STUDIA INFORMATICA 2010
Volume 31 Number 2A (89)

Michał WIDERA
Politechnika Śląska, Instytut Informatyki

SPRZĘTOWE WSPOMAGANIE PRZETWARZANIA DANYCH W STRUMIENIOWYM SYSTEMIE ZARZĄDZANIA DANYMI

Streszczenie. W artykule przedstawiono architekturę strumieniowego systemu zarządzania danymi wyposażonego w sprzętowe wspomaganie procesu przetwarzania danych. System zarządzania rozbudowano o biblioteki wspierające technologię CUDA. Przedstawiono przykład użycia w oparciu o dedykowany język deklaratywny oraz projekt wewnętrznej struktury zmodyfikowanego systemu.

Słowa kluczowe: Strumieniowe bazy danych, przetwarzanie sygnałów, Systemy Monitorujące, CUDA

HARDWARE SUPPORT FOR DATA PROCESSING IN DATA STREAM MANAGEMENT SYSTEM

Summary. This paper present data stream management system architecture supported by hardware accelerated data processing module. Projected system was extended by CUDA libraries. Paper presents use case based on dedicated query language and projects of inner structure of modified system.

Keywords: Data Stream Management System, Signal processing, Monitoring Systems, CUDA

1. Wstęp

Zagadnienie wspomagania sprzętowego, równoległego przetwarzania dużych ilości danych jest analizowane w literaturze od dłuższego czasu [1]. Równocześnie, równie dokładnie przedstawiono na forum naukowym wyniki prac nad Strumieniowymi Systemami Zarządzania Danych [2, 3] DSMS – *Data Stream Management System*. Prowadząc badania literaturowe spotkamy jednak niewiele prac, w których przeprowadzono analizę korzyści oraz proble-

mów wynikających z połączenia wniosków wynikłych z obu kierunków badań. Tego typu sytuacja sugeruje, że przeoczono istotny element związany z zastosowaniem bardzo wydajnych metod przetwarzania danych w systemach zarządzania danymi.

2. Sprzętowe przetwarzanie strumieni danych

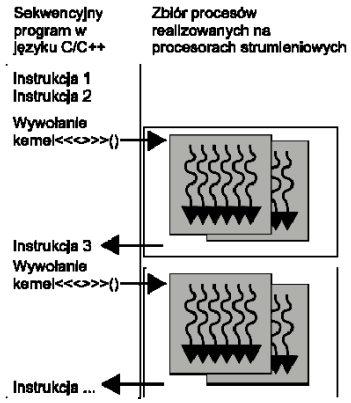
Podstawowym problemem związanym z doborem architektury sprzętowego, równoległego przetwarzania danych jest nadmiar dostępnych rozwiązań. Jednym z najbardziej niezależnych od architektury sprzętowej rozwiązań jest rozszerzenie RapidMind [4], wspierające wieloprocesorowe architektury producentów takich, jak: Nvidia, ATI, IBM, Intel oraz AMD. Zgodnie z opublikowaną w literaturze [1] opinią, biorąc pod uwagę brak konieczności tworzenia dedykowanego kodu, rozwiązanie RapidMind wydaje się być najbardziej elastyczne. Jednak z drugiej strony, w wyniku braku niezależnych testów wydajności, nie jesteśmy w stanie ocenić, czy to rozwiązanie jest rzeczywiście najlepsze. Dodatkowym problemem związanym z tym rozwiązaniem jest szereg barier natury licencyjnej oraz kosztów związanych z dostępem do tego narzędzia. Oczywiste jest, że poszukiwane jest rozwiązanie oparte na otwartym i ogólnodostępnym kodzie, nie wymagające wnoszenia dodatkowych opłat i możliwe do zbadania na sprzęcie dostępnym dla przeciętnego użytkownika.

Rezygnując z warunku dotyczącego wprowadzenia zmian w istniejącym kodzie, możemy wskazać rozwiązanie spełniające powyższe wymagania. W 2006 roku firma Nvidia zaprezentowała pierwszą, testową wersję architektury obliczeń numerycznych CUDA (*Compute Unified Device Architecture*). Od tego czasu bardzo intensywnie rozwijają się badania [4, 5] nad zastosowaniem dostarczonych rozwiązań w ramach istniejących systemów przetwarzania danych [5].

Kilka lat temu odkryto fakt, że zastosowane w kartach graficznych specjalizowane procesory (GPU) mogą przetwarzać nie tylko dane związane z grafiką, ale również dane ogólnego przeznaczenia. Dla tego sposobu prowadzenia obliczeń przedstawiono określenie GPGPU (*General-purpose computing on graphics processing units*). Stosowany jednak do tej pory model był bardzo nieporęczny i wymagający od programistów bardzo silnego uwarunkowania tworzonego kodu od zastosowanego rozwiązania sprzętowego, tj. wybranej karty graficznej.

Średniej klasy karta graficzna np. GeForce GTS 250 udostępnia obecnie ponad sto procesorów przeznaczonych do strumieniowego przetwarzania danych. Wspomniane zasoby zostały w stosunkowo prosty sposób udostępnione dla programisty. Zapewnia to opublikowana i dobrze udokumentowana przez producenta karty graficznej biblioteka API CUDA. Wsparcie zostało zapewnione poprzez dostarczanie narzędzi umożliwiających kompilację kodu

w oparciu o istniejące, darmowe i komercyjne kompilatory. Osiągnięte i publikowane wyniki badań wskazują na kilkudziesięciokrotne lub kilkusetkrotne przyspieszenie realizowanych obliczeń [6].



Rys. 1. Model przepływu sterowania i przetwarzania danych dla CUDA
Fig. 1. Data flow and control for CUDA

W pierwszym projekcie rozszerzenia istniejącego systemu zarządzania danymi podstawowe przetwarzanie danych zrealizowano w ramach sekwencyjnego programu korzystającego z zasobów jednostki centralnej. Wybrane, bardzo czasochłonne operacje powierzono realizacji procesorom dostępnym w ramach zasobów GPU. Na rysunku 1 przedstawiono sposób przepływu sterowania pomiędzy tymi systemami. Należy podkreślić, że w opisywanej architekturze równoległej jednostka centralna oraz zbiór procesorów GPU dysponują własnymi zasobami pamięci operacyjnej. CPU dysponuje pamięcią operacyjną, natomiast GPU dysponuje pamięcią dostępną na karcie graficznej. Dlatego też wywołania specjalizowanej funkcji, np. kernel<<<>>>() są poprzedzone wymianą danych pomiędzy wymienionymi zasobami pamięci.

Istotną cechą aplikacji wykorzystujących platformę CUDA w ten sposób jest fakt, że programista nie musi tworzyć i nadzorować kodu wielowątkowego. Automatyczne zarządzanie tworzonymi procesami jest niezaprzeczalną zaletą, szczególnie w momencie, kiedy dostępne zasoby umożliwiają pojawienie się równocześnie kilkunastu tysięcy procesów.

3. Przetwarzanie danych w strumieniowym systemie zarządzania danymi

Projektowany, strumieniowy system zarządzania danymi od samego początku był rozważany jako system o wysokiej wydajności. Opracowana algebra [8], deklaratywny język zapytań oraz wewnętrzne procedury przetwarzania i składowania danych były tworzone z myślą przyszłego zrównoleglenia prowadzonych operacji. Dlatego architektura, która z dużym prawdopodobieństwem uzyska przewagę, a następnie zdominuje technologię równoległego

przetwarzania danych, zasługuje na szczególną uwagę w aspekcie zastosowania jej w projektowanym systemie zarządzania danymi.

Projektowany, strumieniowy system zarządzania danymi jest tworzony w języku C/C++ z wykorzystaniem bibliotek BOOST. Biblioteka API CUDA udostępnia mechanizmy równoległego przetwarzania danych dla programów napisanych w języku C/C++. Dlatego technologiczne połączenie obu rozwiązań stanowi stosunkowo prosty i nieskomplikowany proces. Należy podkreślić, że pewne problemy technologiczne można napotkać korzystając z rozwiązań darmowych w 64-bitowym środowisku systemu Windows. Jednak ich pokonanie nie powinno stanowić większego problemu dla zespołu programistów obeznanych ze specyfiką środowiska języka C/C++.

3.1. Przykład realizacji zapytania

W ramach opisywanego systemu zarządzania danymi realizowane są ciągłe zapytania, stanowiące cechę charakterystyczną systemów strumieniowych [2, 3], oparte na planie realizacji zapytań. Plan ten powstaje w wyniku procesu tłumaczenia tekstu wejściowego na ciąg wewnętrznych poleceń systemu zarządzania danymi. Do tego celu użyto dostępnego w ramach biblioteki BOOST analizatora składni SPIRIT. Stworzony ciąg poleceń realizowany jest w systemie zarządzania danymi i to w ramach tego komponentu należy przeprowadzić połączenie wspomnianych sposobów przetwarzania danych.

Deklarując wstępnie istnienie następujących strumieni danych:

```
DECLARE wartosc1 STREAM core,1
DECLARE wartosc2 STREAM core1,0.5
```

oraz biorąc za przykład następujące zapytanie:

```
SELECT ( core[0]+1 ) * core1[0] as Pole1
STREAM StrumienWynikowy
FROM core + core1
```

opracowany analizator składni języka zapytań stworzy dla systemu zarządzania danymi następujący plan realizacji zapytania:

```
StrumienWynikowy:
1: ARGUMENT core
2: ARGUMENT core1
3: Operacja połączenia strumieni operatorem +

StrumienWynikowy.Pole1:
1: PUSH core[0]
2: PUSH 1
3: Operacja arytmetyczna na stosie +
4: PUSH core1[0]
5: Operacja arytmetyczna na stosie *
```

Przedstawione dwa ciągi poleceń stanowią jeden, spójny i nierozłączny plan realizacji zapytania. Pierwszy ciąg poleceń o etykiecie StrumienWynikowy stanowi plan budowy strumienia danych w oparciu o dane napływające z istniejących strumieni. Na jego podstawie system zarządzania danymi jest poinformowany, które krotki i w jakiej kolejności należy połączyć w celu stworzenia kolejnego strumienia. W realizacji tego celu znajduje zastosowanie opracowana [8] algebra strumieni danych.

Drugi plan, oznaczony etykietą StrumienWynikowy.Pole1, jest stosowany w trakcie tworzenia zawartości krotki. Wartości pól wewnątrz kolejnych krotek strumienia danych wyliczane są za pomocą klasycznej algebry i klasycznego algorytmu ze stosem. W oparciu o tak przygotowane dane wejściowe strumieniowy system zarządzania danymi przetwarza napływające dane.

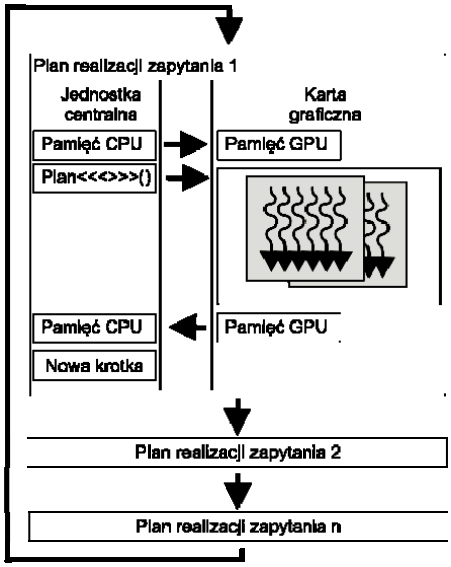
Należy zwrócić uwagę, że oba, stworzone plany realizacji zapytań nie zawierają instrukcji skoku. Ta właściwość charakteryzuje wszystkie realizowane w systemie plany zapytań. Iteracja realizowana jest poprzez algorytm, który zamyka plany w jednej, martwej pętli. Kolejnością wykonania planów zarządza proces odpowiedzialny za dobór kolejności realizacji.

3.2. GPGPU w ramach realizacji planu zapytania

Implementując GPGPU przy zastosowaniu CUDA API pracę rozpoczęto od przygotowania zbioru operatorów zdolnych do przetwarzania danych pochodzących ze strumieni danych na karcie graficznej. Główną pętlę sterującą pozostawiono pod kontrolą jednostki centralnej. Kolejne instrukcje planu realizacji zapytania były realizowane sekwencyjne. Dopiero instrukcje wymagające przeliczenia większego, zgromadzonego w strumieniu danych zbioru poprzedzano przesyłem pamięci z zasobów CPU do zasobów GPU. Kolejnym krokiem było uruchomienie zbioru procesów przetwarzających dane równolegle wewnątrz GPU.

Opisaną sekwencję przetwarzania danych przedstawiono na rysunku 2. Cykliczne wywoływanie planów realizacji zapytań dla uproszczenia przedstawiono w zamkniętej pętli. Co prawda taki algorytm jest możliwy do zastosowania i przez pewien czas był stosowany na etapie prototypowym, jednak w takcie dalszych prac przedstawiono bardziej zaawansowane metody rozdziału zasobów i czasu realizacji planów w trakcie przetwarzania strumieni przez DSMS.

Przestawiona konstrukcja systemu zarządzania danymi dała stosunkowo niewielki wzrost wydajności w przypadku zapytań realizowanych na podstawie kilku lub kilkunastu wierszy strumienia danych. Pewien znaczący wzrost wydajności osiągnięto dopiero przy zapytaniach, których plany realizacji wymagały przetworzenia kilkunastu lub kilkuset wierszy z różnych strumieni danych w ramach jednego zapytania. W trakcie prac pojawiły się również pewne problemy związane z synchronizacją procesów realizowanych wewnątrz GPU a głównym procesem sterującym znajdującym się pod kontrolą CPU.



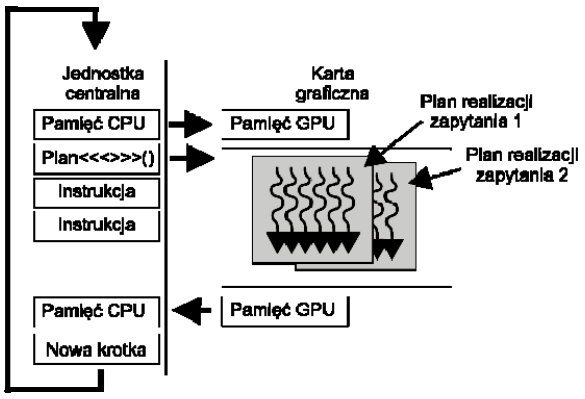
Rys. 2. Podstawowa architektura GPGPU wewnątrz DSMS
Fig. 2. Base GPGPU architecture inside DSMS

3.3. Modyfikacja architektury równoległego przetwarzania

Z uwagi na przedstawione ograniczenia rozpoczęto prace nad realizacją zmodyfikowanej architektury równoległego przetwarzania danych. Biblioteka CUDA udostępnia zbiór funkcji umożliwiających bezpośrednie przetwarzanie strumieni danych. W dokumentacji określono tę technikę jako asynchroniczne współbieżne wykonanie procesów i udostępniono szereg funkcji wspierających tego typu model przetwarzania danych.

```
cudaError_t cudaStreamCreate ( cudaStream_t * pStream );
cudaError_t cudaStreamDestroy ( cudaStream_t stream );
cudaError_t cudaStreamQuery ( cudaStream_t stream );
cudaError_t cudaStreamSynchronize ( cudaStream_t stream );
```

Różnica pomiędzy poprzednim rozwiązaniem a projektowaną modyfikacją umożliwi przeprowadzenie przetwarzania planów realizacji zapytań bezpośrednio wewnątrz zasobów karty graficznej.



Rys. 3. Projekt architektury DSMS realizującej zapytania wewnątrz GPU
Fig. 3. DSMS architecture that process queries inside GPU

Na rysunku 3 przedstawiono projekt architektury umożliwiającej przetwarzanie większości napływających danych wewnątrz GPU. Program sterujący, znajdujący się pod kontrolą jednostki centralnej, nadal będzie realizować podstawowe zadania związane z rozdziałem zasobów oraz kolejnością realizacji poszczególnych zadań. Symbolicznie przedstawiono ten proces poprzez zapętlenie kolejnych instrukcji realizowanych w ramach zasobów jednostki centralnej.

4. Podsumowanie

Projektowanie i budowanie strumieniowych systemów zarządzania danymi stanowi bardzo czasochłonny i wymagający dużej wiedzy oraz badań proces. Do dnia dzisiejszego zaledwie kilka firm zaproponowało komercyjne rozwiązania stosujące strumieniowe metody zarządzania danymi. Należy podkreślić, że toczy się również dyskusja na forum naukowym dotycząca podstaw teoretycznych oraz polemika nad wzrostem potencjalnej wydajności przetwarzania danych [9, 10]. Faktycznie, typowy dotychczasowy udokumentowany wzrost szybkości przetwarzania danych w systemie strumieniowym w stosunku do systemu relacyjnego był zaledwie kilkukrotny. Jednak projektowane, sprzętowe rozszerzenie, możliwe do efektywnego zastosowania jedynie w systemach strumieniowych, powinno przynieść kilkudziesięciokrotny lub kilkusetkrotny [7] wzrost wydajności w stosunku do systemów relacyjnych. Dalsza polemika z tego typu wynikami badań powinna być bardzo trudna.

Należy też zwrócić uwagę, że na rynku zaczynają się pojawiać urządzenia wspierające technologię CUDA, pozbawione funkcji wyświetlania obrazu. Tak wiec, dysponując odpowiednimi środkami, możemy na dzień dzisiejszy wyposażyć komputer osobisty w urządzenie, którego możliwości obliczeniowe przekraczają istniejące dotychczas rozwiązania. Dysponując wymaganym sprzętem oraz odpowiednio przygotowanym systemem zarządzania danymi, a w szczególności strumieniowym systemem zarządzania danymi, którego podstawowym zadaniem jest przetwarzanie sygnałów, będziemy w stanie realizować obliczenia w sposób deklaratywny. Podsumowując, można stwierdzić, że zaproponowane rozwiązanie stanowi potencjalne znaczące rozszerzenie istniejącego zbioru metod przetwarzania danych w oparciu o sprzętowe wspomaganie przetwarzania danych.

BIBLIOGRAFIA

1. Halfhill T. R.: Parallel processing with CUDA. Microprocessor Report, 2008.
2. Golab L., Özsu M. T.: Issues in Data Stream Management. SIGMOD Record, 2003, s. 5÷14.
3. Mazur Z., Pawluk P.: Strumieniowe bazy danych. Bazy danych. Struktury, Algorytmy, Metody, WKiŁ, 2006, s. 77÷88.
4. McCool M. D., D'Amora B.: Programming using RapidMind on the Cell BE. 2006, s. 222.
5. Dössel O., Schlegel W.C.: The Future of Volume Graphics in Medical Virtual Reality. IFMBE Proceedings, 2009, s. 1349÷1352.
6. Bordoloi U.D., Chakraborty S.: GPU-based Acceleration of System-level Design Tasks. International Journal of Parallel Programming, 2010 (w druku).
7. Ford E.B.: Parallel Algorithm for Solving Kepler's Equation on Graphics Processing Units: Application to Analysis of Doppler Exoplanet Searches 2009. New Astronomy, Vol. 14(4), 2009, s. 406÷412.
8. Widera M.: Deterministic method of data sequence processing. Annales UMCS Sectio AI Informatica, Vol. IV, 2006, s. 314÷331.
9. Stonebraker M., Cetintemel U.: One Size Fits All: An Idea Whose Time Has Come and Gone. CIDR, 2007, s. 2÷11.
10. Hoppe A., Gryz J.: Stream Processing in a Relational Database: a Case Study. IDEAS, 2007, s. 216÷224.

Recenzent: Dr hab. Zygmunt Mazur, prof. Pol. Wrocławskiej

Wpłynęło do Redakcji 31 stycznia 2010 r.

Abstract

There have been a lot of papers presenting various Data Stream Management System (DSMS) and Compute Unified Device Architecture (CUDA) applications recently. Unfortunately, there are no papers that present both issues simultaneously. This paper presents the project on DSMS, which supports the parallel hardware data processing module.

The paper starts with the analysis of the existing parallel support solutions, presenting several popular commercial solutions to the problem. In accordance with the requirements,

we have chosen CUDA as the basis for our further research. The paper presents a simple case of the parallel execution of the given query plan.

This is an analysis of the first approach to the problem of the extending existing architecture. We also present the conclusions we reached when trying to find the solution to this problem. Summing up, it is possible to say that this direction of research is very promising.

Adres

Michał WIDERA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, michal@widera.com.pl .