Maciej POLIWODA
POLITECHNIKA SZCZECIŃSKA, WYDZIAŁ INFORMATYKI

Automatyczne zrównoleglenie pętli, efektywność zrównoleglonego kodu

Dr inż. Maciej POLIWODA

Od 1998 r zatrudniony na Wydziale Informatyki Politechniki Szczecińskiej. W roku 2003 otrzymał stopień naukowy doktora. Jego praca doktorska była zatytułowana „Formalizacja, implementacja i weryfikacja metody hiperpłaszczyzn dla automatyzacji procesu zrównoleglenia pętli programowych".

e-mail: mpoliwoda@wi.ps.pl

Streszczenie

Artykuł przedstawia wyniki badań efektywności kodu pętli zrównoleglonego metodą hiperpłaszczyzn w odniesieniu do kodu pętli zrównoleglonego innymi metodami. Celem przeprowadzonych badań było określenie efektywności kodu zrównoleglonego różnymi metodami oraz obszaru, w jakim zrównoleglony kod efektywnie wykorzystuje zasoby systemu wieloprocesorowego z uwzględnieniem procesorów wielordzeniowych.

Słowa kluczowe: zrównoleglenie pętli, OpenMP, metoda hiperpłaszczyzn.

Automatically loops parallelized, efficiency of parallelized code

Abstract

The results of loops code efficiency parallelized by hyperplanes method compared with loops code efficiency parallelized with other methods are presented in this paper. The main goal is to determinate when the loops parallelized by different methods are efficient and the multiprocessor system or multi core processors are utilized effectively.

Keywords: loops parallelization, OpenMP, hyperplane method.

1. Wstęp

Proces przekształcenia kodu sekwencyjnego do postaci równoległej (zrównoleglenie) jest złożony i czasochłonny. Polega on na znalezieniu w sekwencyjnym kodzie programu fragmentów, które będą mogły być wykonane przez niezależnie pracujące procesory systemu wieloprocesorowego. Głównym celem przekształcenia kodu programu sekwencyjnego do postaci równoległej jest skrócenie czasu potrzebnego do jego wykonania i maksymalne wykorzystanie możliwości oferowanych przez system wieloprocesorowy.

Aby uzyskać dużą efektywności zrównoleglenia programu, czas poświęcony na wykonanie fragmentu kodu poddanego procesowi zrównoleglenia powinien stanowić znaczną część czasu poświęconego na wykonanie kodu całego programu. Wymaganie to jest w większości przypadków spełnione przez kod pętli programowych.

Przy zrównolegleniu pętli metodą hiprpłaszczyzn wykorzystano własne narzędzie automatycznie przekształcające kod pętli do postaci równoległej. Wynikiem pracy narzędzia jest opis zależności między iteracjami za pomocą wektorów zależności, zbiór macierzy transformacji umożliwiających zrównoleglenie pętli, jedna lub kilka wersji zrównoleglonego kodu pętli po przeprowadzonych transformacjach.

W niniejszym artykule przedstawiono wyniki badań efektywności kodu zrównoleglonego różnymi metodami oraz obszaru, w jakim zrównoleglony kod efektywnie wykorzystuje zasoby systemu wieloprocesorowego z uwzględnieniem procesorów wielordzeniowych.

2. Proces zrównoleglenia pętli

Prezentowanym w niniejszym artykule badaniom została poddana pętla wykonująca operacje mnożenia macierzy rys. 1. Pierwszy krok w procesie zrównoleglenia pętli polega na wyznaczeniu zależności między iteracjami pętli. Określenie wszystkich zależnych iteracji i relacji między nimi pozwala na wyodrębnienie iteracji niezależnych i ich podział między niezależnie pracujące wątki. Metody analizy zależności między iteracjami były przedmiotem wielu prac [1, 3, 10, 12]. Iteracja $\overline{J}$ opisana wektorem $\overline{J} = [j_1,\ldots,j_n]^T$ jest zależna od iteracji $\overline{I}$ opisanej wektorem $\overline{I} = [i_1,\ldots,i_n]^T$ wówczas, gdy odwołują do tego samego obszaru pamięci, przy czym jedno z tych odwołań musi dotyczyć operacji zapisu. Przy czym iteracje $\overline{I}$ i $\overline{J}$ spełniają leksykograficzną zależność $\overline{I} \prec \overline{J}$, czyli posiadają takie współrzędne k dla $1 \le k \le n$, gdzie $i_1 = j_1,\ldots,\ i_{k-1} = j_{k-1}$ oraz $i_k < j_k$.

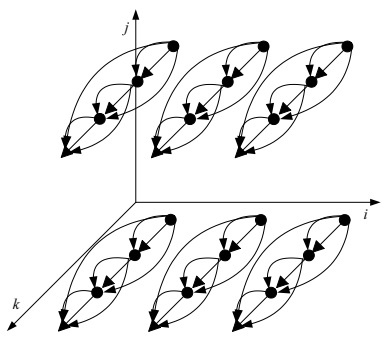
```
for(i = 0;i < s1;i++)
    for(j = 0;j < s2;j++)
        for(k = 0;k < s3;k++)
        a[i][j]+= b[i][k]*c[k][j];
```

Rys. 1. Przykładowa pętla
Fig. 1. Example loop

Zależności między iteracjami mogą być opisane przez wektory zależności, gdzie wektor zależności między iteracją $\overline{I} = [i_1,\ldots,i_n]^T$ a iteracją $\overline{J} = [j_1,\ldots,j_n]^T$ definiujemy jako

$$\overline{K} = \overline{J} - \overline{I} = [j_1 - i_1,\ldots,j_n - i_n]^T,$$

gdzie wektor jest leksykograficznie większy od zera $\overline{K} \succ \overline{0}$.



Rys. 2. Zależności w przestrzeni iteracji pętli
Fig. 2. Lloop iteration space with dependencies

Na rysunku rys. 2 zostały przedstawione zależności prosta, odwrotna i po wyjściu występujące między iteracjami pętli rys. 1, które opisuje wektor $K = [0 \quad 0 \quad 1]^T$.

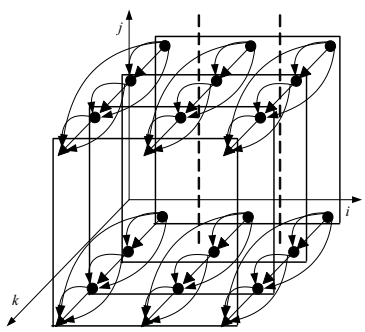
3. Metoda hiperpłaszczyzn

Wykorzystywana w zaimplementowanym narzędziu automatycznego zrównoleglenia technika znana jako „hyperplane method", „wave-front-method" lub „Lamport method" jest przedstawiona w pracach [6, 7, 11]. Technika ta polega na wyznaczeniu $m \times n$ wymiarowej macierzy C, gdzie $m \le n$. Każdy wiersz macierzy C, wektor $\overline{C}_i$ dla $i = 1,2,...,m$ opisuje i-tą hiperpłaszczyznę. Dla wszystkich wektorów zależności $\overline{K} \succ \overline{0}$ między iteracjami występującymi w m-wymiarowej pętli, macierz C musi spełniać warunek $C\overline{K} > 0$ gwarantujący zachowanie porządku leksykograficznego zależnych iteracji. Wyznaczenie macierzy C o wymiarze $m \times n$ rozpoczyna się od wyznaczenia wektora $\overline{C}_i$. Jeśli wektor $\overline{C}_i$ dla wszystkich wektorów zależności $\overline{K} \succ \overline{0}$ spełnia warunek $\overline{C}\overline{K} > 0$ wszystkie iteracje $\overline{I}$ takie, że $C\overline{I} = t$, mogą być wykonane równolegle. Jeśli wektor $\overline{C}_i$ dla wszystkich wektorów zależności $\overline{K} \succ \overline{0}$ spełnia warunek $\overline{C}\overline{K} \ge 0$, wyznaczamy kolejne wiersze macierzy C, dopóki warunek $C\overline{K} > 0$ nie jest spełniony. Dwie iteracje $\overline{I}$ i $\overline{J}$ należące do jednej hiperpłaszczyzny, spełniające warunek $C\overline{I} = C\overline{J}$, mogą być wykonane równolegle. Dla pętli rys. 1 macierz C spełniająca warunek $C\overline{K} > 0$ dla wektorów zależności rys. 2 jest następująca:

$$C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

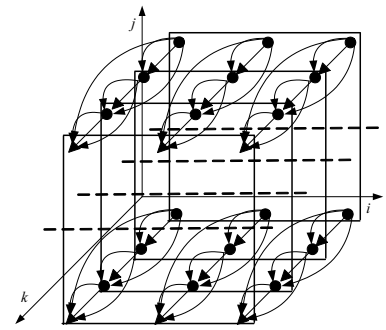
Na podstawie proponowanej macierzy opisującej hiperpłaszczyznę można wygenerować dwie wersje kodu równoległego przedstawione na rysunkach rys. 3 i rys. 4 uzyskane po wykonaniu transformacji [4, 10, 11] polegającej na zamianie miejscami pętli zewnętrznej i najbardziej wewnętrznej. W obu wersjach sekwencyjnie wykonywane są iteracje pętli zewnętrznej, iteracje pętli wewnętrznych mogą być wykonane równolegle.

Alternatywne do metody hiperpłaszczyzn rys. 3 i rys. 4 możliwości zrównoleglenia pętli rys. 1 przedstawiono na rysunkach rys. 5 i rys. 6.




```
#pragma omp parallel private(k,j) shared(i,a,b,c)
{
        for(k=0;k<size;k++){
#pragma omp for schedule(runtime)
                for(i=0;i<size;i++){
                        for(j=0;j<size;j++){
                                a[i][j]+=b[i][k]*c[k][j];
                        }
                }
        }
}
```

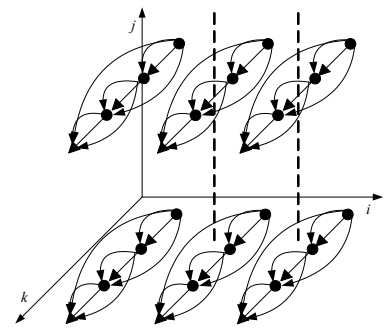
Rys. 3. Kod pętli zrównoleglonej metodą hiperpłaszczyzn (hpari)
Fig. 3. Parallel code created with hyperplane method (hpari)




```
#pragma omp parallel private(k,i) shared(j,a,b,c)
{
        for(k=0;k<size;k++){
                for(i=0;i<size;i++){
#pragma omp for schedule(runtime)
                        for(j=0;j<size;j++){
                                a[i][j]+=b[i][k]*c[k][j];
                        }
                }
        }
}
```

Rys. 4. Kod pętli zrównoleglonej metodą hiperpłaszczyzn (hparj)
Fig. 4. Parallel code created with hyperplane method (hpari)

W pętli rys. 5 zrównoleglono wykonanie niezależnych iteracji zewnętrznej pętli. Kod przedstawiony na rysunkach rys. 6 a został zrównoleglony przez podział iteracji środkowej pętli, analogicznie jak w przypadku na rys. 5 nie występują zależności między podzielonymi iteracjami. Przykład przedstawiony na rys. 6b przedstawia zrównoleglony kod wewnętrznej pętli poprzez zastosowanie redukcji.




```
#pragma omp parallel for private(j,k,sum) schedule(runtime)
        for(i=0;i<size;i++){
                for(j=0;j<size;j++){
                        sum = 0;
                        for(k=0;k<size;k++){
                                sum+=b[i][k]*c[k][j];
                        }
                        a[i][j]+=sum;
                }
        }
```

Rys. 5. Zrównoleglone iteracje pętli zewnętrznej (pari)
Fig. 5. Parallel code outer loop parallelized (pari).

a)

```
#pragma omp parallel private(i,sum) shared(j,k,a,b,c)
{
        for(i=0;i<size;i++){
#pragma omp for nowait private(k) schedule(runtime)
                for(j=0;j<size;j++){
                        sum = 0;
                        for(k=0;k<size;k++){
                                sum+=b[i][k]*c[k][j];
                        }
                        a[i][j]+=sum;
                }
        }
}
```

b)

```
#pragma omp parallel private(i,j,sum) shared(a,b,c)
{
        for(i=0;i<size;i++){
                for(j=0;j<size;j++){
                        sum = 0;
#pragma omp for nowait schedule(runtime)
                        for(k=0;k<size;k++){
                                sum+=b[i][k]*c[k][j];
                        }
#pragma omp critical
                        a[i][j]+=sum;
                }
        }
}
```

Rys. 6. Zrównoleglone iteracje pętli wewnętrznych a) (parj), b) (park)
Fig. 6. Parallel code inner loops parallelized a) (parj), b) (park)

4. Środowisko eksperymentu

W celu weryfikacji efektywności zrównoleglonego kodu rys. 3, rys. 4, rys. 5 , rys. 6 zostały przeprowadzone pomiary przyspieszenia $S(P)=\frac{T(1)}{T(P)}$, gdzie $T(1)$ - czas sekwencyjnego wykonania pętli, $T(P)$ - czas równoległego wykonania pętli, P - liczba procesorów.

Pomiary przyspieszenia przeprowadzono dla macierzy kwadratowych o wymiarach z przedziału od 100 do 5000 zwiększając wymiar o 100 oraz różnych kompilatorów [15, 16, 17, 18] w celu uwzględnienia zależności wartości przyspieszenia od wielkości problemu i użytego kompilatora.

Pomiary zostały wykonane z wykorzystaniem następującego środowiska:

Sprzęt:

2 x Intel Quad Core Xeon E5310 1,60 GHz 1U
The Intel Workstation Board S5000Xvn 1.97 GB RAM

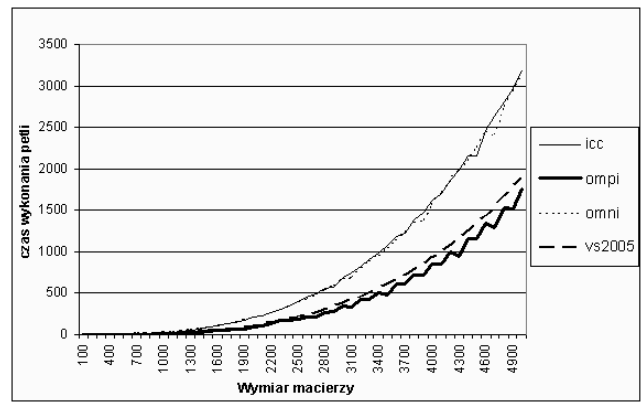
Środowisko:

Konfiguracja I
System operacyjny Linux Ubuntu
Kompilatory [15, 16]:
Omni OpenMP Compiler (omni)
OMPi OpenMP C Compiler (ompi)

Konfiguracja II
Windows XP Professional x64 Edition Ver.2003
Kompilatory [17, 18]:
Intel ® C++ Compiler for Windows* - Evaluation (icc)
Microsoft Visual Studio 2005 (vs2005)

5. Wyniki

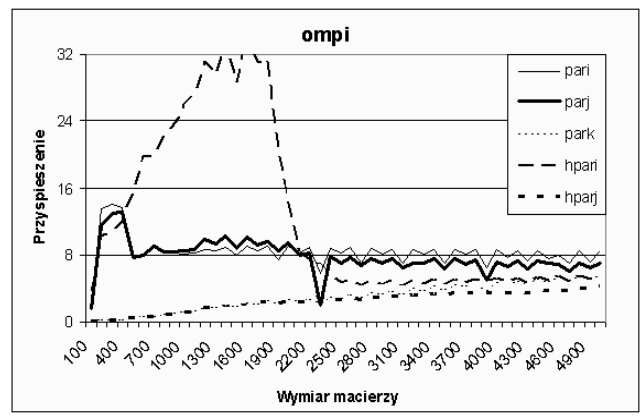
Pomiar czasu wykonania pętli rys. 1 w sposób sekwencyjny został przedstawiony na rysunku rys. 7. Najkrótszy czas wykonania osiągnął kod skompilowany przez kompilatory ompi oraz vs2005. Należy zauważyć, że mimo różnicy między systemami operacyjnymi czasy wykonania kodu skompilowanego przez vs2005 i ompi są mocno zbliżone.



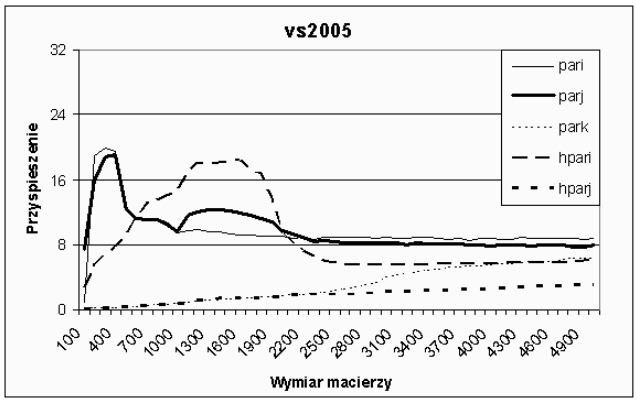
Rys. 7. Czas wykonania pętli sekwencyjnej
Fig. 7. Loop sequential execution time

Również w przypadku wykonania kodu pętli rys. 3, rys. 4, rys. 5, rys. 6 w sposób równoległy tendencja ta jest zachowana. W dalszej części zostały przedstawione w postaci wykresów rys. 8 i rys. 9 wyniki pomiarów przyspieszenia dla kodu skompilowanego przez kompilatory vs2005 i ompi.

Należy zauważyć, że dla rozmiaru macierzy od 400 do 2200 w przypadku obu kompilatorów przyspieszenie kodu równoległego rys. 3 osiąga wartości znacznie przekraczające liczbę jednostek przetwarzających. Podobną sytuację obserwujemy w przypadku kodu równoległego rys. 5 i rys. 6a, lecz dla mniejszego zakresu rozmiaru macierzy. Opisany powyżej efekt pojawia się również dla kodu kompilowanego pozostałymi kompilatorami i jest wynikiem lepszego wykorzystania pamięci podręcznej procesorów.



Rys. 8. Przyspieszenie wykonania kodu pętli, kompilator ompi
Fig. 8. Loop speedup, ompi compiler



Rys. 9. Przyspieszenie wykonania kodu pętli, kompilator vs2005
Fig. 9. Loop speedup, vs2005 compiler

6. Wnioski

Zrównoleglenie kodu pętli pozwala na efektywne wykorzystanie systemów wieloprocesorowych i procesorów wielordzeniowych.

Dalsza poprawa efektywności zrównoleglonego kodu pętli jest możliwa poprzez zwiększenie lokalności danych, lepsze wykorzystanie pamięci podręcznej procesorów.

Efektywność kodu zrównoleglonego metodą hiperpłaszczyzn jest gorsza dla macierzy o dużych wymiarach, niż kodu, gdzie zrównoleglono zewnętrzną pętlę. Należy jednak zauważyć, że zrównoleglenie pętli zewnętrznej często jest niemożliwe ze względu na zależności miedzy iteracjami. Metoda hiperpłaszczyzn pozwala zrównoleglić znacznie większy zbiór pętli [6, 7, 11].

Celem kolejnych badań jest określenie możliwości zwiększenia obszaru, w którym przyspieszenie kodu zrównoleglonego metodą hiperpłaszczyzn jest maksymalne, poprzez zwiększenie lokalności danych, lepsze wykorzystanie pamięci podręcznej procesorów.

7. Literatura

[1] Allen, R, Kennedy, K.: Optimizing Compilers for Modern Architectures, Morgan Kaufmann, 2001

[2] D. Bacon, S. Graham, and O. Sharp. Compiler transformations for high-performance computing. Computing Surveys, 26(4):345-420, December 1994.

[3] U. Banerjee. Loop Transformations for Restructuring Compilers. Kluwer Academic, 1993.

[4] V. Beletskyy, M. Poliwoda. Parallelizing perfectly nested loops with non-uniform dependences. In Proceedings of the Advanced computer systems, pages 83-98, October 2002.

[5] H. Cameron , H. Tracey. Parallel and Distributed Programming Using C++. Prentice Hall Professional , 2003, 720 pp

[6] Darte, A., Robert, Y.., Vivien, F.: Scheduling and Automatic Parallelization. Birkhäuser Boston, 2000.

[7] L. Lamport. The Parallel Execution of DO Loops. Communications of the ACM, Vol. 17, No.2, Feb. 1974, pp. 83-93.

[8] Schrijver. Theory of Linear and Integer Programming. Wiley, Chichester, 1986.

[9] D. Watkins, M.Hammond, B. Abrams, Programming in the .NET Environment. Addison-Wesley, 2003.

[10] M. Wolfe. High Performance Compilers for Parallel Computing. Addison-Wesley Publishing Company,1995.

[11] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In Proc. ACM SIGPLAN 91 Conference on Programming Language Design and Implementation, pages 30–44, June 1991.

[12] H. Zima and B. Chapman. Supercompilers for Parallel and Vector Computers. ACM Press, 1990.

[13] http://www.openmp.org/drupal/

[14] W. Bielecki, M. Poliwoda Hyperplane method for loops parallelization in the .Net environment, Advanced Computer Systems - materiały konferencyjne, Szczecin, 2006

[15] Omni OpenMP Compiler Project; http://phase.hpcc.jp/Omni/

[16] OMPi OpenMP C Compiler; http://www.cs.uoi.gr/~ompi/

[17] Intel ® C++ Compiler for Windows* - Evaluation; http://www.intel.com/ cd/software/products

[18] Microsoft Visual Studio 2005; http://www.microsoft.com

Artykuł recenzowany