STUDIA INFORMATICA 2011
Volume 32 Number 2A (96)

Marcin GORAWSKI
Politechnika Śląska, Instytut Informatyki
Politechnika Wrocławska, Instytut Informatyki
Damian LIS
Politechnika Śląska, Instytut Informatyki

ARCHITEKTURA CUDA W BEZOPÓŹNIENIOWYCH HURTOWNIACH DANYCH

Streszczenie. Rośnie znaczenie i potrzeba zapewnienia aktualności danych oraz efektywności ich przetwarzania w bezopóźnieniowych hurtowniach danych. Standardowe podejście, oparte na tradycyjnym procesie ekstrakcji danych (ETL), okazało się niewystarczające ze względu na potrzebę podziału czasu dostępu, na czas odświeżania danych oraz czas ich analizy. W artykule przedstawiono system ETL dla bezopóźnieniowych hurtowniach danych. System ten realizuje algorytm WINE-HYBRIS, bazujący na architekturze CUDA oraz CPU. Przedstawiono testy wydajnościowe tego systemu opartego na dwóch całkowicie różnych architekturach, umożliwiając zobaczenie możliwości, jakie niesie za sobą wykorzystanie architektury CUDA w systemach hurtowni danych.

Słowa kluczowe: ETL, CUDA, WINE-HYBRIS, bezopóźnieniowe hurtownie danych

CUDA ARCHITECTURE IN ZERO-LATENCY DATA WAREHOUSE

Summary. There is a growing importance and the need to ensure data actualisation and efficiency of their processing in zero-latency data warehouses. The standard approach, based on the traditional process of data extraction (ETL) was not sufficient because of the need for time-sharing access during the process of refreshing the data and the time of analysis. The paper presents an ETL system for the zero-latency data warehouse. This system implements the WINE-HYBRIS algorithm based on CUDA and CPU architectures. Presented performance testing of the system, is based on two completely different architectures, allowing the possibility of observing upcoming opportunities that arises during usage of the CUDA architecture in data warehousing systems.

Keywords: ETL, CUDA, WINE-HYBRIS, Zero-latency data warehouse

1. Wprowadzenie

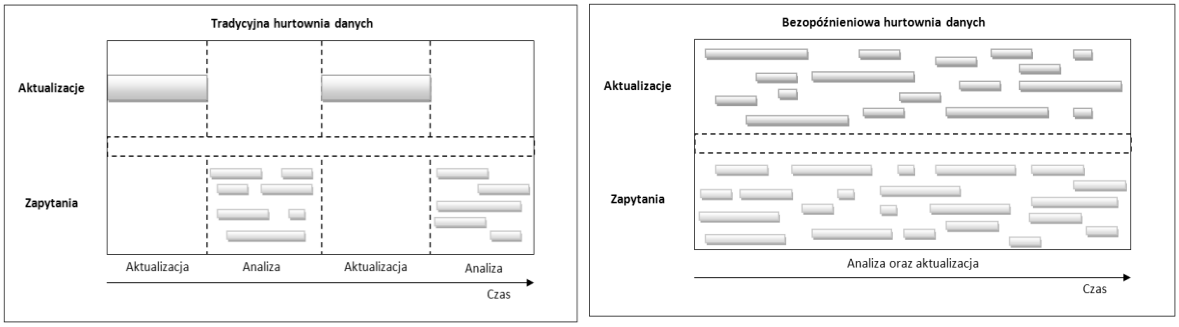
Podejście projektowania bezopóźnieniowych hurtowni danych [1, 6], oparte na tradycyjnym procesie ETL (ang. *Extract, Transform, Load*) [9, 10, 14] okazało się nie efektywne, ponieważ nie uwzględniało rozdziału czasu dostępu, na czas odświeżania danych oraz czas ich analizy. Ze względu na ciągły napływ danych, każda pojawiająca się modyfikacja danych źródłowych lub zmiana środowiska bezopóźnieniowych hurtowni danych musi być wprowadzona automatycznie i natychmiastowo. Stąd stawiany jest problem zarówno metod podziału czasu dostępu do hurtowni, jak i metod ładowania danych. Zagadnienia badawcze bezopóźnieniowych hurtowni danych zostały przedstawione w pracy [6].

W artykule zostanie zaprezentowany autorski system ekstrakcji danych, zorientowany na obsługę bezopóźnieniowych hurtowni danych bazujący na algorytmie WINE-HYBRIS, osadzony w dwóch różnych architekturach – CUDA oraz CPU. Rozbudowa tego systemu została oparta na już istniejących rozwiązaniach algorytmu WINE [19] i systemu LEMAT [15]. Nie zawsze efektywne jest ładownie wszystkich danych od razu, z takim samym priorytetem. Należy brać pod uwagę rzeczywiste zapotrzebowanie użytkowników hurtowni danych – w pierwszej kolejności ładować i przetwarzać te dane, które są potrzebne w danym momencie. Dzięki temu możliwe jest zwiększenie dostępności hurtowni danych. Natomiast użytkownicy otrzymają dane, których świeżość (aktualność) będzie utrzymywana na odpowiednim poziomie (użytkownik dostanie tak aktualne dane jakie potrzebuje).

Wykorzystanie architektury CUDA w systemie bezopóźnieniowych hurtowni danych ma na celu zwiększenie wydajności przetwarzanie zapytań i aktualizacji danych.

2. Proces ETL w bezopóźnieniowych hurtowniach danych

Standardowy proces ETL, na którym oparte są klasyczne hurtownie danych, składa się z ekstrakcji danych z bazowych źródeł, czyszczenia danych oraz ładowania ich do hurtowni danych [10]. Taki proces ETL jednak nie jest satysfakcjonujący dla użytkowników bezopóźnieniowych hurtowni danych (podczas czasu aktualizacji użytkownik nie ma dostępu do danych). Ponadto, w oknie ładowana danych inicjowane są kolejno wszystkie aktualizacje, bez względu na ich rzeczywiste zapotrzebowanie. Z kolei podczas wykonania analizy (zapytań) nie są wykonywane żadne aktualizacje danych, co skutkuje tym, że dane znajdujące się w hurtowni są coraz bardziej nieaktualne.

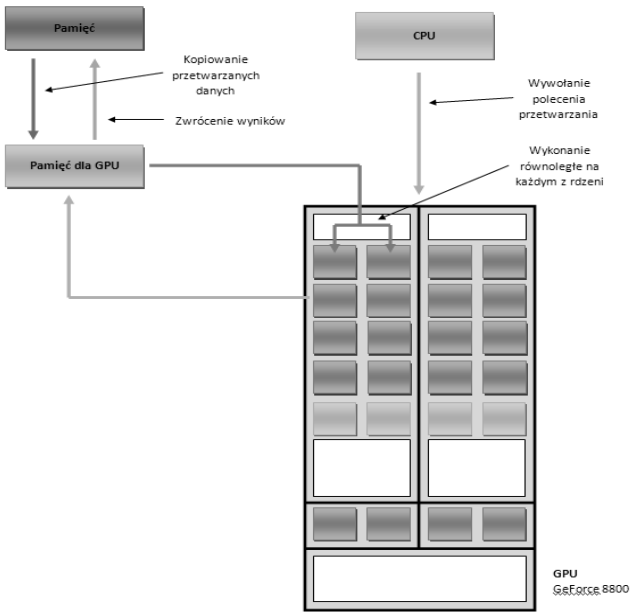


Rys. 1. Podział czasu dla aktualizacji i zapytań w hurtowniach danych
Fig. 1. Time partitioning for updates and queries in data warehouses

Dzięki wprowadzeniu rozbudowanego procesu ETL, możliwe jest rozwiązanie problemu, związanego z podziałem i zastosowanie wspólnego przedziału dla aktualizacji i zapytań. W następstwie, możliwe jest wytypowanie aktualizacji, które mają zostać wykonane przed zapytaniem użytkownika [15].

3. Architektura CUDA

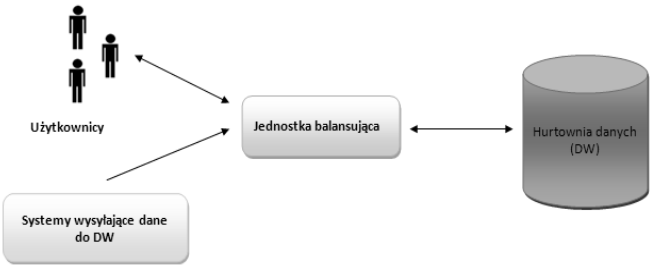
Architektura CUDA (ang. *Compute Unified Device Architecture*) [3, 4], opracowana przez firmę NVidia, umożliwia wykorzystanie mocy obliczeniowej procesorów wielordzeniowych (np. procesorów karty graficznej). Pozwala ona na zrównoleglenie obliczeń (np. inżynierskich), także na m.in. przyspieszenie szyfrowania i kompresji oraz tworzeniu efektów specjalnych w grafice komputerowej.



Rys. 2. Przetwarzanie danych na podstawie architektury CUDA [4]
Fig. 2. Data processing based on CUDA architecture [4]

4. Algorytm WINE-HYBRIS

Algorytm WINE-HYBRIS został stworzony z myślą o zarządzaniu w systemie bezopóźnieniowych hurtowniach danych procesem ETL. Zaimplementowany w jednostce balansującej obciążenie (ang. *Workload Balancing Unit*) [19], odpowiada za wykonanie wszystkich operacji związanych z przekształcaniem i analizą zapytań oraz aktualizacji, przesyłanych do hurtowni danych.



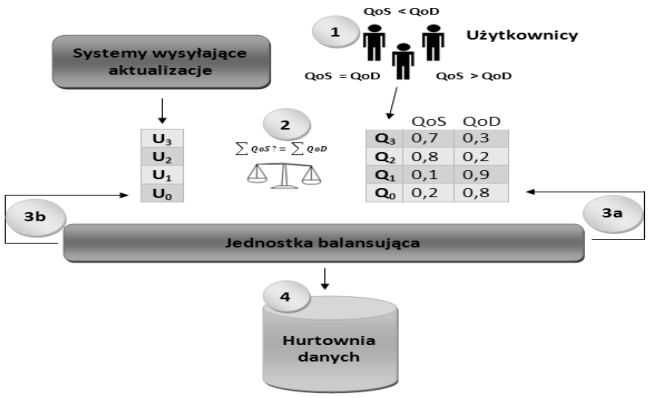
Rys. 3. Wykorzystanie jednostki balansującej w hurtowniach danych
Fig. 3. The use of WBU in data warehouses

Proponowany schemat algorytmu WINE-HYBRIS w znacznym stopniu bazuje na założeniach dwupoziomowego algorytmu planowania WINE (ang. *Workload Balancing by Election*) [19]. Algorytm WINE, przez wzgląd na preferencje użytkowników, jest w stanie równoważyć i nadawać priorytety zapytaniom oraz aktualizacjom, dzięki czemu możliwe jest zachowanie optymalnego podziału pomiędzy świeżością otrzymywanych danych a czasem odpowiedzi. Dodatkowej analizie poddaje on również transakcje (zapytania, aktualizacje) wprowadzając pojęcie partycji, która może być reprezentowana przez podzbiór danych tabeli hurtowni danych. Założenia w dużym stopniu nie zostały zachowane. Niemniej jednak, metoda ustalania priorytetu oraz czasu wykonywania zapytań, a także aktualizacji została rozszerzona o schemat dwóch algorytmów z grupy FIFO [19], w których nie zastosowano definiowania poziomów dotyczących jakości danych oraz szybkości ich otrzymywania. Algorytmy FIFO-QH (ang. *First In First Out- Query High*) oraz FIFO-UH (ang. *First In First Out-Update High*) nastawione są na prostotę działania oraz ograniczenie liczby operacji przetwarzania kolejek [15].

Zgodnie z założeniami algorytmu WINE, wszystkie transakcje podzielone są na dwie grupy – tylko do odczytu (ang. *read-only*) oraz tylko do zapisu (ang. *write-only*). Zapytania przychodzące do systemu przechowywane są w kolejce zapytań – $Q=\{q_i|i \geq 0\}$, natomiast aktualizacje w kolejce aktualizacji – $U=\{u_j|j \geq 0\}$. Każda transakcja oznaczana jest znacznikiem czasowym (zapytania znacznikiem t_q, natomiast aktualizacje znacznikiem t_u), który określa czas przybycia transakcji do systemu.

4.1. Szeregowanie zapytań i aktualizacji

Główny schemat szeregowania w autorskim algorytmie WINE-HYBRIS obejmuje dwa, zależne od siebie etapy, które realizowane są tylko w przypadku, gdy obie kolejki mają przynajmniej po jednym elemencie (żadna z kolejek – aktualizacji oraz zapytań – nie jest pusta). Na wyższym (pierwszym) poziomie szeregowania następuje alokacja źródeł, które mają większy priorytet. Na niższym poziomie analizowane, a następnie przetwarzane są zapytania względem wartości QoS (ang. *Quality Of Service*) bądź aktualizacje względem wartości QoD (ang. *Quality of Data*).Wykorzystanie podziału szeregowania umożliwia wytypowanie kolejki o wyższym priorytecie.



Rys. 4. Główny schemat przetwarzania algorytmu WINE-HYBRIS
Fig. 4. Main diagram of the WINE-HYBRIS processing

W pierwszej fazie (na wyższym poziomie) szeregowania obliczana jest całkowita wartość miar QoS i QoD wszystkich zapytań, które oczekują w kolejce. Następnie na podstawie obliczonych wartości ustalany jest priorytet kolejek. Dla przykładu (rys. 4) analizując dane, sumy wynoszą kolejno $\sum QoS = 1{,}8$ oraz $\sum QoD = 2{,}1$. Ponieważ suma QoD jest większa od sumy QoS, wyższy priorytet zostaje ustawiony dla kolejki aktualizacji. Po przydzieleniu priorytetów, przeprowadzona zostaje druga faza szeregowania, w której następuje wybór transakcji (zapytania lub aktualizacji), który jest uzależniony od wyniku uzyskanego w pierwszym etapie szeregowania.

Po pierwszym etapie szeregowania, gdy wyższy priorytet otrzymała kolejka zapytań, znajdujące się w niej zapytania szeregowane są malejąco względem wartości QoS i rosnąco względem znacznika czasu t_{qi}. Jednakże takie założenie może prowadzić do pomijania i braku realizacji zapytań z małą wartością QoS (gdy do systemu ciągle będą napływać zapytania z dużą wartością QoS). Dlatego każdorazowo po operacji wykonania zapytania następuje inkrementacja wartości QoS, o wartość delta d[19], która zależy od długości kolejki zapytań |Q| oraz parametru r_{max} i wyraża się ona wzorem:

$$d = \frac{1}{|Q| \cdot r_{\max}} \tag{1}$$

Wartość parametru r_{max} stanowi integralną część metody, ponieważ każda modyfikacja wartości wpływa na szybkość wzrostu wartości inkrementacyjnej delta.

Podczas drugiego podetapu, drugiej fazy szeregowania (priorytetowanie aktualizacji), zatwierdzane zostają aktualizacje najbardziej pożądane przez użytkowników. Transakcje przed zaakceptowaniem zostają uszeregowane malejąco względem wartości w(u) oraz rosnąco względem znacznika czasowego t_{ui}. Wartość w(u) [19] obliczana jest według następującego wzoru:

$$w(u) = \sum_{\forall q_i, |P_{qi} \cap P_u| = 1} \frac{qod_{qi}}{1 + pos_{qi}} \tag{2}$$

Dzięki takiemu równaniu, aktualizacje cały czas zależą od zapytań, a dokładniej od ich miejsca w kolejce zapytań oraz wartości QoD. Aktualizacje są połączone z zapytaniami za pomocą wspólnych partycji, do których się odnoszą. Dodatkowo, poprzez sortowanie względem znacznika czasu, wiadomo również, że nie zdarzy się sytuacja nadpisywania aktualnych danych.

Trzeci etap drugiej fazy szeregowania jest wykorzystywany, gdy kolejka zapytań bądź aktualizacji nie ma żadnych transakcji. Wykorzystanie algorytmów FIFO-QH oraz FIFO-UH powoduje zmniejszenie liczby operacji i obliczeń przez przetwarzanie względem tylko znaczników czasowych. Wykorzystanie algorytmu FIFO-QH pozwala na przetworzenie zapytań, które znajdowały się na końcu kolejki ze względu na potrzebę aktualnych danych, w takiej kolejności, w jakiej zostały dostarczone do systemu (sortowanie jedynie względem znacznika czasowego). Dzięki zastosowaniu trzeciego etapu szeregowania, zapytania wszystkich użytkowników mają jednakowy priorytet, który nie zależy od parametrów QoS oraz QoD. W przypadku gdy kolejka zapytań jest pusta, możliwe jest wykonanie aktualizacji danych w hurtowni, które przez dłuższy czas nie były używane (odświeżane).

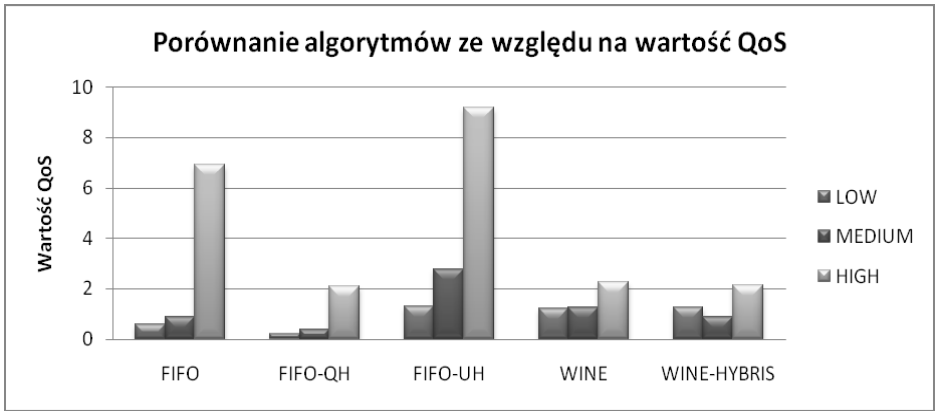
5. Testy i porównania

Przeprowadzono testy porównawcze wykonań algorytmu WINE-HYBRIS z grupą algorytmów FIFO oraz algorytmami, które wykorzystują preferencje użytkowników. Zaprezentowano zachowanie algorytmów oraz porównano szybkość przetwarzania zapytań i aktualizacji.

Wszystkie eksperymenty zostały wykonane na dwóch różnych maszynach, w celu weryfikacji zachowania algorytmu WINE-HYBRIS w odmiennych środowiskach sprzętowych. Pierwsza maszyna została wyposażona w 3 GB pamięci RAM DDR2 oraz procesor E8400, firmy Intel. Oprócz tego, komputer został doposażony w kartę graficzną NVidia GeForce 9600GT. Drugi z komputerów ma procesor i5 o taktowaniu 2.67 GHz. Jest to model proceso-

ra, w którym została zastosowana technologia Intel Turbo Boost, pozwalająca na automatyczne „podkręcenie" procesora, gdy komputer potrzebuje większej mocy obliczeniowej. Komputer został również wyposażony w 4 GB RAM oraz kartę graficzną GeForce GTX 260.

Pierwszy wykonany test miał na celu wykazanie szybkości przetwarzania zapytań – użyta została wartość QoS, która definiowana jest jako średni czas retencji, co oznacza średni czas pomiędzy przybyciem (t_q) i wykonaniem wszystkich zapytań z kolejki zapytań (wszystkich $q_i \in W$).



Rys. 5. Realizacja algorytmów w wymiarze parametru QoS
Fig. 5. Algorithms realization in aspect of the QoS parameter

Algorytm FIFO spisuje się bardzo dobrze jedynie dla małego obciążenia – przy zwiększającej się liczbie zapytań, a także aktualizacji zaczyna również rosnąć czas oczekiwania na odpowiedź (wysyłanie zapytań lub aktualizacji do DW, w takiej kolejności, w jakiej przyszły do systemu). Dany algorytm nie wykorzystuje również preferencji użytkowników, przez co zapytania z wysokim priorytetem oczekują w kolejce wraz z innymi transakcjami. Również algorytmy FIFO-QH oraz FIFO-UH, priorytetując kolejki zapytań bądź aktualizacji podczas wyboru operacji nie uwzględniają upodobań użytkowników. Algorytm FIFO-QH faworyzuje kolejkę zapytań, a co za tym idzie, nie wykonywane są żadne aktualizacje do czasu występowania zapytań w kolejce. Pomimo bardzo dobrego wyniku (algorytm FIFO-QH posiada najlepsze wyniki spośród wszystkich), dane otrzymywane przez użytkowników hurtowni nie są aktualne. Przeciwieństwem jest algorytm FIFO-UH, który ma bardzo wysokie wartości QoS, oznaczające, że zapytania oczekują dłuższy okres w kolejce, co jest spowodowane wykonywaniem wszystkich aktualizacji, a są realizowane dopiero w kolejnym kroku (faworyzowanie aktualizacji). Algorytmy WINE oraz WINE-HYBRIS działają stabilniej w porównaniu z algorytmami należącymi do grupy FIFO. Najważniejszą ich cechą jest wykorzystywanie preferencji użytkowników, przez co wykonywane są zapytania względem nadanych im priorytetów. Obydwa algorytmy dla małych obciążeń sprawują się podobnie – związane jest to z przyjętymi założeniami, mówiącymi o oparciu budowy algorytmu WINE-HYBRIS na schemacie algorytmu WINE. Pomimo tego, już dla średniego obciążenia możli-

we jest dostrzeżenie różnicy, która jest spowodowana faktem, iż kolejka aktualizacji w pewnym przedziale czasu została pusta, co spowodowało ograniczenie wykonywanych operacji do minimalnej liczby, poprzez zastosowanie trzeciego etapu drugiej fazy szeregowania w algorytmie WINE-HYBRIS.

Kolejne porównania dotyczą jedynie algorytmów WINE oraz WINE-HYBRIS, ze względu na fakt uwzględniania przez nie preferencji użytkowników. Dzięki wykorzystaniu dodatkowej architektury (CUDA) w algorytmie WINE-HYBRIS, możliwe jest zaobserwowanie wzrostu wydajności systemu, jak również zwiększonej liczby przetwarzanych zapytań oraz aktualizacji (rys. 6).

Rys. 6. Zależność czasu przetwarzania do liczby zapytań w kolejce w algorytmach WINE i WINE-HYBRIS

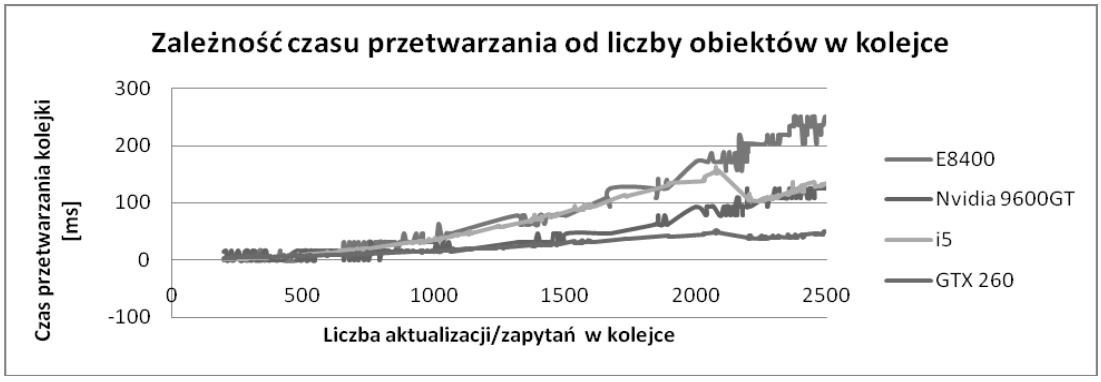
Fig. 6. The processing time dependence of the number of queue requests in WINE and WINE-HYBRIS algorithms

Na rys. 6 można zaobserwować skracanie czasu przetwarzania algorytmu WINE-HYBRIS. Zmiana oraz redukcja czasu przetworzenia kolejki następują dopiero po przekroczeniu wartości granicznej (wartości określającej liczbę elementów w kolejce), wtedy właśnie opłacalne jest wykorzystanie architektury CUDA. Z porównania algorytmów WINE oraz WINE-HYBRIS możemy wywnioskować, że ten drugi, oparty na architekturze CUDA jest efektywniejszy, pomimo ograniczeń magistrali, która transportuje dane pomiędzy kartą graficzną a urządzeniem hosta. Ze względu na dane limity transferu, tak ważne jest określenie odpowiedniej granicy, która oddziela wykonanie algorytmu oparte na architekturze CPU oraz CUDA.

Każde urządzenie lub podzespół komputera ma swoje specyficzne właściwości, dlatego niemożliwe jest ujednolicenie wartości granicznej dla każdego rodzaju karty graficznej, a także dla komputera. Rysunek 7 przedstawia porównanie czasu przetwarzania kolejek w dwóch komputerach – pierwszego, opierającego się na procesorze E8400 i karcie NVidia 9600GT oraz drugiego komputera z procesorem i5 i kartą graficzną GTX 260.

Podczas analizy rys. 7, prezentującego czas przetwarzania kolejek przy użyciu zarówno procesorów CPU, jak i procesorów GPU, możliwe jest zauważenie wielu interesujących zachowań. Pierwszym parametrem, który różni ww. komputery jest wartość graniczna – dla

komputera opartego na procesorze i5 i karcie GTX 260 jest ona mniejsza, co powoduje, że architektura CUDA jest wykorzystywana już przy mniejszej liczności kolejek. Drugą różnicą jest czas przetwarzania kolejek. Komputer z procesorem E8400 i kartę NVidia 9600GT gorzej radzi sobie z przetwarzaniem kolejek (zarówno ich czas przetwarzania w procesorze, jak i czas przetwarzania w karcie graficznej są większe). Przy zwiększeniu obciążenia, dzięki zastosowaniu w procesorze i5 nowej technologii (Intel Turbo Boost) jest on w stanie otrzymać wyniki porównywalne z kartą graficzną starszej generacji, a mianowicie NVidia 9600GT. Z kolei karta GTX 260 ma najkrótszy czas przetwarzania, co osiąga dzięki swojej budowie oraz liczbie 192 procesorów (karta NVidia 9600GT ma ich tylko 64). Jak widać z poniższego wykresu, użycie nowego typu karty graficznej, a co za tym idzie architektury CUDA w hurtowniach danych pozwala na kilkukrotne skrócenie czasu przetwarzania każdej z kolejek i wykonanie większej liczby zapytań i aktualizacji.



Rys. 7. Zależność czasu przetwarzania do liczby transakcji w kolejce dla architektur GPU oraz CPU

Fig. 7. Processing time in dependence of the number of transactions in the queue for GPU and CPU architectures

6. Podsumowanie

Ważne jest pełne wykorzystywanie mocy obliczeniowej urządzeń poprzez stosowanie odpowiednich algorytmów lub systemów. Stosowanie autorskiego algorytmu WINE-HYBRIS w architekturze CUDA, w budowie systemów ekstrakcji danych dla bezopóźnie-niowych hurtowni danych zapewnia wzrost liczby przetwarzanych zapytań oraz aktualizacji. Architektura CUDA, na której bazuje algorytm WINE-HYBRIS pozwala na zwiększenie wydajności poprzez użycie procesorów karty graficznej. System oparty na architekturze CUDA, można uruchomić na dowolnej karcie NVidia, zapewniając jego przenośność. W konsekwencji, bezopóźnieniowe hurtownie danych będą działały efektywniej i stabilniej.

BIBLIOGRAFIA

1. Bruckner R., Tjoa A. M.: Capturing Delays and Valid Times in Data Warehouses – Towards Timely Consistent Analyses. J. Intell. Inf. Syst. 19(2), 2002, s. 169÷190.
2. Bruckner R., List B., Schiefer J.: Striving towards Near Real-Time Data Integration for Data Warehouses. Data Warehousing and Knowledge Discovery, 4th International Conference, DaWaK'02, LNCS, Vol. 2454, France 2002, s. 317÷326.
3. CUDA w badaniach naukowych, http://www.nvidia.pl/object/cuda_home_new_pl.html.
4. CUDA processing flow, http://en.wikipedia.org/wiki/CUDA.
5. Gorawski M.: Zaawansowane hurtownie danych. Wydawnictwo Politechniki Śląskiej, (Rozprawa habilitacyjna), Gliwice 2009, s. 387.
6. Gorawski M.: Bezopóźnieniowe przestrzenne hurtownie danych z zapytaniami klasy kNN. CPI, IX edycja konferencji z cyklu hurtownie danych i business intelligence, Warszawa 2007.
7. Gorawski M., Marks P.: Grouping and Joining Transformations in Data Extraction Process. AI Informatica, Annales Univ.Marii Curie-Skłodowska, Vol. 4, 2006, s. 135÷147.
8. Gorawski M., Siódemak P.: Graficzne projektowanie aplikacji ETL. Studia Informatica, Vol. 24, No. 4(56), Wydawnictwo Politechniki Śląskiej, Gliwice 2003, s. 345÷367.
9. Gorawski M.: Ekstrakcja i integracja danych w czasie rzeczywistym. Kwiecień A., Gaj P. (red.): Współczesne problemy systemów czasu rzeczywistego. Wydawnictwa Naukowo-Techniczne, Warszawa 2004, s. 435÷445.
10. Gorawski M., Jabłoński P.: Uniwersalne środowisko graficzne do modelowania procesów ekstrakcji i odtwarzania. Studia Informatica, Vol. 26, No. 3, Wydawnictwo Politechniki Śląskiej, Gliwice 2005, s. 7÷28.
11. Gorawski M., Marks P.: Data Loading Based on UB-Tree Index Implemented in Design-Resume JavaBeans Environment. Studia Informatica, Vol. 25, No. 1, 2004, s. 141÷153.
12. Gorawski M., Piekarek M.: Rozproszony proces ekstrakcji danych z protokołem SimpleRMI. Kozielski S. i in. (red.): Bazy danych. Modele, technologie, narzędzia Analiza danych i wybrane zastosowania, tom 2. Wydawnictwa Komunikacji i Łączności, 2005, s. 43÷50.
13. Gorawski M.: 3 perspektywy procesu ekstrakcji danych. Nowak J. S., Grabara J. K., Szyjewski Z. (red.): Strategie informatyzacji i zarządzanie wiedzą. WNT, 2004, s. 295÷341.
14. Gorawski M.: Charakterystyka procesu ekstrakcji danych. Studia Informatica, Vol. 24, No. 4(56), Wydawnictwo Politechniki Śląskiej, Gliwice 2003, s. 211÷232.

15. Gorawski M., Wardas R.: The workload balancing ETL system basing on a learning machine. Studia Informatica, Vol. 31, No. 2A (89), Wydawnictwo Politechniki Śląskiej, Gliwice 2010, s. 517÷530.
16. Huiming Qu., Labrinidis A.: Preference-Aware Query and Update Scheduling in Web-database. Data Engineering, ICDE 2007.
17. JCUDA, Java and CUDA, http://www.jcuda.de/
18. Rahm E., Hai Do H.: Data Cleaning: Problems and Current approches. Bulletin of the Technical Committee on Data Engineering, Vol. 23. 2000.
19. Thiele M., Fischer U., Lehner W.: Partition-based Workload Scheduling in Living Data Warehouse Environments, DOLAP'07, Portugal ACM 2007.

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

Wpłynęło do Redakcji 31 stycznia 2011 r.

Abstract

The article presents newly designed WINE-HYBRIS algorithm, which is used in zero-latency DW. The system is based on two completely different architectures, which allows increased productivity and the number of processed queries and updates. The described system is also able to adjust to user's preferences. Due to the introduction of partitioning, the system updates only selected, revesed b queries parts (partitions) of whole data warehouse. Using the best features of the algorithm FIFO-QH, FIFO-UH, and WINE, the system performance is quite satisfying and is more efficient than the standard system based only on theWINE algorithm. Summing up, a system described in this paper bases on the WINE-HYBRIS algorithm, increases the number of queries and updates that are processed, and therefore, increases the system's capacity.

Adresy

Marcin GORAWSKI: Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Marcin.Gorawski@polsl.pl; Politechnika Wrocławska, Wydział Informatyki i Zarządzania, Instytut Informatyki, ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Marcin.Gorawski @pwr.wroc.pl.
Damian LIS: Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Damian. Lis@ polsl.pl.