ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992
Seria: INFORMATYKA z.16 Nr kol.1140

Zdzisław SZCZERBIŃSKI

OCENA EFEKTYWNOŚCI FUNKCJONOWANIA KOMPUTERÓW O ARCHITEKTURZE RÓWNOLEGŁEJ (PRZEGLĄD)

Streszczenie. Ocena efektywności funkcjonowania komputerów równoległych jest obecnie istotnym zagadnieniem wobec szybkiego rozwoju i upowszechnienia tych struktur. Praca stanowi przegląd teoretycznych dokonań w tej dziedzinie. W pierwszej części zaprezentowano klasyfikację architektur równoległych, ze szczególnym uwzględnieniem systemów SIMD typu potokowego (wektorowego) i systemów wieloprocesorowych (MIMD) oraz krótkim omówieniem pozostałych typów. Skrótowo naszkicowano zagadnienie superkomputerów, ilustrując je tabelą istniejących obecnie maszyn tego typu. Następnie omówiono ogólnie metody charakteryzacji wydajnościowej. W dalszej części przedstawiono następujące modele analityczne wyżej wymienionych struktur:

- model Calahana-Amesa,
- model Bucher-Simmons,
- model Hockneya.

Ostatni model, jako obecnie najpopularniejszy, omówiony został dokładniej. Po prezentacji wersji klasycznej (dla jednopotokowych procesorów wektorowych), opisano wersję wielopotokową; uwzględniono również przesyły procesor-pamięć. Następnie przedstawiono wersję wieloprocesorową. Prezentacji materiału towarzyszyły wnioski dotyczące sensu (nie zawsze istniejącego) implementacji algorytmów w formie równoległej, co poparto prezentacją, w załączniku, prawa Amdahla i jego interpretacji.

Summary. Performance evaluation of parallel computers is an important issue in the face of the fast development and general availability of high-speed computing structures. The paper is a survey of theoretical advances in this field. In the first part, the parallel architecture classification is presented, with emphases placed upon pipeline SIMD (vector) and MIMD (multiprocessing) systems; a short description of other classes is given. An outline of supercomputers is given, illustrated by a table of current commercial machines. The following chapter discusses methods of performance characterization. Subsequently, the following analytical models of the abovementioned structures are presented:

- Calahan-Ames's,
- Bucher-Simmons's,
- Hockney's.

Hockney's model, as currently the most frequently used one, has been elaborated. Upon introduction to the original version (for single-pipe vector processors), the multipipe model is described; also, the processor-memory transfers are considered. Next, the multiprocessor version of Hockney's model is analyzed. Conclusions regarding the point in implementing algorithms in the parallel form are made. In the appendix, Amdahl's law is presented and interpreted.

Резюме. Оценка производительности параллельных вычислительных систем является существенной проблемой ввиду быстрого развития и все более общего применения высокоскоростных ЗВМ. Статья составляет обозрение теоретических достижений в этой области. В первой части представлена классификация параллельных архитектур, с особенным положением упора на конвейерные системы ОКМД (векторные процессоры) и системы МКМД (многопроцессорные) и кратким обзором других классов. Помещены основные сведения о супер ЗВМ. Дальше, в общих чертах обсуждены методы характеризации производительности. В следующей части приведены аналитические модели вышеупомянутх структур:

- модель Калахана-Амеса,
- модель Бухер-Симмонс,
- модель Хокни.

Последняя модель обсуждена более подробно. После представления ее классической версии (для одноконвейерных векторных процессоров), обсуждена ее многоконвейерная разновидность, рассмотрены также пересылы процессор - память. Дальше, проанализирована многопроцессорная модель Хокни. Теоретическому анализу сопутствуют выводы касающиеся смысла вводения алгоритмов в параллельной форме. В приложении представлен закон Амдала.

1. WSTĘP

Dokonujący się w świecie w ostatnich latach dynamiczny rozwój komputerów o architekturach odbiegających od klasycznego, sekwencyjnego modelu von Neumanna oraz pojawiające się już w Polsce pierwsze urządzenia tego typu (karty transputerów, specjalizowane procesory potokowe do przetwarzania obrazów itp.) sprawiają, że istotny staje się problem oceny efektywności funkcjonowania (oceny wydajności) takich systemów. Dokonanie takiej oceny jest konieczne (lub przynajmniej wskazane), zanim przystąpi się do implementacji określonych algorytmów numerycznych na określonej maszynie równoległej, może się bowiem w wielu wypadkach okazać, że zysk czasowy w działaniu programów, będący główną motywacją konstrukcji architektur równoległych, jest nieznaczny lub nawet żaden.

W odróżnieniu od znanych z literatury (np. [8;20;5]) metod oceny wydajności dużych systemów komputerowych, których podstawą są zazwyczaj modele teorii masowej obsługi (modele kolejkowe), a zadaniem - oszacowanie, dla całego systemu lub pewnych jego elementów, takich miar, jak przepustowość (mierzona np. liczbą obsługiwanych w jednostce czasu programów lub zgłoszeń napływających od użytkowników pracujących w trybie interaktywnym) czy wykorzystanie (rozumiane jako procent czasu, przez jaki system realizuje wspomnianą obsługę użytkowników, w stosunku do całego czasu pracy), metody oceny efektywności funkcjonowania maszyn równoległych koncentrują się na pojedynczym programie, realizującym określony algorytm (zazwyczaj numeryczny) i zawierającym dużą liczbę działań na liczbach zapisanych w postaci zmiennoprzecinkowej. Program taki jest bardzo czasochłonny, jeśli realizować go na procesorze sekwencyjnym (czasem nawet niemożliwy do realizacji w rozsądnym czasie). Dla maszyny o architekturze równoległej zadaniem wspomnianych metod jest oszacowanie czasu pracy takiego programu i wynikającej stąd szybkości obliczeń, mierzonej w milionach operacji zmiennoprzecinkowych na sekundę (Mflops).

W celu zobrazowania wspomnianych różnic w metodzie oceny wydajności systemu komputerowego, posłużymy się przykładem ośrodka obliczeniowego wyposażonego w 4-procesorowy superkomputer. Z punktu widzenia kierownika instalacji, system pracuje wydajnie, jeśli obsługuje maksymalną liczbę użytkowników - kieruje się więc pierwszą metodyką (masowej obsługi) i odpowiednio ustawiając parametr instalacyjny, nie zezwala użytkownikom na wykonanie programów w trybie wieloprocesorowym, gdyż w takim przypadku jeden program blokowałby wszystkie procesory i pozostali użytkownicy musieliby czekać (w ostatecznym rachunku czasowym nie stanowiłoby to dla wspomnianego kierownika różnicy, gdyby nie fakt dużych często i omówionych w dalszej części artykułu nakładów czasowych związanych z synchronizacją pracy procesorów i komunikacją między nimi w trakcie realizacji programu w trybie wieloprocesorowym).

W wyniku decyzji kierownika instalacji, każdemu programowi przydziela się więc tylko jeden procesor, co z kolei nie satysfakcjonuje użytkownika, który pragnąłby otrzymać wyniki swego programu jak najszybciej, preferuje więc pracę współbieżną na wszystkich 4 procesorach. Mamy tu więc do czynienia z drugą metodyką (oceny komputera z punktu widzenia minimalizacji czasu obliczeń

własnego programu). Jak widać z powyższego przykładu, obie metodyki są nie tylko różne, lecz wręcz sprzeczne.

Artykuł poniższy poświęcony jest analizie wydajnościowej maszyn równoległych, nakreślonej zgodnie z ich klasycznymi modelami. Rozdział 2 stanowi przegląd architektur równoległych, konieczny dla pełnego zrozumienia części analitycznej, stanowiącej treść rozdziałów 4-6 i poprzedzonej krótkim omówieniem typowych parametrów wydajnościowych i metod ich wyznaczania (rozdział 3). Uwagi końcowe zawarto w rozdziale 7.

2. KOMPUTERY O ARCHITEKTURZE RÓWNOLEGŁEJ

2.1. Podział komputerów wg organizacji algorytmicznej

Komputery można ogólnie podzielić, pod kątem organizacji obliczeń w ich procesorach, na 4 grupy [6]:

a) sekwencyjne,

b) grupowo-sekwencyjne,

c) o strumieniach luźno powiązanych,

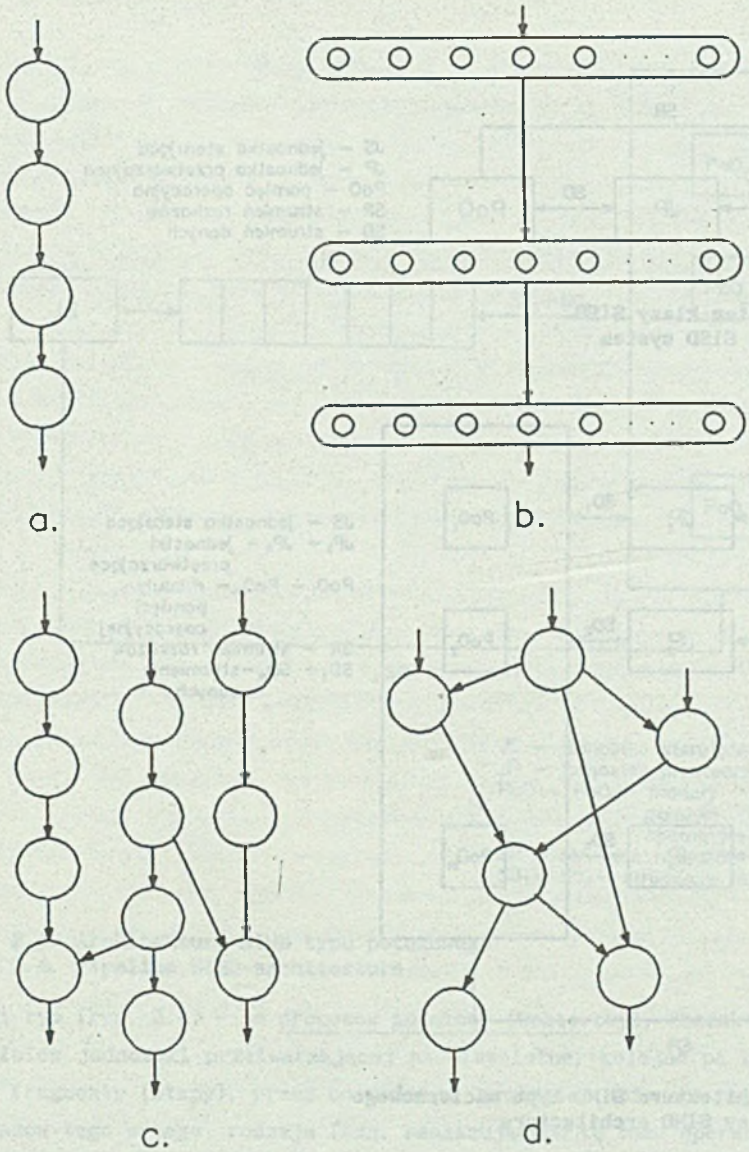
d) ogólnie współbieżne.

Struktury algorytmiczne odpowiadające powyższym grupom przedstawiono na rys. 2.1 [6].

ad a) W procesorze sekwencyjnym rozkazy wykonywane są kolejno, jeden za drugim. Odpowiada to klasycznej architekturze von Neumanna, określanej w klasyfikacji Flynna [9] symbolem SISD (Single Instruction stream / Single Data stream) i przedstawionej schematycznie na rys. 2.2 [17].

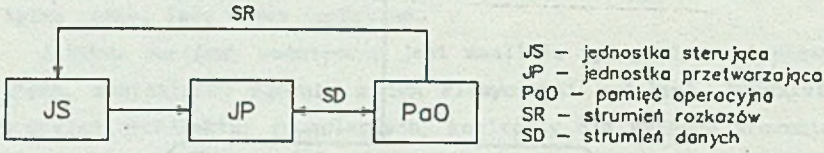
Szybkość procesora typu SISD jest ograniczona jego sekwencyjną naturą i określona przez czas wykonania pojedynczego rozkazu. Przykładami architektur SISD są procesory maszyn IBM 360, RIAD, ICL 1900, ODRA, procesory Intel 8086/8088.

ad b) Procesor grupowo-sekwencyjny wykonuje sekwencyjnie grupy rozkazów. W ramach grupy rozkazy są wykonywane całkowicie równolegle (równocześnie) lub częściowo równolegle (potokowo). W klasyfikacji Flynna odpowiada to architekturze SIMD (Single Instruction stream / Multiple Data stream). Istnieją dwa typy systemów SIMD, przedstawione na rys. 2.3 [17] i 2.4.

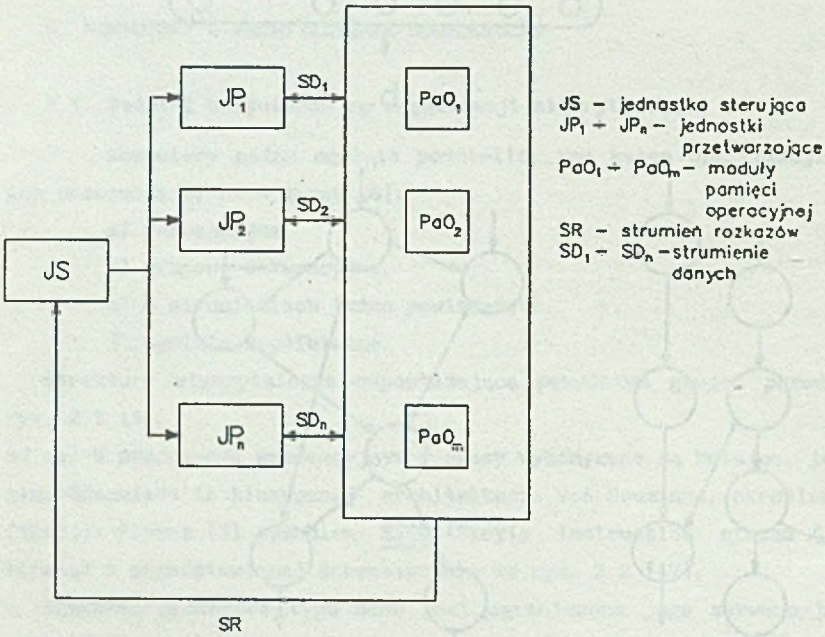


Rys. 2.1. Struktury algorytmiczne
a) sekwencyjna, b) grupowo-sekwencyjna,
c) o strumieniach luźno powiązanych, d) ogólnie współbieżna

Fig. 2.1. Algorithmic structures
a) sequential, b) group-sequential,
c) loosely coupled streams, d) general concurrent

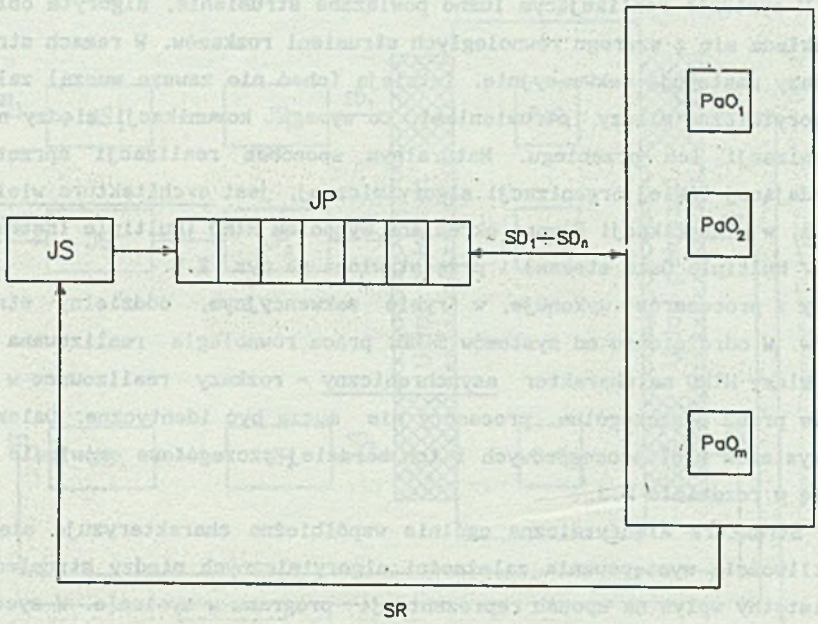


Rys. 2.2. System klasy SISD
Fig. 2.2. The SISD system



Rys. 2.3. Architektura SIMD typu macierzowego
Fig. 2.3. Array SIMD architecture

Pierwszy typ architektury SIMD (rys. 2.3) określany jest mianem procesora macierzowego. Charakteryzuje się on zwielokrotnieniem jednostek przetwarzających (arytmetyczno-logicznych), wykonujących w danym momencie ten sam rozkaz.



JS – jednostka sterująca
JP – jednostka przetwarzająca
$PaO_1 \div PaO_m$ – moduły pamięci operacyjnej
SR – strumień rozkazów
$SD_1 \div SD_n$ – strumienie danych

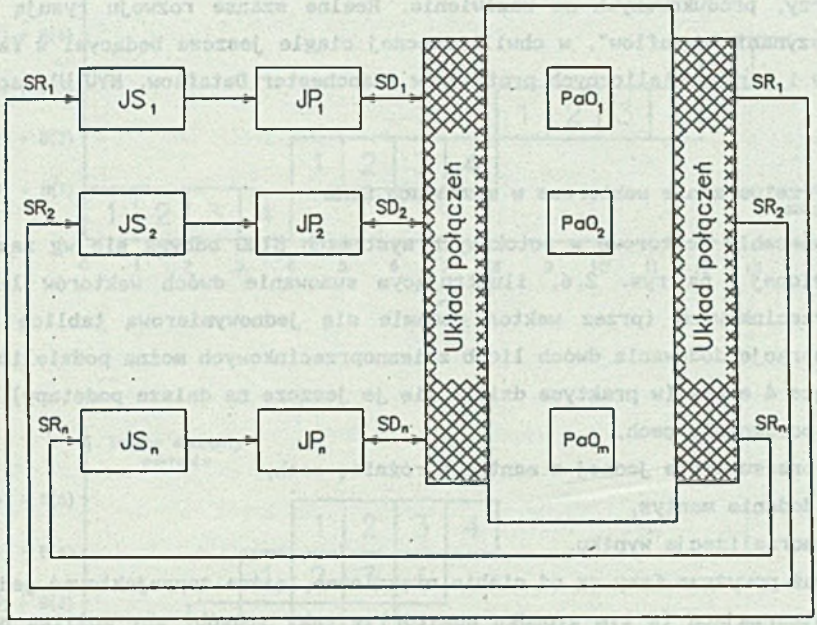
Rys. 2.4. Architektura SIMD typu potokowego
Fig. 2.4. Pipeline SIMD architecture

Drugi typ (rys. 2.4) - to procesor potokowy (wektorowy), charakteryzujący się podziałem jednostki przetwarzającej na niezależne, kolejno po sobie następujące fragmenty (etapy), przez co w danym momencie może ona wykonywać kilka rozkazów tego samego rodzaju (tzn. realizujących tę samą operację), na zasadzie fabrycznej linii montażowej. Wspólna dla obu typów SIMD jest synchroniczność pracy równoległej - na wszystkich strumieniach danych jest w określonym momencie wykonywana ta sama operacja. Szczegółowe omówienie procesora potokowego znajduje się w rozdziale 2.2.

ad c) W systemie realizującym luźno powiązane strumienie, algorytm obliczeniowy składa się z szeregu równoległych strumieni rozkazów. W ramach strumieni rozkazy następują sekwencyjnie. Istnieją (choć nie zawsze muszą) zależności algorytmiczne między strumieniami, co wymaga komunikacji między nimi i synchronizacji ich przebiegu. Naturalnym sposobem realizacji sprzętowej, odpowiadającej takiej organizacji algorytmicznej, jest architektura wieloprocesorowa, w klasyfikacji Flynna określana symbolem MIMD (Multiple Instruction stream / Multiple Data stream) i przedstawiona na rys. 2.5.

Każdy z procesorów wykonuje, w trybie sekwencyjnym, oddzielny strumień rozkazów. W odróżnieniu od systemów SIMD, praca równoległa realizowana przez system klasy MIMD ma charakter asynchroniczny - rozkazy realizowane w danym momencie przez poszczególne procesory nie muszą być identyczne. Dalszy podział systemów wieloprocesorowych i ich bardziej szczegółowe omówienie znajdują się w rozdziale 2.3.

ad d) Struktura algorytmiczna ogólnie współbieżna charakteryzuje się dużą częstotliwością występowania zależności algorytmicznych między strumieniami. Ma to istotny wpływ na sposób reprezentacji programu w systemie. W systemach przedstawionych w punktach a), b) i c) zachowana została, w mniejszym lub większym stopniu, liniowa kolejność obliczeń, wynikająca z kolejności adresów odpowiadających rozkazów w pamięci (kolejność ta naruszana jest tylko przez rozkazy skoków). Nad kolejnością wykonywania rozkazów czuwa licznik rozkazów (w systemie MIMD istnieją, oczywiście, oddzielne liczniki dla poszczególnych procesorów). Powyższa zasada (von Neumanna) określa reguły funkcjonowania opisanych wcześniej maszyn, nazywanych również maszynami sterowanymi przepływem rozkazów (ang. control flow). Można zauważyć, że architektura von Neumanna nie jest optymalna dla obliczeń równoległych, gdyż pewne operacje, niezależne od wcześniej wykonywanych, można by wykonać bez czekania na wyniki tych wcześniejszych. Stąd, aby maksymalnie wykorzystać paralelizm (równoległość) programów, pojawił się projekt maszyny sterowanej przepływem danych (ang. dataflow), w której wykonanie rozkazu następuje w momencie, gdy argumenty rozkazu stają się osiągalne. W maszynie takiej nie istnieje licznik rozkazów, a fizyczne położenie rozkazu w programie nie ma żadnego znaczenia - rozkazy nie są więc uporządkowane. Wykonaniem programu steruje wyłącznie zależność algorytmiczna między danymi. Ponieważ w maszynie "dataflow" nie ma w ogóle mowy o



$JS_1 \div JS_n$ – jednostki sterujące
$JP_1 \div JP_n$ – jednostki przetwarzające
$PaO_1 \div PaO_m$ – moduły pamięci operacyjnej
$SR_1 \div SR_n$ – strumienie rozkazów
$SD_1 \div SD_n$ – strumienie danych

Rys. 2.5. Architektura MIMD
Fig. 2.5. The MIMD architecture

strumieniach rozkazów i danych, ten typ architektury nie jest w klasyfikacji Flynna (powstałej zresztą znacznie wcześniej) w ogóle reprezentowany.

W dalszej części artykułu analizowane będą tylko następujące typy architektury:

- SIMD typu potokowego (nazywana dalej krótko SIMD),
- MIMD.

Uzasadnienie jest następujące: Spośród przedstawionych typów architektur równoległych (punkty b), c), d)), tylko dwa powyższe znalazły istotnie szerokie zastosowanie i zaowocowały licznymi implementacjami, produkowanymi seryj-

nie. Pozostałe architektury nie wyszły poza fazę prototypów czy pojedynczych egzemplarzy, produkowanych na zamówienie. Realne szanse rozwoju rysują się przed maszynami "dataflow", w chwili obecnej ciągle jeszcze będącymi w fazie projektów i bardzo nielicznych prototypów (Manchester Dataflow, NYU Ultracomputer).

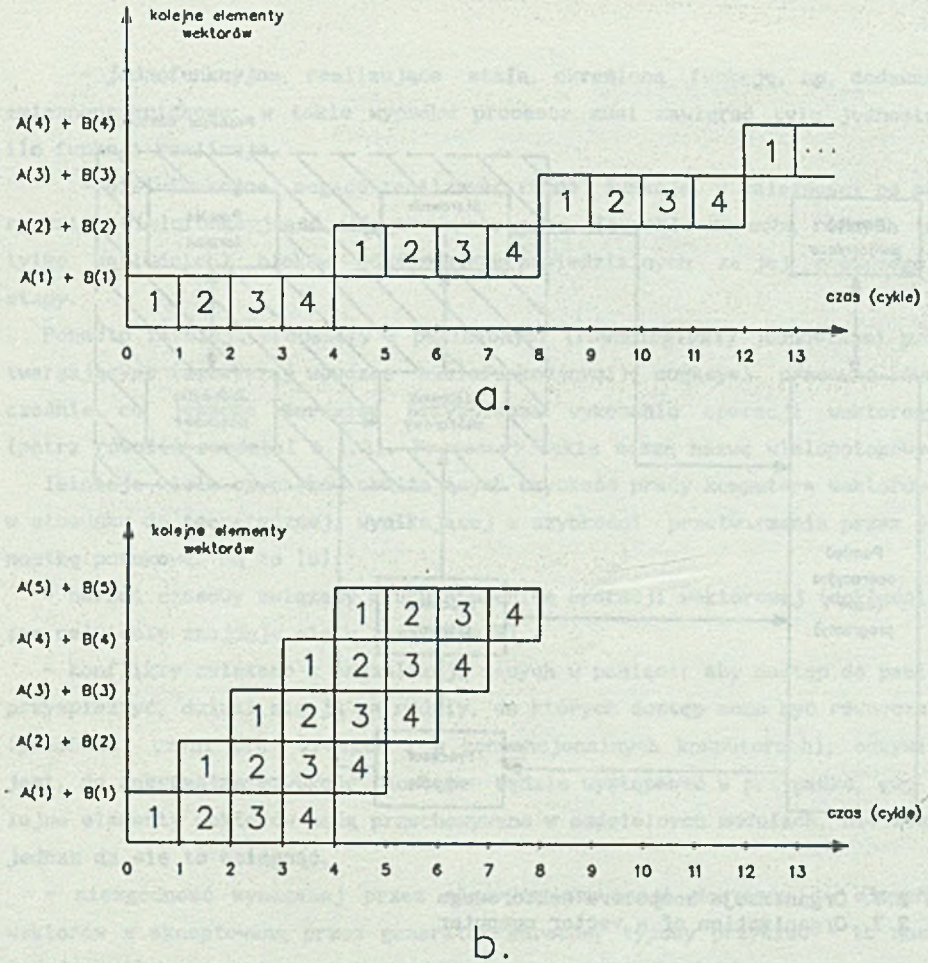
2.2. Przetwarzanie wektorowe w systemach SIMD

Przetwarzanie wektorowe w potokowych systemach SIMD odbywa się wg zasady przedstawionej na rys. 2.6, ilustrującym sumowanie dwóch wektorów liczb zmiennoprzecinkowych (przez wektor rozumie się jednowymiarową tablicę danych). Operację dodawania dwóch liczb zmiennoprzecinkowych można podzielić na następujące 4 etapy (w praktyce dzieli się je jeszcze na dalsze podetapy):

1) porównanie cech,
2) przesunięcie jednej z mantys o różnicę cech,
3) dodanie mantys,
4) normalizacja wyniku.

Ponieważ powyższe fazy są od siebie niezależne, można zaprojektować jednostkę przetwarzającą (w tym wypadku sumator) w taki sposób, aby możliwa była równoczesna praca nad czterema parami liczb, przy czym każda para w danym momencie zajmuje jedną fazę, w następnym etapie następną itd. (podobnie jak w fabrycznej linii montażowej). W ten sposób proces sumowania wektorów takich liczb ulega wyraźnemu przyspieszeniu, kolejny wynik (rezultat sumowania dwóch liczb) uzyskujemy bowiem co fazę, jak na rys. 2.6.b), a nie co 4 fazy, jak na rys. 2.6.a). W podobny, jak sumowanie, sposób można rozbić na fazy (i dokonać adekwatnej konstrukcji jednostek przetwarzających) inne operacje arytmetyczne i logiczne.

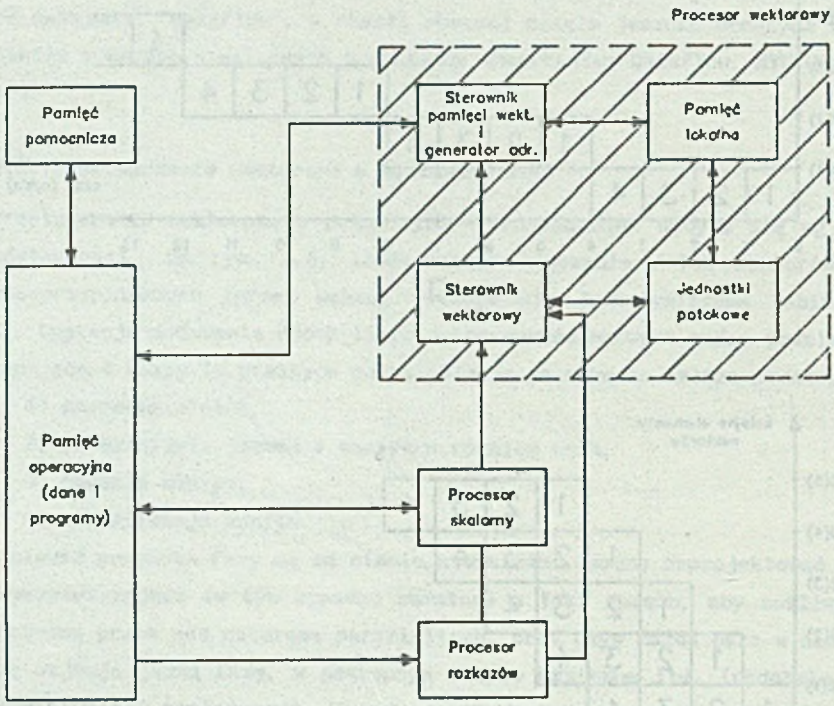
Na rys. 2.7 przedstawiono organizację typowego komputera wektorowego [7]. Jednostki potokowe realizują przetwarzanie wektorów według wyżej opisanego schematu. Sterownik pamięci wektorów i generator adresów odpowiedzialne są za organizację przesyłu wektorów z pamięci do procesora (wektory danych) i w kierunku odwrotnym (wektory wyników). Sterownik wektorowy inicjuje operacje wektorowe na podstawie parametrów otrzymanych w rozkazie wektorowym z procesora rozkazów. Przez rozkaz wektorowy rozumie się rozkaz dotyczący operacji na całym wektorze (wektorach) danych (np. analizowane powyżej dodawanie dwóch



Rys. 2.6. Sumowanie dwóch wektorów liczb zmiennoprzecinkowych
a) przez sumator konwencjonalny
b) przez sumator potokowy

Fig. 2.6. Addition of two floating point vectors as performed by a
a) conventional adder
b) pipelined adder

wektorów). Procesor rozkazów rozdziela rozkazy wektorowe od normalnych (skalarnych), przesyłając je odpowiednio do procesora wektorowego lub skalarnego. W celu przyspieszenia operacji komputera, rozkazy skalarne są często wykonywane równocześnie z wektorowymi (o ile, oczywiście, są one od siebie niezależne), czemu sprzyja rodzielenie procesora skalarnego od wektorowego. Występu-



Rys. 2.7. Organizacja komputera wektorowego
Fig. 2.7. Organization of a vector computer

jąca na schemacie pamięć lokalna, to zespół rejestrów wektorowych przechowujących wektory w całości lub w dużych "kawałkach" (typowa długość rejestru - 64 liczby 8-bajtowe) i umożliwiających ograniczenie liczby przesyłów z/do pamięci.

Komputery wektorowe różnią się między sobą przede wszystkim szybkością przetwarzania danych w jednostkach potokowych (zdeterminowaną czasem cyklu zegara), obecnością rejestrów wektorowych (architektura "rejestr-rejestr") lub ich brakiem (architektura "pamięć-pamięć") (w pierwszym przypadku również liczbą i długością tch rejestrów) oraz samym charakterem jednostek potokowych. Istnieją dwa typy tych jednostek:

- jednofunkcyjne, realizujące stałą, określoną funkcję, np. dodawanie zmiennoprzecinkowe; w takim wypadku procesor musi zawierać tyle jednostek, ile funkcji realizuje,

- wielofunkcjne, mogące realizować różne funkcje, w zależności od sterowania; wielofunkcyjność osiąga się tu przez łączenie ze sobą różnych (nie tylko sąsiednich) bloków jednostki odpowiedzialnych za jej poszczególne etapy.

Ponadto istnieją procesory z powielonymi (równoległymi) jednostkami przetwarzającymi (zazwyczaj wówczas wielofunkcyjnymi), mogącymi pracować równocześnie, co jeszcze bardziej przyspiesza wykonanie operacji wektorowych (patrz również rozdział 6.1.2). Procesory takie noszą nazwę wielopotokowych.

Istnieje wiele czynników obniżających szybkość pracy komputera wektorowego w stosunku do teoretycznej, wynikającej z szybkości przetwarzania przez jednostkę potokową. Są to [6]:

- narzut czasowy związany z uruchomieniem operacji wektorowej (dokładniejsze omówienie znajduje się w rozdziale 4),

- konflikty związane z organizacją danych w pamięci; aby dostęp do pamięci przyspieszyć, dzieli się ją na moduły, do których dostęp może być równoczesny (podobnie czyni się zresztą i w konwencjonalnych komputerach); oczywiste jest, że maksymalna szybkość dostępu będzie występować w przypadku, gdy kolejne elementy wektorów będą przechowywane w oddzielnych modułach, nie zawsze jednak da się to osiągnąć,

- niezgodność wymaganej przez algorytm sekwencji dostępów do elementów wektorów z akceptowaną przez generator adresów; typowy przykład - to dostęp do elementów rozmieszczonych w różnych odstępach od siebie w momencie, gdy generator adresów umożliwia dostęp tylko do elementów kolejnych lub rozmieszczonych ze stałym odstępem,

- wzajemna zależność między operacjami skalarnymi i wektorowymi (wspomniana wyżej); prowadzi ona do "zatrzymania" jednostki (jednostek) potokowej na czas wykonywania operacji skalarnej.

Przykładami komputerów wektorowych są: CRAY-1, Cyber 205, SX-2, FACOM VP-200, Hitachi S-810/20 (patrz też tablica 2.1 w rozdziale 2.4).

2.3. Systemy wieloprocesorowe

Systemy wieloprocesorowe (MIMD) można ogólnie podzielić na 2 klasy:

- ze wspólną (dzieloną) pamięcią, zwane także silnie powiązanymi (ang. tightly coupled),
- z pamięcią rozproszoną (przesyłem komunikatów), zwane również luźno powiązanymi (ang. loosely coupled).

Powyższy podział wynika z mechanizmu synchronizacji i komunikacji między procesami realizowanymi w procesorach. W systemie wieloprocesorowym z pamięcią dzieloną komunikacja odbywa się poprzez dostęp do wspólnych, dla komunikujących się procesorów, obszarów pamięci. W systemie z pamięcią rozproszoną każdy procesor ma własną pamięć i komunikacja między procesorami odbywa się na zasadzie przesyłu komunikatów (podobnie jak w sieciach komputerowych). Ogólnie schematy obu architektur przedstawiono na rys. 2.8

W systemie wieloprocesorowym ze wspólną pamięcią nieuniknione są zwłoki czasowe wynikające z przesyłu danych przez wspólną sieć połączeń oraz z konfliktów dostępu do pamięci. Aby je zminimalizować, projektuje się sieci połączeń tak, aby zapewniały wysoką przepustowość; również pamięć wymaga szczególnej organizacji. Typowe rozwiązania, to:

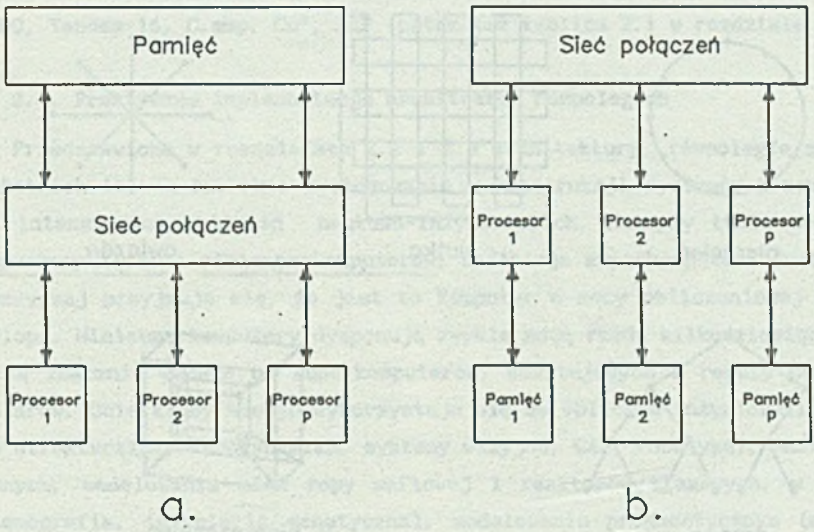
- systemy ze wspólną magistralą,
- systemy z przełącznicą krzyżową,
- systemy z pamięcią wieloportową,
- systemy z wieloetapową siecią połączeń.

Szczegółowe omówienie powyższych zagadnień znaleźć można w [22], [17].

Systemy wieloprocesorowe z przesyłem komunikatów są koncepcyjnie nowsze od poprzednio omówionych. Istnieje wiele rozwiązań sieci połączeń między procesorami. Typowe połączenia to (rys. 2.9):

- pierścień,
- siatka,
- drzewo,
- gwiazda,
- hipersześcian.

We wszystkich powyższych systemach komunikaty przechodzą przez węzły (procesory) pośrednie. Na szczególną uwagę zasługuje hipersześcian, gdyż liczba węzłów pośrednich jest w nim sprowadzona do minimum. Hipersześcian może być

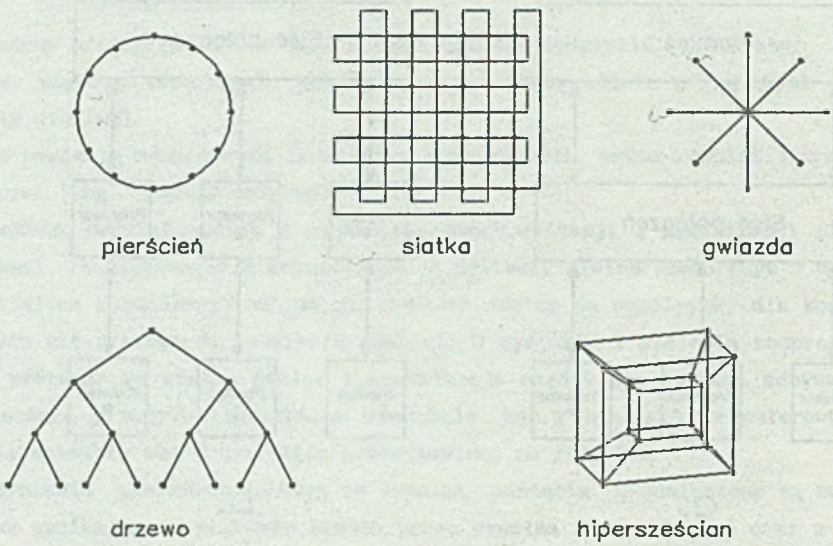


Rys. 2.8. Organizacja systemu wieloprocesorowego
a) ze wspólną pamięcią
b) z pamięcią rozproszoną

Fig. 2.8. Multiprocessor system organization
a) with shared memory
b) with distributed memory

ogólnie n-wymiarowy (na rys. 2.9 jest on 4-wymiarowy); liczba procesorów $p = 2^n$. Dodatkową zaletą jest mała (równa n) liczba połączeń obsługiwanych przez pojedynczy procesor. Architektury oparte na zasadzie hipersześcianu zyskują coraz większą popularność (patrz tablica 2.1 w rozdziale 2.4).

Systemy z przesyłem komunikatów zaczynają dominować wśród architektur równoległych, ze względu na możliwość zastosowania tanich elementów (mikroprocesory) jako pojedynczych elementów przetwarzających. Przyśpieszenie obliczeń uzyskuje się drogą użycia bardzo dużej liczby takich procesorów (sięgającej setki lub nawet tysięcy), mimo że pojedyncze procesory reprezentują małą moc obliczeniową. Technika ta nosi nazwę paralelizmu masywnego. Należy jednak podkreślić, że mechanizm synchronizacji i komunikacji oparty na przesyle komunikatów, obarczony jest istotnym narzutem czasowym (większym niż przy synchronizacji komunikacji przez wspólną pamięć), stąd jest on efektywny



Rys. 2.9. Typowe rozwiązania sieci połączeń w systemach z pamięcią rozproszoną
Fig. 2.9. Typical interconnection networks in systems with distributed memory

tylko w przypadku stosunkowo rzadkich powiązań między realizowanymi procesami (strumieniami rozkazów, patrz rozdział 2.1, punkt c.), co ogranicza użyteczność systemów z paralelizmem masywnym w stosunku do wielu algorytmów.

Podobnie jak w przypadku systemów SIMD, szybkość pracy systemu wieloprocesorowego (teoretycznie równa iloczynowi liczby procesorów p przez szybkość jednego procesora) ograniczona jest przez takie czynniki, jak [6]:

- stopień współbieżności algorytmu, czyli liczba procesów, które mogą być równocześnie wykonywane; jeśli jest on niższy niż p, wówczas nie wszystkie procesory pracują; w praktyce tylko w niewielu algorytmach utrzymany jest wysoki stopień współbieżności przez cały czas realizacji algorytmu,
- konflikty w dostępie do pamięci (w odniesieniu do systemów ze wspólną pamięcią),
- narzut czasowy spowodowany synchronizacją procesów (dokładniejsze omówienie znajduje się w rozdziale 6.2).

Przkładami systemów wieloprocesorowych są maszyny IBM 3033, 3081, Univac 1100, Tandem-16, C.mmp, Cm*, HEP (patrz też tablica 2.1 w rozdziale 2.4).

2.4. Praktyczne implementacje architektur równoległch

Przedstawione w rozdziałach 2.2 i 2.3 architektury równoległe znalazły w ostatnich latach szerokie zastosowanie w konstrukcji systemów przewidzianych do intensywnych obliczeń naukowo-inżynierskich. Maszyny takie noszą nazwę superkomputerów i minisuperkomputerów. Definicja superkomputera jest umowna - zazwyczaj przyjmuje się, że jest to komputer o mocy obliczeniowej ponad 100 Mflops. Minisuperkomputery dysponują zwykle mocą rzędu kilkudziesięciu Mflops i są znacznie tańsze od superkomputerów, kosztujących z reguły ponad 5 mln dolarów. Obie klasy maszyn wykorzystuje się do obliczeń inżynierskich (analiza strukturalna, aerodynamika, systemy wizyjne, CAD, robotyka), badań sejsmicznych, modelowania złóż ropy naftowej i reaktorów atomowych, w medycynie (tomografia, inżynieria genetyczna), modelowaniu prognostycznym (meteorologia, oceanografia, astrofizyka, ekonomia) oraz do badań podstawowych (mechanika kwantowa, chemia polimerów, krystalografia, dynamika płynów, kwantowa teoria pola, dynamika molekularna, projektowanie układów VLSI). Dużą część stanowią zastosowania militarne.

Tradycyjne, najbardziej znane superkomputery produkowane seryjnie od połowy lat 70., oparte są na procesorze wektorowym. Najstarsze konstrukcje zawierały tylko jeden procesor; nowsze posiadają ich zazwyczaj kilka do kilkunastu, łączą więc w sobie przetwarzanie wektorowe z wieloprocesorowym. Obecnie produkowane (mini)superkomputery są coraz częściej implementacją paralelizmu masywnego (patrz rozdział 2.3). Wiele prognoz przewiduje, że przyszłość superkomputerów leży właśnie w paralelizmie masywnym, ze stopniowym odchodzeniem od konstrukcji wektorowych.

W tablicy 2.1 zaprezentowano najbardziej znane superkomputery i minisuperkomputery.

Obok (mini)superkomputerów, architektury równoległe wkraczają również do rodziny systemów mniejszych (minikomputery, stacje graficzne, systemy wizyjne, komputery osobiste), zazwyczaj w formie dołączanych układów (karty akceleratorów obliczeń). W ostatnim czasie coraz większego znaczenia nabierają

Tablica 2.1

Superkomputery i minisuperkomputery (gwiazdki przy liczbie procesorów oznaczają architektury typu "hipersześcian")

Komputer	Producent	Liczba procesorów	Max szybkość (w Mflops)	System operacyjny
CRAY-1	Cray Research	1 wektorowy	160	COS
CRAY X-MP	Cray Research	1,2,4 wekt.	210/proc.	COS,Unix
CRAY-2	Cray Research	4 wektorowe	2000	Unix
CRAY Y-MP	Cray Research	8 wektorowych	2666	COS,Unix
Cyber 205	Control Data	1 wektorowy	400	VSOS
ETA[10]	ETA Systems	2,4,6,8 wek.	800/proc.	VSOS,Unix
FACOM VP-400	Fujitsu	1 wektorowy	1142	MVS
S-810/20	Hitachi	1 wektorowy	630	MVS
SX-2	NEC	1 wektorowy	1333	ATSS-AF
FX/8	Alliant Comp.	8 wektorowych	94	Unix
C-1	Convex Comput.	1 wektorowy	60	Unix
SCS-40	Scient. Comp.	1 wektorowy	44	COS
Ametek 14	Symult Systems	16 - 256*	15	Unix
Ametek 2010	Symult Systems	4 - 512*	215	React.Kern
Butterfly PP	BBN Adv.Comp.	2 - 256	brak danych	Chrysalis
Connect.Mach.	Thinking Mach.	65536*	brak danych	Unix
ELXSI 6400	ELXSI	1 - 12	14	Unix
Multimax	Encore Comput.	2 - 20	brak danych	Unix
Flex/32	Flexible Proc.	2 - 20	brak danych	MMOS,Unix
FPS Series T	Floating Point	16 - 16384*	16/proc.	Occam,Unix
i PSC/2	Intel	32 - 128*	1000	Unix
MPP	Goodyear	16384	470	brak
NCube/10	NCube	4 - 1024*	500	Unix
Sequent	Sequent Comp.	2 - 30	brak danych	Unix

układy (karty) transputerów, będące jedną z form systemów wieloprocesorowych z pamięcią rozproszoną.

Bardziej szczegółowe niż w niniejszym artykule informacje na temat przetwarzania i komputerów równoległych znaleźć można m.in. w [2;10;11;12;13;17;18;22;26;27;28].

3. METODY CHARAKTERYZACJI WYDAJNOŚCIOWEJ

Zadanie charakteryzacji efektywności funkcjonowania komputera, zwłaszcza równoległego, za pomocą kilku tylko parametrów, nie należy do prostych, efektywność ta jest bowiem funkcją zarówno architektury komputera (włączając technologię wykonania), jak i konkretnych zastosowań programowych.

Najprostsza charakterystyka wydajnościowa komputera równoległego, to charakterystyka jednoparametrowa [25]. Producenci komputerów równoległych, zwłaszcza superkomuterów, reklamują je podając maksymalną szybkość przetwarzania (ang. peak performance) r_∞. Jako miara "czystej" mocy obliczeniowej, parametr ten jest często cytowany w licznych dyskusjach efektywności komputerów. Jest to, w przypadku procesora wektorowego, asymptotyczna szybkość osiągana przez potokową jednostkę funkcjonalną przy założeniu nieskończenie długich wektorów danych. Dla systemu wieloprocesorowego maksymalna szybkość przetwarzania równa jest iloczynowi liczby procesorów i maksymalnej szybkości dla jednego procesora. Należy od razu podkreślić, że jest to szybkość teoretyczna, praktycznie osiągana jedynie w warunkach absolutnie optymalnych przez bardzo krótki okres. Stąd parametr ten jest użyteczny jedynie przy uwzględniającym naturę samego programu szacowaniu rzeczywistych szybkości.

Innym rodzajem charakterystyki jednoparametrowej jest podanie rzeczywistej szybkości przetwarzania dla konkretnego programu. Uzyskuje się ją drogą doświadczalną, przez podzielenie znanej liczby operacji zmiennoprzecinkowych w programie przez zmierzony czas działania programu. Charakterystyka taka może więc być podana tylko dla maszyny już użytkowanej, a uzyskana w wyniku powyższego procesu miara wydajności zależna jest w dużym stopniu od rodzaju realizowanego programu. Dlatego aby charakterystyka była jak najpełniejsza, przygotowuje się całe zestawy takich testowych programów (ang. benchmark), z których każdy bada efektywność procesora dla pewnych, typowych w obliczeniach naukowo-inżynierskich operacji bądź całych algorytmów (szybka transformata

Fouriera, dodawanie/mnożenie/transpozycja macierzy, rozwiązywanie układów równań liniowych, rozwiązywanie cząstkowych równań różniczkowych, metoda Monte Carlo i in.). Wśród użytkowników superkomputerów popularność zyskało kilka takich standardowych zestawów, powszechnie uważanych za najbardziej reprezentatywne (Livermore Kernels, Los Alamos Benchmarks, NAS Benchmarks Perfect Club).

Uzyskane wyniki prezentowane są bądź w postaci tabelki zawierającej szybkości zaobserwowane dla poszczególnych programów zestawu, bądź też jako średnia z całego zestawu. W tym drugim przypadku średnia może być

a) arytmetyczna

$$E = \sum_{i=1}^{n} \frac{r_i}{n}$$

b) geometryczna

$$E = \sqrt[n]{\prod_{i=1}^{n} r_i}$$

c) harmoniczna

$$E = \frac{n}{\sum_{i=1}^{n} \frac{1}{r_i}}$$

gdzie: n - liczba testów,,

r_i - zmierzona szybkość dla testu i.

Niejako na drugim końcu zakresu charakterystyk komputera równoległego znajduje się taka, która uwzględnią dokładnie każdy aspekt jego architektury. W takim przypadku liczba parametrów może przekraczać 100 i model matematyczny tworzony przez równania wiążące ze sobą te parametry staje się praktycznie bardzo trudny do rozwiązania. Dlatego najczęściej stosuje się wówczas programową symulację, cykl po cyklu, realizacji przez komputer określonego algorytmu obliczeniowego. Podobnie jak w przypadku każdej innej symulacji, wadą jest tutaj długi czas trwania. Ponadto dla wydajnego planowania algorytmów dla maszyny równoległej, analityk musi korzystać ze znacznie prostszego mode-

lu, z możliwie małą liczbą parametrów, przy jednoczesnym dobrym uchwyceniu zależności czasowych, którymi może on łatwo manipulować (zwłaszcza minimalizować). Oczywiste jest, że dokładność opisu działania komputera w postaci modelu matematycznego proporcjonalna jest do liczby użytych w tym modelu parametrów, narzuca się więc konieczność rozsądnego kompromisu. Nawet jednak gdy parametrów będzie tylko kilka, zbudowany na nich uproszczony model może być bardzo użyteczny, gdyż pozwalać będzie na szacowanie, które algorytmy prawdopodobnie będą wydajne (szybkie) na danej maszynie. Podobnie można porównywać szacunkowe czasy działania czy szybkości przetwarzania dla wybranego algorytmu na maszynach o różnych architekturach.

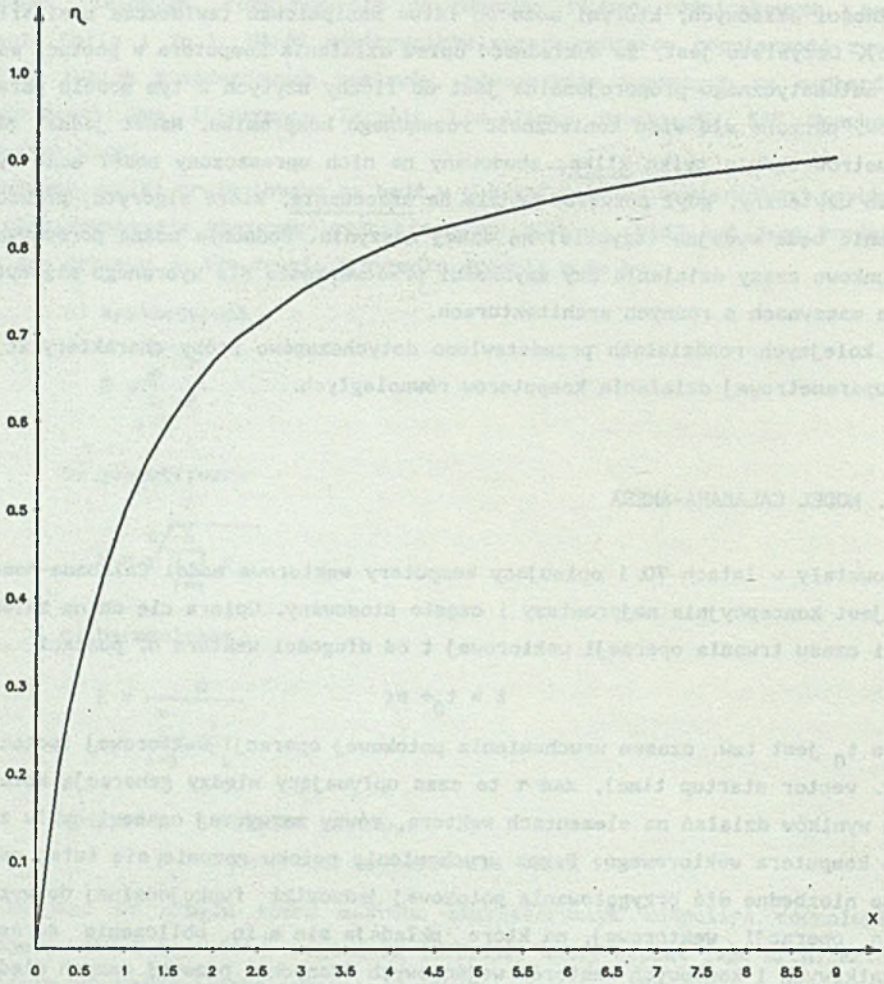
W kolejnych rozdziałach przedstawiono dotychczasowe próby charakteryzacji kilkuparametrowej działania komputerów równoległych.

4. MODEL CALAHANA-AMESA

Powstały w latach 70. i opisujący komputery wektorowe model Calahana-Amesa [4] jest koncepcyjnie najprostszy i często stosowany. Opiera się on na zależności czasu trwania operacji wektorowej t od długości wektora n, postaci

$$t = t_0 + n\tau \tag{4.1}$$

gdzie t_0 jest tzw. czasem uruchomienia potokowej operacji wektorowej (potoku) (ang. vector startup time), zaś τ to czas upływający między generacją kolejnych wyników działań na elementach wektora, równy zazwyczaj czasowi cyklu zegara komputera wektorowego. Przez uruchomienie potoku rozumie się tutaj operacje niezbędne dla przygotowania potokowej jednostki funkcjonalnej do wykonania operacji wektorowej, na które składają się m.in. obliczenie adresów początkowych i końcowych wektorów wejściowych (danych), przesył danych między pamięcią i procesorem oraz "załadowanie" jednostki funkcjonalnej, tzn. zapełnienie wszystkich jej segmentów (z wyj. ostatniego) kolejnymi parami argumentów (elementów wektorów wejściowych). Ze względu na ostatni składnik, czas uruchomienia potoku jest różny dla różnych jednostek funkcjonalnych (np. mnożenia i dodawania).



Rys. 4.1. Efektywność wykorzystania potokowej jadnostki funkcjonalnej, wg Calahana-Amesa

Fig. 4.1. Utilization of the pipelined functional unit, according to Calahan-Ames

Wychodząc z (4.1) można w prosty sposób obliczyć, dla m operacji wektorowych składających się na wybrany algorytm, średnia szybkość przetwarzania r jako

$$\frac{\sum_{i=1}^{m} n_i}{\sum_{i=1}^{m} t_{o_i} + \tau \sum_{i=1}^{m} n_i} = \frac{\frac{1}{\tau}}{1 + \frac{1}{\tau}\frac{\bar{t}_o}{\bar{n}}} \tag{4.2}$$

gdzie $\bar{t}_o$ - jest średnim czasem uruchomienia potoku (parametr procesora), a $\bar{n}$ - średnią długością wektora (parametr algorytmu). Mnożąc (4.2) obustronnie przez τ, otrzymujemy znormalizowaną (w skali od 0 do 1) efektywność wykorzystania potokowej jednostki funkcjonalnej $\eta = r_\tau$

$$\eta = \frac{1}{1 + x^{-1}}$$

gdzie $x = \frac{n\tau}{\bar{t}_o}$. Wykres η w funkcji x przedstawia rys. 4.1.

Jak łatwo zauważyć, przy $x \longrightarrow \infty$, $\eta \longrightarrow 1$. Przy stałych τ i $\bar{t}_o$ oznacza to, że dla operacji na nieskończenie długich wektorach danych, asymptotyczna szybkość przetwarzania r_∞ równa jest odwrotności czasu cyklu zegara

$$r_\infty \equiv \lim_{n \to \infty} r(n) = 1/\tau$$

Interesujący jest również przypadek x=1. Wówczas $\eta = 1/2$, $r = r_\infty/2$; połowę czasu obliczeń zajmują w tym wypadku czynności przygotowawcze do obliczeń (uruchomienie potoku), połowę zaś - właściwe obliczenia.

5. MODEL BUCHER-SIMMONS

Bucher i Simmons [3] stworzyły model dla procesorów wektorowych typu "rejestr-rejestr" (patrz rozdział 2.2). Analizując przedstawiony w rozdziale 4 model Calahana-Amesa pod kątem przydatności dla konkretnych architektur wektorowych, łatwo zauważyć, że jego czysto liniowa charakterystyka, wynikająca z przyjęcia za podstawę liniowej zależności (4.1), będąc stosunkowo dobrą aproksymacją dla komputerów "pamięć-pamięć" (np. Cyber 205, ETA[10]), niezbyt

precyzyjnie opisuje funkcjonowanie komputerów "rejestr-rejestr", stanowiących większość wśród produkowanych obecnie superkomputerów wektorowych (maszyny firmy Cray, Fujitsu, Hitachi, NEC i większość minisuperkomputerów). Różnice w charakterystykach obu typów procesorów przedstawiono na rys. 5.1; jako przykłady posłużyły superkomputery CRAY-1 i Cyber 205.

Schodkowy charakter wykresu dla procesora CRAY-1 wynika z określonej długości rejestrów wektorowych (dla CRAY-1 równej 64), co powoduje, że dłuższe wektory muszą być dzielone na "kawałki" o długości równej rozmiarowi rejestru wektorowego, a operacja wektorowa składa się z ciągu operacji wykonywanych na takich kawałkach. Oczywista w takim przypadku zwłoka czasowa przy przejściu do nowego kawałka (na wykresie 5.1 "schodek" od wielokrotności 64 do wielokrotności 64 + 1) związana jest przede wszystkim z koniecznością całkowitego opróżnienia rejestru danych, tzn. przesłania wszystkich argumentów (elementów kawałka wektora) do jednostki funkcjonalnej, zanim można będzie załadować nowy kawałek. Powyższy proces Bucher i Simmons modelują jako

$$t = t_L + \left\lceil \frac{n}{l_r} \right\rceil t_s + n\tau \qquad (5.1)$$

gdzie t - czas trwania całej operacji wektorowej na wektorze o długości n,

l_r - długość rejestru wektorowego,

t_L - czas trwania inicjalizacji pętli programowej, sterującej operacją podziału wektora na kawałki,

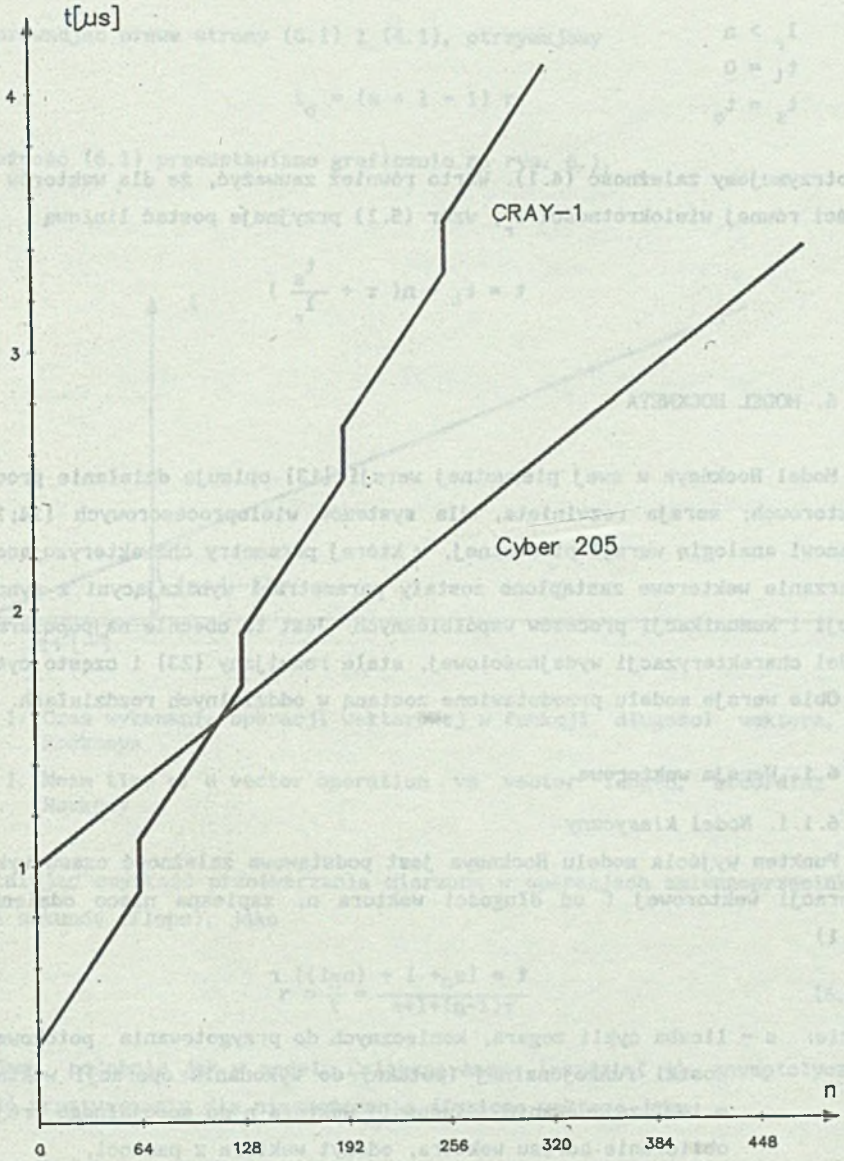
t_s - zwłoka czasowa związana z ładowaniem nowego kawałka,

τ - czas cyklu zegara.

Wykorzystując zależność (5.1) do obliczenia asymptotycznej szybkości przetwarzania r_∞ dla bardzo długich wektorów, otrzymujemy:

$$r_\infty \equiv \lim_{n \to \infty} \frac{n}{t} = \frac{1}{\tau + \frac{s}{l_r}} < \frac{1}{\tau}$$

Widać stąd, że podział wektora na kawałki powoduje zmniejszenie rzeczywistej maksymalnej szybkości przetwarzania w stosunku do teoretycznej szybkości maksymalnej $1/\tau$. Za pomocą wzoru (5.1) można również modelować procesory typu "pamięć-pamięć". Wówczas



Rys. 5.1. Średni czas przetwarzania wektora w funkcji jego długości
Fig. 5.1. Mean time for processing a vector vs its length

$l_r > n$

$t_L = 0$

$t_s = t_o$

i otrzymujemy zależność (4.1). Warto również zauważyć, że dla wektorów o długości równej wielokrotności l_r, wzór (5.1) przyjmuje postać liniową

$$t = t_L + n(\tau + \frac{t_s}{l_r})$$

6. MODEL HOCKNEYA

Model Hockneya w swej pierwotnej wersji [13] opisuje działanie procesorów wektorowch; wersja rozwinięta, dla systemów wieloprocesorowych [14;15;16], stanowi analogię wersji pierwotnej, w której parametry charakteryzujące przetwarzanie wektorowe zastąpione zostały parametrami wynikającymi z synchronizacji i komunikacji procesów współbieżnych. Jest to obecnie najpopularniejszy model charakteryzacji wydajnościowej, stale rozwijany [23] i często cytowany. Obie wersje modelu przedstawione zostaną w oddzielnych rozdziałach.

6.1. Wersja wektorowa

6.1.1. *Model klasyczny*

Punktem wyjścia modelu Hockneya jest podstawowa zależność czasu wykonania operacji wektorowej t od długości wektora n, zapisana nieco odmiennie od (4.1)

$$t = [s + l + (n-1)]\ r \qquad (6.1)$$

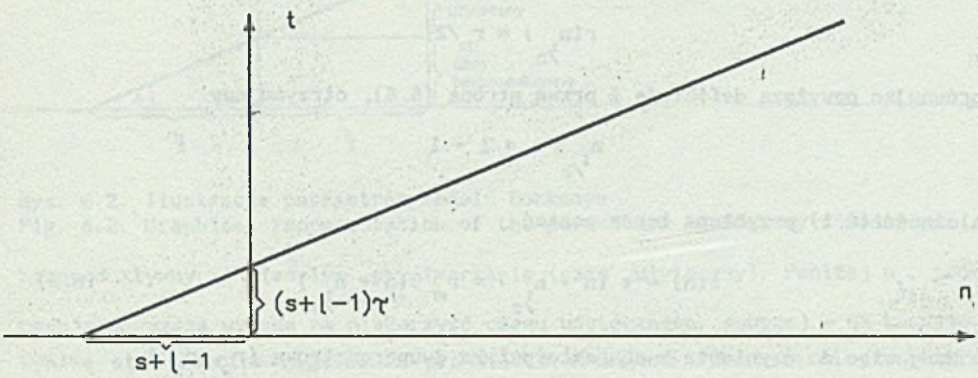
gdzie: s - liczba cykli zegara, koniecznych do przygotowania potokowej jednostki funkcjonalnej (potoku) do wykonania operacji wektorowej, m.in. załadowanie długości wektora n do specjalnego rejestru, obliczenie adresu wektora, odczyt wektora z pamięci,

l - liczba cykli, po której na wyjściu potoku pojawia się pierwszy wynik, tzn. wynik działania na pierwszym elemencie wektora, równy liczbie etapów w potoku (długość potoku).

Porównując prawe strony (6.1) i (4.1), otrzymujemy

$$t_0 = (s + l - 1)\, r$$

Zależność (6.1) przedstawiono graficznie na rys. 6.1.



Rys. 6.1. Czas wykonania operacji wektorowej w funkcji długości wektora, wg Hockneya

Fig. 6.1. Mean time of a vector operation vs vector length, according to Hockney

Definiując szybkość przetwarzania mierzoną w operacjach zmiennoprzecinkowych na sekundę (flops), jako

$$r \equiv \frac{n}{t} = \frac{n}{s+l+(n-1)\tau} \tag{6.2}$$

uzyskujemy, podobnie jak w modelu Calahana-Amesa (rozdział 4), asymptotyczną szybkość przetwarzania dla nieskończenie długiego wektora jako

$$r_\infty \equiv \lim_{n\to\infty} r(n) = 1/\tau \tag{6.3}$$

Możemy więc zapisać r(n) jako

$$r(n) = \frac{n}{s+l+(n-1)} r_\infty \tag{6.4}$$

Hockney wprowadza nowy parametr, $n_{1/2}$, definiowany jako

$n_{1/2}$ - długość połowicznej wydajności; jest to długość wektora, dla której osiąga się szybkość przetwarzania równą połowie szybkości maksymanej (asymptotycznej):

$$r(n_{1/2}) = r_\infty/2$$

Porównując powyższą definicję z prawą stroną (6.4), otrzymujemy

$$n_{1/2} = s + l - 1$$

Zależność (6.1) przybiera teraz postać

$$t(n) = \tau\,(n + n_{1/2}) = r_\infty^{-1}(n + n_{1/2}) \tag{6.5}$$

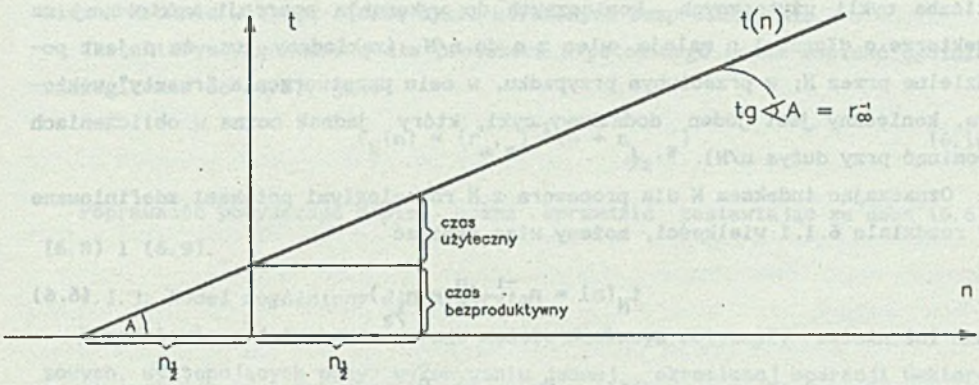
Mamy więc do czynienia z charakterystyką dwuparametrową (r_∞, $n_{1/2}$). Powtórzmy jeszcze raz znaczenie obu parametrów:

r_∞ - asymptotyczna (maksymalna) szybkość przetwarzania, mierzona liczbą operacji zmiennoprzecinkowych na parach liczb na sekundę (flops),

$n_{1/2}$ - długość połowicznej wydajności (szybkości), czyli długość wektora, dla której wydajność (szybkość) potoku równa jest połowie wydajności (szybkości) maksymalnej.

Graficzną reprezentację obu parametrów przedstawiono na rys. 6.2 [23].

r_∞ stanowi odwrotność tangensa nachylenia charakterystyki czasowej do osi długości wektora n. $n_{1/2}$ - to "ujemną" długość wektora, odciętego na osi n przez charakterystykę czasową. $n_{1/2}$ można więc potraktować jako długość wyimaginowanego wektora, który należy przetworzyć w jednostce potokowej zanim zacznie ona dostarczać właściwe wyniki. Odkładając $n_{1/2}$, jak na rysunku, na dodatniej półosi n, można również interpretować ten parametr jako długość wektora, którego czas przetwarzania rozkłada się dokładnie w połowie na przygotowanie do właściwego przetwarzania (tzn. dostarczania przez potok jednostkowego wyniku co cykl), stanowiące z punktu widzenia efektywności obliczeń czas



Rys. 6.2. Ilustracja parametrów modelu Hockneya
Fig. 6.2. Graphical representation of the parameters in Hockney's model

bezproduktywny, i właściwe przetwarzanie (czas użyteczny). Poniżej $n_{1/2}$, proporcja powyższa wypada na niekorzyść czasu użytecznego, powyżej - na korzyść. Wynika stąd istotna reguła, że prawdziwie efektywne wykorzystanie komputera wektorowego następuje przy $n \gg n_{1/2}$. Stąd znajomość $n_{1/2}$ dla konkretnej architektury wektorowej (ściślej: dla każdego jej potoku, gdyż wartość ta może różnić się w zależności od operacji wykonywanej przez jednostkę funkcjonalną) jest podstawą konstrukcji algorytmów efektywnych dla tej architektury.

Warto jeszcze przedyskutować sens powyższych parametrów w nieco głębszym ujęciu. r_∞ stanowi odwrotność czasu cyklu zegara, jest to więc parametr czysto sprzętowy, charakteryzujący technologię użytą przy konstrukcji procesora wektorowego. Stąd nie odgrywa on żadnej roli przy wyborze algorytmów obliczeniowych. $n_{1/2}$ natomiast charakteryzuje sprzęt jako miara równoległości (wynikającej z użycia jednostki potokowej) w architekturze. Ze względu na zależność efektywności obliczeń od stosunku $n/n_{1/2}$ można uznać, że długość wektora n stanowi miarę równoległości zadania, zaś stosunek $n/n_{1/2}$ - miarę równoległości komputera w czasie wykonywania tego zadania.

6.1.2. Procesor wielopotokowy [23]

W przypadku procesorów wielopotokowych (patrz rozdział 2.2) liczba równoległych potoków N staje się w modelu Hockneya istotnym parametrem. W takim

przypadku N jednostek funkcjonalnych produkuje N wyników w jednym cyklu, liczba cykli użytecznych koniecznych do wykonania operacji wektorowej na wektorze o długości n maleje więc z n do n/N (zakładamy tu, że n jest podzielne przez N; w przeciwnym przypadku, w celu przetworzenia "reszty" wektora, konieczny jest jeden dodatkowy cykl, który jednak można w obliczeniach pominąć przy dużym n/N).

Oznaczając indeksem N dla procesora z N równoległymi potokami zdefiniowane w rozdziale 6.1.1 wielkości, możemy więc zapisać

$$t_N(n) = r_\infty^{-1} \left(\frac{n}{N} + n_{1/2}\right) \tag{6.6}$$

$$r_N(n) = \frac{n}{t_N(n)} = \frac{n}{\frac{n}{N} + n_{1/2}} r_\infty \tag{6.7}$$

$$r_{\infty,N} = \lim_{n\to\infty} r_N(n) = Nr_\infty \tag{6.8}$$

Ostatnie równanie stanowi matematyczny zapis oczywistego faktu, że asymptotyczna szybkość przetwarzania wzrasta N-krotnie przy N-krotnym powieleniu potoków. Definiując, podobnie jak w rozdziale 6.1.1, długość połowicznej wydajności $n_{1/2,N}$, otrzymujemy

$$r_N(n_{1/2,N}) = \frac{r_{\infty,N}}{2} = N\frac{r_\infty}{2}$$

Podstawiając równocześnie $n_{1/2,N}$ do wzoru (6.7)

$$r_N(n_{1/2,N}) = \frac{n_{1/2,N}}{\frac{n_{1/2,N}}{N} + n_{1/2}} r_\infty$$

Porównując prawe strony obu powyższych równań, otrzymujemy

$$n_{1/2,N} = Nn_{1/2} \tag{6.9}$$

Jak widać, podobnie jak w przypadku asymptotycznej szybkości r_∞, długość połowicznej wydajności wzrosła N-krotnie w porównaniu z procesorem jednopoto-

kowym, co nie jest, w przeciwieństwie do wzrostu r_∞, korzystne, oznacza bowiem N-krotnie większą liczbę cykli straconych bezproduktywnie.

Charakterystykę czasową dla procesora N-potokowego można zapisać ogólnie, analogicznie do (6.5), jako

$$t_N(n) = (r_{\infty,N})^{-1}(n + n_{1/2,N}) \tag{6.10}$$

Poprawność powyższego zapisu można sprawdzić zestawiając ze sobą (6.6), (6.8) i (6.9).

6.1.3. Model uogólniony dla algorytmu

Przedstawione dotychczas wersje modelu Hockneya dotyczyły zależności czasowych, występujących przy wykonywaniu jednej, określonej operacji wektorowej. Model można uogólnić dla całego algorytmu obliczeniowego, będącego sekwencją operacji wektorowych. Ignoruje się w tym wypadku wszelkie operacje skalarne, co prowadzi do nieadekwatności takiego modelu dla algorytmów, w których obliczenia skalarne w istotny sposób wpływają na czas wykonania (patrz Załącznik).

Czas wykonania algorytmu T można zapisać jako

$$T = r_\infty^{-1}(s + n_{1/2}q) \tag{6.11}$$

lub

$$T = r_\infty^{-1}q(\bar{n} + n_{1/2}) \tag{6.12}$$

gdzie: s - całkowita liczba operacji elementarnych na liczbach zmiennoprzecinkowych (tzn. między parami takich liczb),

q - liczba operacji wektorowych w algorytmie,

$\bar{n}$ - średnia długość wektora przetwarzanego w algorytmie, $\bar{n}=s/q$.

W obu powyższych równaniach zakłada się, że parametr $n_{1/2}$ jest stały i niezależny od typu operacji wektorowej. W praktyce jest to dość rzadki przypadek (warunkiem koniecznym, choć niewystarczającym, jest identyczna długość potoków realizujących wszystkie q operacji). Dlatego, aby apoksymacja była lepsza, można przyjąć, że użyte w równaniach (6.11) i (6.12) $n_{1/2}$ jest wartością średnią dla wszystkich operacji wektorowych.

6.1.4. Wpływ przesyłów procesor-pamięć

Szybkość przetwarzania przez jednostkę potokową jest zazwyczaj kilkakrotnie większa od szybkości przesyłu danych między procesorem i pamięcią. Powyższe zjawisko nosi w literaturze nazwę "'wąskiego gardła' von Neumanna" lub "wąskiego gardła w dostępie (transmisji) do pamięci". Efektywna szybkość przetwarzania zależeć więc będzie od ilości obliczeń przypadających na dostęp do pamięci. Hockney [15] stworzył adekwatny model, zakładający, że czas każdego dostępu do pamięci, polegającego na odczycie/zapisie wektora z/do pamięci, spełnia zależność analogiczną do (6.5)

$$t = (n + n^m_{1/2}) / r^m_\infty$$

gdzie indeks m oznacza parametr odnoszący się do dostępu do pamięci. Możemy więc w tym momencie mówić o "potoku" transmisji, działającym w odniesieniu do przesyłu danych. Parametry potoku arytmetycznego, omówione w rozdziale 6.1.1, oznacza się w tym przypadku indeksem a, co nadaje równaniu (6.5) postać:

$$t = (n + n^a_{1/2}) / r^a_\infty$$

Przy założeniu f operacji zmiennoprzecinkowych przypadających na 1 dostęp do pamięci, można średni czas wykonania operacji wektorowej $\bar{t}$, przy uwzględnieniu przesyłów do pamięci, zapisać jako

$$\bar{t} = (n + \bar{n}_{1/2}) / \bar{r}_\infty$$

gdzie:

$$\bar{r}_\infty = \frac{r^a_\infty}{x^{-1}}, \quad x = \frac{f}{f_{1/2}}, \quad f_{1/2} = \frac{r^a_\infty}{r^m_\infty}$$

$$\bar{n}_{1/2} = \frac{n^m_{1/2} + n^a_{1/2}\, x}{1 + x}$$

Parametr $f_{1/2}$ wprowadzony tu został przez analogię do $n_{1/2}$; oznacza on wartość f wymaganą dla osiągnięcia przez $\bar{r}_\infty$ (asymptotyczna szybkość przetwarzania z uwzględnieniem dostępów do pamięci) połowy wartości r^a_∞ (asymptotyczna

szybkość przetwarzania przez samą jednostkę potokową, bez uwzględniania dostępów do pamięci). $f_{1/2}$ jest parametrem sprzętowym; porównanie f (charakteryzującego algorytm) z $f_{1/2}$ jest pomocne przy szacowaniu "rzeczywistej asymptotycznej" szybkości przetwarzania $\bar{r}_{\infty}$. Można zauważyć, że przebiegi funkcji $r^{a}(n)$ (wzór 6.4) i $\bar{r}_{\infty}(f)$ są identyczne.

6.2 Wersja wieloprocesorowa

Rozwinięciem modelu Hockneya dla architektur typu MIMD jest (również dwuparametrowa), wersja wieloprocesorowa [15,16]. Traktuje ona program równoległy (tzn. program, którego fragmenty wykonywane są współbieżnie w oddzielnych procesorach) jako sekwencję tzw. segmentów pracy równoległej. W każdym z takich segmentów wykonywane obliczenia (praca) dzielone są między szereg logicznie niezależnych strumieni rozkazów; każdy z tych strumieni wykonywany jest przez oddzielny procesor. Przez logiczną niezależność strumieni rozumie się tutaj brak potrzeby komunikacji między nimi. Istotą segmentu pracy równoległej jest synchronizacja obliczeń w momentach jego rozpoczęcia i zakończenia. Wszystkie obliczenia (strumienie rozkazów) w segmencie muszą zostać zakończone, zanim można będzie rozpocząć nowy segment. Oznacza to, że w ramach segmentu, jeśli długości strumieni rozkazów będą różne, to strumienie dłuższe, które zdążyły się już wykonać, będą "czekać" na krótsze, które się jeszcze wykonują. Synchronizacja taka wiąże się z istotnym nakładem czasowym, na który składają się:

1) inicjacja każdego ze strumieni,
2) badanie, czy wszystkie strumienie się już zakończyły.

Powyższe czynności wykonywane są przez nadzorczy program sterujący. Wynikający stąd dodatkowy nakład czasowy w przeliczeniu na (użyteczne) operacje zmiennoprzecinkowe, jakie mogłyby w tym czasie być wykonane, symbolizowany jest przez parametr $s_{1/2}$ (patrz rozdział 4.1.1); różnica polega jedynie na rodzaju równoległości, jakiej parametr dotyczy. Wzorowana na (6.5) zależność czasowa dla i-tego segmentu pracy równoległej ma postać:

$$t = (r_{\infty,i})^{-1}(s_i + s_{1/2,i}) \qquad (6.13)$$

gdzie: s_i - całkowita (uwzględniając wszystkie strumienie rozkazów) liczba operacji zmiennoprzecinkowych wykonywanych na parach liczb w i-tym segmencie,

$r_{\infty,i}$ - asymptotyczna (maksymalna możliwa) szybkość przetwarzania; uwzględniona jest już tutaj wieloprocesorowość, tzn. r_∞ jest iloczynem asymptotycznej szybkości przetwarzania w jednym procesorze przez liczbę procesorów, aktywnych w i-tym segmencie,

$s_{1/2,i}$ - koszt synchronizacji, w przeliczeniu na operacje zmiennoprzecinkowe.

Wartość $s_{1/2}$ zależna jest przede wszystkim od metody synchronizacji i podobnie jak r_∞, od liczby strumieni rozkazów (równej liczbie aktywnych procesorów) w segmencie.

W równaniu (6.13) zakłada się domyślnie, że wszystkie strumienie rozkazów mają równą długość, tzn. obliczenia zostały w równym stopniu rozdzielone między procesory. Sytuację taką określamy mianem <u>równowagi obciążenia</u>. Jeśli brak takiej równowagi, wówczas konieczne jest wprowadzenie dodatkowego parametru, którym jest efektywność wykorzystania procesorów E_i, zdefiniowania w [19]. $E_i=1$ przy równowadze obciążenia, $0<E_i<1$ przy jej braku, przy czym im nierównowaga większa, tym E_i mniejsze. Równanie (6.13) przybiera teraz postać:

$$t = (r_{\infty,i})^{-1} (s_i/E_i + s_{1/2,i}) \qquad (6.14)$$

Podobnie jak w rozdziale 6.1.3, można powyższy model rozszerzyć na cały algorytm (program). Odbywa się to w sposób naturalny przez zsumowanie prawych stron równań (6.14) dla wszystkich segmentów. Przyjmując w miejsce $r_{\infty,i}$ i $s_{1/2,i}$ odpowiednio wartości średnie dla wszystkich segmentów, r_∞ i $s_{1/2}$, otrzymujemy

$$T = r_\infty^{-1} (s/E + qs_{1/2}) \qquad (6.15)$$

gdzie: q - liczba segmentów pracy równoległej,

s - całkowita ilość obliczeń w programie (całkowita liczba operacji zmiennoprzecinkowych, wykonywanych na parach liczb)

$$s = \sum_{i=1}^{q} s_i,$$

E - średnia efektywność wykorzystania procesorów w całym programie

$$E = \frac{s}{\sum_{i=1}^{q} \frac{s_i}{E_i}}$$

Występujące w równaniu (6.15) wielkości można podzielić na 2 grupy. Pierwsza, (r_∞, $s_{1/2}$), charakteryzuje środowisko programowe, na które składają się sprzęt i oprogramowanie systemowe (włączając kompilatory). Druga, (s, q, E), opisuje sam algorytm. Wracając do parametru $s_{1/2}$, poprzez porównanie z całkowitą ilością pracy s wskazuje on, czy w danym algorytmie warto dzielić tę pracę między równoległe strumienie rozkazów. Zgodnie z jego definicją, szybkość przetwarzania większą od połowy szybkości maksymalnej można osiągnąć tylko wówczas, gdy $s>s_{1/2}$. Oszacowanie, jak się ma szybkość przetwarzania w stosunku do szybkości połowicznej, stanowi niewątpliwie pewną miarę opłacalności wieloprocesorowości w konkretnym przypadku.

W istniejących systemach równoległych, $s_{1/2}$ osiąga zazwyczaj duże wartości, rzędu setek lub tysięcy operacji zmiennoprzecinkowych. Stąd ważne jest, aby program równoległy składał się raczej z niedużej liczby współbieżnych procedur o znacznej długości, niż odwrotnie, gdyż w pierwszym przypadku $s_{1/2}$ jest małe w stosunku do s, co jest korzystne.

7. UWAGI KOŃCOWE

W artykule przedstawiono przegląd architektur równoległych oraz typowe modele analityczne dwu klas tych maszyn: procesorów potokowych (wektorowych) oraz systemów wieloprocesorowych. Dokładniej omówiono model Hockneya, jako obecnie najpopularniejszy i koncepcyjnie spójny dla obu klas. Stanowi on podstawę dalszych badań [23,24], mających na celu lepsze uchwycenie reguł funkcjonowania specyficznych wersji procesorów potokowych. Opierając się na mode-

lu Hockneya można dojść do istotnych wniosków dotyczących efektywności funkcjonowania systemów wieloprocesorowch. Okazuje się bowiem, że optymalny program równoległy nie powinien być "zbyt równoległy", tzn. nie powinien całkowicie zatracić sekwencyjnego charakteru procedur, te ostatnie zaś powinny być odpowiednio długie. Stanowi to interesujący kontrast z najnowszymi osiągnięciami konstrukcyjnymi w dziedzinie systemów wielomikroprocesorowych, w których liczba procesorów przewyższa nieraz tysiąc. Pojawia się pytanie: czy warto do takiego stopnia mnożyć procesory, skoro nigdy nie będą one efektywnie wykorzystane? Przyszłość odpowie i na to, i na inne pytania rodzące się w trakcie nieuchronnego przechodzenia od obliczeń sekwencyjnych do równoległych. Modele analityczne pomagają niewątpliwie odpowiedzi na te pytania przewidzieć już teraz.

LITERATURA

1. G.Amdahl: The Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities, Proc. AFIPS, Vol.30, 1967.
2. J.Bond: Parallel-Processing Concepts Finally Come Together in Real Systems, Computer Design, June 1, 1987.
3. J.Y.Bucher, M.L.Simmons: A Close Look at Vector Performance of Register-to-Register Vector Computers and a New Model, Proc. SIGMETRICS' 87, w: Performance Evaluation Review, Vol.16, No.1, 1988.
4. D.A.Calahan, W.G.Ames: Vector Processors: Models and Applications, IEEE Transactions on Circuits and Systems, Vol.CAS-26, No.9, 1979.
5. T. Czachórski, M. Kowalówka, Z. Szczerbiński, A. Wilk: Analityczne metody rozwiązywania probabilistycznych modeli systemów komputerowych. Pakiet programowy AMOK, Pol.Śl. Gliwice 1987, skr.1383.
6. M.D.Ercegovac, T.Lang: General Approaches for Achieving High Speed Computations, w: Supercomputers: Class VI Systems, Hardware and Software. Praca zbiorowa pod redakcją S.Fernbacha, North Holland 1986.
7. M.D.Ercegovac, T.Lang: Vector Processing, w: Supercomputer: Class VI Systems, Hardware and Software. Praca zbiorowa pod redakcją S.Fernbacha, North Holland 1986.

8. D.Ferrari: Computer Systems Performance Evaluation, Prentice-Hall, 1978.
9. M.J.Flynn: Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, Vol.C-21, No.9, 1972
10. G.C.Fox, P.C.Messina: Advanced Computer Architectures, Scientific American, Oct.1987.
11. J.Gallant: Parallel Processing Ushers In a Revolution in Computing, EDN, Sep.1, 1988.
12. High Speed Computation, pod redakcją J.S.Kowalika, Springer-Verlag, 1984.
13. R.W.Hockney, C.R.Jesshope: Parallel Computers, Adam Hilger Ltd, Bristol 1981.
14. R.W.Hockney: Performance of Parallel Computers, w [12].
15. R.W.Hockney: $(r_\infty, n_{1/2}, s_{1/2})$ Measurements on the 2-CPU CRAY X-MP, Parallel Computing, No.2, 1985.
16. R.W.Hockney, Parallel Computers: Architecture and Performance, w:Parallel Computing 85 pod redakcją M.Feilmeiera, G.Jouberta i U.Schendela, North Holland, 1986.
17. K.Hwang, F.A.Briggs: Computer Architecture and Parallel Processing, McGraw-Hill, 1984.
18. W.Iszkowski: Superkomputer CRAY, Informatyka nr 5, 1985.
19. D.J.Kuck: Computers and Computation, Wiley, 1978.
20. E.D. Lazowska, J.Zahorjan, G.S. Graham, K.C. Sevcik: Quantitative System Performance, Prentice-Hall, 1984.
21. J.L.Martin, D.Mueller-Wichards: Supercomputer Performance Evaluation: Status and Directions, IBM Research Report RC 11893, 1986.
22. Multiprocessors and Parallel Processing, COMTRE Corporation, pod redakcją P.H.Enslowa jr, Wiley, 1974 (przekład polski: Systemy cyfrowe wieloprocesorowe, WNT, Warszawa 1978).
23. W.Schoenauer: Scientific Computing on Vector Computers, North Holland, 1987.
24. M.Shimasaki: Performance Analysis of Vector Supercomputers by Hockney's Model, Data Processing Center, Kyoto University, Kyoto 606, 1987.
25. J.E.Smith: Characterizing Computer Performance with a Single Number, Communications of the ACM, Vol.31, No.10, 1988.
26. Z.Szczerbiński: Cray X-MP, Informatyka, nr 6, 1989.

27. Z.Szczerbiński: Cray-2, Informatyka, nr 7, 1989.
28. V.Zakharov: Parallelism and Array Processing, IEEE Transactions on Computers, Vol.C-33, No.1, 1984.

Załącznik

WPŁYW OBLICZEŃ SEKWENCYJNYCH NA EFEKTYWNOŚĆ WYKORZYSTANIA ARCHITEKTURY RÓWNOLEGŁEJ

Już w 1967 r. Amdahl [1] ustalił analityczną zależność korzyści uzyskiwanej z zastosowania architektury równoległej od stopnia równoległości (paralelizmu) w wykonywanym programie. Nosi ona odtąd nazwę prawa Amdahla. Prawo Amdahla jest uniwersalne; dotyczy ono zarówno architektur potokowych (wektorowych), jak i wieloprocesorowych czy innych równoległych, opiera się bowiem na ogólnym pojęciu paralelizmu architektury, bez wnikania w jego szczególny charakter.

Załóżmy, że p oznacza teoretyczne (idealne) przyśpieszenie obliczeń wynikające z paralelizmu architektury (w procesorze wektorowym będzie to więc stosunek szybkości wektorowej do skalarnej, w systemie wieloprocesorowym - liczba procesorów). Oznaczając przez f_s tę część (ułamek) obliczeń w programie, która musi być wykonana w trybie sekwencyjnym (w procesorze wektorowym - w trybie skalarnym, w systemie wieloprocesorowym - w trybie jednoprocesorowym), przez T - czas działania programu (całego, w trybie sekwencyjnym, można zapisać przyśpieszenie działania programu wynikające z paralelizmu, s_p, jako

$$s_p = \frac{T}{f_s T + \frac{1-f_s}{p} T} = \frac{1}{f_s + \frac{1-f_s}{p}}$$

Graficzną reprezentację prawa Amdahla stanowią rysunki Z.1 (s_p w funkcji $f_p = 1-f_s$, dla p=8) oraz Z.2 (s_p w funkcji p).

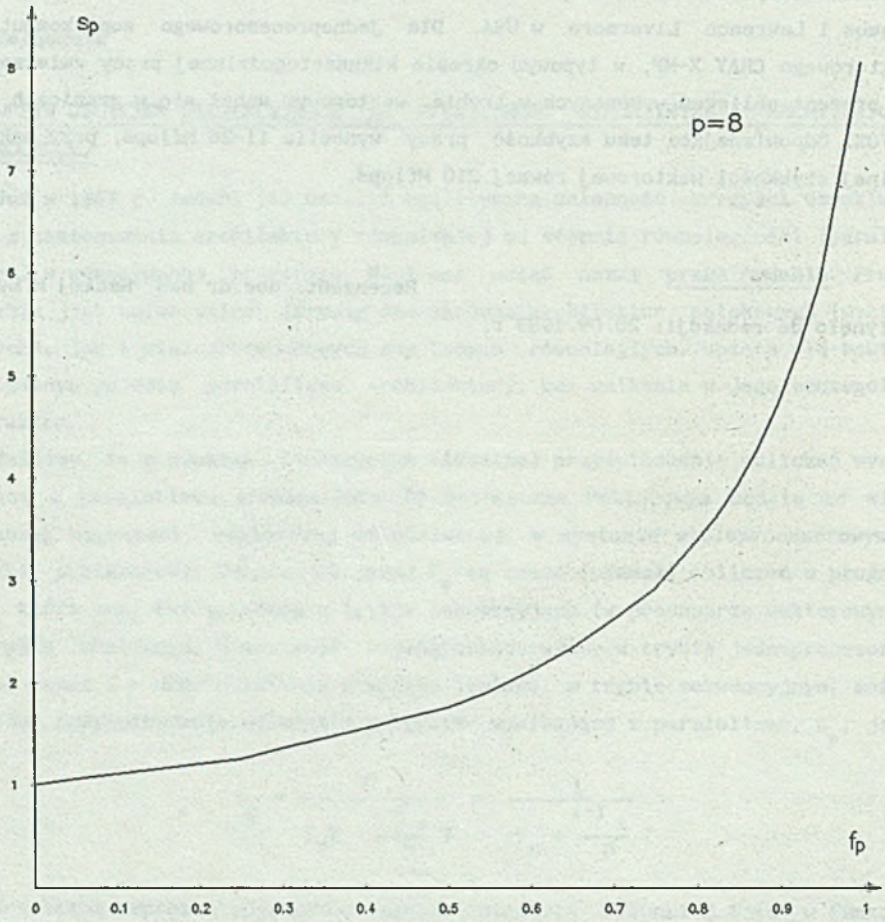
Wnioski płynące z prawa Amdahla są następujące:

1. Maksymalne (r->∞) przyśpieszenie działania programu przy wykorzystaniu obliczeń równoległych wynosi tylko $1/f_s$.

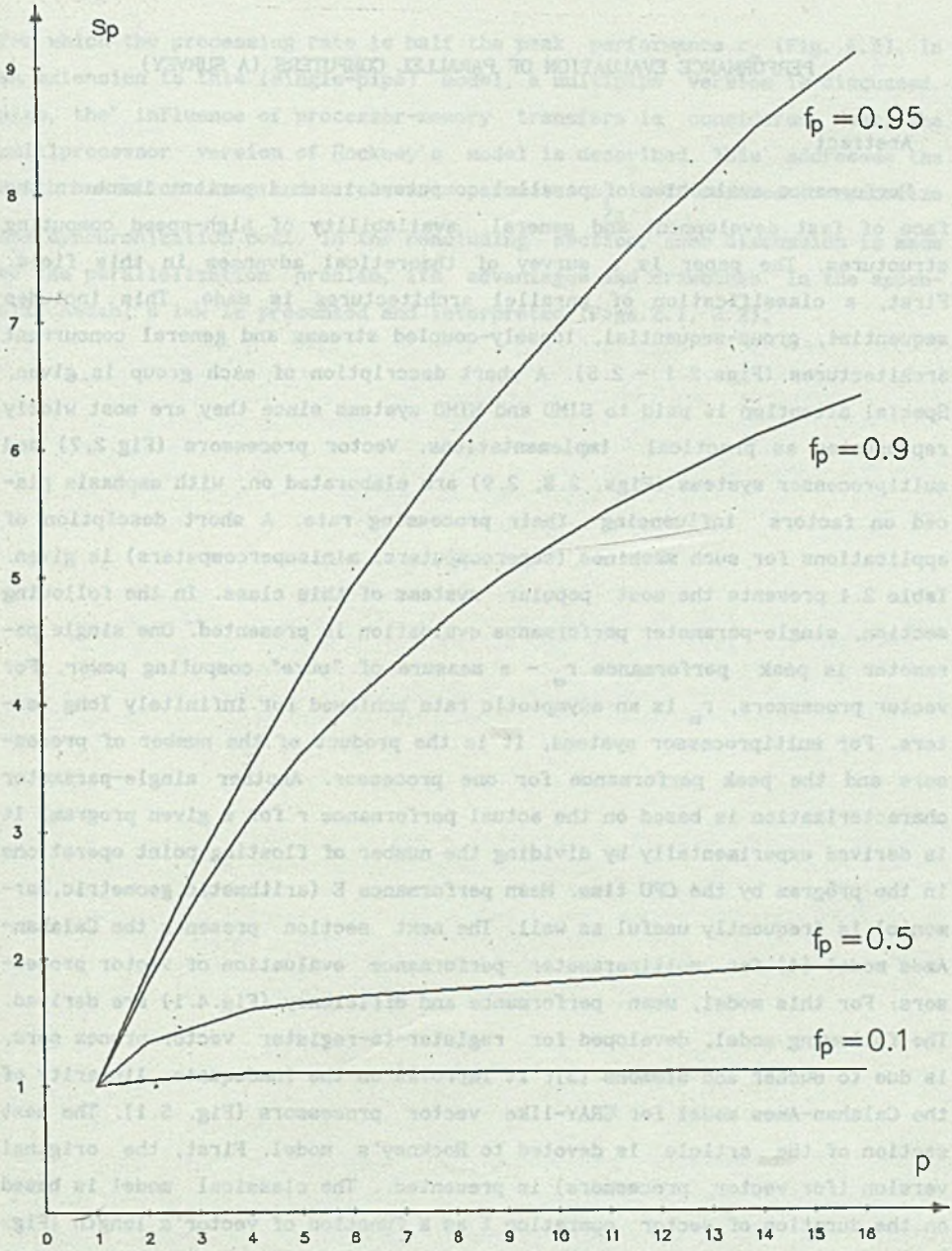
2. Aby efektywność wykorzystania architektury równoległej była istotnie wyraźna, procent kodu programu wykonywany w trybie równoległym musi być bardzo wysoki. W przeciwnym razie, funkcjonowanie systemu jest, pod względem analizy efektywności, zdominowane przez obliczenia sekwencyjne. Warto w tym momencie zacytować za [21] wyniki testów przeprowadzonych w laboratoriach Los Alamos i Lawrence Livermore w USA. Dla jednoprocesorowego superkomputera wektorowego CRAY X-MP, w typowym okresie kikunastogodzinnej pracy zmierzono, że procent obliczeń wykonanych w trybie wektorowym wahał się w granicach 50% - 70%. Odpowiadająca temu szybkość pracy wynosiła 11-26 Mflops, przy maksymalnej szybkości wektorowej równej 210 Mflops.

Recenzent: doc.dr hab. Maciej M.Sysło

Wpłynęło do redakcji: 20.09.1989 r.



Rys. Z.1. Zależność przyspieszenia od stopnia paralelizmu w programie
Fig. Z.1. Speedup vs amount of parallelism in the program



Rys. Z.2. Zależność przyspieszenia od stopnia paralelizmu w architekturze
Fig. Z.2. Speedup vs amount of parallelism in the architecture

PERFORMANCE EVALUATION OF PARALLEL COMPUTERS (A SURVEY)

Abstract

Performance evaluation of parallel computers is an important issue in the face of fast development and general availability of high-speed computing structures. The paper is a survey of theoretical advances in this field. First, a classification of parallel architectures is made. This includes sequential, group-sequential, loosely-coupled streams and general concurrent architectures (Figs.2.1 - 2.5). A short description of each group is given. Special attention is paid to SIMD and MIMD systems since they are most widely represented as practical implementations. Vector processors (Fig.2.7) and multiprocessor systems (Figs. 2.8, 2.9) are elaborated on, with emphasis placed on factors influencing their processing rate. A short desciption of applications for such machines (supercomputers, minisupercomputers) is given. Table 2.1 presents the most popular systems of this class. In the following section, single-parameter performance evaluation is presented. One single parameter is peak performance r_∞ - a measure of "pure" computing power. For vector processors, r_∞ is an asymptotic rate achieved for infinitely long vectors. For multiprocessor systems, it is the product of the number of processors and the peak performance for one processor. Another single-parameter characterization is based on the actual performance r for a given program. It is derived experimentally by dividing the number of floating point operations in the program by the CPU time. Mean performance E (arithmetic, geometric, harmonic) is frequently useful as well. The next section presents the Calahan-Ames model [4] for multiparameter performance evaluation of vector processors. For this model, mean performance and efficiency (Fig.4.1) are derived. The following model, developed for register-to-register vector proces sors, is due to Bucher and Simmons [3]; it improves on the inadequate linearity of the Calahan-Ames model for CRAY-like vector processors (Fig. 5.1). The next section of the article is devoted to Hockney's model. First, the original version (for vector processors) is presented.. The classical model is based on the duration of vector operation t as a function of vector's length (Fig. 6.1). Hockney introduces a new parameter, $n_{1/2}$, defined as the vector length

for which the processing rate is half the peak performance r_∞ (Fig. 6.2). In an extension to this (single-pipe) model, a multipipe version is discussed. Also, the influence of processor-memory transfers is considered. Next, the multiprocessor version of Hockney's model is described. This addresses the synchronization issue and a relevant parameter $s_{1/2}$ is introduced to symbolize the synchronization cost. In the concluding section, some discussion is made of the parallelization problem, its advantages and drawbacks. In the appendix, Amdahl's law is presented and interpreted (Figs.Z.1, Z.2).