HERA: A RECONFIGURABLE AND MIXED-MODE PARALLEL COMPUTING ENGINE ON PLATFORM FPGAS*

Xiaofang Wang and Sotirios G. Ziavras

Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
{xw23, ziavras}@njit.edu

ABSTRACT

The high price, long design and development cycles, programming difficulty and high maintenance cost of supercomputers limit their range of potential applications. Recent advances in Field-Programmable Gate Arrays (FPGAs) have made feasible the development of high-performance and programmable parallel systems on a programmable chip (PSOPC). PSOPC's yield high-performance at low cost for many parallel applications. We present in this paper the design and implementation of our HERA (HEterogeneous Reconfigurable Architecture) machine that employs FPGAs to allow the simultaneous execution of a variety of parallel processing modes, including SIMD (Single-Instruction, Multiple-Data), MIMD (Multiple-Instruction, Multiple-Data) and M-SIMD (Multiple-SIMD). The processing element is centered on a single-precision IEEE 754 floating-point unit (FPU) and employs a 7-stage pipeline. To demonstrate the robustness and viability of our approach, we propose a data partitioning scheme and employ mixed-mode scheduling for Cannon's matrix-matrix multiplication algorithm with matrices of arbitrary size and shape. Performance results on our 64-PE machine that employs a dual-FPGA system are better than the optimized performance on a dual-Xeon PC.

Key Words: Parallel processing, Reconfigurable architecture, SIMD/MIMD mixed-mode computing, Floating-point arithmetic, Matrix multiplication, Cannon's algorithm

1. Introduction

Proprietary supercomputer development has suffered in recent years after more than 20 years of tremendous investments [1-2]. Many concepts of parallelism are commonly present in modern microprocessors, such as memory hierarchy, superpipelining, multiple functional units, superscaling, out-of-order execution, branch prediction, SIMD processing, and speculative and predicated execution. One of the reasons for the decline of supercomputer development is the partial or complete dependence on custom chips, often making systems suffer from high prices, long turn-around times, prohibitively high upgrade costs and little flexibility. Many research efforts have shifted to COTS (commercial-off-the-shelf)-based platforms in recent years, such as symmetric multiprocessors (SMP) or clusters of PCs. However, these approaches do not often deliver the highest level of performance due to many inherent disadvantages of the underlying sequential platforms and "the divergence problem" [2]. In order to satisfy the greedy need of many scientific and engineering applications, whose time complexities increase at a faster rate than Moore's Law [3], for higher computation power, we need to explore new, sustainable, cost-effective, flexible and energy-efficient strategies.

With steady advances in silicon technologies and the promise of billion-transistor chips in just a few years time, single-chip parallel architectures seem to be gaining popularity. Recent research efforts in this direction include Hydra [4], SCMP [5], RAW [6], etc. The main driving force is the diminishing gain in performance by further exploring ILP (Instruction Level Parallelism) techniques in superscalar microprocessors and the frequently limited amount of ILP in programs [7]. Another important reason is the reverse scaling of wires compared to transistors in deep-micron processes. As a result, a major shift from ILP to TLP (Thread Level Parallelism) is undergoing in both the industry and research communities. However, these approaches also suffer from the same problem as supercomputers: a high volume is required to amortize the high development and NRE costs. The ever-shortening product cycles, and the high design complexity of these solutions also limit their applicability.

The recent advent of multi-million gate FPGAs having richer embedded feature sets, such as plenty of on-chip memory, DSP blocks and embedded hardware microprocessor IP cores, facilitates high performance, low power consumption and high density. These traits make feasible the implementation of *Parallel Systems on a Programmable Chip* (PSOPCs) at affordable costs. FPGAs provide a great opportunity to system designers to combine the high-performance of ASIC devices with the programming flexibility of microprocessors. More

*This work was supported in part by the U.S. Department of Energy under grant DE-FG02-03CH11171.

importantly, by using FPGAs we can customize the computing hardware appropriately at runtime to match the characteristics of the parallel algorithms. Although previous FPGA-based custom computing machines had demonstrated considerable performance gains at reduced costs over general-purpose microprocessors for many computation-intensive applications [8-10], the processing capability of such systems is often limited. For example, in most scientific and engineering applications, floating-point representation is frequently required in order to deal with large dynamic data ranges. Because floating-point units consumed a large proportion of resources in earlier FPGAs, very few reconfigurable machines supported floating-point arithmetic [8-10]. Although it has been shown recently that the peak floating-point performance of FPGAs has outnumbered in the last 1-2 years that of modern microprocessors and is growing much faster than the latter [11], we have not seen yet any programmable parallel system based on FPGAs that supports floating-point operations; the only exception is our previous research efforts on parallel LU factorization on a configurable multiprocessor [12].

In this paper, we present the design and implementation of our HERA machine which is based on Xilinx platform FPGAs. HERA is a new, mixed-mode heterogeneous parallel computing system supporting floating-point arithmetic. Every PE in our machine can be reconfigured dynamically at the instruction level to operate in SIMD or MIMD. Thus, the whole system can support a variety of independent or cooperating computation modes, such as SIMD, MIMD and M-SIMD, to better match subtask characteristics in a single application. An important motivation for mixed-mode computing [13] is that a typical application may have several subtasks that require different computation modes. For a fixed machine configuration, the performance on most subtasks is not normally optimal because of the architecture's expected unsuitability. The low communication overhead and low cost of our approach add much needed promise to the high-performance computing field. Our machine has a standard RISC instruction set that can be applied to many problems.

Matrix-matrix multiplication (MMM) is a key operation in many fundamental algorithms. Its high computation complexity, $O(N^3)$, where $N \times N$ is the matrix size, has attracted many research efforts. The reduction of this complexity can also benefit many applications involving the solution of systems of linear equations, the triangular decomposition of matrices, matrix inversion, the computation of determinants and watermarking in image processing. Cannon's MMM algorithm [14] is a memory efficient parallel implementation for torus-connected processor arrays, where each processor communicates directly with its four neighbors immediately to the north, east, west and south (NEWS connections). The memory efficiency of Cannon's algorithm is advantageous to FPGA-based designs because of their limited on-chip memory resources. The original algorithm by Cannon assumes that the input matrices and the partitioned matrix blocks are all square. In this paper, we show that by employing an innovative data partitioning scheme and also mixed-mode scheduling for our HERA machine, Cannon's algorithm can be efficiently applied to matrices of arbitrary shape and size.

2. HERA: A Reconfigurable Mixed-Mode Parallel Computer

SIMD and MIMD are distinct computation modes in parallel processing. SIMD machines consist of an array of identical PEs which are capable of performing simultaneously the same operation on different sets of data, under the control and coordination of a global control unit. Low inter-processor communication and synchronization overheads make it superior to MIMD in performing data-parallel algorithms. On the other hand, MIMD is preferable for medium- to large-grain parallelism, where multiple processes or routines are implemented in parallel.

2.1 System Organization

Fig. 1 shows the general diagram of our HERA machine with m x n PEs interconnected via a 2-D mesh network. The 2-D mesh topology is ideal for our target matrix-based applications and is also scalable. The computing fabric is controlled by the system Sequencer that communicates with the host processor via the PCI bus. Interrupt logic between the Sequencer and the host processor is implemented. The Global Control Unit (GCU), included in the system Sequencer, fetches instructions from the global program memory (GPM) for PEs operating in SIMD.

The total number of PEs is determined by the available resources in target FPGA devices and the resource requirements of the application. Due to the presence of FPGAs, besides that the system operating mode is reconfigurable at runtime, the capabilities of each PE and the number of PEs can be easily reconfigured based on the application requirements. The host can load different FPGA images in the same C code to finish different subtasks at runtime. Thus, FPGAs provide another dimension of flexibility toward optimizing the hardware to match the specific characteristics of the applications.

We employed fast, direct NEWS (North, East, West and South) connections for communications between nearest neighbors. Nearest PE pairs on the same row or column can also communicate through one port of the data memory of the PEs to the west and north. Since every PE also has a Local Control Unit (LCU), the decoding of instructions is carried out by the LCU. By giving the decoding work to the LCUs, we avoid broadcasting a large number of control signals to all the PEs. We still need global communication. Every column has a Cbus and all the Cbuses are connected to the Column Bus.

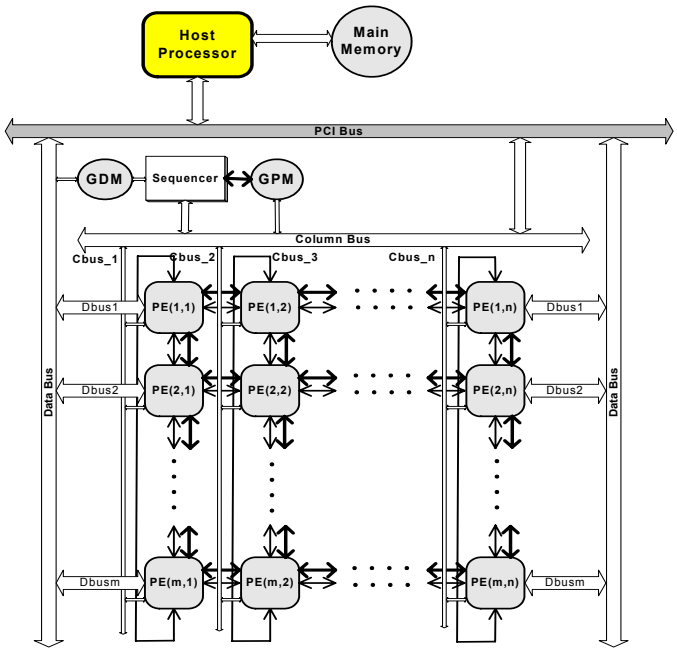

Fig. 1. HERA system architecture

2.2 PE Architecture

We followed a RISC load-store architecture for our PE to save hardware resources. Fig. 2 shows the block diagram of the PE. All data paths are 32 bits. The PE contains several major components: a 7-stage, pipelined, 32-bit floating-point function unit (FFU), an LCU, 32-bit dual-ported local program memory (LPM), 32-bit dual-ported local data memory (LDM) and eight NEWS communication ports. The sizes of LPM and LDM were determined by the number of memory blocks in the FPGA device. They are configured as dual-ported 32-bit memories. The A port of LDM is accessed by the local PE, and the B port is shared with the neighbors to the south and east. A PE can directly write to or read from the LDMs of its west and north neighbors via their B ports. This feature facilitates large block data transfer and data I/O. Data in one of the NEW_IN registers can be sent to any of the four neighboring NEWS_OUT registers by using one instruction.

Our HERA design implements IEEE 754 single-precision pipelined floating-point operations in each PE. We employed a 3-stage pipeline in the floating-point adder, subtractor and multiplier and a 28-stage pipeline in the floating-point divider. The frequencies after place-and-route for a Xilinx VirtexII FPGA XC2V6000 are 128.3MHz (add/sub), 150.8MHz (mul) and 165.4MHz (div). Since we focused on prototyping the concept to prove the viability of our mixed-mode design approach, we did not spend much time in this first phase to improve the performance of our FPU.

Every PE in the processor array has an ID number. The last 7 bits in all instructions select a particular PE or a group of PEs. Every PE holds a mask bit and computes the masking value with every instruction. The destination register of *Get_N/E/W/S* instructions or the source register of *Send_N/E/W/S* instructions can also be one of the four NEWS_OUT registers. This way, data can bypass a PE to reach the next PE because we can use shared NEWS registers between PE pairs. Each PE comprises 32 32-bit general-purpose registers (GPRs) and several system registers: local instruction register (LIR), local program counter (LPC), data memory address register (DMAR), program memory address register (PMAR), local status register (LSR), local masking register (LMR) and 1-bit operating mode register (OMR). Similar to some other RISC processors, the R0 GPR is fixed at zero.

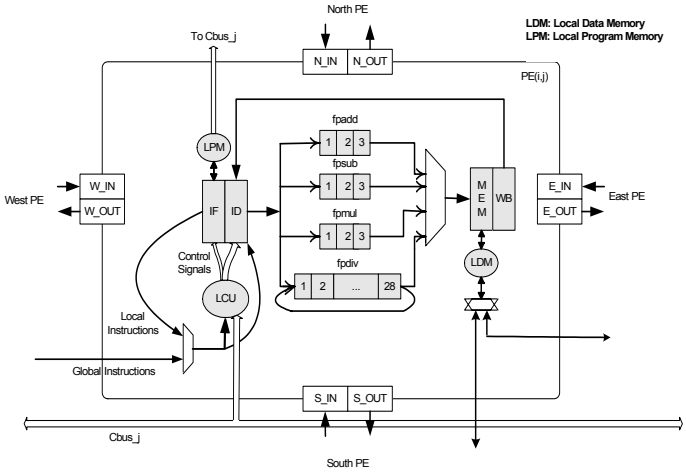

Fig. 2. HERA processing element

Our design philosophy for instructions is to have a small and highly optimized instruction set for our target applications, while not losing generality for the sake of programming efficiency. The instructions can be classified into five major groups: floating-point arithmetic (add/sub/mul/div), memory access, jump and branch, PE NEWS communication, and system control. All the instructions follow a three-field general format and are 32 bits wide. The instructions support immediate, register and base addressing.

The operating mode of each PE is configured dynamically by the host processor through its OMR by using the *Configure* instruction: "0" indicates SIMD and "1" sets the PE into MIMD. All PEs operate in SIMD when powered up. To switch a PE to MIMD from SIMD, the sequencer first distributes the instructions to the LPM of the PE through the Column Bus and Cbus, and then sends a *JumpI* instruction to the PE with the starting address in the MIMD code. OMR is set to 1. To switch back to SIMD, OMR is reset to "0" and the PE then listens for the broadcasting of a global instruction. The data in the registers and memories remain intact during switching. The instructions come from GPM in SIMD and from LPM in MIMD. The masking in the SIMD mode can use the PE's ID number and/or LMR.

2.3 Implementation Results

Our first implementation was carried out on the high-performance WILDSTARII-PCI [15] FPGA board from Annapolis Microsystems. The board is populated with two Xilinx XC2V6000 Virtex II FPGA devices and 24MB of DDRII SRAM memory. The board communicates with the host computer via the PCI bus interface. Every PE was assigned 4KB for LPM and 8KB for LDM. The interface to the PCI bus operates at 133MHz and the datapath is 64 bits. The computing fabric is clocked at 80MHz. The system frequency could be higher if we only use one FPGA for the whole system. We also could employ a commercial IP package to further improve the system performance. We removed the subtractor and divider from the PE in the case of MMM, and 64 PEs were implemented in the two FPGAs. Our hardware design was implemented in VHDL and can easily retarget other FPGA boards.

3. Generalized Cannon's Algorithm on HERA

3.1 Data Partitioning and Mapping

Due to limited space in this paper, please refer to textbooks about parallel MMM or [14] for the details of Cannon's MMM algorithm. In our implementation, matrices ***A*** and ***B*** can be of any shape and size (still, the row numbers of ***A*** and the column numbers of ***B*** should be the same). Let ***A*** and ***B*** be matrices of size $N1 \, x \, N2$ and $N2 \, x \, N3$, respectively. We assume that the on-chip memory can store $3m^2$ floating-point elements. To be able to store complete blocks from the input and output matrices, the maximum size of a matrix block should be m x m. Let $p1 = \lfloor N1/(q*m) \rfloor$, $p2 = \lfloor N2/(q*m) \rfloor$ and $p3 = \lfloor N3/(q*m) \rfloor$. In general, we first partition ***A*** and ***B*** into 2 x 2 block-based matrices as shown in the example of Fig. 3, in such a way that the sizes of A(1,1) and B(1,1) are $\{p1*(q*m)\} \mathrm{x} \{p2*(q*m)\}$ and $\{p2*(q*m)\} \mathrm{x} \{p3*(q*m)\}$, respectively. The remaining blocks A(2,1), A(1,2) and A(2,2) of ***A*** are decomposed into blocks with maximum dimension m. ***B*** is partitioned similarly. Blocks A(1,1) and B(1,1) are then partitioned into $p1$ x $p2$ and $p2$ x $p3$ blocks of size $(q*m) \mathrm{x} (q*m)$ again and are distributed into the processors in a cyclic checkerboard-like fashion. Nevertheless, there are some special cases we need to mention. If $N1 \bmod (m*q) = 0$, $N2 \bmod (m*q) = 0$ and $N3 \bmod (m*q) = 0$, then there are no A(1,2)/B(1,2), A(2,1)/B(2,1) and A(2,2)/B(2,2) pairs. If $p1 = 0$, then there are no A(1,1) and A(1,2). Similarly, if $p2 = 0$, then there are no A(1,1)/B(1,1) and A(2,1)/B(1,2) pairs. If $p3 = 0$, then there are no B(1,1) and B(2,1). These special cases still can be solved by the presented algorithm.

Cannon's algorithm asks for row and column wraparound connections while our machine has only column wraparounds. In order to solve this problem, the Sequencer pipes the needed blocks through the Data Bus into PE (i, 8) (see Fig. 1), instead of PE (i, 1).

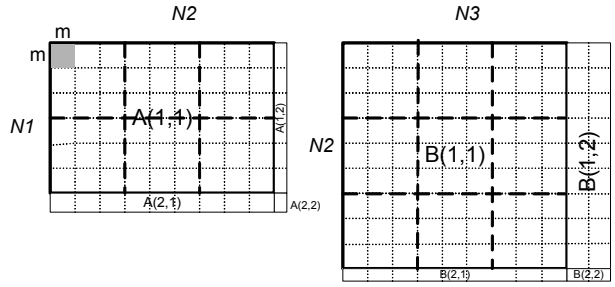

Fig. 3. A partitioning example for matrices *A* and *B* $(q = 3, p1 = 2, p2 = 3, p3 = 3)$

3.2 Dynamic Task Scheduling on HERA

If the matrices ***A*** and ***B*** are square, and can be partitioned into an integer multiple of q, then Cannon's algorithm works best in the SIMD mode; all the PEs are then busy all the time except during the initial alignment. If the ***A*** and ***B*** matrices are not square or cannot be partitioned in such a way that N (the matrix dimension) is a multiple integer of $q*m$, then the multiplication of the border blocks is not efficient in the SIMD mode because the sizes and numbers of blocks are irregular. Some PEs are idle while other PEs are busy at some point because SIMD is a implicitly synchronous mode. We solved this problem by changing the computation mode of the PEs. Also, we skip the initial alignment by assigning data blocks in a pre-skewed way. Because our PE is pipelined, we assume that multiplication, addition and shift operations all take one clock cycle, T_{clk}. The total execution time for Cannon's procedure on one partition is approximately $T_c(n) = q*(2T_{shift} + T_{mul} + T_{add}) = 2(\frac{n^3}{q^2} + \frac{n^2}{q})*T_{clk}$, assuming that the size of the submatrix is $n \, x \, n$.

The dynamic mixed-mode scheduling procedure for our modified Cannon's algorithm on HERA is as follows.

Step 1. Carry out block multiplications involving $A(1,1)*B(1,1)$ by using Cannon's algorithm; our divide-and-conquer technique carries out a total of $p1*p2*p3$ block multiplications. So, the total time is about $p1*p2*p3*T_c(200)$. All the PEs take part in this step and are configured into the SIMD mode.

Step 2. Carry out the summation of intermediate results for $A(1,1)*B(1,1)$; we have a total of $p1*p3*(p2-1)$ additions of matrix blocks of size 200 x 200. Again, all the PEs take part in this step and operate in the SIMD mode.

Step 3. If the size of A(1,2) and/or B(2,1) is larger than ½($q*m$), then carry out $A(1,1)*B(1,2)$ and/or

$A(2,1)*B(1,1)$ in SIMD using Cannon's procedure. Otherwise, go to Step 4.

Step 4. From now on, a job is a multiplication of two blocks. Jobs are divided into two groups: SIMD and MIMD jobs. SIMD jobs are those corresponding to similar number of operations on the PEs. The remaining jobs go to an MIMD queue. Count the number of jobs and their associated number of operations in the remaining work. Determine the IDs of PEs that will work in the SIMD and MIMD mode based on the job information.

Step 5. Configure individual PEs in the system into either the SIMD or the MIMD mode based on the decision in the previous step. The system now works in the mixed mode. Assign the SIMD jobs to the PEs running in the SIMD mode and distribute the MIMD jobs to the PEs running in the MIMD mode. The host dispatches jobs to the PEs whenever they are ready for the next job.

Step 6. Let the host assign the addition jobs (i.e., the addition of blocks belonging to the same block in the result matrix) to PEs after all the multiplication jobs are finished.

For the computation of the quadrants in the result matrix, $A(1,1)*B(1,1)$, $A(1,1)*B(1,2)$ and $A(2,1)*B(1,1)$ consume most of the execution time. In all the steps, except *Step1*, data locality is a priority factor in job assignment.

4. Performance Results and Analysis

We first implemented our mixed-mode MMM technique on regular square matrices of size up to 1000 x 1000 by using HERA's instruction set. The execution times on HERA are presented in Fig. 4. In order to compare the performance of modern FPGA-based machines with that of microprocessors, we also implemented block-based MMM with C code on two commercial PCs and comparative results are shown in Fig. 4. The time for data I/O for all machines is not included in Fig. 4. The block-based MMM code for the DELL PCs was optimized by several techniques, including using the L1 and L2 cache sizes to determine the best block size, compiler flags and copy optimization [16]. We can see that our results on HERA are better than both the dual-Xeon 2.66GHz and the uni-Pentium IV 2GHz systems despite HERA's much lower clock (i.e. 80MHz). There are several key contributing factors: (1) HERA, or FPGA-based machines in general, usually have a much more efficient, application-optimized instruction set. So they have many fewer instructions than general-purpose microprocessors for the same application. (2) The performance gap between microprocessors and memory chips is growing, while HERA is equipped with tightly coupled memory; (3) With FPGAs, we can build highly parallel machines based on the available resources and take advantage of many optimized parallel algorithms. Microprocessors are inherently sequential and many techniques exploring instruction level parallelism (ILP) suffer from rather small extracted ILP in algorithms and applications.

The speedups of the parallel implementation on the 64-PE HERA over the sequential one on an 1-PE HERA are shown in Fig. 5. The speedup for the 100 x 100 case is low because the ratio of computation to communication times is much lower than for the other cases. We could improve the ratios and speedups in all cases by using a bigger local memory because the complexity of multiplication with a single PE for a pair of blocks is $O(n^3)$ and that of communication (i.e. shifting) is $O(n^2)$, where $n \times n$ is the block size. With increases in the problem size, the speedup and, of course, the efficiency stabilize in a very narrow range. We also evaluated the performance of mixed-mode scheduling on a variety of matrices having irregular sizes and shapes. An SIMD mapping, where the PEs work in the SIMD mode all the time, was implemented for these matrices and the results are shown in Table 1. From this table we can see that our dynamic mixed-mode scheduling can greatly boost performance when the multiplication of irregular matrices is needed. This need arises in applications such as the parallel LU factorization of sparse Block-Diagonal-Bordered matrices [12].

5. Conclusions

This paper focused on the design of our HERA machine, a reconfigurable, pipelined computing engine implemented on platform FPGAs. It is flexible enough to allow mixed-mode execution of various parallel processing modes including SIMD, MIMD and M-SIMD. The system efficiency is improved significantly by employing several techniques, such as PE pipelining, pipelined data I/O, a small and optimized set of instructions, novel memory interface, etc. Taking advantage of the robustness in our mixed-mode hardware design, we extended Cannon's MMM algorithm for matrices of arbitrary sizes and shapes. The results show that the system efficiency stabilizes with increases in the problem size, which shows the good scalability of our architecture and algorithm. Also, HERA has shown better peak floating-point performance for a real application (i.e. MMM) than modern PCs in spite of its much lower system frequency. Many applications can benefit from the high-performance and flexibility resulting from dynamic mixed-mode scheduling; this is especially true for programs rich in conditional and non-deterministic operations. Our work shows that new-generation FPGAs have made feasible the building of highly parallel complex systems supporting floating-point arithmetic. Of course, another major advantage is that these systems are easily accessible and portable. With the anticipated speed and density improvements for FPGAs, reconfigurable computers can enter mainstream parallel computing because they have the potential to narrow the growing performance gap between algorithms and chips.

6. References

[1] M. D. Theys, T. D. Braun, and H. J. Siegel, Widespread Acceptance of General-Purpose, Large-Scale Parallel Machines: Fact, Future, or Fantasy? *IEEE Concur.*, *6*(1), January-March 1998, 79-83.

[2] H.D. Simon, The Divergence Problem, *18th International Supercomputer Conference (ISC2003)*, Heidelberg, Germany, June 2003.

[3] J. M. Rabaey, Silicon Platforms for the Next Generation Wireless Systems – What Role does Reconfigurable Hardware Play? *10th Intl. Conf. on Field-Program.Logic and Appl. (FPL 2000),* Berlin, Germany, 2000, 277-285.

[4] K. Olukotun, B.A. Nayfeh, L. Hammond, K. Wilson and K. Chang, The Case for a Single-Chip Multiprocessor, *Seventh International Symp. Architectural Support for Programming Languages and Operating Systems,* Oct. 1996, 2-11.

[5] J. M. Baker Jr., S. Bennett, M. Bucciero, B. Gold and R. Mahajan, SCMP: A Single-Chip Message-Passing Parallel Computer" *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, NV, June 2002, 1485-1491.

[6] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe and A. Agarwal, Baring it All to Software: Raw Machines, *IEEE Computer*, Sept. 1997, 86-93.

[7] R. Ronen, A. Mendelson, K. Lai, S-L. Lu, F. Pollack and J. Shen, Coming Challenges in Microarchitecture and Architecture, *Proceedings of the IEEE*, *89(*3), March 2001.

[8] K. Compton and S. Hauck, Reconfigurable Computing: A Survey of Systems and Software, *ACM Comput. Surveys*, *34*(2), June 2002, 171-210.

[9] K. Bondalapati and V.K. Prasanna, Reconfigurable Computing Systems, *Proceedings of the IEEE*, *90*(7), July 2002, 1201–1217.

[10] M.C. Smith, S.L. Drager, L. Pochet and G.D. Peterson, High Performance Reconfigurable Computing Systems, *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems,* 2001, Vol. 1, Aug. 2001, 462–465.

[11] K. Underwood, FPGAs vs. CPUs: Trends in Peak Floating-Point Performance, *ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays,* Monterey, CA, Feb. 2004, 171-180.

[12] X. Wang and S.G. Ziavras, Parallel LU Factorization of Sparse Matrices on FPGA-Based Configurable Computing Engines, *Concurrency and Computation: Practice and Experience, 16*(4), 2004, 319-343.

[13] H.J. Siegel, M. Maheswaran, D.W. Watson, J. K. Antonio and M. J. Atallah, Mixed-Mode System Heterogeneous Computing, in *Heterogeneous Computing* (M. M. Eshaghian (Ed.), Artech House, Norwood, MA, 1996).

[14] L. E. Cannon, *A Cellular Computer to Implement the Kalman Filter Algorithm,* PhD Thesis, Montana State University, 1969.

[15] http://www.annapmicro.com/.

[16] D. Parello, O. Temam and J. Verdun, On Increasing Architecture Awareness in Program Optimizations to Bridge the Gap between Peak and Sustained Processor Performance--Matrix Multiply Revisited, *2002 ACM/IEEE Conference on Supercomputing*, Baltimore, Maryland, Nov. 2002, 1-11.

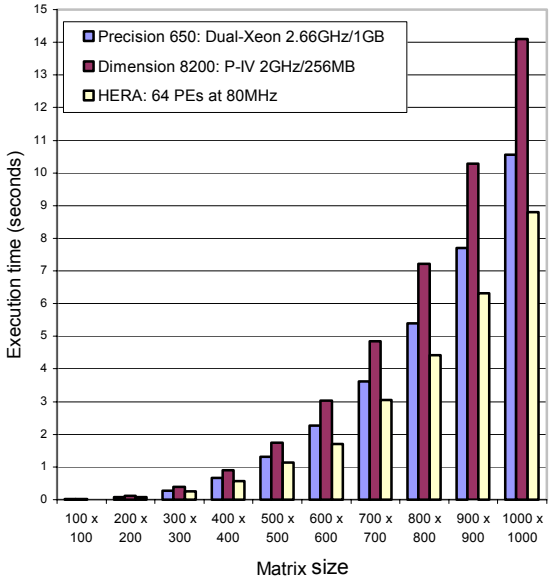

Fig. 4. Performance comparison of MMM on HERA and two DELL PCs (optimized code was run on all the machines)

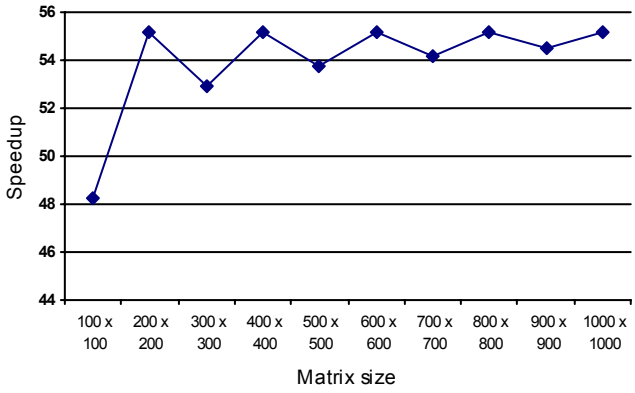

Fig. 5. Speedup of parallel execution over uni-PE execution

Table 1. HERA execution times for irregular matrices
(Clock frequency: 80MHz)

Matrix Dimensions			HERA in SIMD mode (sec)	HERA in mixed-mode (sec)	Improvement (%)
N1	N2	N3			
105	101	113	0.0179	0.0161	10.1
201	215	323	0.135	0.113	16.3
324	599	315	0.578	0.526	9.8
05	611	613	1.446	1.227	15.1
509	301	201	0.338	0.296	12.4
677	202	677	0.787	0.742	5.7
711	713	403	2.085	1.810	13.2
955	957	976	8.762	8.105	7.5