STUDIA INFORMATICA 2010
Volume 31 Number 4A (92)

Dariusz R. AUGUSTYN
Politechnika Śląska, Instytut Informatyki

WYDAJNOŚĆ MECHANIZMÓW MODUŁU PARALLEL COMPUTING TOOLBOX SYSTEMU MATLAB W ZRÓWNOLEGLONEJ REALIZACJI SYMULACJI RUCHU UKŁADÓW CIAŁ W POLU GRAWITACYJNYM

Streszczenie. W pracy przedstawiono zrównoleglone programy symulacji wykonane w systemie MATLAB, pozwalające na ilustrację wpływu warunków początkowych na ruch ciała lekkiego w trójkątnym układzie trzech ciał ciężkich, uzyskanie zbioru torów złożonych układów N-ciał, prezentacje trajektorii ruchu wokół stabilnych punktów libracji Lagrange'a w ramach ograniczonego, kołowego problemu trzech ciał. Do implementacji wykorzystano mechanizmy modułu Parallel Computing Toolbox, m.in. takie jak: pętla zrównoleglona (parfor), komenda działająca zgodnie z koncepcją jednoczesnego uruchamiania tego samego kodu programu dla wielu danych (spmd), zadania wsadowe (jobs). Zastosowane metody zrównoleglenia symulacji pozwoliły na przyspieszenie realizacji zadań modelowania ruchu układów ciał w polu grawitacyjnym, uruchamianych na komputerach z procesorami wielordzeniowymi. W pracy dokonano porównania wydajności zaproponowanych programów symulacyjnych i użytych mechanizmów zrównoleglania obliczeń.

Słowa kluczowe: symulacja ruchu układów ciał w polu grawitacyjnym, programowanie w systemie MATLAB, Parallel Computing Toolbox, programowanie dla maszyn z procesorami wielordzeniowymi, efektywność i skalowalność zrównoleglonych programów symulacji

EFFECTIVENESS OF MECHANISMS OF PARALLEL COMPUTING TOOLBOX FOR MATLAB USED IN SIMULATIONS OF BODIES SYSTEM'S MOVEMENTS IN GRAVITATIONAL FIELD

Summary. The paper describes parallelized programs implemented in MATLAB designated for simulations of bodies movements in a gravitational field. They allow to illustrate hyper-sensivity a weightless body movement to initial conditions in the triangular system of 3 heavy bodies. Programs generate orbits for complex N-body systems. They present trajectories near stable libration points (Lagrangian points) for

the circular restricted three-body problem. Some mechanisms of Parallel Computing Toolbox were used in simulation program implementations, e.g.: parallel FOR-loop (parfor), statement based on the concept – single program multiple data (spmd) and jobs/tasks. Those methods of parallelization let speedup simulations executed on machines with multi-core processors. The paper presents experimental results that show effectiveness of proposed solutions and applied parallelization methods.

Keywords: simulation of movements of bodies systems in gravitational field, MATLAB programming, Parallel Computing Toolbox, multicore-aware software, effectiveness and scalability of parallel simulation programs

1. Wstęp

Pojawienie się komputerów z procesorami wielordzeniowymi spowodowało nieuchronny rozwój oprogramowania, pozwalającego na wykorzystanie mocy obliczeniowej wynikającej z takiej architektury. Pojawiła też się potrzeba środowisk wytwórczych, umożliwiających tworzenie programów, których zrównoleglone, niezależne fragmenty mogłyby być wykonywane jednocześnie na kilku rdzeniach procesora – ang. multicore-aware software. Przykładem takiego środowiska programowego jest system MATLAB wraz z modułem rozszerzającym Parallel Computing Toolbox [20].

Celem niniejszej pracy jest pokazanie środków programowych, wynikających z użycia Parallel Computing Toolbox, prowadzących do tworzenia zrównoleglonych wersji programów symulacyjnych dla systemu MALTAB. W artykule omówione będzie zastosowanie różnych mechanizmów zrównoleglenia, dostępnych dzięki Parallel Computing Toolbox, takich jak: pętla zrównoleglająca (*parfor*), blokowa komenda działająca wg modelu przetwarzania - jeden program wiele danych (*spmd*), zadania wsadowe (*jobs/ tasks*).

Mechanizmy zrównoleglenia użyte zostały do stworzenia programów, pozwalających na symulacje ruchu ciał w polu grawitacyjnym, działających efektywnie na maszynach wielordzeniowych. Programy symulacyjne służą do pokazania rozwiązań klasycznych, astronomicznych problemów z zakresu mechaniki nieba. W artykule rozpatrywane są następujące trzy przykłady zadań modelowania ruchu:

1) uzyskanie obrazu charakteryzującego ruch czwartego ciała lekkiego w trójkątnym układzie odniesienia trzech ciał ciężkich – ilustracja charakterystyki wrażliwości ruchu na zmianę warunków początkowych,
2) problem N-ciał – badanie ewolucji układu złożonego z dużej liczby ciał wzajemnie oddziałujących ze sobą (ewolucja układów typu galaktyka czy gromada gwiazd),
3) ograniczony kołowy problem ruchu trzech ciał – uzyskanie rodziny orbit ciała lekkiego w układzie odniesienia dwóch ciał ciężkich – prezentacja torów ciała lekkiego wokół trójkątnych punktów równowagi.

W artykule omówiono modele matematyczne, opisujące ww. problemy, co umożliwiło implementację programów symulacyjnych dla każdego z trzech przykładów.

W pracy rozważono sytuacje, w których zadanie główne daje się lub nie daje się podzielić na podzadania o jednakowej złożoności (przykład 2 – jednakowa złożoność podzadań, przykłady 1 i 3 – podzadania z różną złożonością).

Celem artykułu jest jakościowa (względem rodzaju mechanizmu zrównoleglenia) i ilościowa (względem liczby uruchomionych procesów workerów MATLABa, względem liczby rdzeni procesora) analiza wydajności stworzonych rozwiązań zrównoleglonych.

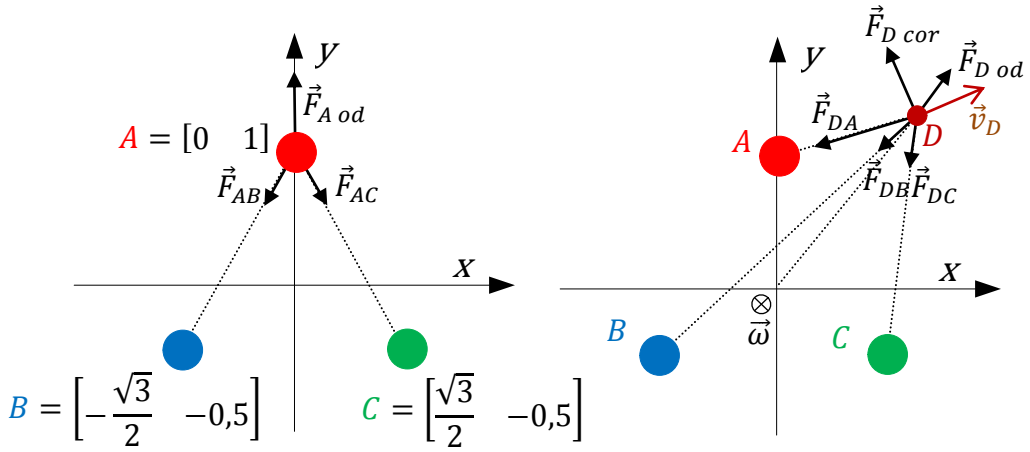
2. Analiza wrażliwości ruchu ciała lekkiego w nieinercjalnym układzie odniesienia trzech obracających się ciał ciężkich na zmianę warunków początkowych

2.1. Opis modelu ciągłego układu dynamicznego, koncepcja programu symulacji

Pierwszy przykład analizowanego układu ciał dotyczy płaskiego ruchu ciała lekkiego w pobliżu trzech obracających się ciał ciężkich. Ruch analizowany jest w nieinercjalnym układzie odniesienia, w którym ciała ciężkie pozostają nieruchome. Celem zadania jest uzyskanie portretu ilustrującego wpływ warunków początkowych na przebieg ruchu ciała lekkiego.

Ciała ciężkie umieszczone są w nieinercjalnym układzie odniesienia *xy* w wierzchołkach trójkąta równobocznego (rys. 1a). W układzie nieinercjalnym siła odśrodkowa $\vec{F}_{A\,od}$, działająca na nieruchome ciało ciężkie *A*, równoważona jest wypadkową sił grawitacji wynikających z oddziaływania z ciałami ciężkimi $\vec{F}_{AB}$ i $\vec{F}_{AC}$ (rys 1.a). Względem układu inercjalnego układ ciał ciężkich (oznaczonych przez *A*, *B*, *C*) obraca się wokół osi przechodzącej przez środek układu *xy* i prostopadłej do płaszczyzny rysunku. W układzie inercjalnym układ ciał ciężkich obraca się zgodnie z ruchem wskazówek zegara, tzn. wektor prędkości kątowej $\overrightarrow{\omega}$ jest zwrócony w kierunku przeciwnym do obserwatora.

Znikoma masa ciała lekkiego *D* nie wpływa na ruch ciał ciężkich. W układzie nieinercjanym na ciało lekkie działają następujące siły (rys. 1b): $\vec{F}_{DA}$, $\vec{F}_{DB}$, $\vec{F}_{DC}$ – siły przyciągania grawitacyjnego, wynikające z oddziaływania z ciałami ciężkimi, $\vec{F}_{D\,od}$– siła odśrodkowa, $\vec{F}_{D\,cor}$ – siła Coriolisa prostopadła do wektora prędkości $\vec{v}_D$.



a) b)

Rys. 1. Rozmieszczenie ciał ciężkich: a) siły działające na nieruchome ciało ciężkie *A* w nieinercjalnym układzie odniesienia, b) siły działające na poruszające się ciało lekkie *D* w nieinercjalnym układzie odniesienia

Fig. 1. Location of heavy bodies: a) forces acting on the non-moving heavy body *A* in the non-inertial reference frame, b) forces acting on the moving weightless body *D* in the non-inertial reference frame

Rezultatem wykonania programu symulacji będzie stworzenie portretu wpływu warunków początkowych (położenia ciała lekkiego) na dalszy przebieg toru. Rozmiary kulistych ciał ciężkich są jednakowe (zadana wartość średnicy *d*). Zadany jest maksymalny czas symulacji pojedynczego eksperymentu. Pojedynczy eksperyment polega na numerycznym wyznaczaniu toru ciała *D* i sprawdzeniu, czy miało miejsce jedno z pięciu poniżej wymienionych zdarzeń:

1) ciało lekkie *D* zderzyło się z ciałem ciężkim *A* (lub zostało umieszczone „wewnątrz" kulistego ciała *A*),
2) ciało lekkie *D* zderzyło się z ciałem ciężkim *B*,
3) ciało lekkie *D* zderzyło się z ciałem ciężkim *C*,
4) ciało lekkie *D* nie zderzyło się z żadnym z ciał ciężkich ani nie opuściło otoczenia układu ciał ciężkich, czyli nie oddaliło się ani razu powyżej zadanej odległości (*R*), liczonej od początku układu *xy*,
5) ciało lekkie *D* opuściło układ ciał ciężkich, tzn. oddaliło się co najmniej raz na odległość większą niż *R*, licząc od początku układu *xy*.

Wynikiem działania programu symulacji jest wykonanie obrazu (dla zadanej rozdzielczości pionowej *ResolX* i poziomej *ResolY*), ilustrującego historię ruchu ciała dla zadanych położeń ciała lekkiego w prostokątnym obszarze zainteresowania [*XMin*, *XMax*, *YMin*, *YMax*]. Dla każdego wybranego punktu z obszaru zainteresowania przeprowadzany jest pojedynczy eksperyment, po czym następuje klasyfikacja toru do jednej z pięciu wymienionych kategorii. W zależności od wystąpienia określonego zdarzenia punkt obrazu (startowe położenie ciała

D) kolorowany jest następująco: zdarzenie 1) – kolor czerwony, 2) – niebieski, 3) – zielony, 4) – żółty, 5) – czarny[1].

Płaski ruch ciała lekkiego może być scharakteryzowany czterowymiarowym wektorem stanu $xx = [x_D, y_D, v_{Dx}, v_{Dy}]$, gdzie x_D, y_D to zmienne stanu określające położenie, a v_{Dx}, v_{Dy} to zmienne stanu określające składowe prędkości ciała lekkiego. Dynamika ruchu opisana została równaniami stanu [1] – równaniami różniczkowymi rzędu pierwszego, gdzie lewą stronę każdego równania stanowi pochodna zmiennej stanu. Zakładając, że $\vec{F}_D = [F_{Dx},\ F_{Dy}]$ jest wypadkową siłą działająca na ciało *D*, korzystając z II zasady dynamiki Newtona, można znaleźć wektor przyspieszenia $\vec{a}_D = \frac{\vec{F}_D}{m_D}$, gdzie $\vec{a}_D = [\dot{v}_{Dx},\ \dot{v}_{Dy}]$ i m_D – masa ciała lekkiego. Stąd równania stanu mają następującą postać:

$$\begin{aligned} \dot{x}_D &= v_{Dx} \\ \dot{y}_D &= v_{Dy} \\ \dot{v}_{Dx} &= \frac{F_{Dx}}{m_D} \\ \dot{v}_{Dy} &= \frac{F_{Dy}}{m_D} \end{aligned} \tag{1}$$

Szczegóły dotyczące sposobu wyznaczania siły wypadkowej $\vec{F}_D = [F_{Dx},\ F_{Dy}]$ (suma wektorów: 3 sił grawitacji, siły odśrodkowej i siły Coriolisa) można znaleźć w pracy [5]. Sposób wyznaczania siły $\vec{F}_D$ jest bardzo zbliżony do sposobu wyznaczania siły wypadkowej w układzie dotyczącym ograniczonego zagadnienia ruchu trzech ciał, przedstawionym szczegółowo w rozdziale 3.

Numeryczne uzyskanie rozwiązania z wykorzystaniem języka MATLAB wymaga utworzenia M-funkcji (wyznaczającej wartości prawych stron równań stanu wg wzoru (1)) o następującej sygnaturze:

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

gdzie `dxx` – wektor pochodnych zmiennych stanu, `t` – skalar, czas – chwilowa wartość zmiennej niezależnej, `moveD` – dowolnie przyjęta nazwa M-funkcji, dotycząca omawianego przykładu.

Znalezienie numerycznego rozwiązania – wartości zmiennych stanu w dyskretnych chwilach czasu – odbywa się przez „scałkowanie” prawych stron równań stanu, za pomocą funkcji całkującej (ang. ODE solver – *Ordinary Differental Equations solver*) [4]. Postać wywołania funkcji całkującej jest następująca:

[*t*, *xx*] = *ODEsolver* (*M-function-rstate*, [*T0 Tmax*], *xx0*, *option*),

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

gdzie:

- *t* – wektor wartości dyskretnych chwil czasu,
- *xx* – macierz wartości zmiennych stanu w dyskretnych chwilach czasu; liczba kolumn jest równa liczbie zmiennych stanu, liczba wierszy odpowiada rozmiarowi wektora *t*,
- *ODEsolver* – nazwa funkcji całkującej MATLABa, wynikająca z przyjętej metody całkowania; w omawianym rozwiązaniu przyjęto funkcję *ode45* opartą na RK45 (met. Rungego-Kutty-Dormand-Prince – metoda jednokrokowa o zmiennym kroku całkowania),
- *M-function-rstate* – nazwa M-funkcji wyznaczającej wartości pochodnych zmiennych stanu; w omawianym rozwiązaniu jest to funkcja '*moveD'*,
- *T0*, *Tmax* – moment rozpoczęcia, zakończenia całkowania,
- *xx0* – wektor wartości początkowych zmiennych stanu (w chwili *T0*),
- *option* – opcje całkowania; w szczególności błąd bezwzględny całkowania (*AbsTol*) i błąd względny (*RelTol*).

Realizacja pojedynczego eksperymentu (dla wybranego punktu startowego) sprowadza się do wywołania funkcji całkowania równań stanu.

2.2. Opis sekwencyjnego programu symulacji

Program symulacji tworzy prostokątny obraz, będący charakterystyką ruchu ciała lekkiego dla zbioru punktów startowych. Sekwencyjny program symulacji (oznaczony nazwą ***seq***) ma następującą postać:

```
% Mapa kolorów; czerwony, zielony, niebieski, czarny, zółty w konwencji RGB
colormap([1 0 0; 0 1 0; 0 0 1; 0 0 0; 1 1 0]);

% Przedziały dokładności w kierunkach obu osi układu
dx = (XMax - XMin)/ResolX; dy = (YMax - YMin)/ResolY;

tic % Początek pomiaru czasu
for i=0:ResolX
   for j=0:ResolY
    % Wyznaczenie warunków początkowych zmiennych stanu:
    %[x0 y0] na podstawie i, j; [vx0 vy0] = [y0 -x0];
    x0 = XMin + i*dx ; y0 = YMin + j*dy; vx0 = y0 ; vx0 = y0 ; vy0 = -x0;

    % Wykonanie eksperymentu:
    [t, xx_result] = ode45 ('moveD', [0  TMax], [x0 y0 vx0 vy0], opt);

    % Klasyfikacja ruchu ciała lekkiego do jednej z 5-ciu kategorii,
    % na podstawie wartości położeń ciała xx_result(:,1)i xx_result(:,2) -
    % przypisanie odpowiedniego kodu zdarzenia do zmiennej end_event:
    % upadek na ciało A (end_event = 1), B (end_event=2) lub C (end_event=4,
    % opuszczenie sąsiedztwa ciał ciężkich (end_event =4),
    % żaden z powyższych 4 przypadków (end_event = 5).

    % Odnotowanie kodu zdarzenia w elemencie tablicy events_table:
    events_table(i+1,j+1) =  end_event;
  end
end
toc % Koniec pomiaru czasu
```

```
% Utworzenie obrazu - charakterystyki ruchu ciała lekkiego
image([XMin XMax],[YMin YMax], events_table);
```

W ramach ciała podwójnie zagnieżdżonej pętli następuje wyznaczenie wartości początkowych zmiennych stanu, wykonanie eksperymentu (wywołanie *ode45*) i klasyfikacja toru ruchu. Po opuszczeniu obu pętli na podstawie tablicy kodów zdarzeń (*events_table*) następuje utworzenie wynikowego obrazu (komenda *image*).

Samo tworzenie i wyświetlanie obrazu nie jest objęte pomiarem czasu wykonania programu (występuje poza zakresem wyznaczonym przez komendy *tic* i *toc*) i nie jest brane pod uwagę w testach wydajnościowych, których wyniki przedstawiono w podrozdziale 2.5. Podobnie postąpiono w przypadku pomiaru czasu zrównoleglonych wersji programów symulacji, pokazanych w podrozdziale 2.4.

W pokazanym przykładzie przyjęto, że początkowo ciało lekkie *D* spoczywa w układzie inercjalnym, czyli posiada pewną prędkość początkową w obracającym się układzie nieinercjalnym. Stąd też początkowe wartości wektora stanu w momencie rozpoczęcia eksperymentu są następujące $[x_{D0}, y_{D0}, v_{Dx0}, v_{Dy0}] = [x_{D0}, y_{D0}, y_{D0}, -x_{D0}]$, przyjmując, że wartość prędkości kątowej obrotu układu nieinercjalnego wynosi $\omega = 1$.

2.3. Wynik symulacji

Symulację (tj. uruchomienie programu sekwencyjnego albo każdego innego, zrównoleglonego, opisanego w podrozdziale 2.4) wykonano dla następujących wartości parametrów:

- parametry modelu: d = 0,2, R = 3, $\omega = 1$,
- parametry eksperymentu: metoda całkowania = RK45, $T_{\mathrm{Max}} = 2\pi$, $AbsTol = 10^{-7}$, $RelTol = 10^{-6}$,
- parametry programu symulacji: [*XMin XMax YMin YMax*] = [-3 3 -3 3], *ResolX* = 100, *ResolY* = 100.

Wyniki symulacji pokazane zostały na rys. 2.

Punkty startowe sklasyfikowane jednakowo (oznaczone tym samym kolorem) nazywane są basenami (obszarami) przyciągania. Nieregularne kształty obszarów przyciągania pokazują szczególną wrażliwość położenia początkowego na późniejszą historię ruchu ciała lekkiego (w niektórych rejonach niewielka zmiana położenia powoduje istotną zmianę w historii ruchu ciała). Oczywiście, omówienie takich chaotycznych zachowań prezentowanego układu dynamicznego wykracza poza zakres niniejszej pracy.

Rys. 2. Charakterystyka ruchu ciała lekkiego w nieinercjalnym układzie odniesienia z nieruchomymi ciałami ciężkimi

Fig. 2. The characteristic of movement of the weightless body in the non-inertial reference frame where heavy bodies are motionless

2.4. Opisy zrównoleglonych wersji programów symulacji

System MATLAB pozwala na wykorzystanie mechanizmów programowych umożliwiających efektywną wielowątkową realizację zadań symulacji z wykorzystaniem możliwości komputerów z procesorami wielordzeniowymi.

W czasie realizacji programu sekwencyjnego nie jest wykorzystywana pełna moc obliczeniowa maszyny z procesorem wielordzeniowym. Zakładając użycie maszyny *n*-rdzeniowej, wykorzystanie procesora jest mniej więcej na poziomie $\frac{1}{n}100\%$ (odpowiada to wykorzystaniu jednego rdzenia, chociaż nie zawsze tego samego, z powodu przełączania zadania pomiędzy rdzeniami). Odpowiednie użycie mechanizmów zrównoleglających może skutkować osiągnięciem pełnego wykorzystania mocy obliczeniowej[2], a tym samym efektywniejszą realizację całego zadania. Na rys. 3 pokazano wartości parametru wykorzystania procesora w przypadku uruchomienia programu sekwencyjnego oraz uruchomienia programu zrównoleglonego (użycie instrukcji *parfor*, jak w podrozdziale 2.4.1) przy zastosowaniu procesora 8-rdzeniowego.

[2] Oczywiście bezpośrednim celem zrównoleglania nie jest osiągnięcie 100% wykorzystania procesora, tylko uzyskanie przyspieszenia, mierzonego skróceniem czasu realizacji. Wykorzystanie jest tylko jednym z mierników sposobu zrównoleglenia. Zastosowanie mechanizmów zrównoleglających skutkuje m.in. realizacją czynności związanych z synchronizacją wątków, co powoduje, że przyspieszenie w stosunku do wersji sekwencyjnej nie będzie *n*-krotne, ale niższe (nawet przy 100% wykorzystaniu procesora).

Rys. 3. Skumulowane wykorzystanie procesora oraz wykorzystanie każdego z rdzeni podczas uruchomienia: a) programu sekwencyjnego, b) zrównoleglonego
Fig. 3. Aggregated processor utilization and utilization of each processor cores during an execution of: a) the sequential program, b) the parallel one

Dzięki zastosowaniu modułu rozszerzającego Parallel Computing Toolbox [20] zadanie symulacji nie jest realizowane w ramach pojedynczego procesu (tak jak ma to miejsce w przypadku wariantu sekwencyjnego). W zależności od przyjętej konfiguracji uruchamiane są dodatkowe procesy, zwane workerami lub labami. Zadanie symulacji może zostać podzielone na podzadania za pomocą jednego z opisanych niżej mechanizmów. Podzadania są rozdystrybuowywane pomiędzy procesy workerów.

Maksymalną liczbę dostępnych workerów użytkownik może ustawić z poziomu menu głównego programu MATLAB (opcja: *Parallel*/*Configuration Manager*/wybór konfiguracji typu *local*/ zakładka *Schedule*/ pole *Number of workers available to Schedule*). Niestety, prawdopodobnie względy komercyjne spowodowały, że liczba ta nie może przekraczać czterech dla „starszego” systemu MATLAB 2007B i MATLAB Distributed Computing Engine 3.2 oraz ośmiu dla MATLAB 2009B i Parallel Computing Toolbox 4.2. To ostatnie ograniczenie (dla nowszej wersji systemu MATLAB) uniemożliwia efektywne, pełne zrównoleglenie programów uruchamianych na lokalnym komputerze, wyposażonym w procesor z liczbą rdzeni powyżej ośmiu.

Uruchomienie puli workerów (powołanie `nw` procesów) odbywa się za pomocą komendy:

```
matlabpool open local nw;
```

gdzie `local` jest nazwą konfiguracji, a `nw` jest liczbą uruchamianych workerów[3] (co najwyżej równą maksymalnej, zadeklarowanej liczbie workerów dla konfiguracji `local`). Zakończenie puli procesów workerów odbywa sie przez wykonanie komendy:

[3] Wykonanie komendy spowoduje uruchomienie `nw` procesów workerów, niezależnie od działającego głównego procesu MATLABa, zwanego procesem klienta.

```
matlabpool close;
```

2.4.1. *Proste zrównoleglenie zewnętrznej pętli z użyciem instrukcji parfor*

Pierwsza zrównoleglona wersja programu symulacji wynika z zastosowania zrównoleglenia w stosunku do zewnętrznej pętli (użycie komendy *parfor*). W związku z tym rozwiązanie oznaczone zostało jako ***par_for_outer***. Kod programu ma następującą postać:

```
%  ...
parfor (i = 1 : ResolX+1)
   local_events_vect = zeros(1, ResolY+1);
   for j = 1 : ResolY+1
     % ...
     [t, xx_result] = ode45 ('moveD', [0  TMax], [x0 y0 vx0 vy0], opt);
     % ...
     % Odnotowanie kodu zdarzenia w elemencie zmiennej typu „sliced output”
     % o nazwie local_events_vect:
     local_events_vect(j) =  end_event;
  end
  % Utrwalenie wektora w wierszu zmiennej events_table
  events_table(i,:) =  local_events_vect;
end
% ...
```

gdzie elementy kodu tożsame z rozwiązaniem sekwencyjnym oznaczono zakomentowanym symbolem wielokropka - `% ...`)

Dla każdej wartości *i* ciało pętli (w tym w całości pętla wewnętrzna „po” *j*) wykonywane jest w niezależnym podzadaniu. Podzadania rozdystrybuowane są pomiędzy wszystkie uruchomione wcześniej procesy workerów. Każde podzadanie wykonywane jest niezależnie, a kolejność realizacji iteracji zewnętrznej pętli dla poszczególnych wartości *i* nie jest zdeterminowana.

Koncepcja wykorzystania *parfor* zakłada wprowadzenie pewnych rodzajów zmiennych i skutkuje pewnymi ograniczeniami ich stosowania, co szczegółowo opisane zostało w [6] w sekcji *Classification of Variables*. Dostosowanie się do tych wymogów spowodowało użycie zmiennej *local_events_vect* - zmiennej typu warstwowego (ang. *sliced variable*), podzielonej na segmenty (warstwy), które mogą być przetwarzane niezależnie przez uruchomione workery.

2.4.2. *Podział zadania symulacji z dokładnością do liczby workerów z wykorzystaniem instrukcji pętli zrównoleglającej parfor*

Innym wariantem zastosowania zrównoleglonej pętli *parfor* jest użycie tej konstrukcji do podziału zadania symulacji według liczby uruchomionych workerów (zewnętrzna pętla iterowana „po” numerze workera). W związku z tym rozwiązanie oznaczone zostało jako ***par_for_workes.*** Przy takim podejściu uzyskuje się mniejszą (w stosunku do ***par_for_outer***) granulację podziału zadania symulacji (przy oczywistym założeniu, że rozdzielczość *ResolX*

jest większa od liczby workerów). Odpowiedni kod programu symulacji ma następującą postać:

```
%  ...
%  Pobranie liczby aktualnie uruchomionych procesów workerów
totalWorkers = matlabpool ('size');

parfor (nWorker =1 : totalWorkers)
  %  Wyznaczenie zakresu wierszy events_table dla bieżącego podzadania,
  %  wynikający z bieżącego numeru workera.
  %  Zakłada się „możliwie równy” podział tablicy dla poszczególnych workerów
  r = ResolX+1 ;
  nlo = floor((r * (nWorker -1))/ totalWorkers +1) ;
  nhi = floor((r * (nWorker))/ totalWorkers) ;

  %  Zerowanie lokalnej (dla bieżącego workera) tablicy kodów zdarzeń
  events_table = zeros(ResolX+1, ResolY+1);

  for i = nlo:nhi
     for j=1:ResolY+1
       % ...
       [t, xx_result] = ode45 ('moveD', [0  TMax], [x0 y0 vx0 vy0], opt);
       % ...
       % Odnotowanie kodu zdarzenia w elemencie (i,j) tablicy events_table,
       % ale dla indeksu i jedynie w zakresie od nlo do nhi:
       events_table(i,j) =  end_event;
     end
  end
  % Umieszczenie wyników w trójwymiarowej tablicy,
  % pierwszy wymiar indeksowany bieżącym numerem workera
  worker_events_table(nWorker,:)= events_table;
end % parfor

% Scalanie wyników wypracowanych przez poszczególne workery
all_workers_events_table(:,:) = worker_events_table (1,:,:);
for nWorker = 2 :totalWorkers
  temp (:,:) = worker_events_table (nWorker, :,:);
  all_workers_events_table = all_workers_events_table + temp ;
end
% Tworzenie obrazu
image([XMin XMax],[YMin YMax], all_workers_events_table);
```

2.4.3. Wykorzystanie komendy spmd z zastosowaniem pionowego podziału macierzy danych pomiędzy workery

W systemie MATLAB 2009B z rozszerzeniem Parallel Computing Toolbox 4.2 dostępny jest mechanizm zrównoleglenia oparty na koncepcji przetwarzania – jeden program, wiele danych (ang. *single program multiple data*). W celu jego użycia należy posłużyć się strukturalną komendą *spmd*, definiującą blok ciągu instrukcji, który może być przetwarzany równolegle [7].

Skuteczne wykorzystanie *spmd* wymaga odpowiedniego przygotowania danych – skorzystania z mechanizmu partycjonowania macierzy [9]. Użycie tego mechanizmu – wykorzystanie tzw. tablic rozproszonych (ang. *distributed table*) – zostanie zilustrowane przez zastosowanie komendy *pmode* [8] (ang. *interactive parallel execution mode*). Uruchomienie *pmode* otwiera konsolę Parallel Command Window. Przez tę konsolę można wydawać polecenia

jednoczesnego wykonania jednej komendy dla wielu workerów i obserwować ich niezależne działania w oddzielnych oknach. Poniżej pokazano przykładową sekwencję instrukcji w ramach sesji z konsolą programu MATLAB:

```
% Przebieg sesji w głównej konsoli MATLABa (program klienta)

% Utworzenie macierzy o rozmiarze 4x7
A = [1:7; 1:7; 1:7; 1:7]
% Uruchomienie interaktywnego równoległego trybu pracy z automatycznym
% wystartowaniem 4 workerów - labów (zgodnie z konfiguracją 'local')
pmode start local 4
% Przesłanie z przestrzeni klienta do każdego z czterech workerów
% kopii macierzy A
pmode client2lab A 1:4
% Przejście do okna Parallel Command Window
```

oraz sesji z konsolą Parallel Command Window:

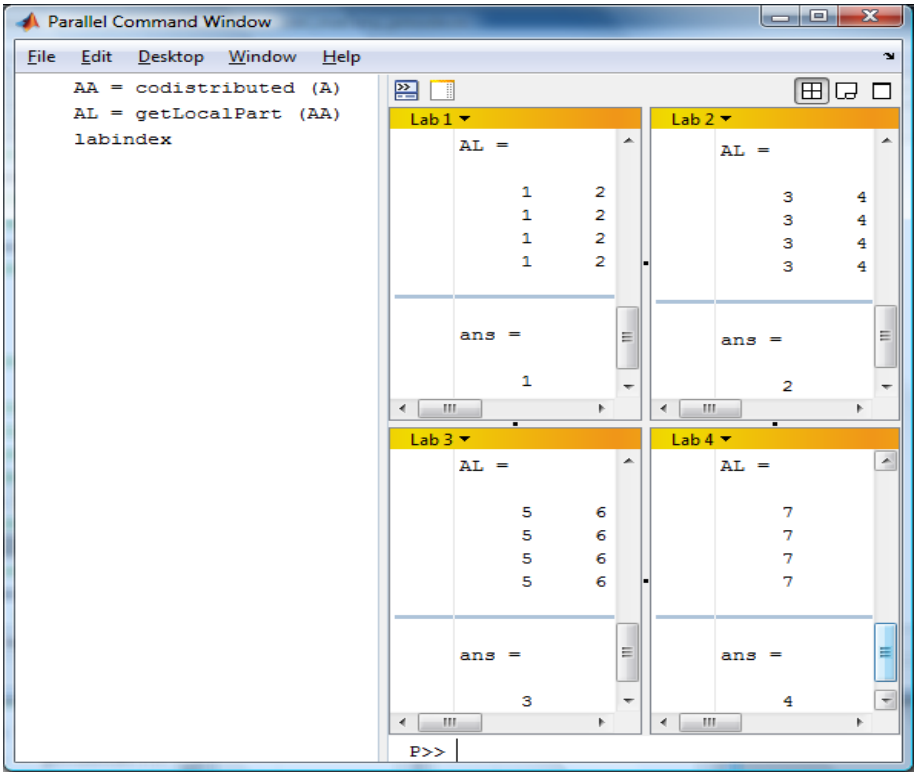
```
% Przebieg sesji w konsoli w interaktywnym, równoległym trybie pracy

% Utworzenie zmiennej opisującej podzieloną (rozdystrybuowaną) macierz A.
% Zastosowano domyślny sposób dystrybucji.
% Podział macierzy wg. 2 wymiaru tzn. kolumn,
% Przydzielona możliwie jednakowa liczba kolumn dla każdego workera.
AA = codistributed (A)
% Pobranie (utworzenie) lokalnej partycji macierzy w każdym workerze
AL = getLocalPart (AA)
% wyświetlenie numeru bieżącego workera
labindex
%  Operacja potęgowania każdego elementu tablicy, wykonana równolegle
%  w każdym workerze (możliwe, że jednocześnie)
AL = AL.^2
%  Uzyskanie zmiennej opisującej podział (dystrybucję) macierzy AA
%  - pobranie uchwytu dystrybutora macierzy AA
d = getCodistributor(AA)
% Wyświetlenie wymiaru wg którego nastąpił podział macierzy
d.Dimension  % ans = 2 - drugi wymiar macierzy (kolumny)
% Wyświetlenie wielkości podziałów dla każdego z workerów
% (po ile kolumn kolejno na każdy z czterech workerów)
d.Partition % ans = [2 2 2 1] - podział 7 kolumn macierzy A na 4 workery
% Konsolidacja lokalnych macierzy AL na podstawie uchwytu dystrybutora
AA = codistributed.build(AL, d)
% Budowa dwuwymiarowej wynikowej macierzy (w przestrzeni roboczej każdego
% workera), zawierającej dane z każdej dwuwymiarowej macierzy lokalnej AL
AResult = gather (AA)
% Przesłanie macierzy wynikowej z przestrzeni roboczej dowolnego workera
% (tutaj z workera numer 1) do przestrzeni roboczej klienta
pmode lab2client AResult 1
% Zakończenie sesji PMODE
exit
```

Działania te prowadzą do zrównoleglonej operacji podnoszenia do kwadratu elementów macierzy *A*:

- macierz *A*, zdefiniowana w programie głównym (proces klienta), zostaje rozdystrybuowana pomiędzy workery,
- wykonane zostają operacje na lokalnych fragmentach macierzy (macierze *AL*) w ramach każdego z czterech uruchomionych workerów,
- następuje scalenie fragmentów (utworzenie macierzy *AResult*) i przesłanie do klienta.

Rysunek 4 pokazuje widok konsoli Parallel Command Window w trakcie realizacji instrukcji przedstawionych na ostatnim listingu.



Rys. 4. Przykładowa sesja *pmode* – praca w interaktywnym trybie równoległej realizacji komend
Fig. 4. Example of a *pmode* session – working in the interactive parallel execution mode

Program realizujący zrównoleglona symulację ruchu w polu grawitacyjnym z użyciem komendy *spmd* ma następującą postać:

```
%  ...
%  globalna tablica kodów zdarzeń, utworzona w ramach programu głównego:
full_events_table = zeros (ResolX+1, ResolY+1);

spmd
  %  Dystrybucja tablicy kodów zdarzeń
  dist_events_table = codistributed (full_events_table);
  %  Uzyskanie lokalnej (dla danego workera) tablicy kodów zdarzeń
  local_events_table = getLocalPart (dist_events_table);
  %  Pobranie rozmiarów (liczba wierszy i kolumn) lokalnej tablic kodów zdarzeń
  sz =  size (local_events_table); rows = sz(1); cols = sz(2);

  %  Uzyskanie uchwytu do obiektu dystrybutora macierzy
  d = getCodistributor(dist_events_table);

  %  Uzyskanie wektora podziału kolumn tablicy full_events_table
  %  przydzielonych do kolejnych workerów
  Partitions =  d.Partition;

  %  Wyznaczenie przesunięcia (offsetu) w globalnej tablicy kodów zdarzeń
  %  dla bieżącego workera (labindex – numer bieżącego workera)
  if labindex == 1
     ofs =0;
  else
     ofs = sum (Partitions(1:labindex-1));
```

```
    end

    %  Przetwarzanie lokalnej tablicy kodów zdarzeń o rozmiarze rows x cols
    for i = 0:rows-1
       for j= 0:cols-1

          % Wyznaczenie warunków początkowych:
          % [x0 y0] na podstawie i, j oraz ofs dla danego workera
          x0 = XMin + i*dx ; y0 = YMin + (j + ofs)*dy;

          % ...
          [t, xx_result] = ode45 ('moveD', [0  TMax], [x0 y0 vx0 vy0], opt);
          % ...
          % Odnotowanie kodu zdarzenia w elemencie lokalnej tablicy kodów zdarzeń
          local_events_table(i+1,j+1) =  end_event;
       end
    end
    % Scalanie wyników wypracowanych przez poszczególne workery
    % (z uwzględnieniem obiektu dystrybutora d)
    full_events_table = codistributed.build(local_events_table, d);
end % spmd

% Tworzenie obrazu
image([XMin XMax],[YMin YMax], full_events_table);
```

Obsługa tablicy kodów zdarzeń (*full_events_table*) rozdzielona została pomiędzy workery za pomocą domyślnego sposobu dystrybucji macierzy (najprostsze użycie funkcji *codistributed* [9]). W związku z tym omawiane rozwiązanie oznaczono jako ***spmd_codistr***. Taki sposób dystrybucji prowadzi do pionowego (kolumnowego) podziału macierzy pomiędzy workery.

2.4.4. Wykorzystanie komendy spmd z zastosowaniem losowej dystrybucji elementów macierzy danych pomiędzy workery

W poprzednim podrozdziale zastosowano rozwiązanie problemu z użyciem komendy *spmd* z domyślnym, kolumnowym podziałem pomiędzy workery obszaru punktów startowych (domyślna „dystrybucja" tablicy kodów zdarzeń). Tablica kodów zdarzeń została podzielona na spójne, prostokątne obszary. Podział taki jest nietrudny w realizacji, ale może nie zapewniać zrównoważenia obciążenia workerów, biorąc pod uwagę, że złożoność obliczeń jest różna dla każdego punktu startowego. Złożoność procesu całkowania równań stanu może być znacząco różna dla poszczególnych punktów startowych. Złożoność eksperymentów, mierzona liczbą wywołań równań stanu w zależności od punktu startowego, została pokazana w [5], rys. 5. Można łatwo zauważyć, że obsługę obszarów charakteryzujących się istotnie większą złożonością, należałoby równomiernie rozdzielić pomiędzy workery.

Statystycznie lepsze zrównoważenie obciążenia można uzyskać przez zastosowanie losowego przydziału poszczególnych punktów startowych do workerów.

Kod źródłowy programu opartego na takim założeniu, oznaczonego przez ***spmd_monte_carlo*** został pokazany poniżej:

```
%  ...
%  Pobranie liczby aktualnie uruchomionych procesów workerów
totalWorkers = matlabpool ('size');
```

```
%  Utworzenie wstępnej zawartości tablicy kodów zdarzeń.
%  Przypisanie każdemu elementowi tablicy losowo wygenerowanego numeru workera
%  Inicjalnie, do każdego elementu wpisywana są ujemna liczba, oznaczająca
%  „ujemny” numer workera, który ma przetworzyć bieżący punkt startowy.
%  Po przetworzeniu, w elemencie będzie wartość nieujemna ze zbioru {0,1,...,5},
%  określająca albo dodatni kod zdarzenia, dotyczącego historii ruchu ciała
%  lekkiego albo wartość 0, oznaczająca fakt, że element celowo nie został
%  przetworzony przez bieżący worker.
events_table = -(floor(rand (ResolX+1, ResolY+1) * totalWorkers) +1);

spmd

  for i = 0:ResolX
     for j= 0:ResolY
        % Każdy z workerów posiada własną kopię tablicy events_table.
        % (events_table staje się obiektem typu sliced variable)
        if events_table (i+1,j+1) == -labindex
            % Wykrycie, że bieżący element (i,j) ma być przetwarzany
            % przez bieżący worker

            % ...
            [t, xx_result] = ode45 ('moveD', [0  TMax], [x0 y0 vx0 vy0], opt);
            % ...
            % Odnotowanie kodu zdarzenia w elemencie tablicy kodów zdarzeń.
            % Odnotowanie kodów zdarzeń ma miejsce tylko dla wybranych
            % elementów tablicy events_table
            % (zgodnie z inicjalnym, losowym rozkładem).
            events_table(i+1,j+1) =  end_event;
        else
            % Wykrycie, że bieżący element (i,j) NIE ma być przetwarzany
            % przez bieżący worker (przypisanie zera)
            events_table(i+1,j+1) = 0;
        end % if
     end
  end
end % spmd

% Pobranie liczby warstw dla obiektu events_table
% (events_table traktowana jako zmienna sliced variable).
% Liczba warstw jest równa liczbie workerów.
nslices = length (events_table);

% Scalanie wyników na podstawie sumy lokalnych kopii events_table,
% wypracowanych przez workery.
% Dostęp do „lokalnej kopii” events_table poprzez odniesienie do warstwy obiektu
% z wykorzystaniem konstrukcji {}, indeksującej zmienne typu cell array
result_events_table = zeros (ResolX+1, ResolY+1);
for k=1:nslices
    result_events_table = result_events_table + events_table{k};
end

% Tworzenie obrazu
image([XMin XMax],[YMin YMax], result_events_table);
```

W omawianym rozwiązaniu, inicjalnie, dla każdego elementu tablicy kodów zdarzeń (*events_table*) została wylosowana liczba pseudolosowa (rozkład dyskretny równomierny) z przedziału od 1 do maksymalnej liczby uruchomionych workerów. Do elementu tablicy wpisywana jest liczba ze znakiem minus, określająca numer workera, który ma przetworzyć dany punkt startowy. W ramach realizacji *spmd* każdy z workerów, przetwarzając swoją lokalną kopię tablicy punktów startowych, czyli lokalną tablicę kodów zdarzeń (*events_table*),

albo wykonuje eksperyment i ustawia kod zdarzenia (przetwarzanie punktu) albo tylko wpisuje 0 (punkt ma być przetworzony przez inny worker). Lokalne kopie (tablice typu *sliced variable*) są ostatecznie scalane w ramach *result_events_table*.

2.5. Rezultaty eksperymentów i interpretacja wyników – porównanie efektywności zastosowanych zrównoleglonych rozwiązań

Testy wydajnościowe przeprowadzono dla różnych konfiguracji sprzętowych, tzn. z użyciem komputerów z procesorami 2-, 4- i 8-rdzeniowymi, których specyfikację pokazano w tabeli 1.

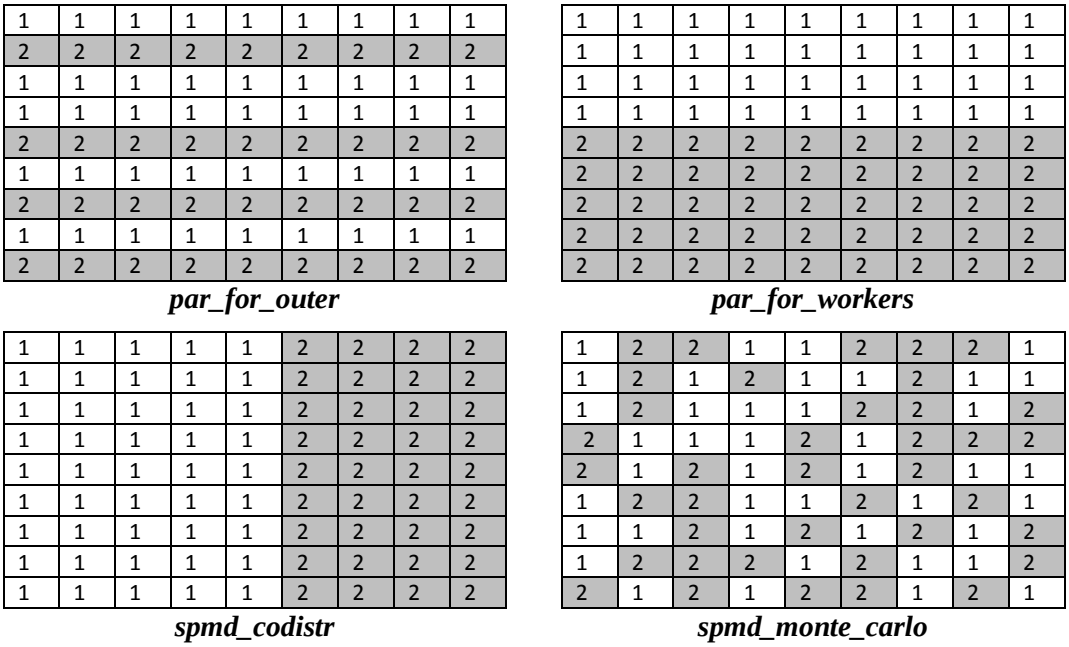
Tabela 1

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)
Pamięci RAM	2 GB	2 GB	8 GB

W omówionych rozwiązaniach zrównoleglonych, wykorzystujących komendy *parfor* albo *spmd*, zastosowano różne metody podziału tablicy kodów zdarzeń (podział obszaru punktów startowych) pomiędzy workery. Sposoby podziału przykładowej tablicy o rozmiarze 9×9 dla konfiguracji, w której użyto 2 workery, przedstawiono na rys. 5.

W rozwiązaniu ***par_for_outer***, ponieważ zrównolegleniu poddana jest zewnętrzna pętla, następuje losowy przydział workera do pojedynczego wiersza tablicy kodów zdarzeń. W ***par_for_workers*** następuje przydział całej grupy wierszy (program dokonuje podziału tak, by liczby wierszy w grupie wierszy były możliwie równe). W ***spmd_codistr*** następuje kolumnowy podział tablicy na możliwie równe prostokątne obszary (grupy kolumn) i przydział tych obszarów do konkretnych workerów. W ***spmd_monte_carlo*** następuje losowy przydział pojedynczych elementów tablicy do workerów (największa granulacja podziału). Sposób przydziału elementów tablicy kodów do workerów ma istotny wpływ na wydajność danego rozwiązania.



Rys. 5. Metody przydziału elementów tablicy kodów zdarzeń do workerów dla konfiguracji z dwoma workerami

Fig. 5. Methods of distribution of events table elements between workers for a configuration with two workers

Wyniki wydajnościowe symulacji zaprezentowane zostały w tablicy 2. Kolumny 4-7 przedstawiają wartości przyspieszeń (uśredniony stosunek czasu wykonania symulacji w wersji sekwencyjnej rozwiązania (***seq***) do czasu wykonania wersji zrównoleglonej).

Tabela 2

Przyspieszenie wersji zrównoleglonych w stosunku do sekwencyjnej wersji programu symulacji w zależności od liczby rdzeni procesora oraz liczby uruchomionych workerów

	1	2	3	4	5	6	7
	Liczba rdzeni procesora	**Liczba workerów**	**Średnie wykorzystanie procesora [%]**	**Przyspieszenie**			
				par_for_outer	**par_for_workers**	**spmd_codistr**	**spmd_monte_carlo**
1	2	2	100	1,60	1,71	1,62	1,67
2	2	4	100	1,66	1,70	1,54	1,72
3	2	8	100	1,63	1,69	1,48	1,75
4	4	2	50	1,61	1,78	1,69	1,76
5	4	4	100	3,16	2,46	2,15	3,36
6	4	8	100	3,26	3,42	2,48	3,37
7	8	2	25	1,73	1,90	1,81	1,88
8	8	4	50	3,36	2,57	2,27	3,59
9	8	8	100	3,83	3,78	3,01	4,08

Zapewnienie efektywności rozwiązania wymaga, żeby liczba workerów była co najmniej równa liczbie rdzeni. Wtedy wykorzystanie procesora jest rzędu 100% (w innych wypadkach jest istotnie mniejsze, bo niektóre rdzenie procesora nie biorą udziału w przetwarzaniu). Zatem tylko takie najwydajniejsze konfiguracje będą brane pod uwagę w dalszych rozważaniach (wiersze tabeli 1 zaznaczone kolorem szarym).

Wbrew oczekiwaniom najefektywniejszymi nie okazały się konfiguracje, w których liczba workerów jest dokładnie równa liczbie rdzeni. Na ogół można stwierdzić, że większa liczba workerów wpływa korzystniej na wydajność. Przy małej liczbie workerów nie następuje nadmierne przełączanie wątków oraz mniejsze jest zużycie pamięci operacyjnej (przypadającej na każdy proces workera). Mała liczba workerów, dopasowana do liczby rdzeni, może się sprawdzać w symulacjach, w których przydzielane workerom podzadania mają bardzo zbliżoną złożoność. Jeśli jednak tak nie jest (tzn. występuje zróżnicowanie złożoności podzadań jak w omawianym przykładzie), to większa liczba workerów może pośrednio doprowadzić do lepszego zrównoważenia obliczeń. Dla omawianego problemu modelownia ruchu ciał w większości rozwiązań (***par_for_outer***, ***par_for_workers***, ***spmd_monte_carlo*****)** najlepsze lub bliskie najlepszemu wyniki uzyskuje się dla maksymalnej dopuszczalnej liczby workerów, tzn. 8.

Rozwiązania z użyciem pętli zrównoleglającej *parfor* – ***par_for_outer*** i ***par_for_workers*** – mają porównywalną wydajność. ***par_for_outer*** jest lepiej zrównoważone, ale generuje dużo podzadań. ***par_for_workers*** może być ściśle dopasowane do liczby rdzeni, ale nieliczne podzadania mogą różnić się złożonością, co skutkuje gorszym zrównoważeniem obciążenia workerów. Wraz ze wzrostem liczby workerów wydajność obu rozwiązań jest coraz bardziej zbliżona (wzrost liczby workerów generuje większą liczbę podzadań dla ***par_for_workers***, a tym samym upodabnia to rozwiązanie do rozwiązanie ***par_for_outer***). Ponieważ różnice w wydajności ***par_for_outer*** i ***par_for_workers*** są niewielkie (mniejsze niż 10%), więc biorąc pod uwagę prostotę ***par_for_outer***, należy uznać je za rozwiązanie optymalne, przy uwzględnieniu dwóch kryteriów – uzyskanego przyspieszenia i nakładu pracy związanego ze zrównoleglającą modyfikacją kodu źródłowego (liczba zmian kodu w stosunku do rozwiązania ***seq***).

Z rozwiązań opartych na komendzie *spmd* zdecydowanie efektywniejsze jest rozwiązanie ***spmd_monte_carlo***, oparte na przypadkowej dystrybucji poszczególnych danych dla workerów. Takie podejście skutkuje statycznie najlepszym zrównoważeniem obciążenia workerów (asymptotycznie równe zrównoważenie workerów dla bardzo dużych rozdzielczości obrazu charakterystyki ruchu). Na podstawie tabeli 1 (kolumna 7, wiersze 3, 6, 9) można stwierdzić, że rozwiązanie ***spmd_monte_carlo*** jest najwydajniejsze z wszystkich omawianych (niezależnie od konfiguracji).

Skalowalność – rozumiana tutaj jako wartość przyspieszenia w funkcji liczby rdzeni – nawet dla najwydajniejszych rozwiązań daleka jest od liniowej. Wyniki dla 2 i 4 rdzeni można uznać za zadowalające. Wynik dla 8 rdzeni raczej nie – przyspieszenie zbliżone jest do konfiguracji dla 4 rdzeni (niewiele większe od 4).

Taka charakterystyka skalowalności wynika nie tylko z wewnętrznych mechanizmów MATLABa, ale również ze specyfiki samego programu symulacji, który tylko w części „daje się dobrze zrównoleglić". Korzystając z prawa Amdahla [10, 11], można wyznaczyć „część szeregową" i „część równoległą" (w sensie czasu przetwarzania), tak jak w pracy [5], i ekstrapolować wartości przyspieszenia na maszyny o większej liczbie rdzeni. Jednak na razie obowiązujące ograniczenie do 8 workerów podważa (chwilowo) sens takich rozważań.

3. Problem N-ciał – ewolucja złożonych układów ciał w polu grawitacyjnym

Symulacje problemu N-ciał (ang. *N-body problem*) mają istotne znacznie w badaniach nad zachowaniem złożonych układów licznych ciał lub cząstek. Mają zastosowanie w astrofizyce (oddziaływanie grawitacyjne), fizyce cząsteczkowej (siły elektrostatyczne) czy grafice komputerowej (tzw. metoda energetyczna do wyznaczenia globalnego rozkładu oświetlenia scen trójwymiarowych – ang. *radiosity*).

W ramach niniejszej pracy zbadano ruch złożonych układów ciał w polu grawitacyjnym, w przestrzeni trójwymiarowej. Celem jest obserwacja ewolucji układów ciał, takich jak galaktyki czy otwarte lub kuliste gromady gwiazd.

W rozwiązaniu przyjęto najprostszy sposób realizacji, zakładając wyznaczanie oddziaływania każdego ciała z każdym innym. Takie podejście (zwane *all-paired*) ma złożoność rzędu $O(N^2)$, gdzie N oznacza liczbę ciał układu (tzn. aby określić ruch pojedynczego ciała, należy wyznaczyć wypadkową siłę, wynikającą z oddziaływania z pozostałymi $N - 1$ ciałami, czyli, w sumie, należy wykonać wyliczenie siły grawitacji $\frac{N(N-1)}{2}$ razy). Istnieją inne podejścia, wykorzystujące przybliżone wartości sił grawitacji, ale o znacznie lepszej złożoności rzędu $O(N \log N)$, np. rozwiązanie oparte na hierarchicznym podziale trójwymiarowej przestrzeni z użyciem drzew ósemkowych (metoda BH-tree [12, 23]).

3.1. Opis modelu ciągłego układu dynamicznego – koncepcja programu symulacji

Ruch dowolnie wybranego z układu i-tego ciała o masie m_i można opisać za pomocą 6-ciu zmiennych stanu, tj.: x_i, y_i, z_i – składowych położenia $\vec{r}_i$ w przestrzeni trójwymiarowej oraz $v_{x,i}$, $v_{y,i}$, $v_{z,i}$ – składowych wektora prędkości ciała $\vec{v}_i$. Przyspieszenie ciała i-tego można

wyznaczyć na podstawie II zasady dynamiki Newtona, tj. $\vec{a}_i = \frac{\vec{F}_i}{m_i}$, gdzie $\vec{F}_i$ jest wektorem siły wypadkowej, uwzględniającej przyciąganie pozostałych $N - 1$ ciał.

Zakładając znajomość wartości zmiennych stanu ciała i-tego w chwili t_k, można wyznaczyć ich przybliżone wartości w następnej chwili t_{k+1} następująco:

$$t_{k+1} = t_k + dt, \tag{2}$$

$$\begin{bmatrix} x_{i,k+1} \\ y_{i,k+1} \\ z_{i,k+1} \end{bmatrix} = \begin{bmatrix} x_{i,k} \\ y_{i,k} \\ z_{i,k} \end{bmatrix} + \begin{bmatrix} v_{x,i,k} \\ v_{y,i,k} \\ v_{z,i,k} \end{bmatrix} dt, \tag{3}$$

$$\begin{bmatrix} v_{x,i,k+1} \\ v_{y,i,k+1} \\ v_{z,i,k+1} \end{bmatrix} = \begin{bmatrix} v_{x,i,k} \\ v_{y,i,k} \\ v_{z,i,k} \end{bmatrix} + \frac{1}{m_i} \begin{bmatrix} F_{x,i,k} \\ F_{y,i,k} \\ F_{z,i,k} \end{bmatrix} dt, \tag{4}$$

gdzie dt jest przyjętym kwantem czasu (determinującym dokładność symulacji), a $[F_{x,i,k}\ F_{y,i,k}\ F_{z,i,k}]^T = \vec{F}_{i,k}$ jest siłą wypadkową, działającą na ciało i-te w chwili t_k. Wzory 3 i 4 wynikają z przyjętego sposobu przybliżonego wyznaczania przyspieszenia ciała i jego prędkości jako odpowiednich ilorazów różnicowych[4]:

$$\vec{v}_{i,k} \approx \frac{\vec{r}_{i,k+1} - \vec{r}_{i,k}}{dt}, \quad \vec{a}_{i,k} \approx \frac{\vec{v}_{i,k+1} - \vec{v}_{i,k}}{dt}. \tag{5}$$

Siłę oddziaływania pomiędzy wybranym i-tym ciałem a dowolnym innym j-tym można wyrazić następującym wzorem (zgodnie z prawem grawitacji Newtona):

$$\vec{F}_{ij} = \begin{bmatrix} F_{x,i,k} \\ F_{y,i,k} \\ F_{z,i,k} \end{bmatrix} = G \frac{m_i m_j}{r_{ij}^3} \begin{bmatrix} x_j - x_i \\ y_j - y_i \\ z_j - z_i \end{bmatrix}, \tag{6}$$

$$r_{ij}^2 = (x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2, \tag{7}$$

gdzie r_{ij} oznacza odległość pomiędzy ciałami.

Siłę wypadkową dla i-tego ciała można wyznaczyć przez sumę wszystkich sił grawitacji:

$$\vec{F}_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \vec{F}_{ij}. \tag{8}$$

Zakłada się, że rozmiary wszystkich ciał są zaniedbywalne i wszelkie zderzenia ciał mają charakter zderzeń sprężystych.

[4] Takie podejście jest równoważne zastosowaniu najprostszej z jednokrokowych metod całkowania równań stanu ze stałym krokiem całkowania (metoda Eulera) [1].

3.2. Opis sekwencyjnego programu symulacji

Program symulacji (oznaczony symbolem ***seq***) wyznacza i prezentuje tory ruchu poszczególnych ciał, wchodzących w skład układu ciał. Celem symulacji jest pokazanie ewolucji układu (układów) ciał w zadanym okresie czasu.

Najbardziej czasochłonnym elementem wykonania programu jest wyznaczenie siły wypadkowej dla każdego ciała. Należy zauważyć, że ten fragment programu ma jednakową złożoność niezależnie od wyboru ciała. Program obsługuje macierz ciał *bodies*, reprezentującą stan układu w danej chwili. Macierz ma rozmiar $N \times 7$. *i*-ty wiersz macierzy reprezentuje *i*-te ciało. Pierwsza kolumna reprezentuje masy ciał. Następne trzy kolumny to składowe położenia. Ostatnie trzy kolumny to składowe prędkości. Zgodnie ze wzorami 2-8 program wyznacza macierze (nowe wartości w kolumnach 2-7) w kolejnych chwilach czasu, różniących się o wartość *dt*.

Program dokonuje również prezentacji położeń ciał: początkowych, pośrednich (w chwilach będących małymi wielokrotnościami *dt*) i końcowych.

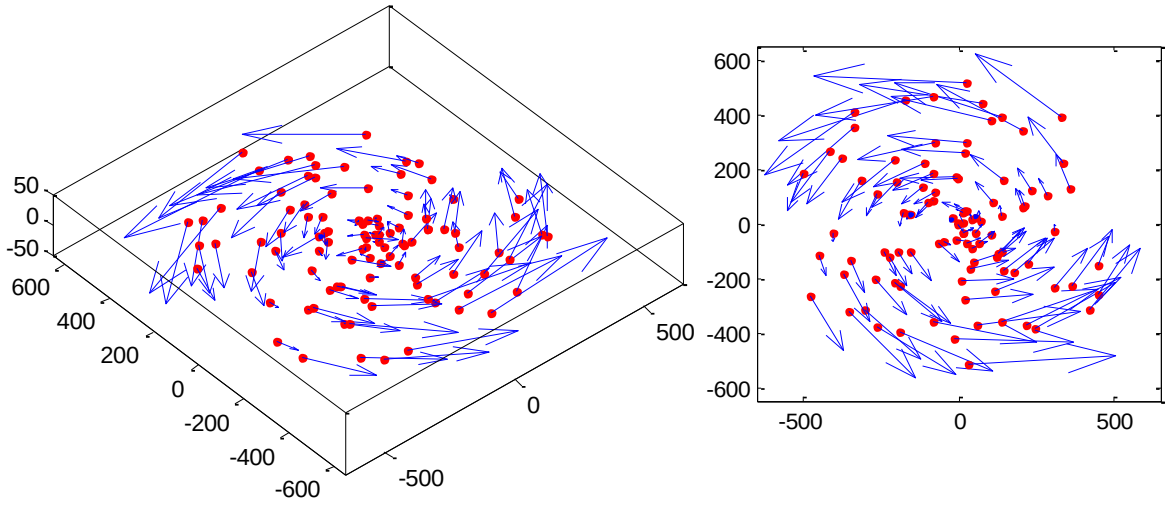
Dla pojedynczego kroku algorytmu (kroku związanego z przesunięciem czasu o kwant *dt*) jego czasochłonność w funkcji N można określić jako $aN^2 + bN + c$, gdzie:

- składnik aN^2 dotyczy części algorytmu wyznaczającej siły dla wszystkich ciał (oddziaływanie każdy z każdym),
- składnik bN dotyczy głównie czasochłonności wyświetlania położeń N ciał,
- a, b, c to pewne dodatnie stałe.

Ta kwadratowa zależność opisująca czasochłonność realizacji zostanie wykorzystana w podrozdziale 3.5.

3.3. Wynik symulacji

W ramach pracy przeprowadzono symulację ewolucji małych, eliptycznych galaktyk, składających się z N jednakowych ciał, gdzie $N = 100, 1000, 10000$. Przykładowy, inicjalny rozkład położeń i prędkości ciał dla $N = 100$ został pokazany na rys. 6.

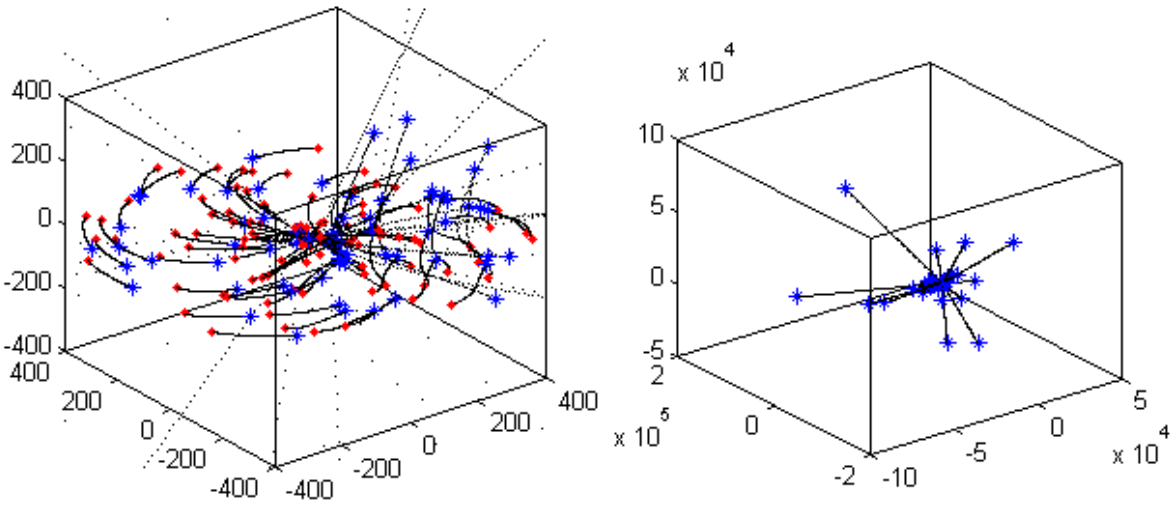


a) b)

Rys. 6. Początkowy rozkład położenia i prędkości ciał układu: a) widok „trójwymiarowy”, b) rzut na płaszczyznę *xy* startowych położeń i prędkości ciał

Fig. 6. Initial locations and velocities of bodies: a) a 3D view, b) a *xy* view of startup locations and velocities of bodies

W wyniku symulacji, po upływie 10000 jednostek czasu (z przyjętym kwantem czasu 0.001), uzyskano obraz układu zaprezentowany na rys. 7. Z rys. 7b można zauważyć, że niektóre ciała opuściły umowne otoczenie układu (wyleciały daleko poza obszar prostopadłościanu o krawędziach wielkości rzędu 10^2). Jednak większość ciał kontynuuje ruch w ramach układu eliptycznego – rys. 7a (tzn. znajduje się nadal w bliskim sąsiedztwie środka układu galaktyki).



a) b)

Rys. 7. Wyniki symulacji – położenia początkowe ciał (czerwone koła), tor (czarne kropki tworzące linie), położenie końcowe (niebieskie gwiazdki): a) obraz ruchu w pobliżu środka układu, b) ucieczka niektórych ciał z sąsiedztwa galaktyki

Fig. 7. Simulation results – initial locations of bodies (red circles), trajectories (black dots creating lines), final locations (blue asterisks): a) bodies movements near the center of the system, b) escaping of some bodies from the galaxy

3.4. Opis zrównoleglonej wersji programu symulacji z użyciem komendy *parfor*

Zrównoleglona wersja programu symulacji ruchu układu N-ciał (oznaczona przez ***par_for***) oparta jest na komendzie *parfor*. Zrównolegleniu podlega część kodu wyznaczająca nowe wartości składowych położenia i prędkości (wzory 2-8). Wyznaczenie nowych wartości zmiennych stanu (w kolejnej chwili czasu) dla pojedynczego ciała z *i*-tego wiersza macierzy *bodies* (polegające na uwzględnieniu wszystkich innych wierszy macierzy *bodies* w celu wyznaczenia składowych wypadkowej siły działającej na ciało *i*-te), stanowi pojedynczą iterację zrównoleglonej pętli. Operacja wyznaczania nowych wartości zmiennych stanu łatwo poddaje się zrównolegleniu, ponieważ siłę wypadkową dla każdego z ciał można wyznaczać niezależenie (przez różne workery).

Zrównoleglona została część algorytmu symulacji, której złożoność jest rzędu $O(N^2)$. Po każdym (zrównoleglonym) wyznaczaniu nowego stanu całego układu, w kolejnej chwili czasu (nowe wartości położeń i prędkości wszystkich ciał), może nastąpić wyświetlenie (niezrównoleglone) aktualnych położeń. Złożoność tej niezrównoleglonej części programu jest rzędu $O(N)$.

3.5. Rezultaty eksperymentów i interpretacja wyników – ocena efektywności rozwiązania zrównoleglonego

Wyniki przyspieszenia zrównoleglonego programu symulacji ruchu układu N-ciał w stosunku do sekwencyjnego w zależności od liczby workerów (liczby workerów i rdzeni są równe) przedstawiono w tabeli 3. Biorąc pod uwagę, że podzadania mają jednakową złożoność (wyznaczenie siły wypadkowej dla każdego ciała), zastosowano konfigurację, w której liczba rdzeni była równa liczbie workerów – zgodnie z oczekiwaniami uruchamianie dodatkowych workerów przy podzadaniach o bardzo zbliżonej złożoności nie poprawiało istotnie efektywności rozwiązania.

Tabela 3

Przyspieszenie zrównoleglonej wersji programu symulacji problemu N-ciał w stosunku do sekwencyjnej

Liczba rodzeni i liczba workerów	**Liczba ciał**		
	100	1000	10000
	Przyspieszenie *par_for* względem *seq*		
2--2	1,4	1,72	1,81
4--4	1,7	3,25	3,28
8--8	1,91	4,82	4,91

Wraz ze wzrostem *N* najbardziej czasochłonna staje się realizacja fragmentu programu o złożoności rzędu $O(N^2)$, związanego z wyznaczaniem sił wypadkowych dla wszystkich ciał (fragment podlegający zrównolegleniu). Inne fragmenty programu, o niższym rzędzie złożo-

ności, przestają być istotne, stąd następuje nasycenie charakterystyki współczynnika przyspieszenia dla dużych wartości *N*.

Takie asymptotyczne „nasycenie” można łatwo uzasadnić. Przyjmując

- czasochłonność części mogącej podlegać zrównolegleniu jako:

$$a_p N^2 + b_p N + c_p, \tag{9}$$

- czasochłonność części zawsze sekwencyjnej jako:

$$a_s N^2 + b_s N + c_s, \tag{10}$$

gdzie a_p, b_p, c_p, a_s, b_s, c_s są pewnymi dodatnimi stałymi, można określić współczynnik przyspieszenia wersji zrównoleglonej względem sekwencyjnej, dla *n* workerów, jako:

$$k(N,n) = \frac{\text{czasochłonność wersji sekwencyjnej}}{\text{czasochłonność wersji zrównoleglonej z użyciem } n \text{ workerów}} \tag{11}$$

czyli:

$$k(N,n) = \frac{a_p N^2 + b_p N + c_p + a_s N^2 + b_s N + c_s}{\frac{a_p N^2 + b_p N + c_p}{n} + a_s N^2 + b_s N + c_s}. \tag{12}$$

Wartości stałych oraz interpretacja znaczenia stałych nie będą ważne z punktu widzenia dalszych rozważań, ale oczywiście, w świetle wcześniejszego opisu algorytmu (w poprzednim akapicie), istotne są a_p (dotyczy wyznaczania sił wypadkowych) i b_s (dotyczy wyświetlania położeń ciał).

Korzystając ze wzoru 12, dla bardzo dużych *N* można znaleźć asymptotyczne ograniczenie $k(n)$ jako:

$$k(n) = \lim_{N\to\infty} k(N,n) = \frac{a_p + a_s}{\frac{a_p}{n} + a_s}, \tag{13}$$

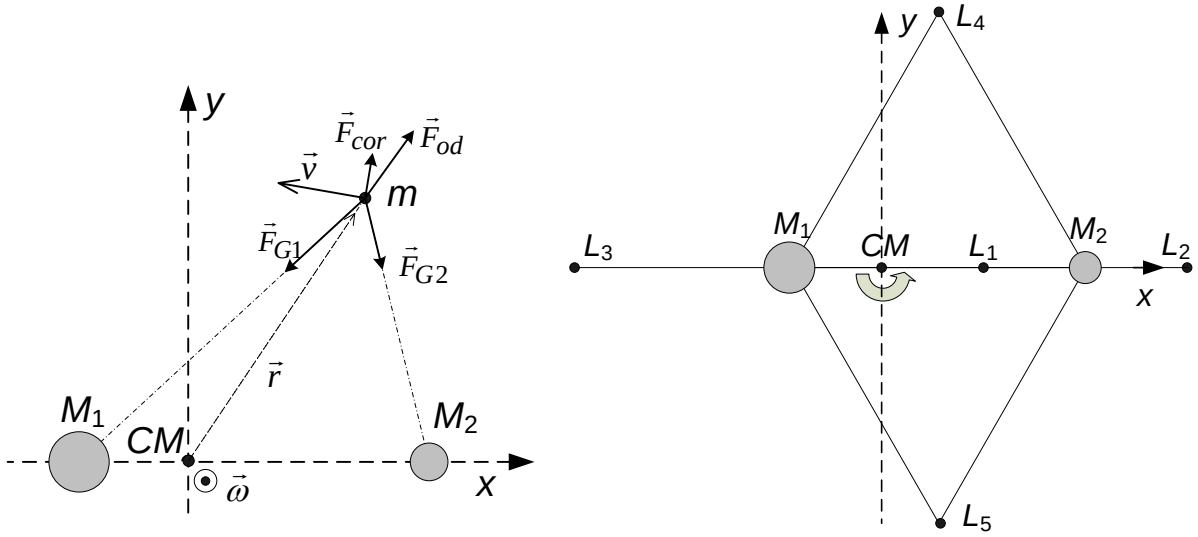
którego wartość jest stała dla danego *n*.

Ponieważ dalszy wzrost wartości współczynnika przyspieszenia *k* ze wzrostem *N* jest niewielki (tzn. dla 1000 i 10000 różnica przyspieszeń wynosi poniżej 0,1), nie podano wyników symulacji dla *N* powyżej 10000 – przyjęto, że wartości uzyskane w ostatniej kolumnie tabeli 3 są wystarczająco bliskie wartości „nasycenia” (patrz wzór 13).

Wyniki eksperymentów pokazują (ostatnie kolumna w tabeli 3), że podobnie jak w przypadku zagadnienia opisywanego w rozdziale 2, wykorzystanie programu zrównoleglonego na komputerze z czterema rdzeniami jest korzystne (przyspieszenie powyżej 3), ale dla ośmiu rdzeni jest mało zadowalające (chociaż nieco korzystniejsze niż w przykładzie opisywanym w rozdziale 2).

4. Ograniczone kołowe zagadnienie ruchu trzech ciał – uzyskanie torów ruchu ciała lekkiego

Klasycznym zagadnieniem znanym z astronomii, dotyczącym dynamiki ruchu ciał, jest tzw. ograniczony kołowy problem trzech ciał (ang. CRTB problem – *circular restricted three-body problem*) [13, 14]. W ramach tego problemu rozpatrywany jest płaski ruch ciała lekkiego (nieważkiego) w nieruchomym, nieinercjalnym układzie dwóch różnych ciał ciężkich, pokazany na rys. 8a. Względem dowolnego układu inercjalnego nieinercjalny układ odniesienia obraca się przeciwnie do kierunku wskazówek zegara, a oś obrotu jest prostopadła do płaszczyzny rysunku i przechodzi przez środek masy układu ciał o masach M_1 i M_2 (stały wektor prędkości kątowej $\overrightarrow{\omega}$ jest skierowany do obserwatora). Zakłada się, że $M_1 > M_2$.



a) b)

Rys. 8. Położenia dwóch ciał ciężkich w nieruchomym, nieinercjalnym układzie odniesienia: a) ilustracja wektorów sił i wektora prędkości liniowej ciała lekkiego, b) położenia punktów równowagi L_1~L_5 w układzie nieinercjalnym

Fig. 8. Locations of two heavy bodies in a motionless non-inertial referential frame: a) illustration of forces vectors and velocity vector of a weightless body, b) locations of libration points L_1~L_5 in the non-inertial referential frame

W układzie nieinercjalnym na ciało lekkie o masie *m* działają następujące siły (rys. 8a):

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

$$\vec{F}_{Gi} = -G\frac{M_i m}{r_i^3}\vec{r}_i = -G\frac{M_i m}{((x-x_i)^2+(y-y_i)^2)^{\frac{3}{2}}}[x-x_i, y-y_i], \tag{14}$$

gdzie $i = 1, 2$, M_i – masa i-tego ciała ciężkiego, $[x_i, y_i]$ – 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:

$$\vec{F}_{od} = m\omega^2\vec{r} = m\omega^2[x, y], \tag{15}$$

gdzie ω – wartość prędkości kątowej obrotu nieinercjalnego układu odniesienia,

- siła Coriolisa:

$$\vec{F}_{cor} = -2m(\vec{\omega}\times\vec{v}) = 2m(\vec{v}\times\vec{\omega}), \tag{16}$$

gdzie $v = [v_x, v_y]$ – prędkość ciała lekkiego w nieinercjalnym układzie odniesienia, $\vec{\omega}$ – wektor prędkości kątowej układu odniesienia, prostopadły do płaszczyzny ruchu (w szczególności zawsze prostopadły do wektora $\vec{v}$); uwzględniając (16) i ortogonalne położenie wektora $\vec{\omega}$ względem $\vec{v}$, siłę Coriolisa można wyrazić następująco:

$$\vec{F}_{cor} = -2m\omega[v_x,\ v_y]. \tag{17}$$

Suma czterech wymienionych sił pozwala na wyznaczanie siły wypadkowej, działającej na ciało lekkie. Wyrażenia na składowe x i y siły wypadkowej pozwalają na zbudowanie równań stanu układu dynamicznego, określających pochodne czterech zmiennych stanu, opisujących płaski ruch ciała lekkiego.

Zakładając, że odległość pomiędzy ciałami ciężkimi wynosi R i środek masy układu

$$x_{sr} = \frac{M_1x_1 + M_2x_2}{M_1 + M_2} \tag{18}$$

znajduje się w środku układu współrzędnych, współrzędne x_1 i x_2 położeń ciał ciężkich można znaleźć z następującego układu równań:

$$\frac{M_1x_1 + M_2x_2}{M_1 + M_2} = 0 \tag{19}$$

$$-x_1 + x_2 = R. \tag{20}$$

Przyjmując:

$$\mu = \frac{M_2}{M_1 + M_2}, \tag{21}$$

na podstawie wzorów 19 i 20 można otrzymać następujące wartości współrzędnych:

$$x_1 = -\mu R, \tag{22}$$

$$x_2 = (1-\mu)R. \tag{23}$$

W dalszych rozważaniach przyjmuje się, że $R = 1$, $G = 1$, $\omega = 1$, $M_1 + M_2 = 1$, co nie zmniejsza ogólności rozważań. Zatem, uwzględniając wzory 21~23, położenia ciał ciężkich wynoszą odpowiednio $[x_1, y_1] = [-\mu, 0]$ i $[x_2, y_2] = [1-\mu, 0]$, a masy $M_1 = 1-\mu$ i $M_2 = \mu$.

Analiza matematyczna układu prowadzi do znalezienia punktów równowagi, tzn. punktów, w których wypadkowa trzech sił – dwóch grawitacyjnych i odśrodkowej (czyli tylko tych, uzależnionych od położenia ciała lekkiego w nieinercjalnym układzie odniesienia) wynosi zero. Punkty te, oznaczone na rys. 8b jako L_1~L_5, nazywane są punktami libracji lub

punktami Lagrange [14]. Współrzędne x dla trzech współliniowych punktów równowagi L_1, L_2, L_3 można obliczyć rozwiązując w sposób przybliżony równania wielomianowe piątego stopnia. Przykładowo, równanie (po wyeliminowaniu masy ciała lekkiego m) pozwalające na obliczenie współrzędnej x dla punktu libracji L_1, ma następującą postać:

$$x+\frac{\mu}{(x-1+\mu)^2}-\frac{1-\mu}{(x+\mu)^2}=0, \tag{24}$$

gdzie trzy składniki lewej strony równania to wartości: siły odśrodkowej, siły przyciągania ciała lżejszego o masie μ i przeciwnie skierowanej do obu siły przyciągania ciała cięższego o masie $1-\mu$. Podobne równania łatwo można skonstruować dla punktów L_2 i L_3.

Rozwiązanie równania 23, tzn. szukana, przybliżona wartość składowej x położenia punktu L_1 wynosi:

$$x_{L_1}\approx 1-\left(\frac{\mu}{3}\right)^{\frac{1}{3}}+\frac{1}{3}\left(\frac{\mu}{3}\right)^{\frac{2}{3}}+... \tag{25}$$

Kolejne składniki w wyrażeniu na x_{L_1} są pomijalnie małe (zależą od małego μ co najmniej w pierwszej potędze). Przybliżone położenia punktów L_1, L_2, L_3 [22] mogą być wyrażone następująco:

$$L_1:\left[1-\left(\frac{\mu}{3}\right)^{\frac{1}{3}},0\right],\ L_2:\left[1+\left(\frac{\mu}{3}\right)^{\frac{1}{3}},0\right],\ L_3:\left[-\left(1+\frac{5}{12}\mu\right),0\right]. \tag{26}$$

Punkty L_4 i L_5, nazywane trójkątnymi punktami libracji, położone są w wierzchołkach dwóch trójkątów równobocznych, których dwa wierzchołki umieszczone są w pozycjach ciał ciężkich (rys. 8b). Bardzo prosty dowód, że wypadkowa sił obu grawitacyjnych i siły odśrodkowej wynosi zero w punktach L_4 i L_5 nie zostanie przedstawiona. Dokładne współrzędne położeń L_4 i L_5 są następujące:

$$L_4:\left[\frac{1}{2}-\mu,\frac{\sqrt{3}}{2}\right],\ L_5:\left[\frac{1}{2}-\mu,-\frac{\sqrt{3}}{2}\right]. \tag{27}$$

Punkty równowagi układu dynamicznego można znaleźć inną metodą – przyrównując do zera prawe strony równań stanu (tzn. przyjmując, że pochodne zmiennych stanu są równe 0) [1]. Odpowiednie równania można otrzymać korzystając z kodu programu zamieszczonego w podrozdziale 4.1 (ciało M-funkcji *CTRB_rstate*, implementującej równania stanu układu dynamicznego) i przyjmując za zero wartości zmiennych: *dx3*, *dy3*, *dvx3*, *dvy3*.

Teoretyczna analiza własności wymienionych punktów równowagi prowadzi do wniosków, że punkty współliniowe L_1~L_3 są niestabilne, natomiast punkty trójkątne L_4 i L_5 są stabilne. Nieformalnie można temu nadać następującą interpretację – ciało umieszczone w punkcie L_4 lub L_5, „lekko wytrącone" z położenia w tym punkcie nie opuści otoczenia tego punktu.

Na potrzeby uzasadnienia wspomnianych wniosków, dotyczących stabilności punktów libracji, wprowadzone zostanie pojęcie potencjału efektywnego (zwanego również pseudopotencjałem), oznaczonego jako Ω.

Składowa potencjału wynikająca z działania pola grawitacyjnego, którego źródłem jest M_1, wynosi:

$$V_{G1}(x,y) = -G\frac{M_1}{r_1} = -\frac{1-\mu}{r_1}, \tag{28}$$

gdzie $r_1{}^2 = (x - x_1)^2 + (y - y_1)^2 = (x + \mu)^2 + y^2$.

Składowa potencjału, wynikająca z działania pola grawitacyjnego, którego źródłem jest M_2, wynosi:

$$V_{G2}(x,y) = -G\frac{M_2}{r_2} = -\frac{\mu}{r_2}, \tag{29}$$

gdzie $r_2{}^2 = (x - x_2)^2 + (y - y_2)^2 = (x - 1 + \mu)^2 + y^2$.

Przyjmując oznaczenie V_{od} jako potencjał charakteryzujący działanie siły odśrodkowej i korzystając z zależności, że siła w polu potencjalnym jest liczona jako masa razy minus gradient potencjału[5], można uzyskać:

$$m\omega^2 r = -m\frac{dV_{od}}{dr}. \tag{30}$$

Zatem, potencjał V_{od} można znaleźć jako:

$$V_{od} = -\int_0^r \omega^2 r dr = -\omega^2\frac{r^2}{2} = -\frac{1}{2}r^2, \tag{31}$$

gdzie $r^2 = x^2 + y^2$.

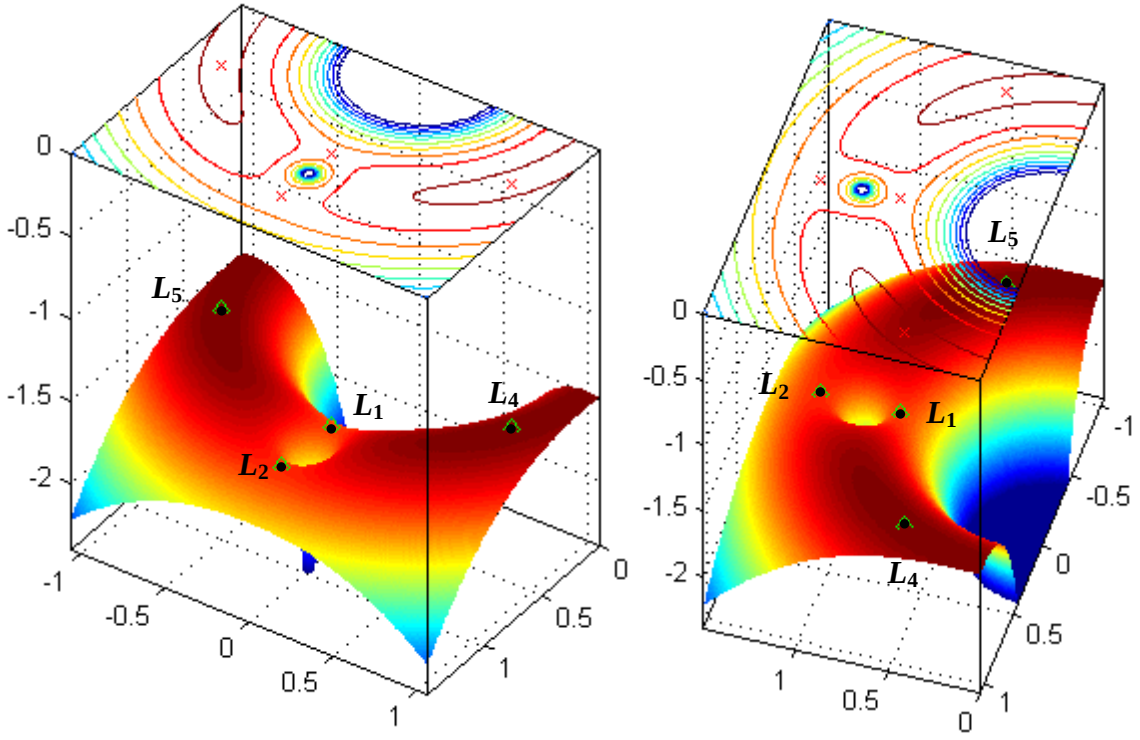
Potencjał efektywny $\Omega = V_{G1} + V_{G1} + V_{od}$, uwzględniając wzory 28, 29 i 31, można zapisać jako:

$$\Omega(x,y) = -\frac{1-\mu}{\sqrt{(x+\mu)^2 + y^2}} - \frac{\mu}{\sqrt{(x-1+\mu)^2 + y^2}} - \frac{1}{2}(x^2 + y^2). \tag{32}$$

Na potrzeby niniejszej pracy stworzono pomocnicze programy w MATLABie, ilustrujące funkcję potencjału efektywnego. Uzyskaną postać $\Omega(x, y)$, zwaną powierzchnią Jacobiego [13], pokazano na rys. 9. Pokazany fragment powierzchni dotyczy okolic punktów L_1, L_2, L_4 i L_5.

Rysunki 9 i 10 pokazują „poziomo przyciętą" funkcję pseudopotencjału, ponieważ w miejscach obu lokalizacji ciał ciężkich potencjał efektywny Ω, zadany wzorem 32, przyjmuje wartość $-\infty$.

[5] We wzorze (30) pochodna V_{od} po r odpowiada gradientowi potencjału V_{od}.

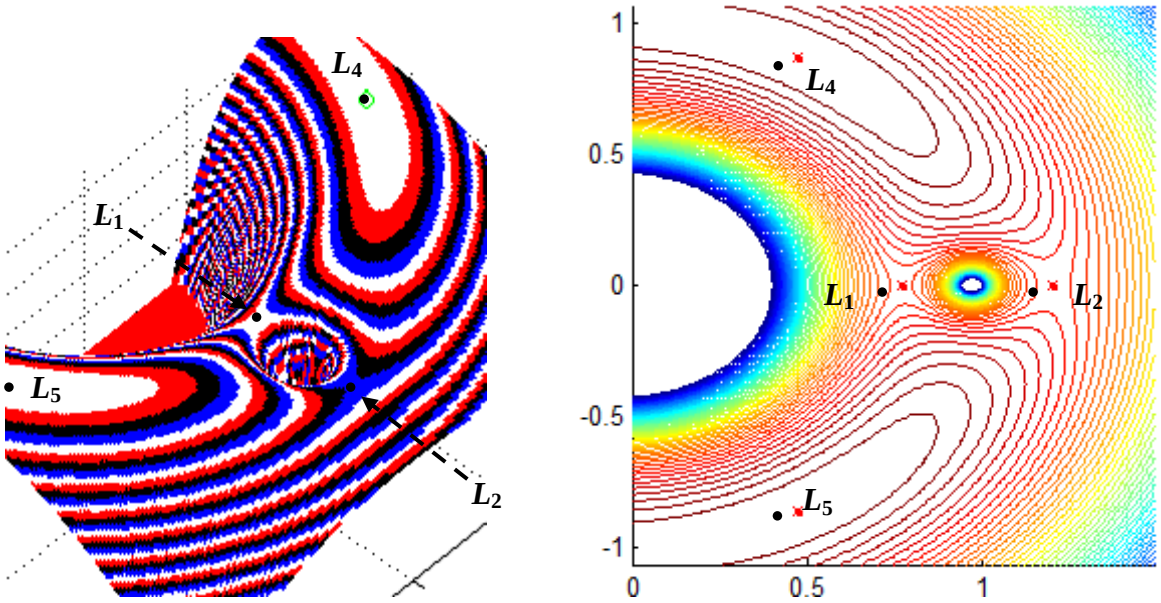


a) b)

Rys. 9. Wykresy funkcji potencjału efektywnego Ω (powierzchnia Jacobiego) w sąsiedztwie punktów równowagi L_1, L_2, L_4, L_5

Fig. 9. Graphs of an effective potential function Ω (a Jacobi surface) near libration points L_1, L_2, L_4, L_5

Punkty równowagi znajdują się w miejscach zerowania pochodnych cząstkowych: $\frac{\partial \Omega}{\partial x}$, $\frac{\partial \Omega}{\partial y}$. Rysunek 10 przedstawia linie ekwipotencjalne Ω w sąsiedztwie punktów równowagi. Upraszczając i skracając dalsze rozważania, na podstawie twierdzeń teorii stabilności układów dynamicznych, można stwierdzić, że punkty L_4 i L_5 są stabilne, gdyż występują w miejscach lokalnego maksimum pseudopotencjału. Współliniowe punkty równowagi (na rysunkach pokazano tylko L_1 i L_2) są niestabilne, gdyż są zlokalizowane w miejscach występowania punktów siodłowych powierzchni pseudopotencjału.



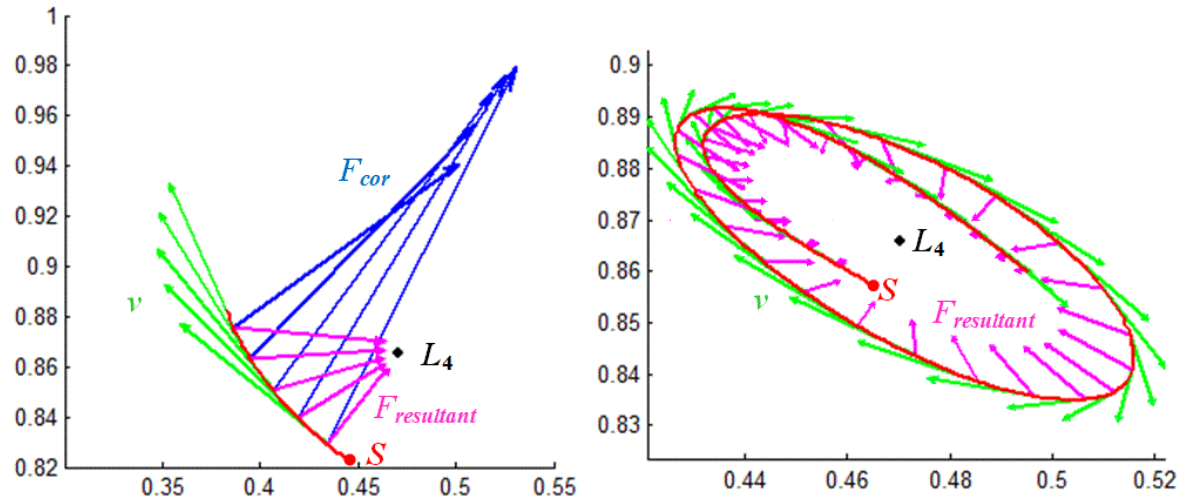
Rys. 10. Ilustracje linii ekwipotencjalnych powierzchni Jacobiego w pobliżu punktów libracji: maksima funkcji potencjału efektywnego w stabilnych trójkątnych punktach równowagi L_4 i L_5; punkty siodłowe w niestabilnych, współliniowych punktach równowagi L_1 i L_2

Fig. 10. Illustrations of equipotential lines of the Jacobi surface near libration points: maximums of the effective potential function at stable triangle libration points L_4 and L_5; saddle points at unstable collinear libration points L_1 and L_2

Dalsze rozważania dotyczyć będą stabilnych, trójkątnych punktów libracji. Siła Coriolisa, działająca na poruszające się ciało lekkie, powoduje, że ciało to może pozostawać w okolicach punktu L_4 (lub L_5). Występowanie siły Coriolisa powoduje, że wypadkową siłę działającą na ciało można potraktować w przybliżeniu jako „siłę dośrodkową", działającą w „mniej więcej" kierunku punktu L_4. W wyniku tego ciało okrąża punkt L_4 (tzw. ruch wokół „niczego" – punkt L_4 pozornie przyciąga ciało lekkie).

W ramach niniejszej pracy zaimplementowano programy w MATLABie, ilustrujące ruch wokół trójkątnego punktu libracji. Rysunki 11a i 11b pokazują wektory: v – prędkości ciała lekkiego (kolor zielony), F_{cor} – siły Coriolisa (kolor fioletowy), $F_{resultant}$ – siły wypadkowej (suma sił: F_{cor}, F_{G1}, F_{G2}, F_{od}; kolor niebieski) oraz linia krzywa - tor ciała (kolor czerwony) dla wybranych chwil czasu i warunków początkowych. Literą S oznaczono punkt startowy (dla chwili $t = 0$).

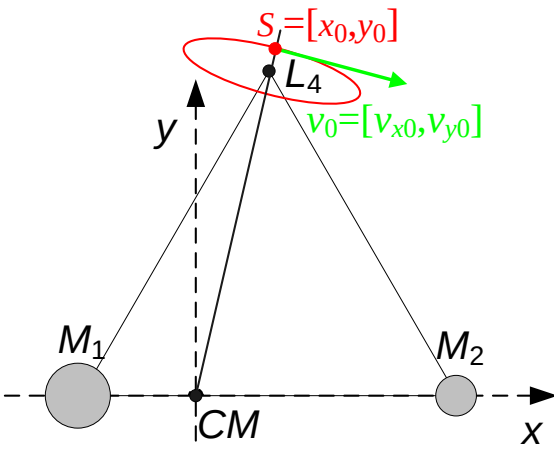
Celem symulacji jest uzyskanie rodziny orbit ciała lekkiego, poruszającego się wokół punktu L_4. Kształty orbit będą uzależnione od przyjętych wartości warunków początkowych, w których założono, że położenia początkowe lekkiego ciała (punkty S) należą do linii łączącej punkt L_4 i środek układu współrzędnych i leżą w pobliżu L_4, a wektory prędkości początkowych (wektory v_0) są prostopadłe do ww. linii, co ilustruje rys. 12a.



a) b)

Rys. 11. Ruch ciała lekkiego wokół punktu równowagi L_4; ilustracja toru, wektorów siły Coriolisa, siły wypadkowej i prędkości w dyskretnych chwilach czasu: a) krótki czas symulacji, b) długi czas symulacji

Fig. 11. Movement of the weightless body around the libration point L_4; presentation of a trajectory and Coriolis force vectors, resultant force ones and velocity ones at some discrete moments of time: a) a short time of simulation, b) a long time of simulation



a)

	S		v_0	
	x_0	y_0	v_{x0}	v_{y0}
1	0,4995	0,8669	0,0013	-0,00075
2	0,5040	0,8747	0,0130	-0,0075
3	0,5140	0,8920	0,0390	-0,0225
4	0,5190	0,9007	0,0515	-0,0297
5	0,5260	0,9128	0,0701	-0,0405
6	0,5265	0,9136	0,0712	-0,0411
7	0,5265	0,9137	0,0712	-0,0411
8	0,5290	0,9180	0,0779	-0,0450

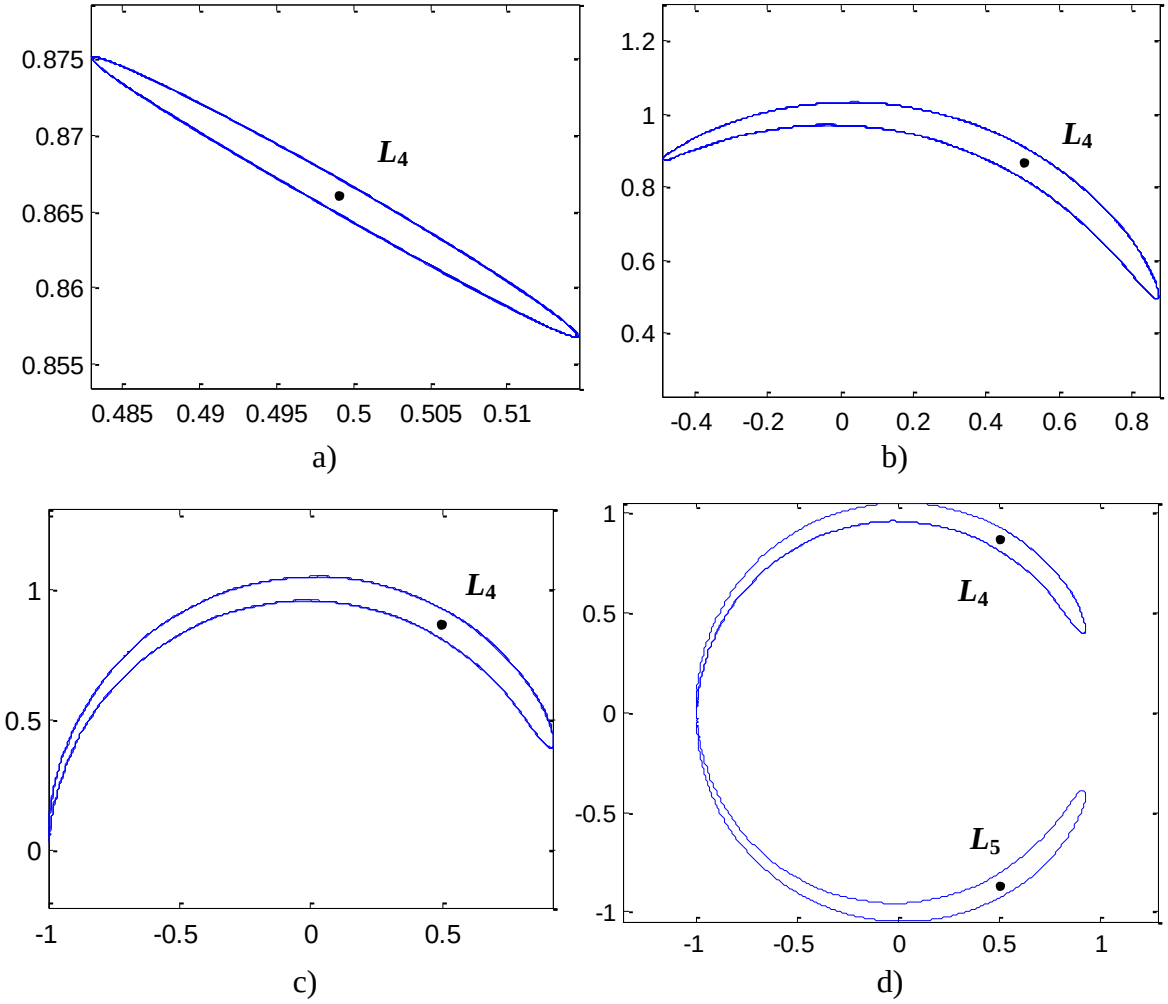
b)

Rys. 12. Warunki początkowe dla ruchu ciała lekkiego – początkowe położenie S i początkowa prędkość v_0: a) ilustracja, b) przyjęty zbiór wartości S i v_0 (dla μ=0,001)

Fig. 12. Initial conditions for the movement of the weightless body – an initial position S and an initial velocity v_0: a) illustration, b) the assumed set of values of S and v_0 (for μ=0,001)

Przykładowe, pojedyncze orbity ciała lekkiego, dla μ = 0,001, uzyskane w MATLABie w wyniku przeprowadzenia symulacji, przedstawiono na rys. 13a~13d. Dla niewielkich odchyleń od punku L_4 i małej prędkości (wiersz nr 1 w tabeli na rys. 12b) orbita kształtem zbliżona jest do elipsy/owalu (rys. 13a). Dla większych odchyleń położenia od L_4 i większej prędkości początkowej (wiersz nr 4) orbita przyjmuje kształt tzw. zakrzywionej kijanki (rys. 13b). Następnie wydłuża się (war. początkowe jak w wierszu nr 6), osiągając oś x (rys. 13c). Po czym (war. początkowe jak wierszu nr 7) ciało lekkie zaczyna okrążać zarówno punkt L_4,

jak i L_5 (rys. 13d), poruszając się po orbicie w kształcie tzw. podkowy, symetrycznej względem osi x.



Rys. 13. Kształty orbit wokół punktu równowagi L_4 dla różnych początkowych położeń i prędkości ciała lekkiego

Fig. 13. Shapes of orbits around the libration point L_4 for different initial positions and velocities of the weightless body

Model CRTB opisuje rzeczywiste, fizyczne układy ciał. Od stuleci obserwowane są przez astronomów zgrupowania planetoid w pobliżu obu trójkątnych punktów libracji w układzie Słońce-Jowisz [14]. Grupa skupiona w okolicy punktu Słońce-Jowisz-L_4 jest określana nazwą „Grecy", a ta skupiona w okolicy Słońce-Jowisz-L_5, określona nazwą „Trojanie"[6]. Podobne zachowanie obserwowane jest w układzie: Saturn i jeden z jego większych księżyców – Thetys. W okolicach punktów trójkątnych L_4 i L_5 występują niewielkie księżyce Saturna: Telesto i Calypso. Również w układzie Ziemia-Księżyc, w okolicach punktów trójkątnych,

[6] Takie nazewnictwo nawiązuje do mitologii greckiej – opisanej w Iliadzie pogoni Achillesa za Hektorem wokół murów Troi.

występują bardzo drobne, ledwo obserwowalne liczne ciała, tzw. księżyce pyłowe Kordylewskiego.

W okolicach niestabilnych, współliniowych punktów równowagi również krążą ciała – sztuczne satelity, umieszczone tam przez człowieka [14]. Oczywiście, utrzymanie ruchu na takich tzw. quasi-periodycznych orbitach (np. orbitach Lissajou), wymaga sporadycznego „wspomagania" w postaci korekty toru z użyciem silników rakietowych (co jest zbędne w utrzymaniu ruchu wokół L_4 lub L_5). Najbardziej znanym przykładem „wykorzystania" współliniowego punktu libracji jest ruch wokół punktu Słońce-Ziemia-L_1 satelity SOHO (ang. *Solar and Heliospheric Observatory*), przeznaczonego do badań Słońca [15].

4.1. Opis modelu ciągłego układu dynamicznego – koncepcja programu symulacji

Poniżej przedstawiono M-funkcję implementującą sposób wyznaczania prawych stron równań stanu, opisujących ruch ciała lekkiego w nieinercjalnym układzie odniesienia w ramach problemu CRTB.

Sekwencja komend *quiver*, włączana opcjonalnie (przez zmianę wartości zmiennej *show_vector*), pozwala na prezentację wektorów: prędkości liniowej, siły Coriolisa i siły wypadkowej, działającej na ciało lekkie, jak pokazano na rys. 11.

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

x3  = xx(1);   % składowe położenia ciała lekkiego
y3  = xx(2);
vx3 = xx(3);   % składowe prędkości ciała lekkiego
vy3 = xx(4);

% ustalenie wartości mi (wzór definicyjny 21)
mi = 0.001;

% masy ciał ciężkich
M1 = 1 - mi; M2 = mi;

% pozycje ciał ciężkich
x1 = -mi; y1 = 0;
x2 = 1 - mi; y2 = 0;

% obliczanie odległości ciała lekkiego od ciał ciężkich
deltax13 = x1-x3; deltay13 = y1-y3;
deltax23 = x2-x3; deltay23 = y2-y3;
r13_sqr = deltax13*deltax13 + deltay13*deltay13; r13 = sqrt(r13_sqr);
r23_sqr=  deltax23*deltax23 + deltay23*deltay23; r23 = sqrt(r23_sqr);

% obliczanie wartości sił grawitacyjnych działających na ciało lekkie
F13 = M1/r13_sqr;
F23 = M2/r23_sqr;

% obliczanie składowych sił grawitacyjnych
F13x = F13 * deltax13/r13; F13y = F13 * deltay13/r13;
F23x = F23 * deltax23/r23; F23y = F23 * deltay23/r23;

% obliczanie składowych siły odśrodkowej (wzór 15)
Fodx = x3; Fody = y3;
```

```
% obliczanie składowych siły Coriolisa (wzór 17)
Fcorx =   2 * vy3 ;  Fcory =  -2 * vx3 ;

% obliczanie składowych przyspieszenia wypadkowego ciała lekkiego
ax3 = F13x+F23x+Fodx+Fcorx; ay3 = F13y+F23y+Fody+Fcory;

% opcjonalne wykreślenie wektorów
global show_vector
if (show_vector ==1)
  % ..... (sprawdzenie czy nadszedł moment czasowy wyświetlenia wektorów)
  hold on
  quiver (x3, y3, Fcorx, Fcory, 'b') ; % siła Coriolisa
  quiver (x3, y3, vx3, vy3, 'g') ; % prędkość
  quiver (x3, y3, ax3, ay3, 'm'); % siła wypadkowa(lub przyspieszenie wypadkowe)
end

% obliczanie wartości pochodnych zmiennych stanu
dx3  = vx3;
dy3  = vy3;
dvx3 = ax3;
dvy3 = ay3;

dxx = [ dx3 ; dy3 ; dvx3 ; dvy3] ;
```

4.2. Opis sekwencyjnego programu symulacji

Poniżej przedstawiono funkcję o nazwie *exper*, realizującą pojedynczy eksperyment dla zadanych warunków początkowych w postaci czteroelementowego wektora *single_ic*.

```
function position = exper (single_ic)
   opt = odeset ('RelTol', 1e-9, 'AbsTol', 1e-10);
   [t,x] = ode45 ('CRTB_rstate', [0 2000] , single_ic, opt);
   position = x(:,1:2);
end
```

Na kolejnym listingu pokazano kod źródłowy M-skryptu implementującego sekwencyjną wersję programu symulacji i pozwalającego na uzyskanie zbioru orbit dla zestawu zadanych warunków początkowych w macierzy IC (wartości macierzy podano w tabeli na rys. 12b).

```
% IC - 8-wierszowa macierz warunków początkowych,
% o wartościach przedstawionych w tabeli na rys. 12b
IC = [. . . ];
[r c] = size (IC);
tic;
x = cell(r,1); % inicjalizacja tablicy wynikowej (wektora typu cell array)
for (i=1:r)
   x {i}(:,:) = exper(IC(i,:));
end
toc
for (i=1:r)
   plot (x{i}(:,1), x{i}(:,2)); hold on
end
```

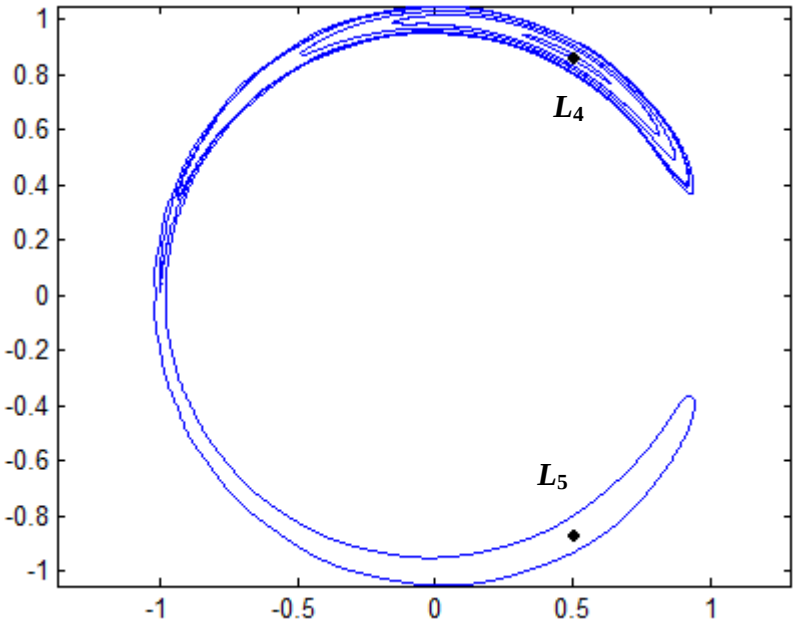
Pomiar czasu (czas wykonania bloku instrukcji objęty komendami *tic* i *toc*) dotyczy jedynie wykonania serii eksperymentów (tzn. pomiar celowo nie obejmuje czasu tworzenia prezentacji wyników). Wyniki wszystkich eksperymentów umieszczane są w zmiennej *x* typu *cell array* [16, 17] (zmienna typu *cell array* może przechowywać obiekty różnych typów,

a w szczególności tablice o różnych rozmiarach). Takie podejście (użycie wektora typu *cell array*) będzie również wykorzystane przy pozyskiwaniu i scalaniu wyników w rozwiązaniu zrównoleglonym, opisanym w podrozdziale 4.4.

4.3. Wynik symulacji

Symulacje przeprowadzono dla następujących wartości parametrów: określenie zależności pomiędzy masami ciał ciężkich – μ=0,001, liczba eksperymentów – 8, wartości początkowe dla pojedynczego eksperymentu w tabeli zamieszczonej na rys. 12b, czas pojedynczego eksperymentu – T_{MAX} = 2000.

Rodzinę orbit uzyskanych w wyniku symulacji pokazano na rys. 14.



Rys. 14. Zbiór orbit wokół punktu L_4 dla przyjętego ośmioelementowego zbioru warunków początkowych

Fig. 14. Set of orbits around the L_4 point for the assumed 8-elements set of initial conditions

4.4. Opis zrównoleglonej wersji programu symulacji

Poniżej przedstawiono kod źródłowy M-skryptu stanowiącego implementację zrównoleglonej wersji programu symulacji. Zrównoleglenie zrealizowano przez zastosowanie „klasycznego" mechanizmu zadań wsadowych (ang. *Regular Jobs*). Każde zadanie niezależnie realizowało pojedynczy eksperyment dla zadanego wektora warunków początkowych.

```
% Ustawienie wartości warunków początkowych (jak w wersji sekwencyjnej)
IC = [ ... ];

% Utworzenie uchwytu identyfikującego lokalny scheduler zadań wsadowych
% (definicja sposobu obsługi zadań wsadowych), na podstawie lokalnej
% konfiguracji(gdzie jest m.in. ustawiona liczba uruchamianych workerów)
sched = findResource('scheduler', 'configuration', 'local2');
```

```
% Utworzenie identyfikatora zadań wsadowych (typu regular job)
job = createJob(sched);

% Dołączenie pliku do definicji środowiska zadań wsadowych
% (zapewnienie „widoczności” pliku)
set(job, 'FileDependencies', {'exper.m'});

[r c] = size (IC);

tic;
for (i=1:r)
% Utworzenie pojedynczego zadania;Skojarzenie z identyfikatorem zadań wsadowych;
% Przekazanie nazwy funkcji - exper i parametrów – czteroelementowy wektor
% warunków początkowych w postaci parametru typu cell array
   t = createTask(job, @exper, 1, {IC(i,:)});
end ;
% Uruchomienie zadań wsadowych; Wystartowanie odpowiedniej liczby procesów wor-
kerów (2, 4 lub 8), w zależności od ustawień w konfiguracji „local2”
submit(job);

% Blokujące oczekiwanie na zakończenie wszystkich zadań,
% skojarzonych z uchwytem o nazwie job
waitForState(job,'finished');

% Pobranie wyników wszystkich zadań
x = getAllOutputArguments(job);
toc
for j=1:r
    plot (x{j}(:,1), x{j}(:,2)); hold on
end
destroy (job); clear (job);
```

W programie określono sposób obsługi zadań wsadowych (funkcja *findResouce*). Uzyskano w ten sposób (w zmiennej *sched*), pośrednio, odniesienie do odpowiedniej konfiguracji), gdzie wcześniej interaktywnie zdefiniowano liczbę Wolkerów, mających brać udział w przetwarzaniu (były to odpowiednio liczby: 2, 4 i 8, tak jak w tabeli 4). Następnie utworzono obiekt stanowiący uchwyt do zbioru zadań wsadowych (funkcja *createJob*). Po czym zdefiniowano poszczególne zadania wsadowe (wywołania funkcji *createTask*, m.in. ustawiające nazwę funkcji i wartości parametrów, jakie mają być przetwarzane przez każde z zadań), które mają realizować pojedynczy eksperyment. Uruchomienie wszystkich zadań obywa się przez wykonanie funkcji *submit*, która to funkcja uprzednio uruchamia dodatkowe procesy workerów. Każde z zadań wykonywane jest tylko przez jeden worker, ale każdy z workerów może być uruchomiony na innym rdzeniu procesora! (stąd przyspieszenie działania w stosunku do wersji sekwencyjnej). Funkcja *waitForState* realizuje blokujące oczekiwanie na zakończenie zadań. Dodatkowo, po zakończeniu realizacji zadań (także w ramach *waitForState*) zamykane są wszystkie procesy workerów.

Na potrzeby zrównoleglenia obliczeń, w podobny sposób do przedstawionego mechanizmu *Regular Jobs*, możliwe jest użycie jeszcze innych, zbliżonych mechanizmów MALTABa: *Parallel Jobs* lub *MATLAB Pool Jobs* [18, 19].

4.5. Rezultaty eksperymentów i interpretacja wyników – ocena efektywności rozwiązania zrównoleglonego

Porównanie przyspieszenia zrównoleglonej wersji programu symulacji dla ograniczonego kołowego problemu trzech ciał, w stosunku do wersji sekwencyjnej, w zależności od liczby rdzeni procesora i liczby uruchomionych workerów pokazano w tabeli 4. W tabeli tej podano również średnie, skumulowane wykorzystanie procesora.

Tabela 4

Porównanie wydajnościowe rozwiązania zrównoleglonego i sekwencyjnego

Liczba workerów	**Liczba rdzeni**					
	2		**4**		**8**	
	przysp.	**%**	**przysp.**	**%**	**przysp.**	**%**
2	1,10	100	1,18	50	1,31	25
4	1,14	100	1,83	100	2,12	50
8	1,18	100	2,04	100	3,25	100

Nawet najlepsze rezultaty (dla 8 uruchamianych workerów) nie są wystarczająco zadowalające – charakterystyka przyspieszenia (stosunek czasu wykonania wersji sekwencyjnej do czasu wykonania wersji zrównoleglonej) w zależności od liczby rdzeni procesora daleka jest od liniowej. Oczywiście, należy zwrócić uwagę na fakt, że czas realizacji wersji zrównoleglonej, opartej na mechanizmie zadań wsadowych (czas rzędu minuty), obejmuje również czas startowania i zamykania procesów workerów (co nie miało miejsca w rozwiązaniach opisywanych w podrozdziałach 2.4 i 3.4). Między innymi stąd słabsza wydajność takiego rozwiązania w stosunku do tych omawianych w 2.4 i 3.4.

5. Podsumowanie

W artykule przedstawiono metody zrównoleglenia obliczeń, dostępne w systemie MATLAB dzięki użyciu modułu Parallel Computing Toolbox [20]. Analizowane przykłady dotyczą modelowania ruchu ciał w polu grawitacyjnym i wynikają z klasycznych problemów z zakresu mechaniki nieba, znanych z astronomii. W artykule zawarto teoretyczne rozważania uzasadniające konstrukcje odpowiednich modeli matematycznych, leżących u podstaw zaproponowanych programów symulacji. Stworzone w MATLABie programy, pozwalające na zrównoleglonąą symulację odpowiednich układów dynamicznych, mogą być wykonywane efektywnie, gdy są uruchamiane na komputerach z procesorami wielordzeniowymi. Jakościowa i ilościowa analiza wydajności takich rozwiązań została przestawiona w niniejszym artykule.

Przykład pierwszy dotyczy utworzenia charakterystyki wrażliwości układu dynamicznego na zmianę warunków początkowych – obrazuje wpływ zmian warunków początkowych na przebieg ruchu ciała lekkiego w trójkątnym układzie ciał ciężkich (uzyskanie portretu obrazującego końcowe położenie ciała lekkiego – ang. *gravitational fractal*). Przykład służy do ilustracji sposobów zrównoleglania i prezentacji efektywnościowych wyników zrównoleglenia programu przy pomocy komend *parfor* i *spmd*. Charakter problemu jest taki, że podział całego zadania na podzadania nie jest równy w sensie czasochłonności każdego z podzadań (co jest konsekwencją zastosowania metod całkowania o zmiennym kroku całkowania). Stąd dokładne zrównoważenie obliczeń na komputerach z procesorami wielordzeniowymi nie jest możliwe. Wyniki eksperymentów prowadzą do wniosków, że rozwiązanie z użyciem pętli *parfor* – ang. *parallel for* (rozwiązanie nazwane *par_for_outer*) jest optymalne w sensie kryterium – łatwość implementacji vs. względne przyspieszenie w stosunku do wersji sekwencyjnej. Najefektywniejsze okazało się rozwiązanie z użyciem komendy *spmd* – ang. *single program multiple data* (rozwiązanie nazwane *spmd_monte_carlo*, z losową dystrybucją obsługi punktów startowych pomiędzy procesy workerów). Wyniki eksperymentów pokazują, że czas realizacji symulacji dla rozwiązania zrównoleglonego jest na ogół kilkakrotnie krótszy od czasu symulacji dla rozwiązania sekwencyjnego. Niestety, przyspieszenie zdecydowanie nie skaluje się liniowo w zależności od liczby rdzeni procesora (charakterystyki dla 2, 4 i 8 rdzeni w tabeli 2). Podobna nieliniowa charakterystyka występuje w przypadku kolejnych, rozpatrywanych przykładów.

Przykład drugi dotyczy problemu N-ciał, tzn. symulacji ruchu złożonych układów bardzo licznych ciał (obserwacja ewolucji galaktyki eliptycznej). Zadanie rozwiązujące problem daje się podzielić na podzadania o jednakowej złożoności (jednakowe podzadanie wyznaczenia siły wypadkowej dla pojedynczego ciała). Stąd zastosowano tylko jedno podejście z użyciem komendy *parfor* (każda iteracja zrównoleglonej pętli dotyczy wyznaczania siły wypadkowej pojedynczego ciała układu). Względne przyspieszenia wersji zrównoleglonej w stosunku do wersji sekwencyjnej były nieznacznie wyższe niż te uzyskane w przykładzie pierwszym. Ogólnie jednak należy stwierdzić, że skalowalność rozwiązania zrównoleglonego względem liczby rdzeni procesora również jest daleka od liniowej.

W przykładzie trzecim rozpatrywany jest tzw. ograniczony kołowy problem ruchu trzech ciał (uzyskanie rodziny orbit ciała lekkiego w nieruchomym układzie dwóch ciał ciężkich). Rozwiązanie zrównoleglone problemu oparto na mechanizmie zadań wsadowych (ang. *jobs*). Takie podejście jest najbardziej zbliżone do klasycznego modelu programowego rozpraszania/zrównoleglenia obliczeń, polegającego na jawnym utworzeniu niezależnych podzadań przez podanie funkcji i wartości parametrów (tutaj z użyciem *createTask*). Pod względem efektywnościowym to podejście dawało najmniejsze przyspieszenie wersji zrównoleglonej do sekwencyjnej (w porównaniu z wynikami uzyskanymi w przykładach poprzednich), ale

należy uwzględnić fakt, że uruchomienie wersji zrównoleglonej obejmowało również uruchomienie i zamknięcie wszystkich procesów workerów.

Efektywność rozwiązań zrównoleglonych w stosunku do sekwencyjnych z użyciem MATLAB + Parallel Computing Toolbox można odnieść do efektywności innych specjalizowanych rozwiązań, np. wykonanych z użyciem .NET + Parallel Extensions to .NET Framework [5, 21]. Względne przyspieszenia rozwiązania zrównoleglonego w MATLABie z przykładu pierwszego (rozwiązanie *par_for_outer* dla 8 workerów i 8 rdzeni) i odpowiedniego rozwiązania dla platformy .NET (8 rdzeni) wynoszą odpowiednio: 3,86 (tabela 2) i 3,68 (tabela 2 w [5]). Jednak porównując bezpośrednio zrównoleglone wersje dla MATLABa oraz .NETa (uwzględniając dane z tabeli 3 w [5]), uzyskano wynik przyspieszenia równy 17,8 na korzyść rozwiązania .NET. Oczywiście, rozwiązania zaproponowane w [5] nie są uniwersalne w odróżnieniu od systemu MATLAB, ale mogą stanowić alternatywę w pewnych zastosowaniach (m.in. ze względu na brak ograniczeń na liczbę rdzeni czy uproszczone środowisko uruchomieniowe, sprowadzające się jedynie do użycia .NET Framework w określonej wersji).

Podsumowując, system MATLAB wraz z modułem Parallel Computing Toolbox stanowi bardzo wygodną platformę tworzenia i uruchamiania efektywnych programów, działających wydajnie na maszynach z procesorami wielordzeniowymi. Istnieje wiele wariantów środków programowych, pozwalających na implementację zrównoleglenia, praktycznie zastosowanych i omówionych w niniejszej pracy. Przykłady z dziedziny astronomii pokazują, że zastosowanie modułu Parallel Computing Toolbox pozwala na utworzenie rozwiązań efektywniejszych od odpowiednich rozwiązań sekwencyjnych, ale skalowalność takich zrównoleglonych rozwiązań bywa daleka od liniowej. Przy wykorzystaniu modułu Parallel Computing Toolbox należy uwzględnić ograniczenie, dotyczące możliwości wykorzystania maksymalnie 8 rdzeni (ograniczenia takie nie występują przy tworzeniu rozwiązań uruchamianych w infrastrukturze sieci komputerowej z użyciem modułu MATLAB Distributed Computing Server).

BIBLIOGRAFIA

1. Skowronek M.: Modelowanie cyfrowe. Wydawnictwo Politechniki Śląskiej, Gliwice 2008.
2. The MathWorks. (2010). MATLAB and Simulink for Technical Computing. http://www.mathworks.com.
3. Parallel Computing Toolbox – MATLAB (2010). http://www.mathworks.com/products-/parallel-computing.
4. Solve initial value problems for ordinary differential equations – MATLAB (2010). http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ode113.html?BB=1.

5. Augustyn D. R., Kunc S.: Efektywność programów symulacji ciągłych układów dynamicznych, wykorzystujących moduł Parallel Extensions to .NET Framework, uruchamianych na komputerach z procesorami wielordzeniowymi. Studia Informatica Vol. 31, No. 3 (91), Gliwice 2010.
6. Advanced Topics :: Parallel for-Loops (parfor) (Parallel Computing Toolbox™). (2010) http://www.mathworks.com/access/helpdesk/help/toolbox/distcomp/brdqtjj-1.html#bq-_of7_-1.
7. Single Program Multiple Data (spmd) (Parallel Computing Toolbox™). (2010) http://www.mathworks.com/access/helpdesk/help/toolbox/distcomp/brukbno-1.html.
8. Interactive Parallel Command Window – MATLAB. (2010) http://www.mathworks.com-/access/helpdesk/help/toolbox/distcomp/pmode.html.
9. Working with Codistributed Arrays :: Math with Codistributed Arrays (Parallel Computing Toolbox™). (2010) http://www.mathworks.com/access/helpdesk/help/toolbox-/distcomp/bqi9fln-1.html.
10. 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.
11. Amdahl's law - Wikipedia. (2010). http://en.wikipedia.org/wiki/Amdahl's_law.
12. Barnes J., Hut P.: A hierarchical O(NlogN) force calculation algorithm, Nature, vol. 324 1986.
13. Morzymas J. : Poglądowa geometria równowagi trzech ciał. Nauczanie fizyki w wyższych szkołach technicznych. XIII Konferencja, Wrocław 2000. http://www.if.pwr.wroc.pl-/~kon2000/PRACE/Mozrzymas.doc.
14. Lagrangian point. (2010) http://en.wikipedia.org/wiki/Lagrangian_point.
15. Solar and Heliospheric Observatory Homepage. (2010) http://sohowww.nascom.nasa.gov.
16. Programming and Data Types :: Function Reference (MATLAB®). Cell Arrays. (2010) http://www.mathworks.com/access/helpdesk/help/techdoc/ref/f16-42340.html#f16-6840.
17. Cell Arrays and Their Contents | Loren on the Art of MATLAB. (2006) http://blogs.mathworks.com/loren/2006/06/21/cell-arrays-and-their-contents.
18. Parallel Computing Toolbox. Job and Task Programming. (2010) http://www-.mathworks.com/access/helpdesk/help/toolbox/distcomp/f1-6010.html#f1-7659.
19. Charles A.: Notes: Parallel MATLAB. (2010) http://users.ece.gatech.edu/~acharles6-/MATLAB_ParDoc.pdf.
20. Parallel Computing Toolbox – MATLAB (2010). http://www.mathworks.com/products-/parallel-computing.

21. 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. 31 No. 3 (91), Gliwice 2010.
22. Wierzbiński S.: Mechanika nieba. PWN, Warszawa 1973.
23. Augustyn D.R., Warchał Ł.: Cloud Service Solving N-Body Problem Based on Windows Azure Platform. Communications in Computer and Information Science, Volume 79, Computer Networks, Springer-Verlag, 2010.

Recenzent: Dr inż. Maciej J. Bargielski

Wpłynęło do Redakcji 13 września 2010 r.

Abstract

The paper presents MATLAB parallelization mechanisms used for developing effective simulation programs applied for modeling celestial mechanics. Presented parallelization methods implemented in Parallel Computing Toolbox let develop programs which can be effectively executed on machines with multicore processors. Such programs are known as multicore-aware software.

Classical problems known in dynamics of celestial bodies are considered. Sequential and parallelized programs was developed in MATLAB for such problems as:

- presenting of a hyper-sensivity a weightless body movement to initial conditions in the triangular system of 3 heavy bodies (obtaining gravitational fractal images),
- generating orbits for complex N-body systems (observing evolution of systems of numerous bodies e.g. galaxies, closed clusters of stars),
- presenting trajectories of a weightless body near stable libration points (Lagrangian points) for the circular restricted three-body problem.

The main goal of the paper is qualitative and quantitative analysis of applied parallelization mechanisms from Parallel Computing Toolbox. There are considered:

- parallelized FOR-loop (*parfor*),
- *spmd* command (single program multiple data),
- batch jobs and tasks.

Characteristics of speedup of parallel programs relative to sequential ones is shown. The paper presents how the speedup parameter depends on number of additional MATLAB worker processes and number of processor cores. Effectiveness and scalability of parallel simulation programs are considered for each problem.

Adres

Dariusz Rafał AUGUSTYN: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, draugustyn@polsl.pl.