Marcin PIETROŃ, Maciej WIELGOSZ, Kazimierz WIATR
AKADEMIA GÓRNICZO-HUTNICZA, ACK – CYFRONET, ul. Nawojki 11, 30-950 Kraków
AKADEMIA GÓRNICZO-HUTNICZA, KATEDRA ELEKTRONIKI, Al. Mickiewicza, 30-059 Kraków

Implementacja oraz porównanie algorytmów tekstowych w środowiskach przetwarzania równoległego na przykładzie procesorów wielordzeniowych i kart graficznych

Dr inż. Marcin PIETROŃ

Ukończył studia na AGH (2003), wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki na kierunku Elektronika i Telekomunikacja oraz na kierunku Informatyka (2005). Obronił pracę doktorską na Wydziale Informatyki, Elektroniki i Telekomunikacji. Obecnie jest pracownikiem Akademickiego Centrum Komputerowego Cyfronet AGH. Jego zainteresowania naukowe dotyczą obliczeń i algorytmów równoległych oraz data-mining'u .

e-mail: pietron@agh.edu.pl

Dr inż. Maciej WIELGOSZ

Ukończył studia na AGH (2005), wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki na kierunku Elektronika i Telekomunikacja. Obronił pracę doktorska w 2010 roku. Obecnie jest pracownikiem Katedry Elektroniki AGH i bierze czynny udział w pracach badawczych realizowanych w zespole rekonfigurowalnych systemów obliczeniowych. Jego zainteresowania naukowe dotyczą sprzętowej akceleracji obliczeń, strumieniowej analizy treści oraz architektur sprzętowych dla algorytmów sztucznej inteligencji.

e-mail: wielgosz@agh.edu.pl

Prof. dr hab. inż. Kazimierz WIATR

Studia AGH Kraków (1980), dr nauk technicznych (1987), dr habilitowany (1999) i profesor (2002). Profesor zwyczajny na Akademii Górniczo-Hutniczej oraz Dyrektor Akademickiego Centrum Komputerowego Cyfronet AGH. Prowadzone prace badawcze dotyczą komputerowego sterowania procesami, systemów wizyjnych, systemów wieloprocesorowych, układów programowalnych, rekonfigurowalnych systemów obliczeniowych i sprzętowych metod akceleracji obliczeń.

e-mail: wiatr@agh.edu.pl

Streszczenie

Artykuł przedstawia implementację algorytmów tekstowych w wybranych platformach przetwarzania równoległego. Dostępność procesorów wielordzeniowych oraz kart graficznych ogólnego przeznaczenia sprawia, iż badania nad równoległą implementacją algorytmów w celu ich akceleracji nabierają coraz większego znaczenia. Algorytmy tekstowe są niezwykle istotnym i często niezbędnym elementem zaawansowanych algorytmów analizy tekstu oraz są także składowymi funkcji wyszukiwania wzorców w tekście wielu języków programowania. W pracy dokonano analizy najpopularniejszych algorytmów tekstowych oraz dokonano ich analizy pod kątem ich zrównoleglenia w celu ich implementacji w procesorze wielordzeniowym oraz karcie graficznej ogólnego przeznaczenia. Analizowanymi algorytmami są: boyer-moore, algorytm naiwny oraz algorytm knuth-morris-pratt. Następnie dokonano porównania efektywności ich realizacji na wymienionych platformach sprzętowych.

Słowa kluczowe: algorytmy tekstowe, GPGPU, obliczenia równoległe, text-mining.

Multicore and GPGPU implementation of chosen text algorithms

Abstract

This paper presents implementation of text algorithms in multicore CPU and GPGPU. The text algorithms are very common algorithms used in text analysis process and they are a part of functions used for text patterns recognition. The library functions for text searching implemented in many languages very often use most popular text-algorithms. The paper describes the analysis of these algorithms for parallel implementations in multicore processors and general purpose graphic cards. The research work presented in this paper shows that text algorithms can be partially parallelized. The process of acceleration can be done by appropriate dividing the input text between parallel threads (data parallelism). The comparative studies were performed for the following algorithms: boyer-moore (horspool) , naive and knuth-morris-pratt algorithm. The presented results show the efficiency of these algorithms in the case of different type and size of patterns. In the case of GPU the implementation was made in the CUDA framework. The OpenMP library was used for a multicore version.

Keywords: text algorithms, GPGPU, parallel computing, text-mining.

1. Wstęp

Dostępność akceleratorów sprzętowych i procesorów wielordzeniowych oraz środowisk ich programowania powoduje, że stają się one coraz bardziej popularne i powszechne. Stwarza to nowe możliwości realizacji wielu algorytmów. Odpowiednia analiza oraz implementacja algorytmów w tych jednostkach obliczeniowych umożliwia często znaczną akcelerację wykonania aplikacji.

Algorytmy tekstowe są algorytmami, które bardzo często wykorzystywane są w wielu zaawansowanych algorytmach analizy tekstu bądź też w funkcjach bibliotecznych realizujących wyszukiwanie wzorców w tekście. Najbardziej znanymi i powszechnie stosowanymi algorytmami tekstowymi są: algorytm naiwny (złożoność - O(mn)), algorytm KMP (Knuth-Morris-Pratt – O($m+n$)), Boyer-Moore'a (algorytm Horspool - O($m+n$)) [4, 6].

Analiza struktury oraz zależności między instrukcjami tych algorytmów pokazuje, że mogą one być częściowo zrównoleglone. Kolejnym zagadnieniem badawczym pozostaje wpływ rozmiaru przeszukiwanych danych na czas ich wykonania, skalowalność algorytmów oraz porównanie uzyskiwanej wydajności pomiędzy poszczególnymi platformami przetwarzania równoległego [1].

Badania zaprezentowane w pracy są częścią pracy badawczej autorów nad algorytmami z dziedziny text-mining i ich akceleracją [5]. Algorytmy tekstowe bardzo często są elementem bardziej zaawansowanych algorytmów analizy danych tekstowych.

2. Platformy obliczeń równoległych

2.1. Procesory wielordzeniowe

W pracy zastosowano równoległe jednostki obliczeniowe w postaci procesorów wielordzeniowych oraz kart graficznych ogólnego przeznaczenia. Od wielu lat możemy zaobserwować znaczny rozwój jednostek mocy obliczeniowej standardowych procesorów polegający na systematycznym zwiększaniu liczby rdzeni przy zachowanej na stałym poziomie częstotliwości taktowania układu. Od momentu pojawienia się procesorów dwu-rdzeniowych mamy do czynienia z ciągłym zwiększaniem mocy obliczeniowej procesorów poprzez stopniowe zwiększanie liczby rdzeni. Powstanie procesorów wielordzeniowych umożliwiło wprowadzenie technologii programowania równoległego na pojedynczym procesorze (OpenMP [3], Cilk++ [2]). Do najważniejszych modeli procesorów wielordzeniowych należy zaliczyć procesory zbudowanie w oparciu o architekturę QuadCore. To między innymi rodzina procesorów Intel Xeon, procesory i5 czy i7. Najnowocześniejszym i najwydajniejszym procesorem firmy Intel jest procesor

Intel Xeon Phi (maksymalna moc obliczeniowa przekraczająca 1 TFlop). Powstał on jako konkurencja dla kart graficznych. Pojedynczy koprocesor wyposażony jest w jednostkę obliczeniową z ponad 50 rdzeniami (najnowsza wersja posiada 64 rdzenie) oraz co najmniej 8 GB pamięci GDDR5. Wysoka wydajność obliczeniowa jest także zasługą wsparcia dla 512-bitowej architektury SIMD (przetwarzanie wielu danych za pomocą jednej instrukcji, jednostka SSE).

2.2. Karty graficzne ogólnego znaczenia

Od kilku lat obserwuje się znaczny wzrost zainteresowania kartami graficznymi jako akceleratorami obliczeń naukowo-inżynierskich. GPU specjalizuje się w intensywnych obliczeniowo, wysoce równoległych obliczeniach. Procesory graficzne posiadają więcej jednostek przeznaczonych do przetwarzania danych i obliczeń kosztem modułów odpowiedzialnych za sterowanie czyli m.in. buforowanie danych oraz kontrolę przepływu instrukcji. Budowa karty graficznej umożliwia wykonywanie programów zgodnie z architekturą SIMD (ang. *Single Instruction Multiple Data stream*). Taka architektura umożliwia wykonywanie tych samych instrukcji (programu) na różnych danych. W przypadku takiego modelu brak buforowania i wynikające z tego opóźnienia związane z dostępem do danych minimalizowane są dużą równoległością wykonywanych instrukcji. Podstawową jednostką kart graficznych jest multiprocesor strumieniowy SM (ang. *Stream Multiprocessor*). Każda karta w zależności od konkretnego modelu posiada swoją liczbę multiprocesorów strumieniowych. Multiprocesor tworzy, zarządza i wykonuje równolegle wątki oraz kontroluje dostarczanie danych. Mechanizm dostarczania danych do jednostek obliczeniowych pozwala na niezależne pobieranie danych i wykonywanie obliczeń. Należy podkreślić, że bariera (synchronizacja wątków) potrzebuje tylko jednej instrukcji procesora. Dzięki niskiemu kosztowi czasowemu bariery można znacznie rozdrabniać zadania na poszczególne wątki. Multiprocesor zarządza wątkami w grupach liczących po 32 wątki zwanych *warpami*. Wszystkie wątki wewnątrz *warpa* powinny wykonywać te same instrukcje. W sytuacji rozgałęzienia przy zastosowaniu instrukcji warunkowej, wszystkie wątki z *warpa* wykonują instrukcje z obu rozgałęzień, co zmniejsza efektywność implementowanego algorytmu. Wątki ponadto grupowane są w bloki ze wspólnym dostępem do pamięci dzielonej, natomiast bloki grupowane są w zbiór zwany *gridem.*

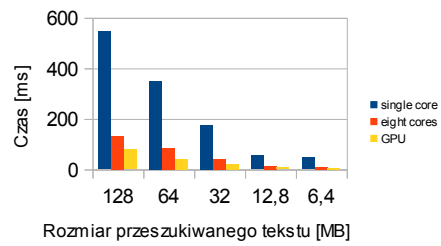
3. Implementacja algorytmów tekstowych na platformach przetwarzania równoległego

Analizowane algorytmy tekstowe składają się z dwóch pętli programowych. Pętla wewnętrzna to najczęściej pętla *while.* Jest ona odpowiedzialna za sprawdzenie występowania wzorca od określonego indeksu w badanym tekście. Iteracje tej pętli są ściśle od siebie zależne. Pętla zewnętrzna polega na wyznaczaniu kolejnych początkowych indeksów, od których analizowany tekst ma być sprawdzany pod kątem występowania wzorca (sprawdzenie wykonywane przez pętle wewnętrzną). W przypadku algorytmu naiwnego indeksy początkowe sprawdzania wzorca w tekście są zwiększane o jeden (nie występuje zależność między iteracjami pętli zewnętrznej). Natomiast w przypadku algorytmu KMP oraz algorytmów Boyer-Moore'a indeksy początkowe zależne są od poprzedniej iteracji. W przypadku algorytmu KMP indeks początkowy w danej iteracji zależy na jakim etapie (indeksie wzorca) zakończył się etap porównywania. Pozycja we wzorcu z poprzedniej iteracji poprzez odczyt z przygotowanej tabeli przesunięć pozwala ustalić indeks, od którego zacznie kolejna iteracja proces wykrywania wzorca. Natomiast iteracje pętli zewnętrznej w algorytmie Boyer-Moore'a zależą od znaku, który spowodował zakończenie proces porównywania w poprzedniej iteracji. Kod tego znaku stanowi indeks do specjalnie obliczonej tablicy, z której algorytm odczytuje wartość wymaganego przesunięcia w analizowanym tekście. Oba wyżej wymienione algorytmy ze względu na swoją strukturę (zależność iteracji w pętlach) nie dają się łatwo zrównoleglić. Jedyną metodą jest odpowiedni podział analizowanego tekstu wejściowego i wykonanie tych algorytmów równolegle na podzielonym tekście. Każdy wątek w przypadku równoległej realizacji otrzymuje tekst o długości: *text_length/N + pattern_length,* gdzie: *text_length* to długość analizowanego tekstu, *N* liczba wątków oraz *pattern_length* oznacza długość poszukiwanego wzorca. Jeden z wątków (na ogół ostatni) w celu zapewnienia warunków brzegowych dokonuje analizy na podzbiorze o rozmiarze: *text_length/N + text_length%(text_length/N)*.

W przypadku implementacji algorytmów w procesorze wielordzeniowym użyto środowiska OpenMP. W równoległej pętli for (omp parallel for) każdy wątek poprzez przekazanie odpowiednich wskaźników do współdzielonej pamięci przechowującej tekst wykonuje dany algorytm. Implementacja w kartach graficznych wymaga natomiast odpowiedniego rozdzielenia przetwarzanych danych w hierarchii pamięci karty graficznej oraz odpowiedniego mapowania części algorytmu na pojedyncze wątki. Przeszukiwany tekst przesyłany jest do pamięci globalnej karty graficznej jako parametr w funkcji jądra. Struktury pomocnicze przechowujące przesunięcia w danym algorytmie dla danego wzorca obliczane są na procesorze CPU przed wywołaniem algorytmu w karcie. Tablica pomocnicza oraz wzorzec kopiowane ze względu na rozmiar i charakter dostępu kopiowane są do pamięci współdzielonych poszczególnych bloków. Takie rozwiązanie zapewnia szybki dostęp do danych wszystkim uruchomionym wątkom na karcie. Przeszukiwany tekst dzielony jest równomiernie pomiędzy wątkami karty graficznej. Kopiowanie części analizowanego tekstu z pamięci globalnej do pamięci współdzielonej wykonują równolegle poszczególne wątki bloków. Każdy wątek odpowiedzialny jest za przesłanie *text_size/number_of_threads* bajtów pamięci, gdzie: *text_size* to rozmiar analizowanego tekstu, *number_of_threads* to liczba uruchomionych wątków w karcie (maksymalna liczba wątków w bloku dla kart Tesla m2090 wynosi 1024). Algorytm wykonywany poprzez poszczególne wątki realizowany jest na odpowiednio wyznaczonej części analizowanego tekstu (według podziału danych opisanego powyżej). Wskaźniki wyznaczające odpowiednie fragmenty tekstu na których ma być wykonany algorytm obliczane są za pomocą zmiennych reprezentujących indeksy wątków na karcie graficznej.

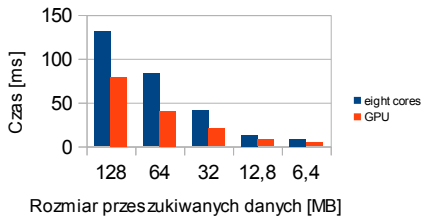
4. Wyniki implementacji

Rozdział ten przedstawia uzyskane wyniki implementacji przedstawionych algorytmów w procesorach wielo-rdzeniowych oraz kartach graficznych. Czasy realizacji algorytmów zostały zmierzone dla jednego rdzenia, 8-rdzeni oraz implementacji w karcie graficznej. Poszczególne wykresy przedstawiają wyniki dla różnych rozmiarów przeszukiwanego tekstu przy zachowanej stałej długości wzorca. Ponadto pokazane są wyniki dla innego rozmiaru wzorca, aby pokazać wpływ tego czynnika na skalę zmian efektywności algorytmów. Badania przeprowadzone zostały na karcie graficznej NVIDIA Tesla m2090 oraz procesorze Intel Xeon E5645.



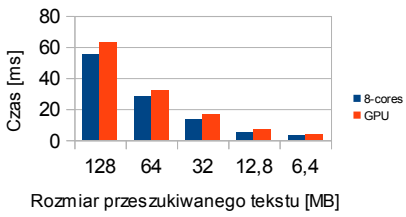
Rys. 1. Wyniki implementacji algorytmu naiwnego
Fig. 1. Results of the naïve algorithm implementation

W przypadku implementacji algorytmu naiwnego można zaobserwować dobrą skalowalność algorytmu (rys. 1 oraz rys. 2). Algorytm charakteryzuje równomierny dostęp do danych. Szczególnie znaczenie ma to, w przypadku kart graficznych, gdzie wątki odczytują dane w równych odstępach z pamięci. Dostęp ten ma charakter ciągły (*ang. aligned*).

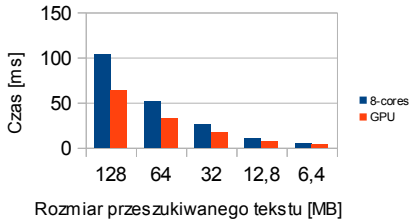


Rys. 2. Wyniki implementacji algorytmu naiwnego (GPU i multicore)
Fig. 2. Results of the naïve algorithm implementation (GPU and multicore)

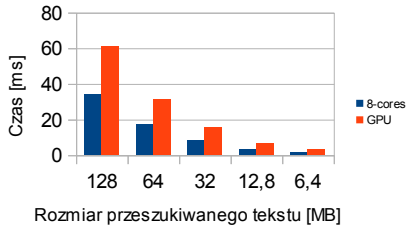
Rys. 3, rys. 4 oraz rys. 5 przedstawiają wyniki implementacji algorytmów KMP i Boyer-Moore'a. Można zaobserwować na nich, iż w przypadku algorytmu KMP dla analizowanego tekstu, który nie zawiera wiele powtarzających się podciągów wzorca (prefixów) implementacja w GPU jest znacznie bardziej efektywna niż w przypadku realizacji na 8 rdzeniach (rys. 4). W sytuacji częstszego występowania wzorca przewaga efektywności realizacji w GPU maleje (rys. 3) i algorytm w środowisku wielordzeniowym może uzyskiwać lepsze czasy, niż w karcie graficznej. Ta sama sytuacja występuje w przypadku algorytmu Boyer-Moore'a. Należy zauważyć jednak, iż różnice pomiędzy uzyskiwanymi czasami wykonania algorytmu w CPU i GPU są mniejsze, niż w przypadku algorytmu KMP. W sytuacji często powtarzających się elementów wzorca w analizowanym tekście algorytm Boyer-Moore'a w 8-rdzeniowym CPU jest zauważalnie szybszy od realizacji w GPU (rys. 5). Ponadto jak można zauważyć na przedstawionych wykresach implementacje w GPU są mniej czułe na charakter analizowanego tekstu. Niezależnie od relacji i występowania w nim elementów wzorca uzyskiwane czasy wykazują małą wariancję (przy stałym rozmiarze analizowanego tekstu).



Rys. 3. Wyniki algorytmu KMP z powtarzającymi się fragmentami wzorca
Fig. 3. Results of the KMP algorithm (with repeated parts of pattern in the analyzed text)

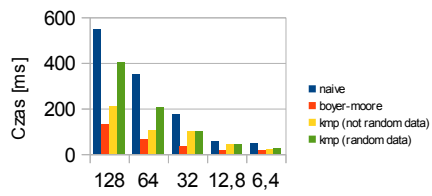


Rys. 4. Wyniki implementacji algorytmu KMP dla losowego tekstu
Fig. 4. Results of the KMP algorithm for a random text

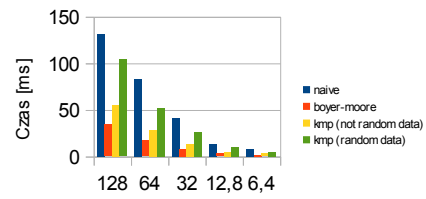


Rys. 5. Wyniki implementacji algorytmu Boyer-Moore'a
Fig. 5. Results of the Boyer-Moore algorithm

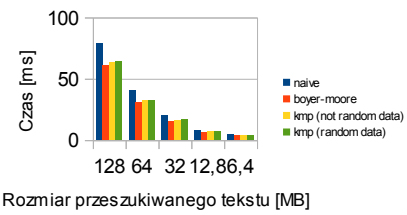
Rys. 6, rys. 7 i rys. 8 przedstawiają porównanie czasów wykonania algorytmów na poszczególnych platformach sprzętowych: jeden rdzeń CPU (rys. 6), 8-rdzeni CPU (rys. 7), karta graficzna (rys. 8). Różnice czasowe na rys. 6 wynikają ze złożoności obliczeniowej algorytmów. Rys. 7 pokazuje skalowalność algorytmów na wielu rdzeniach CPU. Na rys. 8 przedstawione są czasy działania badanych algorytmów w GPU. Różnice czasowe między poszczególnymi algorytmami są znacznie mniejsze niż w przypadku ich realizacji w CPU. Spowodowane to jest szybkim dostępem do pamięci typu współdzielonego. Drugim czynnikiem, który ma znaczący wpływ to charakter dostępu do pamięci. Im bardziej regularny i ciągły dostęp do danych tym mniejsze opóźnienia wynikające z dostępu do pamięci. Z tego powodu algorytm naiwny w GPU jest nieznacznie wolniejszy od dwóch pozostałych. Algorytmy Boyer-Moore'a oraz algorytm KMP dla małej długości wzorca (na rys. 8 wzorzec ma długość 10 bajtów) uzyskują w GPU zbliżone czasy.



Rys. 6. Wyniki algorytmów tekstowych na jednym rdzeniu
Fig. 6. Results of the text algorithms on a single core

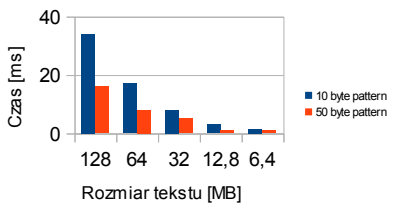


Rys. 7. Wyniki algorytmów tekstowych na 8 rdzeniach CPU
Fig. 7. Results of text algorithms on eight CPU cores

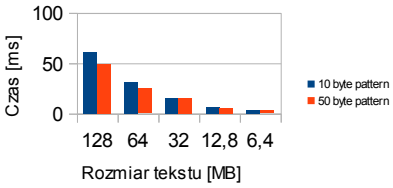


Rys. 8. Wyniki algorytmów tekstowych w GPU
Fig. 8. Results of text algorithms in a GPU

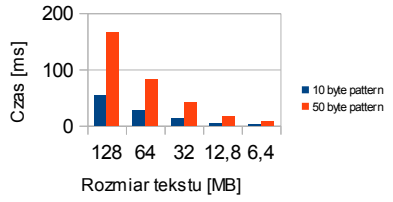
Rys. 9 - 12 przedstawiają wyniki wpływu zmieniającej się długości szukanego wzorca na czas działania algorytmów. Rys. 9 i rys. 10 pokazuje porównanie czasów działania algorytmu Boyer-Moore'a dla losowo wygenerowanego tekstu. W tym przypadku wraz ze wzrostem długości wzorca szybkość przeszukiwania tekstu rośnie (częstsze przesunięcia równe rozmiarowi wzorca). Natomiast rys. 11 oraz rys. 12 prezentują wyniki dla algorytmu KMP dla wysokiej częstotliwości pojawiania się w analizowanym tekście prefiksów wzorca. W tym przypadku zwiększanie długości wzorca znacznie zwiększa czas działania algorytmu. Karty graficzne są znacznie mniej wrażliwe na zmieniającą się długość szukanego wzorca niż standardowe jednostki CPU.



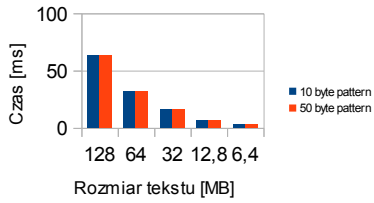
Rys. 9. Wyniki algorytmu Boyer-Moore'a dla różnych długości wzorca (8 - rdzeni)
Fig. 9. Results of the Boyer-Moore algorithm for different pattern length (8-cores)



Rys. 10. Wyniki algorytmu Boyer-Moore'a dla różnych długości wzorca (GPU)
Fig. 10. Results of the Boyer-Moore algorithm for different pattern length (GPU)



Rys. 11. Wyniki algorytmu KMP dla różnych długości wzorca (8-rdzeni)
Fig. 11. Results of the KMP algorithm for different pattern length (8-cores)



Rys. 12. Wyniki algorytmu KMP dla różnych długości wzorca (GPU)
Fig. 12. Results of the KMP algorithm for different pattern length (GPU)

5. Wnioski

Uzyskane wyniki implementacji algorytmów tekstowych pokazują, iż wykorzystanie ogólnodostępnych środowisk przetwarzania równoległego znacznie zwiększa efektywność wykonywania tych algorytmów. Ponadto przedstawione rezultaty prezentują skalowalność algorytmów na procesorach wielordzeniowych oraz kartach graficznych w zależności od charakterystyki danych wejściowych. Pozwala to na wybranie odpowiedniej platformy i algorytmu w zależności od analizowanych danych.

6. Literatura

[1] Kouzinopoulus C.S., Margaratis K.G.: String matching on a multicore GPU using CUDA, PCI'09, Informatics, pp. 14-16, September 2009.
[2] Intel, Intel Cilk++ SDK programmer's guide. http://software.intel.com/en-us/articles/intel-cilk/, 2009.
[3] OpenMP, Summary of OpenMP 3.0 C/C++ syntax. http://openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf ,2009.
[4] Cormen T., Leierson H., Rivest E.: Introduction to algorithms, MIT Press and McGraw-Hill, 1990.
[5] Wielgosz M., Jamro E., Russek P., Pietroń M., Żurek D., Janiszewski M., Wiatr K.: Implementation of algorithms for fast text search and files comparison , KU KDM 2013, Marzec, Zakopane, Proceedings, pp. 83 – 84.
[6] Sunday D.M.: A very fast substring search algorithm. Communications of the ACM, 33(8): 132-142, 1990.

otrzymano / received: 06.02.2014
przyjęto do druku / accepted: 01.04.2014 *artykuł recenzowany / revised paper*