Ernest Jamro*, Marcin Janiszewski**,
Krzysztof Machaczek**, Pawel Russek*, Kazimierz Wiatr*,
Maciej Wielgosz*

COMPUTATION ACCELERATION ON SGI RASC: FPGA BASED RECONFIGURABLE COMPUTING HARDWARE

In this paper a novel method of computation using FPGA technology is presented. In several cases this method provides a calculations speedup with respect to the General Purpose Processors (GPP). The main concept of this approach is based on such a design of computing hardware architecture to fit algorithm dataflow and best utilize well known computing techniques as pipelining and parallelism. Configurable hardware is used as a implementation platform for custom designed hardware. Paper will present implementation results of algorithms those are used in such areas as cryptography, data analysis and scientific computation. The other promising areas of new technology utilization will also be mentioned, bioinformatics for instance. Mentioned algorithms were designed, tested and implemented on SGI RASC platform. RASC module is a part of Cyfronet's SGI Altix 4700 SMP system. We will also present RASC modern architecture. In principle it consists of FPGA chips and very fast, 128-bit wide local memory. Design tools avaliable for designers will also be presented.

Keywords: *custom computing, single-purpose processors, FPGA, high performance computing, SGI RASC*

AKCELERACJA OBLICZEŃ NA PLATFORMIE SGI RASC: MODULE OBLICZEŃ ZA POMOCĄ LOGIKI REKONFIGUROWALNEJ

Autorzy prezentują nową metodę prowadzenia obliczeń wielkiej skali, opartą na układach FPGA. W szczególnych przypadkach jej zastosowanie prowadzi do skrócenia czasu obliczeń. Podstawą metody jest prowadzenie obliczeń za pomocą architektur obliczeniowych projektowanych dla danego algorytmu. Ponieważ architektura stworzona została specjalnie dla zadanego algorytmu, lepiej wykorzystuje możliwości równoległej i potokowej realizacji obliczeń. Jako platformę realizacji architektur dedykowanych zastosowano układy rekonfigurowalne. Artykuł prezentuje także wyniki zastosowania wspomnianej techniki w takich obszarach, jak kryptografia, analiza danych i obliczenia naukowe podwójnej precyzji. Wskazano również na inne dziedziny nauki, gdzie opisywana technika jest z powodzeniem stosowana

* ACC „Cyfronet" AGH, Dept. of Electronics AGH, University of Science and Technology, Krakow, Poland
** ACC „Cyfronet" AGH, University of Science and Technology, Krakow, Poland

(np.: bioinformatyka). Zrealizowane algorytmy były uruchomione i przetestowane na zainstalowanym w ACK Cyfronet AGH module SGI RASC, będącym częścią systemu SMP Altix 4700. Przedstawiono architekturę zastosowanego modułu RASC oraz narzędzia i metody projektowania dostępne dla programistów.

Słowa kluczowe: *sprzętowa akceleracja obliczeń, procesory dedykowane, FPGA, obliczenia wielkiej skali, SGI RASC*

1. Introduction

The importance of numerical calculation is unquestionable at science and engineering today. For example, numerical modeling replaced the major part of practical experiments in many disciplines. This is due to the cost and efficiency of such approach. It is better and cheaper to proceed natural experiment with simulation to anticipate natural objects features in the way of numerical analysis. That is true for chemistry, biology, physics, aerodynamics, construction, etc. Sometimes there is also a need for computer aided experimental data processing. When experiment results in enormous amount of information usage of computer is essential.

There are two main approaches how to achieve necessary computing performance and throughput. One is to build multi-processors installations which are able to offer almost 10^{15} of floating point instructions executed per second today. An example of such system is BlueGene/L installation in Lawrence Livermore National Laboratory in California offers 596 TFlops (10^{12}) of peak computing power (XI 2007). Another one is to combine several computing sites into a computing grids. Grid organizations like EGEE (Enabling Grids for E-SciencE) bring together computing capabilities of many computers using network infrastructure. Cost of building high computing power infrastructure is usually enormous. The maintenance of such systems is also expensive even if we consider only electric power costs. Regarding the power we must take both supply power and air conditioning costs into account.

Up to now the constant demand for higher computing capabilities was satisfied by progress in semiconductor technology. According to formulated in 60's Moore's Law a number of transistors on a single chip is doubled every 18 months. That was true since the first microprocessor appeared on the market but it seems that the Moore's Law rule is going to collapse in the near future. With 45 nm semiconductor technology, the size of transistors are closer and close to atom size (ca. 0.1 nm). Nobody proposed sub-atom switching device at the moment, so we will probably hit the wall in the near future and transistor resize technique will not be possible any more. On the other hand architectures of a today processors are still improved. There are a lot of improvements to perform software algortihms better. Today processor architecture is much more sophisticated than the architecture of first microprocessors but it still seems not to be optimal for executed tasks. Of course computer and processor designers know the typical algorithm bottleneck and try to avoid them but the true is that processors are optimized for the algorithms treated as a set of problems. From the perspective

of a single algorithm and their properties today's processors architecture is far from optimal. It is because for different algorithms optimal architecture is different. General Purpose Processor (GPP) is a single universal hardware machine designed to perform implemented instructions to complete any algorithm. It is the basic paradigm of GPP based computations.

2. Field Programmable Gate Arrays

Gates are the basic building blocks of any digital device. They are itself elementary digital devices but they are such simple that can only perform fundamental logical operations like: NOT, AND, OR. Gates are made of transistors. Any computing device such as adder, multiplier or register must be constructed using gates. Semiconductor devices like microprocessors are usually called full-custom devices. That means that the entire process of their design and fabrication starts from blank semiconductor wafer. This is very expensive way of device design as a lot of unique preparation steps must be performed to start chip fabrication. Each extra step costs of course. In fact, this fabrication preparation costs don't matter if the volume of produced semiconductor chips is high because it is shared among all sold devices. If someone anticipates average, not very high, final quantity of designed chip it is better to use another approach – so called semi-custom process. In semi-custom process some of the steps are common to the set of different silicon devices and so the total quantity of final devices that use the same process increase. In such an approach the number of fabrication preparation steps is reduced and also costs are reduced as well. There are many semi-custom technologies. One of them is called Mask Programmable Gate Arrays (MPGA). In MPGA technology silicon wafers go with gates already implemented but not connected. Such prepared intermediate product is used for final implementation. The necessary steps include only preparation of connections between gates. To create connections an appropriate mask must be designed. That makes the name of technology.

In the late 80's new interesting semiconductor technology appeared. It was an array of logic gates like in MPGA technology but in opposite, the connections were also already implemented. Technology was useful in different designs because the implemented connections were flexible (they could be freely configured). That was the birth of Field programmable Gate Arrays (FPGA) technology. Implemented connections consisted of both wires and programmable switches those could be set by electrical field applied to the device. Unlike in MPGA where process of connections configuration must be performed in semiconductor factory, analogous process in FPGA can be performed at customer/user location by a downloading of a proper configuration file. During configuration FPGA behaves like memory because data is stored inside their structure. Each bit of downloaded data sets appropriate switch to be 'on' or 'off'. According to that the design process is not only cheaper than in any other semiconductor technology but also much faster and easier. What more, single FPGA chip can be configured many times. When it is necessary at any time configuration can be

replaced by different one. Thanks to that feature single FPGA chip can acts as a set of different chips (not simultaneously of course).

3. Custom computing

Hardware acceleration of algorthms is a method for faster algorithm execution. Speedup is achieved by a proper adoption of a host platform architecture. If the architecture of computer is fixed the only way to accelerate algorithm execution is to write and compile a software code with target system structure in mind. We call this software acceleration of algorithm. For example, if the cache size is known, programmer should avoid to perform operations on a bigger than cache size data structures and if it is possible to split such structures into smaller pieces of data to process them separately. When the architecture of the processor or computer is modified for faster execution of distinguished instructions we call this technique hardware acceleration. For GPP hardware acceleration means processor or computer architecture optimization to allow for efficient execution of selected operations which are common bottlenecks in majority of algorithms. Multiply and accumulate (MAC) is an example of such a operation. It is broadly used in linear algebra, digital signal analysis, image processing, etc. For contemporary GPPs, thanks to hardware tailoring, MAC operation is performed with execution speed close to processors peak computing power. This in turn makes programmers to write their codes using MAC operations as frequent as possible. In this way hardware and software fit each other. The MAC operation is an example how dedicated processor architecture can support execution of selected operation.

Due to a big variety of algorithms, GPP can get limited profits out from hardware acceleration techniques. When the family of carried out problems is narrowed down the potential of presented techniques grows. Usually we consider algorithms related one another to find out as much common features as possible. Then we build custom hardware that fit those characteristics.

Hardware acceleration is beneficial in computing power increase and in saving energy, so what stops us from it wide universal use? The shortest answer for this question is: “Costs”. As it was mentioned before, digital system development process requires several steps. Those are: behavioral modeling, validation of model, structural design, project simulation, prototype construction, testing. Some or all of the steps are repeated until system fits all the requirements. If digital system technology platform is an integrated circuit, those stages are quite expensive. For example to get a real prototype of semiconductor chip, layer masks are necessary. In 65 nm process cost of a single set of masks is 1 Million dollars. All costs beard to produce the very first unit of the system are called Non-Returned-Expenso (NRE) costs. The NRE-cost of each digital system must be shared by a part of price of all sold items. That is why only devices expected to be sold in very high quantities can be considered to be offered by companies. This condition is fulfilled in a case of GPPs and GPs of course. Status of the considered algorithm must be special to aspire to hardware acceleration.

From the above argumentation, it is obvious that in case of custom specific algorithms the only way to use hardware acceleration method is to use low NRE technology. That is FPGA technology.

4. Reconfigurable computing

The most expensive design is for full-custom technology. Semi-custom technology is less expensive but the cheapest is FPGA technology. Unfortunately cheaper means also slower and less capacious. That must be kept in mind when HPC algorithms are moved to FPGA. Approximately, FPGA are 10 times slower than full custom processors and offer 3 times less resources. We can conclude that full-custom devices have 30 times functional advantage over FPGA. Despite that, as we shall see, in some cases FPGA can perform better even under such circumstances. The best architecture to drive profit form hardware acceleration is to build system provided with both GPP and FPGA. Such co-processor based solutions are well known but novelty of this particular proposal is based on ability of a co-processor to reconfigure its structure and to work as free defined hardware processor architecture. It acts according to downloaded at the moment configuration file (Fig. 1).

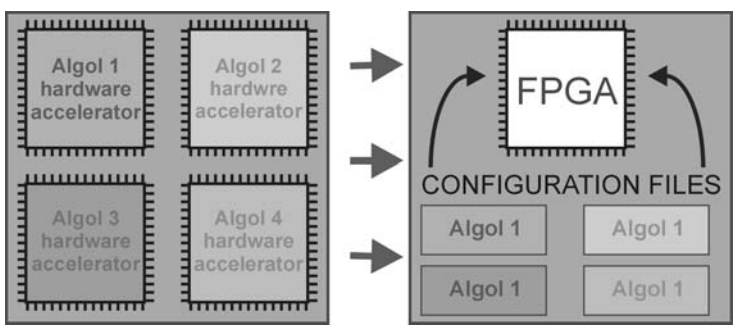

Fig. 1. Reconfigurable logic can replace several hardware accelerators

Capability for hardware acceleration comes from utilization of fine-grain parallelism. FPGAs can be used to build custom dataflow processors on an algorithm basis that would do away with the instruction fetch and decode overhead and a serial nature of the Von Neuman architecture. With this extra efficiency it is possible to achieve an increase of computational throughput.

The lack of high level programming ecosystem is a problem of reconfigurable computing. The GPPs have rich support of programming tools those have evolved over decades with effort of millions of individuals. The FPGA has adopted semiconductor technology from GPPs and also a lot of compilers technology can be adopted but there is still a lack of methods for full automation of design process.

5. Computing platforms

The most important issue when hardware acceleration platform is considered is fast and efficient data transfer between system operation memory and hardware accelerator or (depends on architecture) hardware accelerator cache memory.

In the first, PC based reconfigurable computing systems, FPGA accelerators were attached to the computer through peripheral bus like PCI or PCI-X bus for instance. Such solution has limitations those keep down good acceleration results. In such case, in opposite to reconfigurable accelerator, processor disposes of very fast local system bus with instant access to system memory. Hardware acceleration is not so attractive because to hardware accelerator processing time, data transfer delay from and to system memory must be added. Even if GPP performs slower than FPGA it can be recognized as better solution because of quick data transfer. In taught computing applications FPGA must be closer integrated with the rest of the computing system.

There is another approach implemented in HPC systems where FPGA is treated as integral system component rather than peripheral. It is linked directly to processor's resources through high speed connections and so overcomes the biggest bottleneck of FPGA co-processing. To provide maximum performance, FPGA co-processor has the same capabilities of operational memory access as GPP. For example, in Cray XD1 reconfigurable system, FPGA is integrated with processor system bus. Respectively, SGI Corp. proposed adequate solution in its Altix family systems. Altix is a family of SGI's SMP (Siymmetric Multi Processing) solutions. Its distinguishing feature is that each processor has both fast access local memory and slower access to local memories of other processors in the system. Data exchange between processors is achieved thanks to NUMALink bus. Thanks to NUMALink bus, data exchange between processors is relatively fast. The NUMALink interconnect is hierarchial system bus. It allows for global addressing and scalability of SMP system. Maximum NUMALink data transfer is 6,4 GB/s. The integral component of Altix system can be Reconfigurable Application Specific Computing (RASC) module. It is SGI's technology that enabling users to develop application specific hardware using reconfigurable logic elements. The SGI RASC is tightly integrated with NUMALink. From the hardware perspective FPGA is no longer co-processor mode in this model. With NUMALink FPGA has access to global shared memory and there is no need to load and unload data. The RASC is coupled with two Virtex4LX200 FPGA chips [15]. Each offers 200k of reconfigurable logic cells. Additionally there are two blocks of 64 MB QDR RAM memory. This memory acts like second level cache for FPGA. First level cache is implemented inside Virtex4LX200 structure and is called BlockRAM. Bidirectional data interface implemented for FPGA has 128-bit width and is clocked with frequency of 200 MHz.

All the following implementations, presented in this paper were executed on SGI RASC, which is installed in Academic Computing Centre „Cyfronet" AGH University of Science and Technology.

6. FPGA High Performance Computing

At present, the potential of FPGA as reconfigurable computing engines in HPC has been recognised. Recent advances in speed and density bring FPGA to supercomputing solutions. Also the progress in programming environments for reconfigurable computing is important in this process. Thanks to HLLs it is no longer necessary to be hardware engineer to get advantages out of reconfugurable computing.

There are also many tasks in typical computing job those do not make sense to be accelerated into FPGA fabric. Conventional procesors are highly efficient in many common computational problems. On the other hand there are many algorithms those do not fit well into GPP fixed structure. For applications that are highly parallel on fine grain level and spend much of their computational time in integer and fixed point calculations FPGA can be an attractive alternative. Reconfigurable computing friendly algorithms also rely primarily on local data. Such application stand to gain 10 times or more overall application performance with FPGA acceleration.

In biomedical applications (like DNA sequence alignment for example) that are extremely compute intensive algorithms are well suited to hardware acceleration. For example a Smith Waterman algorithm runs 50× speed of GPP on average size FPGA [6, 16]. The Smith Waterman algorithm is for comparing DNA and amino acid sequences against known genes and proteins to point the ideal candidate. That is extreme case example but here are some other:

- In seismic imaging seeking of patterns in sensor data can gain 17× acceleration [7].
- For simulation when performing vehicular traffic simulation, it is 300× faster on FPGA such like Xilinx VirtexII(XC2V6000) relative to 1,7 GHz Xeon.

Other areas are computational chemistry, encryption, automation, security, geology, finance applications, etc. Seismic processing and simulation incorporate significant amount of single precision floating point operation which is typically considered a no-go for FPGAs.

There is also a class of green computing solutions. The FPGAs demonstrates improved computing performance per watt, per dollar and cubic meter over traditional processors. Where we are considering over 100 W for high-end microprocessor, FPGA typically consume around 15 W when executing high performance algorithms [11]. We are able to reduce electricity costs, air conditioning costs and machine room floor space through reduced thermal density.

7. Example hardware implementations

For the evaluation purpose, SGI RASC platform was purchased and installed as a part of Altix4700 system. Several algorithm and functions were implemented. Here are brief description and implementation results achieved. These are intended as a reference for evaluation of expected speedup for another candidate HPC functions hardware implementation.

7.1. Pattern matching hardware acceleration

Bloom filter was realised in hardware for the pattern matching purpose [1]. Bloom filter is a method suitable for matching of a large set of binary or character patterns in input data. Input data is sequentially hashed and then computed hash is compared to the stored search pattern's hashes. In implemented hardware both hashing and matching process are executed in parallel.

Hashing is a compression process of 'w'-bit information into 'h'-bit information, where w>h. Usually it is realised by a process described by polynomial division. The goal is to achieve even distribution of the input data in the output data set. Traditionally hashing requires processing of one byts (character) at each clock cycle. In our solution, a hash-compare process is 16× multiplied to fit SGI RASC maximum data throughput and hardware interface width. The clock frequency of the solution is 100 MHz and 16 bytes of data are transferred at each clock cycle. There is 16 parallel Bloom filter structures to process 16 bytes at once. As a result hardware matcher can search through 16 GBytes of data per second. That means that in practice our matcher performance is limited my IO operation performance – if data is stored on hard disk for example.

The important limitation of Bloom filter algorithm is that match process is not deterministic. Every match reported by algorithm should be additionally confirmed by direct matching. Bloom filter works as selector that limits number of candidates for direct matching algorithms. The bigger 'h' is the Bloom algorithm is more reliable. In practice in hardware big 'h' value means a big memory necessary to store data. The other parameter that reflects in filter reliability is a number of searched patterns. If the size of the memory is small and there is a lot of positions marked in it, the number of wrong hits increases. In our implementation to limit resources utilization when a big number of patterns are searched we implemented up to six Bloom filters inside FPGA structure. The number of filters depend on the maximum pattern length. There are several implementations available which vary in number of pattern length and number of filters implemented. Example implementation results are presented in Table 1.

Table 1

A set of six bloom filters implementation result.
Pattern lengths: 128, 192, 192, 256, 256 bits;
128 bit data interface; 42-bit hash length

Used FPGA resources	78%
Used FPGA memory resources	92%
Data throughput	1,6 GByte/sec

7.2. Implementation of AES coder

Contemporary computing and telecommunication systems require data security. One of the safety measures is cryptography. The simplest way to code the data is to use

software program of the coder and decoder executed on microprocessor. In many application such a solution is not acceptable due to time constrains. The obvious solution is to implement cipher algorithm as hardware accelerator.

In our case, Advance Encoding System (AES) [12] was implemented in FPGA. The AES is a successor of an obsolete Data Encryption Standard (DES). The AES is based on Rijndael algorithm, which won the competition in 1997 for a new safer coding algorithm. Rijndael algorithm codes 128-bit, 192-bit or 256-bit size blocks of data. In our implementation basic version of algorithm was implemented where size of data block and cipher key is 128-bit. This is known as AES-128 algorithm. Rijndael is relatively very fast and very safe algorithm. Up to now, despite many ties nobody was able to crack the AES coded data. Before 128-bit block of data is coded it is organized as an 4×4 table of 8-bit elements. The same data representation is adopted for the key. These tables are sole data for complete data encryption process. The whole coding procedure consists of 9 rounds executed sequentially. All the round are the same and consist of basic operations like XOR, data shift, integer add and LUT (Look-Up Table) substitution. The only difference between rounds is that each of them use different cipher key. An appropriate set of nine keys is generated from single key in key extension procedure. Decoding of data is very similar to coding process and runs in an opposite direction. Detailed description of Rijndael algorithm can be found in references [4].

In our implementation, coding and decoding processes are organized as a pipeline process. In pipeline computation new data is applied to the processor while former data is still processed at the further computational stages. Thanks to pipeline computation, AES coder can accept a new data at each clock cycle. Beside acceleration due to pipeline execution, there is also parallel computation in our architecture. At each processing stage similar operations are executed in parallel for each element from a data table. When each stage of algorithm is completed the temporary results are latched by ragisters. At the next clock period that data is read by next processing stage. Because of pipelining and several execution stages the coded data results appear on coder output after 11 clock cycles The latency of designed module is 11 clock cycles. Timing results and logic resources of implemented AES processor are presented in Table 2.

Table 2
Implementation results of AES algorithm

AES Implementation	Coder	Decoder
FPGA logic resources	40%	40%
Data throughput	21 Gbit/seconds	16 Gbit/seconds

For the reference authors found AES-128 bit assembler implementation for Pentium4 3,2 GHz. The data throughput for that solution was 1,5 Gbit/s. Such we achieve speedup of 0,067 for coding and 0,1 for decoding process.

7.3. Montgomery multiplier

RSA is the most common cryptographic algorithm for establishing secured connection between two parties [13]. The key operation of RSA algorithm, involved in decryption and encryption process is modular exponentiation. Main operations involved in computing modular exponentiation are repeated modular multiplications and squaring. In our research we have implemented Montgomery modular multiplication [10] – the core operation involved in computing modular exponentiation.

Basic Montgomery algorithm consists of series of additions. Single addition, due to carry propagation is hard to parallelise. without additional logic. Since argument count thousands of bits special techniques must be used to make parallel execution possible. In our case problem was addressed by dividing long addition carry chains into parts, implementing mechanism allowing execution on all of them simultaneously. We have implemented our module in Virtex4 LX200 FPGA device. Results for 2048 bit Montgomery multiplier for different adder widths are shown in Table 3.

Table 3
Implementation results for 2048 bit Montgomery multiplier

Adder length [bits]	FPGA logic	Clock speed [MHz]
2	16%	200
3	15%	164
8	14%	165
16	14%	169

Analyzing speed of our module, theoretical speed of Exponentiation can be calculated. When using right to left binary algorithm and repeating modular multiplication appropriate number of times, modular exponent can be calculated and comparison between hardware and software implementations can be made (Table 4). Our fastest architecture (for 2 bit adder lengths) shows speedup only for 32 bit GPP (General Purpose Processors)

Table 4
Comparison of modular exponentiation execution time on three platform

Exponent size [bit]	Execution time		
	Athlon XP 2600+(32bit)	FPGA	Athlon 64 3500+(64bit)
1024	9,4 ms	5,5 ms	2,7 ms
2048	60 ms	27 ms	17 ms

Further speedup however can be achieved by using higher radices in computations. This means that instead of executing binary algorithm and analyzing one bit per cycle, larger numbers are analyzed. Such optimization allows consideration of applying FPGA to hardware acceleration of RSA cryptography.

7.4. Implementation of exponential function

Many hardware examples of single precision FP exponential function hardware implementations can be found contrary to efficient double precision standard one [3], which are unknown to the authors. This disproportion results from the fact that commonly known table-based or polynomial methods are not straightforward applicable to this double precision elementary function. Therefore some novel solutions were adopted to the proposed exp() calculation module not only to preserve compatibility to double precision standard but also to achieve high processing speed (200 MHz) and satisfying accuracy. The exp() module is fully pipelined (max. pipeline latency is 30 clks) [8]. The exp function is accelerated on SGI RASC [5] board with two Virtex-4 LX200 FPGA. The exp() function alone occupies less than 3% Virtex-4 LX200 FPGA. Exp() arguments are fetched to the FPGA's and results are sent back to processors over the system bus working at speed of NUMAlink 6,4 GB/s. The exponential module reaches the processing speed of 200 MHz, the external memory interface limits the number of operation to two exp() every clock cycle per a FPGA. The overall end-to-end algorithm execution speedup achieves 0,1 as compared to a sequential implementation of the algorithm executed on a single 1,5 GHz Intel Itanium2 microprocessor. If no seperate argument fetch is necessary for different modules, due to low resources consumption of the single exp() unit, up to 15 of the modules can fit in the Virtex 4 LX-200 which results in the huge speedup over the GPU implementation. For example quantum chemistry calculations involves several exp() operation what is regarded as the main advantage of FPGA [9]. It is worth mentioning that within the module calculations are conducted in the fixed-point standard. Only front and end interfaces of the module are IEEE-754 compatible. That's the reason of successful implementation and total speedup of 0,0167 for all 15 modules working together.

7.5. Implementation of GEMM function

There is a set mathematical functions widely utilized in scientific computations. The foundation of many calculations are linear algebra operations. To provide both software portability between platforms and ability to efficient software implementation of most common used functions, standard linear algebra libraries were defined. There are a few linear algebra libraries, like LAPACK, LINPACK, etc. Operations that are the part of linear algebra libraries like LAPACK and LINPACK seem to be too complex for hardware implementation. There are also too many functions to implement them all in hardware. Fortunately, all mentioned functions are higher level libraries and so perform operation harnessing Basic Linear Algebra Subroutines (BLAS): lowest level linear algebra library. BLAS defines three types of functions: vector-vector operations, matrix-vector operations and matrix-matrix operations. Those operations are matrix multiplications combined with transpositions and inversions. From the hardware implementation perspective the matrix-matrix [2] operations are the most promising because there is the best ratio of computations over data transfer. If we consider simple matrix multiplications each row of first matrix is multiplied with each column

from second matrix. So each column is transferred from the memory only ones and multiplied several times. There are also single precision and double precision floating point operations defined in BLAS. We implemented double precision matrix multiplications performed by GEMM function of BLAS [14]. The GEMM function is also the most frequently speed optimized function due to its wide usage in software. In each computer system it is coded very carefully to achieve best performance. Thanks to that when matrix multiplication is performed microprocessors achieve almost 90% of their peak computing power whereas the sustained system performance rarely achieves 50%. This make GEMM very eagerly used function. For hardware implementation, such big population of GEMM function in software is a good news. Thanks to that it is worth implementation effort.

Logical resources in Virtex4LX200 which is located on SGI RASC allow for implemantation of 24 double precision Multiply and Accumulate (MAC) operators. Beside general purpose logic, Virtex4 FPGA family offers also dedicated for digital signal processing blocks. So called DSP48 blocks. The DSP48 allows for efficient MAC implementation. Finally there are two types of MACs in matrix multiplication implementation: with and without DSP48 usage. Because MAC implementation requires one multiplier, one adder and additionally steering logic it is possible to fit 24 MACs in Virtex4LX200 (6 of them based on DSP48 blocks). Resources utilized for MAC implementation are presented in Table 5. Designed architecture is pipelined and it can perform 24 MAC operation at each clock cycle. Considering, that RASC clock is 200 MHz we can achieve 9,6 GFLOPs of computing power. As a reference we can quote the Ithanium2 1,5 GHz power which is 6 GFLOP. It must be highlighted that Itanium and Virtex4 are produced in the same 90 nm semiconductor technology.

In practice computing power of FPGA and Itanium for double precision matrix multiplication is the same. The number of MACs in Virtex had to be reduced. With almost 100% of logic resource utilization we couldn't fulfill appropriate timing constrains to perform with 200 MHz clock.

Table 5
Implementation results for GEMM function. Matrix multiplier: 24 MACS, 128 bit data interface

FPGA logic resources	50%
FPGA DSP48 resources	100%
Preformance	4,8 GFlops

It is worth notice, that implemented hardware regarded double precision floating point multiplication. If single precision would be considered we could overperform Itanium2 which calculates with the same speed despite data precision. In FPGA hardware implementation we could implement more MACs because single precision requires less logic resources. According our evaluation, in single precision case we could achieve 0,5 speedup. From the other hand there are dedicated processors like

Clearspeed CSX600 that offers 24 GFLOPS for double precision matrix multiplication. We would conclude that FPGA hardware accleration is not always the solution of best choice.

8. Conclusions

The article regards reconfigurable high performance computing. Both basic principles and example applications were presented in this paper.

Under some circumstances, reconfigurable computing which is custom hardware oriented computing implemented in reconfigurable logic can be attractive choice if higher computing power is necessary.

The reconfigurable custom hardware should be considered if there is no dedicated hardware in fixed-wired semiconductor technology. FPGA are configuration flexible but this feature makes them slower than any other non-reconfigurable hardware. There is also number of resources (logic gates) limitation in reconfigurable chips.

The FPGA accelerated algorithm shouldn't rely on data transfer. If big amount of data transfer is required over computation the advantages of reconfigurable logic utilization are limited. Before selected algorithm can be successfully accelerated by custom hardware, data representation should be also carefully considered. One should always use as little data bits for data representation as possible to get higher speedup. Also it should be tried to perform operations in fixed-point precision rather than in floating point for better results in FPGA.

Custom hardware speedup stems from pipelining and paralled execution so accelerated algorithms should be likely to be performed in such a manner. There are also some kind of operations where GPP are not effective at all. If logical operations on single bits had to be executed custom hardware is the best solution of choice.

Accelerated software should be characterized by definite computation kernel. In practice for the complete problem solution, usually we rely on hybrid systems where GPP is supported by FPGA. In accelerated applications only a few lines from computational kernels are moved from software to hardware. In practice only few lines converted to hardware can fit the FPGA capacity. If the kernel is meaningful in calculations, acceleration is more successful.

Reconfigurable computing is also a green, environment friendly computing. FPGA chips consume an order of magnitude less power. This allow for energy reduction in both supply and cooling electrical power.

References

[1] Bloom B. H.: *Space/time trade-offs in hash coding with allowable errors*. Commun. ACM, 13(7), pp. 422–426, 1970

[2] Dongarra J. J., Du Croz J., Hammarling S., Duff I. S.: *A set of level 3 basic linear algebra subprograms*. ACM Trans. Math. Softw., 16(1), pp. 1–17, 1990

[3] Doss C. C., Riley R. L. Jr.: *Fpga-based implementation of a robust ieee-754 exponential unit.* In FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 229–238, Washington, DC, USA, 2004. IEEE Computer Society

[4] Faria D. B., Cheriton D. R.: *Dos and authentication in wireless public access networks.* In WiSE '02: Proceedings of the 1st ACM workshop on wireless security, pp. 47–56, New York, NY, USA, 2002. ACM

[5] Silicon Graphics. *SgiR rasct rc100 blade, dramatic application speed-up with next generation reconfigurable compute technology.* `http://www.sgi.com`

[6] Harris B., Jacob A. C., Lancaster J. M., Buhler J., Chamberlain R. D.: *A banded smith-waterman fpga accelerator for mercury blastp.* International Conference on Field Programmable Logic and Applications, 2007, FPL 2007, pp. 765–769, 27–29 Aug. 2007

[7] He C., Lu M., Sun C.: *Accelerating seismic migration using fpga-based coprocessor platform.* In FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 207–216, Washington, DC, USA, 2004. IEEE Computer Society

[8] Jamro E., Wiatr K., Wielgosz M.: *Fpga implementation of 64-bit exponential function for hpc.* International Conference on Field Programmable Logic and Applications, 2007, FPL 2007, pp. 718–721, 27–29 Aug. 2007

[9] Wielgosz M., Piteron M., Jamro E., Russek P., Wiatr K.: *Two electron integrals calculation accelerated with double precision exp() hardware module.* Reconfigurable Systems Summer Institute, RSSI proceedings, July 2007

[10] Montgomery P. L.: *Modular multiplication without trivial division.* Mathematics of Computation, pp. 519–521, 1985

[11] Prasanna V. K.: *Energy-efficient computations on fpgas.* J. Supercomput., 32(2), pp. 139–162, 2005

[12] *Federal Information Processing.* Fips pub 197, advanced encryption standard (aes), November 2001

[13] Rivest R. L., Shamir A., Adelman L. M.: *A method for obtaining digital signatures and public-key cryptosystems.* Technical Report MIT/LCS/TM-82, 1977

[14] Wiatr K., Russek P.: *Dedicated architecture for double precision matrix multiplication in supercomputing environment.* IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, Cracow, April 2007

[15] Xilinx. *Virtex-4 User Guide.* `http://www.xilinx.com`, 2007

[16] Zhang P., Tan G., Gao G. R.: *Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform.* In HPRCTA '07: Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications, pp. 39–48, New York, NY, USA, 2007. ACM