ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 35 Nr kol.1402

Urszula GESSING
Politechnika Śląska, Instytut Informatyki

OCENA JAKOŚCI WYBRANYCH KOMPILATORÓW JĘZYKA C/C++ POD WZGLĘDEM OPTYMALIZACJI KODU WYNIKOWEGO DLA PROCESORÓW SUPERSKALARNYCH NA PRZYKŁADZIE PROCESORA PENTIUM

Streszczenie. Artykuł poświęcony jest badaniu kompilatorów języka C/C++ pod względem optymalizacji kodu wynikowego dla procesora Pentium jako reprezentanta procesorów superskalarnych. Uwzględniono zagadnienia szeregowania rozkazów dla procesora Pentium.

QUALITY ESTIMATION OF CHOSEN C/C++ COMPILERS IN RESPECT OF OUTPUT CODE OPTIMIZATION FOR SUPERSCALAR PROCESSORS IN EXAMPLE OF PENTIUM PROCESSOR

Summary. Article is dedicated to testing of C/C++ compilers in respect of output code optimization for Pentium processor as representant of superscalar processors. The problem of Pentium instruction scheduling is considered.

1. Wstęp

Podstawowym problemem dla różnej klasy procesorów jest szybkość przetwarzania rozkazów. Istnieją dwie podstawowe metody zwiększenia wydajności procesorów. Pierwszą z nich jest korzystanie ze zredukowanej listy rozkazów, których wykonanie jest szybsze. Dzięki takiej koncepcji możliwa jest budowa układów pracujących z częstotliwościami przekraczającymi 500 MHz. Przykładem mogą być procesory typu Alpha[1].

Innym sposobem zwiększenia prędkości przetwarzania jest stosowanie kilku niezależnych jednostek, mogących pracować równolegle. Przykładem tego rodzaju procesorów są układy

przetwarzające rozkazy równolegle (ang. Low Level Parallel Processor). W literaturze noszą one nazwę układów superskalarnych.

W wyniku tych dążeń powstały procesory superskalarne o większej liczbie jednostek przetwarzających i co za tym idzie, większej mocy obliczeniowej. Jednym z przykładów takiego układu jest procesor Pentium[2].

Niniejszy artykuł poświęcony jest badaniu kompilatorów języka C/C++ pod względem optymalizacji kodu wynikowego dla procesorów superskalarnych na przykładzie procesora Pentium. Został zbadany czas wykonania rozkazów w potokach przetwarzających procesora. Do przeprowadzenia testów wybrano dwa kompilatory języka C: Borland C for Windows 5.0 oraz Watcom 10.0b. Testy zawierające nieuszeregowane rozkazy skompilowano z wyłączonymi oraz włączonymi opcjami optymalizacji. Porównano czas wykonania testów.

2. Podstawowe elementy budowy procesora Pentium

Procesor Pentium jest procesorem superskalarnym, ponieważ ma dwie niezależne jednostki przetwarzające. Każda z jednostek przetwarza rozkazy potokowo; jednostki te nazwane są potokiem U i potokiem V. Rozkazy mogą być przetwarzane niezależnie i równolegle w obu potokach. W potoku U wykonywane są instrukcje na danych całkowitych i zmiennopozycyjnych, natomiast w potoku V - tylko proste instrukcje z danymi całkowitymi oraz instrukcje zamiany danych zmiennopozycyjnych w rejestrach, pomocne w obliczeniach pośrednich. Potoki nie są więc sobie równoważne ani zamienne między sobą.

W procesorze Pentium nie wszystkie rozkazy mogą być jednak przetwarzane równolegle. Wynika to z budowy potoków. Sposób wykonania rozkazów w potokach opisują zasady łączenia rozkazów w pary (patrz rozdz. 6). Ponadto nie dla wszystkich instrukcji czas wykonania wynosi 1 cykl zegarowy. Najprostsze są instrukcje operujące na rejestrach, bez dostępu do pamięci i dla nich czas wykonania wynosi najczęściej 1 cykl. One również najczęściej mogą być wykonywane współbieżnie.

2.1. Sposób przetwarzania rozkazów w procesorze Pentium

W procesorze Pentium rozkaz ma 5 faz:

1. **PF** (ang. prefetch) - **wstępne pobranie rozkazu**. Rozkaz pobierany jest z pamięci podręcznej kodu lub z pamięci głównej. W fazie tej bufory wstępnego pobierania (ang. prefetch buffers) oraz bufor BTB (ang. branch target buffer) przetwarzają dwie niezależne linie 32-bajtowe.

2. **D1** (ang. instruction decode) - **faza I dekodowania rozkazu**. Dekoder pobiera i dekoduje dwa następne rozkazy. Według zasad łączenia rozkazów w pary decyduje, czy dane dwa rozkazy mogą być wykonane równolegle.

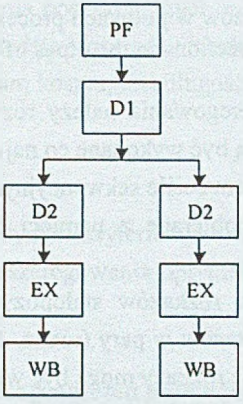
3. **D2** (ang. address generate) - **faza II dekodowania rozkazu.** Ustalenie adresu argumentów rezydujących w pamięci.

4. **EX** (ang. execution) - **wykonanie rozkazu.** Jest to faza, w której wykonywane są operacje jednostki arytmetyczno-logicznej i operacje dostępu do pamięci podręcznej danych.

5. **WB** (ang. write back) - **zapis wyniku operacji.** Następuje tutaj ostateczne wykonanie rozkazu.

Do wykonania dwóch pierwszych faz (wstępne pobieranie i pierwsza faza dekodowania) procesor posiada po jednej jednostce przetwarzającej. Rozdział na dwa niezależne potoki następuje dopiero dla trzech ostatnich faz (druga faza dekodowania, wykonanie i zapis wyniku) (rys. 1).

Podczas fazy D1 pobierane są rozkazy z jednostki wstępnego pobierania rozkazów. Dekoder decyduje o tym, czy mogą być przetwarzane dwa rozkazy równocześnie i sprawdza, czy między rejestrami występują uzależnienia. Następnie generowane jest słowo sterujące, wyznaczające przepływ rozkazów w obu potokach i używane w systemie sterowania procesora, nadzorującym potokowe przetwarzanie rozkazów. W fazie D2 w każdym z potoków następuje zdekodowanie słowa sterującego i obliczenie adresów argumentów w pamięci.



Rys. 1. Wykonanie rozkazów stałopozycyjnych w potokach procesora Pentium
Fig. 1. Execution of fixed-positioned instructions in the pipes of Pentium processor

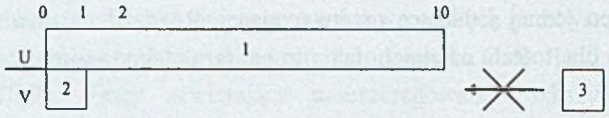
2.2. Charakterystyczne cechy procesora Pentium

Jeśli czas wykonania grupy rozkazów ma być jak najkrótszy, rozkazy należy optymalnie uszeregować. W tym celu należy wybrać odpowiedni algorytm szeregujący (rozdz. 3.6). Charakterystyczne są następujące cechy dotyczące działania procesora Pentium, mające wpływ na wybór algorytmu szeregowania:

1. Rozkazy w potokach U i V przesuwają się synchronicznie przez poszczególne stopnie przetwarzania. Jeżeli rozkaz w jednym potoku zakończy daną fazę wcześniej, to czeka, aż ten

drugi również ją zakończy tak, by oba mogły przejść jednocześnie do następnej fazy. Określa się to jako „ściśle uporządkowany" model przetwarzania.

Jeżeli jeden z dwóch równocześnie przetwarzanych rozkazów zakończył się wcześniej, a przetwarzanie drugiego jeszcze trwa, to nie jest możliwe rozpoczęcie przetwarzania trzeciego rozkazu w wolnym potoku podczas trwania wykonywania nieskończonego rozkazu. Trzeci rozkaz może być przetwarzany dopiero po zakończeniu wykonania obydwu rozkazów (rys. 2).



a) Rozkaz 3 nie może być wykonany równolegle z rozkazem 1.

b) Rozkaz 3 będzie wykonany jako następny w potoku U, ewentualnie w parze z rozkazem następującym po nim.

Rys. 2. Przepływ rozkazów w potokach procesora Pentium
Fig. 2. Flow of the instructions in the pipes of Pentium processor

W związku z tym problem szeregowania należy rozwiązywać dla modelu, w którym w tym samym przedziale czasu mogą być wykonane co najwyżej dwa rozkazy.

2. Procesor Pentium jest modelem ściśle sekwencyjnym, tzn. że rozkazy są wykonywane w takiej kolejności, w jakiej są pobierane z pamięci podręcznej, a wyniki zapisywane w kolejności ich generowania.

3. Na model szeregowania dla rozkazów stałopozycyjnych procesora Pentium mają wpływ również zasady łączenia rozkazów w pary (rozdz. 4). Powodują one, że potoki nie są równoważne, ponieważ nie wszystkie rozkazy mogą być wykonywane w obu potokach.

4. Istotna jest także liczba cykli, w których wykonywany jest każdy rozkaz. Liczba cykli dla różnych rozkazów jest różna.

3. Problem szeregowania zadań

Opisując ogólny model szeregowania należy wymienić takie pojęcia, jak zasoby, system zadań, kryteria optymalności, ograniczenia kolejnościowe. Model ten opisany jest w [3].

Ogólny model szeregowania zadań został sprowadzony do omawianego przypadku: modelu szeregowania rozkazów w procesorze Pentium.

3.1. Zasoby

Dla zadanego modelu zbiorem zasobów są dwa niezależne potoki obliczeniowe U i V, znajdujące się w jednym procesorze. Nie są one sobie równoważne, ponieważ nie wszystkie operacje stałopozycyjne mogą być wykonywane w potoku V. Budowa potoków uniemożliwia wykonanie niektórych operacji równolegle.

3.2. System zadań

Systemem zadań jest zbiór rozkazów procesora Pentium. Dla rozkazów wykonywanych przez procesor przyjmujemy następujące założenia:

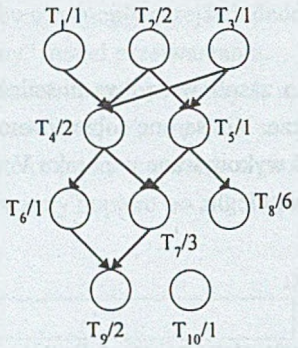
1. Zbiór zadań T jest zbiorem rozkazów stałopozycyjnych do przetworzenia w potokach U i V.
2. π oznacza relację częściowego porządku zdefiniowaną na T i określającą *ograniczenia kolejnościowe*. $T_i \pi T_j$ oznacza, że zadanie T_j nie może się rozpocząć przed zakończeniem wykonywania zadania T_i. Relacje częściowego porządku ilustrują grafy ograniczeń kolejnościowych.
3. Znane czasy wykonania poszczególnych rozkazów odpowiadają macierzy czasów wykonania zadań. Czasy wykonania poszczególnych rozkazów (oraz klasyfikacja rozkazów według zasad łączenia w pary) przedstawione są w tabeli 3.
4. Jedyną wagą, czyli kryterium kosztu, jest długość uszeregowania $\omega(f_i)=f_i$.

3.3. Kryteria optymalności

W omawianym przypadku jedynym kryterium kosztu jest **długość uszeregowania**, tzn $\omega(f_i)=f_i$. Będziemy więc szukać uszeregowania optymalnego ze względu na czas wykonania rozkazów.

3.4. Graficzna reprezentacja ograniczeń kolejnościowych

Ograniczenia kolejnościowe będą reprezentowane za pomocą acyklicznego grafu skierowanego (digrafu) $G=(A,V)$, gdzie zbiór A (zbiór wierzchołków) oznacza zbiór zadań do wykonania, a zbiór V (zbiór krawędzi) odpowiada ograniczeniom kolejnościowym (rys. 3). Każdy wierzchołek opisany jest przez T_x/y, gdzie x oznacza numer zadania, natomiast y - czas wykonania tego zadania (w cyklach zegarowych).

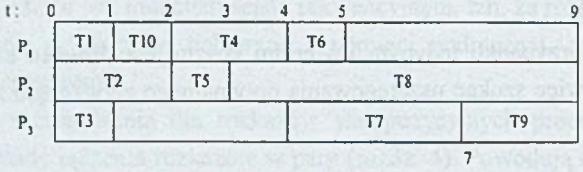


Rys. 3. Digraf
Fig. 3. Digraph

3.5. Pojęcie uszeregowania

Problem szeregowania polega na tym, że *m* **procesorów** P_j (j = 1,...,m) musi przetworzyć *n* **zadań** T_i (i = 1,...,n). **Uszeregowanie** polega na przydzieleniu każdemu z zadań jednego lub wielu przedziałów czasowych na jednej lub wielu maszynach tak, że spełnione są zadane ograniczenia.

Uszeregowania będą reprezentowane za pomocą diagramów Gantta (rys. 4). Liczba linii poziomych będących osiami czasu odpowiada liczbie procesorów, a puste pola - przedziałom czasu, w których dany procesor jest bezczynny. Uszeregowanie na rys. 4 odnosi się do digrafu z rys. 3.



Rys. 4. Wykres czasowy (diagram Gantta) dla digrafu z rys. 3
Fig. 4. Time diagram (Gantt charts) for digraph on the fig. 3

3.6. Algorytmy szeregowania rozkazów

Aby znaleźć optymalne uszeregowanie rozkazów, należy rozpatrzyć wszystkie ich permutacje. Algorytm uwzględniający wszystkie permutacje jest algorytmem NP-zupełnym, tzn. że ma złożoność n!. Istnieją również inne algorytmy, mające krótszy czas wykonania. Nazywamy je algorytmami heurystycznymi. Nie uwzględniają one jednak wszystkich permutacji i dają rozwiązanie zbliżone do optymalnego. Ponadto, aby móc je stosować,

muszą być spełnione pewne ograniczenia. Pewne ograniczenia dotyczące budowy procesora Pentium i zbioru jego rozkazów mają wpływ na wybór algorytmu szeregowania.

4. Zasady łączenia w pary

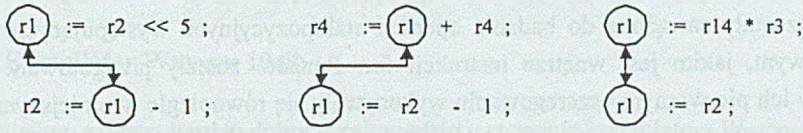
Rozkazy procesora Pentium dzieli się na cztery grupy ze względu na sposób ich wykonania w potokach. Są to następujące grupy:

- **NP** - instrukcje, których nie można łączyć w pary,
- **UV** - instrukcje, które można łączyć w pary i które mogą być wykonywane w potoku U i w potoku V,
- **PU** - operacje muszą być wykonywane w potoku U i mogą być równolegle wykonywane w parach z innymi operacjami potoku V; instrukcje te nigdy nie są wykonywane w potoku V,
- **PV** - instrukcje mogą być wykonywane w obu potokach, ale występują w parach tylko wtedy, gdy są w potoku V.

W testach zostały poddane badaniom rozkazy głównie z grupy UV, oraz część z grupy PU. Grupa PV nie została poddana badaniom, ponieważ obejmuje ona instrukcje skoku, a badania były przeprowadzone w ramach bloków podstawowych.

Istnieją również inne ograniczenia wykonania instrukcji w parach. Przede wszystkim są to zależności między danymi (rejestrami).

Wyróżniamy następujące zależności między rejestrami (rys. 5):



a) zależność przepływu b) antyzależność c) zależność wyjść

Rys. 5. Rodzaje zależności danych
Fig. 5. Types of data dependences

1. Pierwsza instrukcja zapisuje do rejestru, z którego czyta druga instrukcja (zależność przepływu - rys. 5a), np.:

```
mov eax, 8
mov [ebp], eax
```

2. Obie instrukcje zapisują do tego samego rejestru (zależność wyjść - rys. 5c), np.:

```
mov eax, 8
mov eax, [ebp]
```

W obu przypadkach instrukcje muszą być wykonane sekwencyjnie, by nie zmienić wyników obliczeń.

3. Natomiast para instrukcji, z których pierwsza czyta rejestr, a druga zapisuje do niego dane (antyzależność - rys. 5b), może być wykonana równolegle, np.:

```
mov eax, ebx
mov ebx, [ebp]
```

Są pewne instrukcje, które mogą być wykonane w parach, mimo że odbywa się to niezgodnie z ogólnymi zasadami [4].

Inne ograniczenia dotyczące wykonywania instrukcji parami.

Są pary instrukcji, których przetwarzanie może być rozpoczęte równolegle, ale faza ich wykonania nie może przebiegać równolegle:

1. Jeżeli obie instrukcje wymagają dostępu do tego samego bloku pamięci podręcznej danych, to po wygenerowaniu drugiego żądania dostępu z potoku V (pierwsze zostanie wygenerowane żądanie z potoku U) rozkaz w potoku V musi czekać, aż zakończy się obsługa żądania z potoku U.

2. Podczas wykonywania rozkazów równolegle, kolejność dostępów do pamięci z obu potoków jest tak zorganizowana, by chronić porządek dostępu do pamięci. Instrukcje wielocykliczne w potoku U będą wykonywane pojedynczo aż do zrealizowania ostatniego dostępu do pamięci [4].

5. Wstęp do opisu badań

5.1. Sposób przeprowadzenia testów

Testy zostały zawężone do badania operacji stałopozycyjnych występujących w bloku podstawowym, jakim jest wnętrze instrukcji for. Ponadto zostały przygotowane tak, by rozkazy w ich pierwotnym uszeregowaniu wykonywały się równolegle w mniejszym stopniu lub wcale. Rozkazy mogą być ewentualnie wykonywane parami dopiero po ich odpowiednim uszeregowaniu. Umożliwia to zbadanie, czy kompilator dokonuje szeregowania rozkazów w celu optymalizacji wykonania ze względu na czas. Grupa kilku rozkazów została umieszczona w pętli wykonywanej 10 milionów razy po to, by móc zmierzyć czas ich wykonania. Wewnątrz pętli test ograniczony jest z obu stron przez asemblerowy rozkaz „not". Należy on do grupy rozkazów, które nie mogą być wykonywane równolegle w procesorze Pentium (grupa NP). Umieszczenie tego rozkazu przed i po testowanym fragmencie ma na celu wyczyszczenie potoku tak, by pierwszy rozkaz z fragmentu testowanego nie wykonywał się równolegle z rozkazem poprzedzającym go (rys. 6).

Bezpośrednio przed i po pętli funkcją gettime jest pobierany czas systemowy.

Dla każdego testu, dla każdego z kompilatorów był badany czas wykonania z wyłączonymi opcjami optymalizacji, następnie z włączonymi opcjami, po których można było spodziewać się optymalizacji dla procesora Pentium. Każdy kompilator został przetestowany przez kilka programów testujących.

Najpierw mierzony był czas wykonania samej pętli z dwoma rozkazami czyszczącymi potoki. Potem zmierzono czasy wykonania testów.

Z powodu konieczności pobierania czasu systemowego przerwania procesora pozostały nie wyłączone. Wpływa to na wyniki pomiarów. Liczbę cykli wykonania testów obliczono korzystając z mierzonych czasów wykonania testów oraz z danej częstotliwości pracy procesora 133MHz.

a) zaburzony porządek łączenia w pary bez "wyczyszczenia" potoków

b) rozkazy poprawnie łączą się w pary po "wyczyszczeniu" potoków

p - ostatni rozkaz przed pierwszym rozkazem badanego testu
np - rozkaz typu NP "czyszczący" potoki
1,2,3,4 - kolejne rozkazy badanego testu

Rys. 6. Cel czyszczenia potoków procesora Pentium
Fig. 6. The aim of the pipes clearing in Pentium processor

6. Opis badań

W celu porównania poniżej został przedstawiony ten sam test dla obu kompilatorów.

6.1. Kompilator Watcom 10.0b

Pozytywne, a nawet bardzo dobre rezultaty dało testowanie kompilatora Watcom 10.0b.

Wśród szeregu opcji kompilatora Watcom dotyczących optymalizacji zostały przetestowane następujące opcje:

- 5 - optymalizacja dla Pentium,
- od - kompilacja bez optymalizacji,
- **or - zmiana kolejności instrukcji w celu lepszego wykorzystania potoku,**
- ot - optymalizacja ze względu na czas wykonania.

Kod programu skompilowanego z opcją –od zawierał złożone rozkazy korzystające z pamięci. Wykonują się one w ciągu kilku taktów i należą do grupy rozkazów, których nie można przetworzyć równolegle w procesorze Pentium. Skompilowanie programu z opcją –5 dało większą liczbę rozkazów prostszych, jednotaktowych, działających na rejestrach. Były to rozkazy, które mogą być wykonane równolegle, ale nie zostały one uszeregowane. Program skompilowany z opcją –ot zawierał taki sam kod, jak program z opcją –5.

Dopiero skompilowanie z opcją –or dało oczekiwane rezultaty. Kody programów z opcjami –or i –5 zawierają identyczne lub prawie identyczne rozkazy, z tym, że w programie z opcją –or są one uszeregowane.

Aby ocenić jakość szeregowania, jakiego dokonuje kompilator Watcom dla procesora Pentium, przedstawione zostaną dwa programy: z opcją –5 oraz –or.

Poszczególne testy zostały oznaczone w następujący sposób:

- test5p - pusta pętla, kompilacja z opcją –5,
- testrp - pusta pętla, kompilacja z opcją –or,
- test5 - test pierwszy, kompilacja z opcją –5,
- testr - test pierwszy, kompilacja z opcją –or.

6.1.1. Pomiary dla pustej pętli

W tabeli 1 przedstawiono pomiary czasu wykonania programu testowego zawierającego pętlę for i..., wykonującą się 10 milionów razy (każda próba jest średnią z 10 wykonań programu). W pętli zostały umieszczone dwa rozkazy „not eax", czyszczące potok.

Tabela 1
Pomiar czasu wykonania testów z pustą pętlą
(w sekundach)

Próby	Test5p.exe	Testrp.exe
1 próba	0,341	0,33
2 próba	0,341	0,335
3 próba	0,335	0,329
Średnio:	0,339	0,331

6.1.2. Przykładowy test

Poniżej został przedstawiony jeden z kilku testów wykonanych na kompilatorze Watcom. Każdy test został przedstawiony za pomocą grafu ograniczeń kolejnościowych i uszeregowany przy użyciu diagramu Gantta.

Liczba cykli wykonania rozkazu oraz grupa, do której należy rozkaz, znajdują się w tabeli 3.

6.1.2.1. Postać źródłowa

Postać źródłowa programów „testr.cpp", „test5.cpp":

```
a-=2;
a|=0x11111111;
a^=0x22222222;
a&=0x44444444;
b+=4;
b&=0x22222222;
b^=0x11111111;
b|=0x44444444;
```

6.1.2.2. Pomiar czasu wykonania

Tabela 2

Pomiar czasu wykonania testów (w sekundach)

	Test5.exe	Testr.exe
1 próba	0,923	0,67
2 próba	0,923	0,67
3 próba	0,923	0,67
Średnio:	0,923	0,67

Po odjęciu czasu wykonania samej pętli:	0,584	0,339
Czas wykonania ciągu rozkazów jeden raz:	$0,584*10^{-7}$	$0,339*10^{-7}$
Liczba cykli dla jednego wykonania ciągu rozkazów:	7,8	4,5

6.1.2.3. Analiza kodów programów

Podczas badań okazało się, że rozkazy programu „testr.exe" są identyczne lub prawie identyczne z rozkazami programu „test5.exe". „Testr.exe" zawiera dodatkowy rozkaz inkrementacji pętli, który w programie „test5.exe" był umieszczony na zewnątrz instrukcji „not eax" rozpoczynających i kończących test.

6.1.2.3.1. Analiza programu „test5.exe" po disasemblacji

Kod programu „test5.exe".

```
0  sub   esi, 00000002H      - 1 UV
1  or    esi, 11111111H      - 1 UV
2  xor   esi, 22222222H      - 1 UV
3  and   esi, 44444444H      - 1 UV
4  add   ecx, 00000004H      - 1 UV
5  and   ecx, 22222222H      - 1 UV
6  xor   ecx, 11111111H      - 1 UV
7  or    ecx, 44444444H      - 1 UV
```

Zależności między danymi i uszeregowanie dokonane przez kompilator są przedstawione na rysunku 7.

6.1.2.3.2. Analiza programu „testr.exe" po disasemblacji

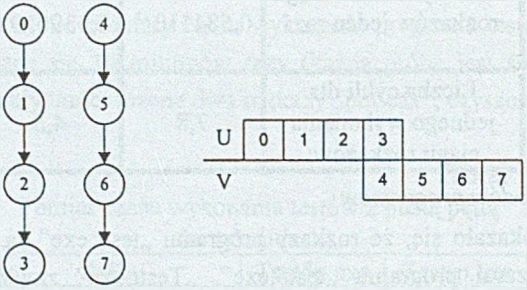
Kod programu „testr.exe" po disasemblacji.

```
0.  sub   esi, 00000002H      - 1 UV
1.  add   ecx, 00000004H      - 1 UV
```

2. inc ebp - 1 UV
3. or esi, 11111111H - 1 UV
4 and ecx, 22222222H - 1 UV
5. xor esi, 22222222H - 1 UV
6. xor ecx, 11111111H - 1 UV
7 and esi, 44444444H - 1 UV
8 or ecx, 44444444H - 1 UV

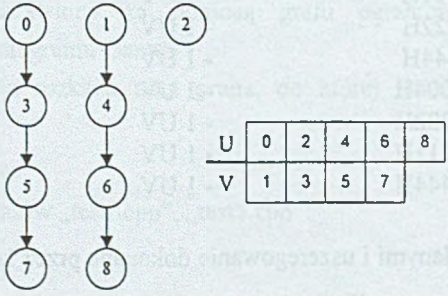
Według obliczeń teoretycznych test byłby wykonany w 7 cyklach. Wynik testowania prawie pokrywa się z obliczeniami teoretycznymi.

Zależności między danymi i uszeregowanie dokonane przez kompilator są przedstawione na rysunku 8.



Rys. 7. Digraf zależności między danymi i uszeregowanie dokonane przez kompilator dla testów z opcją –5

Fig. 7. Digraph of data dependences and scheduling made by compiler for the tests with option –5



Rys. 8. Digraf zależności między danymi i uszeregowanie dokonane przez kompilator dla testu z opcją –or

Fig. 8. Digraph of data dependences and scheduling made by compiler for the test with option –or

Kompilator zmienił kolejność rozkazów tak, że wpłynęło to pozytywnie na czas wykonania testu. Porównując diagramy Gantta dla obu testów (rys. 7 i 8) można wywnioskować, że dla programu kompilowanego z opcją –or możliwie najwięcej rozkazów zostało wykonanych równolegle, tzn. kompilator znalazł optymalne uszeregowanie.

Rozkazy w programie „testr.exe" zostały uszeregowane w następujący sposób (według numeracji instrukcji kodu programu „test5.exe"): 0,4,inkr.pętli,1,5,2,6,3,7.

Przy takim uszeregowaniu teoretycznie wykonanie powinno zająć 5 taktów, łącznie z rozkazem inkrementacji pętli. Wynikiem jest liczba cykli po odjęciu inkrementacji pętli, więc wynik powinien wynosić 4 taktów. Według pomiarów wynik wynosi 4,5.

6.2. Kompilator Borland C for Windows 5.0

Najpierw był badany kompilator Borland C for Windows 4.52. Nie zmieniał on jednak kolejności rozkazów. W związku z tym przetestowano nowszą wersję: kompilator Borland C for Windows 5.0. Wśród opcji dotyczących optymalizacji zostały przetestowane następujące opcje:

- uniemożliwienie wszelkiej optymalizacji
- szeregowanie instrukcji dla procesora Pentium.

Programy testowe zostały nazwane następująco:

- tests – kompilacja z opcją „szeregowanie instrukcji dla procesora Pentium",
- testd – kompilacja z opcją „uniemożliwienie wszelkiej optymalizacji",
- testsp – pusta pętla, kompilacja z opcją szeregowania,
- testdp – pusta pętla, kompilacja z opcją bez optymalizacji.

Dla obu testowanych programów rozkazy były identyczne. „Tests" różnił się jednak od programu „testd" tym, że zawierał rozkazy uszeregowane.

6.2.1. Pomiary dla pustej pętli

Podobnie jak poprzednio, w pętli zostały umieszczone dwa rozkazy „not eax", czyszczące potok.

Tabela 1

Pomiar czasu wykonania testów z pustą pętlą
(w sekundach)

Próby	Testdp.exe	Testsp.exe
1 próba	0,34	0,34
2 próba	0,34	0,35
3 próba	0,34	0,34
Średnio:	0,34	0,34

6.2.2. *Przykładowy test*

6.2.2.1. *Pomiar czasu wykonania*

Tabela 2

Pomiar czasu wykonania testów (w sekundach)

	Testd.exe	Tests.exe
1 próba	0,836	0,68
2 próba	0,92	0,68
3 próba	0,93	0,68
Średnio:	0,9	0,68

Po odjęciu czasu wykonania samej pętli:	0,56	0,34
Czas wykonania ciągu rozkazów jeden raz:	$0{,}56*10^{-7}$	$0{,}34*10^{-7}$
Liczba cykli dla jednego wykonania ciągu rozkazów:	7,4	4,5

6.2.2.2. *Analiza kodów programów*

6.2.2.2.1. **Analiza programu „testd.exe" po disasemblacji**

Kod programu „testd.exe".

```
0  sub  ebx, 00000002   - 1 UV
1  or   ebx, 11111111   - 1 UV
2  xor  ebx, 11111111   - 1 UV
3  and  ebx, 44444444   - 1 UV
4  add  esi, 00000004   - 1 UV
5  and  esi, 22222222   - 1 UV
6  xor  esi, 11111111   - 1 UV
7  or   esi, 44444444   - 1 UV
```

Zależności między danymi i uszeregowanie dokonane przez kompilator są przedstawione na rysunku 9.

Według obliczeń teoretycznych, test byłby wykonany w 7 cyklach. Wynik ten prawie pokrywa się z wynikiem pomiarów.

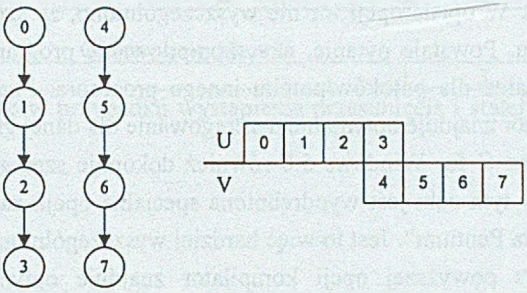
6.2.2.2.2. **Analiza programu „tests.exe" po disasemblacji**

Kod programu „tests.exe" po disasemblacji.

```
0  sub  ebx, 00000002   - 1 UV
```

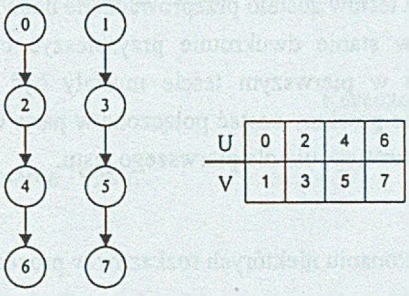
```
1  add   esi, 00000004   - 1 UV
2  or    ebx, 11111111   - 1 UV
3  and   esi, 22222222   - 1 UV
4  xor   ebx, 11111111   - 1 UV
5  xor   esi, 11111111   - 1 UV
6  and   ebx, 44444444   - 1 UV
7  or    esi, 44444444   - 1 UV
```

Zależności między danymi i uszeregowanie dokonane przez kompilator są przedstawione na rysunku 10.



Rys. 9. Digraf zależności między danymi i uszeregowanie dokonane przez kompilator dla testów z opcją „uniemożliwienie optymalizacji"

Fig. 9. Digraph of data dependences and scheduling made by compiler for the tests with option „disable all optimization"



Rys. 10. Digraf zależności między danymi i uszeregowanie dokonane przez kompilator dla testu z opcją „szeregowanie rozkazów dla procesora Pentium"

Fig. 10. Digraph of data dependences and scheduling made by compiler for the test with option „Pentium instruction scheduling"

Porównując rysunki 9 i 10 dla obu testów można wywnioskować, że szeregowanie jest wykonane optymalnie.

Rozkazy w programie „tests.exe" zostały uszeregowane w następujący sposób (według numeracji instrukcji kodu programu „testd.exe"): 0,4,1,5,2,6,3,7.

Wykonanie teoretycznie powinno zająć 4 takty. Według pomiarów wynik wynosi 4,5.

7. Wnioski

Okazało się, że dla kompilatora Watcom 10.0b istnieje opcja wykonująca szeregowanie rozkazów tak, że dodatkowo przyspiesza to czas wykonania dla procesora Pentium w stosunku do programu z opcją –5. Sądząc z opisu opcja ta wpływa na optymalne wykorzystanie potoku. W opisie opcji -or nie wyszczególniono, że szereguje ona specjalnie dla procesora Pentium. Powstaje pytanie, czy skompilowanie programu z tą opcją dałoby uszeregowanie optymalne dla potoków/potoku innego procesora. Przeprowadzone badania wykazały, że kompilator znajduje optymalne uszeregowanie dla danej grupy rozkazów.

Kompilator Borland C for Windows 5.0 również dokonuje szeregowania rozkazów dla procesora Pentium. W tym celu jest wyodrębniona specjalna opcja nazwana „szeregowanie rozkazów dla procesora Pentium". Jest to więc bardziej wyszczególnione, niż w kompilatorze Watcom. Przy użyciu powyższej opcji kompilator znajduje optymalne uszeregowanie, podobnie jak Watcom. Czas wykonania testów również jest szybszy dla testów z uszeregowanymi rozkazami.

Jakość szeregowania obu kompilatorów jest podobna w zakresie, w jakim były one badane. Oba kompilatory szeregują optymalnie.

Czasy wykonania programów są trochę większe, niż w obliczeniach teoretycznych. Przyczyną mogą być nie wyłączone przerwania podczas przeprowadzania testów.

Przy okazji wykonywania testów zostało przeprowadzone doświadczenie, które wykazało, że procesor Pentium jest w stanie dwukrotnie przyspieszyć czas wykonania. Dobrano odpowiednie rozkazy, które w pierwszym teście musiały być wykonane sekwencyjnie, natomiast w drugim wszystkie powinny zostać połączone w pary. Czas wykonania drugiego testu był dokładnie dwa razy krótszy, niż dla pierwszego testu.

Tabela 3

Informacje o wykonaniu niektórych rozkazów w procesorze Pentium

Symbol rozkazu	Opis rozkazu	Liczba taktów	Grupa
ADD	Dodawanie całkowite		
Add reg, imm		1	UV
add mem, imm		3	UV*
DEC	Zmniejszenie o 1		
dec mem		3	UV

cd. tabeli 3

INC	Zwiększenie o 1		
inc r8		1	UV
inc r16		1	UV
inc r32		1	UV
inc mem		3	UV

LEA	Umieszczenie w rejestrze adresu efektywnego		
lea r16, mem		1	UV
lea r32, mem		1	UV

MOV	Przesłanie danych		
mov mem, reg		1	UV
mov reg, mem		1	UV

SUB	Odejmowanie całkowite		
Sub mem, imm		3	UV*

* - oznacza, że w przypadku wystąpienia przesunięcia i stałej rozkaz nie może być łączony w pary.

LITERATURA

1. Kozielski S., Szczerbiński Z.: Komputery równoległe. Architektura, elementy oprogramowania. WNT, Warszawa 1993.
2. Optimization's For Intel's 32-Bit Processors - Intel, October 1995.
3. Coffman E.G.: Teoria szeregowania zadań. WNT, Warszawa 1980.
4. Schmit M.L.: Pentium ™ Processor Optimization Tools, USA 1995.
5. Lipski W.: Kombinatoryka dla programistów. ISBN, Warszawa 1989.

Recenzent: Dr inż. Maciej Bargielski

Wpłynęło do Redakcji 26 marca 1998 r.

Abstract

The main problem of different processors is the instruction processing speed. One of the main method of processor speed improving is using a few undependence units, which can work parallel (superscalar structures).

Example of processors with more number of processing units is Pentium processor.

The article presents investigations of the C/C++ compilers in respect of output code optimization for Pentium processor as one of the superscalar processors. Time of the

instruction processing in Pentium pipes is measured. This is made for two C compilers: Borland C for Windows (version 5.0) and Watcom (version 10.0b). Quality of compilers optimization in respect of instruction scheduling is investigated.

The way of execution of fixed-positioned instructions in two pipes of Pentium processor is shown on the fig. 1. The way of parallel performance of Pentium is shown on fig. 2. Data dependences are presented as digraph on the fig. 3. Scheduling is shown as Gantt charts (fig. 4). There are kinds of data dependences presented on fig. 5. On the fig. 6 the reason of using of pipes clearing during the investigations is shown.

Chapter 6 describes investigations. There is described one of the tests in it. Teoretical calculations of the time of performance are compared with practical effects. Table 1 and table 2 contain the time of test performance. Fig. 7-10 show data dependences and scheduling for the tests.

Table 3 consists description of number of cycles and group of Pentium instructions.