STUDIA INFORMATICA 2000
Volume 21 Number 3 (41)

Urszula GESSING, Michał KOLANO
Politechnika Śląska, Instytut Informatyki

SZEREGOWANIE ROZKAZÓW JAKO ZAGADNIENIE OPTYMALIZACJI DYSKRETNEJ

Streszczenie. We współczesnych potokowych procesorach superskalarnych łączny czas przetwarzania ciągu rozkazów ściśle zależy od ich wzajemnej kolejności. Zależność ta bierze się z faktu, iż procesor superskalarny wykorzystuje wiele współbieżnych jednostek przetwarzających, przy czym równolegle przetwarzane są wyłącznie te rozkazy, pomiędzy którymi nie wykryto współzależności. Niniejszy artykuł jest próbą sformalizowania definicji problemu szeregowania rozkazów dla hipotetycznego potokowego procesora superskalarnego o definiowanej architekturze.

THE OPTIMIZATION PROBLEM OF INSTRUCTION SCHEDULING

Summary. Total time of instruction processing strictly depends on instruction order in contemporary pipe processors. This dependence is the result of the fact that superscalar processor uses many units working parallel and only those instructions are parallel processed, between which there are no dependencies known. This article is the trial of formalising of instruction scheduling problem definition for hypothetical superscalar pipe processor with defined structure.

1. Wprowadzenie

Wartość rynkowa mikroprocesora bezsprzecznie związana jest z jego wydajnością, definiowaną jako *liczba elementarnych operacji arytmetycznych, jaką układ zdolny jest wykonać w określonym czasie.* Tak rozumiana wydajność może być zwiększana na drodze *zmniejszania czasu wykonania poszczególnych operacji* albo poprzez *zwiększenie liczby operacji przypadających na jednostkę czasu pracy* układu, lub też przy wykorzystaniu obu metod jednocześnie [1][2].

Pierwsza z wymienionych metod jest historycznie związana z koncepcją procesorów o zredukowanej liście rozkazów. Maksymalne skracanie czasów trwania poszczególnych rozkazów jest bowiem podstawowym, obok zmniejszenia listy rozkazów oraz ograniczenia liczby trybów adresacji, założeniem architektury RISC. Wykorzystując mechanizm potokowania (*podział procesu wykonywania rozkazu na wiele krótkich, niezależnych faz*) możliwe staje się zaprojektowanie układów, w których większość rozkazów posiada krótkie, jednostkowe czasy przetwarzania.

Druga z przytoczonych metod wiąże się z utylizacją tzw. *równoległości niskiego poziomu* (ang. *ILP-instruction level parallelism*). Podejście to polega na zwiększeniu liczby informacji przetwarzanych w każdym cyklu pracy procesora, głównie dzięki dystrybucji rozkazów do niezależnych jednostek przetwarzających [3]. Metoda ta stosowana jest w układach superskalarnych, zakładających równoległe wykonywanie rozkazów w zwielokrotnionych jednostkach przetwarzających, w układach VLIV (ang. *very long instruction word*) opierających się na wykorzystaniu tzw. długich słów rozkazowych oraz w rozwiązaniach klasy SIMD (ang. *single instruction stream multiple data stream*) wykorzystujących rozkazy operujące równolegle na wielu danych.

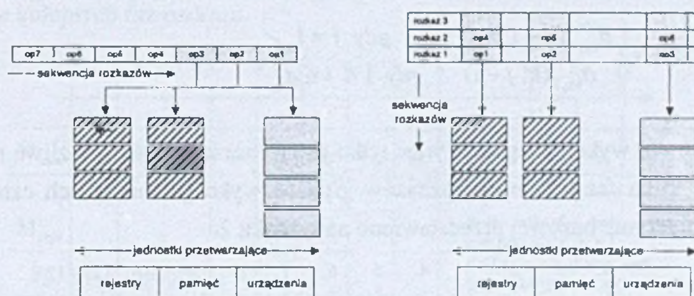
Architektura współczesnych mikroprocesorów zakłada integracje obydwu rozwiązań: kolejne rozkazy programu dystrybuowane są do niezależnych jednostek przetwarzających, w większości o organizacji potokowej. Taki schemat umożliwia utylizację równoległości programu procesora przy zachowaniu wysokiego tempa przetwarzania.

W trakcie badań wydajności architektur procesorowych odkryto, że dla pewnych uprzywilejowanych sekwencji rozkazów łączny czas przetwarzania jest krótszy niż dla pozostałych. Zjawisko to związane z utylizacją zasobów procesora i powoduje, że program zakodowany za pomocą tego samego zestawu rozkazów może posiadać różne czasy wykonania. Niniejsze opracowanie jest próbą opisania i wyjaśnienia tego zjawiska. Problem rozpatrzono dla hipotetycznego procesora o otwartej, definiowalnej architekturze.

2. Charakterystyka metod przetwarzania

Potokowe przetwarzanie superskalarne jest, jak dotąd, najbardziej zaawansowaną formą przetwarzania opartego na pojedynczym źródle instrukcji (ang. *single instruction stream*). Procesor superskalarny posiada zdolność do automatycznego wykrywania i utylizowania równoległości programu procesora, tzn. do identyfikowania rozkazów, których wykonanie, pomimo sekwencyjnego charakteru samego programu, może mieć charakter równoległy. Proces ten odbywa się w czasie rzeczywistym, bez wsparcia ze strony kompilatora.

Koncepcyjne układy superskalarne są o wiele bardziej złożone od układów klasy VLIW, w których powiązanie rozkazów z jednostką przetwarzającą ma charakter deterministyczny (rysunek 1). Dzieje się tak głównie ze względu na obecność skomplikowanych układów logicznych, gwarantujących, że pierwotna semantyka *sekwencyjnego* programu (*kolejność operacji obliczeniowych*) zostanie zachowana pomimo faktycznego *rozproszenia obliczeń*. W przypadku układów VLIW ciężar rozproszenia obliczeń spoczywa wyłącznie na kompilatorze (pewnym odejściem od tej koncepcji jest tutaj architektura EPIC).



Rys. 1. Porównanie architektury superskalarnej i architektury VLIW
Fig. 1. The comparison of superscalar and VLIW architecture

Dla dalszych rozważań niezbędne jest sformalizowanie poszczególnych metod przetwarzania. W opracowaniu założono wykorzystanie pewnej hipotetycznej klasy procesorów (maszyn) $\mathcal{M}$, wychodząc od najprostszej, sekwencyjnej *maszyny bazowej* ($\mathcal{M}_B$) poprzez *maszyny potokową* ($\mathcal{M}_P$) i *superskalarną* ($\mathcal{M}_S$), na maszynie *superskalarnej potokowej* ($\mathcal{M}_{SP}$) kończąc.

Dla wszystkich typów maszyn przyjmuje się następujące oznaczenia: maszyna $\mathcal{M}_x$, $x \in \{B, P, S, SP\}$ przetwarza n rozkazów r_i ze zbioru $\mathcal{R} = \{ r_i \mid i = 1..n \}$ w kolejności $r_{q_1}, r_{q_2}, \ldots, r_{q_n}$, gdzie $q_1, q_2, \ldots, q_n$ (lub krótko $\{q_n\}$) jest n-elementową permutacją zbioru $\{1, 2, \ldots, n\}$. Każdy z rozkazów $r_i \in \mathcal{R}$ składa się z d_i niezależnych, sekwencyjnych faz $f_{i_1}, f_{i_2}, \ldots, f_{i_{d_i}}, \ldots, f_{i_{d_i}}$ o czasach trwania odpowiednio $t_{i_1}, t_{i_2}, \ldots, t_{i_{d_i}}, \ldots, t_{i_{d_i}}$. Czasy rozpoczęcia i zakończenia j-tej fazy k-tego rozkazu w sekwencji (czyli r_{q_k}) oznacza się odpowiednio przez $\varphi^f_{\{q_n\}}(k, j)$ i $\delta^f_{\{q_n\}}(k, j)$, a czasy rozpoczęcia i zakończenia k-tego rozkazu w sekwencji przez $\varphi_{\{q_n\}}(k)$ i $\delta_{\{q_n\}}(k)$, przy czym przez *rozpoczęcie* i *zakończenie* rozkazu uważa się odpowiednio rozpoczęcie pierwszej i ostatniej fazy rozkazu:

$$\varphi_{\{q_n\}}(k) = \varphi^f_{\{q_n\}}(k, 1), \qquad \delta_{\{q_n\}}(k) = \delta^f_{\{q_n\}}(k, d_{q_k}) \tag{2.1}$$

Czasu rozpoczęcia i zakończenia rozkazu r_i określa zależność:

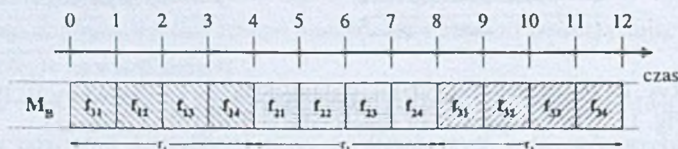
$$\varphi^r_{\{q_n\}}(i) = \varphi_{\{q_n\}}(\arg q_n(i)), \qquad \delta^r_{\{q_n\}}(i) = \delta_{\{q_n\}}(\arg q_n(i)) \tag{2.2}$$

2.1. Maszyna bazowa $\mathcal{M}_B$

W przypadku maszyny bazowej $\mathcal{M}_B$ podział procesu wykonania rozkazu na fazy ma charakter formalny: z punktu widzenia użytkownika rozkaz $r_i \in \mathcal{R}$ wykonywany jest przez $\sum_{j=1}^{d_i} t_{i_j}$ taktów. W rzeczywistości architektura procesora opiera się na dystrybucji faz, przy czym ma ona charakter ściśle sekwencyjny, tzn. następna faza rozkazu rozpoczyna się po zakończeniu poprzedniej, a kolejny rozkaz po zakończeniu poprzedniego:

$$\varphi^j_{\{q_n\}}(k,j) = \begin{cases} \delta^j_{\{q_n\}}(k-1, d_{q_{k-1}}) & gdy\ j=1 \\ \delta^j_{\{q_n\}}(k, j-1) & gdy\ 1 < j \le d_{q_k} \end{cases} \tag{2.1.1}$$

W danej chwili wykonywany jest więc tylko jeden rozkaz; nie jest możliwe równoczesne przetwarzanie kilku faz kolejnych rozkazów. Sposób wykonywania trzech czterofazowych rozkazów na maszynie bazowej przedstawiono na rysunku 2.



Rys. 2. Sposób wykonania rozkazów na maszynie bazowej
Fig. 2. Instruction processing in the base machine

Ponieważ czas wykonywania rozkazu nie jest w żaden sposób zakłócony oczekiwaniem na zwolnienie zasobów procesora, można zapisać:

$$\delta^j_{\{q_n\}}(k,j) = \varphi^j_{\{q_n\}}(k,j) + t_{(q_k)_j}$$
$$\delta_{\{q_n\}}(k) = \varphi_{\{q_n\}}(k) + \sum_{j=1}^{d_{q_k}} t_{(q_k)_j} \tag{2.1.2}$$

Wychodząc od czasu zakończenia ostatniej fazy rozkazu kończącego sekwencję $\delta^j_{\{q_n\}}(n, d_{q_n})$ i zakładając, że rozpoczęcie programu następuje w chwili $\varphi^j_{\{q_n\}}(1,1) = 0$, równania (2.1.1) i (2.1.2) wyznaczają łączny czas wykonania programu $T_{MB}(n)$, który w żaden sposób nie zależy od doboru sekwencji $\{q_n\}$.

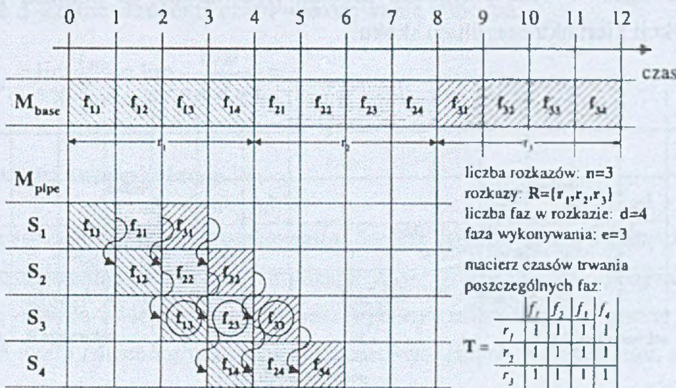
$$T_{MB}(n) = \delta^j_{\{q_n\}}(n, d_{q_n}) = \sum_{i=1}^{n} \sum_{j=1}^{d_i} t_{i_j} \tag{2.1.3}$$

Przy założeniu stałej liczby faz dla wszystkich rozkazów oraz przy jednostkowym czasie trwania każdej z faz czas wykonania programu upraszcza się do wyrażenia:

$$T_{MB}(n) = n \cdot d \tag{2.1.4}$$

2.2. Maszyna potokowa $\mathcal{M}_P$

Mechanizm potokowania zastosowany w jednostce przetwarzającej $\mathcal{M}_P$ polega na podziale procesu wykonywania rozkazu $r_i \in \mathcal{R}$ na d_i niezależnych podprocesów (faz). Dzięki temu możliwe staje się *równolegle* wykonywane faz $f_{(q_{i-d})_{d_1}}, f_{(q_{i-d+1})_{d_2}}, \ldots, f_{(q_i)_{d_{q_i}}}$ kilku następujących po sobie rozkazów (rysunek 3). Aby osiągnąć taki efekt, jednostka przetwarzająca poddawana jest podziałowi na d niezależnych stopni, odpowiadających za wykonanie kolejnych faz rozkazu.



	f_1	f_2	f_3	f_4
r_1	1	1	1	1
r_2	1	1	1	1
r_3	1	1	1	1

Rys.3. Porównanie wydajności jednostek o architekturze bazowej i potokowej
Fig. 3. Performance comparison in base and pipe architecture

Mechanizm potokowy zachowuje sekwencyjny charakter przetwarzania faz w obrębie rozkazu – każda kolejna faza rozpocznie się dokładnie w chwili, gdy zakończy się jej poprzednik. Różnica dotyczy *możliwości przetwarzania d faz jednocześnie*, co powoduje, iż *momentem rozpoczęcia procesu* wykonywania kolejnego rozkazu staje się *zwolnienie pierwszego stopnia potoku*, a nie – jak w maszynie $\mathcal{M}_B$ - *ukończenie przetwarzania poprzedniego* rozkazu.

$$\varphi^{f}_{(q_n)}(k, j) = \begin{cases} \delta^{f}_{(q_n)}(k-1, 1) & gdy\ j = 1 \\ \delta^{f}_{(q_n)}(k, j-1) & gdy\ 1 < j \le d_{q_k} \end{cases} \tag{2.2.1}$$

Z przetwarzaniem potokowym wiąże się kilka ważnych problemów. Pierwszy polega na konieczności *zapewnienia właściwego porządku i płynności obliczeń*, czyli takiej organizacji przepływu danych pomiędzy poszczególnymi stopniami potoku (argumenty i rezultaty obliczeń), aby zachować zgodność z organizacją strumienia rozkazowego $r_{q_1}, r_{q_2}, \ldots, r_{q_n}$.

Powyższy problem rozwiązano ograniczając dostęp do jednostki arytmetyczno-logicznej procesora: w trakcie wykonywania rozkazu r_i całość obliczeń *musi być ukończona* w trakcie tzw. *fazy wykonywania* (ang. *execute*, na rysunku 3 oznaczona okręgiem), której stopień w potoku pozostaje niezmienny dla wszystkich rozkazów r_i. Pozostałe fazy pełnią pozostałe funkcje niezbędne dla prawidłowego wykonania rozkazu, czyli np.: *pobieranie, dekodowanie, generowanie adresów argumentów, wymiana danych z pamięci, zapisywanie rezultatów* itp.

Drugi problem wiąże się ze znaczną utratą wydajności potoku przy operacji wykonania skoku warunkowego. W tym przypadku problem rozwiązano za pomocą zastosowania jednostek predykcji kierunku i rezultatu skoku.

Rys. 4. Wydajność układu dla dwóch sekwencji rozkazów
Fig. 4. The machine performance for two instruction sequences

Trzeci problem wiąże się z dostępnością poszczególnych stopni potoku i występuje wtedy, gdy czasy poszczególnych faz rozkazów $\mathcal{R}$ są od siebie różne (przypadek ogólny). Rysunek 4a ilustruje taką sytuację: w trzecim takcie pracy procesora następuje konieczność wydłużenia fazy f_1 rozkazu r_2, ponieważ stopień S_2 zajęty jest przez fazę f_2 rozkazu r_1. Taka sytuacja powoduje, że dla procesora potokowego w porównaniu do $\mathcal{M}_B$ (2.1.2) zmienia się powiązanie pomiędzy czasem rozpoczęcia i zakończenia j-tej fazy k-tego rozkazu w sekwencji $\{q_n\}$.

$$\delta^{f}_{\{q_n\}}(k, j) = \varphi^{f}_{\{q_n\}}(k, j) + t^{f}_{\{q_n\}}(k, j) \tag{2.2.2}$$

Konflikt przy dostępie do stopni potoku nie może wystąpić dla żadnej fazy pierwszego rozkazu z sekwencji $\{q_n\}$ ani dla ostatniej fazy żadnego z rozkazów.

$$t^{f}_{\{q_n\}}(k, j) = \begin{cases} t_{q_k} & gdy\ k = 1 \vee j = d \\ \max(t_{q_k}, t^{f}_{\{q_n\}}(k-1, j+1)) & w\ przeciwnym\ wypadku \end{cases} \tag{2.2.3}$$

Na rysunku 4b pokazano, że przez odpowiedni dobór sekwencji rozkazów można uzyskiwać mniejszą ilość konfliktów, a co za tym idzie, krótsze czasy przetwarzania programu. Rozpatrując wyłącznie jednostkowe czasy trwania faz dla d-stopniowego potoku, można oszacować łączny czas przetwarzania ciągu n rozkazów jako

$$T_{MP}(n) = n + d - 1 \tag{2.2.4}$$

Porównując ten wynik z czasem przetwarzania n rozkazów na maszynie bazowej $\mathcal{M}_B$ (2.1.4) można otrzymać maksymalne przyśpieszenia uzyskiwane dzięki zastosowaniu mechanizmu potokowania. Zastosowanie d-stopniowego potoku umożliwia więc maksymalnie d-krotne skrócenie czasu wykonywania obliczeń.

$$S_{MP_{max}} = \lim_{n\to\infty} \frac{T_{MB}}{T_{MP}} = \lim_{n\to\infty} \frac{n \cdot d}{n + d - 1} = d \tag{2.2.5}$$

2.3. Maszyna superskalarna $\mathcal{M}_S$

Maszyna *superskalarna bez potokowania* $\mathcal{M}_S$ jest tworem czysto teoretycznym, służącym do zademonstrowania możliwości utylizacji tzw. *równoległości programu procesora*. Podstawową metodą zwiększania szybkości wykonywania programu jest w tym przypadku zastosowanie *wielu równoległych* jednostek przetwarzających. Każda z nich należy do jednej z s rozłącznych *klas jednostek* $U=\{U_1,U_2,...,U_s\}$ mogących wykonywać jedynie rozkazy z jednego z s rozłącznych podzbiorów $R_j \subseteq \mathcal{R}$.

$$R_1 \cup R_2 \cup ... \cup R_s = R \wedge \forall_{\substack{i,j \in <1,s> \\ i \neq j}} R_i \cap R_j = \varnothing \tag{2.3.1}$$

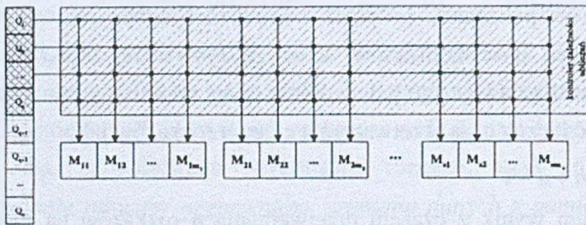
Specyfikacja każdego z rozkazów $r_i \in \mathcal{R}$ zostaje rozszerzona o tzw. *specyfikację jednostki docelowej* v_i $|v_i \in <1,s>$, która jednoznacznie identyfikuje *typ* jednostek przetwarzających. Jednostki te posiadają zdolność do rozpoznawania i przetwarzania danego rozkazu.

Liczebność jednostek przetwarzających występujących w procesorze $\mathcal{M}_S$ dana jest wektorem $\bar{m}$. I-ty element m_i oznacza liczebność zbioru maszyn *klasy i*, $i=1 ... s$. Procesor $\mathcal{M}$ składa się więc z następujących jednostek przetwarzających:

$$\begin{matrix} M_{11} & M_{12} & ... & M_{1m_1} \\ M_{21} & M_{22} & ... & M_{2m_2} \\ ... & ... & ... & ... \\ M_{s1} & M_{s2} & & M_{sm_s} \end{matrix}$$

przy czym M_{ab} oznaczono b-tą jednostkę a-tej klasy.

Automatyczne rozproszenie obliczeń uzyskuje się dzięki zmianie organizacji *kolejki rozkazów* oraz poprzez zastosowanie układu *śledzenia zależności* występujących pomiędzy poszczególnymi operacjami (rozkazami). Ogólny schemat takiego układu przedstawiono na rysunku 5.



Rys. 5. Schemat wirtualnego procesora o idealnej architekturze superskalarnej
Fig. 5. Architecture schema of ideal virtual superscalar processor

Główne elementy procesora to kolejka rozkazów[1] $Q_1, Q_2, ..., Q_q, ..., Q_n$ o organizacji FIFO i dostępie swobodnym (z punktu widzenia jednostek) w obrębie pierwszych q stopni. Układ kontroli kolejności obliczeń śledzi sposób wykonywania poszczególnych operacji i w przypadku wykrycia zależności pomiędzy dwoma rozkazami w sekwencji $\{q_n\}$ opóźnia rozpoczęcie wykonywania rozkazu zależnego.

$$\forall_{\substack{r_{q_i}, r_{q_j} \in R \\ i,j \in \langle 1,q \rangle}} (i < j) \wedge (r_{q_i} \prec r_{q_j}) \Rightarrow \delta_{(q_j)}(i) \leq \varphi_{(q_i)}(j) \quad (2.3.2)$$

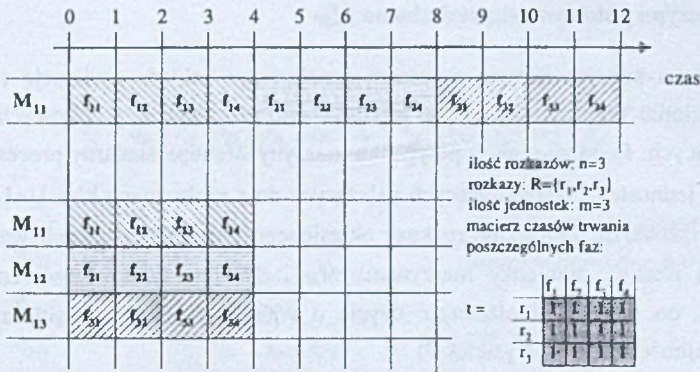
Ze względu na fakt, że koniec wykonania rozkazów może następować w kolejności niezgodnej z sekwencją $\{q_n\}$, taki sposób przetwarzania określa się czasem jako *przetwarzanie dynamiczne* albo *przetwarzanie niezgodnie z pierwotną kolejnością* (ang. *out of order processing*).

Maksymalne przyspieszenie, jakie uzyskać można dzięki zastosowaniu maszyny superskalarnej bez potokowania do przetworzenia sekwencji n rozkazów (rysunek 6), można oszacować na podstawie *prawa Amdahla:*

$$S_{MS\max} = \lim_{f \to 0} \frac{T_{MB}}{T_{MP}} = \lim_{f \to 0} \frac{n \cdot d}{f \cdot n \cdot d + \frac{(1-f) \cdot n \cdot d}{\sum_{i=1}^{s} m_i}} = \lim_{f \to 0} \frac{1}{f + \frac{(1-f)}{\sum_{i=1}^{s} m_i}} = \sum_{i=1}^{s} m_i \quad (2.3.3)$$

gdzie przez f, $f \in \mathcal{R}$, $0 \leq f \leq 1$ oznaczono tę część rozkazów ze zbioru $\mathcal{R}$, która musi być wykonana szeregowo.

[1] W modelu założono, że liczba stopni kolejki jest zgodna z liczbą rozkazów w sekwencji $\{q_n\}$; w rzeczywistości kolejka sprzężona jest z pamięcią podręczną rozkazów i ma znacznie mniejszy rozmiar. Przyjęcie takiego uproszczenia nie niesie żadnych konsekwencji dla prowadzonych rozważań.



Rys. 6. Porównanie wydajności jednostek o architekturze bazowej i superskalarnej
Fig. 6. Performance comparison in base and parallel architecture without pipelining

Podobnie jak w przypadku maszyny bazowej $\mathcal{M}_B$, podział procesu wykonania rozkazu na fazy ma jedynie charakter formalny: z punktu widzenia użytkownika rozkaz $r_i \in \mathcal{R}$ wykonywany na maszynie $\mathcal{M}_S$ jest przez $\sum_{j=1}^{d_i} t_{i_j}$ taktów pracy procesora; czas ten w żaden sposób nie zależy od doboru sekwencji $\{q_n\}$.

$$\begin{aligned} \delta^f_{\{q_n\}}(k,j) &= \varphi^f_{\{q_n\}}(k,j) + t_{(q_k)_j} \\ \delta_{\{q_n\}}(k) &= \varphi_{\{q_n\}}(k) + \sum_{j=1}^{d_{q_k}} t_{(q_k)_j} \end{aligned} \qquad (2.3.4)$$

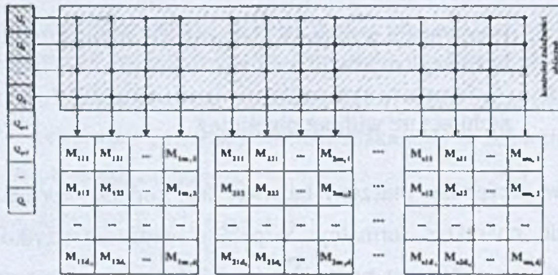
Możliwość rozpoczęcia przetwarzania k-tego rozkazu z sekwencji $\{q_n\}$ ograniczona jest trzema zjawiskami: *gotowością do rozpoczęcia obliczeń* (zgodnie ze wzorem 2.3.2), dostępnością przynajmniej jednej *wolnej jednostki dedykowanej* oraz przebywaniem w tej części kolejki, do której jednostki mają *dostęp swobodny*. Oznaczając przez *usage* funkcję *wykorzystania jednostek* w danej chwili oraz przez *queue* funkcję przyjmującą wartość numeru stopnia kolejki, w którym rozkaz znajduje się w danej chwili, otrzymać można zestaw warunków, jakie musza być spełnione dla rozpoczęcia *k-tego* zadania w sekwencji.

$$\delta_{\{q_n\}}(k) = \lambda \Rightarrow \begin{cases} \forall_{l \in (1,k)} r_{q_l} \prec r_{q_k} \Rightarrow \lambda \geq \varphi_{\{q_n\}}(l) \\ usage(v_i, \lambda) \leq m_{v_i} \\ queue(q_k, \lambda) \leq q \end{cases} \qquad (2.3.5)$$

Dodatkowo należy zaznaczyć, że rozkaz rozpoczyna się w najwcześniejszej możliwej chwili oraz że rozkazy analizowane są zgodnie z organizacją FIFO (odpowiednio od l do k).

2.4. Maszyna potokowa superskalarna $\mathcal{M}_{SP}$

Architektura superskalarnego procesora potokowego zakłada utylizację równoległości niskiego poziomu poprzez równoległe wykonywanie rozkazów w potokowych jednostkach przetwarzających. Podobnie jak w przypadku maszyny $\mathcal{M}_S$, superskalarny procesor potokowy składa się z jednostek przetwarzających należących do s rozłącznych klas $U=\{U_1,U_2,...,U_s\}$, mogących przetwarzać wyłącznie rozkazy określonego typu (spełniających warunek 2.3.1). Podstawową różnicą pomiędzy maszynami $\mathcal{M}_{SP}$ i $\mathcal{M}_P$ jest fakt zastosowania jednostek potokowych, co oznacza konieczność użycia o wiele bardziej złożonych mechanizmów kontroli kolejności obliczeń (rysunek 7).



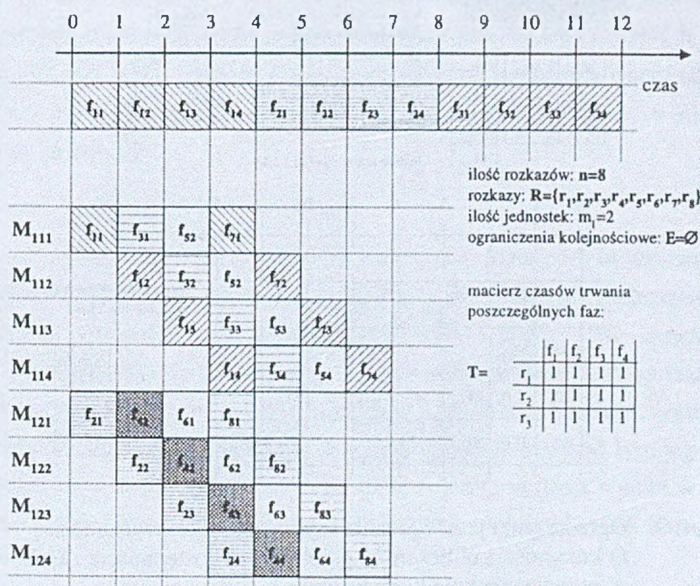
Rys. 7. Schemat wirtualnego procesora o architekturze superskalarnej potokowej

Fig. 7. Architecture schema of virtual superscalar pipe processor

Liczebność jednostek poszczególnych typów dana jest wektorem $\vec{m}$, którego i-ty element oznaczany przez m_i oznacza liczebność zbioru maszyn klasy i. Łącznie procesor składa się więc z $\sum m_i$ jednostek przetwarzających M_{ij}, gdzie M_{ij} oznacza j-tą jednostkę i-tej klasy. W odróżnieniu od maszyny $\mathcal{M}_S$ każdą z jednostek przetwarzających procesora $\mathcal{M}_{SP}$ poddano podziałowi na d_i niezależnych stopni o organizacji $M_{ij1}, M_{ij2}, ..., M_{ijk}, ..., M_{ijd}$ odpowiadających kolejnym fazom wykonania rozkazu.

Mechanizm przetwarzania z wykorzystaniem metod przetwarzania równoległego i technik potokowych przedstawiono na rys.8. W przypadku gdy zbiór ograniczeń kolejnościowych jest pusty, zastosowanie o d-stopniowych potoków zaowocuje maksymalnie $d\sum m_i$ -krotnym wzrostem prędkości obliczeń.

W rzeczywistości osiągi potokowych procesorów superskalarnych są zdecydowanie słabsze (rys. 9). Podobnie jak w przypadku maszyn $\mathcal{M}_P$ i $\mathcal{M}_S$, na ten stan rzeczy wpływ mają zarówno ograniczenia związane z programem (*kolejność obliczeń*), jak również z konfiguracją procesora (*głębokość kolejki, ilość jednostek przetwarzających* oraz *czasy przetwarzania poszczególnych faz rozkazów*).



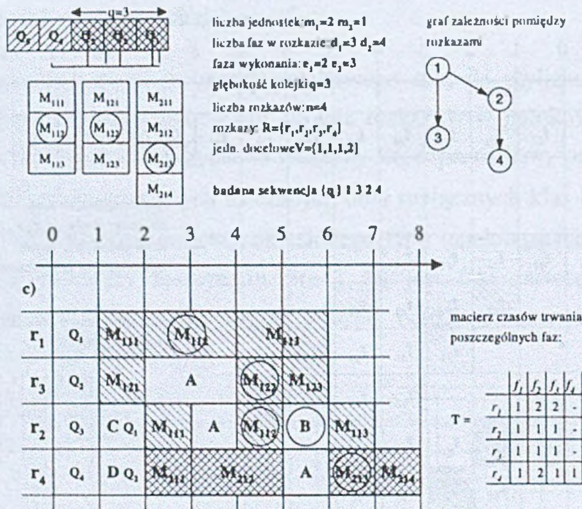
Rys. 8. Przetwarzanie rozkazów na maszynie superskalarnej potokowej
Fig. 8. Instruction processing on superscalar pipe machine

W związku z tym dla procesora $\mathcal{M}_{SP}$ relacja pomiędzy czasem rozpoczęcia i zakończenia j-tej fazy k-tego rozkazu w sekwencji $\{q_n\}$ przestaje być stała; algorytm znajdowania tej wartości zamieszczono w pracy [4].

$$\delta^{j}_{\{q_n\}}(k,j) = \varphi^{j}_{\{q_n\}}(k,j) + t^{j}_{\{q_n\}}(k,j) \tag{2.4.1}$$

W potokowym procesorze superskalarnym zachowanie właściwej kolejności przetwarzania zagwarantowane jest poprzez zachowanie *odpowiedniej synchronizacji faz wykonywania* - rozkaz oczekuje na wejście do fazy wykonania tak długo, aż wszystkie poprzedzające go w sekwencji $\{q_n\}$ i powiązane z nim rozkazy zakończą swoje fazy wykonywania. Warunki związane z kolejką oraz dostępnością zasobów i stopni potoków są zgodne z (2.3.5) i (2.2.3).

$$\underset{\substack{r_{q_i}, r_{q_j} \in R \\ i,j \in <1,q>}}{\forall} (i<j) \wedge (r_{q_i} \prec r_{q_j}) \Rightarrow \delta^{j}_{\{q_n\}}(i,e_{q_i}) \le \varphi^{j}_{\{q_n\}}(j,e_{q_j}) \tag{2.4.2}$$



Rys. 9. Ograniczenia przetwarzania superskalarnego związane z: a) kolejnością obliczeń, b) potokiem, c) dostępnością zasobów, d) organizacją kolejki rozkazów

Fig. 9. The limits of superscalar processing related to: a) computation sequence, b) pipe resource availability, d) instruction queue organisation

3. Przestrzeń rozwiązań

W punkcie drugim wspomniano, że maszyna $\mathcal{M}_x$, $x \in \{B, P, S, SP\}$ przetwarza n rozkazów r_i ze zbioru $\mathcal{R} = \{ r_i \mid i = 1..n \}$ w kolejności $r_{q_1}, r_{q_2}, \dots, r_{q_n}$, gdzie $q_1, q_2, \dots, q_n$ jest n-elementową permutacją zbioru $\{1, 2, \dots, n\}$. Niniejszy rozdział jest próbą odpowiedzi na pytanie, które permutacje nie zmieniają semantyki programu procesora. Podzbiór takich dozwolonych permutacji jest jednocześnie przestrzenią rozwiązań dla problemu znajdowania optymalnych uszeregowań rozkazów.

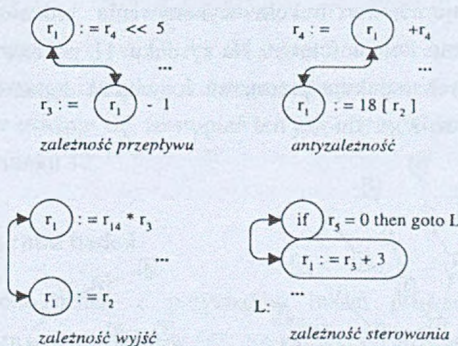
3.1. Ograniczenia kolejnościowe i ich reprezentacja

Rozpatrując problem szeregowania rozkazów, nie można pominąć problemu zależności danych, na których operują poszczególne rozkazy. Przetwarzając rozkazy $\mathcal{R} = \{ r_i \mid i = 1..n \}$, procesor musi zachować narzuconą kolejność, wyrażoną w postaci częściowego porządku na zbiorze $\mathcal{R}$. Porządek ten powoduje, że nie wszystkie permutacje zbioru $\mathcal{R}$ są dopuszczalne – podstawowym ograniczeniem jest tutaj konieczność zachowania prawidłowej kolejności

obliczeń. Ta z kolei wynika z semantyki programu procesora i dana jest skierowanym acyklicznym grafem zależności danych (ang. *Data Dependence Dag*) $DDD=(\mathcal{R}, \mathcal{E})$, gdzie $\mathcal{R}$ jest zbiorem rozkazów, a $\mathcal{E}$ - zbiorem krawędzi (ograniczeń kolejnościowych) takich, że jeśli rozkaz r_i ma być wykonany przed rozkazem r_j (co zapisywane jest jako $r_i \prec r_j$), to krawędź (r_i, r_j) należy do zbioru $\mathcal{E}$:

$$E = \{(r_i, r_j) \mid \forall_{r_i, r_j \in R, i \neq j} \; r_i \prec r_j \Rightarrow (r_i, r_j) \in E\} \tag{3.1.1}$$

Relacja $r_i \prec r_j$ zachodzi wtedy, gdy rozkaz r_j nie może być wykonany wcześniej niż po zakończeniu rozkazu r_i. Literatura wyróżnia cztery przypadki zależności rozkazów (rys.10). Rozkazy pozostają w zależnościach: *przepływu* (ang. *flow dependence, true dependence, data dependence*), jeśli rozkaz wykorzystuje wyniki wykonania rozkazu poprzedniego; *antyzależności* (ang. *anti-dependency*), jeśli rozkaz zapisuje wynik w rejestrze (komórce pamięci), z którego korzysta rozkaz poprzedni, niszcząc przy tym jego zawartość; *zależności wyjść* (ang. *output dependency*), jeśli oba rozkazy zapisują wyniki w tym samym rejestrze lub *zależności sterowania*, jeśli w treści programu występuje skok warunkowy.



Rys. 10. Rodzaje zależności pomiędzy rozkazami
Fig. 10. The types of data dependencies

Na potrzeby niniejszego opracowania przyjmuje się, że graf DDD jest acykliczny, tzn. dotyczy zbioru instrukcji w obrębie tzw. bloku podstawowego, czyli *ciągu operacji z jednym wejściem i wyjściem, którymi są odpowiednio pierwsza i ostatnia operacja.*

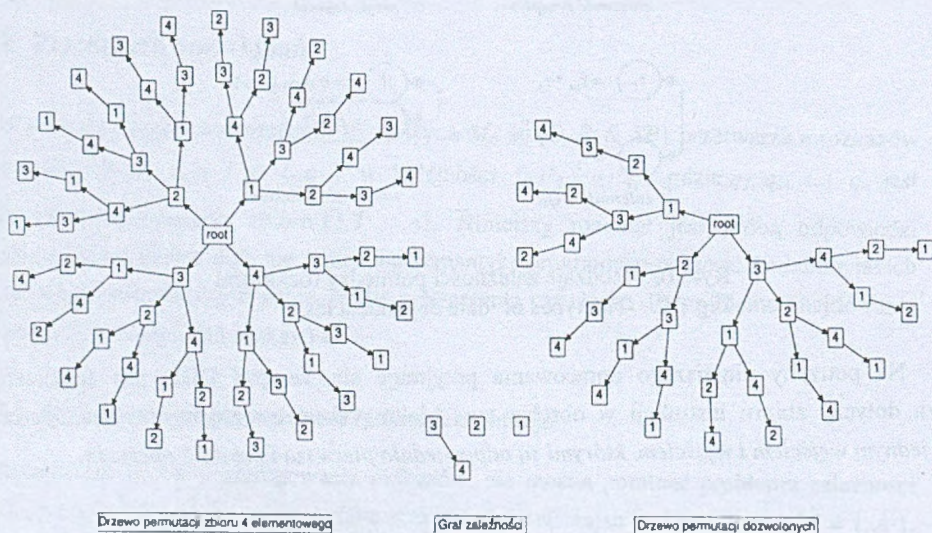
Tabela 1

Przykłady różnych rodzajów zależności między danymi

Mov eax, 8 Mov [ebp], eax	Zależność przepływu polega na tym, że pierwszy rozkaz wykonuje zapis do rejestru, z którego drugi odczytuje dane.
Mov eax, 8 Mov eax, [ebp]	Zależność wyjść ma miejsce w przypadku, gdy oba rozkazy wykonują zapis do tego samego rejestru.
Mov eax, ebx Mov ebx, [ebp]	Para rozkazów, z których pierwsza wykonuje odczyt, a druga zapisuje z rejestru tworzy antyzależność.

3.2. Sortowanie topologiczne grafu

Ograniczenia kolejnościowe zadane grafem DDD powodują, że jedynie pewien podzbiór zbioru wszystkich permutacji $\mathcal{R}$ jest dopuszczalny. Każda z permutacji $q_1,q_2,...,q_n$ musi bowiem posiadać tę własność, że jest uporządkowaniem topologicznym grafu zależności danych DDD, tzn. jeśli $r_i \prec r_j$, to q_i musi poprzedzać q_j w sekwencji $\{q_n\}$. Innymi słowy, podzbiór permutacji dozwolonych zbioru rozkazów jest zbiorem uporządkowań topologicznych grafu DDD. Przetwarzanie rozkazów zgodnie z uporządkowaniem topologicznym gwarantuje, że w trakcie wykonywania procesor zachowa wszystkie narzucone mu ograniczenia kolejnościowe. Na rysunku 11 pokazano, w jaki sposób graf ograniczeń kolejnościowych redukuje przestrzeń rozwiązań dopuszczalnych (przykładowy graf składa się z n=4 rozkazów).



Rys. 11. Redukcja przestrzeni rozwiązań dopuszczalnych
Fig. 11. Reduction of admissible solution space

4. Wyniki badań empirycznych nad utylizacją równoległości niskiego poziomu

Przeprowadzone do tej pory [4][5] badania ILP (ang. *instruction level parallelism*) wykazały, że średni stopień współbieżności współczesnych programów wynosi około 2. Wobec tego, aby uzyskać optymalnie przetworzony kod, w zupełności wystarczy zastosowanie maszyn superskalarnych stopnia równego co najwyżej 3. Stosowanie maszyn wyższego stopnia nie jest więc opłacalne, gdyż umożliwia tylko nieznaczne polepszenie wyników.

4.1. Przyczyny niedokładności pomiarów

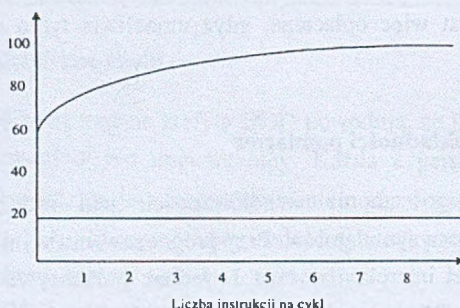
Badania maksymalnego stopnia współbieżności, jaki mogą osiągnąć procesory, przeprowadza się za pomocą symulatorów. Przy prognozowaniu wyników często zakłada się, że czas wykonania każdej instrukcji wynosi 1. Jednak w rzeczywistości czas ten w wielu przypadkach jest dłuższy. Dlatego po obliczeniu przewidywanych wyników i porównaniu ich z rzeczywistymi pomiarami doświadczalnymi otrzymujemy znaczne rozbieżności. Przykładowo, dla komputera typu CRAY-1 stopień współbieżności wynosi 2.7, przy założeniu że czas wykonania instrukcji każdego rodzaju wynosi 1. Jednak po dokonaniu rzeczywistych pomiarów okazuje się, że stopień ten jest niższy. Różnica dla obu przypadków jest przedstawiona na rysunku 12.

4.2. Przykładowe wyniki badań

Poniżej przedstawiono jeden z przykładów badań prowadzonych nad stopniem równoległości współczesnych procesorów [4]. Na zaimplementowanym symulatorze została przeprowadzona symulacja wykonania kilku programów na różnych procesorach. System symulatora oparty był na procesorze MIPS R2000 RISC. Eksperyment został zasymulowany na kilku rodzajach maszyn. Maszyna podstawowa składała się z jednostki ALU, dodawania zmiennoprzecinkowego (FP), mnożenia (FP), dzielenia (FP i konwersji FP), potoków odczytu (ang. *Load*) i zapisu (ang. *Store*) danych z pamięci, jednostki przewidującej kierunki skoków warunkowych (ang. *Branch Prediction Unit*) oraz rejestru przesuwnego (ang. *Shifter*).

Badano zależność stopnia współbieżności symulowanych maszyn od liczby uszeregowanych w danym momencie instrukcji i od liczby niezależnych jednostek wykonawczych. Założono, że jednostka przewidująca kierunki rozkazów skoków zawsze przewiduje adresy skoków oraz że bufor pobierania jest w stanie umieścić w kolejce podczas jednego cyklu tyle rozkazów, by nie było w nim wolnych miejsc. Wynik przedstawiono

na rysunku 13a. Na osi poziomej każda z liczb 1,2,3,4 oznacza maszynę składającą się z innych jednostek przetwarzających. Opis konfiguracji tych czterech maszyn znajduje się w tabeli 2. Podany wynik jest stosunkiem liczby cykli potrzebnych do wykonania programu dla klasycznej architektury skalarnej do liczby cykli potrzebnych do wykonania danego programu dla architektury superskalarnej.



Rys. 12. Różnice w wynikach badań: a) przy uwzględnieniu rzeczywistych czasów wykonania rozkazów, b) przy założeniu idealnych czasów wykonania równych 1
Fig. 12. Differences in investigation results: a) taking into consideration the real time of instruction execution, b) with the assumption of ideal execution time equal 1

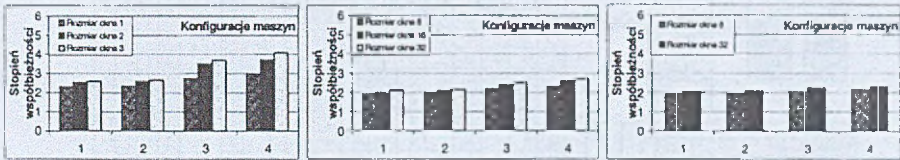
Tabela 2

Rodzaje symulowanych maszyn, które brały udział w testach

Procesor	Konfiguracja
1	Maszyna podstawowa
2	1 + jednostka ładowania potoku
3	1 + jednostka stałoprzecinkowa ALU
4	1 + jednostka ładowania potoku i jedn. stałoprzecinkowa ALU

Otrzymany wynik waha się od 1.9 do 2.5. Okazuje się, że stopień współbieżności jest większy dla konfiguracji zawierających dodatkową jednostkę ALU. Natomiast większy rozmiar okna i większa liczba jednostek przetwarzających mają mniejszy wpływ na stopień współbieżności. Przyczyną tego zjawiska jest fakt, że w rzeczywistości maszyny te nie mają możliwości głębszego wglądu w kod programu z powodu ograniczeń narzuconych przez kolejność obliczeń.

Rysunek 13b przedstawia rezultaty badań przeprowadzonych po zredukowaniu liczby konfliktów między rejestrami poprzez dynamiczną zmianę nazw rejestrów. Po sprawdzaniu tylko zależności przepływu wyniki są bardziej korzystne i wynoszą od 2.3 do 4.1. Jak widać na rysunku, pomiary dla różnych rozmiarów okien są więc bardziej zróżnicowane. Wynika to ze zmniejszenia liczby zależności między danymi, co umożliwiło wgląd w dalsze instrukcje okna. Jednak również i w tym przypadku najbardziej widoczna różnica dotyczy maszyn z dodatkową jednostką ALU.



Rys. 13. Przyspieszenie dla maszyny superskalarnej: a) wszystkie zależności między danymi, idealna jednostka pobierania, b) tylko zależności przepływu, idealna jednostka pobierania, c) tylko zależności przepływu, poziom prawidłowej predykcji BPU: 85%

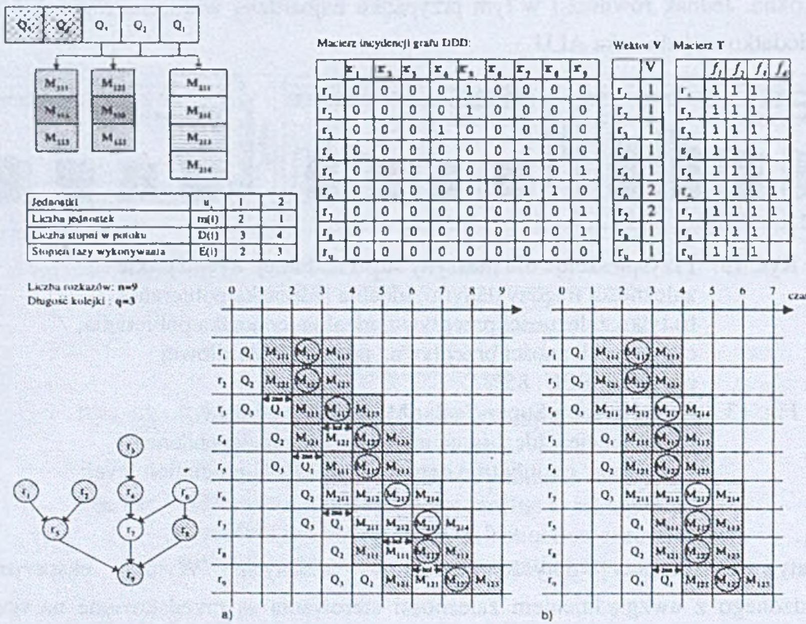
Fig. 13. Speedup for a Super-Scalar Machine: a) all data dependencies, ideal issue unit, b) only true dependencies, ideal issue, c) only true dependencies, BPU prediction level: 85%

Rezultaty te niestety dotyczą modelów idealnych. Wyniki eksperymentu przeprowadzonego z uwzględnieniem zależności sterowania są przedstawione na rys.13c. Przy założeniu że liczba instrukcji wziętych pod uwagę przy przewidywaniu skoków wynosi cztery (co piąty rozkaz jest rozkazem skoku) oraz że układ przewidywania jest w stanie dobrze przewidzieć 85% wszystkich testowanych skoków, wyniki leżą pomiędzy 1.9 a 2.3, a dodanie drugiej jednostki ALU nie wpływa wyraźnie na zwiększenie stopnia równoległości.

5. Podsumowanie

Znajdowanie optymalnych sekwencji rozkazów dla ogólnego przypadku potokowego procesora superskalarnego to nietrywialne zadanie optymalizacyjne, nawet gdy liczba wierzchołków w grafie DDD nie jest duża. W praktycznych zastosowaniach [1] liczba wierzchołków waha się w granicach od kilkunastu do kilkudziesięciu. Dopuszczalna przestrzeń rozwiązań jest w tym przypadku tak duża, że praktycznie uniemożliwia znajdowanie rozwiązań optymalnych. W takim przypadku należy rozważyć możliwość zastosowania heurystyk, opartych np. na teorii szeregowania zadań.

Zysk uzyskany dzięki zastosowaniu odpowiednich heurystyk jest znaczący. Rysunek 14 ilustruje dwa uszeregowania topologiczne grafu programu składającego się z n=9 rozkazów. Różnica w czasach wykonania programów uszeregowanych w oparciu o obie sekwencje wynosi ponad 20%, co jest wystarczającym argumentem do prowadzenia dalszych badań.



Rys. 14. Wpływ uszeregowania rozkazów na sumaryczną prędkość przetwarzania: a) sekwencja 1,2,3,4,5,6,7,8,9, b) sekwencja 1,3,6,2,4,7,5,8,9

Fig. 14. Influence of instruction scheduling on total speed processing sequence: a) sequence 1,2,3,4,5,6,7,8,9, b) sequence 1,3,6,2,4,7,5,8,9

LITERATURA

1. Torng H.C., Vassiliadis S.: Instruction-Level Parallel Processors, IEEE Computer Society Press L.A. 1995.
2. Noonburg B, Shen J.:Theoretical modeling of Superscalar Processor Performance, Department of Electrical and Computer Engeneering, Carnegie Mellon University.
3. Rudd K.:Instruction Level Parallel Processors – a new architectural model for simulation and analysis, Technical Raport, Standford University 1994.

4. Jouppi N.P., Wall D.W.: Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines. IEEE Computer Society Press, LosAlamitos, California, 1995.
5. Smith M.D., Johnson M, Horowitz M.A.: Limits on Multiple Instruction Issue. IEEE Computer Society Press, LosAlamitos, California, 1995.
6. Czech Z - red.: Raport z badań kierunkowych Politechnika Śląska, Gliwice 1998.

Recenzent: Dr Mieczysław Wodecki

Wpłynęło do Redakcji 16 listopada 1999 r.

Abstract

The growth of the computational performance requirements could be resolved in two ways: by *shortening of machine cycle time* or by utilising a *low-level parallelism* of the processor source program. Modern superscalar pipelined processors integrate both techniques: sequenced instructions are distributed into multiplied parallel machines processing units, developed in the pipeline technology. Thanks to this, it is possible to work with very huge frequencies when processing a parallel dozen instructions.

In real situation the processing acceleration of such machines is strongly limited, due to dependencies between instructions, resource conflicts and queue organisation. Among many others, there exist sequences, which utilise resources in a better way. The problem of finding such sequences is called *instructions scheduling problem*. The aim of this article is the try of formulation of this problem as the discrete optimisation problem.

On the figure 1 we can see the comparison of superscalar and VLIW architecture. Figures 2 and 3 shows the comparison of performance respectively for base and pipelined architecture. Figure 4 illustrates the pipelined machine performance for two instruction sequences. Next four figures show the scheme and the acceleration of the ideal superscalar (fig. 5 and 6) and pipelined superscalar processor (fig. 7 and 8); figure 9 shows constraints of such techniques. The types of data dependencies are shown on the figure 10 and the examples of them are inserted in the table 1. The reduction of admissible solution space is placed in the figure 11. There are also shown the differences in investigation results taking into consideration real time of instruction execution and with an assumption of ideal execution time equal 1 on the fig. 12. On the fig.13 we can see the results of these investigations made on four types of machines described in the table 2. Subsequently, there is also shown the influence of instruction scheduling on total speed processing on the figure 14.