Agnieszka KAMIŃSKA
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY, WYDZIAŁ INFORMATYKI,
ul. Żołnierska 49, 71-210 Szczecin

Obliczeniowe szacowanie czasu wykonania programu

Mgr inż. Agnieszka KAMIŃSKA

Mgr inż. Agnieszka Kamińska jest doktorantką w Katedrze Inżynierii Oprogramowania na Wydziale Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego w Szczecinie. Do jej zainteresowań badawczych należą: optymalizacja kompilacji, lokalność danych, przetwarzanie równoległe.

e-mail: agkaminska@wi.zut.edu.pl

Streszczenie

Określenie czasu wykonywania programu poprzez jego uruchomienie nie zawsze jest możliwe w zagadnieniach praktycznych, przykładowo w kompilacji iteracyjnej, ze względu na duże wydłużenie czasu tworzenia oprogramowania. Jednakże w wielu sytuacjach nie ma potrzeby dokładnego określenia tego czasu; wystarczyłoby go oszacować. W niniejszym artykule przedstawiono propozycję sposobu obliczeniowego szacowania czasu wykonania programu w oparciu o samą postać jego kodu źródłowego i znane parametry środowiska sprzętowego.

Słowa kluczowe: lokalność danych, pamięć podręczna, blokowanie.

Estimation of program execution time

Abstract

The program execution time is one of criteria which are taken into account during assessment of widely comprehended software quality. The general purpose is to make program execution time as short as possible. The program execution time depends on many, very different, factors. The most obvious of these are: the form of its source code and the hardware environment in which the program is executed. In practice, even a very minor change in the form of the source code of a program can result in a significant change in its execution time. The same effect can be caused by a slight change in the values of hardware parameters. Although the interpretation of program execution time as a quality assessment criterion is very simple, it is sometimes very difficult to precisely measure program execution and taking necessary measurements requires running the program. However, there is very often no need to know this time precisely; it would be sufficient to estimate it with some error which is known in advance. The paper presents – using the matrix multiplication problem for reference – a proposal of a method which can be used for estimating the execution time of a program, based only on its source code and a priori known hardware parameters. The idea of the proposed method is to elaborate a mathematical model combining statistical approach and the Wolfe's method for calculating data locality. The paper discusses the results of using the elaborated model on a control sample and indicates directions of further works.

Keywords: data locality, cache memory, tiling.

1. Wstęp

Czas wykonania programu jest jednym z kryteriów branych pod uwagę przy ocenie szeroko rozumianej jakości oprogramowania. Ogólnym dążeniem jest, by czas ten był możliwie najkrótszy.

Czas wykonania programu jest zależny od wielu, bardzo różnorodnych, czynników; jednymi z najbardziej oczywistych są: postać kodu źródłowego programu oraz środowisko sprzętowe, w którym uruchomiono program. W praktyce, nawet drobna zmiana postaci kodu źródłowego programu rozwiązującego dany problem może pociągać za sobą znaczącą zmianę czasu wykonania tego programu. Podobne efekty może przynieść niewielka zmiana wartości parametrów sprzętowych.

Chociaż czas wykonania programu jest kryterium o bardzo prostej interpretacji; to zmierzenie tego czasu bywa bardzo trudne i z definicji wymaga uruchomienia programu. Jednakże w wielu sytuacjach nie ma potrzeby dokładnego określenia czasu wykonania programu w sposób empiryczny; wystarczyłoby go oszacować z pewnym, z góry znanym błędem.

Celem niniejszego artykułu jest przedstawienie, na przykładzie mnożenia macierzy, propozycji sposobu obliczeniowego szacowania czasu wykonania programu w oparciu o samą postać jego kodu źródłowego i znane a priori, wybrane parametry środowiska sprzętowego.

2. Założenia do konstrukcji modelu

Punktem wyjścia do obliczeniowego oszacowania czasu wykonania programu jest utworzenie modelu matematycznego opisującego zależność między czasem wykonania programu a postacią jego kodu źródłowego oraz środowiskiem wykonania programu.

Ze względu na występowanie znaczącej dysproporcji pomiędzy szybkością procesorów a czasem dostępu do podsystemu pamięci [1], to właśnie pamięć - a zwłaszcza szybko dostępna pamięć podręczna (ang. *cache*) procesora – jest tym elementem środowiska sprzętowego, który determinuje czas wykonania programu. Istotny jest tu zarówno rozmiar dostępnej pamięci podręcznej, jak i sposób jej wykorzystania. W idealnej sytuacji, dane znajdujące się w pamięci podręcznej cechują się pełną lokalnością, tzn. wszystkie dane potrzebne procesorowi podczas wykonywania programu są dostępne w pamięci podręcznej natychmiast po zgłoszeniu na nie zapotrzebowania, a nie dopiero wówczas pobierane z pamięci głównej do pamięci podręcznej. Z kolei rozmieszczenie danych w pamięci podręcznej podczas wykonywania programu jest uzależnione od postaci jego kodu źródłowego. Innymi istotnymi czynnikami, wpływającymi na czas wykonania programu a zarazem wiążącymi postać jego kodu źródłowego ze środowiskiem wykonania są: łączny rozmiar wszystkich danych przetwarzanych w programie oraz liczba operacji wykonywanych w programie.

Formalizując powyższe rozważania, można zapisać, że czas wykonania programu Y (mierzony liczbą cykli procesora) jest zależny od:

- lokalności danych (*X1*), to jest stopnia, w jakim pamięć podręczna jest wypełniona danymi przetwarzanymi w programie,
- łącznego rozmiaru danych przetwarzanych w programie, rozpatrywanego w odniesieniu do rozmiaru pamięci podręcznej (*X2*),
- liczby wykonywanych operacji (*X3*), tzn.:

$$Y = f(X1, X2, X3) \tag{1}$$

Ponieważ, zgodnie z założeniami przedstawionymi we wstępie niniejszego artykułu, model (1) ma umożliwiać oszacowanie czasu wykonania programu w oparciu o samą postać jego kodu źródłowego i znane z góry, wybrane parametry środowiska sprzętowego – wartości zmiennych *X1*, *X2*, *X3* należy wyznaczyć wykorzystując wyłącznie kod źródłowy programu i informacje o środowisku jego wykonania.

Do wyznaczenia wartości zmiennej *X1* została wykorzystana metoda szacowania lokalności danych zaproponowana przez Wolfe'a [2]. Metoda Wolfe'a ma zastosowanie do pętli programowych – czyli tych fragmentów kodu programu, w których realizowane jest przetwarzanie znacznych ilości danych i które w związku z tym determinują czas wykonania programu. Metoda ta pozwala wyznaczyć tzw. zbiorczy odcisk danych, który określa minimalną pojemność pamięci podręcznej (wyrażoną liczbą linii pamięci podręcznej lub, po odpowiednim przeliczeniu, liczbą bajtów) niezbędną, aby pomieścić wszystkie dane przetwarzane w rozpatrywanej pętli.

Wartość zmiennej *X1* to:

$$X1 = \frac{odcisk\_danych\_wg\_Wolfe'a}{rozmiar\_pamieci\_podrecznej} * 100\% \quad (2)$$

X2 to wskaźnik obliczany następująco:

$$X2 = \frac{rozmiar\_danych\_w\_programie}{rozmiar\_pamieci\_podrecznej} \quad (3)$$

Liczbę wykonywanych operacji (*X3*) można, dla uproszczenia, utożsamić z łączną liczbą wszystkich iteracji pętli, w których jest realizowane przetwarzanie danych.

Biorąc pod uwagę opisane powyżej powiązania między zmiennymi *Y* oraz *X1*, *X2*, *X3* w modelu (1) można przyjąć za strukturę modelu (f) iloczyn funkcji potęgowych, sprowadzając go do postaci:

$$Y = a * X1^{p1} * X2^{p2} * X3^{p3} \quad (4)$$

gdzie: a, p1, p2, p3 to parametry o nieznanych wartościach.

Wartości parametrów a, p1, p2, p3 zostały obliczone dla konkretnego przykładu – mnożenia macierzy, poprzez zastosowanie metody najmniejszych kwadratów dla odpowiednich danych empirycznych, zebranych w ramach przeprowadzonych w tym celu pomiarów.

3. Badania eksperymentalne

Badania eksperymentalne przeprowadzono dla mnożenia macierzy.

Kod programu, realizującego mnożenie macierzy kwadratowych, jest postaci:

```
float D[N][N], E[N][N], F[N][N];
(...)
for(i=0; i<N; i++){
  for(j=0; j<N; j++){
   for(k=0; k<N; k++){
    D[i][j] = D[i][j] + E[i][k] * F[k][j];
   } //enfor k
  } //endfor j
} //enfor i
```

Kod ten poddano transformacji znanej jako blokowanie (ang. *tiling*), przekształcając go do następującej postaci:

```
float D[N][N], E[N][N], F[N][N];
(...)
  for(it=0; it<N; it=it+BLOCK_SIZE){
   for(jt=0; jt<N; jt=jt+BLOCK_SIZE){
    for(kt=0; kt<N; kt=kt+BLOCK_SIZE){
     for(i=it; i<min(it+BLOCK_SIZE,N); i++){
      for(j=jt; j<min(jt+BLOCK_SIZE,N); j++){
       for(k=kt; k<min(kt+BLOCK_SIZE,N); k++){
        D[i][j] = D[i][j] + E[i][k] * F[k][j];
       } //endfor k
      } //endfor j
     } //endfor i
    } //enfor kt
   } //endfor jt
  } //enfor it
```

Wykonanie blokowania było niezbędne dla uzyskania różnych – zależnych od przyjętej wartości *BLOCK_SIZE* – semantycznie równoważnych postaci wyjściowego kodu źródłowego, pociągających za sobą inną lokalność danych.

Dla każdego rozmiaru bloku *BLOCK_SIZE* i dla każdego analizowanego rozmiaru macierzy *N* (*N* = 100, 200, 300, 400, 500, 600) przeprowadzono po 5 niezależnych pomiarów czasu wykonania programu z blokowaniem, w środowisku testowym o parametrach przedstawionych w tabeli 1. W ramach każdego z przeprowadzonych pomiarów, kod z blokowaniem był wykonywany *p* razy, przy czym wartość *p* była dobierana empirycznie tak, by łączny czas wykonania *p* powtórzeń wynosił około 1,5 do 2 minut (od *p*=8 000 dla *N*=100 do *p*=70 dla *N*=600). Dzięki temu podejściu, wartości uzyskane z pomiarów można uznać za prawdziwe wartości mierzonych wielkości.

Tab. 1. Charakterystyka środowiska testowego
Tab. 1. Parameters of the test environment

Procesor	Intel Core 2 Quad Q6600
Liczba rdzeni / wątków	4 / 4
Pamięć podręczna L1 Data	4 x 32 KB, 8-way set associative, 64-byte line size
System operacyjny	Linux Ubuntu 9.04 - Jaunty Jackalope
Kompilator	gcc (Ubuntu 4.3.3-5ubuntu4) 4.3.3
Optymalizacja kompilacji	Wyłączona (kompilacja z opcją –O0)

Otrzymane wyniki zostały uśrednione (obliczono średnią dla każdego rozmiaru macierzy i rozmiaru bloku, z 5 przeprowadzonych pomiarów). Dodatkowo, dokonano oszacowania odcisku danych, zgodnie z metodą Wolfe'a. Ze względu na brak ogólnie dostępnego oprogramowania szacującego lokalność danych wg metody Wolfe'a, do wyznaczenia szacunkowego odcisku danych wykorzystano autorski moduł szacujący lokalność danych wg ww. metody [3].

Tab. 2. Wyniki pomiarów
Tab. 2. Test results

N	*BLOCK_SIZE*	Liczba cykli CPU	Odcisk danych [B]
100	16	11 935,50	3 072
100	26	11 657,50	8 112
100	32	11 718,25	12 288
100	37	11 594,75	16 428
100	45	11 676,50	24 300
100	48	11 708,25	27 648
100	52	11 475,50	32 448
200	16	73 212,00	3 072
200	26	71 698,00	8 112
200	32	71 576,00	12 288
200	37	71 316,00	16 428
200	45	70 982,00	24 300
200	48	71 316,00	27 648
200	52	70 560,00	32 448
300	16	233 435,00	3 072
300	26	229 095,00	8 112
300	32	227 900,00	12 288
300	37	227 835,00	16 428
300	45	225 950,00	24 300
300	48	226 510,00	27 648
300	52	184 841,00	32 448
400	16	543 266,67	3 072
400	26	532 306,67	8 112
400	32	529 426,67	12 288
400	37	526 920,00	16 428
400	45	524 720,00	24 300
400	48	525 826,67	27 648
400	52	523 946,67	32 448
500	16	1 055 050,00	3 072
500	26	1 032 625,00	8 112
500	32	1 026 250,00	12 288
500	37	1 023 300,00	16 428
500	45	1 014 050,00	24 300
500	48	1 013 550,00	27 648
500	52	1 011 225,00	32 448
600	16	1 804 571,43	3 072
600	26	1 767 171,43	8 112
600	32	1 757 257,14	12 288
600	37	1 755 485,71	16 428
600	45	1 743 771,43	24 300
600	48	1 742 257,14	27 648
600	52	1 739 400,00	32 448

Rozmiar bloku (*BLOCK_SIZE*) był dobrany tak, by wszystkie bloki razem wzięte nie przekraczały swoim rozmiarem dostępnej pamięci podręcznej, tzn.:

$$3 * BLOCK\_SIZE^2 \text{ [elementów]} \leq 32\,768 \text{ [B]} / 4 \text{ [B/element]}$$
$$\text{czyli } BLOCK\_SIZE \leq 52{,}256$$

Podsumowanie wyników pomiarów (średnia z 5 niezależnych pomiarów) przedstawia tabela 2.

W oparciu o wyniki pomiarów oraz przyjętą postać modelu (patrz wzór (4)) wyznaczono wartości parametrów: a, p1, p2, p3, wykorzystując w tym celu metodę najmniejszych kwadratów i operując na zlinearyzowanej postaci modelu:

$$\log Y = \log a + p1*\log X1 + p2*\log X2 + p3*\log X3 = b + p1*\log X1 + p2*\log X2 + p3*\log X3 \quad (5)$$

Obliczenia wykonano zakładając, że wyraz wolny (b) w modelu zlinearyzowanym może być zarówno zerowy, jak i niezerowy. Wartości zmiennej *X1* wyznaczono korzystając z wzoru (2). Wartości zmiennej *X2* wyznaczono korzystając z wzoru (3); jako rozmiar danych w programie przyjęto:

N^2[elementów macierzy]*3 [liczba macierzy]*4 [B/element macierzy]

Jako wartości zmiennej *X3* przyjęto N^3. Uzyskane wyniki przedstawiono w tabeli 3.

Tab. 3. Modele statystyczne, uszeregowane wg rosnącej wartości r^2
Tab. 3. Statistical models sorted by the ascending value of r^2

Zmienne modelu zlinearyzowanego	Parametry modelu zlinearyzowanego	r^2 i $s_e y$ modelu	Błąd procentowy oceny *y* w modelu potęgowym [%]
Y, X1, X2, X3, b	p1=-0,021118767 p2=0 p3=0,935443062 b=-1,54007071	r^2 = 0,998704954 $s_e y$ = 0,027551693	6,5496
Y, X1, X2, X3	p1=-0,021118767 p2=0,448181109 p3=0,636655656 b=0	r^2 = 0,999976049 $s_e y$ = 0,027551693	6,5496

Na podstawie uzyskanych ocen (r^2, $s_e y$) modeli zlinearyzowanych widać, że charakteryzują się one bardzo wysokim dopasowaniem (r^2 powyżej 0,998), przy czym większą wartość r^2 osiągnięto dla modelu, w którym b = 0.

Wartości $s_e y$ są identyczne dla obu modeli i bardzo niewielkie (około 0,028); także po przejściu do postaci potęgowej błędy procentowe oceny y są niewielkie (około 6,55 %).

Ostatecznie, na potrzeby dalszej analizy przyjęto model o wyższej wartości r^2 – czyli model, w którym po przejściu do równoważnego modelu potęgowego występują zmienne: *X1*, *X2*, *X3*, zaś stała a ma wartość 1.

Parametry tego modelu to:

$$p1 = -0{,}021118767$$
$$p2 = 0{,}448181109$$
$$p3 = 0{,}636655656$$

Model w postaci zlinearyzowanej:

$$\log Y = -0{,}021118767* \log X1 + 0{,}448181109* \log X2 + 0{,}636655656* \log X3 \quad (6)$$

przy czym:

$$r^2 = 0{,}999976049$$
$$s_e y = 0{,}027551693$$

Model w postaci potęgowej:

$$Y = X1^{-0{,}021118767} * X2^{0{,}448181109} * X3^{0{,}636655656} \quad (7)$$

Błąd procentowy oceny *y* dla modelu potęgowego wynosi 6,5496 %.

Następnie, zweryfikowano model (7) wykorzystując w tym celu czasy zmierzone dla macierzy kontrolnej o rozmiarze *N*=(100 + 600)/2 = 350, która nie była wykorzystana do wyznaczenia wartości parametrów modelu. Wartości zmiennej *X1* wyznaczono korzystając z wzoru (2). Wartości zmiennej *X2* wyznaczono korzystając z wzoru (3); jako rozmiar danych w programie przyjęto:

N^2[elementów macierzy]*3[liczba macierzy]*4[B/element macierzy]

Jako wartości zmiennej *X3* przyjęto N^3. Uzyskane wyniki przedstawiono w tabeli 4.

Tab. 4. Weryfikacja uzyskanego modelu dla *N*=350
Tab. 4. Verification of the elaborated model for *N*=350

BLOCK_SIZE	*CPU*= liczba cykli CPU	Odcisk danych [B]	*Y* wg modelu (7)	Błąd modelu [%] (*Y-CPU*)/*CPU**100%
16	367 585,45	3 072	379 223,46	3,1660676
26	360 567,27	8 112	371 526,05	3,0393152
32	358 269,09	12 288	368 281,95	2,7947864
37	358 349,09	16 428	366 030,50	2,1435559
45	356 334,55	24 300	363 016,72	1,8752543
48	357 054,55	27 648	362 028,51	1,3930538
52	356 007,27	32 448	360 806,63	1,3481053

Wyniki przedstawione w Tabeli 4 pozytywnie weryfikują poprawność modelu (7). Dla macierzy kontrolnej o wymiarze *N*=350, różnica względna pomiędzy faktycznym czasem wykonania programu a czasem teoretycznym wynikającym z przyjętego modelu nie przekraczała 4%.

4. Wnioski

Uzyskane wyniki wskazują na możliwość i zasadność obliczeniowego szacowania czasu wykonania programu w oparciu o modele statystyczne.

W niniejszym artykule przedstawiono model statystyczny dla bardzo prostego przykładu – mnożenia macierzy kwadratowych. W przyszłych pracach planowane jest wyznaczenie pewnych charakterystycznych klas postaci kodu źródłowego, a następnie stworzenie dla nich bardziej uogólnionych modeli statystycznych do obliczeniowego szacowania czasu wykonania programu.

5. Literatura

[1] Stallings W.: Organizacja i architektura systemu komputerowego. Projektowanie systemu a jego wydajność, Wydawnictwa Naukowo-Techniczne, 2004.

[2] Wolfe M.: High Performance Compilers for Parallel Computing, Addison Wesley, 1996.

[3] Kraska K., Kamińska A.: Koncepcja metody zwiększania lokalności danych na poziomie pamięci podręcznej oparta na transformacjach pętli programowych. Metody Informatyki Stosowanej, Nr 2/2010, Tom 23, s. 63-72.

otrzymano / received: 03.11.2011
przyjęto do druku / accepted: 03.01.2012 *artykuł recenzowany / revised paper*