Problemy równoległej optymalizacji dyskretnej

PROBLEMY RÓWNOLEGŁEJ OPTYMALIZACJI DYSKRETNEJ

Redaktorzy:

Leon Słomiński

Ignacy Kaliszewski

1. Wprowadzenie

Niniejsza książka prezentuje wyniki prac prowadzonych w Zakładzie Programowania Matematycznego Instytutu Badań Systemowych PAN w latach 1986-92, w ramach tematu badawczego "*Optymalizacja na strukturach kombinatorycznych z uwzględnieniem obliczeń równoległych*". Podjęcie tego tematu, którym kierował pierwszy z współredaktorów, poprzedziło roczne wspólne seminarium Zakładu Programowania Matematycznego IBS PAN oraz Pracowni Metod Numerycznych Instytutu Podstaw Informatyki PAN. Niektóre pomysły dyskutowane podczas seminarium znalazły swoje odbicie w treści tej książki.

Celem książki jest przybliżenie polskiemu Czytelnikowi problematyki obliczeń równoległych w zadaniach optymalizacji, a specjalnie w zadaniach optymalizacji dyskretnej oraz uzupełnienie, chociażby częściowe, luki jaka jest w tym zakresie zauważalna w krajowej literaturze. Z publikacji w języku polskim poświęconych w całości lub w części metodom i algorytmom równoległym optymalizacji dyskretnej można wymienić pozycje [1 ÷ 3]. Postawiony cel chcemy osiągnąć przez zaprezentowanie wyboru zadań optymalizacyjnych z zakresu naszych zainteresowań i przedstawienie, na ich przykładzie, problemów związanych z konstrukcją algorytmów równoległych. Problemy te ukazujemy na tle znacznie szerszego wachlarza zagadnień związanych z obliczeniami równoległymi.

Książka ma następujący układ. Rozdział drugi zawiera podstawowe pojęcia związane z obliczeniami równoległymi i z maszynami równoległymi. Przedstawiono w nim najważniejsze modele obliczeń równoległych, typowe architektury maszyn równoległych i ich charakterystyki, a także podstawowe wyniki teorii złożoności dla

algorytmów równoległych. Dopełnieniem tego rozdziału jest omówienie nowatorskiej realizacji sprzętowej - transputera, którego sieci wydają się szczególnie przydatne do rozwiązywania zadań optymalizacji dyskretnej za pomocą ogólnych i wąsko ukierunkowanych algorytmów równoległych. W rozdziale trzecim przedstawiono algorytmy równoległe dla rozwiązywania zadania wyznaczania dendrytu minimaksowego w grafie skierowanym z wagami na łukach. W celu pokazania związku, który zachodzi między złożonością algorytmu i wybranym modelem maszyny równoległej, użyto różnych modeli maszyny i różnych pierwowzorów algorytmów sekwencyjnych. Rozdział czwarty przedstawia algorytmy wyznaczania elementu maksymalnego w skończonym zbiorze wektorów oraz opis ich realizacji na sieci transputerów, wraz z wynikami testów numerycznych. W rozdziale piątym opisano asynchroniczny algorytm równoległy, do rozwiązywania algebraicznego zagadnienia k - przydziału, przedstawiono realizację tego algorytmu na sieci transputerów oraz wyniki eksperymentu numerycznego. Kolejny, szósty rozdział prezentuje język programowania równoległego MODEST. Jest to język ogólnego zastosowania zawierający mechanizmy do uruchamiania procesów i kontroli przebiegu obliczeń równoległych. Dwa dodatki zawierają: Dodatek 1 - Polsko - angielski słownik nazw i pojęć z zakresu obliczeń równoległych, Dodatek 2 - Listę artykułów i raportów ZPM IBS PAN, które ukazały się w latach 1986 - 1992 i które dotyczą problematyki tej książki.

Warszawa, styczeń 1993.

Leon Słomiński
Ignacy Kaliszewski

Literatura

1. Błażewicz J. (1988): Złożoność obliczeniowa problemów kombinatorycznych. WNT, Warszawa.
2. Słomiński L., Kaliszewski I. (1988): "Problemy obliczeń równoległych". *Prace IBS PAN*, 168, Warszawa.
3. Sysło M.M. (1985): "Maszyny i algorytmy równoległe". *Raport Instytutu Informatyki Uniwersytetu Wrocławskiego*, 154, Wrocław.

2. Obliczenia równoległe

2.1. Wstęp

Rozdział ten nosi charakter ogólnego wprowadzenia do problematyki obliczeń równoległych, rozumianych jako zespół rozwiązań sprzętowych i softwarowych służących uzyskaniu mocy obliczeniowych istotnie większych od możliwych do uzyskania na maszynach sekwencyjnych, skracających tym samym czas potrzebny dla rozwiązywania odrębnych zadań, szczególnie tych o dużych rozmiarach i o wysokiej złożoności obliczeniowej. Przedstawione informacje i wyniki pochodzą z wielu źródeł, które przytaczamy na końcu rozdziału bez odwoływania się do nich w tekście. Uzupełnieniem rozdziału i literatury źródłowej są dodatkowe wskazówki: lista ważniejszych książek i bibliografii oraz lista tytułów czasopism, które poświęcają swoje łamy, w całości lub w dużym stopniu, interesującej nas problematyce.

2.2. Uwagi ogólne

Rozwojowi obliczeń równoległych towarzyszył, a w mniejszym natężeniu wciąż towarzyszy, interesujący spór o to, czy stałemu zpotrzebowaniu na coraz większe moce obliczeniowe systemów komputerowych nie można sprostać przez technologiczne, architektoniczne i programowe doskonalenie maszyn sekwencyjnych. Wiele zastrzeżeń i mylnych hipotez dotyczących celowości i możliwości rozwijania maszyn równoległych wynikało z przekonania, że czas zużyty na organizację obliczeń w tych systemach (obejmujący m.in. czasy: komunikacji, synchronizacji i przydziału zadań procesorom) będzie skutecznie niwelował spodziewane przyśpieszenia obliczeń. Wątpliwości te zostały rozstrzygnięte na korzyść maszyn

wieloprocesorowych, co potwierdza zarówno ich różnorodna i bogata oferta rynkowa, jak i duża liczba zaawansowanych projektów badawczych i konstrukcyjnych w tej dziedzinie. Obliczenia równoległe stały się tym samym rzeczywistością i chociaż wiele problemów dotyczących szczególnie języków oraz kompilatorów, a także oprogramowania użytkowego, pozostaje wciąż nierozwiązanych, to dynamiczny trend rozwojowy należy uznać za nieodwracalny.

Tematyka obliczeń równoległych jest obecna w programach badawczych wielu laboratoriów naukowych, a także w praktyce gospodarczej. O jej obecności w nauce świadczy rosnąca długość listy zawierającej nowe tytuły czasopism poświęconych niemal wyłącznie tej tematyce (np. *Concurrency, International Journal of Parallel Programming, Journal of Parallel and Distributed Computation, Parallel Computing, Supercomputer, Transputer Communications*); wiele renomowanych czasopism z dziedziny informatyki, matematyki stosowanej i optymalizacji udziela, w ostatnim pięcioleciu, problemom obliczeń równoległych coraz więcej uwagi (np. *Algorithmica, Annals of Discrete Mathematics, Computing, IEEE on Computing, Information Processing Letters, Journal of Algorithms, Operations Research, ORSA Journal on Computing*). Rocznie odbywa się kilkanaście dużych konferencji międzynarodowych poświęconych teorii i zastosowaniom superkomputerów, maszyn wektorowych i systolicznych, a także maszyn bardzo wysokiego stopnia równoległości. Liczbę publikacji na temat różnych aspektów obliczeń równoległych, na koniec 1992 roku, można oceniać na kilka tysięcy.

O znaczeniu obliczeń równoległych dla przemysłu i szeroko rozumianej praktyki gospodarczej świadczyć może liczba superkomputerów i minisuperkomputerów będących w użytkowaniu, a także dynamika nowych ofert rynkowych. Liczbę superkomputerów użytkowanych, u schyłku 1992 r, w przemyśle, w energetyce

jądrowej, w ośrodkach naukowych i na uniwersytetach ocenia się na ponad 500 i liczba ta rośnie bardzo szybko. Wiodącymi przemysłami w tej dziedzinie są: aeronautyka, energetyka, przemysł samochodowy i lotniczy, chemia i przemysł farmaceutyczny, przemysł naftowy i przemysł nowych technologii mikrobiologicznych. Tabela 2.1 przedstawia niewielki fragment z obszernej oferty rynkowej maszyn równoległych.

Tabela 2.1. Maszyny równoległe dostępne na rynku.

MASZYNA	LICZBA PROCESORÓW	ARCHITEKTURA	OPROGRAMOWANIE
ALLIANT FX/8	20	KOMUTATOR/SZYNA	UNIX/FORTRAN/C
AMETEK SYSTEM	256	HIPERSZEŚCIAN	HOS/FORTRAN/C
CDC CYBERPLUS	256	PIERŚCIEN/SZYNA	NOS/VM
CRAY-XMP, CRAY-3	do 16	SZYNA	UNIX
FPS 164	16	SZYNA	UNIX/FORTRAN
GOODYEAR MMP	16384	NAJBLIŻSZY SĄSIAD	PARA PASCAL
ICL MINI DAP	1024	NAJBLIŻSZY SĄSIAD	FORTRAN
INTEL IPSC	128	HIPERSZEŚCIAN	FORTRAN
MEIKO COMP. SURFACE	150	DYNAMICZNA	OCCAM/FORTRAN
MYRIAS 4000	1024	SKUPIENIA	UNIX/FORTRAN/C
CM-2	65536	HIPERSZEŚCIAN	UNIX/FORTRAN/C
NCUBE Ncube 10	1024	HIPERSZEŚCIAN	UNIX/FORTRAN/C
SAXPY MATRIX-1	32	SYSTOLICZNA	VMS/FORTRAN/C

Zadania numeryczne, szczególnie te o dużych rozmiarach i wysokiej złożoności obliczeniowej, potrzebują do rozwiązania zarówno maszyn cyfrowych o dużych mocach, jak i szybkich algorytmów. Maszyny wieloprocesorowe i algorytmy dla nich mogą w jakimś stopniu sprostać stawianym wymaganiom.

Rozważymy bardzo trudne i dobrze znane w literaturze zadanie optymalizacji dyskretnej nazywane kwadratowym zadaniem przy-

działu (KZP). Zadanie to można interpretować jako zadanie wyboru optymalnej lokalizacji obiektów (o zadanych wielkościach strumieni powiązań) na działkach (o zadanych odległościach). Rozpatrzymy KZP w postaci zadania programowania binarnego z kwadratową funkcją celu. Z punktu widzenia nakładu obliczeń zadanie to charakteryzuje liczba zmiennych zero-jedynkowych $n = m^2$, gdzie: m = liczba obiektów (działek). Rozważmy dwa przypadki: przypadek zadania małego rozmiaru - $m_1 = 60$ (liczba zmiennych binarnych $n_1 = 3600$), i przypadek zadania średniego rozmiaru - $m_2 = 200$ ($n_2 = 4*10^4$). Załóżmy, że mamy do dyspozycji dwie różne maszyny o mocy obliczeniowej odpowiednio: 100 Mips (100 Milion integer operations per second) i 2000 Mips (moc nieosiągalna przez maszyny sekwencyjne). Większość znanych algorytmów heurystycznych (sekwencyjnych i równoległych) ma dla KZP, w przypadku ogólnym, nakład obliczeń proporcjonalny do n^3. Dla przypadków szczególnych są znane algorytmy heurystyczne o nakładzie obliczeń rosnącym liniowo ze wzrostem n.

Tabela 2.2. Efektywność obliczeń dla KZP.

m	MASZYNA	ZŁOŻONOŚĆ ALGORYTMU	
		$O(n^3)^*$	$O(n)^*$
60	100 Mips	6,5 godz.	1,30 msek.
	2000 Mips	0,6 godz.	0,09 msek.
200	100 Mips	9000 godz.	20,00 msek.
	2000 Mips	440 godz.	1,00 msek.

Tabela 2.2 podaje oceny czasu rozwiązania zadania dla naszych hipotetycznych maszyn i algorytmów. Dodajmy, że rozwiązanie ścisłe KZP o liczbie $m = 12$, metodą podziału i oszacowań, zajęło 3,12 sekund superkomputera CRAY-XMP/4.

* W książce tej stosujemy następującą symbolikę dla określenia rzędu funkcji f. Funkcja $f(N)$ jest co najwyżej rzędu $h(N)$, co zapisujemy $O(h(N))$, jeżeli istnieje taka stała c, że $|f(N)| \leq c|h(N)|$ dla prawie wszystkich wartości argumentu N. Funkcja $f(N)$ jest co najmniej rzędu $h(N)$ i zapisujemy to $\Omega(h(N))$ jeżeli istnieje taka stała c, że $|f(N)| \geq c|h(N)|$ dla prawie wszystkich N. Funkcja $f(N)$ jest dokładnie rzędu $h(N)$, co zapisujemy $\Theta(h(N))$, jeżeli istnieją stałe c i c' takie, że $c|h(N)| \leq |f(N)| \leq c'|h(N)|$ dla prawie wszystkich N.

2.3. Podstawowe pojęcia

W paragrafie tym wprowadzimy najważniejsze, z naszego punktu widzenia, pojęcia z zakresu obliczeń równoległych, których wieloznaczność może wzbudzać wątpliwości: proces, procesor, maszyna równoległa, obliczenia równoległe, obliczenia współbieżne. Ich bardziej wyczerpujący zestaw znajdzie Czytelnik w Dodatku 1 (pojęcia przytaczane lub przywoływane w dalszym ciągu tego rozdziału, znajdujące się w Dodatku 1, są wyróżnione kursywą).

Proces to autonomiczny ciąg działań maszyny cyfrowej. *Proces obliczeniowy* to ciąg zdarzeń w maszynie cyfrowej mających na celu obliczenie pewnej wartości. *Przetwarzanie* to proces, którego obiektem działania nie muszą być liczby. Proces może się składać z innych procesów, nazywanych *podprocesami* (*zadaniami*).

Procesor (często będzie oznaczany skrótem P) to każde urządzenie cyfrowe do przetwarzania danych. Z naszego punktu widzenia procesorem może być, *element operacyjny* - prosty układ do wykonywania jednego rodzaju operacji i kilku przesyłań, *jednostka funkcyjna* - zespół układów do wykonywania wyspecjalizowanych działań, rozbudowana jednostka centralna, a także jednostka centralna z własną pamięcią i sterowaniem. Element operacyjny bę-

dzie typowym ogniwem *maszyny systolicznej* i *maszyn wysokiego stopnia równoległości*, np. *maszyny Boltzmanna*. Wyspecjalizowane jednostki funkcyjne (zwykle kaskadowe) z reguły wchodzą w skład *maszyny wektorowej*. Jednostki centralne są najczęściej procesorami maszyn *MIMD*. Przykładem takiego procesora jest *transputer*, któremu poświęcamy odrębny paragraf.

Obliczenia równoległe to procesy realizowane jednocześnie przez wiele procesorów w celu szybszego rozwiązania określonego zadania obliczeniowego. W tych obliczeniach uwaga i nacisk są skupione na architekturze wieloprocesorowej, uwzględniającej koordynację działań procesorów, oraz na oprogramowaniu umożliwiającym efektywną organizację procesów (systemy operacyjne, języki programowania, algorytmy).

Maszyna równoległa to wieloprocesorowa maszyna cyfrowa przygotowana pod względem architektury i oprogramowania systemowego do wykonywania obliczeń równoległych. W bliskim związku z pojęciem obliczeń równoległych pozostaje pojęcie obliczeń współbieżnych.

Obliczenia współbieżne to procesy realizowane jednocześnie przez wiele procesorów w celu uzyskania większej wydajności maszyny przy jednoczesnym rozwiązywaniu wielu różnych zadań. Pojęcie to pozostaje w związku z pojęciami wielozadaniowości i wieloprogramowości w obliczeniach.

Z tego co powiedzieliśmy wynika, że procesy składające się na obliczenia równoległe można zawsze traktować jako procesy współbieżne, ale nie odwrotnie.

2.4. Modele obliczeń równoległych

Model maszyny cyfrowej - jej wyidealizowany, abstrakcyjny obraz - jest wygodnym narzędziem do analizy procesu obliczeń i do

szacowania czasu oraz pamięci niezbędnych do rozwiązywania różnych zadań. Dla maszyn sekwencyjnych rolę tę spełnia model *RAM* (*R*andom *A*ccess *M*achine). Teoria złożoności obliczeń sekwencyjnych oparta jest głównie na tym modelu. W przypadku maszyn równoległych wybór modelu obliczeń komplikuje się z wielu powodów, z których najważniejszym jest konieczność uwzględnienia komunikacji między procesorami. W chwili obecnej nie ma uniwersalnego modelu maszyny równoległej, lecz istnieje wiele modeli obliczeń o różnym zastosowaniu. Poniżej przedstawiamy najważniejsze z nich.

Model P-RAM (Parallel-*RAM*), nazywany również *parakomputerem*, jest naturalnym uogólnieniem modelu *RAM*. Nieskończona liczba synchronicznych procesorów ma jednoczesny dostęp do wspólnej pamięci o nieograniczonej pojemności. Procesory komunikują się między sobą przez wspólną pamięć, czas komunikacji nie zależy od liczby procesorów. Również czas wykonania każdej operacji ze standardu, którym dysponuje procesor, nie zależy od liczby procesorów. W chwili rozpoczęcia obliczeń aktywny jest jeden wybrany procesor, który może wykonać standardową operację lub uaktywnić inny procesor. Taką samą możliwości ma każdy aktywny procesor. Obliczenia kończą się z chwilą zatrzymania procesora, który je zapoczątkował.

Zauważmy, że uruchomienie p procesorów wymaga, w najlepszym przypadku, rzędu $\log p$ jednostek czasu (w tej książce symbol log oznacza logarytm o podstawie 2).

Znanych jest kilka odmian modelu *P-RAM* różniących się zasadami korzystania ze wspólnej pamięci. Model, w którym procesory mogą w sposób nieograniczony czytać z tego samego miejsca pamięci, ale nie mogą jednocześnie zapisywać w to samo miejsce pamięci, jest nazywany modelem *P-RAM-CREW* (*P-RAM* - *C*oncurrent *R*ead *E*xclusive *W*rite). Innym wariantem jest model *P-RAM-*

CRCW (*P-RAM* - Concurrent Read Concurrent Write), który dopuszcza jednoczesne czytanie i pisanie z i w to samo miejsca pamięci, z tym że przy próbie jednoczesnego zapisu obowiązuje pewien, z góry ustalony, protokół postępowania.

Model MP-RAM (Message Passing - *RAM*), nazywany również *ultrakomputerem*, odróżnia się od modelu *P-RAM* tym, że nie ma wspólnej pamięci. Każdy procesor, ich liczba pozostaje nieograniczona, dysponuje własną nieskończoną pamięcią. Procesory komunikują się między sobą przez połączenia o stałej lub zmiennej topologii, których przykłady podamy dalej. Model ten pozwala uwzględniać w analizie procesu obliczeń nakłady na komunikację.

Model BDSS (Bardzo Dużej Skali Scalenia) można traktować jako odmianę modelu *MP-RAM*, mającą oddawać lepiej specyfikę maszyn równoległych realizowanych w technologii modułów *BDSS*. Model ten spotyka się w wielu odmianach, różniących się szczegółami przyjmowanych założeń dotyczących liczby warstw, zależności opóźnienia od długości połączeń między elementami itp. Nakład obliczeń, w tym modelu przedstawiany jest wyrażeniem: $AT^2 = f(N)$, gdzie A jest powierzchnią modułu, T - czasem wykonywania wszystkich działań, N - rozmiarem rozwiązywanego zadania.

Do klasy modeli *MP-RAM* i modeli *BDSS* należy wiele interesujących odmian (m.in. model Komunikujących się Procesów Sekwencyjnych - *KPS*), różniących się przede wszystkim topologią połączeń międzyprocesorowych. Przykład realizacji praktycznej modelu *KPS* podamy w punkcie 2.7.

Procesory i połączenia między nimi tworzą sieć, której węzłami są procesory (i ich pamięci), a krawędziami - połączenia. W wielu przypadkach sieć wygodnie jest przedstawiać za pomocą k-wymiarowej tablicy $P(n_{k-1} \times n_{k-2} \times \ldots \times n_0)$, gdzie n_j jest liczbą procesorów w j-tym wymiarze, a łączna liczba procesorów

$p = n_{k-1} * n_{k-2} * \ldots * n_0$. Dla prostoty, ale bez straty ogólności rozważań, przyjmujemy, że łączna liczba procesorów i liczba procesorów dla poszczególnych wymiarów tablicy są potęgami liczby 2: $p = 2^q$, ($q > 1$); $n_j = 2^{s_j}$, ($s_j > 1$), $0 \le j \le k-1$. Procesory można identyfikować według współrzędnych elementów tablicy, lub na podstawie ustalonej numeracji tych elementów. W pierwszym przypadku procesor $P(i_{k-1}, i_{k-2}, \ldots, i_0)$ jest identyfikowany przez współrzędne ($i_{k-1}, i_{k-2}, \ldots, i_0$), gdzie: $0 \le i_j \le n_j - 1$, $0 \le j \le k-1$. W drugim przypadku każdy procesor ma przyporządkowany numer $m = \sum_{r=0}^{k-1} a_r * i_r$, $0 \le m \le p-1$, gdzie: $a_0 = 1$, $a_r = a_{r-1} * n_{r-1}$ dla $1 \le r \le k$. Jest to, jak łatwo można zauważyć, numeracja elementów w porządku wierszy tablicy. Liczbę m, będącą numerem procesora wygodnie jest przedstawiać w jej rozwinięciu binarnym: $m = m_{q-1}\, m_{q-2} \ldots m_0$, $m_l \in \{0,1\}$, $0 \le l \le q-1$. Łatwo zauważyć, że $\log(n_0)$ najmłodszych bitów rozwinięcia reprezentuje wartość współrzędnej i_0, $\log(n_1)$ kolejnych bitów reprezentuje wartość współrzędnej i_1, itd.

Dwa parametry sieci odgrywają szczególną rolę: stopień sieci d_1, będący maksymalną liczbą procesorów, dołączonych bezpośrednio do jednego procesora; średnica sieci d_2, wyrażona liczbą krawędzi w najdłuższej drodze, wybranej spośród najkrótszych dróg dla wszystkich par węzłów. Za przydatne do realizacji uważane są sieci, dla których parametr d_1 nie zależy od p, natomiast wartość parametru d_2 nie rośnie szybciej niż $\log(p)$. Spośród wielu sieci, spełniających postulowane warunki, kilka uzyskało większą popularność. Przedstawimy niektóre z tych sieci.

Krata wielowymiarowa (KW). W tej sieci procesor o numerze $P(i_{k-1}, ..., i_0)$ jest połączony bezpośrednio z procesorami o numerach $P(i_{k-1}, ..., i_j \pm 1, ..., i_0)$, $0 \le j < k$. Dla najbardziej interesującej, z punktu widzenia praktycznego, kraty - kraty dwuwymiarowej (KD) parametry d_1 i d_2 przyjmują następujące wartości: $d_1 = 4$, $d_2 = \Omega (\sqrt{p})$. Procesory brzegowe kraty mogą być połączone między sobą w różny sposób, tworząc kratę z domknięciem.

Hipersześcian (HS). Jest to sieć, w której procesory są umieszczone w węzłach k-wymiarowej kostki ($k \ge 3$), przy czym $n_j = 2^{S_j} = 2^S = const$, dla $j = 0, ..., k-1$. Procesor $P(m)$ jest połączony bezpośrednio z S procesorami o numerach $m^{(b)}$, $0 \le b < S$ powstałych z liczby m przez zanegowanie bitu na pozycji b, $m^{(b)} = m_{s-1} ... m_{b+1} \overline{m}_b m_{b-1} ... m_0$. Dla hipersześcianu mamy: $d_1 = d_2 = S = \Theta (\log p)$.

Idealnie potasowany stos (IPS). Jest to sieć, w której procesory są połączone według poniższej reguły. Każda liczba i, będąca numerem procesora, jest przekształcana w trzy różne liczby za pomocą funkcji: *SHUFFLE* (i), *UNSCHUFFLE* (i), *EXCHANGE* (i). Liczba *SHUFFLE* (i) = $i_{q-2} i_{q-3} ... i_1 i_0 i_{q-1}$ powstaje z liczby i przez przesunięcie cykliczne, w jej rozwinięciu binarnym, każdego bitu o jedno miejsce w lewo. Liczba *UNSCHUFFLE* (i) = $i_0 i_{q-1} i_{q-2} ... i_2 i_1$ powstaje przez przesunięcie cykliczne o jeden bit w prawo. Liczbę *EXCHANGE* (i) = $i_{q-1} i_{q-2} ... \overline{i}_0$ otrzymujemy z liczby i przez zanegowaniu jej najmłodszego bitu. Procesor $P(i)$ jest połączony bezpośrednio z procesorami o numerach: $P($ *SCHUFFLE* (i)), $P($ *UNSCHUFFLE* (i)) i $P($ *EXCHANGE* (i)). Dla sieci IPS parametry przyjmują wartości: $d_1 = 3$, $d_2 = \Theta (\log p)$.

Kilka kolejnych sieci są odmianami sieci hierarchicznej o nazwie drzewo skierowane. *Drzewo skierowane* jest to acykliczna sieć skierowana spełniająca następujące warunki: istnieje dokładnie jeden węzeł, nazywany korzeniem, do którego nie dochodzi żaden łuk (korzeń nie ma węzłów poprzedzających); istnieje droga (można udowodnić, że jest to droga jedyna), prowadząca od korzenia do każdego węzła; każdy węzeł (z wyjątkiem korzenia) ma dokładnie jeden łuk wchodzący do niego. Jeżeli łuk (i, j) należy do sieci to węzeł i nazywa się poprzednikiem węzła j, a węzeł j nazywa się następnikiem węzła i. Węzeł, który nie ma następników, nazywa się liściem. Głębokością węzła i, w drzewie, jest jego odległość (długość drogi mierzona liczbą łuków) od korzenia. Wysokością węzła i jest maksymalna długość drogi od i do liścia. Wysokością drzewa, będziemy ją oznaczać literą h, jest wysokość jego korzenia.

Drzewo binarne (DB) jest to *drzewo skierowane*, w którym każdy węzeł ma co najwyżej dwa następniki. Drzewo binarne nazywa się pełnym, gdy istnieje liczba naturalna q taka, że każdy węzeł o głębokości mniejszej od q ma dwa następniki, oraz węzeł o głębokości q jest liściem. Pełne drzewo binarne ma dokładnie $2^{(q+1)} - 1$ węzłów, w tym 2^q liści. Dla drzewa binarnego mamy: $d_1 = 2$, $d_2 = \Theta(\log p)$.

Dla pełnego drzewa binarnego i dla innych pełnych drzew definiowanych dalej, liczba q jest równa wysokości drzewa h.

Drzewo ternarne (DT) różni się od DB tym, że korzeń ma stopień 3, a pozostałe węzły mają stopień 4. Pełne DT ma $\sum_{i=0}^{h-1} 3^i$ węzłów, w tym 3^{h-1} liści. Dla drzewa ternarnego mamy: $d_1 = 3$, $d_2 = \Theta(\log p)$.

Podwójne drzewo binarne (PDB). Jest to sieć utworzona z połączenia dwóch pełnych DB w ten sposób, że stanowią one swoje

lustrzane odbicie względem osi przechodzącej przez poziom liści, które są wspólne. Korzeń jednego z drzew jest procesorem wejściowym sieci, podczas gdy korzeń drugiego z nich jest procesorem wyjściowym. Procesy obliczeniowe są wykonywane przez procesory liście (mają własną pamięć i własne programy). Pozostałe procesory drzewa wykonują operacje przesyłania danych do procesorów - liści; procesory drugiego drzewa, nie będące liśćmi, wykonują operacje porównywania. Wynik jest przekazywany do procesora dodatkowego, wypełniającego funkcje sterowania i koordynacji (sieć pracuje synchronicznie) i łączącego wejście jednego drzewa z wyjściem drugiego. W PDB jest $(3 \times 2^h - 2)$ procesorów (nie licząc procesora sterującego; $d_1 = 3$, $d_2 = \Theta(\log p)$.

Drzewa ortogonalne (DO) to sieć, która łączy w sobie cechy drzewa binarnego z cechami kraty dwuwymiarowej - średnica sieci rośnie proporcjonalnie do $\log p$, zaś liczba wejść/wyjść - proporcjonalnie do $2h$. Sieć ta jest kompozycją $2n$, ($n = 2^h$, $h \geq 1$) pełnych drzew binarnych. Liście drzew tworzą tablicę o wymiarze $n \times n$. Każdy wiersz i każda kolumna ma nad sobą swoje drzewo. W ten sposób każda para ortogonalnych drzew ma jeden wspólny liść. Parametry tej sieci są takie same jak parametry sieci DB i PDB.

Gwiazda (G) jest siecią o jednym procesorze centralnym, do którego podłączonych jest pozostałe $p - 1$ procesorów. Gwiazdę można traktować jak drzewo o wysokości $h = 1$. Procesor centralny steruje pracą pozostałych procesorów. przydzielając im zadania i gromadząc wyniki. W sieci G mamy: $d_1 = p - 1$ i $d_2 = 1$.

W tabeli 2.3 zestawiono parametry wymienionych sieci.

Większość z wymienionych sieci można łatwo realizować w technologii *BDSS*. Niektóre z nich (DB, PDB, DO) sprawdzają się jako specjalizowane *maszyny systoliczne* przeznaczone do rozwiązywania określonych zadań optymalizacji na grafach.

Tabela 2.3. Parametry wybranych sieci.

SIEĆ	d_1	d_2	SIEĆ	d_1	d_2
KD	4	$\Theta(\sqrt{p})$	PDB	3	$\Theta(\log p)$
HS	$\log p$	$\Theta(\log p)$	DO	3	$\Theta(\log p)$
IPS	3	$\Theta(\log p)$	G	$p-1$	2
DB	3	$\Theta(\log p)$	DT	4	$\Theta(\log p)$

Z punktu widzenia asymptotycznej złożoności obliczeń różne modele maszyn równoległych okazują się, przy określonych założeniach, równoważne.

Przedstawimy niektóre wyniki dotyczące równoważności modelu *P-RAM-CREW* i modelu *P-RAM-CRCW*. Rozważmy następujące trzy reguły rozwiązywania prób jednoczesnego zapisu, przez więcej niż jeden procesor, do tego samego miejsca pamięci:

a) objęte konfliktem komórki pamięci mogą przechowywać jedynie liczby 0 lub 1, w przypadku konfliktu prawo zapisu otrzymuje dowolny procesor zapisujący liczbę 1;
b) jednoczesny zapis jest dopuszczalny jeżeli procesory zapisują tę samą liczbę, (odmianą tej reguły jest dopuszczenie do próby jednoczesnego zapisu różnych liczb, z tym że zapisu dokona jeden przypadkowy procesor);
c) procesory są uporządkowane według ustalonego priorytetu i prawo zapisu otrzymuje procesor o najwyższym priorytecie.

Można dowieść, że jeżeli znany jest algorytm, który rozwiązuje zadanie o rozmiarze N na maszynie *P-RAM-CREW* w czasie T, to istnieje algorytm rozwiązujący to zadanie w czasie $O(T / \log N)$ na maszynie *P-RAM-CRCW*(a). Można także pokazać, że zadanie rozwiązywalne na maszynie *P-RAM-CRCW*(c) w czasie $f(N)$ jest rozwiązywalne na maszynie *P-RAM-CRCW* (a,b) czasie $\alpha f(N)$, gdzie α jest stałą.

Inny rodzaj zależności istnieje między parakomputerem i ultrakomputerem. Można pokazać, że parakomputer emuluje ultrakomputer w czasie niazależnym od liczby procesorów. Z kolei, ultrakomputer o skończonej liczbie procesorów symuluje, konstruktywnie, parakomputer o skończonej liczbie procesorów w czasie, który dąży do $O(\log^2 p)$ z prawdopodobieństwem dążącym do 1 wraz ze wzrostem p. Niekonstruktywna symulacja jest możliwa w czasie deterministycznym $O(\log M * \log p)$, gdzie M jest rozmiarem wspólnej pamięci parakomputera: M, $p < \infty$.

Również dla większości modeli sieciowych dowodzi się wielomianowej równoważności, co pozwala w sposób prosty przenosić wyniki analizy złożoności otrzymane dla jednej z nich na inne.

Następujący przykład ilustruje obliczenia przy użyciu modelu *P-RAM-CRCW*(a). Należy znaleźć wartość minimalną w zbiorze n liczb umieszczonych w n-wymiarowej tablicy A. Jednoczesny zapis dokonywany przez wiele procesorów jest możliwy w n komórkach pamięci M.

Algorytm *MIN*1

PARALLEL1

Procesory $P(i)$, $1 \le i \le n$, wykonują: $M(i) = 0$

END PARALLEL1

PARALLEL2

Procesory $P(i,j)$ dla wszystkich uporządkowanych par (i,j), $1 \le i \le n$, wykonują:
if $A(i) < A(j)$ **then** $M(j) = 1$

END PARALLEL2

PARALLEL3

Procesory $P(i,j)$ dla wszystkich uporządkowanych par (i,j), $1 \leq i \leq n$, wykonują:
if $M(i) = 0$ oraz $i < j$ **then** $M(j) = 1$

END PARALLEL3

PARALLEL4

Procesory $P(i)$, $1 \leq i \leq n$, wykonują:
if $M(i) = 0$ **then** *minimum* $= A(i)$.

END PARALLEL4

END Algorytm *MIN*1

Po wykonaniu instrukcji **PARALLEL2**, $M(i) = 0$ wtedy i tylko wtedy kiedy $A(i)$ jest najmniejszą liczbą w tablicy A. Wykonanie instrukcji **PARALLEL3** gwarantuje, że w pamięci M dokładnie jedna komórka zawiera liczbę 0 (będzie to komórka o najmniejszym numerze pośród komórek zawierających 0 po wykonaniu instrukcji **PARALLEL2**).

Instrukcję **PARALLEL4** wykona już tylko jeden procesor. Złożoność algorytmu jest $O(1)$. Taki wynik jest możliwy dzięki dopuszczeniu konfliktowych zapisów. W modelu *SIMD - CREW* rozwiązanie omawianego zadania wymaga $O(\log n)$ obliczeń. Podobny wynik obowiązuje dla sieci typu HS, IPS i DB.

2.5. Taksonomie modeli

Różnorodność modeli obliczeń równoległych i koncepcji maszyn, które mogą je realizować, skłaniają do poszukiwań dobrego systemu klasyfikacyjnego, lub przynajmniej odpowiedniej taksonomii. Te ostatnie w istocie są powszechnie stosowane, gdyż znane

systemy klasyfikacji są zbyt złożone. Powszechnie przyjęła się taksonomia, której podstawą jest interakcja między strumieniem danych i strumieniem rozkazów, w procesie obliczeń. Z tego punktu widzenia wyróżnia się cztery klasy maszyn.

(1) - Maszyny klasy *SISD* (Single Instruction stream, Single Data stream). W każdej chwili, jeden procesor wykonuje jedną instrukcja na jednym zestawie danych. Jest to model sekwencyjnej maszyny von Neumanna.

(2) - Maszyny klasy *SIMD* (Single Instruction stream, Multiple Data stream). W każdej chwili, w trybie synchronicznym, procesory wykonują identyczną instrukcję na zestawie, niekoniecznie identycznych, danych. Dopuszcza się pauzowanie określonych procesorów w różnych taktach obliczeń (model *SIMD z maską*). Do tej klasy należą m.in. *maszyny systoliczne* i wiele innych synchronicznych maszyn wieloprocesorowych; *maszyny wektorowe* także zalicza się do tej klasy.

(3) - Maszyny klasy *MISD* (Multiple Instruction stream, Single Data stream). W każdej chwili każdy procesor wykonuje odrębną instrukcję na identycznych danych. Ta klasa maszyn nie cieszy się większym zainteresowaniem.

(4) - Maszyny klasy *MIMD* (Multiple Instruction stream, Multiple Data stream). W każdej chwili każdy procesor wykonuje swój zestaw instrukcji na swoim zestawie danych. Wyróżnia się synchroniczny i asynchroniczny typ maszyny *MIMD*. Do tej klasy niektórzy autorzy zaliczają także wielomaszynowe sieci rozproszone.

W myśl przedstawionej taksonomii maszyny *P-RAM* i *MP-RAM* można zaliczyć do maszyn klasy *SIMD* i do klasy synchronicznych maszyn *MIMD*.

Rozwój architektur komputerów, w tym przede wszystkim komputerów równoległych odbywa się także pod wpływem innych, niż

sterowanie strumieniem rozkazów, koncepcji sterowania procesem obliczeń: *sterowanie strumieniem potrzeb* i *sterowanie strumieniem danych*.

Sterowanie strumieniem potrzeb polega na tym, że działania są wykonywane w kolejności określonej przez wartości operandów niezbędnych do wykonania danej operacji. Rozwój tego typu maszyn jest stymulowany głównie potrzebami zadań pojawiających się w obszarze badań nad sztuczną inteligencją.

Sterowanie strumieniem danych polega na wykonywaniu operacji w kolejności dostępności potrzebnych danych.

Inna taksonomia bierze za punkt wyjścia stosunek czasu obliczeń, dokonywanych na pewnej jednostce danych, do czasu związanego z transmisją tej jednostki danych między procesorami (parametr τ). Należy zauważyć, że czas zużyty na komunikację między procesorami i na ich komunikację z otoczeniem, odgrywa istotną rolę w obliczeniach równoległych. Znane są przykłady pokazujące, że czas ten może decydować o łącznej złożoności czasowej algorytmu (np. sortowanie n^2 liczb, lub odwracanie macierzy o wymiarze $n \times n$, w modelu KD o $n \times n$ procesorach, wymaga $O(\log^2 n)$ operacji logiczno-arytmetycznych i $O(n)$ operacji komunikacyjnych). W tej taksonomii wyróżnia się trzy klasy maszyn.

1. Maszyny wieloprocesorowe o pamięci wspólnej lub rozproszonej:

a) skupienia stacji roboczych (50÷100 stacji) - $\tau = 10^3 \div 10^6$,

b) wieloprocesory z pamięcią rozproszoną: $p = 32 \div 100$, - $\tau = 2 \div 10^3$,

c) wieloprocesory z pamięcią wspólną: $p = 4 \div 30$, $\tau = 1 \div 10^2$.

Uwaga: maszyny grupy a) i b) są też nazywane *wielokomputerami*.

2. Maszyny potokowe, z mikro- i makro-potokowaniem obliczeń - (liczba stopni kaskady $K > 1$) $\tau = 1/K$.
3. Maszyny synchroniczne o *równoległości dużej skali* ($10^5 \div 10^7$ elementów operacyjnych), w tym maszyny: systoliczne, neuronowe, Boltzmanna - $\tau < 1$.

2.6. Jakość i złożoność algorytmów równoległych

Model obliczeń służy analizie procesu obliczeniowego, która powinna dać odpowiedź na pytanie, w jaki sposób projektować i realizować algorytmy zapewniające najwyższą sprawność, z punktu widzenia przyjętych kryteriów oceny. Dominującym sposobem analizy jakości algorytmów jest ocena ich złożoności obliczeniowej w odniesieniu do czasu obliczeń i wykorzystania pamięci. W przypadku optymalizacji równoległej należy dodatkowo uwzględnić liczbę wykorzystanych procesorów i nakład czasu związany z organizacją ich pracy. Zagadnienia te są przedmiotem zainteresowania teorii złożoności obliczeń równoległych. Stosuje się tu, powszechnie, metodologię wypracowaną dla obliczeń sekwencyjnych.

Przyjęto utożsamiać algorytmy równoległe z nazwami modeli obliczeń, kórych one dotyczą, mówimy więc o algorytmie *P-RAM*, *MP-RAM*, *SIMD-CREW*, algorytmie *systolicznym* itd.

Złożoność czasowa algorytmu równoległego, rozwiązującego zadania o wymiarze N, z ustalonej klasy zadań, jest to funkcja $f(N)$, która każdemu zadaniu z tej klasy przyporządkowuje maksymalny czas upływający od rozpoczęcia pracy przez pierwszy procesor do zakończenia obliczeń przez ostatni. Tak rozumianą złożoność, szczególnie w odniesieniu do zadań na grafach, nazywa się też głębokością algorytmu lub głębokością obliczeń.

Minimalną liczbę procesorów p, przy której osiąga się daną głebokość algorytmu, nazywa się często szerokością algorytmu.

Kosztem algorytmu równoległego przyjęto nazywać iloczyn $f(N) * g(N)$, gdzie $g(N) = p$ jest funkcją wyrażającą minimalną liczbę procesorów, która zapewnia głębokość $f(N)$. Algorytmem optymalnym nazywa się algorytm, dla którego ten iloczyn jest funkcją liniową rozmiaru N: $f(N) * g(N) = O(N)$.

Przyśpieszeniem s algorytmu równoległego nazywa się iloraz t_{sz} / t_r, gdzie $t_r = f(N)$, a t_{sz} jest złożonością czasową najlepszego znanego (lub potencjalnie możliwego) algorytmu sekwencyjnego. Z tego punktu widzenia, za najlepszy można uważać algorytm o największym przyśpieszeniu. Do oceny algorytmów równoległych stosuje się także pojęcie: efektywność wykorzystania procesorów - $e_{WP} = (s / p) = t_{sz} / (t_r * p)$. Optymalne wykorzystanie procesorów ma miejsce wtedy, gdy $e_{WP} = 1$.

Zauważmy, że pojęcie optymalności w sensie e_{WP} i w sensie $t_r * p = f(N) * g(N) = O(N)$ nie są zgodne. Kryterium pierwsze eksponuje najlepsze wykorzystanie procesorów, co ma znaczenie wówczas gdy koszt procesora jest duży, podczas gdy kryterium drugie preferuje najlepsze dopasowanie algorytmu do równoległości tkwiącej w zadaniu.

Trzecim zasobem, który obok czasu obliczeń i liczby procesorów odgrywa istotną rolę w analizie złożoności obliczeń algorytmu, jest pamięć M. Złożoność pamięciowa algorytmu równoległego rozwiązującego zadania o wymiarze N, z ustalonej klasy zadań, jest to funkcja $g(N)$, która każdemu zadaniu z tej klasy przyporządkowuje maksymalną objętość łącznej pamięci zajmowanej przez wszystkie zaangażowane procesory.

Omówimy najważniejsze wyniki teorii złożoności obliczeń równoległych. Za punkt wyjścia przyjmiemy klasy złożności dla modelu sekwencyjnego *RAM*.

Klasa P obejmuje zadania rozwiązywalne w czasie proporcjonalnym do funkcji wielomianowej rozmiaru zadania.

Klasę PSPACE tworzą zadania, dla rozwiązania których potrzebna jest pamięć o objętości proporcjonalnej do N.

Podklasą w PSPACE jest klasa NP obejmująca zadania, dla których stwierdzenie istnienia rozwiązania wymaga wielomianowego (w funkcji N) nakładu czasu. Zachodzą relacje: $P \subseteq NP \subseteq PSPACE$, przy czym formułowana jest hipoteza, że obydwie relacje zawierania są właściwe. Inną ważną podklasą w PSPACE jest klasa POLYLOGSPACE, obejmująca zadania rozwiązywalne w pamięci proporcjonalnej do wielomianu zmiennej $\log N$. Wiele zadań z klasy P należy do POLYLOGSPACE, niemniej relacja: P = POLYLOGSPACE pozostaje nieudowodniona (wiadomo jednek, że POLYLOGSPACE ≠ PSPACE).

Zbiór PSPACE-zupełny, ze względu na transformację o wielomianowym nakładzie czasu, tworzy zbiór zadań należących do klasy PSPACE takich, że każde inne zadanie z tej klasy można sprowadzić do jednego z nich przez transformację wymagającą wielomianowego nakładu czasu. Podobnie definiowany jest zbiór NP-zupełny klasy NP.

Zbiór P-zupełny, ze względu na przekształcenie logarytmiczne pamięci, obejmuje zadania z klasy P takie, że każde inne zadanie z klasy można sprowadzić do jednego z nich, przez transformację w pamięci o objętości proporcjonalnej do wielomianu zmiennej $\log M$.

Gdyby udało się pokazać przynależność jakiegokolwiek zadania ze zbioru zadań P-zupełnych do klasy POLYLOGSPACE, wówczas prawdziwa byłaby relacja $P \subseteq$ POLYLOGSPACE .

Reasumując, w modelu obliczeń sekwencyjnych można wyróżnić t.zw. zadania "łatwe" - zadania klasy P, dla których istnieje

algorytm o wielomianowym nakładzie czasu obliczeń i zadania "trudne", dla których algorytmu takiego dotąd nie znaleziono.

Łącznikiem między teorią złożoności obliczeń sekwencyjnych i teorią złożoności obliczeń równoległych jest hipoteza nazywana hipotezą obliczeń równoległych: nakład czasu obliczeń maszyny równoległej można wyrazić funkcją będącą wielomianem nakładu pamięci maszyny sekwencyjnej. Oznacza to, że dla dowolnej funkcji $f(N)$ rozmiaru zadania, klasa zadań rozwiązywalnych przez maszynę równoległą o nieograniczonej liczbie procesorów w czasie $f(N)^{O(1)}$ (t. zn. w czasie wielomianowym w funkcji $f(N)$) jest identyczna z klasą zadań rozwiązywalnych na maszynie sekwencyjnej w pamięci o objętości $f(N)^{O(1)}$. Hipotezę tę udowodniono dla niektórych postaci funkcji $f(N)$ (np. $f(N) = O(N)$) i dla niektórych modeli obliczeń (np. *P-RAM*). Z kolei dowodzi się, że hipoteza nie obowiązuje gdy modelem jest komputer zdolny do jednoczesnego uruchomienia skończonej liczby procesorów, które mogą czytać jednocześnie z tego samego miejsca pamięci o skończonej (chociaż b. dużej) objętości. Stąd wniosek, że twierdzenie nie obowiązuje dla modelu *P-RAM* przy założeniu dowolnej postaci funkcji $f(N)$ (aby uruchomić p procesorów potrzeba $O(\log p)$ jednostek czasu).

Nietrudno zauważyć, że nieograniczony model *P-RAM* jest modelem bardzo silnym, gdyż rozwiązuje w czasie wielomianowym wszystkie zadania klasy PSPACE (tym samym np. wszystkie najtrudniejsze zadania optymalizacji dyskretnej). W tej sytuacji bardziej realistycznym modelem wydaje się model *WP-RAM*, t.j. model *P-RAM* z liczbą procesorów rosnącą proporcjonalnie do wielomianu od rozmiaru zadania. Dla tego modelu, klasa zadań rozwiązywalnych w czasie wielomianowym jest równa dokładnie klasie P. Oznacza to, że zadania "trudne" w teorii złożoności obliczeń

sekwencyjnych pozostają zadaniami "trudnymi" w teorii złożoności obliczeń równoległych.

Wiele uwagi poświęcono dokładniejszemu zbadaniu zachowania się zadań klasy P w warunkach równoległości. Okazuje się, że można w niej wyróżnić pewne podklasy. Jedną z nich jest klasa NC - klasa zadań rozwiązywalnych w wielomianowo-logarytmicznym czasie równoległym przez wielomianową liczbę procesorów: $t_r = O(\log^k N)$, $p = O(N^k)$, $k > 0$. Nazwę NC tworzą pierwsze litery: imienia autora, Nic (Nic Pipinger opisał tę klasę) i słowa Class. Do tej klasy należą m.in. zadania: minimalnych i minimaksowych dróg, minimalnych i minimaksowych dendrytów, skojarzenia optymalnego w grafie płaskim, programowania liniowego z ograniczonymi zmiennymi.

Nie podlega wątpliwości relacja: $NC \subset P$, ale wiele wskazuje na to, że prawdziwa jest nierówność $NC \neq P$. Dla zadań należących do klasy NC można podać algorytmy o głębokości proporcjonalnej do $(\log N)^{O(1)}$ i o wielomianowej szerokości, czyli t.zw. algorytmy efektywne. Dla pozostałych zadań klasy P algorytmy takie raczej nie istnieją.

Innym podzbiorem klasy P jest klasa SC, S(teve Cook's) C(lass), obejmująca zadania rozwiązywalne algorytmem sekwencyjnym w wielomianowym czasie i w wielomianowo-logarytmicznej pamięci. Uogólniona hipoteza obliczeń równoległych głosi, że: $NC = SC$, i jest to jeden z istotniejszych, nierozstrzygniętych, problemów złożoności obliczeń równoległych.

Podsumowując tę część rozważań stwierdzamy, że klasę P, klasę zadań "łatwych" w modelu obliczeń sekwencyjnych (rozwiązywalnych algorytmami wielomianowymi) można podzielić, w modelu obliczeń równoległych, na dwie grupy. Pierwszą tworzą zadania "bardzo łatwe" - rozwiązywalne za pomocą algorytmów o wielomianowym, w funkcji logarytmu rozmiaru zadania, czasie i o wielomia-

nowej, w funkcji rozmiaru zadania, pamięci. Do grupy drugiej należą zadania P-zupełne - zadania "nieco trudniejsze", dla których wskutek inherentnie sekwencyjnych fragmentów obliczeń, istnienie algorytmów równoległych o nakładzie wielomianowo - logarytmicznym jest nader wątpliwe. Zauważmy, że istnienie zadania P-zupełnego rozwiązywalnego algorytmem równoległym o wielomianowo - logarytmicznym nakładzie czasu oznaczałoby prawdziwość relacji $P \subseteq$ POLYLOGSPACE, czego raczej nie należy się spodziewać.

2.7. Równoległość zadań i zasady budowy algorytmów równoległych

W podrozdziałach 2.4 - 2.6 przedstawiliśmy najważniejsze modele obliczeń równoległych i podaliśmy najistotniejsze wyniki teorii złożoności obliczeń dla tych modeli. Pojawia się naturalne pytanie, jak można wykorzystać te wyniki do budowy i analizy złożoności obliczeniowej algorytmów równoległych, a specjalnie algorytmów równoległej optymalizacji dyskretnej. Dla obliczeń sekwencyjnych podstawowe odpowiedzi na to pytanie zostały sformułowane w latach 70-ych, po ponad ćwierćwieczu intensywnych badań. Uważa się, dość powszechnie, że dzisiejszy stan wiedzy na ten temat, dla obliczeń równoległych, odpowiada stanowi jaki osiągnięto dla obliczeń sekwencyjnych w bardzo wczesnym okresie ich rozwoju (koniec lat 50-ych). Z tego powodu projektant algorytmów równoległych staje przed problemami, dla których nie ma w chwili obecnej zadowalającego rozwiązania.

W połączonej analizie zadań, algorytmów i maszyn, której celem jest możliwie najpełniejsze wykorzystanie równoległości tkwiącej w zadaniu, ważną rolę odgrywa ziarnistość (stopień rozdrobnienia) zadań i algorytmów i odpowiednich procesów.

Ziarnistość zadania jest to liczba współbieżnych podzadań, na które można je podzielić.

Ziarnistość algorytmu jest to średnia liczba operacji maszynowych wykonywanych przez realizujące go równoległe procesy bez potrzeby komunikowania się między sobą. Procesy o grubej ziarnistości nie wymagają częstej interkomunikacji, podczas gdy procesy o drobnej ziarnistości często się ze sobą komunikują.

Dwa typy często występujących zadań cechuje gruba ziarnistość; są to zadania powtarzalne z różnymi zestawami danych i zadania o charakterze przestrzenno - czasowym. Obliczenia równoległe dla zadań pierwszego typu charakteryzuje zwiększone zapotrzebowanie na pamięć, ale straty czasu na komunikację i koordynację obliczeń są stosunkowo nieduże. Zadania drugiego typu otrzymujemy wówczas, gdy identyczne obliczenia należy wykonać dla zmieniających się, np. w funkcji współrzędnych przestrzennych lub czasu, danych. Zadania takie można łatwo podzielić na niezależne podzadania, pamiętając o potrzebie koordynacji rozwiązań na brzegach podziału. Każde podzadanie ma swój, lokalny, zestaw danych, co zmniejsza zapotrzebowanie na wspólną pamięć, natomiast dodatkowe nakłady czasowe mogą być większe niż dla zadań pierwszego typu.

Do grubej ziarnistości procesów prowadzą niektóre metody stosowane szeroko do rozwiązywania zadań, m.in. w optymalizacji. Należą do nich metody dekompozycji zadań, metody poszukiwania losowego w wielu otoczeniach, metody podziału i oszacowań i inne. Źródłem równoległości drobnoziarnistej są najczęściej techniki iteracyjne, operacje wektorowe i macierzowe. W praktyce wykorzystanie ziarnistości zadań zależy od architektury maszyny oraz od możliwości dostępnych języków programowania.

Dla wielu zadań możliwości zastosowania obliczeń równoległych nie są tak oczywiste jak to ma miejsce w przypadkach ich natural-

nej gruboziarnistości lub drobnoziarnistości. Często wymagana jest głębsza, czasami subtelna, analiza i wyrafinowane techniki algorytmiczne. Pożądana i możliwa do uzyskania ziarnistość procesów zależy od typu zadania, charakteru modułów składających się na program komputerowy, jak również od modelu maszyny równoległej. Modułem programu może być on sam, jego podprogramy (procedury, funkcje itp), instrukcje, wyrażenia, rozkazy lub adresy.

Zasygnalizowane problemy zilustruje prosty przykład. Rozważmy zadanie mnożenia dwóch macierzy o wymiarze $n \times n$, ($C = A \times B$):

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj} ,$$

dla którego sekwencyjny kod fortranowski może mieć postać (początkowo wszystkie $c_{ij} = 0$):

```
do 100  j = 1, n
do 100  i = 1, n
do 100  k = 1, n
      c(ij) = c(ij) + a(ik) * b(kj)
100 CONTINUE
```

Obliczenia równoległe o grubej ziarnistości można wprowadzić przez wykorzystanie faktu, że dwie zewnętrzne pętle **do** generują n^2 niezależnych podzadań, każde z których polega na obliczeniu właściwego elementu c_{ij}. Każde podzadanie może być realizowane przez odrębny procesor wyposażony w niewielką pamięć i zdolny do wykonywania określonych operacji. Narzucającym się modelem maszyny równoległej jest synchroniczna (identyczny czas trwania procesów) maszyna *MIMD*.

Drobniejszą ziarnistość procesów otrzymuje się na poziomie pojedynczych instrukcji rozważanego programu, polegających na wykonaniu mnożenia dla pary elementów: $a_{ik} \times c_{kj}$. Obliczenia o tej ziarnistości można zrealizować na wielu maszynach, wskażemy dwie: systoliczną kratę dwuwymiarową (SKD), i maszynę wektorową.

Systoliczną kratę dwuwymiarową (*maszynę systoliczną*) tworzy n^2 elementów operacyjnych (EO). Każdy EO składa się z kilku rejestrów pamięci i arytmometru. Komunikacja z otoczeniem odbywa się przez elementy na brzegach kraty, które nie są domknięte. Macierze i program znajdują się w procesorze zewnętrznym względem kraty. Istota obliczeń systolicznych polega na synchronicznej, potokowej pracy (patrz niżej) elementów operacyjnych sprowadzającej się do: otrzymania danych, wykonania działań, zapamiętania wyników w lokalnych rejestrach, przekazania otrzymanych danych do EO określonych przez kierunek potoku. W naszym przypadku, macierz A jest podawana do brzegowych EO kraty wierszami (przyjmijmy umownie, że od strony lewej) w ten sposób, że każdy wiersz jest podawany z opóźnieniem jednego taktu, w stosunku do wiersza poprzedniego. Macierz B jest podawana kolumnami (przyjmijmy umownie, że od góry) z identycznym jak dla wierszy przesunięciem macierzy, wykonaniu mnożenia, dodaniu wyniku do zawartości akumulatora (początkowo wszystkie akumulatory są wyzerowane) i przekazaniu elementu wierszowego na prawo, a elementu kolumnowego w dół. Elementy operacyjne wykonują te działania w miarę podawania danych na ich wejścia, skrajne prawe i skrajne dolne elementy operacyjne nie przekazują danych dalej. Można zauważyć, że po $2n - 1$ taktach w akumulatorze procesora $P(i, j)$ znajduje się element c_{ij} macierzy $C = A \times B$. Przyśpieszenie obliczeń, odniesione do klasycznego algorytmu sekwencyjnego o złożoności obliczeniowej $O(n^3)$, wynosi $O(n^2)$ przy efektywności

wykorzystania procesorów = 1. Systoliczną kratę dwuwymiarową można traktować jak specjalistyczną (dedykowaną do pewnych klas) maszynę *SIMD*, szczególnie łatwą do realizacji w technologii *BDSS*.

Potokowanie obliczeń polega na podziale procesu na sekwencję podprocesów wykonywanych synchronicznie i współbieżnie przez procesor o specjalnej architekturze nazywanej kaskadową. Elementy procesora kaskadowego są nazywane *stopniami*, składowe danych tworzące potok - *segmentami*. Stopnie kaskady połączone są w ten sposób, że wyjście stopnia poprzedniego jest wejściem stopnia następnego. Potokowanie obliczeń można realizować na różnych poziomach procesu, od programu po składowe pojedynczej instrukcji (jeden z powszechniejszych sposobów przyśpieszania pracy maszyn jednoprocesorowych). Przyśpieszenie obliczeń potokowych jest proporcjonalne do liczby stopni kaskady K .

Wektoryzacja obliczeń polega na jednoczesnym wykonywaniu działań przez wektor rozkazów na wektorze danych. Z reguły mamy do czynienia z jednorodnym wektorem rozkazów i niejednorodnym wektorem danych.

Maszyna wektorowa jest to *maszyna równoległa* o architekturze umożliwiającej realizację obliczeń wektorowych. Pamięć i sterowanie maszyny zapewniają synchroniczne podawanie danych, w postaci wektora, z pamięci do arytmometru i odwrotnie. Często pojęcie maszyny wektorowej jest zawężane do maszyny, którą tworzą zespoły wąsko wyspecjalizowanych kaskadowych jednostek funkcyjnych przeznaczonych do wykonywania podstawowych operacji obliczeniowych. Jednostki funkcyjne pracują równolegle, a w architekturach sieciowych procesory wykonują obliczenia potokowo.

Program mnożenia macierzy przedstawiony wyżej nie pozwala na bezpośrednie obliczenia drobnoziarniste z wykorzystaniem we-

ktoryzacji. Pożądany efekt otrzymamy modyfikując ten program naastępująco:

```
do 100   k = 1, n
do 100   j = 1, n
do 100   i = 1, n
      c_ij = c_ij + a_ik × b_kj
100 CONTINUE
```

Aby potwierdzić poprawność tego programu zauważmy, że wykonywane jest k-krotnie następujące działanie:

$$c^{(k)} = a^{(1)} \times b_{1k} + a^{(2)} \times b_{2k} + \ldots + a^{(n)} \times b_{nk},$$

gdzie: $c^{(k)}$, $a^{(j)}$ są to wektory kolumnowe, $1 \leq j,\ k \leq n$.

Macierz C otrzymujemy w wyniku n^2 mnożeń wektora przez skalar i $n(n-1)$ dodawań wektorów.

Przedstawimy teraz, w dużym skrócie, przegląd najważniejszych metod stosowanych do budowy równoległych algorytmów optymalizacji. Przystępując do budowy algorytmu równoległego możemy: wykorzystać znany algorytm sekwencyjny, zbudować zupełnie nowy algorytm dostosowany do zadania i architektury danej maszyny, zaadaptować znany algorytm równoległy, który rozwiązuje zadanie podobne do naszego. Należy zauważyć, że efektywne algorytmy sekwencyjne i metody ich budowy okazują się często nieefektywne dla implementacji w warunkach równoległości i na odwrót.

Do najważniejszych, ogólnych, technik stosowanych przy budowie algorytmów równoległych należą: wektoryzacja, potokowanie i metody rekurencyjne (metoda drzewa binarnego, metoda zdwajania odstępów, metoda połowienia, metoda kumulacji).

Wektoryzację i potokowanie obliczeń scharakteryzowano wyżej. Są to metody związane ściśle z architekturą maszyny.

Metody rekurencyjne, których przykłady podamy, są niezależne od modelu maszyny. Wybór przykładów metod rekurencyjnych jest dokonany z myślą o zastosowaniu do rozwiązywania zadań optymalizacji dyskretnej, szczególnie zadań optymalizacji na grafach i uwzględnia specyfikę sposobu reprezentowania danych odnoszących się do tych zadań.

Metoda drzewa binarnego, nazywana także metodą łączenia w pary, ma zastosowanie w przypadku danych reprezentowanych przez uporządkowany zbiór elementów. Niech będzie dany wektor $A = \{ A(1), A(2), ..., A(n) \}$ gdzie dla prostoty przyjmujemy że $n = 2^m$, $m \geq 1$. Mamy $n / 2$ procesorów, które wykonują określone operacje na parach sąsiadujących składowych wektora. Obliczenia są wykonywane iteracyjnie. Pierwszą iterację wykonuje $n / 2$ procesorów i otrzymane wyniki tworzą wektor $A^{(1)} = \{A^{(1)}(1), ..., A^{(1)}(n / 2)\}$. Obliczenia w iteracji i, $1 < i \leq m$, wykonuje $n / 2^{(i+1)}$ procesorów na wektorze o długości $n / 2^{(i)}$. Całe postępowanie można zobrazować pełnym drzewem binarnym, w którego liściach umieszczone są elementy wektora A, a w węzłach kolejnych poziomów są umieszczane składowe kolejnych wektorów.

Metodę drzewa binarnego zilustrujemy algorytmem znajdowania elementu o wartości minimalnej, w zbiorze n-elementowym. Do przedstawienia obliczeń, wykonywanych "od liści do korzenia", wygodnie jest przyjąć, że dana jest tablica o wymiarze $2n$, a elementy zbioru, w którym poszukujemy elementu minimalnego, są umieszczone na pozycjach $A(n), A(n+1), ..., A(2n-1)$. Algorytm *MIN2* (w modelu obliczeń *P-RAM-CREW*) przyjmie postać (po zakończeniu obliczeń wartość minimalna znajduje się na miejscu $A(1)$ tablicy):

Algorytm *MIN2*

for $k = m - 1$ **step** -1 **to** 0 **do**

PARALLEL

Procesory o numerach: $P(j)$, $2^k \leq j \leq 2^{k+1} - 1$ wykonują: $A(j) = min\{A(2j), A(2j+1)\}$.

END PARALLEL

END Algorytm *MIN2*

Metoda zdwajania odstępów jest stosowana zwykle do zadań, których dane przedstawia lista. W metodzie tej jedna iteracja polega na wykonaniu określonego działania z udziałem elementów listy będących w ustalonej odległości (pozycji na liście) względem siebie. Dla iteracji o numerze k działania są wykonywane z udziałem elementów znajdujących się w odległości 2^k.

Metoda połowienia polega na rekurencyjnym dzieleniu zadania na podzadania, które mogą być rozwiązywane równolegle. Jeżeli dla każdej iteracji rozmiar każdego podzadania będzie w stałej proporcji do jego zadania macierzystego, to głębokość rekursji będzie logarytmiczna w funkcji rozmiaru zadania.

Metoda kumulacji jest stosowana często do zadań na grafach. Metoda polega na sprawdzaniu interesującej nas własności grafu na jego częściach. Dla iterecji o numerze k średnica każdej części grafu jest $O(2^k)$. Typowym zastosowaniem tej metody jest znajdowanie składowych spójnych i składowych silnie spójnych dla grafów nieskierowanych i skierowanych.

Wspólną cechą przedstawionych metod rekurencyjnych jest to, że liczba iteracji jest $O(\log n)$, gdzie n jest rozmiarem danej struktury (długością wektora, długością listy itp.).

2.8. Transputer i jego sieci

Transputer jest zintegrowanym, bardzo nowoczesnym w swej architekturze mikroprocesorem, zaprojektowanym i wykonanym z myślą o zastosowaniu w obliczeniach równoległych. Dla przykładu, transputer IMS T9000 (najnowszy produkt firmy INMOS) to jedna kostka zrealizowana w technologii *BDSS*, zawierająca: 32-bitową jednostkę arytmetyczno-logiczną, 64-bitową jednostkę zmiennoprzecinkową, procesor komunikacyjny obsługujący cztery szerokopasmowe, szeregowe sprzęgi dwukierunkowej komunikacji bezpośredniej z innymi transputerami, kilkustopniową pamięć podręczną oraz szerokopasmowy sprzęg do komunikacji z pamięcią RAM. Transputer T9000 pracuje z częstotliwością 50 MHz, jego jednostka centralna ma moc obliczeniową 200 Mips lub 25 Mflops i adresuje pamięć zewnętrzną o pojemności do 4 GB, łącza komunkacyjne pozwalają na transmisję z szybkością 80 MB/s.

Rodzina transputerów obejmuje następujące produkty: 16-bitowy stałoprzecinkowy IMS T225, 32-bitowy stałoprzecinkowy IMS T414 i IMS T425, oraz 32-bitowe (64-bitowa jednostka zmiennoprzecinkowa): IMS T800, IMS T805 i IMS T9000. Transputery wcześniejsze od T9000 mogą pracować z częstotliwością zegara 17, 20, 25, 30 lub 35 MHz. Użytkownik ma do dyspozycji szereg modułów pomocniczych takich jak komutatory sieciowe, adaptory umożliwiające pracę sieci złożonych z różnych typów transputerów i inne. Wiele firm oferuje kompletne transputerowe zestawy obliczeniowe dużej mocy z oprogramowaniem dedykowanym do rozwiązywania konkretnych zadań z dziedziny przetwarzania sygnałów, rozpoznawania obrazów, sterowania w systemach czasu rzeczywistego itp. Popularnym produktem rynkowym są karty wielotransputerowe przeznaczone do współpracy z komputerami osobistymi serii IBM, komputerami VAX, stacjami roboczymi SUN.

Softwarowa oferta rynkowa obejmuje kompilatory języków: OCCAM, Fortran, Pascal, C, Modula-2.

Zakład Programowania Matematycznego IBS PAN korzysta z trzech kart transputerowych zainstalowanych na komputerze osobistym klasy PC/AT. Dwie karty firmy Microway są jednotransputerowe: T414 i T800. Każda karta ma 1 MB zintegrowanej pamięci RAM i odpowiednio 2 KB i 4 KB szybkiej pamięci lokalnej . Karta firmy Quintek Fast 9/4T ma 4 transputery IMS T805, z taką samą pamięcią dla każdego transputera jak ma IMS T800. Karty te są zainstalowane na komputerze osobistym, który korzysta z programu o nazwie AFSERVER, odpowiedzialnego za współpracę sieci (operacje WE/WY) z PC.

Dostępne jest oprogramowanie, które obejmuje: - kompilatory Parallel-Fortran (standard Fortran 77), Parallel C V2.1 (standard K&R), Parallel Pascal (standard ISO), OCCAM2 (standard INMOS); - T-Bugger (Debugger dla transputerów); - T-Graph (pakiet programów grafiki okienkowej dla transputerów).

Tabele 2.4 i 2.5 pokazują niektóre możliwości obliczeniowe transputerów oraz wyniki testu porównawczego Whetstona, wykonanego w Zakładzie Programowania Matematycznego IBS PAN.

Wysoki stopień scalenia połączony z zastosowaniem bardzo szybkich układów, potokowanie w jednostkach arytmetycznych, oraz wielu innych nowatorskich rozwiązań architektonicznych czyni transputer wyjątkowo szybkim procesorem, tak do zastosowań sekwencyjnych jak i przede wszystkim do zastosowań w obliczeniach równoległych. Najważniejszą cechą transputera jest podporządkowanie rozwiązań układowych i programowych jednemu celowi nadrzędnemu - wykorzystaniu procesora jako elementu rozbudowywalnej sieci wieloprocesorowej, która realizuje obliczenia równoległe w modelu *komunikujących się procesów sekwencyjnych.*

Tabela 2.4. Czasy operacji zmiennoprzecinkowych.

OPERACJE ZMIENNO PRZECINKOWE	TRANSPUTER					
	T414-20 µs		T800-20 µs		T800-30 µs	
32 bitowe	ŚRED.	MAX	ŚRED.	MAX	ŚRED.	MAX
+ / -	11,5	15,0	350	450	0,230	0,300
*	10,0	12,0	550	900	0,370	0,600
÷	11,3	14,0	800	1400	0,300	0,930
64 bitowe						
+ / -	28,2	35,0	350	450	0,230	0,300
*	38,0	47,0	1000	1350	0,670	0,900
÷	58,8	771,0	1550	2150	1,030	1,430

Tabela 2.5. Wyniki testów porównawczych.

PROCESOR	W TYSIĄCACH WHETSTONÓW/sek	
	POJEDYŃCZA PREC.	PODWÓJNA PREC.
T800-30	6000	3800
T800-20	4000	2500
VAX 11/780	1083	715
SUN 3	860	790
T414-20	667	163
INTEL 286/287	300	-
IBM RT PC+FPA	200	-
INTEL 8086/8087	178	152
MC 68000	13	-
IBM RT PC	12	-

W modelu *KPS* obliczenie jest traktowane jako zbiór autonomicznych, współbieżnych procesów sekwencyjnych, komunikujących się między sobą przez odpowiednio zdefiniowane kanały.

Zakłada się, że komunikacja między procesami jest bezpośrednia i synchroniczna. Kanały są jednokierunkowe, komunikacja dwukierunkowa między parą procesorów wymaga dwóch kanałów. Każdy proces można traktować jak "czarną skrzynkę" połączoną z otoczeniem przez kanały. Rzeczywista zawartość "czarnej skrzynki" nie jest istotna, może nią być fizyczny układ, program lub inny złożony system komunikujących się procesów.

W przypadku sieci transputerowej pojęcie procesu można identyfikować z procesorem. Transputer może więc być utożsamiany z procesem, który komunikuje się z innymi procesorami za pośrednictwem ośmiu fizycznych kanałów komunikacji. Para kanałów łącząca dany transputer z innym transputerem nazywana jest sprzęgiem (transputer ma 4 sprzęgi). Kanały transputera realizują komunikację synchroniczną i niebuforowaną. Przekazanie komunikatu jest możliwe tylko wtedy kiedy transputer odbierający jest gotów do odbioru.

Pojedynczy transputer również może realizować obliczenia równoległe według opisanego modelu. Każde słowo pamięci transputera może być użyte do zdefiniowania kanału dla komunikacji między dwoma procesami realizowanymi przez dany transputer. Adresy słów definiujących kanały są dostępne dla instrukcji we/wy transputera. Zawartość takiego słowa jest użyta do synchronizacji procesów komunikacyjnych. Z punktu widzenia programowania kanał realizowany sprzętowo i kanał realizowany programowo są traktowane identycznie. Ta równoważność kanałów softwerowych i hardwerowych oznacza, że program przeznaczony do wykonania obliczeń w sieci może być uruchomiony na pojedynczym transputerze, a następnie, bez zmiany, przeniesiony na sieć transputerów (bez potrzeby powtórnej kompilacji).

Sposoby przygotowania programu dla sieci transputerów przedstawimy na przykładzie języka Parallel Fortran (w skrócie P-Fortran, produkt firmy *3L*, implementacja Fortranu 77).

W języku P-Fortran zasadniczą rolę odgrywa proces, który w tym kontekście jest nazywany ZADANIEM (niektóre pojęcia specyficzne dla nazewnictwa stosowanego pzez producentów transputerów i oprogramowania dla nich będą wyróżnione specjalną czcionką). Jest to autonomiczny podprogram z wydzielonym dla siebie i dla danych obszarem pamięci, z określonymi wektorami portów dla wejścia i wyjścia. ZADANIE może być także traktowane jak "czarna skrzynka", której połączenia z adresami kanałów fizycznych wykonuje program (plik) konfigurujący - KONFIGURATOR. Program ten określa także sposób przyporządkowania ZADAŃ do transputerów. ZADANIA przypisane jednemu transputerowi (ich liczbę ogranicza objętość pamięci) mogą mieć dowolną liczbę kanałów, ZADANIA umieszczone na różnych transputerach mogą się komunikować między sobą tylko przez kanały fizyczne.

P-Fortran pozwala także na nieco inną organizację obliczeń w obrębie poszczególnych ZADAŃ. ZADANIA mogą być dynamicznie dzielone na podzadania - *wątki*. Wątki utworzone z danego ZADANIA mają z nim wspólny program. W obrębie macierzystego zadania wątki mają dostęp do jego bloków **COMMON**. Korzystanie ze wspólnych danych jest monitorowane za pomocą *semaforów* (bibliotecznej funkcji *semaphore*). Wątki są realizowane na tym samym transputerze, na którym uruchomione zostało tworzące je ZADANIE. Komunikacja wątków między sobą i z zadaniami odbywa się przez kanały.

ZADANIA, składające się na obliczenie, po skompilowaniu i skonsolidowaniu, tworzą pliki binarne, do uruchomienia których konieczny jest KONFIGURATOR. Plik ten jest pisany przez użytkownika. Możliwe są dwie formy tego pliku: zamknięta i otwarta. W pierwszej

z nich użytkownik musi opisać sieć (podać explicite topologię sieci), na której ma być uruchomiony program oraz wskazać przyporządkowanie ZADAŃ transputerom. Oznacza to, że ciężar odwzorowania konfiguracji logicznej programu na konfigurację sprzętową spoczywa na użytkowniku. Niemniej, zmiana topologii sieci nie wymaga powtórnej kompilacji i konsolidacji, wystarczą odpowiednie zmiany w KONFIGURATORZE.

W przypadku KONFIGURATORA otwartego, użytkownik pisze dwa ZADANIA: ZADANIE NADRZĘDNE i ZADANIE PODRZĘDNE. Transputer z ZADANIEM NADRZĘDNYM nazywany jest TRANSPUTEREM WĘZŁOWYM i do niego należy koordynacja pracy TRANSPUTERÓW PODPORZĄDKOWANYCH. Koordynacja polega na przekazywaniu ZADAŃ PODRZĘDNYCH wolnym transputerom. Odbywa się to automatycznie, bez udziału programisty, w tym celu kompilator P-Fortranu korzysta z pomocy gotowego podprogramu o nazwie ROUTER. Transputer podporządkowany odbiera ZADANIE, wykonuje obliczenia i przekazuje wynik do trasputera węzłowego.

Należy zaznaczyć, że w przypadku KONFIGURATORA otwartego sprzętowa konfiguracja sieci może być użytkownikowi nieznana. Ten typ organizacji obliczeń jest możliwy wówczas, gdy rozwiązywany problem dopuszcza dekompozycję obliczenia na odrębne ZADANIA, które nie muszą sie komunikować ze sobą w trakcie realizacji. Użytkownik musi ocenić, czy pamięć transputera podporządkowanego jest wystarczająca do przechowania programu otrzymanego ZADANIA (wszystkie transputery podporządkowane wykonują identyczny program) oraz związanych z nim wyników pośrednich i końcowych.

Literatura

Wybrane artykuły

1. Alt H., Hagerup T., Mehlhorn K., Preparata F.P. (1987): "Deterministic Simulation of Idealized Parallel Computers on More Realistic Ones". *SIAM Journal on Comput.* 16, 808-835.
2. Bell G.C. (1987): "Algorithms and Cooperation of idustry with science". *SIAM News*, March.
3. Bentley J.L., Kung H.T. (1979): "A Tree Machine for Searching Problems". *Proceedings of 1979 International Conf. on Parallel Processing*, 257-266.
4. Blum N. (1983): "A Note on the Parallel Computation Thesis". *Information Processing Letters* 17, 203-205.
5. Chandra A.K., Kozen D.C., Stockmeyer L.J. (1981): "Alternation". *Journal of the ACM*, 28, 114-133.
6. Cook S. A. (1982): "Towards a Complexity Theory of Synchronous Parallel Computation". *Enseignement Mathematique*, 27, 308-316.
7. Cook S.A. (1983): "An Overview of Computational Complexity". *Communications of the ACM*, 26, 401-408.
8. Emmen Ad (1987): "A Survey of Vector and Parallel Processors". W: Algorithms and Applications on Vectorand Parallel Computers, red. H.J.J. de Riele, Th.J. Dekker, van der Vorst, Elsevier Science Publishers B.V., North-Holland, 1-45.
9. Fortune S., Wyllie J. (1978): "Parallelism in Random Search Machines". *Proceedings of 10th Ann. Symposium on Theory of Computing*, 114-118.
10. Flynn M.J. (1966): "Very High-speed Computng Systems". *Proceedings of IEEE*, 54, 1901-1909.
11. Hoare C.A.R. (1985): Communicating Sequential Processes. Prentice - Hall International, Englewood Cliffs.

12. Horovitz E., Zorat A. (1983): "Divide-and-Conquer for Parallel Processing". *IEEE Transactions on Computers*, C-32, 582-585.
13. Johson D.S. (1983): "The NP-Completeness: An Ongoing Guide". *Journal of Algorithms*, 4, 189-203.
14. Kindervater G.A.P. (1991): "Exercises in Parallel Combinatorial Computing". PPreprint.
15. Kindervater G.A.P., Lenstra J.K. (1987): "Parallel Computing in Combinatorial Optimization". *Report OS - R8720, Center for Mathematics and Computer Science.*
16. Korst J.H.M., Aarts E.H.L. (1989): "Combinatorial optimization on a Boltzmann Machine". *Journal of Parallel and Distributed Computing.* Special Issue: Neural Computing, 6, 331-357.
17. Kucera L. (1982): "Parallel Computation and Conflicts in Memory Access". *Information Processing Letters*, 14, 93-96.
18. Kung H.T., Leiserson C.E. (1980): "Systolic Arrays (for VLSI)". W: Algorithms for VLSI Processor Arrays, red. C. Mead, L. Conway, Addison-Wesley, Academic Press, 271-292.
19. Miller R., Stout Q.F. (1986): "Data Movement Techniques for the Pyramid Computer". *SIAM Journal on Computing*, 15.
20. Nath D., Maheshwari S.N., Bhatt P.C.P. (1983): "Efficient VLSI Networks for Parallel Processing Based on Orthogonal Trees". *IEEE Transactions on Computers*, C-32, 569-581.
21. Pipinger N. (1979): "On Simultaneous Resource Bounds". W: *Proceedings 20th Ann. Symposium on Foundation of Computer Science*, 307-311.
22. Preparata F.P., Vuillemin. J (1981): "The Cube-Connected Cycles: A Versatile Network for Parallel Computation". *Communications of the ACM*, 24, 300-309.
23. Quinn M.J., Deo N. (1984): "Parallel Graph Algorithms". *Computing Surveys*, 3, 319-348.

24. Ribeiro C. C. (1987): "Parallel Computer Models and Combinatorial Algorithms". *Annals of Discrete Mathematics.* Elsevier Science Publishers B. V. (North Holland), 31, 325-364.
25. Roucairol C. (1987): "A Parallel Branch and Bound Algorithm for the Quadratic Assignment Problem". *Discrete Applied Mathematics*, 18, 211-225.
26. Savitch W.J., Stimson M.J. (1979): "Time Bounded Random Access Machines with Parallel Computing". *Journal of the ACM*, 26, 103-118.
27. Schwartz J.T. (1980): "Ultracomputers". *ACM Transactions on Programming Languages and Systems*, 2, 484-521.
28. Siegel H.J. (1977): "Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks". *IEEE Transactions on Computers*, C-26, 153-161.
29. Siegel H.J. (1979): "A Model of SIMD Machines and a Comparison of Various Interconnection Networks". *IEEE Transactions on Computers*, C-28, 907-917.
30. Słomiński L. (1990): "Parallel Computing for Flexible Manufacturing Systems". W: Decision Making Models for Management and Manufacturing, red. R. Kulikowski, J. Stefański Omnitech Press, Warszawa, 215-229.
31. Słomiński L., Kaliszewski I. (1988): "Problemy obliczeń równoległych". *Prace IBS PAN*, 168.
32. Squire J.S., Palais S.M. (1963): "Programming and Design Considerations of a Highly Parallel Computer". *Proceedings of the AFIPS Spring Joint Computer Conference*, 23, 395-400.
33. Stone H.S. (1971): "Parallel Processing with Perfect Shuffle". *IEEE Transactions on Computers*, C-20, 153-161.
34. Sysło M.M. (1985): "Maszyny i algorytmy równoległe". *Raport Instytutu Informatyki Uniwersytetu Wrocławskiego*, 154.

35. Thompson C.D. (1979): "Time - Area Complexity for VLSI". *Proceedings of the 11th Ann. ACM Symposium on Theory of Computing*, 81-88.
36. Treleaven P.C. (1988): "Parallel Architecture. Overview". *Parallel Computing*, 8, 59-70.
37. Treleaven P.C., Browngridge D.R., Hopkins R.P. (1982): "Data -Driven and Demand-Driven Computer Architectures". *ACM Computing Surveys*, 14, 93-143.
38. Ullman J.D. (1984): Computational Aspects of VLSI. Computer Science Press, Rockville, Maryland.
39. Unger S.H. (1958): "A Computer Oriented Toward Spatial Problems". *Proceedings of IRE*, 46, 1744-1750.
40. Upfal E (1984): "A Probabilistic Relation Between Desirable and Feasible Models of Parallel Computation (preliminary version)". *Proceedings of the 16th Ann. ACM Symposium on Theory of Computing*, 258-265.
41. Vishkin U. (1983): "Implementation of Simultaneous Memory Access in Models that Forbid It". *Journal of Algorithms*, 4, 45-50.

Dodatkowe wskazówki literaturowe

a). Książki (Podręczniki, Monografie):

1. Almasi G.S., Gottlieb A.(1989): Highly parallel computing. The Benjamin/Cummings Publish.Co., California.
2. Ben - Ari M. (1989): Podstawy programowania współbieżnego. WNT, Warszawa.
3. Bertsekas D.P., Tsitsiklis J.N. (1989): Parallel and distributed computation. Numerical methods. Prentice-Hall Internat. Editions.
4. Błażewicz J. (1988): Złożoność obliczeniowa problemów kombinatorycznych. WNT, Warszawa.

5. Desrochers G.R. (1987): Principles of parallel and multi- processing. McGraw-Hill, N. York.
6. Gibbons A., Rytter W. (1989): Efficient parallel algorithms. Cambridge University Press, Cambridge.
7. Iszkowski W., Maniecki M.: (1982) Programowanie współbieżne. WNT, Warszawa.
8. Quinn M.J. (1987): Designing efficient algorithms for parallel computers. McGraw-Hill, N. York.
9. Woewodin W.W. (1986): Matematiczeskije modeli i metody w parallelnyh processach. Nauka, Moskwa.

b). Bibliografie

1. Kindervater G.A.P. and Lenstra J.K. (1984): "Parallel Algorithms". W: Combinatorial Optimization. Annotated Bibliography, red. M. O'hEigeartaigh, Lenstra J.K. Rinnoy Kan A.H.G., J. Wiley Ltd., Chichester, rozdz. 8.
2. Zenios S.A. (1987): "An Annotated Bibliography on Parallel Optimization". *Report 87-12-04, DSD The Wharton School of Business, University of Pennsylvania, Philadelphia.*
3. Zenios S.A. (1989): "Parallel Numerical Optimization: Current Status and an Annotated Bibliography". *ORSA J.Comput.* 1, 20-43.

c). Literatura podstawowa dotycząca transputerów

1. (1987): Transputer Reference Manual. INMOS Ltd, Prentice Hall New York.
2. (1988): Parallel C user guide. 3L Ltd, Edinburgh.
3. (1988): Parallel Fortran user guide. 3L Ltd, Edinburgh.
4. Pountain D. and May D. (1988): A tutorial introduction to OCCAM programming. Blackwell Scientific Publications, Ltd, London.

5. (1991): The T9000 transputer. Product overview. Manual. INMOS Ltd.

d). Czasopisma poświęcone w całości, lub w znacznym stopniu obliczeniom równoległym:

1. Algorithmica. An International Journal in Computer Science, (ukazuje się od 1990r).
2. Concurrency:Practice and Experience, (ukazuje się od 1991r).
3. IEEE Transactions on Computers, (ukazuje się od 1951).
4. IEEE Transactions on Software and Engineering, (ukazuje się od
5. International Journal of Parallel Programming, (pod tym tytułem ukazuje się od 1986r).
6. Journal of Parallel and Distributed Computing, (ukazuje się od 1984r).
7. Parallel Computing, (ukazuje się od 1984r).
8. Parallelogram. International Magazine of Super Performance Computing, (ukazuje się od 1990r).
9. Proceedings of the Annual Symposium on Foudations of Computer Science, (pod tym tytułem ukazuje się od 1975r).
10. Proceedings of the International Conference on Parallel Processing, (ukazuje się od 1971r).
11. Supercomputer. Published bimonthly by Amsterdam University Computing Center, (ukazuje się od 1984r).
12. Transputer Communications. J. Wiley & Sons Ltd., Wielka Brytania, (ukazuje się od 1993 r).

Dodatek 2

RAPORTY I PUBLIKACJE ZAKŁADU PROGRAMOWANIA MATEMATYCZNEGO IBS PAN DOTYCZĄCE RÓWNOLEGŁEJ OPTYMALIZACJI DYSKRETNEJ (1986-92)

I. Publikacje

1. Bertocchi M., Brandolini L., Słomiński L., Sobczyńska J. (1992): "A Monte-Carlo Approach for 0-1 Programming Problems". *Computing* 48, 259-274.
2. Dudziński K. (1986): "Wybrane równoległe algorytmy kombinatoryczne". *Roczniki PTM Seria III, Matematyka Stosowana XXVIII*, 91-123.
3. Kaliszewski I., Wojtowicz M. (1991) "Modest. A language for parallel programming". *Prace IPI PAN*, 691, Warszawa.
4. Słomiński L., Kaliszewski I. (1988): "Problemy obliczeń równoległych". *Prace IBS PAN*, 168, Warszawa.
5. Słomiński L. (1988): "Algorytmy równoległe znajdowania minimaksowych rozgałęzień". *Zeszyty Naukowe Politechniki Śląskiej, ser. Automatyka* 94, 287-301.
6. Słomiński L. (1988): "Obliczenia równoległe i optymalizacja". W: Materiały Konferencji Naukowej: Problemy współczesnej radiolokacji, WAT, Warszawa, 124-130.
7. Słomiński L. (1989): "Parallel Algorithms for Optimization". W: Proceedings of the 3rd Polish-Finish Symp. on Methodology and Applications of Decision Support Systems, red. R. Kulikowski, IBS PAN, Warszawa, 219-233.
8. Słomiński L. (1990): "Implementacje algorytmów kombinatorycznych na symulowanych sieciach wielomikroprocesorowych".

Zeszyty Naukowe Politechniki Śląskiej, ser. Automatyka, 100, 287-300.

9. Słomiński L. (1990): "Parallel Computing for Flexible Manufactoring Systems". W: Decision Making Models for Management and Manufacturing, red. R. Kulikowski, Omnitech Press, Warszawa, 215-229.
10. Słomiński L. (1990): "Komputery równoległe dla rozwiązywania zadań optymalizacji dyskretnej i sztucznej inteligencji". Materiały konferencji "*Sztuczna inteligencja w zarządzaniu*", Warszawa, listopad 1989, PTBiOS i IBS PAN, red. A.Straszak, Z. Nahorski, C. Iwański, Warszawa, 201-210.

II. Raporty (latami, wg alfabetu)

1986

1. Dudziński K.: "Obliczenia równoległe: Wybrane algorytmy kombinatoryczne". ZPM 20/86, IBS PAN, Warszawa.
2. Grygiel G.: "Obliczenia równoległe: Algorytm równoległy dla zadań przydziału". ZPM 24/86, IBS PAN, Warszawa.
3. Kaliszewski I.: "Obliczenia równoległe: Modele obliczeń w języku Ameba-Pascal". ZPM 23/86, IBS PAN, Warszawa.
4. Kaliszewski I., Słomiński L.: "Obliczenia równoległe: przegląd zagadnień". ZPM 17/86, IBS PAN, Warszawa.
5. Majchrzak J.: "Obliczenia równoległe: uwarunkowania i prognozy". ZPM 21/86, IBS PAN, Warszawa.
6. Słomiński L.: "Obliczenia równoległe: algorytm równoległy wyznaczania dendrytu minimaksowego w grafie skierowanym". ZPM 19/86, IBS PAN, Warszawa.

7. Waluk B.: "Obliczenia równoległe: Zastosowanie niektórych metod dekompozycji macierzy współczynników do rozwiązywania układów równań". ZPM 22/86, IBS PAN, Warszawa.

1987

1. Grygiel G.: "Zadanie przydziału: algorytm równoległy i eksperymentalna ocena liczby iteracji". ZPM 34/87, IBS PAN, Warszawa.
2. Kaliszewski I.: "Parallel counterparts of algorithms for determining minimal elements of discrete sets". ZPM 30/87, IBS PAN, Warszawa.
3. Kaliszewski I., Wojtowicz M.: "Język Modest". ZPM 31/87, IBS PAN, Warszawa.
4. Kulczycka D.: "Równoległe algorytmy b-skojarzeń w drzewach". ZPM 23/87, IBS PAN, Warszawa.
5. Majchrzak J., Skawiński A.: "Sieci komputerowe - przegląd literatury i ocena możliwości przetwarzania równoległego". ZPM 17/87, IBS PAN, Warszawa.
6. Majchrzak J., Skawiński A.: "Pakiet PPLANEUP do organizacji przetwarzania równoległego w sieci D-Link komputerów IBM PC/XT/AT". ZPM 18/87, IBS PAN, Warszawa.
7. Majchrzak J., Skawiński A.: "Propozycja bezgradientowej metody równoległej optymalizacji nieliniowej bez ograniczeń". ZPM 19/87, IBS PAN, Warszawa.
8. Słomiński L.: "Narzędzia optymalizacji równoległej na maszynach sekwencyjnych". ZPM 13/87, IBS PAN, Warszawa.
9. Słomiński L.: "Problemy równoległych algorytmów dla przepływu maksymalnego w sieci". ZPM 14/87, IBS PAN, Warszawa.

1988

1. Gondek H., Słomiński L.: "Implementacja na symulowanej maszynie równoległej typu dwudrzewo algorytmu Dijkstry wyznaczania drzewa dróg najkrótszych w digrafie". ZPM 37/88, IBS PAN, Warszawa.
2. Grygiel G., Kulczycka D.: "Ocena systemu symulacyjnego JUPITER-86". ZPM 24/88, IBS PAN, Warszawa.
3. Grygiel G.: "Symulacja algorytmu równoległego dla zadania przydziału przy pomocy pakietu JUPITER-86". ZPM 44/88, IBS PAN, Warszawa.
4. Kaliszewski I., Kulczycka D., Słomiński L.: "Równoległy algorytm Prima-Dijkstry wyznaczania dendrytu minimalnego: implementacja na symulowanej strukturze typu dwudrzewo". ZPM 5/88, IBS PAN, Warszawa.
5. Kaliszewski I., Słomiński L.: "Problemy obliczeń równoległych". ZPM 38/88, IBS PAN, Warszawa.
6. Kulczycka D.: "Algorytmy równoległe b-skojarzeń w drzewach: implementacja w systemie symulacyjnym Jupiter". ZPM 6/88, IBS PAN, Warszawa.
7. Majchrzak J.: "Pakiet PPLANEUP wersja 2 dla wspomagania przetwarzania równoległego w sieciach D-Link komputerów IBM PC/XT/AT". ZPM 41/88, IBS PAN, Warszawa.
8. Słomiński L.: "Algorytmy równoległe znajdowania minimaksowych rozgałęzień". ZPM 1/88, IBS PAN, Warszawa.
9. Słomiński L.: "Obliczenia równoległe i optymalizacja". ZPM 4/88, IBS PAN, Warszawa.

1989

1. Gondek H., Słomiński L.: "Symulowane odmiany struktury dwudrzewa dla algorytmów najkrótszych dróg w digrafie". ZPM 10/89, IBS PAN, Warszawa.

2. Gondek H.: "Równoległy algorytm mnożenia macierzy. Implementacja i eksperyment obliczeniowy na symulowanej strukturze typu mesh i hypercube o n^2 procesorach". ZPM 11/89, IBS PAN, Warszawa.
3. Grygiel G.: "Równoległość na poziomie podprocedur na przykładzie zadania przydziału". ZPM 13/89, IBS PAN, Warszawa.
4. Grygiel G.: "Program AZP rozwiązujący algebraiczne zagadnienie przydziału - opis funkcjonalny". ZPM 18/89, IBS PAN, Warszawa.
5. Kaliszewski I.: "VM1 - A parallel algorithms to determine minimal elements of discrete sets". ZPM 16/89, IBS PAN, Warszawa.
6. Kulczycka D.: "Algorytmy równoległe w sieci drzew ortogonalnych - realizacja w systemie symulacyjnym JUPITER-86". ZPM 12/89, IBS PAN, Warszawa.
7. Majchrzak J.: "O programowaniu dla transputerów INMOS T800". ZPM 14/89, IBS PAN, Warszawa.
8. Słomiński L.: "Parallel Algorithms for Optimization". ZPM 1/89, IBS PAN, Warszawa.
9. Słomiński L.: "Komputery równoległe dla rozwiązywania zadań optymalizacji i sztucznej inteligencji". ZPM 3/89, IBS PAN, Warszawa.
10. Słomiński L., Sobczyńska J.: "Programy i dane w języku Fortran 77 dla wybranych zadań dyskretnych, z uwzględnieniem potrzeb wektoryzacji". ZPM 4/89, IBS PAN, Warszawa.
11. Sobczyńska J.: "Charakteryzacja wieloprocesorowej struktury Perfect Shuffle. Równoległy algorytm mnożenia macierzy i jego implementacja w tej strukturze". ZPM 9/89, IBS PAN, Warszawa.

1990

1. Grygiel G.: "Implementacja asynchronicznego algorytmu dla zadania przydziału". ZPM 26/90, IBS PAN, Warszawa.

2. Kaliszewski I.: "On determining minimal elements of discrete sets on a network of transputers". ZPM 4/90, IBS PAN, Warszawa.
3. Kulczycka D.: "Prosty algorytm równoległy znajdowania skojarzenia maksymalnego". ZPM 18/90, IBS PAN, Warszawa.
4. Majchrzak J.: "Transputer i sieci wielotransputerowe". ZPM 30/90, IBS PAN, Warszawa.
5. Słomiński L.: "Komunikacja w systemach wieloprocesorowych i jej wpływ na efektywność obliczeń". ZPM 22/90, IBS PAN, Warszawa.

1991

1. Bertocchi M., Butti A., Sergi P., Słomiński L., Sobczyńska J.: "Stochastic and Deterministic Optimization on 0-1 Multiknapsack Problem". *Quaderni del Dipartimento di Mathematica, Statistica, Informatica e Applicazioni*, No 14/91, Bergamo.
2. Kulczycka D.: " Algorytm O(|E|) skojarzenia w grafie dwudzielnym". ZPM 13/91, IBS PAN, Warszawa.

1992

1. Bertocchi M., Butti A., Słomiński L.: "Fixed Precision Random Search Procedure for the Binary Multiknapsack Problem". (Revised version). *Quaderni del Dipartimento di Mathematica, Statistica, Informatica e Applicazioni*, No 18/92, Bergamo.
2. Grygiel G., Kabanow P., Słomiński L., Sobczyńska J.: "Wykorzystanie sieci transputerowych do rozwiązania wielowymiarowego zadania załadunku metodą Monte-Carlo". /w języku rosyjskim/ ZPM 12/92, IBS PAN, Warszawa.