Anna KOZŁOWSKA
UNIWERSYTET MIKOŁAJA KOPERNIKA, WYDZIAŁ FIZYKI, ASTRONOMII I INFORMATYKI STOSOWANEJ,
ul. Grudziądzka 5, 87-100 Toruń

Estymacja czasów wykonywania algorytmu sterującego w zależności od platformy sprzętowej na użytek diagnostyki obiektu mechanicznego

Mgr inż. Anna KOZŁOWSKA

Ukończyła studia magisterskie na Wydziale Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika w Toruniu na kierunku Fizyka Techniczna w 2011 roku oraz studia inżynierskie na kierunku Automatyka i Robotyka w 2012 roku na tej samej uczelni. Zainteresowania naukowe dotyczą zastosowań konwersji a-c z sygnałem ditherowym, programowania obiektowego oraz z użyciem kart graficznych.

e-mail: annakoz@fizyka.umk.pl

Streszczenie

Opracowanie systemów sterowania obiektami mechanicznymi polega na znalezieniu kompromisu między szybkością działania, a wymaganą dokładnością i jest zagadnieniem o dużej złożoności obliczeniowej. W artykule przedstawiono różne implementacje algorytmu Optymalizacji Rojem Cząstek PSO (ang. Particle Swarm Optimization), który stworzono w celu uzyskania minimalnego czasu obróbki przy zachowaniu zadanej dokładności odtwarzania trajektorii ruchu. Jego działanie zostało porównane w językach: C, C++ i C# oraz na procesorze i karcie graficznej. Z przeprowadzonych badań wynika, że dla małej liczby punktów obliczenia na karcie graficznej są wolniejsze niż na procesorze.

Słowa kluczowe: algorytm optymalizacji rojem cząstek, funkcje testowe, karta graficzna, procesor, czas obróbki, maszyny wieloosiowe.

Estimation of control algorithm execution times in dependence on the hardware platform for use in mechanical object diagnostics

Abstract

Finding the compromise between speed and accuracy is the most important problem in designing control systems. This is a problem of high computational complexity. The paper presents implementation of the algorithm PSO (Particle Swarm Optimization) whose action has been compared in several programming environments (C / OpenCL and C # / Cloo and in C + +) and hardware platforms (CPU and graphics card processor - GPU). PSO is able to achieve the minimum processing time and best possible mapping of a given trajectory. To compare the speed of the PSO algorithm there was made a measurement of the time of test function minimization. The paper describes three test functions commonly used to test the optimization effectiveness. The results show that for a small number of points the calculations on a graphic card are slower than those performed on the CPU. The appropriate use of available parallel computing technologies can significantly improve the characteristics of a multi-axis machine and the expenses incurred for optimization of the PSO can quickly result in important profits. It should be noted that optimization of the processing speed is most needed where the treatment is most complicated. The profit will be negligible for simple trajectories. In special cases, the optimization may extend the processing time without apparent improvement of the characteristics of trajectory mapping.

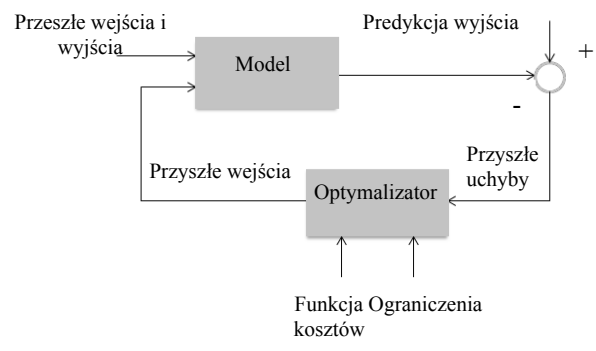
Keywords: particle swarm optimization, test functions, graphic card, processor, processing time, multi-axis machines.

1. Wstęp

Maszyny wieloosiowe sterowane numerycznie stosowane są w szerokim zakresie w przemyśle. Zasięg tych urządzeń obejmuje specjalistyczne maszyny technologiczne, obrabiarki sterowane numeryczne oraz różnego typu manipulatory. Istotne w sterowaniu maszynami wieloosiowymi było wprowadzenie komputerowego sterowania CNC (ang. Computer Numeric Control), a także cyfrowych serwonapędów, które miały znaczny wpływ na poprawę dokładności i wydajności procesów produkcyjnych [1]. W wielu procesach wytwórczych wymagane jest przemieszczenie narzędzia wobec obrabianego przedmiotu po zaprogramowanej trajektorii ruchu. Trajektorie te są realizowane przez złożenie ruchu układów posuwu poszczególnych osi maszyny z możliwie dużą szybkością i dynamiką przy zachowaniu zadanej tolerancji błędów nadążania i konturu. Możliwe jest zastosowanie układów optymalizacyjnych, które pozwolą na wyznaczenie maksymalnej chwilowej prędkości posuwu w zależności od zadanej trajektorii ruchu i własności dynamicznych układu posuwu zapewniając wymaganą dokładność obróbki. Jednym z rozwiązań jest użycie karty graficznej do optymalizacji prędkości w układach sterowania maszyn. Rozwiązanie to znacznie przyspiesza obliczenia, co jest istotne, jeśli wymagane jest sterowanie w czasie rzeczywistym. Zaimplementowano na procesorze i karcie graficznej algorytm PSO (ang. Particle Swarm Optimization), który stanowi fragment generatora trajektorii ruchu w układzie sterowania maszyn i umożliwia uzyskanie minimalnego czasu obróbki przy zachowaniu zadanej dokładności odtwarzania trajektorii ruchu.

2. Regulacja predykcyjna

Dla układów dyskretnych stosuje się regulację predykcyjną, która polega na minimalizacji różnic między wielkościami przewidywanymi w danej chwili, a wartościami zadanymi wyjściowymi (rys.2) na horyzoncie predykcji. W algorytmach regulacji predykcyjnej ważne jest wyznaczenie ciągu przyszłych wartości sygnału sterującego oraz sterowanie według modelu odniesienia poprzez odpowiednio zdefiniowaną trajektorię dla wielkości wyjściowej. Wcześniejsza reakcja regulatora na mającą się dokonać zmianę wartości zadanej kompensuje wpływ czasu opóźnienia [2]. W celu wyznaczenia wartości sygnałów sterujących w chwili bieżącej i następnych wyznacza się minimum tzw. funkcji testowej. Na rys. 1 przedstawiono schemat sterowania predykcyjnego.



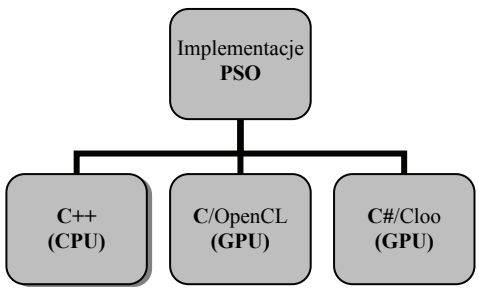
Rys. 1. Schemat algorytmu predykcyjnego [2]
Fig. 1. Diagram of the prediction algorithm [2]

Ważnym elementem całej struktury algorytmu predykcyjnego jest optymalizator, który wylicza wartości funkcji testowej. Wykorzystując te wartości blok optymalizacji ponownie oblicza nową wartość. Wartość ta porównywana jest z poprzednio wyliczoną wartością w celu określenia jakości sterowania. Jeśli jakość sterowania jest niewystarczająca (minimum funkcji celu nie zostało osiągnięte) procedura jest powtarzana. W bloku optymalizacyjnym zastosowano algorytm PSO, który wykazuje podobieństwo metod ewolucyjnych. Wspólnie z algorytmami genetycznymi PSO jest zaliczany do technik inteligencji obliczeniowej lub metod

inspirowanych biologicznie [3, 4]. Algorytm optymalizacji tą metodą polega na wybraniu losowej populacji punktów, a następnie odpowiednim ich ruchu tak, aby ostatecznie osiągnąć punkt minimalny, bądź przynajmniej możliwie najbliższy takiemu punktowi. Potencjalne rozwiązania problemu nazywa się cząstkami, a sam algorytm w języku polskim definiuje się jako optymalizację rojem cząstek.

3. Implementacja algorytmu PSO na procesorze i karcie graficznej komputera PC

Implementacja algorytmu PSO została wykonana dla porównania w kilku środowiskach programistycznych (C/OpenCL i C#/Cloo oraz w języku C++) i na różnych platformach sprzętowych (procesorze - CPU i karcie graficznej - GPU). Za główną linię podziału można przyjąć architekturę, na której będą wykonywane obliczenia, które mają doprowadzić do optymalnego kierowania ruchem maszyny wieloosiowej. Schemat podziału przedstawiono na rys. 2.



Rys. 2. Podział różnych środowisk i platform, na których zaimplementowano PSO
Fig. 2. Division of different environments and platforms, on which PSO is implemented

Kartę graficzną można obrazowo przedstawić, jako dużą liczba procesorów. Jest to oczywiście duże przybliżenie, lecz w praktyce można rozumieć kartę graficzną, jako grupę kilkudziesięciu-kilkuset procesorów, które są w stanie pracować równoległe wykonując obliczenia na swojej partii danych.

Karty graficzne osiągają większą wydajność, ponieważ rezygnują z cache (pamięci podręcznej) oraz skomplikowanej jednostki sterującej na rzecz dużej liczby prostych elementów obliczeniowych. Procesor, w porównaniu do karty graficznej, posiada bardzo niewielkie możliwości zrównoleglenia obliczeń (ang. parallel processing). Możliwość tworzenia wątków w nowoczesnych językach programowania nie jest tym samym co możliwość traktowania pojedynczych jednostek obliczeniowych na GPU jako osobnych procesorów.

Do przedstawienia działania algorytmu PSO z wykorzystaniem karty graficznej użyto języka C# (przy wsparciu biblioteką Cloo) oraz języka C (przy wykorzystaniu OpenCL). Wykonano dwie implementacje algorytmu celem porównania szybkości działania oraz porównania wygody w programowaniu w dwóch różnych językach programowania. Kod obu programów jest napisany w taki sposób, aby można było pokazać obiektywną różnicę w szybkości działania na różnych środowiskach programistycznych z zachowaniem tego samego podzespołu GPU. Główną trudnością w przekształceniu kodu z wersji CPU na GPU okazała się być konieczność konwersji funkcji realizujących kolejne kroki algorytmu do postaci tzw. kerneli, które w praktyce wykonują te same obliczenia, lecz są uruchamiane przez kartę graficzną na kilkudziesięciu jednostkach jednocześnie. Początek programu to w głównej mierze inicjalizacja urządzenia graficznego do takiej postaci aby było ono widoczne dla programu jako obiekt, który będzie w stanie przetwarzać zadane funkcje. W systemach opartych o wiele kart graficznych można podzielić poszczególne karty, tak aby wykonywały odpowiednie, niedublujące się obliczenia, celem zwiększenia szybkości działania.

4. Funkcje testowe algorytmów optymalizacyjnych

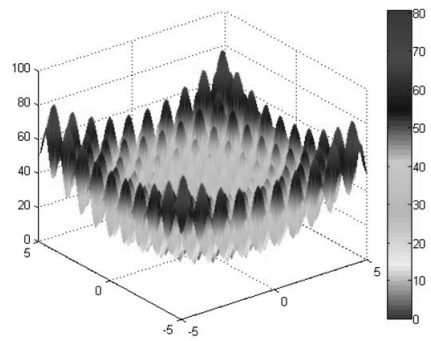
Efektywność metod optymalizacji była sprawdzana z wykorzystaniem klasycznych funkcji testowych, powszechnie stosowanych przy weryfikowaniu algorytmów optymalizacyjnych. W artykule opisano trzy funkcje, dla których wykonywane były badania czasu realizacji obliczeń.

Funkcja Rastrigina

Funkcja Rastrigina (1) jest często stosowana do weryfikacji algorytmów genetycznych i ewolucyjnych (rys.3). Określona jest poniższym wzorem:

$$F(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2), \tag{1}$$

Funkcja posiada jedno minimum globalne, które znajduje się w punkcie [0;0]. Wartość funkcji w tym punkcie wynosi 0. Znalezienie minimum globalnego jest bardzo trudne przy wykorzystaniu klasycznych algorytmów gradientowych ze względu na dużą liczbę minimów lokalnych [5].

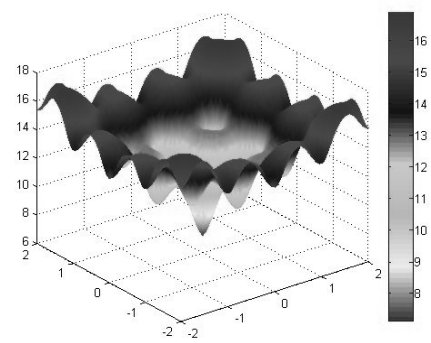


Rys. 3. Dwuwymiarowa funkcja Rastrigina
Fig. 3. Two-dimensional Rastrigin function

Funkcja Ackleya

Funkcja Ackleya została zdefiniowana początkowo jako problem dwuwymiarowy (rys.4), ale później uogólniono ją do *n* wymiarów. Minimum globalne funkcji (2) określone jest przez wektor argumentów x=[0;…;0], dla którego wartość funkcji wynosi 0. Funkcja uogólniona do n wymiarów wyraża się wzorem [6]:

$$F(x) = -20e^{-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{i=n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}, \tag{2}$$

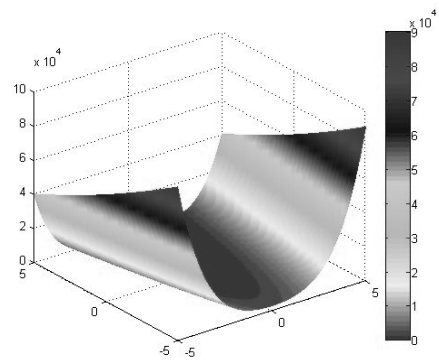


Rys. 4. Dwuwymiarowa funkcja Ackleya
Fig. 4. Two-dimensional Ackley function

Funkcja Rosenbrocka

Główną cechą funkcji Rosenbrocka jest to, że posiada „łagodną dolinę" (rys.5). Funkcja (3) zmienia się w niewielkim stopniu w otoczeniu minimum globalnego. Z tego względu znalezienie otoczenia minimum jest stosunkowo łatwe, jednak znalezienie samego minimum jest dużo trudniejsze. Minimum globalne funkcji znajduje się wewnątrz parabolicznego wgłębienia funkcji w punkcie *(x,y)* = [1;1], dla którego funkcja przyjmuje wartość *f(x,y)* = 0. Funkcję definiuje się wzorem [7]:

$$F(x, y) = (1-x)^2 + 100(y - x^2)^2, \tag{3}$$



Rys. 5. Dwuwymiarowa funkcja Rosenbrocka
Fig. 5. Two-dimensional Rosenbrock function

5. Pomiar szybkości działania algorytmu na różnych platformach sprzętowych

Algorytm PSO oraz funkcje testowe zostały implementowane na karcie graficznej w środowisku C/OpenCL i C#/Cloo oraz na procesorze w języku C++. Do estymacji różnicy czasu użyto funkcji odczytującej czas za pomocą liczby taktów zegara w danym momencie [8]. Różnica pomiędzy liczbą taktów zegara przed wejściem do pętli naliczającej iteracji, a liczbą taktów zegara po wyjściu z tej pętli podzielona przez liczbę taktów zegara w ciągu jednej sekundy pozwala na wyliczenie czasu obliczeń z dokładnością do setnych części sekundy.

Względny czas minimalizacji funkcji testowych zaprezentowany jest w postaci: (Czas minimalizacji danej funkcji testowej)/(Czas minimalizacji funkcji Rosenbrocka). Zakłada się, że czas wyznaczenia wartości funkcji Rosenbrocka wynosi 1. Wyniki zostały zaprezentowane w tab.1 oraz w postaci wykresu słupkowego na rys.6. Pomiar czasów realizacji obliczeń został wykonany dla następujących parametrów:

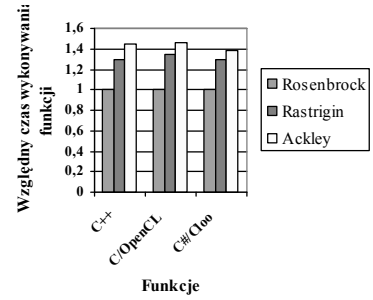
$$n=5,\ N=25,\ i=10\ 000$$

gdzie:

n - liczba wymiarów testowanej funkcji,
N - liczba cząstek w roju PSO,
i - liczba iteracji algorytmu minimalizacji.

Tab. 1. Względny czas wykonania obliczeń dla zadanych funkcji
Tab. 1. The relative calculation time for selected functions

	C/OpenCL (GPU)	C#/Cloo (GPU)	C++ (CPU)
Funkcja	**Względny czas wykonywania obliczeń**		
Rosebrock	1,00	1,00	1,00
Rastrigin	1,35	1,29	1,30
Ackley	1,46	1,38	1,45



Rys. 6. Wykres ilustrujący względne wartości czasów obliczeń dla zadanych funkcji testowych
Fig. 6. The graph illustrating the sum of relative calculation times for selected test functions

Z zaprezentowanych danych wynika, że problem minimalizacyjny zdefiniowany przez funkcję Rosenbrocka charakteryzuje się najmniejszą złożonością, w porównaniu z pozostałymi przedstawionymi funkcjami testowymi. Względne czasy minimalizacji danej funkcji zaimplementowanej w tym samym środowisku są do siebie zbliżone.

W dalszej kolejności zbadano zależność czasu obliczeń od liczby iteracji dla stałej liczby wymiarów i liczby cząstek dla funkcji Rosenbrocka. Testy szybkości działania algorytmów wykonywano przy zastosowaniu procesora AMD Phenom II X2 o taktowaniu 3,6 GHz oraz przy użyciu karty graficznej firmy AMD Radeon HD4850. Badania przeprowadzono dla następujących parametrów.

Dane: $n = 5,\ N = 10,\ 100 \le i \le 500000$

Wyniki przestawiono w tab.2.

Tab. 2. Zależność czasu wykonania obliczeń od liczby iteracji dla *n*=5 i *N*=10 dla procesora o taktowaniu 3,6 GHz.
Tab. 2. Dependence of the calculation time on the number of iterations for *n* = 5 and *N* = 10 for 3.6 GHz processor

	Czas wykonywania obliczeń [s]		
Liczba iteracji (i)	**C/OpenCL (GPU)**	**C#/Cloo (GPU)**	**C++ (CPU)**
100	0,09	0,21	0,02
1000	0,49	0,58	0,19
5000	1,34	1,64	0,94
10000	2,40	2,56	1,94
50000	5,91	6,12	8,41
100000	7,91	8,34	26,00
500000	18,21	19,31	91,11

Poniżej 50000 iteracji obliczenia na procesorze są szybsze od obliczeń na GPU. Wynika to z potrzeby inicjalizacji karty graficznej i kopiowania sporych porcji danych pomiędzy CPU i GPU. Dla większej liczby iteracji pojawia się coraz większa przewaga w szybkości działania algorytmu na GPU w stosunku do obliczeń na CPU.

Czas trwania obliczeń jest wprost proporcjonalny do liczby wymiarów problemu. Wynika to z założeń projektowania kerneli obliczeniowych GPU, a także z zasady obliczeń szeregowych na CPU.

Kolejnym etapem był pomiar czasu wykonywania obliczeń dla różnej liczby cząstek dla 5 wymiarów i 10000 iteracji.

Dane: $n=5,\ i=10000,\ 1 \le N \le 64$

Wyniki dla powyższych danych przestawiono w tab.3. Oznaczenia:

n - liczba wymiarów testowej funkcji,
N - liczba cząstek w roju PSO,
i - liczba iteracji algorytmu minimalizacji.

Tab. 3. Zależność czasu wykonania obliczeń od liczby cząstek dla n=5 i i=10000 dla procesora o taktowaniu 3,6 GHz
Tab. 3. Dependence of the calculation time on the number of particles for the n = 5 and i = 10000 3.6 GHz processor

	Czas wykonywania obliczeń [s]		
Liczba cząstek (N)	**C/OpenCL (GPU)**	**C#/Cloo (GPU)**	**C++ (CPU)**
1	2,31	2,39	0,19
5	2,36	2,48	0,92
10	2,45	2,54	1,90
15	2,55	2,61	2,81
20	2,78	2,82	3,60
25	3,28	3,41	4,65
32	4,20	4,31	5,90
48	5,10	5,21	8,10
64	5,90	6,10	11,80

Dla małej liczby cząstek obliczenia na karcie graficznej są wolniejsze niż te wykonywane na CPU. Wynika to z dłuższego czasu dostępu do pamięci karty graficznej. Przy około 15 cząstkach obliczenia na karcie graficznej zaczynają być korzystne. Pętla sterująca liczbą iteracji i kolejnymi ich wywołaniami jest wykonywana przez procesor komputera, więc taki wpływ jest wytłumaczalny. Dla typowych zastosowań obliczeniowych użycie GPU zamiast CPU przyniesie korzyści tam, gdzie złożoność obliczeniowa jest możliwie największa, a jednocześnie w możliwie największym stopniu da się zrównoleglić obliczenia. Z przedstawionych wyników (tab.2, tab.3) pod względem liczby iteracji oraz liczby cząstek widoczny jest stosunkowo mały wpływ na ogólną szybkość wykonywania obliczeń zarówno na CPU, jak i GPU.

6. Wnioski

W ramach artykułu wykonano implementację algorytmu Optymalizacji Rojem Cząstek (ang. Particle Swarm Optimization – PSO) na karcie graficznej komputera PC, który stanowi fragment generatora trajektorii ruchu w układzie sterowania maszyn. Zaimplementowany moduł optymalizacyjny jest częścią algorytmu wyznaczania maksymalnej prędkości posuwu maszyn wieloosiowych. Algorytm PSO służy do wyznaczenia maksymalnej chwilowej prędkości posuwu, zapewniającej minimalne błędy odtwarzania trajektorii ruchu.

Z przeprowadzonych badań wynika, że obliczenia na karcie graficznej nie w każdym przypadku okazały się być szybsze od podobnych obliczeń na procesorze. Jest to spowodowane w głównej mierze koniecznością przeprowadzenia względnie czasochłonnych operacji kopiowania danych do pamięci karty, a także operacji samej konfiguracji karty do obliczeń równoległych (co należy wykonywać za każdym razem, gdy program startuje). Dla mało złożonych problemów obliczenia na GPU są z tych powodów złym wyborem. Karta graficzna daje duże wsparcie przy złożonych, wielowymiarowych problemach, gdzie można w łatwy sposób zrównoleglić obliczenia. Na efektywność działania algorytmu PSO w implementacji GPU wpływ ma także szybkość procesora. Obliczenia wewnątrz algorytmu wykonuje karta graficzna, lecz odpowiednie kernele wywołuje procesor w pętli głównej programu. Technologia OpenCL rozwija się dość szybko, więc w przyszłości wzrost szybkości obliczeń dobrze zaprojektowanych programów może być większy niż obecnie.

Artykuł powstał w wyniku realizacji grantu 1138-F na rozwój młodych naukowców i uczestników studiów doktoranckich na Wydziale Fizyki, Astronomii i Informatyki Stosowanej na Uniwersytecie Mikołaja Kopernika w Toruniu.

7. Literatura

[1] Honczarenko J.: Obrabiarki sterowane numerycznie, WNT, 2009.
[2] Greblicki W.: Podstawy automatyki, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2006.
[3] Rutczyńska-Wdowiak K.: Modyfikacje algorytmu genetycznego w problemie identyfikacji modelu matematycznego silnika indukcyjnego, PAK 2007 nr 8, s. 60-63.
[4] Kennedy J., Eberhart R..: Particle Swarm Optimalization, International Conference on Neural Networks, 1995.
[5] Tsang P.K.: Problem solving with Genetic Algorithms, Science and Engineering Magazine, No. 6, 1992, str.14-17.
[6] Ackley D.H.: A connectionist machine for genetic hill climbing, Kluwer Academic Publishers, Boston, 1987.
[7] Dixon L.C., Mills D.: Effect of Rounding errors on the Variable Metric Method, Journal of Optimization Theory and Applications 80, 1992.
[8] Globa L.: Projektowanie harmonogramu dla systemów mikroprocesorowych, PAK 2010 nr 12, s.1554-1556.

otrzymano / received: 18.02.2013
przyjęto do druku / accepted: 01.04.2013 *artykuł recenzowany / revised paper*