PRACE NAUKOWE
Uniwersytetu Ekonomicznego we Wrocławiu

RESEARCH PAPERS
of Wrocław University of Economics

Nr 384

Taksonomia 24
Klasyfikacja i analiza danych – teoria i zastosowania

Redaktorzy naukowi
Krzysztof Jajuga
Marek Walesiak

Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu
Wrocław 2015

Redaktor Wydawnictwa: Aleksandra Śliwka

Redaktor techniczny: Barbara Łopusiewicz

Korektor: Barbara Cibis

Łamanie: Beata Mazur

Projekt okładki: Beata Dębska

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

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

ISSN 1899-3192 (Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu)
e-ISSN 2392-0041 (Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu)
ISSN 1505-9332 (Taksonomia)

Wersja pierwotna: publikacja drukowana

Zamówienia na opublikowane prace należy składać na adres:
Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu
tel./fax 71 36 80 602; e-mail:econbook@ue.wroc.pl
www.ksiegarnia.ue.wroc.pl

Druk i oprawa: TOTEM

Spis treści

Summaries

PRACE NAUKOWE UNIWERSYTETU EKONOMICZNEGO WE WROCŁAWIU
RESEARCH PAPERS OF WROCŁAW UNIVERSITY OF ECONOMICS nr 384 • 2015
Taksonomia 24 ISSN 1899-3192
Klasyfikacja i analiza danych – teoria i zastosowania e-ISSN 2392-0041

Krzysztof Najman
Uniwersytet Gdański
e-mail: krzysztof.najman@ug.edu.pl

ZASTOSOWANIE PRZETWARZANIA RÓWNOLEGŁEGO W ANALIZIE SKUPIEŃ

Streszczenie: W badaniach społecznych coraz częściej spotykane są zbiory danych zawierające miliony jednostek opisanych tysiącami cech. Badanie struktury grupowej jednostek zawartych w takich zbiorach nastręcza specyficznych problemów. Jednym z podstawowych jest możliwość dokonania grupowania w możliwym do zaakceptowania czasie. Problem ten wynika z dwóch podstawowych źródeł. Pierwszym jest rozmiar i szybkość powiększania się samego zbioru danych. Drugim z kolei jest złożoność numeryczna stosowanych w analizie skupień algorytmów. Celem prezentowanych badań jest wykazanie możliwości zastosowania przetwarzania równoległego w analizie skupień. Przedstawione zostaną podstawowe metody wielowątkowych algorytmów wyznaczania typowych miar statystycznych stosowanych w analizie skupień, a także wielowątkowy algorytm metody *k*-średnich. Praktyczne aspekty ich zastosowania zostaną pokazane na podstawie badań symulacyjnych w środowisku Matlab.

Słowa kluczowe: akceleracja algorytmów, analiza skupień, przetwarzanie równoległe.

DOI: 10.15611/pn.2015.384.22

1. Wstęp

We współczesnych badaniach społecznych, ekonomicznych, rynkowych czy marketingowych gromadzi się informacje o coraz większej liczbie jednostek opisanych coraz większą liczbą cech. Szczególnie wyraźnie jest to widoczne, gdy przedmiotem badania są usługi świadczone za pośrednictwem Internetu lub innych sieci telekomunikacyjnych. Rejestry rozmów telefonicznych, wysyłanych wiadomości SMS czy email, wiadomości wysyłanych między użytkownikami portali społecznościowych, transakcji z wykorzystaniem kart płatniczych mogą zawierać nawet setki milionów jednostek opisanych setkami cech.

W 2013 roku Intel opublikował swoje szacunki dotyczące dynamiki wzrostu objętości internetowych zbiorów danych w raporcie „What happens in an Internet

Minute?"[1]. Z jego treści wynika, że w ciągu jednej minuty wyszukiwarka Google rejestruje ponad 4 miliony zapytań, użytkownicy serwisu Facebook przeglądają ponad 6 milionów wiadomości, a użytkownicy serwisu Flickr.com przeglądają ponad 20 milionów zdjęć. Badanie struktury grupowej jednostek zawartych w takich zbiorach nastręcza specyficznych problemów. Podstawowym jest możliwość dokonania grupowania w ogóle. Jeżeli w algorytmie grupowania konieczne jest wyznaczenie pełnej macierzy odległości wszystkich zarejestrowanych jednostek, jak np. w aglomeracyjnych metodach hierarchicznych, to może być to niemożliwe ze względu na jej ogromne rozmiary. Macierz odległości dla zaledwie 20 000 jednostek wymaga niemal 3GB pamięci RAM komputera. Zwiększenie liczby jednostek do 30 000 wymaga już 6.7GB pamięci, a dla 100 000 nawet 74GB, co przekracza całkowitą jej ilość w znakomitej większości współczesnych komputerów. Drugim problemem jest złożoność numeryczna stosowanych w analizie skupień algorytmów. Wyznaczenie wartości własnych dla powyższych zbiorów danych może wymagać wielu godzin pracy komputera. W tym samym czasie zbiór danych może się powiększyć o tysiące czy miliony nowych obserwacji. W konsekwencji w momencie uzyskania wyników grupowania mogą być one już nieaktualne.

Oba powyższe problemy próbuje się minimalizować. Wpływ rozmiaru zbioru danych na możliwość realizacji analizy skupień można ograniczyć przez budowę agregatów jednostek czy odpowiednie próbkowanie [Kollios i in. 2003]. Liczbę cech można ograniczyć, stosując odpowiednie metody ich wyboru [Migdał-Najman, Najman 2013, s. 147-153]. Trudniej jest zmniejszyć złożoność obliczeniową stosowanych algorytmów. Ich konstrukcja realizuje zwykle pewien zamysł badacza (np. minimalizację wybranego kryterium grupowania), który nie może być zmieniony. Konieczne jest więc opracowywanie nowych algorytmów, oszczędnie korzystających z zasobów komputera [Migdał-Najman, Najman 2013 s. 237-238].

Zwiększenie szybkości algorytmów grupowania może być dokonywane na wiele sposobów. Jednym z nich jest zastosowanie przetwarzania równoległego korzystającego z wielu rdzeni współczesnych procesorów (CPU, *Central Processing Unit*), a także obliczeń z wykorzystaniem procesorów graficznych (GPU, *Graphics Processing Unit*) [Wasif, Narayanan 2011]. Wydaje się, że gdyby proces grupowania można było rozłożyć na wiele podprocesów, z których przynajmniej niektóre mogłyby być realizowane jednocześnie, czas konieczny na wykonanie niezbędnych obliczeń mógłby zostać wyraźnie skrócony. Celem prezentowanych badań jest weryfikacja powyższej hipotezy.

[1] http://www.intel.pl/content/www/pl/pl/communications/internet-minute-infographic.html [16.10.2014r.].

2. Elementarne czynniki wpływające na szybkość obliczeń numerycznych

Podstawowym problemem w analizie danych zawartych w dużych zbiorach jest sama ich ilość. Wyznaczenie nawet podstawowych statystyk dla poszczególnych cech opisujących miliony jednostek zajmuje współczesnym komputerom zaskakująco dużo czasu (por. tab. 1). Gdyby w iteracyjnej metodzie grupowania konieczne było stukrotne wyznaczenie wektora (o rozmiarze n) odchyleń standardowych dla kwadratowej macierzy danych o rozmiarze 10 000 × 10 000 ($n \times n$), to czas obliczeń wynosiłby ok. 0,05 sekundy[2]. Wraz ze wzrostem rozmiaru macierzy czas obliczeń także szybko rośnie. Już przy n = 500 000 czas ten wyniesie ponad 2 minuty. Gdy cała macierz przestanie mieścić się w pamięci RAM komputera, nastąpi skokowy przyrost czasu obliczeń, związany z koniecznością zapisu i odczytu danych w czasie obliczeń na dysk. Gdy n = 1 000 000, czas obliczeń jednej, elementarnej statystyki wyniesie już ponad 8 minut. Znacznie bardziej wymagające jest wyznaczenie macierzy odległości euklidesowych między badanymi jednostkami. Stukrotne wyznaczenie takiej macierzy (np. w metodzie k-średnich por. [Zechner, Granitzer 2009]) dla 5000 jednostek zajmie niemal 13 minut. W niektórych zastosowaniach szybkość rejestrowania nowych danych jest więc znacząco większa niż szybkość ich przetwarzania.

Tabela 1. Czas analizy wybranych procedur numerycznych dla 100 iteracji

Rozmiar macierzy [$n \times n$]	Macierz odległości	Rozmiar macierzy [$n \times n$]	Odch. standardowe	Rozmiar macierzy [$n \times n$]	Wartości własne	Rozmiar macierzy [$n \times n$]	Metoda k-średnich
	czas w sekundach		czas w sekundach		czas w sekundach		czas w sekundach
100	0,04	10 000	0,07	1 000	0,05	1 000	0,11
200	0,06	20 000	0,22	2 000	0,34	100 000	3,60
400	0,41	40 000	0,81	3 000	1,28	600 000	35,48
800	3,41	100 000	5,02	4 000	3,45	1 100 000	118,65
1000	6,46	150 000	11,42	5 000	6,97	1 600 000	170,19
1500	21,64	200 000	20,61	6 000	12,05	2 100 000	273,40
2000	50,59	300 000	44,74	7 000	19,33	2 600 000	326,94
3000	168,11	400 000	78,87	8 000	30,42	3 100 000	274,61
4000	395,63	500 000	129,03	10 000	56,51	3 600 000	324,86
5000	769,22	1 000 000	500,12	13 000	130,87	4 100 000	337,03

Źródło: opracowanie własne.

Znaczącego skrócenia czasu obliczeń tego typu można dokonać, rozpoczynając od zadeklarowania wszystkich niezbędnych w analizie macierzy. Wielu użytkowników, programując procedury analizy skupień, zapomina o tym, jak wiele czasu

[2] Wszystkie opisane czasy obliczeń dotyczą zestawu: CPU: Intel Core i7 4930K 4.2 GHz (6 rdzeni), GPU: NVIDIA GeForce GTX 780 OC, RAM: 32GB 2133MHz.

potrzeba na wypełnianie zmiennej macierzowej danymi. Powiedzmy, że zmienna X ma zawierać trzy elementy typu rzeczywistego, np. 1, 2 i 3. Jeżeli procedurę przypisania zapiszemy następująco: X(1) = 1, X(2) = 2, X(3) = 3, to w rzeczywistości kompilator dowolnego języka programowania przeprowadzi wiele operacji. Utworzy najpierw zmienną X zawierającą jedno pole i w to pole wpisze następnie wartość 1. W kolejnym kroku poszerzy zakres zmiennej X o nowe dwa pola. Do drugiego przepisze wartość 1, a do trzeciego doda wartość 2. Następnie wykasuje pierwszą pozycję tej zmiennej z wartością 1. Jeżeli z góry zadeklarujemy, że rozmiar wektora wynosi 3, przypisanie nastąpi jednorazowo. Różnica w czasie wykonania obu wersji programu jest zaskakująca. Rozważmy prostą pętlę[3] zapisaną w tab. 2.

Tabela 2. Warianty deklaracji zmiennych w prostej pętli *for*

Pętla bez deklaracji rozmiaru macierzy	Pętla ze zdefiniowanym rozmiarem macierzy X. Macierz wypełniona wartościami 0

`for  i=1:100000`
`  X(i) = i;`
`end.`
	`X = zeros(100000,1);`
	`for  i=1:100000`
	`  X(i) = i;`
	`end.`

Czas wykonania procedury bez deklaracji rozmiaru macierzy X to 0,07662 sekundy. Deklarując jej rozmiar, czas wykonania, i to łącznie z czasem wypełnienia tej macierzy zerami, wynosi zaledwie 0,001053 sekundy, a więc trwa to 72 razy szybciej. Gdy w algorytmie grupowania wystąpi pętla złożona (pętla w pętli), różnica staje się jeszcze większa.

Tabela 3. Warianty deklaracji zmiennych w złożonej pętli *for*

Pętla bez deklaracji rozmiaru macierzy	Pętla ze zdefiniowanym rozmiarem macierzy X. Macierz wypełniona wartościami 0

`for  i=1:5000`
`  for j=1:5000`
`  X(i,j) = i;`
`  end`
`end.`
	`X = zeros(5000,5000);`
	`for  i=1:5000`
	`  for j=1:5000`
	`  X(i,j) = i;`
	`  end`
	`end.`

Czas wypełnienia macierzy bez deklaracji rozmiaru wyniósł tu aż 73 sekundy, podczas gdy z deklaracją jedynie 0,39 sekundy, czyli aż 185 razy szybciej.

Opisana powyżej zasada programistyczna jest bardzo prosta. Przyspieszenie algorytmów nie jest jednak aż tak spektakularne, ponieważ wypełnianie macierzy

[3] Wszystkie procedury zostały zapisane w uproszczeniu, w standardzie środowiska Matlab.

danymi stanowi zwykle mały ułamek czasu pracy danego algorytmu. Gdy tego typu modyfikacje kodu programu nie wystarczą, konieczne jest zastosowanie znacznie bardziej wyrafinowanych technik.

3. Obliczenia równoległe na CPU

Kolejną możliwością zredukowania czasu obliczeń w algorytmach grupowania jest wykorzystanie w tym celu wielu procesorów (lub rdzeni w jednym procesorze). Obecnie na rynku dostępne są procesory zawierające w sobie od 2 do 24 rdzeni, z których każdy może niezależnie od innych wykonywać obliczenia numeryczne. Rozbicie danego algorytmu na części, które będą od siebie numerycznie niezależne[4] , może potencjalnie przyspieszyć obliczenia. Programowanie tego typu algorytmów jest obecnie dość trudne, jednak niektórzy producenci pakietów metod numerycznych bardzo wspomagają użytkowników w tym procesie. Przykładem takiego wspomagania jest biblioteka *Parallel Computing Toolbox* dla środowiska Matlab[5]. Oferuje ona między innymi nowy typ pętli o nazwie *parfor*. Różni się ona od zwykłej pętli *for* tym, że obliczenia w niej zawarte wykonywanie są w sposób wielowątkowy, a więc równocześnie na wielu procesorach.

Korzyść ze stosowania pętli *parfor* jest znaczna. Już dla pętli o 1000 iteracjach przyrosty szybkości są kilkukrotne. Co ważne, nie jest tak, że pracując na 6 procesorach, uzyskujemy przeciętnie sześciokrotne przyspieszenie obliczeń. Część czasu zajmuje zarządzanie procesem rozdzielania pracy między poszczególnymi proceso-

Tabela 4. Przyrost szybkości analizy wybranych procedur dla 1000 iteracji pętli *parfor*.

Rozmiar macierzy [$n \times n$]	Macierz odległości	Rozmiar macierzy [$n \times n$]	Odch. standardowe	Rozmiar macierzy [$n \times n$]	Wartości własne	Rozmiar macierzy [$n \times n$]	Metoda *k*-średnich
	1 CPU / 6 CPU		1 CPU / 6 CPU		1 CPU / 6 CPU		1 CPU / 6 CPU
100	4,10	10 000	0,21	1 000	0,17	1 000	0,57
200	4,70	20 000	1,24	2 000	1,03	100 000	3,59
400	4,90	40 000	3,22	3 000	1,71	600 000	3,16
800	4,90	100 000	5,07	4 000	1,66	1 100 000	2,82
1000	4,94	150 000	5,99	5 000	1,76	1 600 000	2,63
1500	5,10	200 000	7,88	6 000	1,84	2 100 000	2,23
2000	5,11	300 000	8,25	7 000	1,92	2 600 000	2,08
3000	5,11	400 000	8,99	8 000	1,71	3 100 000	1,50
4000	5,12	500 000	8,80	10 000	1,82	3 600 000	1,47
5000	5,12	1 000 000	10,01	13 000	1,52	4 100 000	1,58

Źródło: opracowanie własne.

[4] Jest to warunek konieczny, jednak nie jest to w wielu sytuacjach proste, a czasami okazuje się niemożliwe.

[5] http://www.mathworks.com/products/parallel-computing/index.html.

rami. Dodatkową część czasu zajmuje zarządzanie pamięcią, która musi być niezależnie przydzielana dla każdego wątku obliczeń. Przyrost szybkości obliczeń jest więc zwykle mniejszy niż przyrost liczby procesorów. Nie musi tak być zawsze. Może się bowiem okazać, że te dodatkowe operacje pochłaniające nieco czasu trwają i tak krócej niż zarządzanie jednym, prostym, ale wielokrotnie powtarzanym procesem. Przykładem jest czas wyznaczania podstawowych statystyk (por. tab. 4), który skrócił się nawet 10-krotnie, mimo że obliczenia wykonywano na 6 procesorach. W dwóch przypadkach wzrostu szybkości nie zaobserwowano w ogóle. Dotyczy to najmniejszych macierzy, kiedy czas obsługi procesu wielowątkowego jest dłuższy niż samego przetwarzania danych.

Zastosowanie pętli *parfor* będzie zupełnie nieskuteczne, gdy obliczenia, które wykonujemy, nie są niezależne względem kolejności przebiegu pętli lub pojedyncze jej wykonanie wymaga bardzo małego nakładu obliczeniowego.

4. Obliczenia równoległe na GPU

Pętla *parfor* jest bardzo przyjaznym dla użytkownika rozwiązaniem. Nie wymaga zwykle większych zmian w kodzie programu. W wielu przypadkach pozwala na kilkukrotne przyspieszenie szybkości jego działania. Jeżeli jest to jednak zbyt mało, można skorzystać z innego procesora, który znajduje się w wielu współczesnych komputerach. Jest nim procesor graficzny – GPU. Jest to element urządzenia odpowiadającego za wyświetlania obrazu na ekranie monitora. Aby wyświetlać płynnie fotorealistyczne obrazy (np. w grach czy animacjach komputerowych), procesor ten musi posiadać wielką moc obliczeniową. Faktycznie wiele współczesnych procesorów graficznych posiada strukturę wielordzeniową, zawierając od 256 do nawet 2880 rdzeni. Procesory te nie są równie uniwersalne jak CPU, jednak podstawowe operacje matematyczne wykonują nieporównanie szybciej niż CPU.

Możliwość wykorzystania GPU do obliczeń numerycznych niezwiązanych z wyświetlaniem obrazu dostrzeżono już pod koniec wieku XX, jednak praktyczne możliwości pojawiły się, gdy jeden z producentów GPU, amerykańska firma NVidia, wprowadził jako standard do swoich produktów technologię CUDA (*Compute Unified Device Architecture*). Oprogramowanie CUDA to interfejs dla innych programów pozwalający wykorzystać potencjał setek procesorów graficznych w podobny sposób jak zwykłe CPU. W środowisku Matlab wystarczy zdefiniować zmienną jako typ *gpuArray,* aby system w sposób automatyczny obsługiwał dostępne na GPU procedury[6]. W tabeli 5 pokazano procedurę wyznaczania wartości własnych macierzy na GPU.

[6] W kolejnych aktualizacjach środowiska Matlab producent udostępnia coraz więcej funkcji matematycznych możliwych do wykorzystania na GPU. Ich pełną listę można znaleźć na stronie: http://www.mathworks.com/help/distcomp/run-built-in-functions-on-a-gpu.html.

Tabela 5. Procedura wyznaczania wartości własnych na GPU

```
Z1 = gpuArray(macierz_danych)     % utwórz zmienną GPU
Z2 = eig(Z1)                      % wyznacz wart. własne
gather(Z2)                        % przenieś wynik na CPU
```

Źródło: opracowanie własne.

Zastosowanie obliczeń GPU w analizie skupień może być bardzo efektywne. Wydajna karta graficzna pozwala skrócić czas obliczeń podstawowych statystyk dla dużych macierzy danych kilkadziesiąt do nawet kilkuset razy [Chang i in. 2008]. Wzrost szybkości obliczeń między GPU a CPU wektora odchyleń standardowych macierzy o rozmiarze 1 000 000 × 1 000 000 jednostek jest ponad 150-krotny (por. tab. 6). Im większa macierz danych, tym różnica między CPU a GPU większa. Załamanie następuje dopiero, gdy rozmiar macierzy danych przekracza rozmiar dostępnej dla GPU pamięci[7]. Dla innych badanych procedur[8] następuje dalszy wzrost szybkości obliczeń na GPU w stosunku do pojedynczego, jak i sześciu CPU.

Tabela 6. Przyrost szybkości analizy GPU/CPU wybranych procedur dla 1000 iteracji

Rozmiar macierzy [$n \times n$]	Macierz odległości	Rozmiar macierzy [$n \times n$]	Odch. standardowe	Rozmiar macierzy [$n \times n$]	Wartości własne
	1 CPU / 1 GPU		1 CPU / 1 GPU		1 CPU / 1 GPU
100	0,89	10 000	0,79	1 000	0,08
400	9,90	40 000	11,33	3 000	1,36
800	14,40	100 000	53,97	4 000	1,98
1000	21,64	150 000	77,21	5 000	2,31
2000	25,11	300 000	80,54	7 000	2,75
4000	25,12	500 000	112,33	10 000	3,17
5000	25,12	1 000 000	152,06	13 000	3,45

Źródło: opracowanie własne.

Jeszcze większe przyrosty szybkości obliczeń na GPU można uzyskać, zmieniając typ liczb zmiennoprzecinkowych na *single*[9]. Ten format umożliwia zapis liczb rzeczywistych z dokładnością do 6 cyfr znaczących. W wielu zastosowaniach

[7] Warto tu zauważyć, że nawet najlepsze karty graficzne mają do dyspozycji jedynie 3 do 6 GB pamięci.

[8] W tabeli 7 nie pokazano czasów obliczeń na GPU dla metody *k*-średnich, ponieważ w tym przypadku nie wystarczy zmienić typu zmiennej czy pojedynczego polecenia, a wymaga to zupełnego przebudowania algorytmu.

[9] Standardowo na CPU procedury numeryczne są wykonywane na liczbach zmiennoprzecinkowych typu *double*. Ten typ pozwala precyzyjnie zapisać liczby rzeczywiste z dokładnością do 14 cyfr znaczących. Procesory GPU obsługują bezpośrednio tylko typ *single,* a jego konwersja na typ *double* jest czasochłonna.

Tabela 7. Przyrost szybkości sumowania GPU/CPU dla typu danych *single*

Rozmiar macierzy [$n \times n$]	Suma		
	1 CPU / 6 CPU	1 CPU / 1 GPU	6 CPU / 1 GPU
1000	1,27	56,82	44,58
2000	3,35	109,32	32,63
3000	5,05	1105,69	219,08
4000	5,28	524,89	99,38

Źródło: opracowanie własne.

uproszczenie to będzie wystarczające. Jak pokazano w tab. 7, dla prostej operacji sumowania możliwe jest dla jednego GPU uzyskanie nawet tysiąckrotnego przyspieszenia obliczeń w stosunku do CPU. Nawet w stosunku do pętli parfor dla 6 CPU przyrost szybkości jest ponaddwustukrotny.

5. Zakończenie

Z przeprowadzonych analiz wynikają ważne wnioski dla analityków, w tym specjalistów zajmujących się analizą skupień. Istnieją już narzędzia i techniki pozwalające na znaczne zwiększenie szybkości analiz. W wielu przypadkach możliwe jest kilkukrotne przyspieszenie obliczeń bez znacznego nakładu pracy z zastosowaniem pętli typu *parfor*. Jest to rozwiązanie przyjazne dla analityka i w wielu przypadkach pozwala kilkukrotnie skrócić czas wykonywanych analiz. Warunkiem koniecznym do sukcesu jest tu niezależność wyników uzyskiwanych w kolejnych iteracjach stosowanego algorytmu.

Dalszego przyspieszenia, o znacznie większym potencjale, można dokonać, przenosząc ciężar obliczeń z CPU na GPU. Przyspieszenia rzędu 10x, 20x, a nawet 100x są w tym przypadku realnie możliwe do osiągnięcia. Warto także pamiętać, że kolejnego przyspieszenia można dokonać, zmieniając tam, gdzie jest to możliwe, format danych na *single*.

Przeprowadzone badania wskazują na znaczny potencjał współczesnego sprzętu i oprogramowania w analizie dużych i szybko się zmieniających danych. Potencjał ten może być także z powodzeniem wykorzystany w analizie skupień.

Literatura

Chang D.J., Jones N.A., Li D., Ouyang M., Ragade R.K., 2008, *Compute pairwise euclidean distances of data points with gpus*, Computational Biology and Bioinformatics, CBB '08. IASTED International Symposium, November, s. 278-283.

Kollios G., Gunopulos D., Koudas N., Berchtold, S., 2003, *Efficient biased sampling for approximate clustering and outlier detection in large data sets*, Knowledge and Data Engineering, IEEE Transactions on, vol. 15, no. 5, Sept.-Oct., s. 1170-1187.

Migdał-Najman K., Najman K., 2013, *Samouczące się sztuczne sieci neuronowe w grupowaniu i klasyfikacji danych. Teoria i zastosowania w ekonomii*, Wydawnictwo Uniwersytetu Gdańskiego. Gdańsk.

Wasif M.K., Narayanan P.J., 2011, *Scalable clustering using multiple GPUs*, High Performance Computing (HiPC),18th International Conference on, s.1-10.

Zechner M., Granitzer M., 2009, *Accelerating k-means on the graphics processor via CUDA*, First International Conference on Intensive Applications and Services, s. 7-15.

THE APPLICATION OF THE PARALLEL COMPUTING IN CLUSTER ANALYSIS

Summary: In the social research data sets which include thousands of variables are encountered more and more often. During the research of the group structure of units we encountered a lot of specific problems. One of them is the possibility of making grouping within acceptable time. This problem arises from two basic sources. One of them is the size and speed of expansion of the data set. The second, is the complexity of numerical algorithms, which are used in cluster analysis. The aim of this paper is to demonstrate the applicability of parallel computing in cluster analysis. The basic methods of multithreaded algorithms for determining the typical statistical measures and multithreaded algorithm of k-means in cluster analysis will be presented. The practical aspects of the applications will be shown on the basis of simulations in Matlab.

Keywords: algorithm acceleration, cluster analysis, parallel computing.