STUDIA INFORMATICA 2012
Volume 33 Number 2B (106)

Marcin APTEKORZ, Kamil SZOSTEK, Mariusz MŁYNARCZUK
Akademia Górniczo-Hutnicza, Wydział Geologii, Geofizyki i Ochrony Środowiska,
Katedra Geoinformatyki i Informatyki Stosowanej

MOŻLIWOŚCI AKCELERACJI PRZESTRZENNYCH BAZ DANYCH NA PODSTAWIE PROCESORÓW KART GRAFICZNYCH ORAZ FUNKCJI UŻYTKOWNIKA[1]

Streszczenie. W artykule wykazano, iż możliwe jest przyspieszenie procesów analizy danych przestrzennych, w tym wypadku danych katalogu astrometrycznego gwiazd, bez konieczności modyfikacji kodu źródłowego bazy danych. Wykorzystano do tego celu zewnętrzne funkcje użytkownika UDF oraz technologię CUDA firmy NVIDIA, która pozwala na skuteczną akcelerację obliczeń numerycznych na podstawie procesorów kart graficznych.

Słowa kluczowe: przestrzenna baza danych, GPU, UDF, PostgreSQL, MySQL

SPATIAL DATABASE ACCELERATION USING GRAPHIC CARD PROCESSORS AND USER-DEFINED FUNCTIONS

Summary. This paper proves that it is possible to accelerate spatial data analysis, for example astronomic data, without modifying source code of the database engine. User-defined function (UDF) were used in cooperation with NVIDIA CUDA to increase efficiency and speed of numerical operations.

Keywords: spatial database, GPU, UDF, PostgreSQL, MySQL

1. Wprowadzenie

Wykorzystanie mocy obliczeniowej kart graficznych od dłuższego czasu odnajduje zastosowanie nie tylko w oprogramowaniu rozrywkowym, ale także w profesjonalnych systemach obliczeniowych. Między innymi dzięki technologii NVIDIA CUDA wielordzeniowe procesory graficzne (GPU) pozwalają na znacznie szybsze i tańsze przetwarzanie oraz analizę da-

[1] Praca finansowana w ramach badań statutowych Katedry Geoinformatyki i Informatyki Stosowanej.

nych numerycznych niż profesjonalne i relatywnie drogie procesory [1, 2]. Badania użycia procesorów graficznych, między innymi w eksploracji danych (ang. *data mining*), wykazały, że zmiana kodu źródłowego bazy danych do współpracy z GPU pozwala na osiągnięcie przyspieszeń nawet 20-, 70-krotnych [3, 4].

Celem niniejszej pracy jest zbadanie możliwości implementacji oraz wydajności zewnętrznych funkcji użytkownika (ang. *User-Defined Function* – UDF), korzystających z zasobów obliczeniowych kart graficznych, do rozszerzenia możliwości istniejącej bazy danych. Zastosowanie UDF pozwala na dodanie funkcjonalności GPU bez zmian i kompilacji kodu źródłowego bazy danych (DBMS). Do testów wykorzystano dane z przestrzennej bazy danych katalogu astrometrycznego gwiazd z misji ESA „Hipparcous" [5]. Dane te umieszczone zostały w bazach danych PosgreSQL oraz MySQL, które posiadają rozbudowane rozszerzenia do przestrzennej analizy danych, odpowiednio PostGIS oraz MySQL Spatial [6].

2. Zewnętrzne funkcje użytkownika oraz technologia CUDA

2.1. Zewnętrzne funkcje użytkownika (UDF)

Współczesne DBMS oferują swoim użytkownikom możliwość implementacji ich własnych funkcji napisanych w języku C/C++ przez załadowanie ich ze skompilowanej biblioteki DLL lub SO [7]. Sposoby oprogramowywania funkcji UDF znacznie się różnią między różnymi środowiskami DBMS. W przypadku PostgreSQL zewnętrzna funkcja SQL jest reprezentowana przez jedną funkcję w bibliotece, natomiast w przypadku MySQL można stworzyć trzy funkcje: funkcję inicjalizującą, funkcję obliczeniową (właściwą) oraz funkcję deinicjalizującą [8].

Funkcje użytkownika importowane są za pomocą procedury tworzenia funkcji:

```
CREATE OR REPLACE FUNCTION gpu_dist(point, point)
RETURNS real
AS 'PATH/gpupg.so','gpu_dist'
LANGUAGE 'C' VOLATILE STRICT;
```

a następnie mogą być uruchomione z poziomu zapytania SQL, np.:

```
SELECT gpu_distance(POINT(1,1), coord) FROM hipparcous
```

Podczas wykonania takiego zapytania dane przestrzenne z każdego wiersza stają się argumentem funkcji UDF. W przypadku DBMS PostgreSQL typ POINT reprezentowany jest w języku C jako *POINT. Inne typy geometryczne zdefiniowane są w pliku nagłówkowym *utils/geo_decls.h*, wchodzącym w skład pakietu *postgresql9.1-dev* [7].

2.2. Technologia NVIDIA CUDA

Tworzenie funkcjonalności opartej na procesorach kart graficznych firmy NVIDIA bazuje na technologii CUDA [9]. Technologia ta pozwala na zrównoleglenie wybranych operacji numerycznych dzięki implementacji algorytmów uruchamianych na wielordzeniowych układach kart graficznych. Podstawowym, ale nie jedynym, problemem przy takim rozwiązaniu jest wolny transfer danych do pamięci karty graficznej, który często w kluczowy sposób wpływa na wydajność obliczeń.

NVIDIA dostarcza niezbędnych narzędzi do tworzenia takich aplikacji, w tym kompilator *nvcc*. Pomimo wielu opcji tego kompilatora, implementacja i kompilacja biblioteki UDF generują dużą liczbę błędów. Dlatego należy rozdzielić kod na dwie części, które są kompilowane osobno, odpowiednio: *nvcc* dla bibliotek obsługujących CUDA oraz *gcc/g++* dla bibliotek obsługujących UDF.

3. Implementacja

3.1. Środowisko pomiarowe i dane testowe

Do testów zamierzano użyć dwóch wolnodostępnych DBMS – MySQL w wersji 5.1 i PostgreSQL w wersji 9.1. Zostały one skompilowane dla systemu Canonical Ubuntu Linux 11.10 (Wersja jądra: 3.0.0-12-generic, 64 bit) z uruchomioną funkcjonalnością obsługi UDF.

Testy odbywały się na 16-rdzeniowej karcie graficznej GeForce 315M. Kartę zainstalowano w notebooku Toshiba L750-12W z 2-rdzeniowym procesorem Intel Pentium P6200.

W procesach kompilacji użyto kompilatorów:

- *nvcc* w wersji 4.0 (V0.2.1221) do kompilacji kodu wykonywanego na GPU,
- *gcc* (wersja 4.6.1) do kompilacji biblioteki UDF dla PostgreSQL,
- *g++* (4.6.1) do kompilacji biblioteki UDF dla MySQL.

Pomimo poprawnej kompilacji, narzędzia ochrony systemu (np. *Apparmor*) nie pozwalają DBMS na uruchomienie potoków oraz importowanie funkcji z poziomu zewnętrznych bibliotek UDF. Rozwiązanie tego problemu w sposób zapewniający skuteczną ochronę stanowić będzie element przyszłych badań.

Obliczenia przeprowadzono na przestrzennej bazie danych katalogu astrometrycznego gwiazd z misji ESA „Hipparcous” [5]. Dane przestrzenne z katalogu w formacie ASCII przeliczono na stopnie, a następnie umieszczono w przestrzennej bazie danych, konwertując miary rektascensji i deklinacji na typ geometryczny POINT.

3.2. Metodyka badania

Pomiarowi wydajności zostało poddane wyznaczanie odległości między punktem o współrzędnych (1, 1) a daną przestrzenną *POINT*, według wzoru (1):

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \tag{1}$$

gdzie x_1, y_1 opisują punkt z przestrzennej bazy danych, a x_2, y_2 opisują punkt o współrzędnych (1,1). Wszystkie operacje zostały zawarte wewnątrz funkcji zliczającej *COUNT,* aby wyeliminować czas potrzebny na wyświetlenie wyników.

Jako punkt odniesienia zbadano czasy obliczania odległości za pomocą poniższego zapytania SQL:

```
SELECT SQRT( POW(1 - coord[0], 2) + POW( 1 - coord[1], 2)) FROM Hipparcous
```

oraz bezpośrednio w samej funkcji zewnętrznej UDF. Dodatkowo zbadano szybkość wykonywania funkcji *ST_Distance*, pochodzącej z rozszerzenia PostGIS.

W początkowym etapie testów sprawdzono możliwość implementacji oraz szybkość wywołań zewnętrznych programów z funkcji UDF, przy użyciu dwóch rozwiązań: importowania zewnętrznej funkcji przy użyciu *dlopen* oraz wykorzystaniu potoków funkcją *popen*. Rozwiązania te okazały się możliwe w implementacji w przypadku bazy PostreSQL. Znacząca przewaga w szybkości działania (ok. 3 razy szybciej) rozwiązania *dlopen* zadecydowała o tym, iż w dalszej części wykorzystano tylko tę metodę.

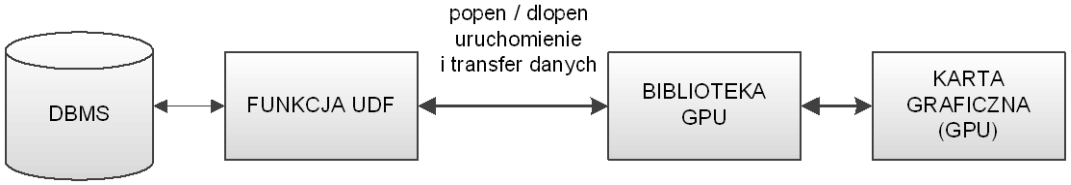
Implementacja algorytmu uruchamianego na karcie graficznej wykonana została na dwa sposoby. Pierwszy z nich (*dlopen CUDA*) polega na przesłaniu z bazy danych o pojedynczym punkcie – jednej krotce – do funkcji UDF, gdzie następnie użyta zostaje funkcja z biblioteki zewnętrznej. W niej dopiero uruchamiane jest środowisko CUDA, a przeliczona odległość jest zwracana do funkcji UDF i stąd do przestrzennej bazy danych, jako wynik operacji na pojedynczej krotce (rys. 1). Potencjał układu GPU pozostaje w tym wypadku niestety niewykorzystany.

Drugi sposób (CUDA RUN) wykorzystuje UDF jako aktywator zewnętrznego programu opartego na GPU. W ten sposób zapytanie SQL wywołuje funkcję UDF z argumentami precyzującymi zapytanie, które zostanie wykonane przez program lub bibliotekę obsługującą GPU:

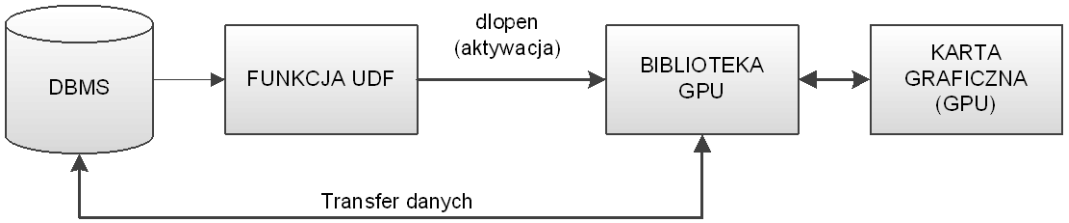
```
SELECT gpu_distance('SELECT coord[0], coord[1] FROM hipparcous',
       'nazwa_tabeli_tymczasowej_do_której_trafia_wyniki');
```

Program pobiera dane z bazy, realizując zapytanie, które zostało przekazane z UDF, a następnie wykorzystuje technologię CUDA do przeliczenia odległości dla wszystkich pobranych z bazy krotek. Zwracanie wyników do bazy danych w tym wypadku może odbywać się na przykład przez polecenie *INSERT* do zdefiniowanej tabeli tymczasowej (rys. 2). Jest to

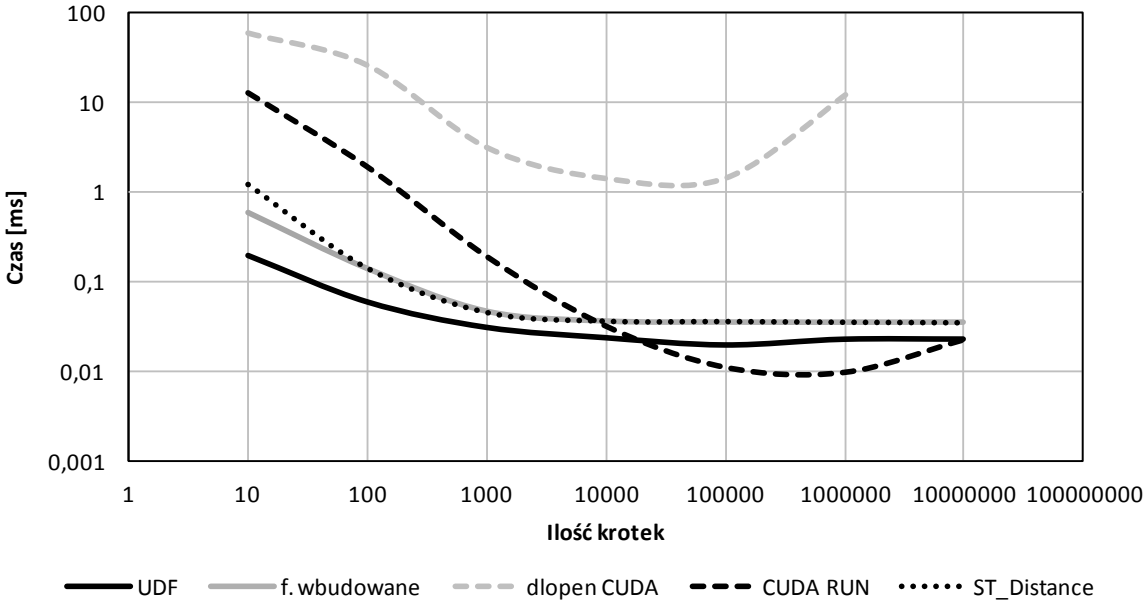
niestety nieoptymalne rozwiązanie, jednak może zostać użyte dla małej liczby zawracanych krotek.



Rys. 1. Schemat zastosowania funkcji UDF w wariancie *dlopen* i *popen*
Fig. 1. Scheme of UDF function usage with *dlopen* and *popen*



Rys. 2. Schemat zastosowania funkcji UDF w wariantach CUDA RUN
Fig. 2. Scheme of UDF function usage with CUDA RUN option



Rys. 3. Średni czas wykonania obliczeń dla jednej krotki, dla różnej liczby danych
Fig. 3. Mean execution time of one record calculation for various length of input data

Implementacja algorytmu uruchamianego na GPU dla tego sposobu nie jest skomplikowana i opiera się na podstawowym modelu działania CUDA: współrzędne punktów są kopiowane do pamięci karty graficznej, a następnie uruchamiany jest *kernel* – proces obliczeniowy na karcie graficznej. Na koniec wyniki kopiowane są do pamięci hosta. W czasie testów nie wykonywano wstawiania wyników do bazy.

Kompilacja obydwu rozwiązań odbywa się dwuetapowo, przez rozdzielenie kodu wykonywanego na GPU i UDF. W przypadku CUDA RUN kod zawiera funkcje połączeniowe do

bazy danych, zatem zachodzi konieczność przekazania kompilatorowi odpowiedniej biblioteki interfejsu *libpq-fe*. W tym wypadku należy uruchomić GCC z argumentem *-lpq*. Kompilacja biblioteki UDF odbywa się standardowo, jak w przypadku bibliotek dzielonych, lecz z koniecznością wskazania katalogu zawierającego pliki nagłówkowe i źródła funkcji dostarczanych przez pakiet narzędzi programistycznych DBMS – *postgres-server-dev*.

4. Wyniki pomiarów i wnioski

Wyniki pomiarów przedstawione na rys. 3 obrazują rozkład prędkości wykonania obliczeń na jednej krotce, przy operacjach na różnej liczbie danych. Dla uzyskania średniego czasu działania na jednej krotce, czas obliczenia wszystkich krotek jest dzielony przez ich liczbę.

Wariant wykorzystujący jedynie funkcje wbudowane języka PostgreSQL, funkcja z biblioteki PostGIS oraz nieco szybsze wykonanie obliczeń w funkcjach UDF ustępują szybkości implementacji CUDA RUN dopiero przy około 100 tysiącach krotek. Jednakże, mimo znaczącej poprawy szybkości samych obliczeń, należy pamiętać, że rezultaty tych operacji nie mogą być zwrócone do bazy danych inaczej niż przez polecenie *INSERT*. Przy tak dużej liczbie danych operacja ta zajmuje ponad 5 minut.

Rozwiązanie *dlopen CUDA,* zgodnie z przewidywaniami, jest dużo wolniejsze od pozostałych metod, głównie z powodu nieoptymalnego wykorzystania architektury GPU. Mimo że dane wynikowe są zwracane do DBMS po operacji na każdej krotce, rozwiązanie może mieć zastosowanie w przypadku skomplikowanych obliczeń oraz gdy wymagane są dodatkowe zasoby w postaci GPU.

Zauważalny spadek wydajności powyżej miliona krotek jest spowodowany ograniczoną wielkością pamięci RAM oraz pamięci karty graficznej.

5. Podsumowanie i przyszłe kierunki badań

Potencjał zastosowania wsparcia kart graficznych do analizy danych przestrzennych jest niekwestionowany, jednakże z powodu konstrukcji GPU i wynikających z tego ograniczeń ważne jest, aby algorytmy optymalnie wykorzystywały możliwości układów graficznych.

W pracy przedstawiono możliwości akceleracji obliczeń w przestrzennych bazach danych, przy wykorzystaniu technologii NVIDIA CUDA oraz mechanizmu zewnętrznych funkcji użytkownika UDF. Dzięki takiemu rozwiązaniu nie są konieczne zmiana kodu i rekompilacja DBMS. Przeprowadzone badania potwierdziły możliwość implementacji algo-

rytmów uruchamianych na GPU, jednak okazało się, że przyspieszenia nie są tak wysokie, jak w przypadku zmiany kodu DBMS [3]. Mimo to, wykorzystanie UDF i technologii CUDA do rozszerzenia możliwości przestrzennej bazy danych może przynosić korzyści w przypadku intensywnej, równoległej eksploatacji danych przez wielu użytkowników oraz w kosztownych obliczeniach, których wyniki nie muszą być zwracane wprost do bazy danych.

W przyszłych badaniach poruszony zostanie problem implementacji innych, bardziej złożonych, zapytań i procedur SQL. Ponadto, badania przeprowadzone zostaną na komputerach o wysokiej mocy obliczeniowej IBM HS21 Blade, wyposażonych w karty graficzne NVIDIA Quadro FX 1600M. Możliwości implementacji zewnętrznych funkcji użytkownika dla tego DBMS pozwalają sądzić, że akceleracja GPU może być skutecznie wykorzystana, zwłaszcza Agregujących Funkcji Użytkownika (ang. *User-Defined Aggregates* – UDA).

BIBLIOGRAFIA

1. Govindaraju N., Gray J., Kumar R., Manocha D.: GPUTeraSort: high performance graphics co-processor sorting for large database management. Proceedings of the 2006 ACM SIGMOD international conference on Management of data (SIGMOD ‘06). ACM, New York, USA 2006, s. 325÷336.
2. Pawłowski R., Mrozek D.: Przyspieszenie algorytmu Smitha-Watermana z użyciem procesora graficznego. Studia Informatica, Vol. 32, No. 2A(96), Wydawnictwo Politechniki Śląskiej, Gliwice 2011, s. 181÷198.
3. Bakkum P., Skadron K.: Accelerating SQL database operations on a GPU with CUDA. Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU ‘10). ACM, New York, USA 2010, s. 94÷103.
4. Govindaraju N. K., Lloyd B., Wang W., Lin M., Manocha D.: Fast Computation of Database Operations using Graphics Processors. Proceedings of SIGGRAPH ‘05 ACM SIGGRAPH 2005 Courses. ACM, Article No. 206, New York 2005.
5. van Leeuwen F.: Hipparcos, the New Reduction of the Raw Data. Astron. Astrophys. 439, (2005), 2007.
6. Piórkowski A.: MySQL Spatial and PostGIS – implementations of spatial data standards. EJPAU, T. 14(1), #03, 2011.
7. PostgreSQL Global Development Group, PostgreSQL 9.1.2 Documentation, http://www.postgresql.org/docs/9.1/static/ (dostęp: 31.12.2011).
8. Oracle Corporation, MySQL 5.0 Reference Manual, http://dev.mysql.com/doc/refman/5.0/en/ (dostęp: 31.12.2011).
9. CUDA Zone, NVIDIA Corporation, http://www.nvidia.pl/object/cuda_home_new_pl.html, (dostęp 31.12.2011).

Wpłynęło do Redakcji 15 stycznia 2012 r.

Abstract

This article presents an innovative approach to the acceleration of spatial database analysis by taking advantage of graphic card processors (GPU) and user-defined functions (UDF). This idea does not require to change or recompile original DBMS engine. Instead, user can write his own function (UDF) in C/C++ and import it with a simple DBMS query. This functions can use NVIDIA CUDA to accelerate calculation by executing them in parallel on a high efficient GPU.

The idea was implemented and tested with astronomic data imported into PostgreSQL database in two methods: first involves *dlopen* to import and execute CUDA functions (Fig. 1.), second executes program that connects to database and performs required calculation on GPU (Fig. 2.).

As the results show, first approach appears to be slower, but might be useful when additional processing unit is required (Fig. 3). The second method is faster than conventional SQL query or direct UDF calculation, but the technique of returning results to DBMS appears to be its week point. It can be done only via *INSERT* method, which is inefficient. Despite that, this approach can be used to perform time consuming calculations that do not produce large amounts of data.

Adresy

Marcin APTEKORZ: AGH Akademia Górniczo-Hutnicza, Wydział Geologii, Geofizyki i Ochrony Środowiska, Katedra Geoinformatyki i Informatyki Stosowanej, al. A. Mickiewicza 30, 30-059 Kraków, Polska, aptekorz@gmail.com

Kamil SZOSTEK: AGH Akademia Górniczo-Hutnicza, Wydział Geologii, Geofizyki i Ochrony Środowiska, Katedra Geoinformatyki i Informatyki Stosowanej, al. A. Mickiewicza 30, 30-059 Kraków, Polska, szostek@agh.edu.pl

Mariusz MŁYNARCZUK: AGH Akademia Górniczo-Hutnicza, Wydział Geologii, Geofizyki i Ochrony Środowiska, Katedra Geoinformatyki i Informatyki Stosowanej, al. A. Mickiewicza 30, 30-059 Kraków, Polska, mlynar@agh.edu.pl