Dariusz Sychel
Wydział Informatyki,
Zachodniopomorski Uniwersytet Technologiczny w Szczecinie
71-210 Szczecin,
Żołnierska 49

Redukcja czasu wykonania algorytmu Cannego dzięki zastosowaniu połączenia OpenMP z technologią NVIDIA CUDA

Słowa kluczowe: przetwarzanie równoległe, programowanie kart graficznych, CUDA, wykrywanie krawędzi, filtry splotowe, algorytm Cannego

1. Wstęp

Obecnie, jedną z popularnych technik umożliwiających zwiększenie wydajności obliczeń jest stosowanie wielordzeniowych procesorów. Jednak powstała dobra alternatywa dla tego rozwiązania, polegająca na wykorzystaniu programowalnych kart graficznych, które podobnie jak nowoczesne jednostki obliczeniowe CPU wykorzystują przetwarzanie równoległe.

W artykule autor skupia się na implementacji algorytmu Cannego, służącego do wykrywania krawędzi na obrazie opartego na połączeniu tych dwóch podejść. Testy przeprowadzone zostaną na maszynach o różnej specyfikacji, w celu sprawdzenia tego rozwiązania na słabszych pod względem mocy obliczeniowej kartach.

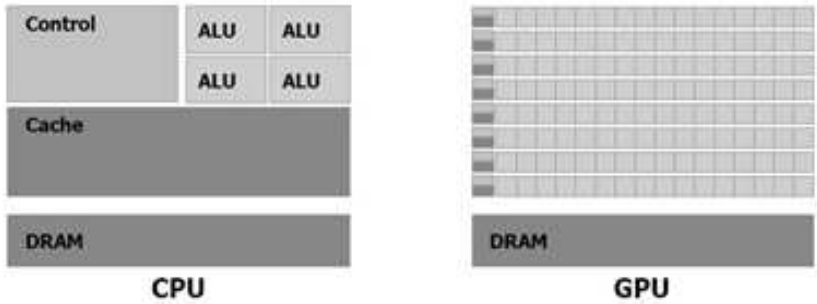
Podobne badania zostały opisane w pracach [1][2]. Podejścia tam przedstawione posiadają jednak pewne wady, zarówno w artykule [1] jak i [2] próbowano za pomocą GPU zrównoleglić algorytm oparty o BFS. Pierwsza skuteczna próba takiego rozwiązania została opisana w 2010 roku [3], 2 lata po publikacji [1]. Z artykułu tego wynika, że jest to złożone zadanie, a jego skuteczność zależy zarówno od ilości wierzchołków jak i zbalansowania grafu. W związku z tym w proponowanej implementacji zrezygnowano ze wsparcia karty graficznej dla tego kroku algorytmu na korzyść alternatywnego rozwiązania opartego o rekurencje oraz technologie OpenMP. Kolejną proponowaną zmianą jest zastąpienie stałej maski proponowanej w publikacji [2], na tworzoną dynamicznie co umożliwia sterowanie stopniem odszumienia.

W artykule opisana zostanie implementacja proponowanej wersji algorytmu oraz wyniki i analiza przeprowadzonych badań.

2. NVIDIA CUDA

Głównym zadaniem kart graficznych jest renderowanie grafiki 3D w czasie rzeczywistym. GPU (Graphic Procesor Unit) oferuje dużo większą w porównaniu do klasycznego CPU prędkość obliczeń na liczbach zmiennoprzecinkowych. Wyraźnie widać to po budowie takiego układu, w którym przeważają jednostki arytmetyczno-logiczne. Różnicę między CPU a GPU prezentuje rys. 1.

NVIDIA CUDA (Compute Unified Device Architecture) [4] jest równoległą architekturą obliczeniową stworzoną w 2006 roku przez firmę NVIDIA i do dzisiaj stosowaną w jej kartach graficznych. Architektura ta pozwala na programowanie kart graficznych w takich językach jak C, C++ czy Fortran, o ile w systemie znajdują się odpowiednie sterowniki oraz SDK. Dzięki czemu ułatwia zadanie programisty.



Rys. 1. Porównanie budowy CPU z GPU (źródło: [4])

Urządzenia zgodne z CUDA są stworzone w oparciu o architekturę SIMT (single instruction multiple thread) [4]. Każda z kart graficznych posiada macierz multiprocesorów strumieniowych, między którymi rozdzielane są wątki z bloku. Multiprocesor jest zbudowany tak, aby przetwarzał jednocześnie wiele wątków. Budowa taka pozwala na równoczesne wykonywanie znacznie większej liczby wątków niż na wielordzeniowych procesorach.

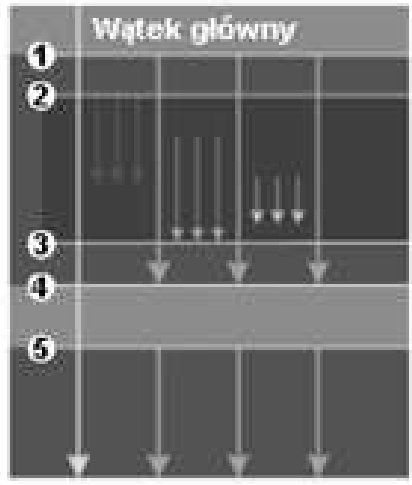
3. OpenMP

OpenMP [5] jest interfejsem programowania aplikacji na systemach z pamięcią współdzieloną, zawierających wiele procesorów. Opisuje on podział kodu, który będzie wykonywany równolegle, pomiędzy wątki na komputerach o różnej liczbie procesorów. Podobnie jak CUDA pozwala na tworzenie programów w językach: C, C++ oraz Fortran.

OpenMP w trakcie swojego działania wykorzystuje model zwany „Fork-Join" [6]. Na rys. 2 pokazane jest działanie tego modelu. Aplikacja rozpoczyna swoją pracę jako pojedynczy wątek. W momencie napotkania przez wątek bloku kodu, który ma zostać

zrównoleglony (1, 2, 5), tworzy on serie wątków pobocznych a sam zostaje wątkiem głównym.

Wątki poboczne w przypadku napotkania na zagnieżdżony blok kodu, który ma zostać zrównoleglony (2), także są w stanie utworzyć własne wątki poboczne. W momencie zakończenia wykonywania zrównoleglonego bloku kodu, wątki poboczne są kasowane, a z bloku wychodzi jedynie wątek główny dla danej sekcji (3, 4).



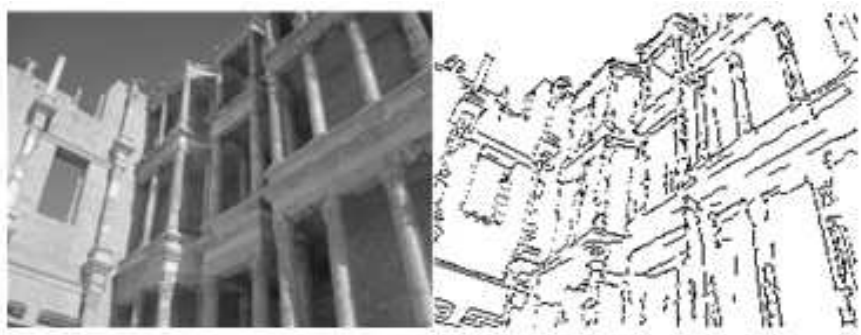
Rys. 2. Podział wątków (źródło: [6])

4. Algorytm Cannego

Algorytm Cannego jest algorytmem, którego zadaniem jest rozpoznanie krawędzi na zadanym obrazie. Metoda ta została przedstawiona przez Johna F. Cannego w roku 1986 [7]. Przykładowy efekt działania algorytmu znajduje się na rysunku 3.

Celem algorytmu jest spełnienie trzech kryteriów [8]:

- Dobra detekcja - algorytm powinien minimalizować liczbę błędów detekcji. Do błędów tego typu zalicza się zarówno błędy powstałe poprzez rozpoznanie krawędzi w miejscu, gdzie tak naprawdę nie występują oraz błędy związane z pominięciem istniejącej krawędzi.
- Dobre umiejscowienie - punkt rozpoznany jako element krawędzi powinien znajdować się jak najbliżej rzeczywistego środka krawędzi.
- Pojedyncza odpowiedź - wyznaczone krawędzie nie powinny mieć szerokości większej niż jeden piksel.



Rys. 3. Przykład działania algorytmu Cannego (źródło: własne)

Cel algorytmu można osiągnąć poprzez wykonanie czterech kroków:

1. Redukcja szumu - w tym celu można dokonać splotu obrazu z filtrem Gaussa.
2. Obliczenie natężenia oraz kierunku gradientu dla każdego punktu obrazu – w tym celu można posłużyć jednym z operatorów służących do wykrywania krawędzi (skorzystano z operatora Sobela):

$$f = \sqrt{Gx + Gy} \tag{1}$$

$$\alpha = arctg\left(\frac{Gy}{Gx}\right) \tag{2}$$

gdzie: f - macierz zawierająca natężenia gradientów, α - macierz zawierająca kierunki gradientów,

Gx, Gy - splot obrazu oryginalnego z filtrem dla kierunku x, y.

3. Usuwanie niemaksymalnych pikseli - zadaniem tego kroku jest uzyskanie krawędzi o szerokości jednego piksela, w tym celu bada się wyznaczone wcześniej kierunki oraz natężenia gradientu w sąsiadujących punktach i na tej podstawie określa się czy dany punkt jest istotny, czy musi zostać usunięty.
4. Progowanie z histerezą - ostatni krok algorytmu ma za zadanie wyeliminowanie słabych krawędzi w taki sposób, aby nie powstały luki w rozpoznanych krawędziach.

5. Implementacja

Podczas dokonywania operacji splotu obrazu, wartość każdego punktu na obrazie wynikowym jest wyznaczana niezależnie, krok 1 można zrównoleglić. Jedynym utrudnieniem jakie występuje w tej fazie algorytmu, jest samo wygenerowanie maski dla filtru (rys. 4, 1A), w związku z koniecznością obliczenia sumy elementów [9]:

$$h_g(n_1, n_2) = e^{\frac{-\left(n_1^2 + n_{21}^2\right)}{2\sigma^2}} \tag{3}$$

$$h(n_1, n_2) = \frac{h_g(n_1,n_2)}{\sum n_1 \sum n_2 h_g} \tag{4}$$

gdzie: (n1, n2) - punkt maski.

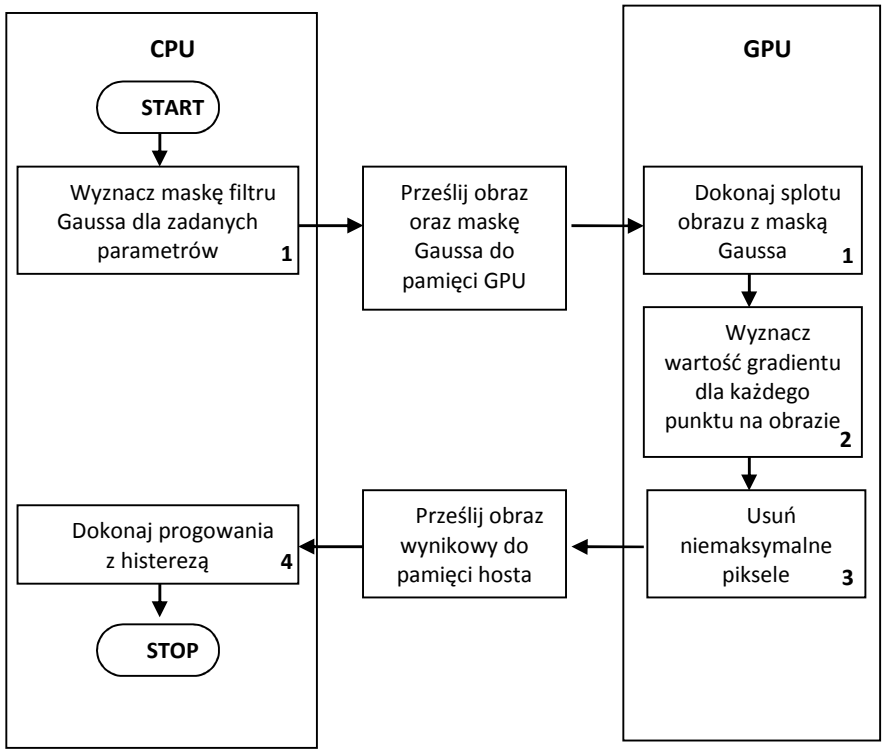
Maska taka nie może być zbyt duża, np. dla σ = 2,0 już przy wymiarach maski 10x10 utracone mogą zostać istotne krawędzie, wraz ze zwiększaniem σ efekt ten się pogarsza. Co za tym idzie nieefektywne staje się przenoszenie jej między pamięcią karty graficznej, a pamięcią hosta w celu dokonania sumowania na CPU. Dodatkowo mały rozmiar maski sprawia, że wykorzystanie OpenMP staje się nieopłacalne, dlatego część ta obliczana jest sekwencyjnie. Po wyznaczeniu wartości dla filtru maska przesyłana jest do pamięci karty i tam dokonywana jest operacja splotu. Problem ten może zostać ominięty poprzez zastosowanie stałej maski [2]. Takie podejście uniemożliwia jednak sterowanie stopniem odszumienia.

Krok 2 w związku z faktem, że zarówno natężenie gradientu jak i jego kierunek dla każdego punktu można liczyć niezależnie, o ile nie nadpisujemy oryginalnych wartości obrazu w trakcie tego procesu, także może zostać zrównoleglony. Wartości te wyznaczane są na podstawie wzorów (1) i (2).

W kroku 3, ponieważ usunięcie niemaksymalnego piksela nie wpłynie na wynik obliczeń dla sąsiednich pikseli, każdy punkt możemy rozpatrywać niezależnie. W kroku tym pobieramy dwa sąsiednie dla badanego punkty, leżące na prostej nachylonej pod wyznaczonym dla tego punktu kątem i na podstawie natężenie gradientu w tych punktach, decydujemy czy badany punkt powinien zostać zachowany. Dodatkowo w proponowanym rozwiązaniu w kroku tym przeprowadzana jest operacja progowania, krawędzią powyżej progu górnego nadawana jest wartość 255, tym które znajdują się między progiem dolnym a górnym wartość 128, pozostałe są zerowane.

Dla kroku 4 autor proponuje zmianę podejścia w stosunku do proponowanych w [1][2] poprzez rezygnacje z zastosowania dla tego kroku karty graficznej oraz zmianę zastosowanych tam algorytmów na zrównoleglony przy użyciu OpenMP algorytm rekurencyjny w celu przyspieszenia obliczeń. Proponowany algorytm, poszukuje punktów o wartości równej 255, następnie bada ich otocznie, każdemu sąsiedniemu punktowi o wartości wynoszącej 128, nadaje się wartość 255 oraz rozpoczyna się badanie jego otoczenia.

Rysunek 4 przedstawia algorytm Cannego za pomocą schematu blokowego, uwzględniając podział operacji między CPU a GPU.



Rys. 4. Zaplanowany rozkład operacji (źródło: własne)

6. Eksperymenty porównawcze

W celu określenia skuteczności badanego rozwiązania przeprowadzono dwa rodzaje badań. Kryterium pierwszego badania był czas wykonania w [ms]. Badanie polegało na porównaniu czasu wykonania czterech wersji algorytmu:

- algorytmu sekwencyjnego napisanego w języku C++ oznaczonego jako [C++],
- algorytmu zrównoleglonego przy użyciu OpenMP oznaczonego jako [OpenMP],
- algorytmu ze wsparciem przez kartę graficzną oznaczonego jako [CUDA],
- algorytmu łączącego w sobie wsparcie karty graficznej oraz zrównoleglenie poprzez zastosowanie technologii OpenMP oznaczonego jako [CUDA + OpenMP]. Kryterium drugiego badania była zgodność obrazu wygenerowanego sekwencyjnie z obrazem uzyskanym przy wsparciu GPU. Badanie to polegało na wygenerowaniu obrazu różnicowego, a następnie jego analizie w celu wykrycia czy istnieją różnice między obrazami oraz czy ewentualne różnice są istotne.

Platformy badawcze

Badania zostały przeprowadzone na sprzęcie różnej jakości. Urządzenia te różniły się między innymi liczbą rdzeni oraz mocą obliczeniową kart graficznych.

W tab. 1 znajduje się spis platform, na których testowany był algorytm w postaci tabelarycznej. Platforma 1 jest najmniej wydajnym komputerem z używanych w badaniu. Natomiast platforma 3 najbardziej, zawiera ona z 2 procesory Xeon oraz najwydajniejszą kartę graficzną z tych 3 platform.

Tab. 1. Wykaz platform badawczych

	Platforma 1	Platforma 2	Platforma 3
Procesor	Intel Core i5-480M	AMD X8 FX 8150	2 Intel Xeon E5440
Max. taktowanie	2,667GHz	3,6GHz	2,833GHz
Liczba rdzeni	2	8	4
Procesory logiczne	4	8	4
Karta graficzna	GeForce GT 540M	GeForce GTX 650	GeForce GTX 580
Wersja CC	2,1	3,0	2,0
Liczba rdzeni CUDA	96	384	512
Częstotliwość zegara	1,344GHz	1,0585GHz	1,544GHz
Max. ilość wątków	1024	1024	1024

Wyniki badań

Wyniki przedstawione w poniższych tabelach są wynikami uśrednionymi z pięciu serii testów. Tabele przedstawiają wyniki dla rozwiązania sekwencyjnego, rozwiązania opartego o OpenMP, rozwiązania wykorzystujące wsparcie GPU, oraz połączenia tych dwóch ostatnich. Dla każdej z trzech platform.

Tab. 2. Czas wykonania w [ms] dla platformy nr 1

Wielkość obrazu	C++	OpenMP	CUDA	CUDA +OpenMP
220x165	5,66	4,86	1,76	1,58
1000x1000	138,34	89,47	16,46	15,67
2000x2000	574,63	354,57	67,37	59,12
3000x3000	1253,24	1065,17	157,45	133,43
4000x4000	2838,80	1817,42	286,98	246,56

Dzięki wsparciu przez kartę graficzną dla komputera 1 przy przetwarzaniu obrazu 4000x4000px uzyskano prawie 10 krotne przyspieszenie w porównaniu z przetwarzaniem sekwencyjnym, zastosowanie dodatkowo OpenMP pozwoliło na uzyskanie prawie 12 krotnego przyspieszenia. Czas ten spadł z 2838,80ms do 246,56ms.

Tab. 3. Czas wykonania w [ms] dla platformy nr 2

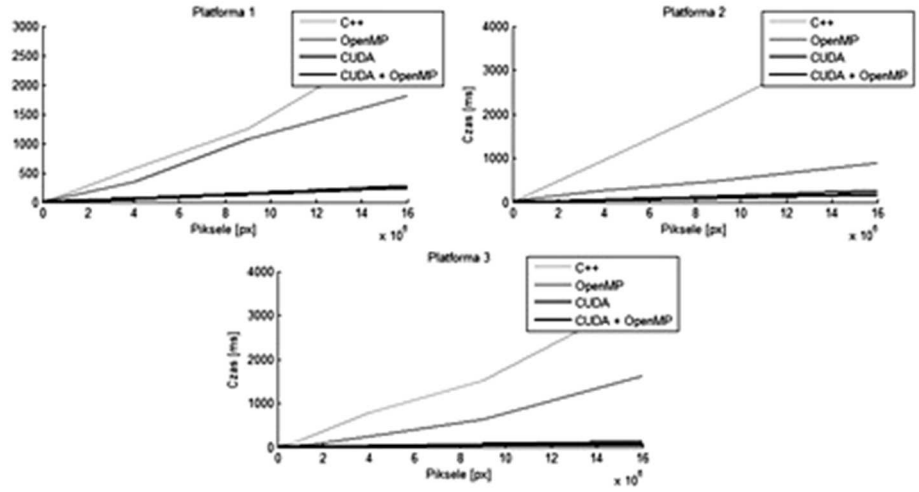
Wielkość obrazu	C++	OpenMP	CUDA	CUDA +OpenMP
220x165	8,22	4,09	1,79	1,58
1000x1000	247,17	97,56	15,55	11,13
2000x2000	963,52	262,28	60,87	46,13
3000x3000	2150,81	492,51	144,98	100,26
4000x4000	3991,20	890,84	274,91	175,60

Zastosowanie średniej klasy karty graficznej, typowej dla komputerów domowych umożliwiło dla tego obrazu uzyskanie 14,5 krotnego przyspieszenia, dzięki dodatkowemu zrównolegleniu części kodu przy użyciu OpenMP uzyskano prawie 23 krotne przyspieszenie. Z początkowych 3991,20ms spadł do 175,60ms.

Tab. 4. Czas wykonania w [ms] dla platformy nr 3

Wielkość obrazu	C++	OpenMP	CUDA	CUDA +OpenMP
220x165	10,78	3,65	1,72	1,05
1000x1000	171,92	41,58	8,20	6,41
2000x2000	775,79	256,58	31,33	20,74
3000x3000	1525,60	628,09	70,91	45,30
4000x4000	3402,15	1624,94	131,30	80,40

Zastosowanie profesjonalnej karty graficznej dla obrazu o tym samym rozmiarze spowodowało skrócenie czasu obliczeń prawie 26 krotnie w porównaniu z sekwencyjnym algorytmem wykonywanym na procesorze XEON, po dodaniu OpenMP liczba ta wzrosła do 42. Czas wykonania spadł z 3402,15ms do 80,40ms.



Rys. 5. Zestawienie wyników dla wszystkich platform

Na rys. 5 znajduje się zestawienie wykresów z wynikami dla wszystkich badanych platform.

Test różnicowy nie wykazał znaczących różnic między obrazami utworzonymi sekwencyjnie, a z wykorzystaniem karty graficznej.

7. Wnioski

Celem badania była redukcja czasu wykonania algorytmu Cannego, dzięki wykorzystaniu potencjału wielordzeniowych procesorów oraz wsparcia ze strony GPU. Dzięki takiemu rozwiązaniu udało się uzyskać nawet 42 krotne przyspieszenie w porównaniu do rozwiązania sekwencyjnego. Czas wykonania dla obrazu o rozmiarach 4000x4000 dla platformy 3 spadł z 3402,15ms do 80,40ms.

Wyniki badań pokazują, że dzięki połączeniu klasycznego podejścia ze wspomaganiem w postaci karty graficznej w celu zrównoleglenia algorytmu Cannego uzyskano znaczące skrócenie czasu obliczeń od 12 do 42 razy dla obrazu 4000x4000 w stosunku do tradycyjnego, sekwencyjnego przetwarzania. Fakt ten może zostać wykorzystany w wielu dziedzinach nauki, techniki, przemysłu, gdzie algorytmy wykrywania krawędzi mają zastosowanie, jak na przykład:

- szybsze przetwarzania obrazów medycznych (w konsekwencji szybsza diagnoza),
- analiza zdjęć satelitarnych (programy typu GIS),
- usprawnienie procesu digitalizacji dokumentów - OCR (wyższa efektywność instytucji mogących operować na dokumentach w wersji cyfrowej).

W celu uzyskania tego efektu nie jest wymagana droga karta graficzna, dlatego rozwiązanie to może mieć także zastosowanie w programach służących do przetwarzania obrazów pisanych na komputery domowe czy laptopy.

8. Literatura

1. Luo Y., Duraiswami R.: Canny edge detection on NVIDIA CUDA. 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 43, 1, 2008, 1-8
2. Ogawa K., Ito Y., Nakano K.: Efficient Canny Edge Detection Using a GPU. ICNC '10 Proceedings of the 2010 First International Conference on Networking and Computing, 2010, 279-280
3. Luo L., Wong M., Hwu M.: An effective GPU implementation of breadth-first search. Design Automation Conference (DAC), 2010 47th ACM/IEEE, 2010, 52-55 [4] NVIDIA.: NVidia CUDA C Programing Guide, Version 4.2.
4. https://www.math.umass.edu/~johnston/CUDA_WG_2012/CUDA_C_Programming_Guide.pdf, dostęp 06.04.2013
5. Blaise B.: OpenMP. https://computing.llnl.gov/tutorials/openMP/, dostęp 06.04.2013
6. Gatlin K. S., Isensee P.: Reap the Benefits of Multithreading without All the Work. http://msdn.microsoft.com/pl-pl/magazine/cc163717%28en-us%29.aspx, dostęp 06.04.2013
7. Canny F.J.: A Computational Approach to Edge Detection. J-IEEE-PAMI, 8, 6, 1986, 679-698.
8. Pratt W.K..: Digital Image Processing - PIKS Scientific Inside. John Wiley & Sons, 2007.
9. The MathWorks, Inc.: Documentation Center. http://www.mathworks.com/help/images/ref/fspecial.html, dostęp 06.04.2013

Streszczenie

Artykuł prezentuje alternatywne podejście do programowania równoległego poprzez wykorzystanie programowalnych kart graficznych w celu wsparcia obliczeń, oraz połączenie tego podejścia z klasycznym zrównolegleniem opartym o wielordzeniowe procesory. Przeprowadzone testy przedstawiają zysk czasu jaki można uzyskać dzięki odpowiedniemu połączeniu OpenMP z technologią CUDA w obliczeniach związanych z wykrywaniem krawędzi na obrazie rastrowym przy użyciu algorytmu Cannego. Badania przeprowadzone zostały na sprzęcie różnej jakości. Napisane algorytmy są zgodne z CC 1,0 (zdolność obliczeniowa karty graficznej).

Abstract

This paper presents an alternative approach to parallel programming by using programmable graphics card to support calculations and combines this approach with a classical parallelization based on multi-core processors. The tests show the gain time that can be achieved through a combination of OpenMP with CUDA technology in the calculation of the edge detection on the raster image using the Canny's algorithm. Tests were carried out on the equipment of varying quality. The algorithms are compatible with CC 1.0 (compute capability graphics card.)

Słowa kluczowe: parallel processing, programming, graphics cards, CUDA, edge detection filters, the Canny's algorithm