POLITECHNIKA WARSZAWSKA
WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

PRACA DYPLOMOWA MAGISTERSKA

INFORMATYKA

Optymalizator kompresji szeregów czasowych na GPU

Autor:

Karol Dzitkowski

Promotor: dr inż. Krzysztof Kaczmarski

Warszawa, luty 2016

....

podpis promotora podpis autora

Abstrakt

Wiele urządzeń takich jak czujniki, stacje pomiarowe, czy nawet serwery, produkują ogromne ilości danych w postaci szeregów czasowych, które następnie są przetwarzane i składowane do późniejszej analizy. Ogromną rolę w tym procesie stanowi przetwarzanie danych na kartach graficznych w celu przyspieszenia obliczeń. Aby wydajnie korzystać z GPGPU przedstawiono szereg rozwiązań, korzystających z kart graficznych jako koprocesory w bazach danych lub nawet bazy danych po stronie GPU. We wszystkich rozwiązaniach bardzo istotną rolę stanowi kompresja danych. Szeregi czasowe są bardzo szczególnym rodzajem danych, dla których kluczowy jest dobór odpowiedniej kompresji wedle charakterystyki danych szeregu. W tej pracy przedstawię nowe podejście do kompresji szeregów czasowych po stronie GPU, przy użyciu planera budującego na bieżąco drzewa kompresji na podstawie statystyk napływających danych. Przedstawione rozwiązanie kompresuje dane za pomocą lekkich i bezstratnych kompresji w technologii CUDA.

Abstract

Many devices such as sensors, measuring stations or even servers produce enormous amounts of data in the form of time series, which are then processed and stored for later analysis. A huge role in this process takes data processing on graphics cards in order to accelerate calculations. To efficiently use the GPGPU a number of solutions has been presented, that use the GPU as a coprocessor in a databases. There were also attempts to create a GPU-side databases. It has been known that data compression plays here the crucial role. Time series are special kind of data, for which choosing the right compression according to the characteristics of the data series is essencial. In this paper I present a new approach to compression of time series on the side of the GPU, using a planner to keep building the compression tree based on statistics of incoming data. The solution compresses data using lightweight and lossless compression in CUDA technology.

Rozdział 1

Wstęp

Poniższa praca zawiera opis implementacji optymalizatora kompresji szeregów czasowych, bazującego na dynamicznie generowanych statystykach danych. Pomysł opiera się na tworzeniu drzew kompresji (kompresja kaskadowa) oraz zbieraniu statystyk o krawędziach takich drzew - jak dobrze dana para kompresji sprawdza się dla napływających danych. System będzie również dynamicznie zmieniał - korygował, takie drzewa w zależności od charakterystyki kolejnych paczek danych. W założeniu system ma umożliwić kompresję dużych ilości danych przy wykorzystaniu potencjału obliczeniowego współczesnych kart graficznych.

1.1 Procesory graficzne

Procesory graficzne stały się znaczącymi i potężnymi koprocesorami obliczeń dla wielu aplikacji i systemów, takich jak bazy danych, badania naukowe czy wyszukiwarki www. Nowoczesne GPU posiadają moc obliczeniową o rząd większą niż zwykle, wielordzeniowe procesory CPU, takie jak AMD FX 8XXX czy Intel Core i7. Dla przykładu flagowa konstrukcja firmy NVIDIA - GeForce GTX Titan X osiąga moc 6600 GFLOPS (miliardów operacji zmiennoprzecinkowych na sekundę), przy $336GB/s$ przepustowości pamięci, podczas gdy najszybsze procesory takie jak Intel Core i7-5960x osiągają niecałe 180 GFLOPS, przy przepustowości $68GB/s$. Kartom graficznym dorównują tylko inne jednostki typu SIMD, na przykład karty oblicze-

niowe Xeon Phi. Pomimo tak oszałamiających wyników, programowanie na jednostkach SIMD jest o wiele trudniejsze, jak również ograniczone przepustowością szyny PCI-E, która wynosi w porywach $8GB/s$, co dodatkowo przemawia za użyciem kompresji przy przetwarzaniu szeregów czasowych, choćby w celu przyspieszenia kopiowania danych z i na kartę graficzną w celu wykonania obliczeń.

1.2 Szeregi czasowe

Terabajty danych w postaci szeregów czasowych są przetwarzane i analizowane każdego dnia na całym świecie. Zapytania i agregacje na tak wielkich porcjach danych jest czasochłonne i wymaga dużej ilości zasobów. Aby zmierzyć się z tym problemem, powstały wyspecjalizowane bazy danych, wspierające analizę szeregów czasowych. Ważnym czynnikiem w tych rozwiązaniach jest kompresja oraz użycie procesorów graficznych w celu przyspieszenia obliczeń. Aby przetwarzać dane na GPU bez konieczności ich ciągłego kopiowania poprzez szynę PCI-E, powstają bazy danych po stronie GPU (najczęściej rozproszone), takie jak MapD lub DDJ (zaproponowana między innymi przeze mnie w poprzedniej pracy - inżynierskiej). Innymi rozwiązaniami są koprocesory obliczeniowe GPU, wspomagające działanie baz takich jak Cassandra, HBase, TepoDB, OpenTSDB czy PostgreSQL. Charakterystyka danych wielu szeregów wskazuje, że przy odpowiedniej obróbce mogą być kompresowane z bardzo dużym współczynnikiem, szczególnie jeśli byłoby możliwe kompresowanie za pomocą dynamicznie zmieniających się ciągów (różnych) algorytmów kompresji i transformacji danych. Dla przykładu, jeśli jakiś fragment szeregu jest stały, z nielicznymi wyjątkami, warto byłoby usunąć wyjątki, a resztę skompresować jako jedną liczbę - uzyskując współczynnik kompresji rzędu długości danych.

1.3 SIMD i lekka kompresja

Bazy danych przechowujące szeregi czasowe są najczęściej zorientowane kolumnowo oraz stosują metody lekkiej kompresji w celu oszczędności pamięci. W tych przypadkach stosuje się metody lekkiej kompresji, takie jak kodowanie słownikowe, delta lub stałej liczby bitów, zamiast bardziej skomplikowanych i wolniejszych metod, które często zapewniłyby lepszy poziom kompresji. Systemy te ładują swoje dane do pamięci trwałej paczkami, które mogą być kompresowane osobo i być może przy użyciu różnych algorytmów, zmieniających się dynamicznie w czasie. Takie kolumny wartości numerycznych tego samego typu wspaniale przetwarza się przy użyciu procesorów typu SIMD, co daje wielokrotne przyspieszenie w stosunku do tradycyjnych architektur. Okazuje się że większość algorytmów lekkiej kompresji może z dużym powodzeniem być w ten sposób zrównoleglona. Również dynamiczne generowanie statystyk napływających danych może być przyspieszone z użyciem SIMD, co otwiera możliwość implementacji wydajnych systemów, dynamicznie optymalizujących użyte kompresje w celu zwiększenia współczynnika kompresji danych. Dodatkowo użycie kaskadowej kompresji może wielokrotnie wzmocnić poziom kompresji, pod warunkiem stworzenia dobrego planu kompresji, właśnie na podstawie wygenerowanych statystyk. Ogromna moc obliczeniowa procesorów graficznych może pozwolić wygenerować taki plan w rozsądnym czasie. Takie użycie jest możliwe np. w bazach danych po stronie GPU, gdzie jest to niezmiernie ważne z powodu ścisłego limitu pamięci na kartach i ich wysokiego kosztu.

1.4 Zawartość pracy

W tej pracy przedstawię planer kompresji (optymalizator) kompresujący napływające paczki danych. Zaprezentuję nowe podejście, planera budującego drzewa kompresji i uczącego się ich konstrukcji na podstawie na bieżąco generowanych statystyk węzłów takich drzew (jak również statystyk napływających danych). Przedstawię również zaimplementowane środowisko oraz użyte algorytmy lekkiej kompresji. W

ramach tej pracy stworzone zostały 4 biblioteki, wykorzystujące technologię NVIDIA CUDA, tworzące framework optymalizatora kompresji oraz program w sposób równoległy kompresujący kolumny podanego szeregu czasowego. Następny podrozdział zawiera opis wcześniejszych prac prowadzonych w tych tematach, a także krótki opis architektury CUDA. W rozdziale 2 omówię stworzony framework oraz metody lekkiej kompresji, ze szczególnym uwzględnieniem kompresji FL oraz GFC. Rozdział 3 jest w całości poświęcony optymalizatorowi kompresji oraz generowaniu drzew i statystyk. Następnie przedstawię wyniki prac i eksperymentów. Ostatni rozdział (piąty) to podsumowanie oraz zakres przyszłych prac i optymalizacji.

1.5 Powiązane prace

1.5.1 Szeregi czasowe

Szeregi czasowe są typem danych dla których istnieje wiele efektywnych sposobów kompresji zależnych od ich charakterystyki. W wielu pracach przedstawiono podejścia do tego problemu od strony lekkiej kompresji. Najczęstszymi z nich są kodowanie ekstremami [12], stałej długości bitów [17], czyli tzw. NULL Suppression (NS) oraz proste kodowania słownikowe np. wszystkich unikalnych wartości, które można zakodować pewną założoną z góry liczbą bitów [24]. Dodatkowo do kompresji szeregów stosuje się metody regresji [8]. Autor stosuje Piecewise Regression - regresję odcinkową, polegającą na przybliżaniu kawałków szeregu funkcją, np. wielomianem. Ma to swoją wersję stratną jak i bezstratną, gdzie możemy zapisać różnicę od zadanej funkcji i wynik zapisać na mniejszej liczbie bitów. Szeregi czasowe to nie tylko liczby całkowitoliczbowe. Analizuje się także wiele sposobów kompresji liczb zmiennoprzecinkowych pojedynczej i podwójnej precyzji. Najczęściej próbuje się zamienić liczbę ułamkową na całkowitą stosując skalowanie [23]. Istnieją też bardziej skomplikowane metody na przykład kompresji liczb double algorytmem FPC [9], który kompresuje liniową sekwencję liczb o podwójnej precyzji (IEEE 754), sekwencyjnie przewidując każdą wartość, a następnie wykonując operację XOR z

prawdziwą wartością szeregu, po czym usuwane są wiodące zera. Rozwiązań jest jeszcze więcej, jak chociażby algorytm GFC [20] który zostanie omówiony w kolejnym rozdziale.

1.5.2 SIMD SSE

Biorąc pod uwagę algorytmy lekkiej kompresji dla szeregów, warto zwrócić uwagę na udane próby optymalizacji z użyciem prostego SIMD jakim są operacje wektorowe SSE na procesorach Intel [38] [16]. W tych pracach pokazano przekład algorytmów kodowania z wyrównaniem do bajtów (Byte-Aligned Coding) oraz do słów (Word-Aligned Coding) oraz zmierzono wydajność implementacji wektorowej wersji tych kodowań z użyciem SSE. Autorzy zastosowali również binarne pakowanie (Binary Packing) w formie algorytmu FOR (Frame of Reference) [39] dzieląc dane na bloki o długości 128 elementów (zmiennych całkowitych o długości 32 bitów) i stosując patchowanie. Taki algorytm okazał się najwydajniejszy. Pokazano, że bez spadku jakości kompresji można uzyskać w ten sposób wzrost szybkości kompresji od 2 do 4 razy w stosunku to tradycyjnej implementacji.

1.5.3 Obliczenia GPU

Dzięki ogromnej mocy obliczeniowej kart graficznych uzyskano znaczący wzrost wydajności wielu algorytmów dających się w mniejszym lub większym stopniu zrównoleglić. Przykładowymi algorytmami o tej właściwości są choćby radix sort[2], hashowanie kukułcze[3, 19], sumy prefixowe[1] i inne zaimplementowane w podstawowych bibliotekach takich jak CUDPP[1] czy Thrust[2]. Najważniejsze są jednak bardzo zadowalające rezultaty zrównoleglania algorytmów używanych w bazach danych takich jak index search[25], wszelkiego rodzaju agregacje i operacje join, scatter i gather[6] oraz obliczanie statystyk danych[37] jak również dopasowywanie wyrażeń regularnych[30]. Dla przykładu wzrost wydajności oferowany przez algorytmy

[1]CUDA Data Parallel Primitives - http://cudpp.github.io/

[2]Parallel algorithms library - https://developer.nvidia.com/thrust

z biblioteki Thrust, która jest niejako odpowiednikiem Std, jest średnio 10-krotny[41] w stosunku do najszybszych wersji CPU. Większość przytoczonych wyżej przykładów również reprezentuje wzrost wydajności o rząd wielkości. W pracach odnośnie akceleracji baz danych za pomocą technologii CUDA, autorzy otrzymują przyspieszenie $20 - 60$ krotne[10], w przypadku operacji *SELECT WHERE* i *SELECT JOIN* z agregacjami[6]. Jednak jest to liczone bez uwzględniania czasu kopiowania danych na GPU. Jak obliczono zajmuje to średnio ok 90% czasu działania algorytmów[13].

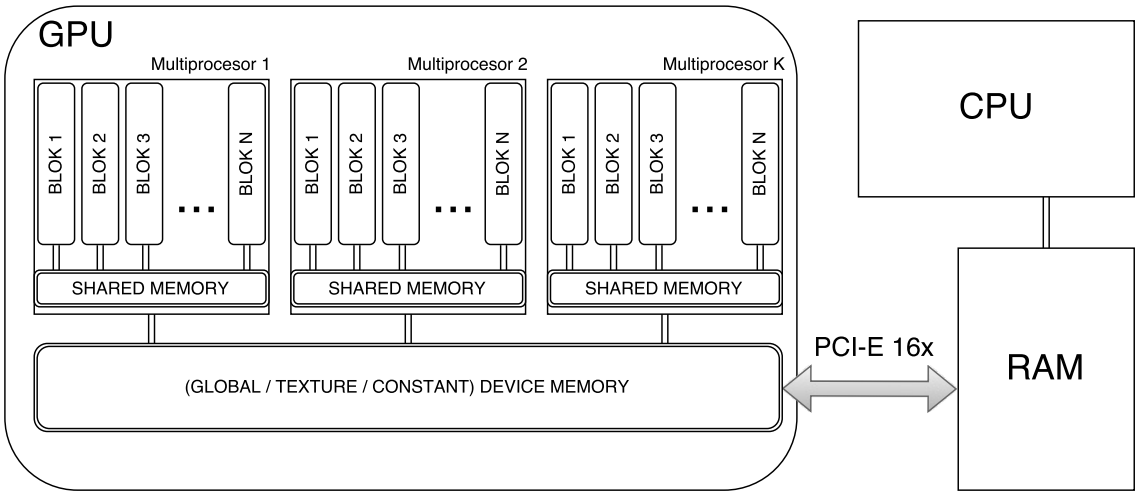
1.5.4 Kompresje

1.5.5 Planery kompresji

Okazuje się, że aby wielokrotnie zwiększyć współczynnik kompresji szeregu czasowego warto go przetransformować przed kompresją bądź nawet skompresować wielokrotnie różnymi algorytmami. W tym celu powstały planery kompresji które działają na zasadzie kodowania kaskadowego, czyli ciągu następujących po sobie kodowań, tworzących drzewo. Dzięki zastosowaniu procesorów graficznych osiągnięto bardzo dobre wyniki zarówno pod względem współczynnika kompresji, jak i szybkości działania. Dla przykładu przepustowość kodowania 45 GB/s oraz dekodowania 56 GB/s została osiągnięta przez Fang et al.[15]. W tym przypadku posługując się heurystykami wykorzystującymi statystyki danych wejściowych, spośród ogromnej ilości dostępnych schematów (> 500 tys.) kodowania (planów), wybierano najlepiej pasujące (np. dla danych posortowanych powinny zaczynać się od RLE itd.). Następnie wybierano spośród nich plan spełniający zdefiniowane normy, np. plan o największym współczynniku kompresji. Statystyki na których oparty był algorytm brane były z informacji o kolumnie w bazie danych. Zanotowano dużo lepszą kompresję niż w przypadku tradycyjnego kodowania pojedynczą metodą, dla realistycznych danych. Kolejnym, podobnym podejściem do planera jest praca Przymusa et al.[23], w której plan złożony jest z trzech warstw metod następujących po sobie: transformacji, kodowania bazowego i pomocniczego. Cechą charakterystyczną tego rozwiązania jest dynamiczny generator statystyk, który uaktualnia statysty-

ki w momencie tworzenia planu, wykorzystując właściwości poszczególnych metod takiej kompresji (szczególnie w przypadku minimalnej ilości bitów potrzebnych do zapisania każdej liczby z danego wektora danych). Dodatkowo praca ta implementuje znajdowanie optimum ze względu na dwie sprzeczne zmienne (bi-objective optimizer): szybkość działania i jakość kompresji, stosując optymalność Pareto[42]. Generowanie statystyk dla takiego planera na GPU zapewnia do 70 razy lepszą wydajność w stosunku do analogicznej implementacji na CPU[13].

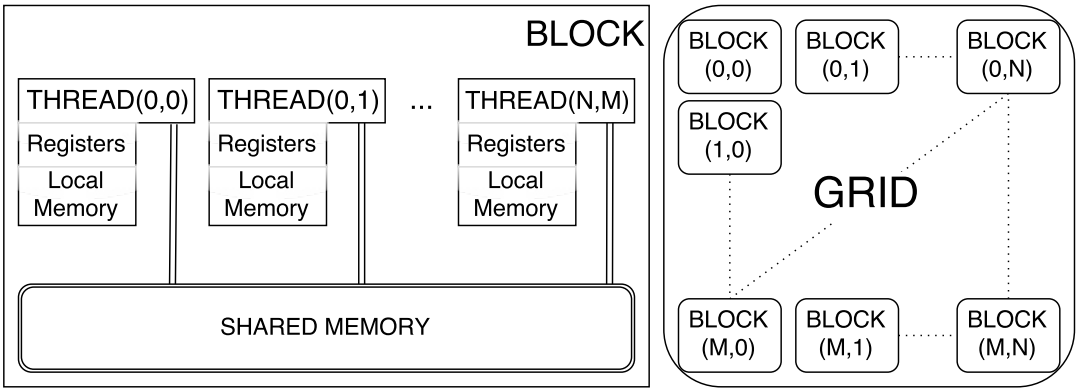
1.6 CUDA



Rysunek 1.1: Architektura NVIDIA CUDA

Jedną z wielu zalet architektury kart graficznych jest to, że składają się z kilku multiprocesorów (SMs - Streaming Multiprocessors) architektury SIMD. Jest to właściwie architektura typu SIMT, gdzie multiprocesor wykonuje wątki w grupach o liczności 32 zwanymi *warps*. Architektura ta jest zbliżona do SIMD z tą różnicą, że to nie organizacja wektora danych kontroluje jednostki obliczeniowe, a organizacja instrukcji pojedyńczego wątku. Umożliwia on zatem pisanie równolegle wykonywanego kodu dla niezależnych i skalowalnych wątków, jak i dla wątkuch koordynowanych danymi. Wszystkie wątki wykonują ten sam kod funkcji kernela. Ponadto CUDA tworzy abstrakcję bloków wątków, które zorganizowane są w siatkę (GRID) i

współdzielą zasoby multiprocesora. Ważna jest również hierarchia pamięci, w której część przydzielana jest wątkom w postaci pamięci lokalnej i rejestrów, oraz pamięci współdzielonej przez wątki z tego samego bloku (Shared Memory) - te pamięci muszą być znane w trakcie kompilacji kernela. Najwolniejsza jest pamięć globalna (Device Memory), wspólna dla wszystkich wątków, wszystkich bloków, na wszystkich multiprocesorach. Dokładny opis tej architektury można znaleźć bezpośrednio na stronie producenta.



Rysunek 1.2: Abstrakcja bloków i siatki w CUDA

Rozdział 2

Opis systemu

Rozdział 3

Optymalizator kompresji

Rozdział 4

Wyniki

Rozdział 5

Podsumowanie

Bibliografia

[1] Mark Harris, Shubhabrata Sengupta, and John D. Owens. *"Parallel Prefix Sum (Scan) with CUDA"*. In Hubert Nguyen, editor, GPU Gems 3, chapter 39, pages 851–876. Addison Wesley, August 2007

[2] Nadathur Satish, Mark Harris, and Michael Garland. *"Designing Efficient Sorting Algorithms for Manycore GPUs"*. In Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium, May 2009

[3] Alcantara, Dan A., et al. *"Real-time parallel hashing on the GPU."* ACM Transactions on Graphics (TOG) 28.5 (2009): 154

[4] Mostak, T., 2013. *"An overview of MapD (massively parallel database)."*, Massachusetts Institute of Technology, Cambridge, MA.

[5] Cloud RL, Curry ML, Ward HL, Skjellum A, Bangalore P. *"Accelerating lossless data compression with GPUs."* arXiv preprint arXiv:1107.1525. 2011 Jun 21.

[6] Pietroń, M., Pawel Russek, and Kazimierz Wiatr. *"Accelerating SELECT WHERE and SELECT JOIN queries on a GPU."* Computer Science 14.2) (2013): 243-252.

[7] Nicolaisen, Anders Lehrmann Vr?nning. *"Algorithms for Compression on GPUs."* (2013).

[8] Mensmann, Jörg, Timo Ropinski, and Klaus Hinrichs. *"A GPU-supported lossless compression scheme for rendering time-varying volume data."* 8th IEEE/EG international conference on Volume Graphics, 2–3 May 2010, Norrköping, Sweden. IEEE, 2010.

[9] Burtscher M, Ratanaworabhan P. *"FPC: A high-speed compressor for double-precision floating-point data."* Computers, IEEE Transactions on. 2009 Jan;58(1):18-31.

[10] Bakkum, Peter, and Kevin Skadron. *"Accelerating SQL database operations on a GPU with CUDA."* Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units. ACM, 2010.

[11] Ferreira, Miguel C. *"Compression and query execution within column oriented databases."* Diss. Massachusetts Institute of Technology, 2005.

[12] Fink, Eugene, and Harith Suman Gandhi. *"Compression of time series by extracting major extrema."* Journal of Experimental & Theoretical Artificial Intelligence 23.2 (2011): 255-270.

[13] Przymus, Piotr, and Krzysztof Kaczmarski. *"Compression Planner for Time Series Database with GPU Support."* Transactions on Large-Scale Data-and Knowledge-Centered Systems XV. Springer Berlin Heidelberg, 2014. 36-63.

[14] Ozsoy, Adnan, Martin Swany, and Anamika Chauhan. *"Pipelined parallel lzss for streaming data compression on GPGPUs."* Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on. IEEE, 2012.

[15] Fang, Wenbin, Bingsheng He, and Qiong Luo. *"Database compression on graphics processors."* Proceedings of the VLDB Endowment 3.1-2 (2010): 670-680.

[16] Lemire, Daniel, and Leonid Boytsov. *"Decoding billions of integers per second through vectorization."* Software: Practice and Experience 45.1 (2015): 1-29.

[17] Przymus, Piotr, and Krzysztof Kaczmarski. *"Dynamic compression strategy for time series database using GPU."* New Trends in Databases and Information Systems. Springer International Publishing, 2014. 235-244.

[18] Mani, Ganapathy. *"Data Compression using CUDA programming in GPU."* (2012).

[19] Alcantara, Dan A., et al. *"Real-time parallel hashing on the GPU."* ACM Transactions on Graphics (TOG) 28.5 (2009): 154.

[20] O'Neil, Molly A., and Martin Burtscher. *"Floating-point data compression at 75 Gb/s on a GPU."* Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2011.

[21] Zu, Yuan, and Bei Hua. *"GLZSS: LZSS lossless data compression can be faster."* Proceedings of Workshop on General Purpose Processing Using GPUs. ACM, 2014.

[22] Silberstein, Mark, et al. *"GPUfs: Integrating a file system with GPUs."* ACM Transactions on Computer Systems (TOCS) 32.1 (2014): 1.

[23] Al-Kiswany, Samer, Ammar Gharaibeh, and Matei Ripeanu. *"GPUs as storage system accelerators."* Parallel and Distributed Systems, IEEE Transactions on 24.8 (2013): 1556-1566.

[23] Przymus, Piotr, and Krzysztof Kaczmarski. *"Improving efficiency of data intensive applications on GPU using lightweight compression."* On the Move to Meaningful Internet Systems: OTM 2012 Workshops. Springer Berlin Heidelberg, 2012.

[24] Abadi, Daniel, Samuel Madden, and Miguel Ferreira. *"Integrating compression and execution in column-oriented database systems."* Proceedings of the 2006 ACM SIGMOD international conference on Management of data. ACM, 2006.

[25] Anh, Vo Ngoc, and Alistair Moffat. *"Inverted index compression using word-aligned binary codes."* Information Retrieval 8.1 (2005): 151-166.

[26] Eirola, Axel. *"Lossless data compression on GPGPU architectures."* arXiv preprint arXiv:1109.2348 (2011).

[27] Shyni, K., and Manoj Kumar KV. *"Lossless LZW Data Compression Algorithm on CUDA."* IOSR Journal of Computer Engineering (IOSR-JCE) 13.1 (2013): 122-127.

[28] Mostak, Todd. *"An overview of MapD (massively parallel database)."* White paper, Massachusetts Institute of Technology, Cambridge, MA (2013).

[29] Buchsbaum, Adam L., et al. *"Engineering the compression of massive tables: an experimental approach."* Symposium on Discrete Algorithms: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms. Vol. 9. No. 11. 2000.

[30] Morishima, Shin, and Hiroki Matsutani. *"Performance Evaluations of Document-Oriented Databases Using GPU and Cache Structure."* Trustcom/BigDataSE/ISPA, 2015 IEEE. Vol. 3. IEEE, 2015.

[31] Ozsoy, Adnan, Martin Swany, and Arun Chauhan. *"Optimizing LZSS compression on GPGPUs."* Future Generation Computer Systems 30 (2014): 170-178.

[32] Patel, Ritesh A., et al. *"Parallel lossless data compression on the GPU."* IEEE, 2012.

[33] Polychroniou, Orestis, Arun Raghavan, and Kenneth A. Ross. *"Rethinking SIMD vectorization for in-memory databases."* Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015.

[34] Bhatotia, Pramod, Rodrigo Rodrigues, and Akshat Verma. *"Shredder: GPU-accelerated incremental storage and computation."* FAST. 2012.

[35] Zukowski, Marcin, et al. *"Super-scalar RAM-CPU cache compression."* Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on. IEEE, 2006.

[36] Eichinger, Frank, et al. *"A time-series compression technique and its application to the smart grid."* The VLDB Journal 24.2 (2015): 193-218.

[37] Przymus, Piotr, and Krzysztof Kaczmarski. *"Time series queries processing with gpu support."* New Trends in Databases and Information Systems. Springer International Publishing, 2014. 53-60.

[38] Zhao, Wayne Xin, et al. *"A General SIMD-based Approach to Accelerating Compression Algorithms."* ACM Transactions on Information Systems (TOIS) 33.3 (2015): 15.

[39] Goldstein, Jonathan, Raghu Ramakrishnan, and Uri Shaft. *"Compressing relations and indexes."* Data Engineering, 1998. Proceedings., 14th International Conference on. IEEE, 1998.

[41] http://developer.download.nvidia.com/compute/cuda/compute-docs/cuda-performance-report.pdf

[42] Marler, R. Timothy, and Jasbir S. Arora. *"Survey of multi-objective optimization methods for engineering."* Structural and multidisciplinary optimization 26.6 (2004): 369-395.

Spis treści

Warszawa, dnia .....

Oświadczenie

Oświadczam, że pracę magisterską pod tytułem: „Tytuł pracy”, której promotorem jest prof. dr hab. Jan Wybitny, wykonałem/am samodzielnie, co poświadczam własnoręcznym podpisem.

.....