AUTOMATYKA • 2009 • Tom 13 • Zeszyt 3

Mirosław Jabłoński*, Zbigniew Bubliński*

Akceleracja algorytmów przetwarzania obrazów z wykorzystaniem zasobów karty graficznej

1. Wprowadzenie

Przetwarzanie obrazów cyfrowych wiąże się z koniecznością obróbki ogromnej ilości danych i w konsekwencji oznacza, że nawet nieskomplikowane algorytmy zajmują dużo czasu procesora CPU (*Central Processing Unit*) i znacznie go obciążają. Celowe więc wydaje się przeniesienie przynajmniej części obliczeń do karty graficznej, której procesor dysponuje dużo większym stopniem równoległości, a pozostaje w znacznej mierze niewykorzystany. W dalszej części niniejszej publikacji przedstawiono przykład wykorzystania zasobów karty graficznej do akceleracji kontekstowych operacji przetwarzania obrazów cyfrowych. Na przykładzie toru wizyjnego dokonano ewaluacji przyspieszenia karty graficznej dla różnej organizacji wątków wykonywanych współbieżnie na wielu procesorach układu karty graficznej. Uzyskane wyniki potwierdzają zasadność przedstawionego sposobu akceleracji w implementacji systemów wizyjnych.

2. Procesor karty graficznej jako element obliczeniowy

Procesory GPU (*Graphics Processing Unit*) współczesnych kart graficznych, chociaż dedykowane do renderowania obrazów w celu wizualizacji oraz wspomagania oprogramowania CAD (*Computer Aided Design*), coraz częściej [3, 4, 5] są stosowane jako alternatywny do procesorów CPU element obliczeniowy. Technika wykorzystania zasobów procesora karty graficznej określana jest mianem GP GPU (*General-Purpose computing on Graphics Processing Unit*). Uniwersalne użycie zasobów karty graficznej możliwe jest dzięki zasobom pamięciowym i obliczeniowym karty graficznej oraz wykorzystaniu dedykowanego oprogramowania, które za pomocą języka wysokiego poziomu umożliwia tworzenie wydajnych aplikacji oraz ich integrację w środowisku systemu operacyjnego komputera. Pakietem dedykowanym do implementacji obliczeń na wielordzeniowych procesorach firmy NVIDIA z poziomu języka C jest CUDA (*Compute Unified Device Architecture*).

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

Zaletą procesorów GPU jest wysoki stopień równoległości danych oraz wiele rdzeni zmiennoprzecinkowych procesorów zdolnych do współbieżnego wykonywania wielu wątków. Masowa równoległość w zastosowanym modelu karty 8800GT z układem GeForce G92 objawia się poprzez dostępność 112 rdzeni, z których każdy może realizować 96 wątków. Nie mniej istotna jest organizacja pamięci i przepustowości magistral, które umożliwiają efektywne wykorzystanie zasobów GPU – magistrala PCI Express x16 w wersji 2.0 zapewnia transfer na poziomie 16 GB/s, pamięć karty graficznej dostępna przez 256-bitową magistralę pozwala przesyłać dane z prędkością 57,6 GB/s, a sam procesor GPU jest taktowany zegarem 1,5 GHz. Zasoby pamięciowe karty graficznej zorganizowane są w kilka poziomów o różnej pojemności oraz zakresie dostępu:

- 1024 zmiennoprzecinkowe rejestry procesorów o wielkości 32 bitów,
- 16 KB pamięci dostępnej dla klastra (8 procesorów),
- 512 MB pamięci globalnej DDR.

Programowy model CUDA zakłada [2] organizację zasobów obliczeniowych GPU w dwuwymiarową siatkę (*grid*) której elementy składają się z bloków (*block*). Każdy blok, dwu- lub trójwymiarowy, składa się z wątków, które mogą współdzielić dane. Wątki wykonywane w różnych blokach nie mogą się jednak komunikować ze sobą. Za przydział czasu procesorów do współbieżnie wykonywanych wątków odpowiada sprzętowy moduł zarządzania Thread Execution Manager [1]. Zarówno rozmiar siatki, jak i organizacja wątków w blokach ustalane są przez programistę, który decyduje w ten sposób o stopniu zrównoleglenia wykonywanego programu.

Karty graficzne z założenia przeznaczone są do generowania, przetwarzania i wizualizacji sztucznych obrazów. Na niektórych etapach tworzenia grafiki struktura danych jest zgodna z organizacją rastrowych obrazów uzyskanych z kamery wizyjnej. Dzięki temu przetwarzanie i analiza obrazów jest szczególnym obszarem zastosowań obliczeń na GPU. W dalszej części artykułu zawarto wyniki badań nad realizacją toru wizyjnego na karcie graficznej.

3. Przetwarzanie obrazów na karcie graficznej

Do realizacji na karcie graficznej wybrano tor wizyjny złożony z kilku elementarnych operacji przetwarzania obrazów o kontekście przestrzennym. Zadanie obliczeniowe stanowiła sekwencja filtrów służąca do detekcji krawędzi w monochromatycznym obrazie o rozmiarach 512×512 pikseli:

1) filtracja uśredniająca w oknie o kontekście 5×5,
2) binaryzacja z progiem ustalanym programowo,
3) operacja morfologicznego otwarcia w oknie o kontekście 3×3,
4) operacja morfologicznego domknięcia w oknie o kontekście 3×3,
5) gradient Sobela w oknie o kontekście 3×3.

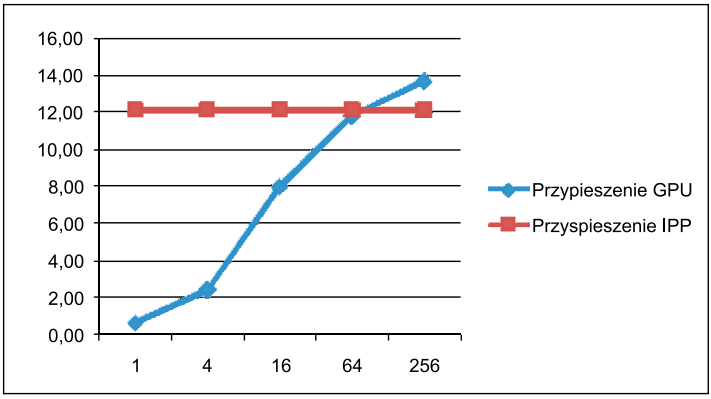
Badania nad implementacją pojedynczego operatora konwolucji przedstawiono w pracy [6]. Autorzy przeprowadzili tam szczegółową analizę złożoności obliczeniowej filtrów realizowanych na CPU i GPU metodą szybkiej transformaty Fouriera, konwolucji w dziedzinie koordynat przestrzennych oraz dokonali jej optymalizacji poprzez separację współczynników filtrów. Opisane rezultaty uwzględniały również różne rozmiary obrazów 256×256, 512×512 i 1024×1024, różne formaty danych: 16 bitów oraz 32 bity oraz różne wymiary macierzy współczynników filtra (5–90). Do implementacji autorzy użyli języka HLSL (*High Level Shader Language*) stanowiącego rozszerzenie biblioteki DirectX. Środowisko to nie umożliwia jawnego specyfikowania stopnia zrównoleglenia, jak to ma miejsce w bibliotece CUDA.

W stosowanych praktycznie systemach wizyjnych najczęściej wykorzystuje się więcej niż jeden filtr. W niniejszej pracy podjęto zatem próbę przebadania przyspieszenia obliczeń dla złożonego toru wizyjnego w zależności od stopnia zrównoleglenia. Wprawdzie wcześniej wspomniana publikacja zawierała wyniki szybkości przetwarzania dla wielu filtrów wykonywanych na jednym obrazie, jednak eksperymenty przeprowadzono jedynie dla filtrów separowalnych, co znacznie zawęża zakres zastosowań. Dlatego też, jako czynnik zmienny wybrano liczbę wątków przydzielonych do bloku podczas przetwarzania obrazu przez kolejne filtry i operacje morfologiczne. Uzasadniony jest pomiar przyspieszenia obliczeń dla sekwencji operacji na obrazie, ponieważ sposób składowania pośrednich wyników w buforach pamięciowych karty graficznej oraz zależności pomiędzy równolegle działającymi procesorami również mogą mieć wpływ na szybkość działania aplikacji.

Badania polegały na rozdzieleniu zadania obliczeniowego pomiędzy procesory zorganizowane w siatkę o zmieniających się rozmiarach oraz mogących korzystać z określonej liczby wątków wykonywanych jednocześnie. Podczas wstępnych badań stwierdzono, że w przypadku siatki zawierającej 16 lub więcej bloków decydujące znaczenie ma liczba wątków przypisana do bloku. Z tego właśnie względu prezentowane wyniki dotyczą tylko zmieniającej się liczby wątków w bloku. Całkowita liczba wątków jest natomiast stała i wynosi 16 384. Mierzono czas wykonania programu podzielonego w każdym bloku na zadaną liczbę wątków, a następnie porównano uzyskane wyniki z czasami wykonania klasycznego jednowątkowego programu korzystającego z CPU. Uzyskane rezultaty zestawiono w tabeli 1 oraz pokazano na rysunku 1.

Tabela 1
Czas wykonania i przyspieszenie w zależności od liczby wątków w bloku

Liczba wątków w bloku	Czas wykonania [ms]	Przyspieszenie względem procesora CPU	Przyspieszenie względem najgorszego czasu GPU
1	72,59	0,64	1,00
4	18,92	2,47	3,84
16	5,82	8,03	12,47
64	3,94	11,87	18,44
256	3,40	13,73	21,33



Rys. 1. Przyspieszenie biblioteki IPP i GPU względem implementacji na CPU, w zależności od liczby wątków w bloku

Przedstawione wyniki wskazują, że podział wykonywanego programu na znaczną liczbę wątków daje dobre rezultaty jedynie prze dobrej ich organizacji. Duża liczba bloków w siatce daje przyspieszenie względem CPU poniżej wartości 1,0. Przy niewielkiej liczbie wątków w bloku (4, 16) możliwe jest uzyskanie znacznego wzrostu przyspieszenia. Dalsze zwiększanie liczby wątków nie daje już tak dużego przyrostu przyspieszenia. Po przekroczeniu 256 wątków na blok program przestaje działać poprawnie. Sterownik karty zgłasza wówczas błędy, co najprawdopodobniej wiąże się z przekroczeniem dostępnej pamięci lokalnej procesorów jednostki GPU. Warto zwrócić uwagę, że tylko w przypadku pojedynczego wątku w bloku, otrzymany wynik jest gorszy niż rezultat przetwarzania z wykorzystaniem CPU (czas wykonania na CPU to 46,72 ms). Godny podkreślenia jest również fakt, że przy 256 wątkach w bloku czas działania programu na GPU jest krótszy niż uzyskany na CPU z wykorzystaniem dedykowanej do przetwarzania obrazów biblioteki Intel IPP (*Integrated Performance Primitives*), pozwalającej uzyskać czas 3,85 ms. Należy zaznaczyć, że implementacja algorytmu wykonującego działania toru wizyjnego nie była w żaden sposób optymalizowana. Podczas eksperymentów aplikację GPU poddawano jedynie dynamicznej parametryzacji ze względu na organizację wątków w blokach.

Uogólnienie uzyskanych wyników na inne algorytmy i platformy GPU niż te opisane w artykule jest kłopotliwe, gdyż zależą one zarówno od wykonywanego programu (jego stopnia skomplikowania), jak i samej karty graficznej – możliwości obliczeniowych oraz zadań, jakie ona w danym momencie realizuje. Warto wspomnieć, że wielowątkowe wykonanie programu użytkownika to zadanie dodatkowe, zaś podstawowym jest renderowanie obrazu wyświetlanego na monitorze komputera. Biblioteka CUDA może być wykorzystana jedynie do urządzeń wyposażonych w układy GPU firmy NVIDIA – w przypadku GPU firmy ATI konieczne jest użycie innych narzędzi, przez co stworzona aplikacja wizyjna nie jest w pełni przenośna.

4. Wnioski

Przeprowadzone badania i uzyskane wyniki potwierdzają celowość wykorzystania biblioteki CUDA i zasobów karty graficznej jako prostego sposobu na zrównoleglenie wykonywanych obliczeń, a co za tym idzie, na skrócenie czasu obliczeń w przetwarzaniu obrazów. Wyniki pokazują dobitnie, że przyspieszenie wykonania programu na GPU zależy nie tylko od liczby wątków które wykonują ten sam kod, ale również od ich organizacji i dostępności zasobów pamięciowych. Objawia się tutaj efekt współdzielenia danych przez wątki w obszarze jednego bloku – jak zauważono w rozdziale 2, wątki wykonywane w odrębnych blokach nie mogą się ze sobą komunikować. Wysokie zrównoleglenie elementów obliczeniowych GPU i współbieżne wykonanie wielu wątków jest warunkiem koniecznym ale nie wystarczającym do efektywnego przetwarzania obrazów. Przedstawione rezultaty są obiecujące. Potrzebne są jednak dalsze szczegółowe badania i eksperymenty, które w konsekwencji pozwolą na bardziej efektywny dobór struktury zasobów (liczby bloków i przydzielanych wątków) do wykonywanego zadania i jego rozmiarów oraz do możliwości użytej karty graficznej.

Podziękowania

Praca została zrealizowana w ramach badań statutowych – umowa AGH nr 11.11.120.612. Autorzy dziękują również studentom wydziału EAIE Akademii Górniczo-Hutniczej: Panu Pawłowi Sawie oraz Panu Norbertowi Śledziowi za udział w realizacji przedstawionych badań.

Literatura

[1] Halfhill T., *Parallel processing with CUDA: Nvidia's High-Performance Computing Platform Uses Massive Multithreading Microprocessor*. 2008, http://www.mpronline.com.

[2] Nvidia, *CUDA* Programming *Guide Version 1.1.* 2007, http://www. nvidia.com.

[3] Ruiz A., Ujaldon M., Andrades J.A., Becerra J., Kun Huang Pan T., Saltz J., *The GPU on biomedical image processing for color and phenotype analysis*. Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering, 2007, 1124–1128.

[4] Sengupta S., Harris M., Zhang Y., Owens J., *Scan primitives for GPU computing.* Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, Eurographics Association , Switzerland, 2007, 97–106.

[5] Yang R., Welch G., *Fast Image Segmentation and Smoothing Using Commodity Graphics Hardware*. Journal of Graphics Tools, 2002, 91–100.

[6] Fialka O., Cadik M., *FFT and Convolution Performance in Image Filtering on GPU.* Tenth International Conference on Information Visualization, 2006, IEEE, 609–614.