An FPGA-Based Parallel Accelerator for Matrix Multiplications in the Newton-Raphson Method*

Xizhen Xu[1], Sotirios G. Ziavras[1], and Tae-Gyu Chang[2]

[1] Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
ziavras@adm.njit.edu

[2] School of Electrical and Electronics Engineering
Chung-Ang University
Seoul 156-756, South Korea

Abstract. Power flow analysis plays an important role in power grid configurations, operating management and contingency analysis. The Newton-Raphson (NR) iterative method is often enlisted for solving power flow analysis problems. However, it involves computation-expensive matrix multiplications (MMs). In this paper we propose an FPGA-based Hierarchical-SIMD (H-SIMD) machine with its codesign of the Hierarchical Instruction Set Architecture (HISA) to speed up MM within each NR iteration. FPGA stands for Field-Programmable Gate Array. HISA is comprised of medium-grain and coarse-grain instructions. The H-SIMD machine also facilitates better mapping of MM onto recent multimillion-gate FPGAs. At each level, any HISA instruction is classified to be of either the communication or computation type. The former are executed by a controller while the latter are issued to lower levels in the hierarchy. Additionally, by using a memory switching scheme and the high-level HISA set to partition applications, the host-FPGA communication overheads can be hidden. Our test results show sustained high performance.

1 Introduction

It is not uncommon in power flow analysis to make a good initial guess regarding the solution, e.g., a hot or flat start[16]. Thus, the Newton-Raphson iterative method is often used in power flow problems because a good initial guess leads to desirable convergence properties. If we profile the code of the NR algorithm, we will find out that the most expensive computations are MMs. Real time solutions to power flow problems are absolutely essential in power grid configurations, operating management and contingency analysis. In this paper, an FPGA-based parallel computing architecture is proposed to speed up the MM component in the NR iterations.

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

Multimillion-gate FPGAs can form promising hardware accelerators for conventional hosts, e.g., a workstation or an embedded microprocessor[1][2][13][14][15]. The workstation-FPGA combination is popular for data-intensive applications due to high FPGA resource efficiency and flexible workstation control. However, the substantial communication and interrupt overheads between the workstation and the FPGAs is also becoming a major performance bottleneck that may prevent further exploitation of the performance benefits gained from the parallel FPGA implementation [3][14].

Specifically, the contributions of our work are: i) We explore the FPGA-based design space to accelerate MM computations in NR iterations. To this extent, a hierarchical multi-FPGA system is proposed where each FPGA works in the SIMD (Single-Instruction Multiple-Data) parallel-processing mode. Under SIMD, all processors execute the same instruction simultaneously but on different data. Due to task partitioning with different granularities at various levels, we can eliminate communication requests of the processing elements (PEs) within the H-SIMD machine if a block-based matrix multiplication algorithm is employed. ii) We employ a memory switching scheme to overlap communications with computations as much as possible at each level. The conditions to fully overlap communications with computations are investigated as well. This technique overcomes the FPGA interrupt overheads and the rather low speed of the PCI bus that connects our FPGA-based target platform to the host [7]. Thus, our proposed methodology makes it possible to synthesize a scheme that brings together the computing power of the workstation and the FPGAs seamlessly for the NR algorithm.

Many research projects have studied MM for reconfigurable systems [4][5][6]. [4] proposed scalable and modular algorithms. Yet the authors point out that the proposed algorithms still incur high configuration overhead and large-sized configuration files. [5] introduced a parallel block-based algorithm for MM with substantial results. Though their design is based on a host-FPGA architecture and pipelined operation control is employed as well, the interrupt overhead from the FPGA to the host is not taken into consideration for a workstation host. Hence, [4][5] can not be used to accelerate MM in the NR method. [6] concluded that FPGAs can achieve higher performance with less memory and bandwidth than a commodity processor.

The rest of this paper is organized as follows. Section 2 analyzes the NR iterative method, and presents the H-SIMD machine design and its memory switching scheme. Section 3 presents the HISA instruction set for NR and analyzes workload balancing for MM across H-SIMDs different layers. Section 4 shows implementation results and a comparative study with other works. Section 5 draws conclusions.

2 Multi-Layered H-SIMD Machine and Newton-Raphson Method

2.1 Newton-Raphson Iterative Method

The NR method employs Taylors series expression for a function with two or more variables [16]. It replaces the Gauss-Seidel method which is characterized by slower convergence. The nonlinear Newton-Raphson-type iteration for finding the reciprocal $1/A$ of matrix A is $X(k+1) = X(k)(2I - AX(k))$, where k is the iteration number and $X(0)$ is the initial guess for A^{-1}. The iterative technique proceeds until the sum of the absolute values of the off-diagonal elements in $AX(k)$ is less than ϵ, where ϵ is the required accuracy. The convergence rate is determined by the initial choice of $X(0)$. The process converges if and only if all eigenvalues of $I - AX(0)$ have absolute value less than one. Convergence, when it occurs, is generally quadratic. An improvement can be made so that the algorithms convergence is cubic. The pseudo-code for the NR algorithm is shown in Fig. 1. We can tell that two matrix multiplications are needed per iteration. Hence, our H-SIMD machine is designed for the acceleration of MM in the NR iterations.

2.2 H-SIMD Architecture

The H-SIMD control hierarchy is composed of three layers: the host controller (HC), the FPGA controllers (FCs) and the nano-processor controllers (NPCs), as shown in Fig. 2. The HC encounters the coarse-grain host SIMD instructions (HSIs) in the application program, which are classified into host-FPGA communication HSIs and time-consuming computation HSIs. The HC executes the communication HSIs only and issues computation HSIs to FCs. Inside each FPGA, the FC further decomposes the received computation HSIs into a sequence of medium-grain FPGA SIMD instructions (FSIs). The FC runs them in a manner similar to the HC: executing communication FSIs and issuing the computation FSIs to the nano-processor array. The NPCs finally decode the received computation FSIs into fine-grain nano-processor instructions (NPIs) and sequence their execution. Due to the difference between computation instructions and communication instructions at all levels, the H-SIMD machine configures one of the FPGAs as the master FPGA which sends an interrupt signal back to the HC once the previously executed computation HSI has been completed. Similarly, one NP within each FPGA is configured as the master NP that sends an interrupt signal back to its FC so that a new computation FSI can be executed.

The communication overhead between the host and the FPGAs is very high primarily due to the nature of the non-preemptive operating system on the workstation. Based on tests in our laboratory, the one-time interrupt latency for a Windows-XP installed workstation running the PCI bus at 133MHz is about 1.5 ms. This penalty is intolerable in high-performance computing [14]. Thus, a design objective of the H-SIMD machine is to reduce the interrupt

```
Initialization & flat start;
X1_matrix= alpha*transpose(A);
do{
   X0_matrix = X1_matrix;
   multiply_matrix(A_matrix, X0_matrix, temp1_matrix);
   multiply_minus(2*I, temp1_matrix, temp2_matrix);
   multiply_matrix(X0_matrix, temp2_matrix, X1_matrix);
}while( ||X1_matrix - X0_matrix|| > 0.000001);
```

Fig. 1. The pseudo-code for the NR iterations

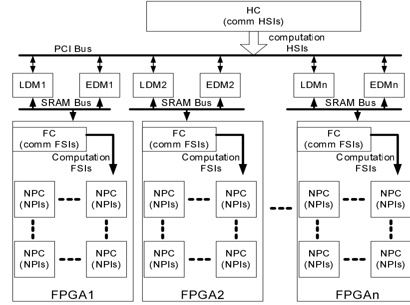

Fig. 2. H-SIMD machine architecture

overheads. A memory switching scheme has been applied successfully before in [5]. However, they did not specify the conditions to fully overlap communications with computations, a focus of our study. [14] studied such conditions but for another application problem.

The HC-level memory switching scheme is shown in Fig. 3. The SRAM banks on the FPGA board are organized into two functional memory units: the execution data memory (EDM) and the loaded data memory (LDM). Both are functionally interchangeable. At one time, the FCs access EDMs to fetch operands for the execution of received computation HSIs while LDMs are referenced by the host for the execution of communication HSIs. When the FCs finish their received computation HSI, they will switch between EDM and LDM to begin a new iteration. The FC is a finite-state machine responsible for the execution of the computation HSI. The FCs have access to the NP array over a modified LAD (M-LAD) bus. The LAD bus was originally developed by the Annapolis Micro Systems company for our target board and was used for on-chip memory references [7]. The M-LAD bus controller is changed from the PCI controller to the FCs. The HSI counter is used to calculate the number of finished computation HSIs. The SRAM address generator (SAG) is used to calculate the SRAM load/store addresses in EDM banks. The FC is pipelined and sequentially traverses the pipeline stages LL (Loading LRFs), IF (Instruction Fetch), ID (Instruction Decode) and EX (execution). The transition condition from EX to LL is triggered by the master NPs interrupt signal. The interrupt request/response latency is one cycle only as opposed to the tens of thousands of cycles between the host and FPGAs, thus enhancing the H-SIMDs performance.

The nano-processor array forms the customized execution units in the H-SIMD machine datapath. Each nano-processor has three large-sized register files: the load register file (LRF), the execution register file (ERF) and the accumulation register file (ARF), as shown in Fig. 4. Both the LRFs and ERFs work in a "memory" switching scheme, similarly to the LDMs and EDMs. The ERFs are used for the execution of computation FSIs while the LRFs are referenced by the

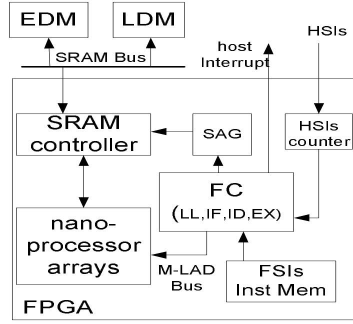

Fig. 3. HC-level memory switching scheme

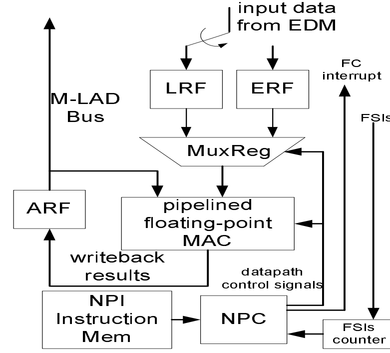

Fig. 4. Nano-processor datapath and control unit

communication FSIs at the same time. The computation results are accumulated in the ARFs which can be accessed by the FCs.

3 HISA and Task Partitioning for MM

3.1 HISA: Instruction Set Architecture for MM

Similar to the approach for PC clusters in [10], we suggest here that an effective instruction set architecture (ISA) be developed at each layer for each application domain. The HC is programmed by host API (Application Programming Interface) functions for the FPGA board. They can initialize the board, configure the FPGAs, reference the on-board/on-chip memory resources and handle interrupts. We present here the tailoring of HSIs for a block-based MM algorithm. More specifically, we assume the problem C=A*B, where A, B, and C are NxN square matrices. When N becomes large, block matrix multiplication is used that divides the matrix into smaller blocks to exploit data reusability. Due to limited space here, refer to [9] for more details about block-based MM.

In the H-SIMD machine, only a single FPGA or NP is employed to multiply and accumulate the results of one block of the product matrix at the HC and FC levels, respectively. Coarse-grain workloads can keep the NPs busy on MM computations, while the HC and FCs load operands into the FPGAs and NPs sequentially. This simplifies the hierarchical design of the architecture and eliminates the need for inter-FPGA and inter-NP communications. Based on the H-SIMD architecture, the HC issues N_hxN_h sub-matrix blocks for all the FPGAs to multiply. N_h is the block matrix size for the HSIs. We have three HSIs here: i)$host_matrix_load(i, S_{LDM}, N_h)$; ii)$host_matrix_store(i, S_{LDM}, N_h)$; iii)$host_matrix_mul_accum(H_A, H_B, H_C, N_h)$. The first two HSIs are the communication instructions while the third one is the computation instruction.

The FC is a finite state machine in charge of executing the computation HSI. It decomposes *host_matrix_mul_accum* of size N_hxN_h into FSIs of size N_fxN_f, where N_f is the sub-block matrix size for the FSIs. Enlisted is the

same block matrix multiplication algorithm as the one for the HC. The code for *host_matrix_mul_accum* is pre-programmed by FSIs and stored into the FC instruction memory. The FSIs are 32-bit instructions with mnemonics as follows: i)$FPGA_matrix_load(i, S_{LRF}, N_f)$; ii)$FPGA_matrix_store(i, S_{ARF}, N_f)$; iii)$FPGA_matrix_mul_accum(F_a, F_b, F_c, N_f)$. They are in charge of the communications and computations at the FPGA level.

The NPIs are designed for the execution of the computation FSI. The code for $FPGA_matrix_mul_accum$ is pre-programmed with NPIs and stored into the NPC instruction memory. There is only one NPI to be implemented: the floating-point multiply accumulation $NP_MAC(R_a, R_b, R_c)$, where R_a, R_b, and R_c are registers for $R_c = R_a * R_b + R_c$. The NPI code for computation FSIs needs to be scheduled to avoid data hazards. They occur when operands are delayed in the addition pipeline whose latency is L_{adder}. Thus, the condition to avoid data hazards is $N_f^2 > L_{adder}$, which can be easily met.

3.2 Analysis of Task Partitioning

The bandwidth of the communication channels in the H-SIMD machine varies greatly. Basically, there are two interfaces in the H-SIMD machine: a PCI bus of bandwidth B_{pci} between the host and the FPGAs; and the SRAM bus of B_{sram} between the off-chip memory and the on-chip nano-processor array. The HSI parameter N_h is chosen in such a manner that the execution time $T_{host_compute}$ of *host_matrix_mul_accum* is greater than $T_{host_i/o}$ which is the sum of the execution time T_{HSI_COMM} of all the communication HSIs and the master FPGA interrupt overhead T_{fpga_int}. If so, the communication and interrupt overheads can be hidden. Let us assume that there are q FPGAs of p nano-processors each. Specifically, the following lower/upper bounds hold for matrix multiplication:
$T_{host_compute} > \tau * N_h^3/p$
$T_{host_i/o} < T_{HSI_COMM} * q + T_{fpga_int} = 4 * b * N_h^2/B_{pci} * q + T_{fpga_int}$
where τ is the nano-processor cycle time and b is the width in bits of each I/O reference. Simulation results in Fig. 5 show that the HSI computation and I/O communication times vary with N_h, p and q for b=64 and τ=7 ns. With increases in the block size of HSIs, the computation time grows in a cubic manner and yet the I/O communication time grows only quadratically, which is exploited by the H-SIMD machine. This means that the host may load the LDMs sequentially while all the FPGAs run in parallel the issued HSI.

For FC-level N_fxN_f block MM, we tweak N_f to overlap the execution time $T_{FPGA_compute}$ of *FPGA_matrix_mul_accum* with the sum $T_{FPGA_i/o}$ of the execution times $T_{NP_i/o}$ of all the communication FSIs and NP interrupt overheads T_{NP_int}. The following lower/upper bounds hold:
$T_{FPGA_compute} > \tau * N_f^3$
$T_{FPGA_i/o} < T_{NP_i/o} * p + T_{NP_int} = 4 * b * N_f^2/(B_{sram} * N_{bank}) * p + T_{NP_int}$
N_{bank} is the number of available SRAM banks for each FPGA. This condition can be easily met [14]. More SRAM banks can provide higher aggregate bandwidth to reduce the execution times of the communication FSIs. By using the above analysis of the execution time, we explored the design space for the lower bound

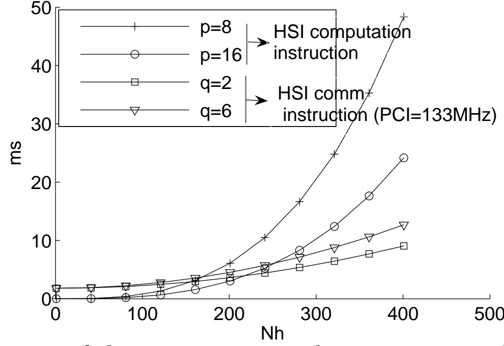

Fig. 5. Execution times of the computation and communication HSIs as a function of N_h, p and q

on N_h and N_f, respectively. On the other hand, the capacity of the on-board and on-chip memories defines the upper bounds on N_h and N_f. For each FPGA on MM operations: $4*r*N_h^2*b < C_{sram}*N_{bank}$ and $4*r*N_f^2*b < C_{on-chip}$, where C_{sram} represents the capacity of one on-board SRAM bank; $C_{on-chip}$ represents the on-chip memory capacity of one FPGA; r stands for the redundancy of the memory systems, so r=2 for our memory switching scheme. In summary, N_h and N_f are upper-bound by $\sqrt{C_{SRAM}*N_{bank}/(8*b)}$ and $\sqrt{C_{on-chip}/(8*b)}$, respectively.

4 Implementation and Experimental Results

The H-SIMD machine was implemented on an Annapolis Wildstar II PCI board containing two Xilinx XC2V6000-5 Virtex-II FPGAs [7]. We used the Quixilica FPU [8] to build the NPs floating point MAC. In our design environment, ModelSim5.8 and ISE6.2 are enlisted as development tools. The Virtex-II FPGA can hold up to 16 NPs running at 148MHz. Broadcasts of FSIs to the nanoprocessor array are pipelined so that the critical path lies in the MAC datapath. The 1024x1024 MM operation was tested. The block size N_f of the FSIs was set to 8. The test results break down into computation HSIs, host interrupt overhead, PCI reference time, and initialization and NP interrupt overhead, as shown in Fig. 6. We can tell that the performance of the H-SIMD machine depends on the block size N_h. When N_h is set to 64, the frequent interrupt requests to the host contribute to performance penalty. When N_h is set to 128, the computation time does not increase long enough to overlap the sum of the host interrupt overhead and the PCI sequential reference overheads. If N_h is set to 512, there is long enough computation time to overlap the host interrupt. However, the memory switching scheme between the EDMs and LDMs does not work effectively because of the limited capacity of the SRAM banks, which results in penalties from both host interrupts and PCI references. If N_h is set to 256, the H-SIMD pipeline is balanced along the hierarchy such that the total execution time is very close to the peak performance $T_{peak} = N^3 * \tau/(p*q)$, where all the nanoprocessors work in parallel. We can sustain 9.1 GFLOPS, which is 95% of the

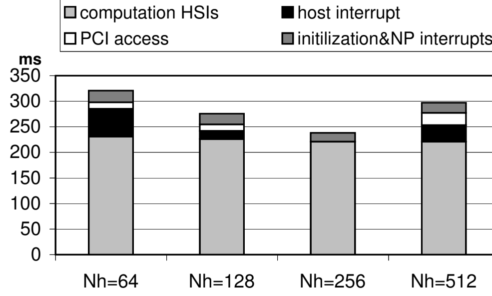

Fig. 6. 1024x1024 MM execution time as a function of N_h

peak performance. The execution overhead on the H-SIMD machine comes from the LDM and LRF initializations and the nano-processor interrupt to the FCs.

For arbitrary sizes of square matrices, a padding technique is employed to align the sizes to multiples of N_f because *FPGA_matrix_mul_accum* works on N_fxN_f matrices. N_f is set to 8 during the test. Let A and B be square matrices of size NxN. If N is not a multiple of eight, then both the A and B input matrices are padded up to the nearest multiples of eight by the ceiling function. Table 1 presents the test results for different cases. For matrices of size less than 512, the H-SIMD machine is not fully exploited and does not sustain high performance. For a large matrix ($N > 512$), the H-SIMD machine with two FPGAs can achieve about 8.9 GFLOPS on the average.

Table 1. Execution times of MM for various test cases

Matrix size	H-SIMD machine(ms)	GFLOPS
200	7	2.28
400	18	7.111
600	50	8.683
1024	238	9.023
2048	1900	9.042
4000	14100	9.078

Table 2. Performance comparisons between H-SIMD and other works for a Virtex II Pro125 FPGA

	H-SIMD	[4]	[5]
Frequency	180	200	200
Number of PEs	26	24	39
GFLOPS	9.36	8.3	15.6
Hide interrupt overhead	Yes	No	No
Size of configuration files (MB/100 cases)	5	500	500

Table 2 compares the performance of our H-SIMD machine with that of previous work on FPGA-based floating-point matrix multiplications [4][5]. Their designs were implemented on Virtex II Pro125 containing 55,616 Xilinx slices as opposed to our Virtex II 6000 FPGA that contains 33,792 slices. We scaled up the H-SIMD size to match the resources in the Virtex II Pro125. After ISE place and route, 26 NPs can fit into one Virtex II Pro125 running at 180MHz and

Table 3. Cost-performance comparison of the H-SIMD machine and the Xeon processor

System	Transistors (millions)	Execution time T(ms)	VLSI Cost	Speedup
2.8GHz	286	3.9	1	1
H-SIMD (2 FPGAs)	700	1.9	0.58	2.05

hence achieve the peak performance of 9.36GFLOPS per FPGA. The H-SIMD running frequency can be further increased if optimized MACs are enlisted. [4][5] presented a systolic algorithm to achieve 8.3 GFLOPS and 15.6 GFLOPS on a single XC2VP125 FPGA. However, the H-SIMD machine can be used as a computing accelerator for the workstation when the NR algorithm is implemented. In contrast, the systolic approach does not fit into this computing paradigm because of the FPGA configuration overheads and the large size of configuration files.

The H-SIMD performance also compares favorably to that of state-of-the-art general purpose processors. The Intel Math Kernel Library (Intel MKL) contains the BLAS implementation that has been highly optimized for Intel processors. For double-precision general-purpose matrix multiplication (DGEMM), a 2.8 GHz Xeon with 512 KB L2 cache achieves 4.5 GFLOPS [11]. The time-consuming computations in the NR algorithm correspond to MMs. Thus, the H-SIMD machine can speed up the NR method by a factor of 1.9. A cost-performance analysis of the H-SIMD machine and a Xeon processor is in order now. The ten million system gates in the Virtex II FPGA consume about 400 million transistors. The H-SIMD machine built on the Annapolis board contains two Virtex II FPGAs. Thus, our current implementation of the H-SIMD machine employs roughly 700 million transistors. On the other hand, a 2.8GHz Xeon processor is comprised of about 286 million transistors [12]. For 2048x2048 MM on IEEE-754 double-precision numbers, it takes 3.9s on a Xeon processor as opposed to 1.9s on H-SIMD. According to a widely used VLSI complexity model, the cost C of implementing an algorithm is defined as $C = A * T^2$, where A is the chip area and T is the execution time. The chip area is directly proportional to the number of transistors, so we substitute in the cost equation the latter for the former. The VLSI cost and speedup results in Table 3 are normalized with respect to the Xeon processor. The H-SIMD machine can speedup the MM by a factor of two with only about half the VLSI cost.

5 Conclusions

In this paper, we analyzed the NR iteration algorithm for power flow problems and designed an FPGA-based MM accelerator for the host workstation. The proposed multi-layered H-SIMD machine paired with an appropriate multi-layered HISA software approach is effective for the host-FPGA architecture and can be synthetically used to speed up MM in NR iterations. To yield high performance, task partitioning is carried out at different granularity levels for the host, the FPGAs and the nano-processors. If the parameters of the H-SIMD ma-

chine are chosen properly, the memory switching scheme is able to fully overlap communications with computations. In our current implementation of matrix multiplication, a complete set of HISA for this application was developed. Its good performance was demonstrated. More recently introduced FPGAs, e.g, XC2VP125, could improve performance even further.

References

1. M. J. Wirthlin, B. L. Hutchings and K. L. Gilson: The Nano Processor: a Low Resource Reconfigurable Processor. Proc. IEEE FPGAs Custom Comput. (March 1994) 22-30
2. X. Wang and S. G. Ziavras: Performance Optimization of an FPGA-Based Configurable Multiprocessor for Matrix Operations. IEEE Intern. Conf. Field-Programmable Tech., Tokyo, (Dec. 2003)
3. P. H. W. Leong, C. W. Sham, W. C. Wong, H. Y. Wong, W. S. Yuen and M. P. Leong: A Bitstream Reconfigurable FPGA Implementation of the WSAT Algorithm. IEEE Trans. VLSI Syst., Vol. 9, No. 1, (Feb. 2001)
4. L. Zhuo and V. K. Prasanna: Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on FPGAs. 18th Intern. Parallel Distr. Proc. Symp., (April 2004)
5. Y. Dou, S. Vassiliadis, G. K. Kuzmanov and G. N. Gaydadjiev: 64-bit Floating-Point FPGA Matrix Multiplication. 2005 ACM/SIGDA 13th Intern. Symp. FPGAs, (Feb. 2005)
6. K. D. Underwood and K. S. Hemmert: Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance. IEEE Symp. Field-Progr. Custom Comput. Mach., (April 2004)
7. Wildstar II Hardware Reference Manual, Annapolis Microsystems, Inc, Annapolis, MD, (2004)
8. Quixilica Floating Point FPGA Cores Datasheet, QinetiQ Ltd, (2004)
9. R. Schreiber: Numerical Algorithms for Modern Parallel Computer Architectures. Springer-Verlag, New York, NY, (1988), 197–208
10. D. Jin and S. Ziavras: A Super-Programming Approach for Mining Association Rules in Parallel on PC Clusters. IEEE Trans. on Parallel Distr. Systems, Vol. 15, No. 9, (Sept. 2004)
11. Performance Benchmarks for Intel Math Kernel Library, Intel Corporation White Paper, (2003)
12. http://www.intel.com/pressroom/kits/quickreffam.htm#Xeon
13. X. Wang and S.G. Ziavras: Parallel LU Factorization of Sparse Matrices on FPGA-Based Configurable Computing Engines. Concurrency and Computation: Practice and Experience, Vol. 16, No. 4, (April 2004) pp.319–343
14. X. Xu and S.G. Ziavras: A Hierarchically-Controlled SIMD Machine for 2D DCT on FPGAs. IEEE International Systems-On-Chip Conference, Washington, D.C., (Sept. 2005)
15. X. Wang and S.G. Ziavras: Parallel Direct Solution of Linear Equations on FPGA-Based Machines. Workshop on Parallel and Distributed Real-Time Systems (in conjunction with the 17th Annual IEEE International Parallel and Distributed Processing Symposium), Nice, France, (April 2003)
16. W. F. Tinney and C. E. Hart: Power Flow Solution by Newton's Method. IEEE Trans. AS, PAS-86, No. 11, (1967) 1449–1460