STUDIA INFORMATICA 2010
Volume 31 Number 3 (91)

Dariusz R. AUGUSTYN, Szymon KUNC
Politechnika Śląska, Instytut Informatyki

EFEKTYWNOŚĆ PROGRAMÓW PRZEZNACZONYCH DO SYMULACJI CIĄGŁYCH UKŁADÓW DYNAMICZNYCH, WYKORZYSTUJĄCYCH MODUŁ PARALLEL EXTENSIONS TO .NET FRAMEWORK, URUCHAMIANYCH NA KOMPUTERACH Z PROCESORAMI WIELORDZENIOWYMI

Streszczenie. Artykuł prezentuje wyniki wydajnościowej analizy programów przeznaczonych do symulacji ciągłych układów dynamicznych, utworzonych z wykorzystaniem modułu Parallel Extensions to .NET Framework. Przedmiotem rozważanym w artykule jest modelowanie ruchu układów ciał w polu grawitacyjnym. W pracy pokazano zalety zrównoleglonych programów, zbudowanych na podstawie technologii .NET. W artykule przedstawiono wydajnościowe porównania zaproponowanego rozwiązania do rozwiązań sekwencyjnych: skryptów systemu MATLAB i programów jednowątkowych, wykonanych w technologii .NET, uruchamianych na komputerach z procesorami wielordzeniowymi. W pracy rozważono zagadnienie skalowalności zaproponowanego rozwiązania.

Słowa kluczowe: symulacja ciągłych układów dynamicznych, moduł Parallel Extensions to .NET Framework, wykorzystanie procesorów o architekturze wielordzeniowej, efektywność i skalowalność przetwarzania zrównoleglonego

EFFECTIVENESS OF PROGRAMS USED FOR SIMULATION OF CONTINUOUS DYNAMICAL SYSTEMS BASED ON PARALLEL EXTENSIONS TO .NET FRAMEWORK RUN ON MULTICORE MACHINES

Summary. The paper presents an effectiveness analysis of programs for simulation of continuous dynamical systems based on Parallel Extensions to .NET Framework. Modeling of a movement of bodies systems in a gravitational field is considered. Advantages of parallel .NET-based programs are shown. Effectiveness comparison of the proposed solution to MATLAB scripts or sequential single-thread .NET based programs (all run on multicore machines) is presented. Scalability of the proposed solution is considered too.

Keywords: simulation of continuous dynamical system, Parallel Extensions to .NET Framework, usage of multicore processor architecture, effectiveness and scalability of parallel processing

1. Wstęp

Rozwój architektury komputerów w kierunku procesorów wielordzeniowych oraz ich powszechna dostępność stwarzają nowe możliwości dla twórców oprogramowania. Obecnie (2010) „przeciętna" stacja robocza wyposażona jest co najmniej w procesor dwurdzeniowy.

Moc obliczeniowa komputerów wieloprocesorowych lub z procesorami wielordzeniowymi jest oczywiście w dużym stopniu konsumowana w związku z obsługą wieloprocesowości w systemie operacyjnym, dzięki możliwości jednoczesnego uruchamiania wielu usług i aplikacji w ramach jednego komputera. Jest to korzystne nawet wtedy, gdy pojedyncze, jednowątkowe aplikacje nie potrafią same wykorzystać możliwości wielordzeniowego sprzętu.

Jednak faktycznym wyzwaniem jest tworzenie wydajnego oprogramowania w pełni wykorzystującego moc obliczeniową wielordzeniowych procesorów (ang. *multicore-aware software*), przez możliwość zrównoleglenia przetwarzania.

Istotną cechą takiego oprogramowania powinna być możliwość automatycznej adaptacji do architektury systemu, na którym program jest uruchamiany. Program powinien wydajnościowo nie być gorszy od swojego sekwencyjnego odpowiednika w przypadku uruchomienia na komputerze z klasycznym, jednordzeniowym procesorem. Środki programowe użyte do wytworzenia takiego oprogramowania (dostępne poprzez API – ang. *application programming interface*) powinny być tak skonstruowane, by kod źródłowy programu zrównoleglonego był możliwie zbliżony do kodu źródłowego programu sekwencyjnego. Zarządzanie dystrybucją zrównoleglonych zadań pomiędzy rdzeniami powinno odbywać się automatycznie i winno być ukryte przed programistą (programista powinien się skupiać na merytorycznych aspektach algorytmu przetwarzania).

Jednym z nowych podejść do omawianego zagadnienia jest możliwość tworzenia wydajnego oprogramowania opartego na technologii .NET firmy Microsoft Corp. i wykorzystującego moduł Parallel Extensions to .NET Framework (Pfx) [16, 17, 18]. Niniejszy artykuł ma pokazać, jak można budować programy symulacji ciągłych układów dynamicznych z użyciem modułu Pfx, działające wydajnie na maszynach z procesorami wielordzeniowymi. Celem artykułu jest pokazanie zalet takiego rozwiązania w stosunku do sekwencyjnych implementacji, wykonanych z użyciem .NET albo wykorzystujących system MATLAB. MATLAB jest środowiskiem dedykowanym do tego typu zadań, związanych z modelowaniem cyfrowym, natomiast zaproponowane, alternatywne rozwiązanie (oparte na Pfx) ma,

w wybranym zakresie zastosowań, poprawić wydajność symulacji. Celem artykułu jest również pokazanie potencjalnych ograniczeń przedstawianych rozwiązań.

2. Modelowanie matematyczne ciągłych układów dynamicznych – zastosowanie jednokrokowych metod całkowania równań stanu

Układy dynamiczne ciągłe, zadane przez równania różniczkowe, mogą być również opisywane przez równania stanu. Takie podejście pozwala na zastosowanie metod numerycznych, prowadzących do znalezienia przybliżonego rozwiązania.

Równania stanu są równaniami różniczkowymi pierwszego rzędu. Znanych jest wiele metod uzyskiwania równań stanu (np. metody: ogólna, kolejnych całkowań, zmiennej pomocniczej, szeregowa i inne [1, 2]). Liczba równań stanu odpowiada rzędowi wyjściowego równania różniczkowego, opisującego zadany układ dynamiczny.

Lewą stronę równania stanu stanowi pochodna zmiennej stanu:

$$\frac{dx}{dt} = f(t, x) \tag{1}$$

Wyznaczenie przybliżonego rozwiązania równania różniczkowego sprowadza się do numerycznego całkowania prawych stron równań stanu (funkcji f we wzorze 1), co pozwala na znalezienie przybliżeń zmiennych stanu (zmienna x we wzorze 1). Numeryczne wyznaczenie wartości zmiennych stanu odbywa się przez zastosowanie tzw. schematów różnicowych, które są równaniami różnicowymi, pozwalającymi na znalezienie przybliżonego rozwiązania w dyskretnych chwilach czasu [1, 13]. Celem symulacji (uzyskania numerycznego rozwiązania) jest znalezienie zbioru par (t_k, x_k), gdzie ciąg $\{t_k\}$ stanowi dyskretną reprezentację zmiennej niezależnej t, a $\{x_k\}$ jest ciągiem przybliżonych wartości zmiennej stanu, czyli $x(t_k)$. Dokładność rozwiązania zależy między innymi od sposobu dyskretyzacji t, czyli doboru wartości kroku całkowania h, spełniającego zależność $t_{k+1} = t_k + h$.

Znajdywanie kolejnych x_{k+1} odbywa się na podstawie wartości uzyskanych poprzednio. W otwartych metodach wielokrokowych (n-krokowych) [1] x_{k+1} oblicza się jako liniową kombinację określoną na poprzednio obliczonych (n) wartościach zmiennych stanu x_k, x_{k-1}, …, x_{k-n+1} oraz wartościach lewych stron równań stanu, tzn. $f(t_i, x_i)$ dla $i = k,\ldots, k - n+1$.

W omawianej pracy posłużono się metodami jednokrokowymi Rungego-Kutty [1, 3], gdzie wartość x_{k+1} jest wyznaczana tylko na podstawie x_k i ciągu wartości f :

$$x_{k+1} = x_k + \sum_{i=1}^{s} b_i K_i, \tag{2}$$

gdzie:

$$
\begin{aligned}
K_1 &= hf(t_k, x_k),\\
K_2 &= hf(t_k + c_2 h, x_k + a_{21}K_1),\\
K_3 &= hf(t_k + c_3 h, x_k + a_{31}K_1 + a_{32}K_2),\\
&\vdots\\
K_s &= hf(t_k + c_s h, x_k + a_{s1}K_1 + a_{s2}K_2 + \cdots + a_{s\,s-1}K_{s-1}).
\end{aligned}
$$

Wartość *s* określa tzw. liczbę etapów metody. Wartości stałych c_j i a_{wr} są dobierane tak, aby zminimalizować tzw. lokalny błąd obcięcia metody ([1] rozdział. 4).

Niech $\tilde{f}_m$ oznacza rozwinięcie *f* w szereg Taylora z dokładnością do *m*-tej pochodnej. Metoda rzędu *m* pozwala na „dokładne całkowanie" takich funkcji *f*, dla których $f = \tilde{f}_m$. Precyzyjna definicja rzędu metody całkowania, wykorzystująca pojęcie lokalnego błędu obcięcia, podana została w [1] na str. 43.

Metody jednokrokowe, stosunkowo łatwe w implementacji, okazały się wystarczająco skuteczne w wyznaczaniu przybliżonych rozwiązań przykładowych zagadnień modelowania, m.in. tych opisanych w podrozdziale 3. Opierając się na wzorze 2, w ramach niniejszej pracy, zaimplementowano w języku C# następujące warianty jednokrokowych metod całkowania:

- Eulera (Rungego-Kutty 1 rzędu) – najprostsza metoda o stałym kroku całkowania.
- RK4 (Rungego-Kutty 4 rzędu) – metoda o stałym kroku całkowania.
- RKF45 (Rungego-Kutty-Fehlberga) – metoda Rungego-Kutty 4 i 5 rzędu; metoda o zmiennym kroku całkowania. Na podstawie wyników etapów roboczych, tzn. przez zastosowanie RK4 i RK5, oraz po porównaniu wartości przybliżeń następuje akceptacja lub odrzucenie rozwiązania. W przypadku odrzucenia następuje redukcja kroku całkowania i powtórzenie obliczeń. W przypadku akceptacji, gdy dodatkowo uzyskane przybliżenia różnią się mniej, niż zostało to ustalone w wymaganiach, następuje zwiększenie kroku całkowania (czyli potencjalne przyspieszenie działania w następnym kroku metody).
- RK45 (Rungego-Kutty-Dormand-Prince) – metoda Rungego-Kutty 4 i 5 rzędu; metoda o zmiennym kroku całkowania. W pojedynczym kroku metoda wykorzystuje siedem wyznaczonych wartości prawych stron równań stanu, ale ostatnia ewaluacja jest współdzielona z następnym krokiem, stąd praktycznie wartość *f* jest wyznaczana sześć razy. Jest to podstawowa metoda całkowania (tzw. *solver*), wykorzystywana w systemie MATLAB-/Simulink [4] i Octave [5] (m.in. użyta w ramach funkcji *ode45* zastosowanej w przykładach w podrozdziale 4).

3. Przykłady zadania modelowania matematycznego – analiza ruchu układu trzech ciał ciężkich i ciała lekkiego w polu grawitacyjnym

Efektywność stworzonych programów symulacji zaimplementowanych w MATLABie i C# została zweryfikowana m.in. przez realizację zadania modelowania matematycznego płaskiego ruchu układu ciał w polu grawitacyjnym.

Pierwszy przykład dotyczy układu czterech ciał. Trzy jednakowe ciała ciężkie umieszczone są w wierzchołkach trójkąta równobocznego (rys. 1a – układ trójkątny). Ciała te obracają się wokół wspólnego środka ciężkości. Oś obrotu, przechodząca przez środek układu współrzędnych, jest prostopadła do płaszczyzny rysunku. W rozwiązaniu przyjęto obracający się nieinercjalny układ odniesienia, w którym trzy ciała ciężkie pozostają nieruchome. W układzie tym siła odśrodkowa działająca na jedno ciało ciężkie jest równoważona przez wypadkową sumę sił grawitacji, wynikających z oddziaływania z pozostałymi dwoma ciałami. Do tak zdefiniowanego układu wprowadzane jest czwarte ciało – ciało lekkie. Jego masa jest na tyle znikoma, że obecność ciała lekkiego nie ma żadnego wpływu na ruch ciał ciężkich. Przedmiotem badań jest analiza ruchu ciała lekkiego w zależności od warunków początkowych (początkowego położenia i prędkości). Opisywany ruch ciała lekkiego jest ruchem płaskim.

Na ciało lekkie w nieinercjalnym układzie odniesienia działają następujące siły:

- siły grawitacji (związane z oddziaływaniem z każdym z trzech ciał ciężkich):

$$\vec{F}_i = -G\frac{m_i m}{r_i^3}\vec{r}_i = -G\frac{m_i m}{\left((x-x_i)^2+(y-y_i)^2\right)^{\frac{3}{2}}}[x-x_i,\, y-y_i], \tag{3}$$

  gdzie i = 1, 2, 3, m_i – masa i-tego ciała ciężkiego, m – masa ciała lekkiego, $[x_i, y_i]$ – położenie i-tego ciała ciężkiego, $[x, y]$ – położenie ciała lekkiego, $\vec{r}_i$ – wektor odległości i-tego ciała ciężkiego od ciała lekkiego, G – stała grawitacji,

- siła odśrodkowa:

$$\vec{F}_{od} = m\omega^2\vec{r} = m\omega^2[x, y], \tag{4}$$

  gdzie ω – prędkość kątowa obrotu układu odniesienia,

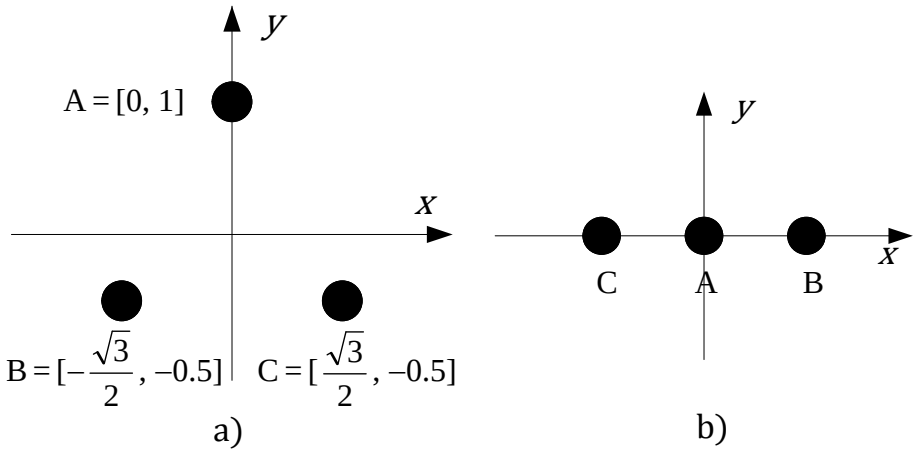
- siła Coriolisa:

$$\vec{F}_{cor} = -2m(\vec{\omega}\times\vec{v}) = 2m(\vec{v}\times\vec{\omega}), \tag{5}$$

  gdzie $\vec{v} = [v_x, v_y]$ – prędkość ciała lekkiego w nieinercjalnym układzie odniesienia, $\vec{\omega}$ – wektor prędkości kątowej układu, prostopadły do płaszczyzny ruchu (w szczególności zawsze prostopadły do wektora $\vec{v}$); uwzględniając (5) i ortogonalne położenie wektora $\vec{\omega}$:

$$\vec{F}_{cor} = 2m[\omega v_x, -\omega v_y], \tag{6}$$

których wypadkowa decyduje o ruchu ciała lekkiego, tzn. determinuje wartość przyspieszenia tego ciała, zgodnie z drugą zasadą dynamiki Newtona.



Rys. 1. Rozmieszczenie ciał ciężkich: a) – w układzie trójkątnym z przykładu 1, b) – w układzie współliniowym z przykładu 2

Fig. 1. Location of heavy bodies: a) – in the triangular system from the 1st example, b) – in the collinear system from the 2nd one

Drugi z rozpatrywanych przykładów dotyczy analizy ruchu czwartego ciała lekkiego w nieinercjalnym układzie trzech współliniowych ciał ciężkich (rys. 1b – układ współliniowy).

W wyniku symulacji ma powstać obraz składający się z punktów, których współrzędne (x, y) odpowiadają początkowemu położeniu ciała lekkiego, a kolor określa historię ruchu ciała lekkiego.

Parametrami symulacji są:

- rozmiary każdego z trzech jednakowych ciał ciężkich (promień – D),
- całkowity czas symulacji (T), czas całkowania dla pojedynczego wektora warunków początkowych,
- promień obszaru określającego kołowe sąsiedztwo układu ciał ciężkich (R),
- współrzędne prostokątnego obszaru zainteresowania, zawierającego badany układ trzech ciał ciężkich [*XMin*, *XMax*, *YMin*, *YMax*],
- rozdzielczość obrazu, liczba punktów startowych; zadana jest w każdym z dwóch wymiarów [*ResolX*, *ResolY*].

W wyniku symulacji (trwającej przez czas T) mogą nastąpić następujące zdarzenia:

- ciało lekkie opadło na ciało ciężkie A; ciało lekkie zbliżyło się na odległość mniejszą lub równą D, liczoną od środka ciała ciężkiego A (wtedy punkt startowy ciała lekkiego zostaje oznaczony kolorem czerwonym[1]),
- ciało lekkie opadło na ciało ciężkie B (punkt startowy oznaczony kolorem niebieskim),

[1] Na portalu zeszytów Studia Informatica: http://znsi.aei.polsl.pl/, artykuł dostępny jest z rysunkami w kolorze.

- ciało lekkie opadło na ciało ciężkie *C* (punkt startowy oznaczony kolorem zielonym),
- ciało lekkie zdołało opuścić układ ciał, tzn. oddaliło się co najmniej raz na odległość większą niż *R* od środka układu współrzędnych (punkt startowy oznaczony kolorem czarnym),
- ciało lekkie nie opuściło układu ani nie opadło na żadne z ciał ciężkich (punkt startowy oznaczony kolorem żółtym).

Zbiór punktów startowych, dla których historia ruchu ciała lekkiego jest jednakowa w powyższym sensie (tzn. można ruch zakwalifikować do jednej z wymienionych pięciu kategorii), nazwano basenem (obszarem) przyciągania. Każdy obszar przyciągania wyróżniony jest innym kolorem. Jednym z ciekawszych elementów badań omawianych układów może być analiza własności obszarów przyciągania. Przykładowe wyniki symulacji (postacie basenów przyciągania) zostały pokazane na rys. 2, 3 i 4.

W omawianych eksperymentach przyjęto następujące założenia dotyczące wartości: prędkości kątowej obrotu układu ciał ciężkich w układzie inercjalnym, stałej grawitacji, masy ciała lekkiego:

$$\omega = 1, G = 1, m = 1. \tag{7}$$

Przyjęcie takich wartości nie zmniejsza ogólności rozwiązania.

4. Implementacja przykładu z wykorzystaniem środowiska MATLAB

Budowa opisu cyfrowego modelu dynamicznego układu ciągłego w systemie MATLAB (wersja 2007) sprowadza się do utworzenia tzw. *m*-pliku funkcyjnego, służącego do wyznaczania wartości prawych stron równań stanu. Sygnatura takiej funkcji ma następującą postać:

```
function dxx = move4 (t, xx),
```

gdzie *move4* to przyjęta tutaj nazwa funkcji, *t* – skalar, chwilowa wartość zmiennej niezależnej, *xx* – wektor wartości zmiennych stanu w chwili *t*, *dxx* – wynikowy wektor wartości pochodnych zmiennych stanu.

W omawianych przykładach 1 i 2 funkcja *move4* opisuje dynamikę ruchu czwartego ciała lekkiego, tzn. wektor $xx = [x, y, v_x, v_y]$, gdzie $[x, y]$ określa położenie, a $[v_x, v_y]$ prędkość ciała lekkiego w układzie nieinercjalnym. W ramach ciała funkcji wypracowywane są wartości $dxx = [v_x, v_y, a_x, a_y]$, gdzie $[a_x, a_y]$ określa wypadkowe przyspieszenie ciała lekkiego. Wyznaczenie przyspieszenia odbywa się z uwzględnieniem wszystkich sił działających na ciało lekkie w układzie nieinercjalnym (siła Coriolisa, odśrodkowa i 3 razy siła grawitacyjna).

W ramach wykonania pojedynczego eksperymentu (realizacja symulacji dla zadanego wektora warunków początkowych) następuje wywołanie *ode45* – funkcji „całkowania" (tutaj metodą RK45), wykorzystującej *move4*:

```
[t, xxres] = ode45 ('move4', [0  2*pi], [x0 y0 vx0 vy0], opt),
```

gdzie [0 2**pi*] – przedział czasu symulacji, [*x*0 *y*0 *vx*0 *vy*0] – warunki początkowe, *opt* – opcje całkowania, pozwalające m.in. określić tolerancję błędu całkowania: *AbsTol*, *RelTol*, *xxres* – wynikowa macierz wartości zmiennych stanu (dwie pierwsze kolumny określają składowe położenia, dwie następne składowe prędkości), *t* – wynikowy wektor wartości zmiennej niezależnej (rozmiar wektora *t* równy jest liczbie wierszy macierzy *xxres*).

Program symulacji dokonuje analizy prostokątnego obszaru warunków początkowych. Strukturę programu można w dużym uproszczeniu objaśnić poniższym pseudokodem (opartym na języku MATLAB):

```
for i=0:ResolX
   for j=0:ResolY
    % Wyznaczenie warunków początkowych:
    %[x0 y0] na podstawie i, j;
    % przykład 1: [vx0 vy0] = [0 0], przykład 2: [vx0 vy0] = [y0 -x0];
    % Wykonanie eksperymentu:
    [t, xxres] = ode45 ('move4', [0  2*pi], [x0 y0 vx0 vy0], opt);
    % Klasyfikacja ruchu ciała lekkiego do jednej z 5 kategorii,
    % na podstawie wartości położenia ciała xxres(:,1) i xxres (:,2):
    % upadek na ciało A, B, lub C, opuszczenie sąsiedztwa ciał ciężkich,
    % żaden z powyższych 4 przypadków; Odnotowanie jednego z ww. zdarzeń
  end
end
%  Zapisanie obrazu - charakterystyki ruchu ciała lekkiego
```

5. Funkcjonalność zrównoleglającego rozszerzenia środowiska .NET – Parallel Extensions to .NET Framework

Parallel Extensions (Pfx – ang. ***P**arallel **E**xtensions to .NET **F**ramework*) [16, 17, 18] jest tzw. zarządzaną (ang. *managed*) biblioteką obsługi współbieżności, przeznaczoną do wykorzystania w środowisku uruchomieniowym .NET 3.5, utworzoną przez dział badawczy firmy Microsoft. Biblioteka została pierwszy raz publicznie udostępniona jako CTP (ang. *Community Technology Preview)* już w listopadzie 2007. W tym czasie implementacja Pfx znajdowała się w osobnym zespole (ang. *assembly*) – *System.Threading.dll*. Obecnie mechanizmy te zostały przeniesione do podstawowego zespołu .NET Framework, czyli do *mscorlib.dll*, a jego finalna wersja stała się integralnym elementem pakietu .NET 4.0 [8], którego premiera miała miejsce w kwietniu 2010 (wraz z premierą środowiska wytwórczego MS Visual Studio 2010 [9]).

Celem powstania tej biblioteki jest ułatwienie współczesnym programistom lepszego wykorzystania potencjału obliczeniowego komputerów wyposażonych w wielordzeniowe proce-

sory. Cel ten ma zostać osiągnięty przez dostarczenie rozwiązań, które uniezależniają twórcę oprogramowania od uwarunkowań sprzętowych, w jakich przyjdzie wykonywać daną aplikację. Z powyższych założeń wynikają wymagania co do skalowalności:

- aplikacja wykorzystująca Pfx nie powinna ustępować efektywnościowo szeregowym odpowiednikom tych samych programów, uruchamianych na komputerach jednoprocesorowych,
- aplikacja powinna elastycznie wykorzystywać dostępne dodatkowe procesory/rdzenie bez konieczności rekompilacji.

W celu spełnienia powyższych wymagań rozszerzenia wprowadzają pewne zmiany w środowisku uruchomieniowym. Środowisko zapewnia, aby liczba pracujących wątków dostosowana była do liczby jednostek wykonawczych na danej maszynie. Co więcej, zakolejkowane zadania do danego wątku mogą zostać przeniesione do kolejki innego wątku, w celu równoważenia obciążenia – tzw. mechanizm *work stealing*. Równocześnie silnik wykonawczy gwarantuje, że rozpoczęte w danym wątku zadanie nie zostanie przeniesione do innego wątku.

Rozszerzenia Parallel Extensions standardowej biblioteki .NET składają się z następujących komponentów:

- Parallel LINQ – komponent deklaratywnego programowania dla zrównoleglonego dostępu do danych opartego na LINQ (ang. *Language-integrated Query*),
- biblioteka równoległych zadań TPL (ang. *Task Parallel Library*) – rozszerzenia umożliwiające zastosowanie równoległości w imperatywnym modelu programowania przez konstrukcje typu *Task*, *Future* oraz konstrukcje na nich bazujące, np. *Parallel.For*, *Parallel.ForEach*,
- struktury koordynujące – zestaw struktur ułatwiających bezpieczną wielowątkowo komunikację, synchronizację pomiędzy zrównoleglonymi zadaniami (*ConcurrentQueue* – struktura FIFO, *ConcurrentStack* – LIFO).

6. Implementacja przykładu z wykorzystaniem modułu Parallel Extensions to .NET Framework

Opis modelu układu dynamicznego dla przykładu 1 w języku C#, oparty na wzorach 3-7, w postaci metody wyznaczającej wartości prawych stron równań stanu, został pokazany poniżej:

```
static double[] move4(double t, double[] xx)
{ // t  - zmienna niezależna
  // xx - wektor zmiennych stanu (ruch ciała lekkiego)
  // Bezpieczna wielowątkowo aktualizacja globalnego
  // licznika wywołań metody równań stanu - odpowiednik cnt++
```

```
        Interlocked.Increment(ref cnt);
        // Określenie parametrów ruchu ciała lekkiego (ciało nr 4)
        // Położenie ciała lekkiego w chwili t
        double x = xx[0];
        double y = xx[1];
        // Prędkość ciała lekkiego w chwili t
        double vx = xx[2];
        double vy = xx[3];
        // Masa każdego ciała ciężkiego
        double m = heavyBodyMass;
        // Współrzędne określające stałe położenie ciał cieżkich
        double x1 = initialPosition.x1;  //  0
        double y1 = initialPosition.y1;  //  1
        double x2 = initialPosition.x2;  //  -Math.sqrt(3)/2
        double y2 = initialPosition.y2;  //  -0.5
        double x3 = initialPosition.x3;  //  +Math.sqrt(3)/2
        double y3 = initialPosition.y3;  //  -0.5
        // Odległości między ciałem lekkim a ciałami cieżkimi
        double d1 = Math.Sqrt((x1 - x) * (x1 - x) + (y1 - y) * (y1 - y));
        double d2 = Math.Sqrt((x2 - x) * (x2 - x) + (y2 - y) * (y2 - y));
        double d3 = Math.Sqrt((x3 - x) * (x3 - x) + (y3 - y) * (y3 - y));
        // Określenie wartosci sił grawitacji działających na ciało lekkie
        double f1 = m / (d1 * d1);
        double f2 = m / (d2 * d2);
        double f3 = m / (d3 * d3);
        // Określenie składowych sił grawitacji działających na ciało lekkie
        double f1x = f1 * (x1 - x) / d1;
        double f1y = f1 * (y1 - y) / d1;
        double f2x = f2 * (x2 - x) / d2;
        double f2y = f2 * (y2 - y) / d2;
        double f3x = f3 * (x3 - x) / d3;
        double f3y = f3 * (y3 - y) / d3;
        // Obliczanie składowych siły odśrodkowej dla ciała lekkiego
        double fodx = x;
        double fody = y;
        // Obliczanie składowych siły Coriolisa działającej na ciało lekkie
        double fcorx = 2 * vy;
        double fcory = -2 * vx;
        // Wektor pochodnych zmiennych stanu
        double[] dxx = new double[4];
        // Wyznaczenie wartości prawych stron równan stanu
        dxx[0] = vx; // Składowa x prędkości ciała lekkiego
        dxx[1] = vy; // Składowa y prędkości
        // Składowa x wypadkowego przyspieszenia ciała lekkiego
        dxx[2] = f1x + f2x + f3x + fodx + fcorx; //
        // Składowa y wypadkowego przyspieszenia
        dxx[3] = f1y + f2y + f3y + fody + fcory; //
        return dxx;
    }
```

Metoda *move4* stanowi odpowiednik pliku *m*-funkcyjnego w języku MATLAB, o tej samej nazwie, o którym była mowa w podrozdziale 4. Zaimplementowane w C# metody całkujące posiadają m.in. parametr typu delegatowego [12] (o sygnaturze zgodnej z *move4*). Metody całkujące wywołują przekazaną metodę *move4* w ramach realizacji eksperymentu (tzw. *callback invoking*). Proces ten odpowiada działaniu funkcji *ode45* w systemie MATLAB.

Kod źródłowy metody *move4* czy innej metody, opisującej równania stanu dowolnego układu dynamicznego, może zostać napisany przez programistę, ale może też być wygenerowany automatycznie przez użycie narzędzia dokonującego translacji kodu w języku MATLAB na kod w języku C#. Takie podejście [11], z wykorzystaniem narzędzia zapropo-

nowanego przez autorów, ma ułatwić wykorzystanie platformy .NET w modelowaniu układów dynamicznych, których opis został wcześniej przygotowany w MATLABie. W szczególności, pośrednio, może ułatwić skorzystanie z możliwości modułu Parallel Extensions to .NET Framework.

Struktura całego sekwencyjnego programu symulacji w języku C# jest analogiczna do struktury programu w języku MATLAB (pokazanej na ostatnim listingu w podrozdziale 4). W uproszczeniu można stwierdzić, że oparta jest na zagnieżdżonych pętlach *for* (dla zmiennych *i* oraz *j*).

Zrównoleglony program symulacji w języku C#, oparty na Pfx, wykorzystuje konstrukcję *Parallel.For* oraz wyrażenia *lambda* [10]. Zewnętrzna pętla *for* (dla *i*) została zastąpiona zrównoleglonym odpowiednikiem następująco:

```
Parallel.For(0, ResolX, () =>
{   // Inicjalizacja zasobu lokalnego,
    // indywidualnego dla każdego zrównoleglonego zadania,
    // przez stworzenie i użycie obiektu typu ThreadLocal,
    // zdefiniowanego wcześniej w programie
    ThreadLocal tLocal = new ThreadLocal();
    tLocal = ...
    return tLocal;
},
(int i, ParallelState<ThreadLocal> loopState) =>
{
  for (int j = 0; j < ResolY; j++)
  {
    // Wyznaczenie warunków początkowych
    // Wykonanie eksperymentu: ... całkowanie z użyciem move4
    //   (m.in. użycie loopState - zasobu lokalnego)
    // Klasyfikacja ruchu ciała lekkiego ...
  }
});
```

Taka struktura programu, jak powyżej – *Parallel.For* w zewnętrznej pętli – okazała się najefektywniejsza. Programy o takiej konstrukcji były brane pod uwagę w wydajnościowej analizie, pokazanej w rozdziale 7.

Jednak przy tworzeniu rozwiązania brane były pod uwagę jeszcze inne warianty, w których np. zastosowano zrównoleglony odpowiednik w wewnętrznej pętli (po zmiennej *j*). Rozpatrzono nawet możliwość zrównoleglenia na najniższym możliwym poziomie granulacji przetwarzania, tj. podziału zrównoleglającego samej metody całkującej (z użyciem konstrukcji *Future* z Pfx). Dla metod wyższych rzędów (np. RK45) zrównolegleniu poddano wyznaczanie każdego z K_i ze wzoru 2. Nie dawało to jednak pożądanych rezultatów, tzn. przyspieszenia w stosunku do wersji sekwencyjnej. Potwierdziły się przypuszczenia autorów, że zadania zrównoleglane muszą być odpowiednio złożone obliczeniowo (czasochłonne), aby pozytywny efekt zrównoleglenia był istotnie zauważalny.

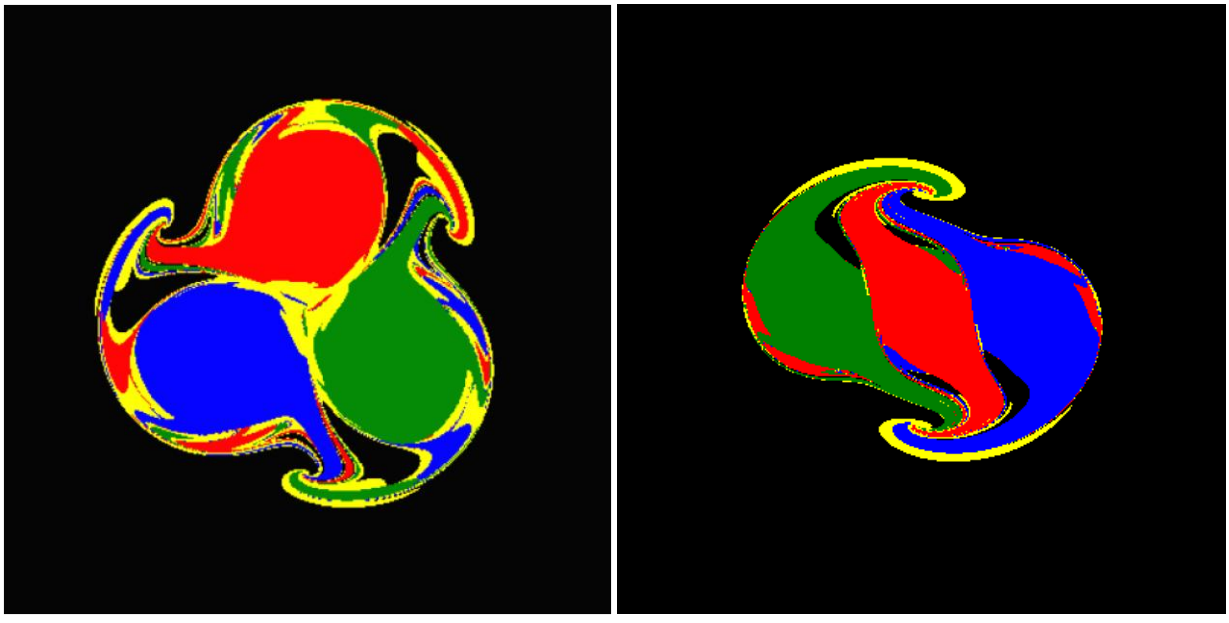
7. Rezultaty symulacji, porównanie wydajnościowe stworzonych rozwiązań

7.1. Wyniki eksperymentów

W wyniku przeprowadzonych symulacji dla przykładów 1 i 2 uzyskano charakterystyki ruchu ciała lekkiego, pokazane na rys. 2 i 3. Rysunek 2 dotyczy warunków początkowych, dla których ciało lekkie ma zerową prędkość początkową w nieinercjalnym układzie odniesienia. Rysunek 3 dotyczy warunków początkowych, dla których ciało lekkie ma niezerową prędkość początkową $v = [v_x\ v_y]$ w układzie nieinercjalnym, tzn. taką, że $v_x = y$ i $v_y = -x$. Takie wartości $v_x\ v_y$, odpowiadają zerowej prędkości początkowej w układzie inercjalnym (ciało lekkie spoczywa w układzie nieobracającym się).

Eksperymenty, których wyniki pokazano na rys. 2 i 3, przeprowadzono dla następujących wartości parametrów, określonych w podrozdziale 3: $D = 0{,}1$, $R = 3$, $T = 2\pi$, [*XMin*, *XMax*, *YMin*, *YMax*] = [-3 3 -3 3], [*ResolX*, *ResolY*] = [400 400]. Zastosowano metodę całkowania RK45. Zadany maksymalny bezwzględny błąd (tolerancja) całkowania $AbsTol = 10^{-7}$. Zadany maksymalny względny błąd całkowania $RelTol = 10^{-6}$.

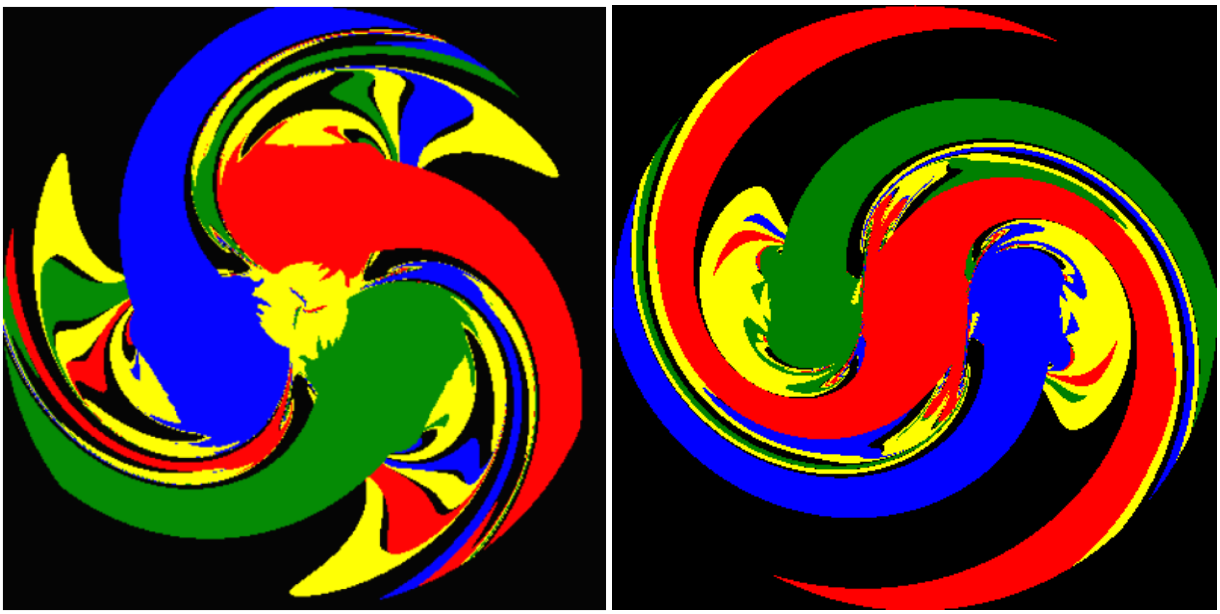
Parametry: *D*, *R*, *T* mają bezpośredni wpływ jakościowy na wynik eksperymentu (postać basenów przyciągania). Parametry: *T*, [*XMin*, *XMax*, *YMin*, *YMax*], [*ResolX*, *ResolY*], *AbsTol*, *RelTol* mają wpływ na czas realizacji eksperymentów.



a) b)

Rys. 2. Charakterystyka ruchu ciała lekkiego: a) – dla warunków początkowych [x y 0 0] dla układu trójkątnego z przykładu 1, b) – dla układu współliniowego z przykładu 2

Fig. 2. Characteristic of movement of weightless body: a) – for initial conditions [x y 0 0] for the triangular system from the 1st example, b) – for the collinear system from the 2nd one

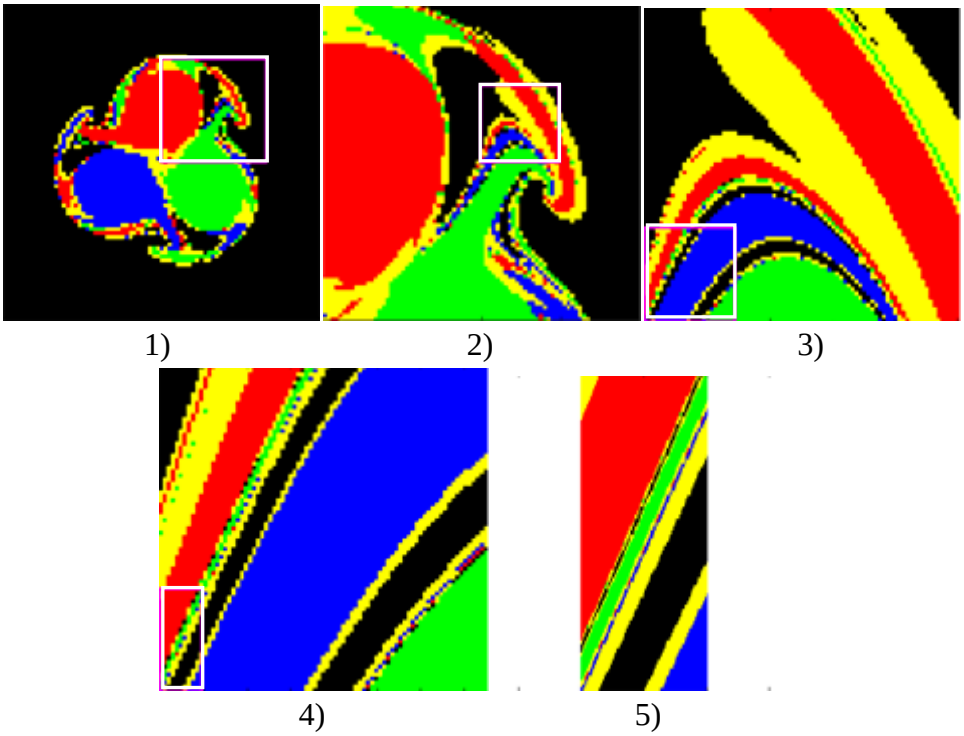


a) b)

Rys. 3. Charakterystyka ruchu ciała lekkiego: a) – dla warunków początkowych [x y y -x] dla układu trójkątnego z przykładu 1, b) – dla układu współliniowego z przykładu 2

Fig. 3. Characteristic of movement of weightless body: a) – for initial conditions [x y y -x] for the triangular system from the 1st example, b) – for the collinear system from the 2nd one

Baseny przyciągania są zbiorami niespójnymi – w dowolnie małym otoczeniu punktu, należącego do pewnych fragmentów brzegu danego basenu, znajdują się punkty należące do innych basenów przyciągania. Nieformalnie można stwierdzić, że uzyskane obszary przyciągania mają pewne własności obiektów zwanych fraktalami (ang. *gravitational fractals*). Na rys. 4, na kolejnych wykresach, pokazane są fragmenty basenów przyciągania dla zwiększającej się skali ekspozycji. Rysunek ten ilustruje wrażliwość rozwiązania na niewielką zmianę warunków początkowych (cecha układów chaotycznych) [14, 15], tzn. niewielka zmiana położenia początkowego skutkuje zmianą basenu przyciągania i zbieżnością do innego rozwiązania końcowego. Analiza matematyczna własności obszarów przyciągania omawianego rozwiązania wykracza poza zamierzony zakres artykułu.



Rys. 4. Ilustracja wrażliwości układu na warunki początkowe; obrazy charakterystyki ruchu w różnej skali powiększenia
Fig. 4. Illustration of system sensitivity to initial conditions; images of the movement characteristic at different scales of magnification

7.2. Specyfikacja sprzętowych konfiguracji testowych

Eksperymenty przeprowadzano na komputerach o różnej mocy obliczeniowej, w szczególności o różnej liczbie rdzeni w procesorach. Wykorzystano komputery o efektywnej liczbie rdzeni: 2, 4 i 8, tak jak zaprezentowano to w tabeli 1.

Tabela 1

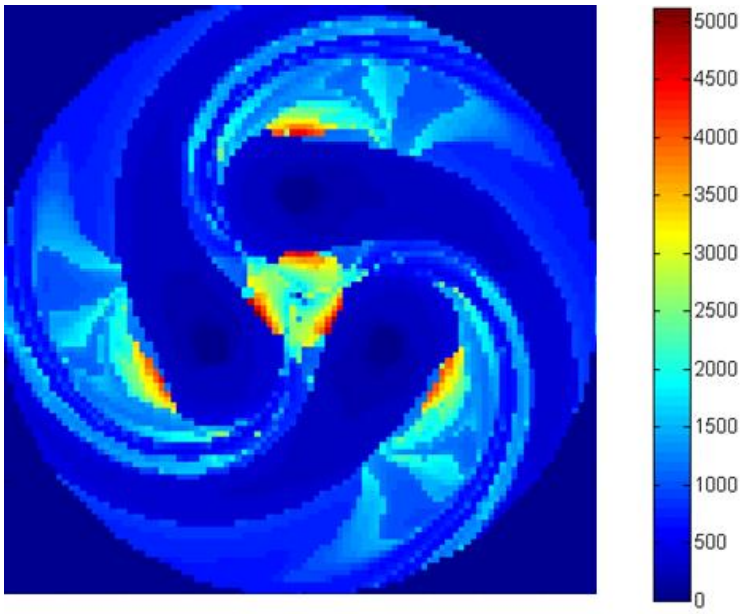
Parametry techniczne konfiguracji testowych

Nr	1	2	3
Procesor	Intel Pentium D	Intel Core 2 Quad Q6600	Intel Xeon E5420
Liczba rdzeni (częstotliwość taktowania)	2 (3.40 GHz)	4 (2.40 GHz)	2 x 4 (2.50 GHz) (dwa procesory czterordzeniowe)
Pojemność RAM	2 GB	2 GB	8 GB

7.3. Sposób pomiaru wydajności rozwiązań

Pojedyncze podzadanie symulacji dla konkretnego wektora warunków początkowych (dla pojedynczego punktu charakterystyki ruchu ciała lekkiego), tzn. realizacja eksperymentu, charakteryzuje się różną złożonością (rys. 5). Jako kryterium złożoności eksperymentu przy-

jęto sumaryczną liczbę wywołań funkcji określającej równania stanu układu (wartość końcowa zmiennej *cnt* np. z listingu kodu metod *move4* w rozdziale 6).



Rys. 5. Zależność liczby wywołań funkcji określającej równania stanu od początkowego położenia ciała lekkiego dla przykładu 1

Fig. 5. Dependency between a number of invoking the state equation function and an initial position of the weightless body for the 1st example

Dla niektórych wektorów warunków początkowych obliczenia przeprowadzane są dokładniej i dłużej, a wynika to z działania zastosowanej, efektywnej metody całkowania RK45 o zmiennym kroku całkowania. Takie zachowanie powoduje jednak, że utrudniony jest podział całego zadania symulacji na podzadania o równej lub zbliżonej złożoności w celu ich skutecznego zrównoleglenia (ale odpowiada rzeczywistym trudnościom, związanym z problemem równoważenia obciążenia w obliczeniowych środowiskach komputerów wielordzeniowych, wieloprocesorowych czy rozproszonych). Oczywiście, możliwe również jest zastosowanie prostszych, ale mniej efektywnych metod, np. Eulera czy RK4 (o stałym kroku całkowania), gdzie wspomniana trudność nie występuje.

Jednym z podstawowych celów pracy jest efektywnościowe porównanie zaproponowanych, następujących rozwiązań:

a) program w języku MATLAB (skrypt uruchamiany w środowisku MATLAB lub jego odpowiednik w postaci skompilowanej wersji EXE (oznaczony na rys. 6 i 8 nazwą *MATLAB*),
b) program jednowątkowy – przetwarzający sekwencyjnie, wykonany w technologii .NET (oznaczony nazwą *Sequence .Net n*a rys 6 i 7),
c) program wielowątkowy w technologii .NET, z wykorzystaniem rozszerzenia zrównoleglającego Pfx (oznaczony nazwą *Parallel .Net* na rys. 6, 7 i 8).

Programy symulacyjne ,wymienione w pkt. a, dla środowiska MATLAB zostały przygotowane w dwóch wersjach implementacyjnych – plik m-skryptowy uruchamiany (interpretowany) w środowisku MATLAB oraz wersja EXE (32-bitowy moduł EXE dla Windows), wygenerowana za pomocą kompilatora „MATLAB to C/C++". Nie stwierdzono, żeby wersja EXE była istotnie szybsza od wersji skryptowej. (Różnice czasu wykonania mieściły się zaledwie w zakresie do 1%).

Pomiarów czasów wykonania programów napisanych w języku C# dokonywano za pomocą obiektu *System.Diagnostics.StopWatch*. Pomiaru czasu wykonania skryptu w języku MATLAB dokonano za pomocą dwóch wbudowanych w to środowisko instrukcji: *tic*, wyznaczającej początek pomiaru czasu oraz *toc*, wyznaczającej koniec pomiaru. Wszystkie wymienione metody pozwalają na pomiar czasu z dokładnością do 1 ms.

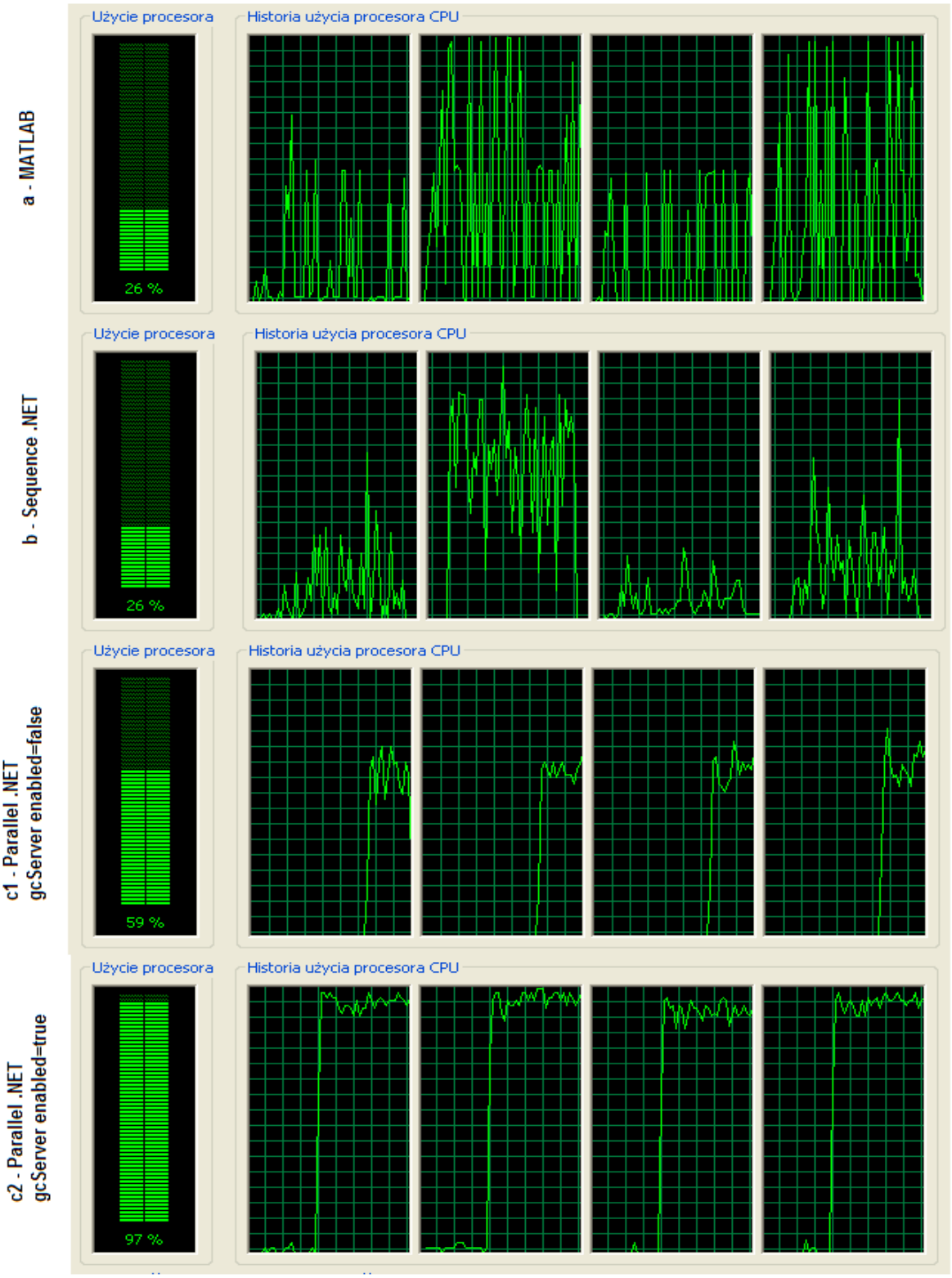
Dodatkowo zadbano, żeby różnice w ocenie efektywności nie wynikały z minimalnych różnic w sposobie działania metod całkowania, wynikających z różnej obsługi typu rzeczywistego (dotyczy metod istniejących w systemie MATLAB i tych nowozaimplementowanych w C#, które mają stanowić dokładne MATLABowe odpowiedniki). W tym celu czas symulacji został każdorazowo przeskalowany przez całkowitą liczbę wywołań równań stanu, jaka miała miejsce w przypadku każdej symulacji (okazuje się, że programy dla tych samych parametrów symulacji, uruchamiane w środowisku MATLAB, wywoływały funkcję określającą równania stanu o ok. 1% więcej razy niż programy dla .NET).

Ponieważ komputery, na których przeprowadzano symulacje, różniły się nie tylko architekturą, ale również szybkością procesorów (częstotliwość taktowania w tabeli 1), chcąc wyznaczyć charakterystykę szybkości realizacji zadania w funkcji liczby rdzeni, do porównania efektywności rozwiązań z pkt. a, b i c, nie zastosowano wartości bezwzględnego czasu wykonania zadania. Jako wskaźnik efektywności przyjęto natomiast stosunek czasu wykonania konkretnego rozwiązania sekwencyjnego do czasu wykonania wersji zrównoleglonej, wykonanej w technologii .NET (osobno dla każdej z konfiguracji sprzętowej z tabeli 1). Pozwala to na uniezależnienie od różnic szybkości procesorów (a tym samym szybkości rdzeni dla poszczególnych konfiguracji sprzętowych).

Wspólne środowisko uruchomieniowe (ang. *common language run time*) [7] platformy .NET może być konfigurowane za pomocą zbioru kilku opcji. Jedna z tych opcji umożliwia wybór trybu pracy menedżera pamięci (modułu odśmiecającego – ang. *garbage collector*) środowiska uruchomieniowego. Opcja ta może zostać zmieniona tylko przez dołączenie do programu wykonywalnego pliku konfiguracyjnego o rozszerzeniu .config. Dostępne wartości tej opcji to:

- tryb stacji roboczej, będący wartością domyślną: `<gcServer enabled="false"/>`; jest to ustawienie zalecane dla aplikacji uruchamianych na komputerach jednoprocesorowych,

- tryb serwera: `<gcServer enabled="true"/>`; tryb zalecany w przypadku uruchamiania na komputerach o dwóch lub więcej procesorach (lub procesorach wielordzeniowych).



Rys. 6. Skumulowane wykorzystanie procesora oraz obciążenie każdego z rdzeni
Fig. 6. Aggregated processor utilization and utilization of each processor cores

Rysunek 6 pokazuje wykorzystanie procesora przy realizacji zadania symulacji uruchomionego na komputerze czterordzeniowym (konfiguracja sprzętowa nr 2) dla różnych implementacji programu symulacji:

a) program uruchamiany w środowisku MATLAB,
b) program .NET – sekwencyjny,
c1) program .NET – zrównoleglony (z wykorzystaniem Pfx), pracujący w trybie stacji roboczej (wyłączona opcja gcServer),
c2) program .NET – zrównoleglony (z wykorzystaniem Pfx), pracujący w trybie stacji serwera (włączona opcja gcServer).

Rysunki 6c1 i 6c2 pokazują wydajniejszą pracę programu z włączoną opcją *gcServer* (wykorzystanie każdego z rdzeni prawie w 100%, pokazane na rys. 6c2). Skutkowało to skróceniem dla opcji c2 całkowitego czasu wykonania aż o 18% (% czasu liczony w stosunku do czasu wykonania dla opcji c1).

Chociaż wskaźnik sumarycznego wykorzystania procesora dla wersji MATLAB i sekwencyjnej wersji .NET był zbliżony (ok. 26%), to czasy realizacji zadania mocno różniły się na korzyść rozwiązania .NET (tabela 3, wiersz 2). Jedną z przyczyn może być większa liczba przełączeń wątków procesu pomiędzy rdzeniami w rozwiązaniu MATLAB niż w rozwiązaniu sekwencyjnej wersji .NET (rys. 6a i 6b). W pokazanym przypadku na rys. 6b program sekwencyjny .NET w większości przetwarzany był w drugim rdzeniu.

Wyniki eksperymentów, przedstawiane poniżej w podrozdziale 7.4, dotyczące uruchomień zrównoleglonej wersji dla .NET, uzyskane zostały przy ustawieniu menadżera pamięci w trybie serwera (jak na rys. 6c2).

7.4. Porównanie wydajnościowe rozwiązań

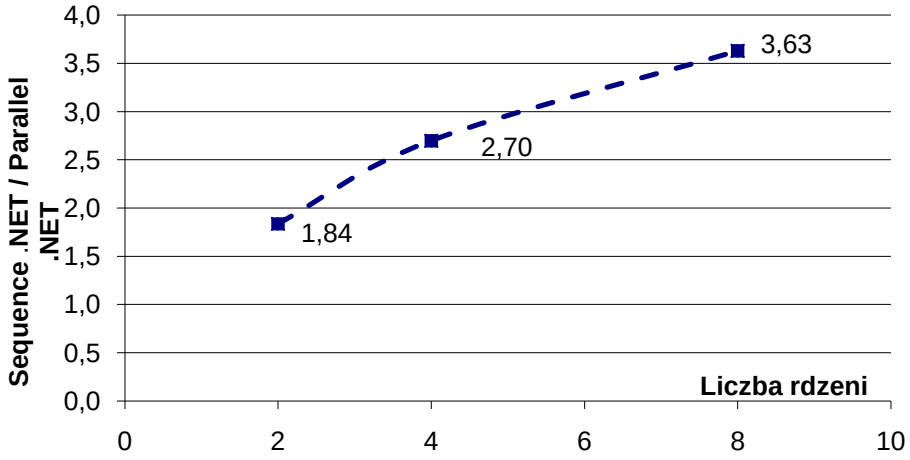
Wydajnościowe rezultaty przeprowadzonych eksperymentów dla różnych rozwiązań technologicznych i różnych testowych konfiguracji sprzętowych zostały pokazane w tabelach 2 i 3 oraz na rys. 7 i 8.

Tabela 2

Porównanie efektywności wersji wykonanych w technologii .NET

Liczba rdzeni	2	4	8
Przyspieszenie wersji zrównoleglonej .NET w stosunku do sekwencyjnej .NET	1,84	2,70	3,63

Wyniki pokazane w tabeli 2 zostały zilustrowanie na rys. 7.



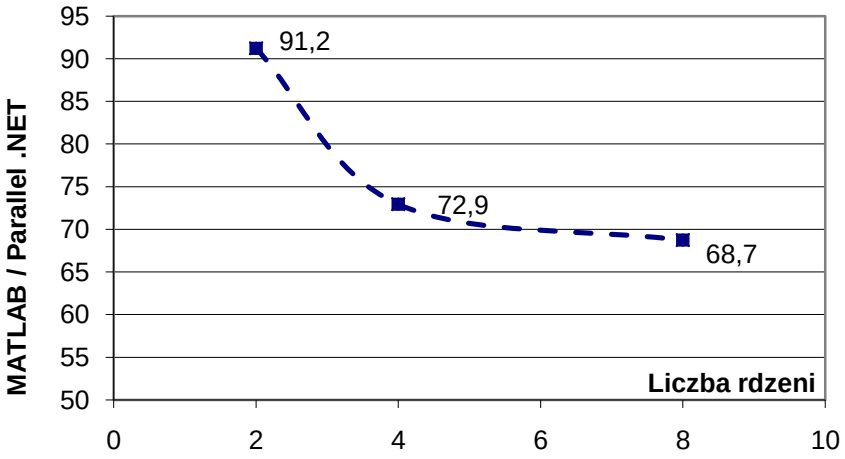
Rys. 7. Wskaźnik względnej efektywności (przyspieszenie) rozwiązań wykonanych w technologii .NET w funkcji liczby rdzeni procesora: stosunek czasu wykonania dla rozwiązania sekwencyjnego do czasu wykonania dla rozwiązania zrównoleglonego z użyciem Pfx

Fig. 7. Dependency between relative effectiveness coefficient (speedup) of .NET-based solutions and number of processor's cores: speedup quotient, while dividend is an executing time of the sequential solution and divisor is an executing time of the parallel solution based on Pfx

Tabela 3

Porównanie efektywności wersji wykonanych w technologii .NET z wersją dla systemu MATLAB

Liczba rdzeni	2	4	8
Przyspieszenie wersji sekwencyjnej .NET w stosunku do wersji MATLAB	49,68	27,04	18,94
Przyspieszenie wersji zrównoleglonej .NET w stosunku do wersji MATLAB	91,21	72,92	68,71



Rys. 8. Względna efektywność rozwiązania opartego na systemie MATLAB w stosunku do rozwiązania zrównoleglonego bazującego na .NET; zależność przyspieszenia rozwiązania zrównoleglonego od liczby rdzeni procesora

Fig. 8. Relative effectiveness of the MATLAB solution to the .NET-based parallel one; dependency between speedup and number of processor cores

Przyspieszenie zrównoleglonej wersji wykonanej w technologii .NET z użyciem Pfx względem wersji z użyciem systemu MATLAB w funkcji zastosowanej liczby rdzeni procesorów (ostatni wiersz tabeli 3) zostało zilustrowane na rys 8.

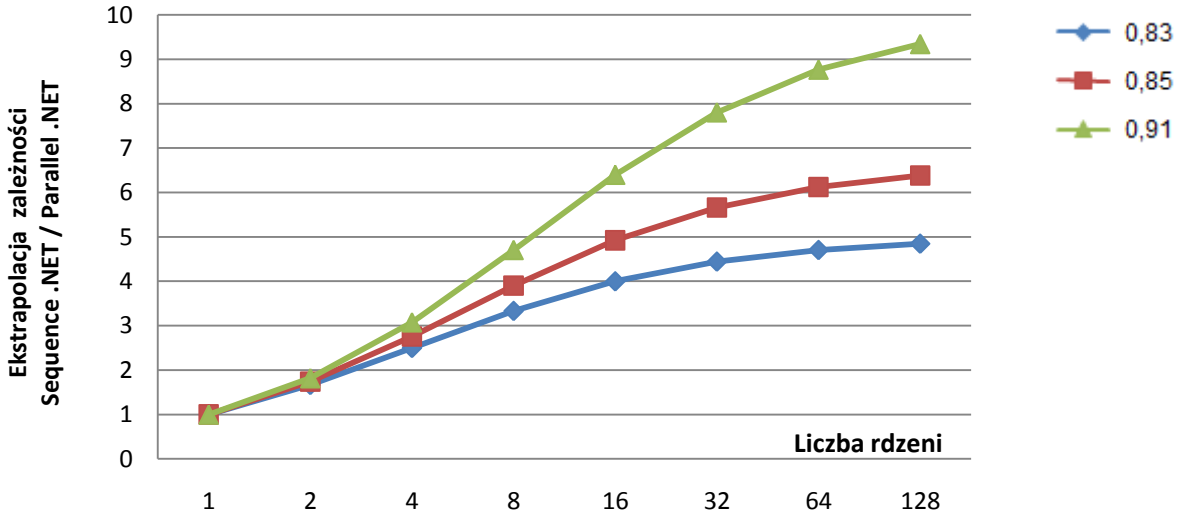
7.5. Ekstrapolacja przyspieszenia w funkcji liczby rdzeni – zastosowanie prawa Amdahla

Zgodnie ze sformułowaną przez Gene Amdahla zależnością [6, 7] można stwierdzić, że jeżeli zostanie przyspieszona n-krotnie część procesu zajmująca dla wersji sekwencyjnej $t * 100\%$ czasu wykonania całego procesu, to przyspieszenie k całego zrównoleglonego procesu (w stosunku do wersji sekwencyjnej) można wyrazić za pomocą prostego wyrażenia:

$$k = \frac{1}{(1-t) + \frac{t}{n}}. \tag{8}$$

Korzystając z przekształconej formuły (8), można wyznaczyć wartość t, czyli taką część zadania, w sensie czasu wykonania, która może podlegać zrównolegleniu, tzn.:

$$t = \frac{1-k}{\frac{k}{n} - k}. \tag{9}$$



Rys. 9. Ekstrapolacja przyspieszenia rozwiązania opartego na Pfx względem sekwencyjnego w funkcji liczby rdzeni na podstawie prawa Amdahla

Fig. 9. Extrapolation of dependency between speedup of the solutions based on Pfx related to the sequence solutions and number of processor cores according to Amdahl's law

Na podstawie wartości w tabeli 2 i wzoru 9 można obliczyć, że wartość wyrażenia $t * 100\%$ zawiera się w przedziale 83-91% (dla odpowiednich konfiguracji sprzętowych z 2, 4 i 8 rdzeniami procesorów, odpowiednie wartości wynoszą t = 0,91, 0,84, 0,83). Pozwala to na ekstrapolację wartości funkcji przyspieszenia rozwiązania zrównoleglonego względem sekwencyjnego dla dużych wartości liczby rdzeni procesorów (rys. 9). Niezależnie od przyjęcia któregoś z trzech oszacowań wartości t, wartość funkcji nigdy nie przekracza wartości

11,12. To ostatnie stwierdzenie wynika z asymptotycznej zależności, wyznaczonej dla najkorzystniejszej wartości $t = 0{,}91$:

$$\lim_{n\to\infty} \frac{1}{(1-t)+\frac{t}{n}}\Bigg|_{t=0{,}91} = 11{,}1(1). \tag{10}$$

Nasuwa to oczywiste wnioski co do nieuzasadnionych działań (nieopłacalność) zmierzających do znaczącego poprawienia efektywności omawianych zadań obliczeniowych przez zastosowanie maszyn o bardzo dużej liczbie rdzeni w procesorach, np. powyżej 32 rdzeni.

8. Rozwiązanie alternatywne z użyciem MALTAB Parallel Computing Toolbox

Na uwagę zasługuje fakt, że firma MathWorks Inc. dystrybuuje moduł programowy Parallel Computing Toolbox [19], przeznaczony do realizacji obliczeń rozproszonych w sieci komputerowej (dodatkowo z użyciem MALTAB Distributed Computing Server) oraz obliczeń z wykorzystaniem maszyn wieloprocesorowych lub wielordzeniowych. Rozwiązanie to głównie przeznaczone jest dla sieciowych obliczeń rozproszonych (tzw. przetwarzania w chmurze lub siatkowego – ang. *cloud or grid computing*), ale w zakresie przetwarzania „wielordzeniowego" może stanowić alternatywę do zaproponowanego rozwiązania wykorzystującego Parallel Extensions to .NET Framework. W szczególności, w tym Toolboxie zaimplementowana jest konstrukcja zrównoleglonej pętli – *parfor*, „podobna" do tej, zaimplementowanej w module Pfx.

Jest jednak kilka powodów, dla których prezentowane rozwiązanie może mieć przewagę nad MATLABowym rozwiązaniem firmowym.

Przyspieszenie rozwiązania w MATLABie z wykorzystaniem Parallel Computing Toolbox względem rozwiązania sekwencyjnego skaluje się podobnie (niewiele lepiej) do zależności przyspieszenia rozwiązania zrównoleglonego zrealizowanego w .NET względem sekwencyjnego (podobnie do rys. 7 dla liczby rdzeni 2 i 4). Dokładne wyniki względnego przyspieszenia wersji MATLAB (zrównoleglonej do sekwencyjnej) wynoszą: 1,6 dla 2 rdzeni, 3,2 dla 4 rdzeni, 3,8 dla 8 rdzeni (pomiary wykonane z użyciem wersji MATLAB Version 7.9.0.529 (R2009b) i Parallel Computing Toolbox Version 4.2). Natomiast wartości przyspieszenia zrównoleglonej wersji .NET względem zrównoleglonej wersji MATLAB są wysokie i wynoszą około: 57 dla 2 rdzeni, 23 dla 4 rdzeni, 18 dla 8 rdzeni.

Architektura Parallel Computing Toolbox zakłada uruchamiane niezależnych, dodatkowych procesów zwanych workerami (lub labami). W większości zastosowań efektywnościowo optymalna konfiguracja zakłada uruchomienie tylu workerów, ile dostępnych jest rdzeni procesora. Niestety, prawdopodobnie ze względów komercyjnych, samo użycie pakietu

Parallel Computing Toolbox skutkuje ograniczeniem do 8 workerów, co uniemożliwia dobrą skalowalność w zakresie efektywności rozwiązania powyżej 8 rdzeni. Takiego ograniczenia oczywiście nie ma zaproponowane rozwiązanie .NET, bazujące na Pfx.

Oczywiście, użycie Parallel Computing Toolbox wiąże się z koniecznością dodatkowego zakupu tego modułu, rozszerzającego standardowe środowisko MATLAB.

9. Podsumowanie

W artykule pokazano zalety zaproponowanego rozwiązania, wykorzystującego platformę .NET i moduł Parallel Extensions, w tworzeniu programów symulacji zachowania ciągłych układów dynamicznych. Takie podejście technologiczne wykorzystano m.in. do implementacji programów służących do modelowania cyfrowego ruchu układów ciał w polu grawitacyjnym.

Omawiana tematyka wpisuje się w nurt popularnej problematyki nowoczesnych sposobów tworzenia wydajnego oprogramowania, uwzględniającego rozwój architektur komputerów, wynikający z powszechnego wprowadzenia procesorów wielordzeniowych.

Artykuł pokazuje efektywnościową przewagę zrównoleglonego rozwiązania zadania modelowania cyfrowego, wykorzystującego infrastrukturę .NET, nad podejściem klasycznym, polegającym na wykorzystaniu popularnego systemu MATLAB (porównanie – tabela 3). W artykule pokazano, w jaki sposób zastosowanie prostych środków programowych dostępnych w Parallel Extensions pozwala na utworzenie oprogramowania zrównoleglonego, nieznacznie różniącego się budową kodu źródłowego od kodu programu przetwarzającego sekwencyjnie, ale jednocześnie szybszego od sekwencyjnego odpowiednika (rys. 7) i znacznie szybszego od rozwiązania opartego na MATLABie (rys. 8), (oczywiście w innych zastosowaniach zalety uniwersalnych systemów, takich jak MATLAB czy Octave są nie do podważenia).

Chociaż wyniki eksperymentów wykazały niewątpliwe zalety rozwiązania zrównoleglonego w stosunku do każdego innego rozważanego w niniejszej pracy, to w artykule pokazano również ograniczenia w zakresie skalowalności rozwiązania (rys. 9). W szczególności przedstawiono ograniczenia asymptotyczne wydajności rozwiązania, pokazujące nieskuteczność wykorzystywania procesorów z bardzo dużą liczbą rdzeni.

Zaproponowane rozwiązanie można uznać za uniwersalne (i łatwiejsze do zastosowania przez programistów) po uwzględnieniu możliwości wynikających z użycia programu translatora języka MATLAB na C#, co najmniej w zakresie tłumaczenia kodu źródłowego funkcji, opisujących równania stanu modelowanego układu [11]. Pozwala to na automatyczną generację części kodu programu w języku C# (na podstawie istniejącego odpowiednika

w MATLABie), w szczególności automatycznej translacji opisu modelu, czyli tej części, która głównie podlega zmianom.

BIBLIOGRAFIA

1. Skowronek M.: Modelowanie cyfrowe. Wydawnictwo Politechniki Śląskiej, Gliwice 2008.
2. Augustyn. D. R i inni: Zadania z modelowania cyfrowego. Praca zbiorowa pod redakcją M. Skowronka. Skrypt Pol. Śl. 2368. Wydawnictwo Politechniki Śląskiej, Gliwice 2005.
3. Wolfram Research. (2009). Runge-Kutta Method. http://mathworld.wolfram.com/Runge-KuttaMethod.html.
4. The MathWorks. (2010). MATLAB and Simulink for Technical Computing. http://www.mathworks.com.
5. John W. Eaton. Octave. (2010). http://www.gnu.org/software/octave.
6. Amdahl G.M.: Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967.
7. Amdahl's law – Wikipedia. (2010). http://en.wikipedia.org/wiki/Amdahl's_law.
8. Microsoft. Common Language Runtime Overview. (2009). http://msdn.microsoft.com/en-us/library/ddk909ch.aspx.
9. The Official Site of Visual Studio 2010. (2010). http://www.microsoft.com/visualstudio-/en-us.
10. Lambda Expressions (C# Programming Guide). (2010). http://msdn.microsoft.com/en-us/library/bb397687.aspx.
11. Augustyn D. R., Kunc S.: Moduł translacji języka MATLAB na C#, wspomagający tworzenie programów symulacji ciągłych układów dynamicznych, działających w środowisku uruchomieniowym .NET. Studia Informatica Vol. 3 No. 3 (91), Gliwice 2010.
12. Burton K. R.: .NET Common Language Runtime Unleashed, SAMS Publishing 2002.
13. Moszyński K.: Rozwiązywanie równań różniczkowych zwyczajnych na maszynach cyfrowych. WNT, Warszawa 1971.
14. Ott E.: Chaos w układach dynamicznych. WNT, Warszawa 1997.
15. Kudrewicz J.: Fraktale i chaos. WNT, Warszawa 1993.
16. Parallel Computing Developer Center. (2010). http://msdn.microsoft.com/pl-pl-/concurrency/default(en-us).aspx.
17. Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4. (2010). http://www.microsoft.com/downloads/
18. details.aspx?FamilyID=86b3d32b-ad26-4bb8-a3ae-c1637026c3ee&displaylang=en.

19. Articles on Parallel Programming with the .NET Framework 4. (2010). http://www-.microsoft.com/downloads/details.aspx?familyid=C3EA8FB5-650D-434B-A216-7E54C53965D1&displaylang=en.
20. Parallel Computing Toolbox – MATLAB (2010). http://www.mathworks.com/products-/parallel-computing.

Recenzent: Dr inż. Maciej J. Bargielski

Wpłynęło do Redakcji 15 czerwca 2010 r.

Abstract

Modern computer architecture is based on multicore processors. Recently, multicore machines became popular and commonly available. This is a reason why effective programs should become multi-core aware.

Parallelized programs should distribute concurrent tasks among processor cores. This should be possibly transparent to developers who should concentrate on processing algorithms rather than details of parallelization methods. One of the newest approach to solve solution this problem can be based on the Parallel Extensions to .NET Framework (Pfx) module.

The paper shows an effectiveness analysis of programs for simulation of continuous dynamical systems which are built with Parallel Extensions to .NET Framework. Modeling of movements of some bodies systems in a gravitational field is considered.

Advantages of parallel .NET-based programs – effectiveness and ease of use of the Pfx are shown. The paper presents the effectiveness comparison of the proposed parallelized solution to either MATLAB scripts or sequential single-thread .NET based programs (all run on machines with different numbers of processor cores). Scalability of the proposed solution, based on an extrapolation using Amdahl's law is considered too.

Adresy

Dariusz Rafał AUGUSTYN: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, draugustyn@polsl.pl.
Szymon KUNC: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, szymon.kunc@gmail.com.