AUTOMATYKA • 2007 • Tom 11 • Zeszyt 3

Marcin Pietroń*, Paweł Russek**, Kazimierz Wiatr**

# Metodyka sprzętowej akceleracji obliczeń w środowisku obliczeniowym komputerów dużej mocy

## 1. Wprowadzenie

Celem realizowanych przez autorów prac jest przyśpieszenie obliczeń naukowo-technicznych realizowanych w Akademickim Centrum Komputerowym „Cyfronet" AGH. Przyśpieszaniem objęte będą obliczenia realizowane przez pakiet *Gaussian* wykorzystywany do numerycznej symulacji właściwości cząsteczek chemicznych. Akceleracja polega na opracowaniu dedykowanej architektury sprzętowej: specjalizowanego koprocesora sprzętowego o architekturze przygotowanej do wydajnego realizowania operacji arytmetycznych specyficznych dla algorytmów realizowanych przez *Gaussian*. Platformą do realizacji tej architektury są układy reprogramowalne FPGA. Dzięki ich zastosowaniu możliwe jest zredukowania kosztu NRE (*non-recurring expense*), który tradycyjnie towarzyszy wszelkim opracowaniom sprzętowym. Praca opracowanego rozwiązania opierać się będzie na współpracy podsystemu programowego i podsystemu sprzętowego. Podział pracy w takim systemie polega na realizowaniu zadań o dużej złożoności wykonywanych instrukcji za pomocą uniwersalnej maszyny liczącej, a zadań, w których nad złożonością realizowanych instrukcji dominuje ilość danych wyjściowych, za pomocą specjalizowanej architektury sprzętowej. W praktyce polega to na wykonywaniu w realizowanym kodzie programu procesora odwołań do procedur przesyłających dane obliczeniowe do koprocesora specjalizowanego. Alternatywnym podejściem jest podniesienie mocy obliczeniowej centrum obliczeniowego poprzez zwiększanie liczby węzłów obliczeniowych z procesorami ogólnego zastosowania. Wymaga ono jednak poniesienia ogromnych kosztów i w praktyce stosowane jest jedynie przez bardzo nieliczne ośrodki. Realizacja wspomnianego koprocesora w technologii FPGA pozwoli na podniesienie mocy obliczeniowej systemu przy poniesieniu konkurencyjnych w stosunku do superkomputerów kosztów. Układy FPGA dają ponadto możliwość czerpania korzyści z wielokrotnego ich programowania. Możliwe jest zatem zastosowanie tej samej struktury krzemowej do przyśpieszania kilku następujących po sobie zadań poprzez ciągłą podmianę konfiguracji zgodnie z aktualnie wymaganą strukturą sprzętu.

* ACK „Cyfronet", Akademia Górniczo-Hutnicza w Krakowie
** ACK „Cyfronet", Katedra Elektroniki, Akademia Górniczo-Hutnicza w Krakowie

## 2. Pokrewne rozwiązania

O nośności podejmowanego tematu świadczy oferta firm z dziedziny HPC (*High Performance Computing*), które w swojej ofercie umieszczają moduły obliczeniowe wyposażone w układy FPGA. Są to na przykład rozwiązania takich firm jak Cray Inc. (http://www.cray.com) czy SGI Inc. (http://www.sgi.com). W wielu zastosowaniach z zakresu HPC układy FPGA potwierdziły już swoją przydatność. Na przykład firma SGI oferuje sprzętową implementację biblioteki BLAST [1] stosowanej w obliczeniach z zakresu biologii molekularnej i genetyki. Również w obliczeniach wielkiej skali z zakresu geologii (poszukiwanie nafty i gazu), sejsmografii czy elektrodynamiki układy FPGA są już stosowane [4]. Autorzy są przekonani, że nie ma również przeszkód, aby zastosować te układy rekonfigurowalne FPGA do obliczeń kwantowo-chemicznych, których główną specyfiką jest realizowanie obliczeń w oparciu o reprezentację zmiennoprzecinkową podwójnej precyzji. Takie badania były podejmowane również wcześniej, ale w niewielkim zakresie i dotyczyły jedynie pojedynczej precyzji [6].

## 3. Architektura sprzętowa

Kluczowym zagadnieniem w sprzętowej akceleracji obliczeń przy użyciu dedykowanego koprocesora jest zapewnienie wydajnego kanału przesyłu danych pomiędzy pamięcią główną systemu komputerowego, a pamięcią lokalna koprocesora sprzętowego. W wielu rozwiązaniach akcelerator sprzętowy dla systemu komputerowego dołączany jest jako karta rozszerzeń wejścia-wyjścia. Karta taka może być umieszczona na przykład na magistrali PCI, czy w nowszych rozwiązaniach na magistrali PCI-X. W praktyce okazuje się, że takie rozwiązania rzadko dają dobre rezultaty. W przypadku, kiedy mamy do czynienia z systemem wyposażonym w procesor o bardzo szybkiej magistrali systemowej, okazuje się, że realizacja obliczeń w układach FPGA przestaje być atrakcyjna, jeżeli do czasu obliczeń dodamy czas potrzebny na przesłanie danych obliczeniowych z pamięci głównej do pamięci umieszczonej na karcie z układem FPGA. Nawet procesor wykonujący obliczenia wolniej od dedykowanego koprocesora FPGA, dysponując szybkim interfejsem do pamięci, nadrabia czas obliczeń szybciej wymieniając dane pomiędzy rejestrami wewnętrznymi i pamięcią. Między innymi dlatego w technologii FPGA sięgnięto do rozwiązania, w którym procesor znajduje się na jednym półprzewodniku z logiką rekonfigurowana, tak, aby, wymiana danych pomiędzy procesorem a koprocesorem FPGA nie konsumowała czasu, czyniąc koprocesor nieprzydatnym. Takie rozwiązanie w środowisku komputerów dużej mocy nie jest oczywiście możliwe. Innym podejściem, stosowanym w systemach wielkich mocy obliczeniowych, jest mocne zintegrowanie koprocesora rekonfigurowalnego z magistralą systemowa procesora (np.: Cray XD1).

Jeszcze inne rozwiązanie zaproponowała firma SGI w swoich systemach Altix. System Altix to system typu SMP (*Simultaneous Multi Processing*), w którym poszczególne procesory dysponując swoją pamięcią lokalną w węźle maja możliwość szybkiego dostępu do pamięci pozostałych procesorów za pośrednictwem wydajnego łącza NUMALink.

NUMALink pozwala na transfer danych o przepustowości 6,4GB/s. Za pośrednictwem tego łącza do systemu Altix może być dołączony moduł z logiką rekonfigurowalną: RASC (*Reconfigurable Application Specific Computing*). Moduł RASC wyposażony jest w dwa układy FPGA serii Virtex4LX200 (http://www.sgi.com). Oferując każdy ok. 200 tysięcy makrokomórek logicznych, 96 bloków dedykowanych typu DSP48 oraz 64kb dwuportowej pamięci RAM, dają wystarczającą ilość zasobów do tego, aby zaimplementować funkcje i operatory zmiennoprzecinkowe podwójnej precyzji. Ponadto na karcie RASC znajduje się 80 MB pamięci QDR RAM stanowiącej dla modułów FPGA pamięć drugiego poziomu (pamięć pierwszego poziomu to wspomniana wcześniej pamięć w strukturze FPGA). Wymiana danych pomiędzy FPGA i pamięcią QDR odbywa się poprzez dwa kanały 128-bitowe pracujące z częstotliwością 200 MHz.

## 4. Profiling

Środkiem, jaki zastosowano w celu przyspieszenia aplikacji *Gaussian* jest poszukiwanie tej części kodu, który pochłania największą ilość czasu w trakcie obliczeń kwantowochemicznych. Jest powszechnie uznanym faktem, że 90% czasu aplikacja spędza wykonując 10% kodu programu. Ze względu na to, że kod *Gaussiana* (http://www.gaussian.com) jest bardzo skomplikowany zadanie to nie okazało się proste i w trakcie realizacji pojawiło się wiele problemów.

### 4.1. Profilery standardowe

Początkowo planowane było użycie standardowych profilerów. Między innymi miały to byc *pfmon* i *gprof* (http://www.gnu.org/software/binutils/manual/gprof/-2.9.1/gprof.html). Pierwszy z nich używany jest w trakcie pracy samego programu bez konieczności rekompilacji jego kodu, co niesie za sobą pewne wady i zalety. Zaletą jest to, że nie trzeba modyfikować długiej i skomplikowanej procedury kompilacji *Gaussiana*. Natomiast zasadniczą wadą jest to, że brak modyfikacji profilowanego kodu (profiler nie ma wpływu na proces kompilacji) może być przyczyną niedokładności profilingu i nie dawać poprawnych rezultatów. Niestety skomplikowany kod i architektura *Gaussiana* uniemożliwiły temu profilerowi uzyskanie miarodajnych wyników. Praktycznie także, okazał się niemożliwy pomiar czasu realizacji funkcji uruchamianych w bibliotekach dynamicznych. Przede wszystkim z tego powodu z otrzymanych wyników nie udało się wychwycić interesujących nas danych. Dlatego został użyty kolejny profiler: *gprof*. *Gprof* w przeciwieństwie do poprzedniego profilera przygotowuje dane wspomagające profilowanie już na etapie kompilacji. Jednak wyniki użycia *gprofa* także nie były zadowalające. Kompilacja z *gprofem* nie dała spodziewanych rezultatów. Głównym tutaj problemem stało się uzyskanie kodu wynikowego kompatybilnego z *gprofem*, z którego *gprof* mógłby stworzyć dane do profilowania. Złożoność pakietu *Gaussian* uniemożliwiała poprawną kompilacje z dołączonym kodem *gprofa*.

### 4.2. Narzędzie do profilowania wybranych części aplikacji

Po niepowodzeniach z zastosowaniem standardowych profilerów postanowiono stworzyć własne środowisko do profilowania i monitorowania obliczeń w *Gaussianie*. W pierwszym etapie realizacji narzędzia do automatycznego monitoringu wprowadziliśmy proste wstawki do kodu przy interesujących nas funkcjach. Następnie przeprowadzaliśmy kompilację całej aplikacji *Gaussian* w celu przetestowania czy wprowadzone zmiany działają poprawnie. Po pozytywnych wynikach przeprowadzonych testów zaczęliśmy rozwijać narzędzie do automatycznego testowania. Pierwszym krokiem było odpowiednie sprecyzowanie wymagań dla stworzonego środowiska tak, aby móc później przeprowadzić implementację wybranych części aplikacji *Gaussian* w FPGA. Zdecydowano, że musi ono zapewniać:

1) sparametryzowanie monitoringu funkcji (nazwa funkcji, lokacje),
2) graf wywołań (sparametryzowanie głębokości wywołań funkcji),
3) realizacja statystyk czasu wywołań funkcji,
4) monitorowanie danych wejściowych funkcji.

Te cztery wyżej wymienione wymagania stanowią główny cel realizowanego profilingu.

Najważniejszym z nich na wczesnym etapie konwersji *Gaussian* do sprzętu jest, jak już wcześniej zostało powiedziane, monitorowanie funkcji. Parametryzacja jego polega na podaniu nazwy funkcji, które chcemy monitorować oraz jej lokalizacji w przypadku wielu miejsc jej wywoływania. Kolejne wymaganie to graf wywołań funkcji, który daje nam informacje o tym, z której funkcji nadrzędnej dana funkcja jest wywoływana. Realizacja statystyk to funkcjonalność, która pozwala nam na uchwycenie wszystkich wykonywanych funkcji w trakcie zadań testowych oraz całkowite czasy ich wywołań. Ostatnim wymaganiem stawianemu naszemu narzędziu jest monitorowanie, jakie dane wejściowe i jaki jest ich zakres wprowadzany na wejście funkcji w trakcie obliczeń. Wszystkie opcje profilingu ustalane są przed kompilacją, a następnie wykonywana jest modyfikacja kodu. W dalszej części wykonywana jest kompilacja zmodyfikowanych przez narzędzie źródeł *Gaussiana*.

Językiem programowania, w którym tworzono nasze środowisko jest Perl, gdyż najlepiej spełnia wymagania do realizacji postawionych celów. Przede wszystkim chodzi tu o szybkie szukanie wzorców i zmiany w kodzie.

Aplikacja *Gaussian* napisana jest w języku Fortran 77. Język ten stwarza wiele problemów w trakcie tworzenia narzędzia. Przede wszystkim chodzi tutaj o etykietowanie, instrukcje skoku oraz pętle z etykietami. Powyższe czynniki znacząco wpływają na złożoność narzędzia, które implementujemy, aby zrealizować postawione cele profilowania. Wyniki, jakie uzyskano z profilowania standardowymi narzędziami, nie dały zadowalających rezultatów. Jedynie za pomocą opracowanego narzędzia wydzielono lokacje, gdzie interesujące nas funkcje występują i jak je odrębnie modyfikować i kompilować. Kolejnym faktem, jaki udało się zaobserwować jest to, że standardowe profilery w przypadku skomplikowanych i heterogenicznych aplikacji rzadko dają zadowalające rezultaty.

Na początku podjęto próbę profilowania funkcji bibliotecznych z *math library*. Zaczęto od funkcji eksponenty. Wybranie eksponenty podyktowane było tym, że po wstepnej

analizie obliczeń kwantowo-chemicznych stwierdzono, że funkcja eksponenty jest jedną z funkcji elementarnych najczęściej wywoływanych w przypadku rozwiązywania równania Hartree–Focka, które jest obliczane w większości typów obliczeń kwantowo-chemicznych. Z tego też powodu rozpoczęto sprzętową implementację funkcji eksponenty. Po wstępnej obserwacji zauważono, że większość wywołań funkcji eksponenty (oraz innych funkcji elementarnych) znajduje się w plikach bibliotecznych, które są statycznie linkowane z resztą modułów aplikacji *Gaussian*. Następnie dzięki narzędziu profilującemu uzyskano linie kodu dla eksponent, które wywołują się w kodzie najczęściej. Po wstępnej obserwacji można było zauważyć, że jest tylko kilka eksponent, które wykonują się zdecydowanie najczęściej. Na przykład liczba wszystkich wywołań eksponent w przypadku optymalizacji geometrii cząsteczki benzenu (złożoność cząsteczki ok. 40 atomów) wynosi kilkadziesiąt milionów.

Równolegle ze zlokalizowaniem eksponenty w kodzie *Gaussian* oraz zmierzeniem względnego czasu jej wywołania dla przykładowych obliczeń, w FPGA zrealizowany został sprzętowy moduł do jej obliczania. Dzięki sprzętowej implementacji uzyskano dwukrotne przyspieszenie czasu obliczania eksponenty [12, 13].

Kolejnym etapem było uruchomienie tak zrealizowanej eksponenty w *Gaussianie.* Jednym z problemów jaki się nasuwa jest narzut komunikacyjny związany z czasem przesyłania danych (argumentów) do modułu FPGA. Aktualnie prowadzone są prace nad tym, aby straty związane z komunikacją były jak najmniejsze. Pewnym rozwiązaniem może być to, aby wraz z obliczeniem eksponenty w FPGA realizowane były jeszcze dodatkowo inne obliczenia operujące na tych samych argumentach co eksponenta. Można to zapewnić analizując kod programu, który otacza eksponentę w *Gaussian*.

Kolejnymi funkcjami, które są intensywnie używane w pakiecie *Gaussian*, są funkcje bibliotek numerycznych BLAS i LAPACK. Podstawy teoretyczne na temat sposobu rozwiązywania równania Hartree–Focka oraz wstępna analiza kodu za pomocą stworzonego narzędzia pozwalają stwierdzić, że warto zbadać czy ich sprzętowa implementacja zwiększy szybkość wykonywanych obliczeń w badanym oprogramowaniu (rozdz. 5).

W najbliższym czasie planowana jest taka rozbudowa automatycznego narzędzia, aby możliwa była automatyczna analiza kodu źródłowego pod kątem implementacji w układach FPGA. Przede wszystkim chodzi o to, aby takie środowisko było parametryzowane przez użytkownika. Dzięki temu użytkownik będzie mógł sprecyzować, co chce dzięki takiej analizie i transformacji uzyskać. Głównym celem takiego środowiska będzie przedstawienie kodu źródłowego w postaci struktur danych, z których będzie można uzyskać efektywny kod VHDL. Wyniki wstępnych badań o tematyce dotyczącej transformacji aplikacji numerycznych do FPGA przedstawione są w cytowanej literaturze [3, 9].

## 5. Wyniki implementacji dla BLAS

Jak zostało wcześniej powiedziane, za pomocą zrealizowanego narzędzia można także wytypować algorytmy algebry macierzy jako często wykonywane i takiej, których przyspieszenie może być zauważalne w czasie realizacji obliczeń w programie *Gaussian*. Ope-

racje, o których mowa są realizowane w pakiecie przy wykorzystaniu standartowej biblioteki LAPACK. Realizacja sprzętowa funkcji z tego pakietu wydaje sie zbyt złożona, aby była opłacalna. Duża ilość funkcji w bibliotece LAPACK powodowałaby bardzo długi czas realizacji takiej implementacji sprzętowej. Ponadto im bardziej złożony jest algorytm tym mniej wydajna jego realizacja sprzętowa. Mówimy o algorytmach zdominowanych przez dane i algorytmach zdominowanych przez instrukcje. O ile te pierwsze świetnie nadają się do realizacji sprzętowych, to implementacji tych drugich w ogóle się nie podejmuje pozostawiając ich sekwencyjną realizacje procesorom. Struktura biblioteki LAPACK bazuje na funkcjach z podrzędnej biblioteki BLAS [1]. Co prawda LAPACK zawiera złożone procedury, ale rozwiązuje je głównie opierając się na funkcjach z pakietu BLAS. Podstawową operacją w pakiecie BLAS jest mnożenie macierzy realizowane w funkcji GEMM. Funkcja GEMM jako funkcja podstawowa dla wielu algorytmów algebry macierzy wyższego poziomu jest w każdym systemie obliczeniowym poddana bardzo uważnemu kodowaniu, tak, aby osiągnąć przy jej wykonywaniu jak największą moc obliczeniową. Faktycznie, w praktycznych systemach realizacja funkcji mnożenia macierzy odbywa sie z mocą obliczeniową dochodzącą do 90% mocy teoretycznej wydajności systemu, podczas gdy dla normalnie realizowanych obliczeń faktycznie wykazywana moc obliczenia nie przekracza 50%. Takie ukierunkowanie obliczeń w superkomputerach na wykorzystanie podstawowych operacji typu GEMM daje możliwość przyspieszenia całych aplikacji metodą skupienia uwagi na przyspieszeniu pakietu BLAS. Zasoby obliczeniowe jednego układu Virtex umieszczonego w module RASC pozwalają na zaimplementowanie 24 zmiennoprzecinkowych operatorów podwójnej precyzji typu MAC (*Multiply and Accumulate*). Ponieważ układ Virtex4 oferuje bloki DSP48, które mogą być wykorzystane do realizacji układów mnożących istnieją dwa sposoby realizacji mnożarek: przy pomocy bloków DSP48 i uniwersalnych bloków logicznych oraz przy pomocy samych układów logicznych. Wykorzystanie zasobów DSP48 oczywiście zmniejsza ilość koniecznych zasobów uniwersalnych potrzebnych do realizacji mnożenia zmiennoprzecinkowego (tab. 1). W rezultacie w implementacji sprzętowej wykorzystujemy układy mnożące dwojakiego rodzaju. Zasoby obliczeniowe konieczne do realizacji dodawania zmiennoprzecinkowego podwójnej precyzji również przedstawiono w tabeli 1. Ponieważ do realizacji operacji MAC potrzeba jednego układu mnożącego i dodającego (plus dodatkowa logika sterująca) otrzymujemy w wyniku 24 układów MAC w jednym układzie FPGA: 6 układów MAC wykorzystujących moduły DSP48 i 18 układów zbudowanych z samej logiki uniwersalnej. Przy częstotliwości pracy RASC-a, która wynosi 200 MHz oznacza to moc obliczeniową 9,6 MFLOP. Porównując tę moc z mocą procesora Itanium2 1,5 MHz o mocy obliczeniowej 6 MFLOP, widzimy, że akceleracja obliczeń oparta na układach FPGA nawet w zastosowaniach obliczeń wielkiej skali może dać dobre rezultaty. Porównanie układów Virtex4 i procesorów Itanium2 jest uzasadnione o tyle, że oba typy układów realizowane są w tej samej technologii półprzewodnikowej 90 nm. Należy jeszcze zauważyc, że osiągnięta w FPGA moc obliczeniowa jest dostępna dla zegara 200 MHz wobec 1,5 GHz w Itanium. Może to mieć znaczenie, jeżeli moc rozpraszana przy realizowanych obliczeniach będzie istotnym parametrem.

**Tabela 1**
Zasoby sprzętowe elementów architektury mnożenia macierzy dla reprezentacji zmiennoprzecinkowej liczb o podwójnej precyzji: double_add –sumator; double_mul1 – układ mnożący wykorzystujący bloki DSP48; double_mul2 – układ mnożący zbudowany w oparciu o logikę ogólnego stosowania; double_MAC1 – moduł pomnóż-i-dodaj wykorzystujący DSP48; double_MAC2 – moduł pomnóż-i-dodaj zbudowany tylko z logiki uniwersalnej

| Element | Liczba Macrocel | Liczba DSP48 |
|---|---|---|
| double_add | 851 | 0 |
| double_mul1 | 559 | 16 |
| double_mul2 | 1335 | |
| double_MAC1 | 1954 | 16 |
| double_MAC2 | 2730 | |

## 6. Podsumowanie

Uzyskane wyniki pozwolą uzyskać odpowiedź na pytanie, którą część *Gaussian* warto implementować w FPGA oraz jakie będzie dzięki temu przyspieszenie obliczeń? Następnie planowane jest wydobycie obliczeń równoległych w aplikacji *Gaussian.* Konieczna będzie wiedza, jak te obliczenia są wykonywane. Do tej pory w celach testowych wykonywane były w *Gaussianie* głównie optymalizacje geometrii. W dalszym planie na uwadze mamy wykonanie także innego typu obliczeń kwantowo-chemicznych, aby dały nam one miarodajne wyniki i pozwoliły pokazać, co jest najbardziej czasochłonną częścią całej aplikacji. Należy podkreślić, że najlepsze rezultaty przyspieszenia w koprocesorze FPGA można osiągnąć przy seryjnej realizacji obliczeń jednego rodzaju. Implementacja sprzętowa mnożenia macierzy i innych wymienionych funkcji da najlepsze rezultaty, jeżeli w kodzie programu operacje te będą grupowane tak, aby odwoływać się do koprocesora FPGA dla większego pakietu danych wyjściowych (operacje typu SIMD). Spełnienie takiego założenie wymaga prac nad przekodowaniem programu *Gaussian*.

## Literatura

[1] Dongarra J.J., Jeremy Du Croz, Hammarling S., Duff I.S.: *A set of level 3 basic linear algebra subprograms.* ACM Trans. On Math. Soft (TOMS), vol. 16, March 1990
[2] Dou Y., Kuzmanov G.K., Vassiliadis, Gaidadijev G.N.: *64-bit Floating Point FPGA Matrix Multiplication.* FPGA'05, Monterey, California, February 2005
[3] Harrias T., Walke R., Kienhuis B., Deprettere E.: *Compilation from Matlab to Process Networks Realized in FPGA.* Kluwer Academic Publishers, April 2002
[4] Karanam R.K., Ravindran A., Mukherjee A., Gibas C., Wilkinson A.B.: *Accelerating Scentific Applications Using FPGAs*. XCell Journal, Third Quarter 2006
[5] Prasanna V.K., Zhuo L.: *High Performance Linear Algebra Operations on Reconfigurable Systems*. Proc. of SuperComputing 2005, Nov. 2005

[6] Prasanna V.K., Zhuo L: *Design Tradeoffs for BLAS Operations on Reconfigurable Hardware*. Proc. of the 2005 International Conference on Parallel Processing, Oslo, Norway, June 2005
[7] Russek P., Wiatr K: *The prospect of computing acceleration using reconfigurable logic technology in huge computational power systems*. Proc. of IFAC Workshop on Programable Devices and Embedded Systems, PDeS 2006 Brno, 14–16 Feb. 2006
[8] Russek P., Wiatr K.: *Perspektywa przyśpieszania obliczeń instalacjach o wielkich mocach obliczeniowych za pomocą układów logiki rekonfigurowalnej*. Półrocznik AGH Automatyka, t. 9, z. 3, 2005
[9] Underwood K., Hemmert S.: *Architectures and APIs: Assessing Requirements for Delivering FPGA Performance to Applications.* Sandia National Technologies, November 2006
[10] Wiatr K.: *Akceleracja obliczeń w systemach wizyjnych.* Warszawa, WNT 2003
[11] Wiatr K.: *Sprzętowe implementacje algorytmów przetwarzania obrazów w systemach wizyjnych czasu rzeczywistego.* Kraków, AGH 2002
[12] Wielgosz M., Jamro E., Wiatr K.: *Modułu obliczający funkcję eksponenty implementowanej w układach FPGA*. Pomiary Automatyka Kontrola, t. 7 bis, 2007
[13] Wielgosz M., Jamro E., Wiatr K.: *Implementacja w układach FPGA operacji eksponenty dla liczb w standardzie IEEE-754 o podwójnej precyzji.* KNWS maj 2007