Tadeusz SCHUBRING

ALGORYTMY RÓWNOLEGŁE W JĘZYKU PROGRAMOWANIA C#

Streszczenie

W artykule opisano możliwość łatwego implementowania równoległych algorytmów w języku programowania C# . Zwrócono szczególną uwagę na zastosowanie metody For klasy Parallel dostępnej na platformie Microsoft .NET.

WSTĘP

Współczesne metody przybliżone stosowane w symulacji systemach kolejkowych, logistycznych, przepływów, układów elektronicznych oraz innych procesów wymagają rozwiązywania skomplikowanych problemów numerycznych. Do metod tych zaliczają się niewątpliwie metody iteracyjne polegające na konstruowaniu ciągu kolejnych przybliżeń prowadzących do rozwiązań z zadanymi dokładnościami. Jeżeli metoda pozwala na wyznaczanie współrzędnych wektora rozwiązania w dowolnej kolejności, to krok iteracyjny może być zrealizowany przy pomocy pewnej liczby współbieżnych wątków. Wątki te, korzystając z architektury równoległej, pozwalają na podniesienie efektywności obliczeń.

1. RÓWNOLEGŁA METODA ITERACYJNA

1.1. Algorytm sekwencyjnej metody iteracyjnej

Przyjmijmy, że algorytm jednopunktowej (p=1) albo wielopunktowej (p>1) metody iteracyjnej (RMI)

$$W=RMI(D)$$

przekształcający wektor danych

$$D=[d_i]_{1..m}$$

w wektor wyników

$$W=[w_i]_{1..n}$$

z zadanym błędem ε można zapisać następująco:

1. Ustal przybliżenie początkowe :

$$W^0, W^1, W^2, ..., W^{p-1}$$

2. Wyznacz kolejne przybliżenia wyniku W^k :

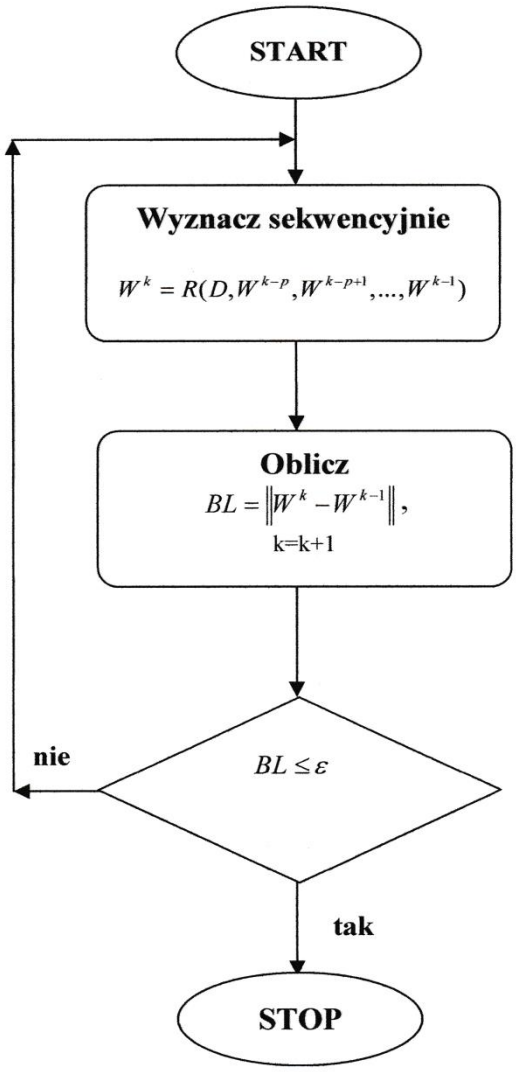
$$W^k = R(D, W^{k-p}, W^{k-p+1}, ..., W^{k-1})$$

dla $k = p...q$,

aby został spełniony warunek:

$$\left\| W^q - W^{q-1} \right\| \le \varepsilon .$$

Najczęściej stosowanymi miarami do oceny dokładności (błędu) są norma max oraz norma euklidesowa. Ogólny schemat postępowania został przedstawiony na Rys. 1.



Rys. 1. *Schemat algorytmu metody iteracyjnej*

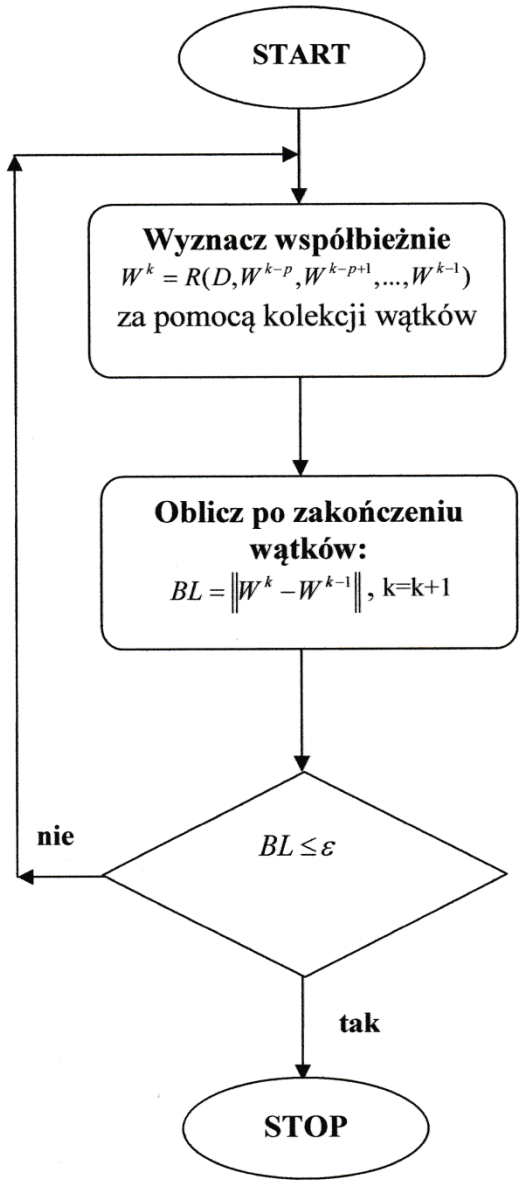
1.2. Algorytm równoległej metody iteracyjnej

Jeżeli metoda pozwala na wyznaczanie współrzędnych przybliżenia wektora rozwiązania

w_i dla i=1...n

w dowolnej kolejności, to współrzędne te mogą być przetwarzane przez współbieżne wątki. Pozwala to na uruchomienie wielu wątków

w procesie iteracyjnym, co w konsekwencji powoduje wzrost efektywności metody. Warto jednak zwrócić uwagę na zrównoważenie obciążenia poszczególnych wątków. Warunkiem poprawnego zakończenia kroku iteracyjnego metody równoległej jest dokonanie synchronizacji (przed oceną błędu aktualnego przybliżenia rozwiązania). Na ogół konieczne jest wstrzymanie głównego wątku do czasu zakończenia wszystkich wątków wyznaczających współrzędne kolejnego przybliżenia rozwiązania. Szkic metody został zamieszczony na Rys. 2.



Rys. 2. *Schemat algorytmu równoległej metody iteracyjnej*

1.3. Implementacja równoległej metody iteracyjnej w C#

Istnieje wiele możliwości zapisu algorytmu równoległej metody iteracyjnej w języku programowania. Można wykorzystać do tego takie języki i biblioteki jak: Java, C++, MPI, POSIX THREADS, Berkeley UPC, C# i wiele innych. W wymienionych środowiskach można w każdym kroku iteracyjnym uruchamiać kolekcję wątków i następnie, przed rozpoczęciem następnego kroku, wywołać odpowiednie funkcje synchronizujące.

W [4] pokazano równoległą implementację metody iteracyjnej dla macierzy rzadkiej w języku Berkeley UPC. Wykorzystano w tym programie instrukcję pętli równoległej

upc_forall(wyr1; wyr2; wyr3; f(i));

f(i) jest funkcją umożliwiającą kojarzenie wątków z iteracjami. Taka postać instrukcji pętli daje możliwość przypisywania iteracji poszczególnym wątkom. Zastosowanie tej instrukcji iteracyjnej wymaga wywołania funkcji bariery *upc_barrier* [3], która służy do synchronizacji pracy wątków.

Silne narzędzia programowania równoległego daje też, rozwijany przez firmę Microsoft, język C# [5]. Język ten znajduje coraz więcej zwolenników ze względu na wygodę, prostotę oraz możliwości zastosowania w wielu dziedzinach. W języku tym można dostrzec związki z syntaktyką i semantyką tak popularnych języków programowania, jak C , C++, Java czy Delphi. Począwszy od wersji 4.0 platformy .NET wprowadzono w C# bibliotekę *TPL* (ang. *Task Parallel Library*), która stała się nowym standardem programowania współbieżnego. Za pomocą instancji klas *Thread*, *Task*, *TaskFactory*, *TaskScheduler* oraz metod synchronizujących, można tworzyć i zarządzać niezależnie wykonywanymi ciągami instrukcji. Wśród zasobów *TPL* dostępna jest klasa *Parallel*. Klasa ta zawiera metodę *For* z przestrzeni nazw *System.Threading.Tasks* (W programie konieczne jest dołączenie: *using System.Threading.Tasks*) pozwalającą na zrównoleglenie pętli *for* bez większych problemów. Metoda *Parallel.For* jest łatwa w stosowaniu. Jej dwa pierwsze parametry określają zakres zmiany indeksu pętli. Za pomocą trzeciego argumentu przekazywane są wartości aktualnych indeksów. Trzeci argument jest delegatem, do którego można przypisać metodę wykonywaną w jednym kroku pętli albo metodę anonimową z wyrażeniem lambda wywoływanym w każdej iteracji pętli. Wyrażenie lambda odnosi się do takiego samego bloku instrukcji jaki występuje w oryginalnej instrukcji *for*. Poniżej przedstawiono przykład zrównoleglenia instrukcji pętli *for*. Sekwencyjna instrukcja pętli *for*

```
for(int i=0; i<N; i++)
  {
    Instrukcje
  }
```

może być zrównoleglona następująco za pomocą *Parallel.For:*

```
Parallel.For(0, N, ( i )=>
{
    Instrukcje
  });
```

Na uwagę zasługuje fakt, że metoda *Parallel.For* automatycznie synchronizuje używane zadania wykonywane w ramach poszczególnych iteracji. Wynika stąd, że po zakończeniu działania metody *Parallel.For* wszystkie instrukcje wykonywane przez współbieżne zadania są zakończone bez konfliktów w dostępie do danych. Trzeba tu jeszcze raz podkreślić, że poszczególne zadania w ramach iteracji *Parallel.For* są wykonywane współbieżnie, ale niekoniecznie w kolejności wzrastania indeksów. Wynika stąd, że żadna iteracja nie może zależeć od wartości policzonych w innych krokach iteracyjnych.

1.4. Przykład zrównoleglenia metody iteracyjnej

Jako przykład zrównoleglenia opisano implementację metody iteracyjnej Jacobiego rozwiązywania układu *n* równań liniowych postaci *Ax = b* w języku C#. Krok iteracyjny wyznaczania i-tej współrzędnej kolejnego przybliżenia rozwiązania układu równań za pomocą tej metody określony jest następująco [2]:

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j=0,\, j\neq i}^{n-1} a_{ij} x_j^{k-1}\right) \qquad (1)$$

Macierz układu równań *Ax* = b w formacie CSR [1, 4] zapisano za pomocą tablic V, J i P. W formacie CSR macierz przechowywana jest w trzech tablicach jednowymiarowych:

- V o wymiarze Mnz (Mnz - liczba niezerowych elementów tablicy) do przechowywania wartości niezerowych macierzy A (typ elementów V jest zgodny z typem elementów A),
- P o wymiarze N+1, zawierający indeksy tablicy V, w której zapisane są elementy A z kolejnych wierszy (elementy wiersza macierzy A o numerze i zapisane są w tablicy V w przedziale wskaźników [P[i], P[i+1] - 1] dla i=0, … , N-1),
- J o wymiarze Mnz, do zapisywania indeksów kolumn macierzy A dla elementów zapisanych w tablicy wartości V.

Kod sekwencyjny w języku C# wyznaczający i-tą współrzędną kolejnego przybliżenia rozwiązania metodą Jacobiego przyjmuje postać:

```
for(uint i=0; i<N; i++)
   {
      double r = 0, rA = 1;
      for (uint k = P[i]; k < P[i + 1]; k++)
         if (J[k] != i) r -= V[k] * x1[J[k]];
         else rA = V[k];
      x[i] = (b[i] + r) / rA;
   }
```

Zrównoleglenie kodu zrealizowano za pomocą metody *For* klasy *Parallel:*

```
Parallel.For(0, N,
          (i) =>
          {
             double r = 0, rA = 1;
             for (uint k = P[i]; k < P[i + 1]; k++)
                if (J[k] != i) r -= V[k] * x1[J[k]];
                else rA = V[k];
             x[i] = (b[i] + r) / rA;
          });
```

Pełny kod w wersji zrównoleglonej przedstawiono na poniższym listingu:

```
ITER=200;
max=1;
iter=0;
eps=1e-10;
t1 = DateTime.Now;
while (iter <= ITER && max > eps)
   {
      Parallel.For(0, N,
       (i) =>
       {
          double r = 0, rA = 1;
          for (uint k = P[i]; k < P[i + 1]; k++)
             if (J[k] != i) r -= V[k] * x1[J[k]];
             else rA = V[k];
          x[i] = (b[i] + r) / rA;
          });
      //Zadania obliczające x[i] zakończyły pracę
      iter++;
      //Obliczanie błędu aktualnego przybliżenia
      //rozwiązania
      max = 0;
      for (uint k = 0; k < N; k++)
      {
         z = (x[k] - x1[k]);
         max = max + z * z;
         x1[k] = x[k];
      }
      max = Math.Sqrt(max);
   }
t2 = DateTime.Now;
Console.WriteLine("Start  : " + t1);
Console.WriteLine("Stop  : " + t2);
Console.WriteLine("\nLiczba iteracji : " + iter);
Console.WriteLine("\nError : " + max);
```

Program w wersji sekwencyjnej i równoległej uruchomiono w języku C# 6.0 na platformie Microsoft NET Framework 4.6. Obliczenia przeprowadzono dla układu równań z niewstęgową, niesymetryczną macierzą rzadką wymiaru 2000000x2000000 o liczbie niezerowych elementów 20000000, spełniającej warunek: w każdym wierszu macierzy A moduł elementu na przekątnej jest większy od sumy modułów współczynników poza przekątną. Obliczenia przeprowadzono przy zadanej dokładności eps=10^{-10}. Rozwiązanie zostało wyznaczone w 161 iteracjach. Czasy obliczeń umieszczono w Tab.1.

Tab. 1. *Czasy obliczeń sekwencyjnego i równoległego programu C#*

Komputer	System operacyjny	Wersja sekwencyjna czas [s]	Wersja równoległa czas [s]
AMD FX 83-50, 8 rdzeni 4 GHz, 32 GB RAM	Windows 7Pro 64 bit	31	10
AMD Phenom GP-9500, 4 rdzenie, 2.2 GHz, 4GB RAM	Windows 10Pro 64 bit	59	28

Z przeprowadzonych testów wynika, że samo zrównoleglenie instrukcji iteracyjnej *for* za pomocą metody *For* klasy *Parallel* zmniejszyło czas obliczeń na komputerze z procesorem 8 rdzeniowym 3-krotnie oraz z procesorem 4-rdzeniowym 2-krotnie. Warto też wspomnieć, nie była potrzebna dodatkowa analiza obciążenia zadań wykonujących instrukcje w ramach poszczególnych kroków iteracyjnych.

Powyższy przykład pokazał, że niewielkim nakładem pracy, można uzyskać bardziej efektywny kod w języku C# w systemie komputerowym pozwalającym na równoległe przetwarzanie zadań.

PODSUMOWANIE

Narzędzia dostępne w języku C# dają szerokie możliwości tworzenia programów równoległych. Oprócz znanych z wielu innych języków możliwości programowania wielowątkowego w języku C# istnieje możliwość tworzenia aplikacji wielozadaniowych z wykorzystaniem klas: *Task, TaskFactory i TaskScheduler*. Dla piszących programy komputerowe i nie wgłębiających się w zagadnienia programowania równoległego istnieje możliwość łatwego zrównoleglenia instrukcji pętli for poprzez zastosowanie metody *Parallel.For,* bez konieczności stosowania narzędzi synchronizujących. Jest to możliwe w sytuacji, gdy kolejna iteracja nie zależy od wartości żadnej zmiennej policzonej w innym kroku iteracyjnym. Wyniki obliczeń wyznaczania układu równań metodą Jacobiego pokazują, że samo zrównoleglenie pętli for w C# spowodowało skrócenie czasu obliczeń na komputerze z procesorem 8 rdzeniowym 3 krotnie, a z procesorem 4 rdzeniowym 2 krotnie.

BIBLIOGRAFIA

1. Krzyżanowski P., *Matematyka obliczeniowa II*. Uniwersytet Warszawski, 2012.
2. Kosma Z., *Metody numeryczne dla zastosowań inżynierskich.* Wydawnictwo Politechniki Radomskiej, 2008.
3. El-Ghazawi T., Carlson W. i inni, *UPC – Distrubuted shared memory Programming*. New Jersey, John Wiley & Sons, 2005.
4. Schubring T., *Równoległa implementacja metod iteracyjnych dla układu równań z macierzami rzadkimi*, Technika Transportu Szynowego 10/2013.
5. Warczak M., Matulewski J., Pawłaszek R., Sybilski P., Borycki D., Dziubak T., *Programowanie równoległe i asynchroniczne w C# 5.0*. Helion 2014.

PARALLEL ALGORITHMS IN THE C# PROGRAMMING LANGUAGE

Abstract

The article describes the ability to easily implement parallel algorithms in the C# programming language. Special attention was paid to the application of the method For class Parallel available on the Microsoft .NET platform.

Autorzy:
dr **Tadeusz Schubring** – Uniwersytet Technologiczno - Humanistyczny w Radomiu, Wydział Mechaniczny, Instytut Mechaniki Stosowanej i Energetyki, Zakład Komputerowych Metod Inżynierskich, email: t.schubring@uthrad.pl.