza część pracy podzielona została według analizowanych architektur. W rozdziale drugim rozważamy sieć liniową, w trzecim gwiazdę i połączenie magistralowe. W kolejnym rozdziale rozważamy połączenie typu krata (ang. *mesh*), a w piątym rozdziale hiperkostkę (ang. *hypercube*) procesorów.

2. Sieć liniowa

W tym rozdziale rozważamy sieć liniową (rys.1). Będziemy zakładać, że po przeprowadzeniu rozproszonych obliczeń procesory nie zwracają wyników do inicjatora. Wykażemy dalej, iż takie uproszczenie nie ogranicza ogólności modeli, które mogą być łatwo rozszerzone, aby uwzględnić zwracanie wyników. Zaprezentujemy modele dla różnych trybów komunikacji między elementami przetwarzającymi. Zaczniemy od modelu najprostszego, a następnie będziemy pokazywać, jak go modyfikować, aby uwzględnić zmieniające się elementy.



Rys. 1. Liniowa sieć elementów przetwarzających
Fig. 1. Chain of processors

Sieć liniowa z koprocesorami komunikacyjnymi bez zwracania wyników [14,17]

Zakładamy, że każdy element przetwarzający jest wyposażony w niezależny „koprocesor" komunikacyjny i może jednocześnie komunikować się i obliczać. Proces dystrybucji danych i obliczeń jest zilustrowany na rys. 2. W przypadku gdy wyniki nie są zwracane do inicjatora (lub czas zwracania wyników można pominąć), wszystkie elementy obliczeniowe muszą zakończyć przetwarzanie jednocześnie. W przeciwnym przypadku możliwe byłoby skrócenie czasu obliczeń przez przesłanie większej ilości danych do procesorów kończących obliczenia wcześniej i zmniejszenie ilości obliczeń na procesorach kończących obliczenia później. Zauważmy, że czas obliczeń na każdym procesorze wysyłającym dane jest równy czasowi transmisji do odbiorcy plus czas obliczeń u odbiorcy (por. rys.2). Na podstawie tej obserwacji sformułujemy układ równań, z którego wyznaczymy optymalny podział zadania obliczeniowego:

$$\begin{aligned}
\alpha_1 A_1 &= (\alpha_2 + \alpha_2 + \ldots + \alpha_N) C_1 + \alpha_2 A_2 \\
\alpha_2 A_2 &= (\alpha_3 + \alpha_4 + \ldots + \alpha_N) C_2 + \alpha_3 A_3 \\
&\cdots \\
\alpha_i A_i &= (\alpha_{i+1} + \alpha_{i+2} + \ldots + \alpha_N) C_i + \alpha_{i+1} A_{i+1} \\
&\cdots
\end{aligned} \tag{1}$$

$$\alpha_{N-1}A_{N-1} = \alpha_N C_{N-1} + \alpha_N A_N$$
$$\alpha_1 + \alpha_2 + \ldots + \alpha_i + \ldots + \alpha_N = V$$

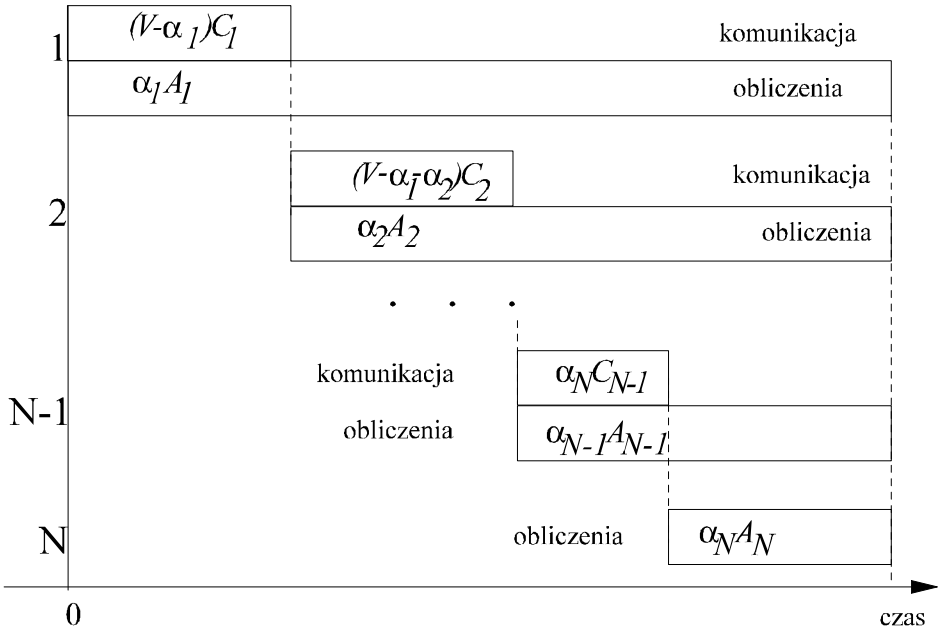
Powyższy układ równań można rozwiązać w czasie *O(N)*, wyrażając niewiadome $\alpha_{N-1},\ldots,\alpha_1$ jako funkcje α_N. Z rozwiązania równań (1) można wyznaczyć czas $\alpha_1 A_1$ trwania całego obliczenia.. Znając ten czas można wyznaczyć przyspieszenie (ang. *speedup*) S_N. Jest ono równe stosunkowi czasu obliczeń na samym inicjatorze i czasu obliczeń z użyciem sieci:

$$S_N = \frac{VC_1}{\alpha_1 C_1} = \frac{V}{\alpha_1}$$

Można również wyznaczać przeciętne wkorzystanie procesorów (ang. *utilization*), które jest stosunkiem przyspieszenia i liczby procesorów:

$$U_N = \frac{S_N}{N} = \frac{V}{N\alpha_1}$$

Znajomość powyższych parametrów umożliwia ocenę efektywności systemu komputerowego [4,16,19]. Przykład takiej oceny podamy w rozdziale 4 dla sieci typu krata.



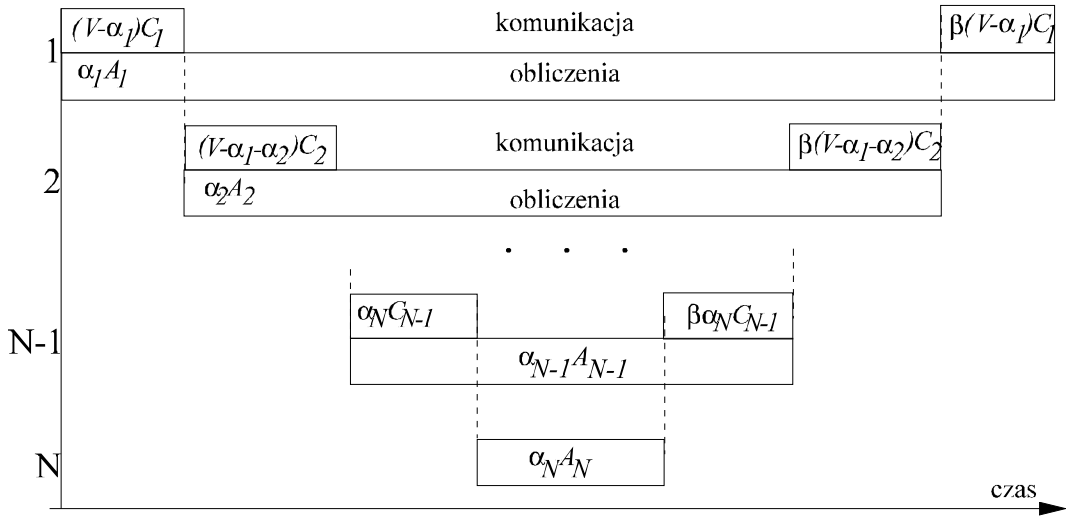
Rys. 2. Diagram komunikacji i obliczeń w sieci liniowej bez zwracania wyników
Fig. 2. Communication-computation diagram for a chain without data return

Sieć liniowa z koprocesorami komunikacyjnymi ze zwracaniem wyników [11,14]

Rozważmy teraz przypadek, w którym procesory zwracają wyniki. Założymy, że ilość zwracanych wyników jest wprost proporcjonalna do ilości otrzymanych danych. Procesor, który przetwarzał część danych α_i, zwraca $\beta\alpha_i$ wyników. Proces dystrybucji i zwracania wyników został zilustrowany na rys.3. Zauważmy, że czas obliczeń na procesorze wysyłającym dane do zdalnego obliczenia jest równy czasowi transmisji do odbiorcy plus czas obliczeń u odbiorcy plus czas transmisji wyników. Układ równań (1) trzeba zmodyfikować i zastąpić równanie *i* (*i=1,...,N-1)* następującą zależnością:

$$\alpha_i A_i = (\alpha_{i+1} + \alpha_{i+2} + ... + \alpha_N)(1+\beta)C_i + \alpha_{i+1}A_{i+1} \ (i=1,...,N-1) \qquad (2)$$

Dla przejrzystości prezentacji w dalszej części pracy będziemy rozważać przypadek bez zwracania wyników.



Rys. 3. Diagram komunikacji i obliczeń w sieci liniowej ze zwracaniem wyników
Fig. 3. Communication-computation diagram for a chain with results returning

Sieć liniowa bez koprocesorów komunikacyjnych [14,17]

W tym przypadku sam procesor jest odpowiedzialny za prowadzenie komunikacji. Jednoczesne obliczanie i komunikowanie nie jest możliwe. Proces dystrybucji danych i obliczeń jest więc następujący: Element przetwarzający odbiera dane, dzieli je na część do lokalnej obróbki oraz część do odesłania sąsiadom. Następnie wysyła dane sąsiadom, a dopiero po tym sam zaczyna lokalne obliczenia. Rozdział obciążeń można wyliczyć stostując zmodyfikowany układ równań (1), w którym każdą parę nadawca - odbiorca wiążą równania:

$$\alpha_i A_i = (\alpha_{i+2} + ... + \alpha_N)C_{i+1} + \alpha_{i+1}A_{i+1} \ (i=1,...,N-2), \qquad \alpha_{N-1}A_{N-1} = \alpha_N A_N \qquad (3)$$

Sieć liniowa z koprocesorami komunikacyjnymi, transmisja z opóźnieniem [11]

W przedstawionych powyżej przypadkach zakładaliśmy, że czas transmisji x bajtów danych przez łącze j jest równy xC_j. Zakładamy teraz, że czas transmisji zawiera opóźnienie startowe R_j (ang. *startup*) wynikające z konieczności zapakowania danych do transmisji, przetworzenia odwołania do sieci przez system operacyjny, uzyskania dostępu do sieci itp. W celu wyznaczenia optymalnego rozdziału obliczeń musimy zmodyfikować układ równań (1), zmieniając pierwsze $N-1$ równań następująco:

$$\alpha_i A_i = R_i + (\alpha_{i+1} + \alpha_{i+2} + ... + \alpha_N)C_i + \alpha_{i+1}A_{i+1} \ (i=1,...,N-1) \qquad (4)$$

Konsekwencją powyższej zmiany jest to, że nie zawsze będzie istniało dopuszczalne rozwiązanie układu równań (4). Wynika to stąd, że wartości α_i nie mogą być ujemne. Gdy korzystając z równań (4) wyrazimy α_i jako funkcję liniową α_N, tzn.: $\alpha_i = k_i \alpha_N + l_i$, w ostatnim równaniu układu (1) otrzymamy:

$$V = \alpha_N (1 + \sum_{i=1}^{i=N-1} k_i) + \sum_{i=1}^{i=N-1} l_i$$

Stąd wnioskujemy, że niektóre wartości α_i mogą być ujemne. Oczywiście, takie rozwiązanie jest niedopuszczalne. Powyższa sytuacja wynika z faktu, iż wysłanie danych do wszystkich *N* procesorów może trwać dłużej niż rozesłanie zadania i obliczenia tylko na kilku procesorach położonych blisko inicjatora. Powstaje pytanie, jak wyznaczyć optymalną liczbę procesorów do wykonania obliczeń. Można to zrobić testując istnienie rozwiązania dopuszczalnego dla kolejnych liczb procesorów zmieniających się od 1 do *N*. Czas ten jest jednak krótszy, jeśli zastosuje się przeszukiwanie binarne. Ponieważ układ równań (4) można rozwiązać w czasie *O(N)*, optymalne rozwiązanie można znaleźć w czasie *O(N logN)*.

Sieć liniowa z koprocesorami komunikacyjnymi, transmisja potokowa [6]

Poprzednio zakładaliśmy, że komunikacja między elementami przetwarzającymi odbywa się w jednym kroku. Inicjator wysyłał wszystkie dane za jednym razem. Każdy z elementów przetwarzających najpierw odbierał wszystkie dane, a następnie odsyłał dalej zbędną część. Nie jest to efektywny sposób komunikacji, gdyż większość procesorów czeka długo na dane, które nie są potrzebne do rozpoczęcia ich części obliczeń. Inicjator może wysyłać dane wprost do każdego z procesorów za pośrednictwem koprocesorów komunikacyjnych elementów przetwarzających znajdujących się na drodze od inicjatora do odbiorcy. Załóżymy, że w jednej chwili możliwa jest transmisja tylko w jednym kierunku. Pośredniczący koprocesor odbiera całe dane, a następnie retransmituje je dalej. Z tego względu jednocześnie można transmitować na wszystkich parzystych i nieparzystych połączeniach, tworząc potok (ang. *pipeline*) dla przesyłanych danych. Zauważmy, że w takiej metodzie komunkacji czas obliczeń na procesorze *i* jest równy czasowi obliczeń na procesorze *i+1* plus czas transmisji danych. Czas transmisji obejmuje przesłanie do procesora *i-1*, z *i-1* do *i* oraz z *i* do *i+1*, gdyż każdy koprocesor transmituje tylko w jednym kierunku. W celu wyznaczenia rozdziału obciążeń należy zmienić układ równań (1):

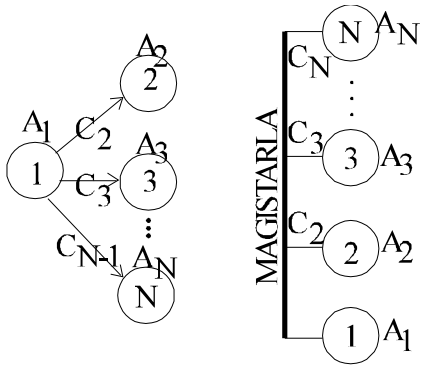
$$\begin{aligned} &\alpha_i A_i = \alpha_{i+1}(C_{i-2} + C_{i-1} + C_i) + \alpha_{i+1} A_{i+1} && (i=1,...,N-1) \\ &\alpha_i C_j \le \alpha_{i-1} C_{j+2} && (i=5,...,N,\ j=1,...,N-1) \\ &C_0 = C_{-1} = 0 \end{aligned} \qquad (5)$$

Powyższy układ równań i nierówności można rozwiązać metodą progamowania liniowego. Dla zaproponowanego algorytmu dopuszczalne rozwiązanie nie zawsze istnieje.

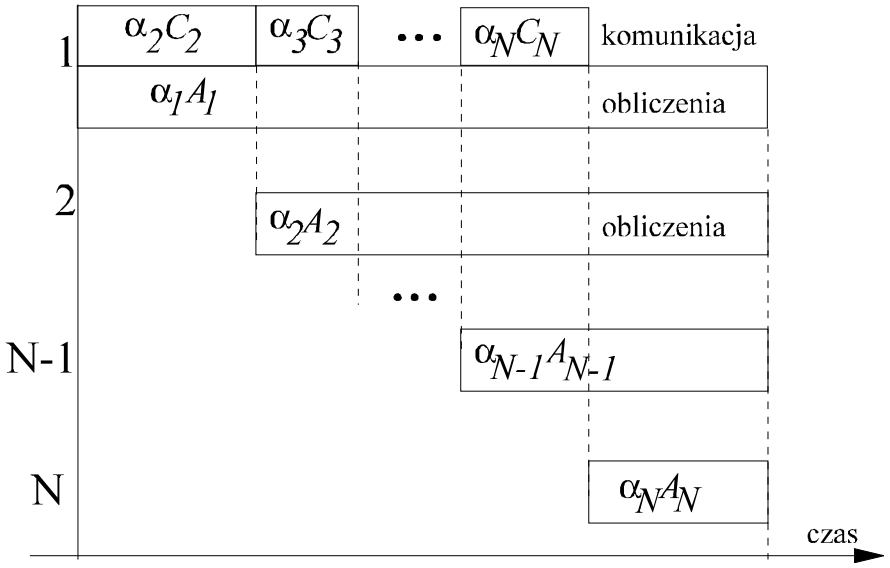
Na zakończenie tego rozdziału należy wspomnieć, że powyższe podejście można rozwinąć na sieć liniową, w której inicjator jest położony w środku (a nie na końcu) [14,17] oraz na pętlę elementów przetwarzających [11].

3. Sieć gwiazdowa i połączenie typu magistralowego

W tym rozdziale rozważamy dystrybuję obliczeń w sieci typu gwiazdowego i w sieci typu magistralowego (rys. 4). Zauważmy, że sieć typu gwiazdowego może być dobrym modelem dla rozproszonych obliczeń typu master-slave. Obie sieci łączy fakt, iż wskutek ich struktury komunikacja odbywa się tylko między jedną parą procesorów: inicjatorem i odbiorcą (rys.5). W dalszej części tego rozdziału dla zwięzłości mówimy tylko o gwieździe, mając na myśli także magistralę. Dla sieci magistralowej w podanych poniżej równaniach można ewentualnie przyjąć równe parametry wszystkich łącz komunikacjnych, gdyż wszystkie komunikacje odbywają się przez tą samą magistralę. Podobnie jak w sieci liniowej, podamy, jak znaleźć optymalny podział obliczeń w przypadku najprostszym, a następnie zademonstrujemy, jak można go uogólnić. Założymy, że jednoczesne rozsyłanie danych do wielu odbiorców nie jest możliwe.



Rys. 4. Struktura sieci gwiazdowej i sieci magistralowej
Fig. 4. Star and bus interconnections



Rys. 5. Diagram komunikacji i obliczeń w sieciach gwiaździstej i magistralowej
Fig. 5. Communication-computation diagram for a star and for a bus

Gwiazda z koprocesorami komunikacyjnymi [5,15]

Inicjator równolegle z lokalnymi obliczeniami wysyła dane do kolejnych sąsiadów. Po odebraniu danych procesory zaczynają liczyć, a inicjator rozpoczyna komunikację z następnym elementem obliczeniowym. Czas obliczeń na procesorze *i* jest zatem równy komunikacji do procesora następnego plus czas obliczeń na procesorze następnym. Optymalny podział obliczeń można wyznaczyć z układu równań:

$$\begin{gathered}\alpha_1 A_1 = \alpha_2 C_2 + \alpha_2 A_2 \\ \alpha_2 A_2 = \alpha_3 C_3 + \alpha_3 A_3 \\ \cdot\ \cdot\ \cdot \\ \alpha_i A_i = \alpha_{i+1} C_{i+1} + \alpha_{i+1} A_{i+1} \\ \cdot\ \cdot\ \cdot \\ \alpha_{N-1} A_{N-1} = \alpha_N C_N + \alpha_N A_N \\ \alpha_1 + \alpha_2 + \ldots + \alpha_i + \ldots + \alpha_N = V\end{gathered} \tag{6}$$

Podobnie jak dla (1), rozwiązanie powyższego układu równań można znaleźć w czasie *O(N)*. Czas obliczeń w sieci jest równy czasowi obliczeń w inicjatorze (rys. 5). Dysponując tym czasem możemy, podobnie jak w sieci liniowej, określić przyspieszenie, efektywość itp. [16]. Powstaje jednak wątpliwość, czy kolejność, w jakiej odwołujemy się do elementów przetwarzających, jest optymalna. Na przykład, czy dostarczając dane najpierw do procesorów najszybszych a potem do wolniejszych nie uzyskamy krótszego czasu obliczeń. Można wykazać [5], że optymalny czas obliczeń uzyskujemy wysyłając dane do procesorów w kolejności malejących prędkości łącz kominukacyjnych. Zaskakujący jest fakt, że w powyższym modelu, kolejność ta nie zależy od prędkości samych procesorów.

Gwiazda bez koprocesorów komunikacyjnych [5,15]

Elementy obliczeniowe bez koprocesora komunikacyjnego nie mogą jednocześnie wykonywać obliczeń i komunikować się. W związku z tym inicjator nie może rozpocząć obliczeń do chwili zakończenia wysyłania danych do ostatniego elementu obliczeniowego. W celu znalezienia optymalnego podziału obliczeń należy zmodyfikować pierwsze równanie układu (6):

$$\alpha_1 A_1 = \alpha_N A_N$$

Gwiazda z koprocesorami komunikacyjnymi, transmisja z opóźnieniem [11]

Rozważymy teraz przypadek, w którym czas komunikacji oprócz składnika liniowego zawiera też opóźnienie startowe. W celu znalezienia rozdziału obliczeń należy rozwiązać układ równań (7), w którym pierwsze *N-1* równań ma postać:

$$\alpha_i A_i = R_{i+1} + \alpha_{i+1} C_{i+1} + \alpha_{i+1} A_{i+1} \quad (i=1,\ldots,N) \tag{7}$$

Konsekwencją powyższej modyfikacji jest, że nowy układ równań (7) nie zawsze ma rozwiązanie, tj. $\alpha_i \geq 0$. Przyczyną jest to, że nim ostatni procesor otrzyma dane, inicjator (i ewentualnie kilka dodatkowych maszyn) zakończy obliczenia. W takiej sytuacji należy użyć mniejszej liczby procesorów. Optymalną liczbę procesorów można wyznaczyć w czasie

O(N logN), testując układ równań (7). Wskazanie optymalnej kolejności transmisji nie jest już jednak tak oczywiste. Naturalnie, gdy łącza komunikacyjne są takie same, można wykazać [11], że należy najpierw wysyłać dane do procesorów szybszych.

Przedstawiona metoda może być łatwo rozszerzona na sieci o strukturze drzewiastej [1,4,15], także dla komunikacji potokowej [7]. Niestety, zarówno dla gwiazdy, jak i dla magistrali, w przypadku z opóźnieniem startowym, NP-trudny jest problem wyznaczenia optymalnej kolejności dystrybucji danych [11]. Jeśli jednak sekwencja jest znana, to optymalny podział obliczeń można wyznaczyć w sposób analogiczny do przedstawionego. Dla połączenia magistralowego analizie poddano także przypadek z wieloma zadaniami jednorodnymi [20] oraz przypadek ze zmienną prędkością łącz i procesorów [21].

4. Krata

W niniejszym rozdziale zademonstrujemy, jak można zastosować koncepcję zadania jednorodnego do wyznaczenia rozdziału obciążeń w dwuwymiarowej sieci typu krata (ang. *mesh*) (rys. 6). Rozwiązanie problemu jest ściśle związane z metodą komunikacji zastosowaną do dystrybucji danych. Analiza efektywności systemu wykorzystującego tryb *store-and-forward* została zawarta w [9]. Tutaj zastosujemy metodę dystrybucji opartą na algorytmie komunikacji Petersa i Syski [18], wykorzystującym komunikację w trybie *wormhole routing* lub podobnym. Algorytm komunikacji zakłada, że krata ma rozmiar $5^{N/2} \times 5^{N/2}$ oraz że istnieją połączenia między przeciwległymi krawędziami, jest to więc torus. Jako *aktywny* określać będziemy procesor, który już otrzymał dane. Algorytm Petersa-Syski składa się z dwóch wykonywanych naprzemiennie "ruchów" (rys.6). W pierwszym z każdego aktywnego procesora dane wysyłane są w czterech kierunkach na raz ruchem konika szachowego: o jednostkę w przód, a następnie o dwie jednostki w lewo. W drugim każdy aktywny procesor wysyła pakiety ruchem w formie krzyża o jedną jednostkę w przód. Jednostka zmniejsza się w każdym kolejnym powtórzeniu opisanych dwóch kroków. Cechą zastosowanego algorytmu dystrybucji danych jest to, iż każdy ruch zwiększa liczbę aktywnych procesorów czterokrotnie, czyli o maksymalną możliwą liczbę. Po i krokach 5^i procesorów jest aktywnych. Zakładamy, że wszystkie połączenia i procesory są identyczne oraz opisane odpowiednio parametrami R, C, A. Elementy obliczeniowe mogą jednocześnie liczyć i komunikować się z 4 sąsiadami. Procesory osiągnięte w tym samym kroku algorytmu dystrybucji nazwiemy *warstwą*. Każdy z $4 \cdot 5^{i-1}$ procesorów warstwy i przetwarza lokalnie α_i danych. Optymalny podział obliczeń można wyznaczyć z układu równań:

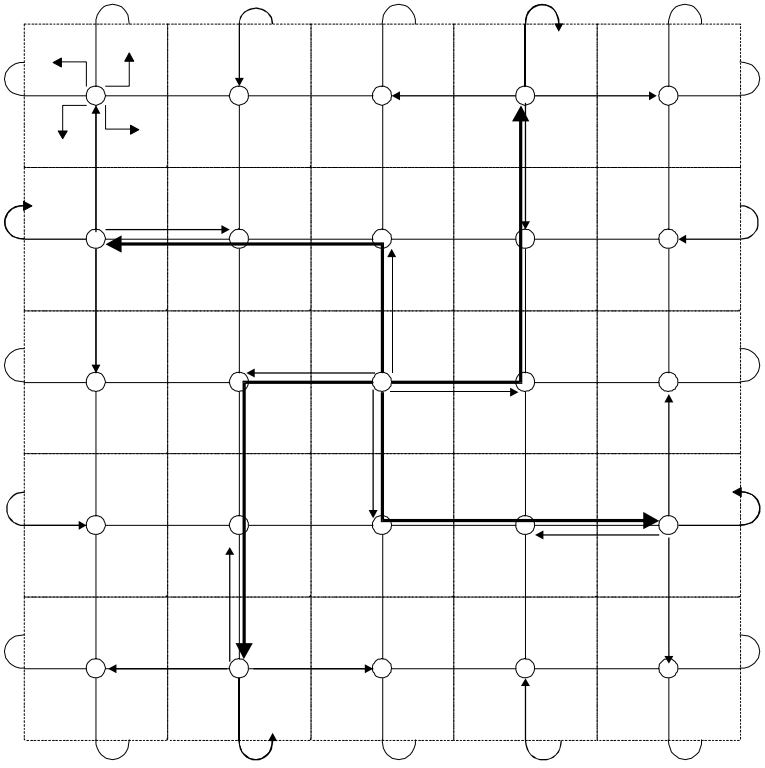
$$
\begin{aligned}
&\alpha_{N-1} A = R + C\alpha_N + A\alpha_N \\
&\alpha_{N-2} A = R + C(\alpha_{N-1} + 4\alpha_N) + A\alpha_{N-1} \\
&\alpha_{N-3} A = R + C(\alpha_{N-2} + 4\alpha_{N-1} + 20\alpha_N) + A\alpha_{N-2} \\
&\ldots \\
&\alpha_{N-i} A = R + C(\alpha_{N-i+1} + 4\sum_{j=2}^{i} \alpha_{j-2} 5^{j-2}) + A\alpha_{N-i+1}
\end{aligned}
\qquad (8)
$$

. . .

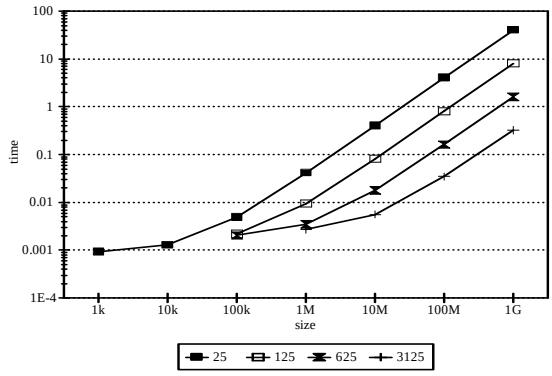
$$\alpha_0 A = R + C(\alpha_1 + 4\sum_{j=2}^{N} \alpha_{j-2} 5^{j-2}) + A\alpha_1$$

$$\alpha_0 + 4\sum_{i=1}^{N} 5^{i-1} \alpha_i = V$$

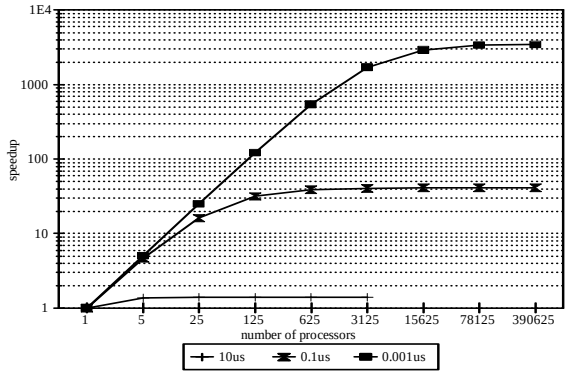
Podobnie jak w innych przypadkach, nie każda liczba warstw jest dopuszczalna. Wysłanie danych do ostatniej warstwy może trwać dłużej niż obliczenia na kilku pierwszych warstwach. Z tego względu optymalną liczbę warstw i optymalny podział obliczeń można znaleźć w czasie *O(N logN)*. Poniżej zamieszczamy przykład zastosowania wyników rozwiązania układu równań do oceny efektywności architektury krata, w której zastosowano algorym Petersa-Syski. Rysunek 7 prezentuje zależność czasu wykonania od wolumenu danych *V* dla różnych liczb użytych procesorów; przyjęto $A=10^{-6}$s/bajt, R=500μs, $C=10^{-12}$s/bajt. Z wykresu można wnioskować, że dla małych wolumenów nie jest możliwe zastosowanie dużych liczb procesorów, gdyż transmisja do nich trwa dłużej niż obliczenia. Dla dużych wolumenów danych czas obliczeń dominuje nad czasem transmisji, stąd liniowa zależność czasu od wolumenu. Na rys.8 zamieszczono zależność przyspieszenia od liczby procesorów i prędkości komunikacji (wyrażonej w μs/bajt); przyjęto $V=10^6$bajtów, $A=10^{-6}$s/bajt, R=5μs. Z wykresu można wnioskować, iż nawet dla szybkich sieci (C=0.01μs/bajt) przy użyciu więcej niż 625 procesorów trzeba liczyć się ze znacznym spadkiem efektywności.



Rys. 6. Dystrybucja danych w sieci typu krata
Fig. 6. Data distribution in mesh connected torus



Rys. 7. Czas wykonania w funkcji *N, V*
Fig. 7. Execution time vs. the number of processors *N* and data volme *V*



Rys. 8. Przyspieszenie w funkcji *N, C*
Fig. 8. Speedup vs. the number of processors *N* and communication rate *C*

5. Hiperkostka

Hiperkostka (ang. *hypercube*) o rozmiarze *d* zawiera $N=2^d$ elementów przetwarzających. Każdy z nich może być oznaczony etykietą, która jest liczbą binarną z zakresu $[0,2^d-1]$. Połączenia komunikacyjne łączą elementy przetwarzające z etykietami różniącymi się na dokładnie jednej pozycji. Zakładamy, że połącznia komunikacyjne i procesory są identyczne oraz opisują je parametry, odpowiednio, *C, R, A*. Ponadto, elementy obliczeniowe wyposażone są w koprocesory komunikacyjne i możliwe jest jednoczesne nadawanie do *d* sąsiadów. Dla hiperkostki podano wiele algorytmów komunikacyjnych. W tej prezentacji posłużymy się prostą metodą opartą na komunikacji typu store-and-forward. Obliczenia i komunikacja przebiegają następująco. Inicjator oblicza lokalnie α_0 danych, a pozostałość odsyła w równych częściach do *d* sąsiadów. Sąsiedzi inicjatora tworzą warstwę 1. *d* procesorów warstwy 1 przetwarza lokalnie α_1 danych każdy, a pozostałość odsyłają w równych porcjach do *d-1* nieaktywnych sąsiadów. (*d-2)d/2* procesorów warstwy 2 przetwarza α_2 danych każdy, a pozostałość odsyłają do *d-2* nieaktywnych sąsiadów. Proces ten jest powtarzany aż do osiągnięcia ostatniego procesora w warstwie *d*. Zauważmy, że w warstwie *i* jest $\binom{d}{i}$ procesorów, które otrzymują dane od *i* sąsiadów, a pozostałość wysyłają do *d-i* nieaktywnych sąsiadów. W celu wyznaczenia optymalnego rozdziału obliczeń należy rozwiązać układ równań:

$$\alpha_0 A = R + \frac{(1-\alpha_0)C}{d} + \alpha_1 A$$

$$\alpha_1 A = R + \frac{(1-\alpha_0 - d\alpha_1)C}{d(d-1)} + \alpha_2 A$$

. . . .

$$\alpha_i A = r + \frac{\left(1 - \alpha_0 - \ldots - \binom{d}{i}\alpha_i\right)C}{\binom{d}{i}(d-i)} + \alpha_{i+1}A \tag{11}$$

$$\ldots$$

$$\alpha_{d-1}A = R + (1 - \alpha_0 - \ldots - d\alpha_{d-1})\frac{(C+A)}{d} = R + \frac{\alpha_d (C+A)}{d}$$

$$\sum_{i=0}^{d}\binom{d}{i}\alpha_i = V$$

Jeżeli powyższy układ *d* równań i *d* niewiadomych ma rozwiązania ujemne, to znaczy to, że nie wszystkie warstwy mogą brać udział w obliczeniach. Maksymalną liczbę warstw można wyznaczyć w czasie *O(d logd)* przy użyciu przeszukiwnia binarnego, testując układ równań (11).

6. Zakończenie

W pracy przedstawiliśmy zastosowanie koncepcji zadania jednorodnego do rozwiązania problemu równoważenia obciążeń. Dzięki temu możliwe stało się podanie prostych, dokładnych rozwiązań dla problemów, które obecnie rozwiązuje się używając heurystyk. Zaproponowana metoda została zastosowana do szerokiej klasy architektur systemów rozproszonych. Istotnym uwzględnianym przez nią elementem jest stosowany algorytm komunikacyjny. Uzyskane wyniki mogą posłużyć do oceny efektywności rozproszonych systemów komputerowych. Kolejnym etapem w rozwoju metody zadania jednorodnego jest jej praktyczne zastosowanie. Badania w tym kierunku są prowadzone.

LITERATURA

[1] Bataineh S., Hsiung T., Robertazzi T.G.: Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job, IEEE Transactions on Computers, vol. 43, No. 10, Oct. 1994, pp.1184-1196.

[2] Bataineh S., Robertazzi T.G.: Distributed Computation for a Bus Network with Communication Delays, Proceedings of the 1991 Conference on Information Sciences and Systems, The Johns Hopkins University, Baltimore MD, March 1991, pp. 709-714.

[3] Bataineh S., Robertazzi T.G.: Bus Oriented Load Sharing for a Network of Sensor Driven Processors, special issue on Distributed Sensor Networks of the IEEE Transactions on Systems, Man and Cybernetics, Sept. 1991, vol. 21, No. 5, pp.1202-1205.

[4] Bataineh S., Robertazzi T.G.: Ulitmate Performance Limits for Networks of Load Sharing Processors, Proceedings of the 1992 Conference on Information Sciences and Systems, Princeton NJ., March 1992, pp.794-799.
[5] Bharadwaj V., Ghose D., Mani V.: Optimal Sequencing and Arrangement in Distributed Single-Level Tree Networks with Communication Delays, IEEE Transactions on Parallel and Distributed Systems, vol.5, No. 9, Sept. 1994, pp. 968-976.
[6] Bharadwaj V., Ghose D., Mani V.: An Efficient Load Distribution Strategy for Distributed Linear Network of Processors with Communication Delays, Computers and Mathematics with Applications, 1995, vol. 29, No. 9, pp.95-112.
[7] Bharadwaj V., Ghose D., Mani V.: Multi-installment Techniques in Tree Networks, IEEE Transactions on Aerospace & Electronic Systems, vol. 31, No. 2, Apr. 1995, pp.555-567.
[8] Blanc J.-Y., Trystram D.: Implementiation of parallel numerical routines using broadcast communication schemes, w: E. Burkhart (edytor) Lecture Notes in Computer Science 457, Proceedings of CONPAR 90-VAPP IV, Joint International Conference on Vector and Parallel Processing, Proceedings, Zurich, Switzerland, Springer-Verlag, Berlin, 1990, pp. 467-478.
[9] Błażewicz J., Drozdowski M.: Performance limits of two-dimensional network of load-sharing processors, 1996, zaakceptowane do druku w Foundations of Computing and Decision Sciences.
[10] Błażewicz J., Drozdowski M.: Scheduling Divisible Jobs on Hypercubes, Parallel Computing, 1995, vol. 21, pp.1945-1956.
[11] Błażewicz J., Drozdowski M.: Scheduling divisible jobs with communication startup costs, Instytut Informatyki Politechniki Poznańskiej, raport RA-94/006.
[12] Błażewicz J., Drozdowski M., Guinand F., Trystram D.: Scheduling under architectural constraints, Instytut Informatyki Politechniki Poznańskiej, raport RA-003/95.
[13] Boryczko K., Bubak M., Kitowski J., Mościński J., Słota R.: Rozproszone obliczenia na sieci stacji roboczych i komputerze Convex C2310, Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 24, Nr kol. 1222, ss.63-75.
[14] Cheng Y.C., Robertazzi T.G.: Distributed Computation with Communication Delays, IEEE Transactions on Aerospace and Electronic Systems, vol.24, No. 6, Nov. 1988, pp.700-712.
[15] Cheng Y.C., Robertazzi T.G.: Distributed Computation for a Tree Network with Communication Delays, IEEE Transactions on Aerospace and Electronic Systems, vol.26, No. 3, May 1990, pp.511-516.
[16] Ghose D., Mani V.: Distributed Computation with Communication Delays: Asymptotic Performance Analysis, Journal of Paralllel and Distributed Computing, 1994, vol. 23, pp.293-305.
[17] Mani V., Ghose D.: Distributed Computation in Linear Networks: Closed-Form Solutions, IEEE Transactions on Aerospace & Electronic Systems, vol. 30, No. 2, Apr. 1994, pp.471-483.
[18] Peters J.G. , Syska M.: Circuit-Switched Broadcasting in Torus Networks, przyjęte do druku w IEEE Transactions on Parallel and Distributed Syststems, 1995.
[19] Robertazzi T.G.: Processor Equivalence for a Linear Daisy Chain of Load Sharing Processors", IEEE Transactions on Aerospace and Electronic Systems, vol. 29, No. 4, Oct. 1993, pp. 1216-1221.
[20] Sohn J., Robertazzi T.G.: A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks, University os Stony Brook, CEAS Technical Report 697, 1994.

[21] Sohn J., Robertazzi T.G.: An Optimum Load Sharing Strategy for Divisible Jobs with Time-Varying Processor Speed and Channel Speed, University os Stony Brook, CEAS Technical Report 706, 1995.
[22] Williams R.: Performance of dynamic load balancing algorithms for unstructured mesh calculations, Concurrency:Practice and Experience, vol. 3, No. 5, Oct. 1991, pp. 457-481.

Recenzent: Dr hab. inż. Zbigniew Czech

Wpłynęło do Redakcji 18 grudnia 1995 r.

Abstract

In this paper we present the most important results in the divisible job scheduling. The divisible job can be divided into parts of arbitrary size processed separately and independently by remote processors. The model of the computational process includes both the time of data distribution to the remote sites as well as the processing time. We presented results for the following computer architectures: chain (fig.1), star, bus system (fig.4), mesh (fig.6) and hypercube networks. A variety of communication characteristics were analyzed: processing elements with/without communication co-processors, linear communication time vs. communication with startup time included, communication to one and to many receivers, store-and-forward vs. wormhole routing etc. The optimal distribution of the computation is found by solving a set of linear equations (1) for a chain, (6) for a star and a bus, (8) for a mesh, (11) for a hypercube. The divisible job theory allows also for the analysis of the computer system performance e.g. for mesh-connected torus illustrated in fig.7 and fig.8.