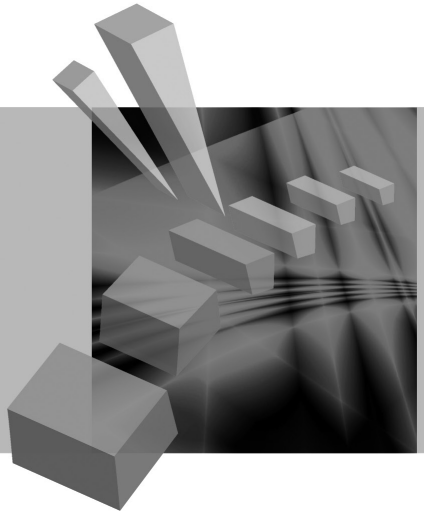
PRACE NAUKOWE
Uniwersytetu Ekonomicznego we Wrocławiu
RESEARCH PAPERS
of Wrocław University of Economics

278

Taksonomia 20

Klasyfikacja i analiza danych – teoria i zastosowania



Redaktorzy naukowi
Krzysztof Jajuga
Marek Walesiak

Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu
Wrocław 2013

Redaktor Wydawnictwa: Aleksandra Śliwka

Redaktor techniczny: Barbara Łopusiewicz

Korektor: Barbara Cibis

Łamanie: Małgorzata Czupryńska

Projekt okładki: Beata Dębska

Publikacja jest dostępna w Internecie na stronach:
www.ibuk.pl, www.ebscohost.com,
The Central and Eastern European Online Library www.ceeol.com,
a także w adnotowanej bibliografii zagadnień ekonomicznych BazEkon
http://kangur.uek.krakow.pl/bazy_ae/bazekon/nowy/index.php

Informacje o naborze artykułów i zasadach recenzowania znajdują się
na stronie internetowej Wydawnictwa
www.wydawnictwo.ue.wroc.pl

Tytuł dofinansowany ze środków Narodowego Banku Polskiego
oraz ze środków Sekcji Klasyfikacji i Analizy danych PTS

ISSN 1899-3192 (Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu)
ISSN 1505-9332 (Taksonomia)

Wersja pierwotna: publikacja drukowana

Druk: Drukarnia TOTEM

Spis treści

Summaries

PRACE NAUKOWE UNIWERSYTETU EKONOMICZNEGO WE WROCŁAWIU
RESEARCH PAPERS OF WROCŁAW UNIVERSITY OF ECONOMICS nr 278 • 2013

Taksonomia 20. Klasyfikacja i analiza danych – teoria i zastosowania ISSN 1899-3192

Andrzej Dudek
Uniwersytet Ekonomiczny we Wrocławiu,

Bartosz Kwaśniewski
Nokia Siemens Networks

PRZETWARZANIE RÓWNOLEGŁE ALGORYTMÓW ANALIZY SKUPIEŃ W TECHNOLOGII CUDA

Streszczenie: W artykule scharakteryzowano podstawy metodologiczne i technologiczne przetwarzania równoległego danych w technologii CUDA (*Compute Unified Device Architecture*) oraz dokonano przeglądu możliwości wykorzystania przetwarzania równoległego w technologii CUDA dla najbardziej znanych algorytmów analizy skupień, w tym klasyfikacji spektralnej [Ng i in. 2001]. W pracy ponadto wskazano miejsca, w których zastosowanie przetwarzania równoległego znacznie przyśpieszy czas ich wykonywania. Całość jest zakończona porównaniem empirycznych charakterystyk czasu pracy i otrzymanych rezultatów równoległych wersji algorytmów analizy skupień z implementacjami tych algorytmów w popularnym środowisku statystycznym **R** i w języku C++.

Słowa kluczowe: analiza skupień, przetwarzanie równoległe, CUDA.

1. Wstęp

Jednym z kluczowych zagadnień problemu klasyfikacji jest fakt, iż wraz ze wzrostem liczby klasyfikowanych obiektów liczba możliwych podziałów zbioru rośnie w tempie wykładniczym [Walesiak 2009, s. 246]. Problem klasyfikacyjny jest ze swej natury NP-zupełny, a co za tym idzie – bardzo istotna jest optymalizacja czasu wykonywania procedur analizy skupień [Gordon 1999, s. 40]. Istotnym elementem takiej optymalizacji jest pełne wykorzystywanie mocy obliczeniowej współczesnych komputerów. Jednym z najbardziej obiecujących sposobów skrócenia czasów wykonywania komputerowych implementacji algorytmów metod skupień jest wykorzystanie przetwarzania równoległego procesorów kart graficznych komputerów. W artykule scharakteryzowano stosunkowo nową architekturę obliczeniową CUDA i przedstawiono jej możliwości aplikacyjne dla algorytmów analizy skupień, a jego celem głównym jest wskazanie miejsc, w których zastosowanie przetwarzania równoległego może znacząco skrócić czasy wykonywania tych algorytmów. Ponadto w pracy opisany jest eksperyment symulacyjny, porównujący czasy wykonywania

procedur obliczenia wartości własnych oraz „pojedynczej" klasyfikacji spektralnej (bez estymacji parametrów klasyfikacji) w środowisku statystycznym R, w języku C++ i z wykorzystaniem technologii CUDA. Całość zamyka podsumowanie i wskazanie problemów otwartych i kierunków dalszych badań.

2. Problemy złożoności czasowej algorytmów analizy skupień

Jednym z najważniejszych problemów poprawnej analizy skupień jest zachowanie równowagi pomiędzy osiąganymi wynikami a czasem realizacji algorytmów, co osiągane jest poprzez odpowiednie konstruowanie algorytmów (zob. [Gordon 1999, s. 40]).

Wyzwania, przed którymi stoją badacze na początku XXI wieku, powodują, że nie zawsze możliwe jest osiągnięcie kompromisu pomiędzy jakością algorytmu analizy skupień a czasem jego realizacji, zwłaszcza w przypadku dużych (zawierających więcej niż 10 000 obiektów) zbiorów danych i zbiorów, w których naturalnie występują klasy o nietypowych kształtach, innych niż elipsoidalne kształty otrzymywane z wielowymiarowego rozkładu normalnego (zob. np. [Dudek 2012]). W takich przypadkach wydaje się, że ograniczenia obliczeniowe „dobrych" algorytmów klasyfikacyjnych (np. spowodowane koniecznością znalezienia wartości własnych w kolejnych jego etapach czy iteracyjnego znajdowania optymalnych parametrów algorytmu) wpływają na ich złożoność czasową i długi czas wykonywania, a z drugiej strony algorytmy „szybkie", takie jak CLARA [Kaufman, Rousseeuw 1990, s. 3] nie dają rezultatów odpowiadających rzeczywistej strukturze klas. Oprócz szukania optymalnych algorytmów, co jest najważniejszym zadaniem analizy skupień, niezbędne jest więc również szukanie metod skracania czasu wykonywania tych algorytmów poprzez odpowiednie wykorzystanie możliwości oferowanych przez współczesne komputery, takich jak przetwarzanie równoległe.

Bardziej szczegółowe opracowanie problemów złożoności czasowej i pamięciowej algorytmów analizy skupień, zwłaszcza dla dużych zbiorów danych, znajduje się w pracy [Dudek 2012].

3. Przetwarzanie równoległe i technologia CUDA

CUDA (*Compute Unified Device Architecture)* jest opracowaną przez firmę NVIDIA, równoległą architekturą obliczeniową, która zapewnia radykalny wzrost wydajności obliczeń dzięki wykorzystaniu mocy układów GPU (*Graphics Processing Unit* – jednostka przetwarzania graficznego) (za: http://www.nvidia.pl/).

Technologia CUDA wykorzystuje fakt, iż nowoczesne karty graficzne skonstruowane są z dużej liczby procesorów wykonujących operacje zmiennoprzecinkowe. W przeciwieństwie do procesorów typu CPU (*Central Processing Unit*) sterujących pracą komputerów w architekturze von Neumanowskiej[1] liczba procesorów (rdzeni)

[1] Co oznacza praktycznie wszystkie współcześnie używane komputery.

może sięgać kilkuset, podczas gdy typowe współczesne komputery zawierają procesory CPU 2- lub 4-rdzeniowe[2]. Procesory graficzne mają wprawdzie zdecydowanie ograniczoną w stosunku do procesorów CPU listę rozkazów, jednak zawierają prawie cały zestaw instrukcji obliczeń zmiennopozycyjnych, więc dla zastosowań typowo obliczeniowych te ograniczenia nie są istotne.

Typowe procesory GPU są około 2-3-krotnie wolniejsze niż procesory CPU. Ze względu jednak na ich liczbę[3] łączna moc obliczeniowa wszystkich rdzeni GPU w nowoczesnym komputerze może być kilkudziesięciokrotnie lub stukilkudziesięciokrotnie większa niż moc obliczeniowa procesora/procesorów jednostki centralnej komputera.

Przykładowo moc obliczeniowa wszystkich rdzeni CUDA karty graficznej GeForce GTX 680 (najbardziej wydajnej według stanu na początek roku 2012 karty graficznej NVIDIA) wynosi ponad 3000 000 000 000 (3 TFLOPS-ów) operacji zmiennoprzecinkowych na sekundę, podczas gdy moc obliczeniowa sześciordzeniowego procesora Intel Core i7 980 XE wynosi 109 000 000 000 (109 GFLOPS-ów).

CUDA jest najbardziej popularną technologią umożliwiającą wykorzystywanie w pełni mocy obliczeniowej rdzeni GPU. Od strony badacza/programisty najczęściej wykorzystywana jest ona przez programy w języku CUDA/C. Typowy program w tym języku składa się z części gospodarza (*host-a*) wykonywanej na procesorze CPU i procedur jądra (*kernel*) wykonywanych na rdzeniach GPU/CUDA. Wywołanie procedur jądra z poziomu gospodarza odbywa się ze wskazaniem struktury i liczby rdzeni (pogrupowanych w większe jednostki – bloki i kraty (*grids*)), na których mają być one wykonywane.

Pisanie efektywnych programów wykorzystujących przetwarzanie równoległe jest zadaniem trudnym, wymagającym od programisty zastosowania zupełnie innych technik niż w obliczeniach jednoprocesorowych w C/C++ czy R. Farber [2011 s. 13-17] wymienia trzy cechy efektywnych programów w języku CUDA/C:

- Minimalizowanie przesyłania danych z jednostki centralnej do GPU i odwrotnie.
- Dbanie o to, aby pojedyncze bloki obliczeń nie były zbyt krótkie, bo wtedy na wydajność zbyt duży wpływ ma czas, który musi zostać zużyty na ich uruchomienie.
- Korzystanie z pamięci wspólnej i wykorzystywanie bloków danych bez przeładowywania, na ile jest to możliwe.

4. Obszary zastosowań technologii CUDA w statystycznej analizie danych

Przetwarzanie równoległe w technologii CUDA, mimo że jest stosunkowo nową techniką obliczeniową, znalazło już wiele rzeczywistych zastosowań w algorytmach

[2] Przykładowo system operacyjny Windows 7 Professional „widzi" maksymalnie 8 rdzeni procesora CPU.

[3] Karta graficzna GeForce GT 640, na której były wykonywane dalsze obliczenia, zawiera 384 rdzenie GPU, taktowane 900 MHz z 2 GB pamięci taktowanej 1,8 Gb/s.

statystycznej analizy danych, przyśpieszając czas ich wykonania 3-300 krotnie. Ingram, Munzner i Olano [2009] opracowali równoległą implementację klasycznego skalowania wielowymiarowego, uzyskując przyspieszenie rzędu 10-15 razy. Firma Hewlett Packard [Wu, Zhang, Hsu 2009] przeprowadziła analizę skupień algorytmem klasycznej metody *k*-średnich w wersji na GPU w czasie poniżej 1 minuty, co oznacza około 300-krotne przyśpieszenie w stosunku do sekwencyjnej wersji tej metody wykonywanej na procesorze CPU. Przyśpieszenie podobnego rzędu otrzymali również [Hong-tao i in. 2009; Ma i Agrawal 2009; Shalom, Dash, Tue 2008].

Z kolei Kumar, Satoor i Buck [2009] opracowali implementację znanego algorytmu E-M wykorzystywanego m.in. w modelowaniu zmiennych dyskretnych i ograniczonych [Agresti 2002]) czy analizie dyskryminacyjnej, uzyskując przyśpieszenie rzędu 170 razy w stosunku do wersji „klasycznej".

Wśród ważnych zastosowań technologii CUDA wymienić można jeszcze implementację metody wektorów nośnych [Vapnik 1998]. Catanzaro, Sundaram, i Keutzer [2008] osiągnęli ponad 150-krotne przyśpieszenie w stosunku do wersji jednoprocesorowej.

W przypadku metod analizy skupień wyróżnić można dwa obszary zastosowań, w których wykorzystanie technologii CUDA może przynieść znaczne skrócenie czasu obliczeń:

- Algorytmy wykorzystujące operacje macierzowe zwłaszcza na dużych macierzach (por. [Farber 2011, s. 149-150, 203-204]). Zastosowanie przetwarzania równoległego znacznie przyśpiesza czasy standardowych operacji mnożenia, odwracania, szukania wyznacznika macierzy, a nawet umożliwia obliczenia na macierzach, których rozmiar jest większy niż pamięć karty graficznej, co nie jest możliwe w przypadku przetwarzania jednoprocesorowego.
- Algorytmy iteracyjnie estymujące parametry kluczowe dla procesu klasyfikacji. O ile każda z tych iteracji nie zależy od poprzedników, można osiągnąć znaczne przyśpieszenie czasu wykonywania tych algorytmów poprzez zastąpienie iteracji równoległym obliczeniem estymowanych parametrów i wybór parametru optymalnego w następnym kroku.

5. Obliczenie wartości własnych w procedurze klasyfikacji spektralnej

Technologia CUDA zostanie zastosowana w celu skrócenia czasu wykonywania procedury klasyfikacji spektralnej. Procedura klasyfikacji spektralnej ma wiele wariantów, dla wszystkich z nich można wyróżnić sześć najważniejszych etapów (wg [Ng, Jordan, Weiss 2002, za Walesiak, Dudek 2009]):

1. Konstrukcja macierzy danych $\mathbf{X} = [x_{ij}]$.

2. Zastosowanie estymatora jądrowego do obliczenia macierzy podobieństw $\mathbf{A} = \left[A_{ik} \right]$ (*affinity matrix*) między obiektami najczęściej obliczanej według wzoru (zob. [Karatzoglou 2006, s. 26]):

$$A_{ik} = \exp(-\sigma \cdot d_{ik}^2)\,,\ i,k = 1,\ \ldots,\ n\,, \tag{1}$$

gdzie: d_{ik} – odległość euklidesowa między obiektami *i* oraz *k*,
σ – parametr skali (szerokość pasma – *kernel width*).

3. Konstrukcja znormalizowanej macierzy Laplace'a $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ ($\mathbf{D}$ – diagonalna macierzy wag, w której na głównej przekątnej znajdują się sumy każdego wiersza z macierzy $\mathbf{A} = [A_{ik}]$, a poza główną przekątną są zera). W rzeczywistości znormalizowana macierz Laplace'a przyjmuje postać: $\mathbf{I} - \mathbf{L}$.

4. Obliczenie wartości własnych i odpowiadających im wektorów własnych (o długości równej jeden) dla macierzy **L**. Uporządkowanie wektorów własnych według malejących wartości własnych. Pierwsze *u* wektorów własnych (*u* – liczba klas) tworzy macierz $\mathbf{E} = [e_{ij}]$ o wymiarach $n \times u$.

5. Przeprowadzenie normalizacji tej macierzy zgodnie ze wzorem $y_{ij} = e_{ij} \Big/ \sqrt{\sum_{j=1}^{u} e_{ij}^{\,2}}$ ($i = 1,\ \ldots,\ n$ – numer obiektu, $j = 1,\ \ldots,\ u$ – numer zmiennej, *u* – liczba klas).

6. Macierz **Y** stanowi punkt wyjścia zastosowania klasycznych metod analizy skupień.

Krokiem o największej złożoności czasowej w procedurze klasyfikacji spektralnej jest krok 4 związany z obliczaniem wartości własnych. W tym celu zastosować można metodę obliczenia wartości własnych przez dekompozycję QR [Francis 1961]. Dekompozycja QR polega na przedstawieniu macierzy wejściowej w postaci iloczynu dwóch macierzy ortogonalnej macierzy **Q** i górnej trójkątnej macierzy **R**. Obliczenie macierzy **S** zawierającej wartości własne macierzy **L** poprzez dekompozycję QR odbywa się według następującego algorytmu:

1. $\mathbf{L} = \mathbf{Q_0} \times \mathbf{R_0}$ – macierz wejściowa, $\mathbf{S_0} = \mathbf{Q_0}$, $\mathbf{A_0} = \mathbf{L}$.
2. Powtarzaj ustaloną liczbę razy (lub do osiągnięcia konwergencji):
   dokonaj dekompozycji: $\mathbf{A_n} = \mathbf{Q_n} \times \mathbf{R_n}$,
   dokonaj podstawień: $\mathbf{A_{n+1}} = \mathbf{R_n} \times \mathbf{Q_n}$; $\mathbf{S_{n+1}} = \mathbf{S_n} \times \mathbf{Q_n}$,
   gdzie: *n* oznacza numer kroku iteracji.
3. Po zakończenie kroku 2 $\mathbf{A_n}$ na przekątnej zawiera wartości własne macierzy **L**, a $\mathbf{S_n}$ zawiera w kolumnach jej wektory własne.

Sama dekompozycja QR może odbywać się na kilka sposobów. Za najbardziej efektywną metodę uznaje się dekompozycję QR poprzez refleksję Householdera [1964]. Refleksja Householdera to przekształcenie (macierz mnożąca macierz wejściową), w którego wyniku otrzymuje się wektor zawierający na pierwszej pozycji wartość niezerową, a na pozostałych zera. Dekompozycja QR odbywa się w tym przypadku zgodnie z (2) i (3)

$$\mathbf{R} = \mathbf{H^{(n)}} \ldots \mathbf{H^{(2)}}\mathbf{H^{(1)}}\mathbf{L}\,, \tag{2}$$

$$\mathbf{Q} = \mathbf{H}^{(1)}\mathbf{H}^{(2)} \ldots \mathbf{H}^{(n)}, \tag{3}$$

gdzie: $\mathbf{L}$ – dekomponowana macierz,
$\mathbf{H}^{(i)}$ – refleksja o wymiarach $(n+1-i)\times(n+1-i)$ uzupełniona o zera poza przekątną główną i jedynki na przekątnej głównej.

Potencjalne przyśpieszenie implementacji tego algorytmu w technologii CUDA może wynikać z faktu, że poszczególne macierze $\mathbf{H}^{(i)}$ mogą być obliczane niezależnie, a co za tym idzie – obliczenia mogą być wykonane równolegle.

6. Eksperymenty symulacyjne

Eksperymenty symulacyjne porównujące czasy wykonania implementacji algorytmów w różnych językach, w tym w języku CUDA/C, zostały podzielone na dwie grupy. W pierwszej porównano czasy obliczeń tylko wartości własnych, w drugim czasy wykonywania pełnej pojedynczej procedury klasyfikacji spektralnej.

W obu eksperymentach porównano implementacje tych samych algorytmów w języku R, języku C++ z zastosowaniem bibliotek LAPACK [Anderson i in. 1999] oraz w języku CUDA/C. Dwie pierwsze implementacje jako procedury jednowątkowe nie różniły się od siebie znacząco, natomiast w implementacji wykorzystującej technologię CUDA zastosowano wykonywane równolegle mnożenia macierzy, mnożenia macierzy przez skalar oraz obliczanie wartości własnych zgodnie z procedurą opisaną w poprzednim punkcie.

Tabela 1 przedstawia czasy obliczania wartości własnych dla różnych rozmiarów symetrycznych macierzy wejściowych. Dla każdego rozmiaru wygenerowanych zostało 30 zbiorów danych, a tab. 1 przedstawia średnie ze wszystkich obliczeń.

Tabela 1. Średnie czasy obliczeń wartości własnych

Rozmiary macierzy	R	C++	CUDA
58 x 58	58 ms.	44 ms.	69 ms
429 x 429	1,02 s.	4,49 s.	743 ms
839 x 839	6,56 s.	22,32 s.	1,21 s.
1975 x 1975	75,34 s.	131,45 s.	7,45 s.

Źródło: opracowanie własne.

Tabela 2 przedstawia średnie (z 30 pojedynczych symulacji) czasy obliczenia pełnej procedury klasyfikacji spektralnej (bez automatycznego znajdowania parametru σ).

W obu przypadkach osiągnięto przyśpieszenie o jeden rząd wielkości (w przybliżeniu 10-krotne) dla większych rozmiarów macierzy wejściowych. W przypadku mniejszych macierzy przyśpieszenia nie osiągnięto lub wręcz zastosowanie techno-

Tabela 2. Średnie czasy wykonywania procedury klasyfikacji spektralnej

Rozmiary macierzy	R	C++	CUDA
50 x 6	480 ms	427 ms	734 ms
60 x 4	414 ms	4,49 s	747 ms
234 x 8	1,62 s	1,41 s	843 ms
520 x 7	11,97 s	13,23 s	2,47 s
1099 x 11	179,4 s	185, 21 s	16,3 s

Źródło: opracowanie własne.

logii CUDA spowodowało wydłużenie czasu wykonywania algorytmu, co jest zgodne z drugą z opisanych zasad programowania równoległego dotyczącą zbyt małych bloków danych.

7. Podsumowanie i problemy otwarte

W artykule scharakteryzowano technologię przetwarzania równoległego CUDA i możliwości jej zastosowań w algorytmach analizy skupień ze szczególnym uwzględnieniem procedury klasyfikacji spektralnej. Osiągnięte wyniki symulacyjne wskazują około 10-krotne przyśpieszenia czasu wykonywania „pojedynczego przebiegu" procedury klasyfikacji spektralnej w stosunku do implementacji w środowisku statystycznym R i w języku C++ w bibliotece LAPACK.

Problemem otwartym jest opracowanie wersji równoległej procedury analizy spektralnej z automatycznym określaniem wartości parametru σ – szerokości pasma [Walesiak, Dudek 2009]), a w szerszej perspektywie opracowanie biblioteki metod analizy skupień i innych gałęzi wielowymiarowej analizy statystycznej wykonywanych na rdzeniach GPU w technologii CUDA i połączenie ich ze środowiskiem R.

Literatura

Agresti A. (2002), *Categorical Data Analysis*, John Wiley and Sons, New York.

Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling A., McKenney A., Sorensen D. (1999), *LAPACK User's Guide*, Third Edition, SIAM, Philadelphia.

Catanzaro B., Sundaram N., Keutzer K. (2008), *Fast support vector machine training and classification on graphics processors*, Proceedings of the 25th international conference on machine learning, New York.

Dudek A. (2012), *Classification of large datasets. Problems, methods, algorithms*, Acta Universitatis Lodziensis, Folia Oeconomica (w druku).

Farber R. (2011), *CUDA Applications Design and Development*, Morgan Kaufman, Amsterdam-Boston(...) Tokyo.

Francis J.G.F. (1961), *The QR transformation, I*. "The Computer Journal", vol. 4, no. 3, s. 265-271.

Gordon A.D. (1999), *Classification*, Chapman & Hall/CRC, London.
Hong-tao B., Li-li H., Dan-tong O., Zhan-shan L., He L. (2009), *K-Means on Commodity GPUs with CUDA*, World Congress on Computer Science and Information Engineering, s. 651-655.
Householder A.S. (1964), *The Theory of Matrices in Numerical Analysis*, Dover Publications, Inc., New York.
Ingram S., Munzner T., Olano M. (2009), *Glimmer: Multilevel MDS on the GPU*, IEEE Transactions on Visualization and Computer Graphics, s. 249-261.
Karatzoglou A. (2006), *Kernel methods. Software, algorithms and applications*, Rozprawa doktorska, Uniwersytet Techniczny we Wiedniu.
Kaufman L., Rousseeuw P.J. (1990), *Finding Groups in Data: an Introduction to Cluster Analysis*, Wiley, New York.
Kumar N., Satoor S., Buck I. (2009*), Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs Using CUDA*. High Performance Computing and Communications, 2009, HPCC '09. Seoul, 103-109.
Ma W., Agrawal G. (2009), *A translation system for enabling data mining applications on GPUs*, Proceedings of the 23rd international conference on Supercomputing, New York.
Ng A., Jordan I., Weiss Y. (2001), *On Spectral Clustering. Analysis and an Algorithm, Neural Information Processing Symposium*.
Shalom S.A., Dash M., Tue M. (2008), *Efficient K-means Clustering Using Accelerated Graphics Processors*, [w:] I.-Y. Song, J. Eder, T. Nguyen, *Data Warehousing and Knowledge Discovery*, Heidelberg, Springer Berlin, s. 166-175.
Vapnik V. (1998), *Statistical Learning Theory*, John Wiley & Sons, New York.
Walesiak M. (2009), *Analiza skupień,* [w:] M. Walesiak, E. Gatnar (red.), *Statystyczna analiza danych z wykorzystaniem programu R*, PWN, Warszawa, s. 407-433.
Walesiak M., Dudek A. (2009), *Odległość GDM dla danych porządkowych a klasyfikacja spektralna*, Prace Naukowe UE we Wrocławiu nr 84, Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu, Wrocław, s. 9-19.
Wu R., Zhang B., Hsu M. (2009), *Clustering billions of data points using GPUs*, Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop. ACM.

PARALLEL PROCESSING OF CLUSTERING ALGORITHMS IN CUDA TECHNOLOGY

Summary: In this paper methodological and technological basis of parallel processing in CUDA (Compute Unified Device Architecture) technology are characterized along with an overview of its application possibilities for most known clustering algorithms including spectral classification [Ng et al. 2001]. Places for which parallel processing can drastically increase performance times are pointed. The paper is completed with empirical comparison of the characteristics of time and received results of the parallel version of the clustering algorithm with implementations of these algorithms from the popular statistical environment **R** and with C++ implementations.

Keywords: clustering, parallel processing, CUDA.