H-SIMD Machine: Configurable Parallel Computing for Matrix Multiplication*

Xizhen Xu and Sotirios G. Ziavras
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
Email: ziavras@njit.edu

Abstract

FPGAs (Field-Programmable Gate Arrays) are often used as coprocessors to boost the performance of data-intensive applications [1, 2]. However, mapping algorithms onto multimillion-gate FPGAs is time consuming and remains a challenge in configurable system design. The communication overhead between the host workstation and the FPGAs is also significant. To address these problems, we propose in this paper the FPGA-based Hierarchical-SIMD (H-SIMD) machine with its codesign of the Hierarchical Instruction Set Architecture (HISA). At each level, HISA instructions are classified into communication instructions or computation instructions. The former are executed by the local controller while the latter are issued to the lower level for execution. Additionally, by using a memory switching scheme and the high-level HISA set to partition the application into coarse-grain tasks, the host-FPGA communication overhead can be hidden. We enlist matrix multiplication (MM) to test the effectiveness of H-SIMD. The test results show sustained high performance.

1. Introduction

In the past decade, multimillion-gate FPGAs have emerged as a powerful computing accelerator to the conventional host, e.g., a workstation or an embedded microprocessor. The workstation-FPGA architecture is popular for large-sized data-intensive applications due to the nature of FPGA resources and flexible workstation control. Conventionally, custom FPGA architectures are widely used to speed up computing performance. However, mapping an application algorithm to multimillion-gate FPGAs is time consuming, and may incur high configuration overhead and large configuration files [4]. The substantial communication and interrupt overheads between the workstation and the FPGAs also is a major performance bottleneck that may prevent further exploitation of the performance benefits gained from the parallel FPGA implementation [3].

Our proposed H-SIMD machine is designed to create a hierarchical and balanced pipeline from the host to the FPGAs. Assuming a multi-FPGA target system, we configure each FPGA chip as a separate SIMD architecture. Multiple FPGA chips can be synchronized to work in SIMD at a higher level controlled by the host. The host sends function IDs as well as corresponding operands to the FPGAs in a manner that attempts to balance the workload across FPGAs. Based on HISA, application tasks are partitioned into three layers: the host, FPGA and nano-processor (NP) layers, in decreasing order of task granularity. Because of this partitioning, it is possible to employ a memory switching scheme for data loads/stores. The switching between pairs of data memory banks overlaps operand computations with communications, thus improving performance. Our test analysis of matrix multiplication (MM) shows that the sustained performance of our H-SIMD machine is very close to its peak performance. Specifically, the contributions of our work are: 1) We propose a hierarchical multi-FPGA system working in the SIMD parallel mode. Due to task partitioning with different granularities at each level, there is no need for communications between processing elements (PEs) in the H-SIMD machine if the block matrix multiplication algorithm is employed. Not only does this hierarchical architecture facilitate this modular design methodology to reduce the development cycles, but also it provides custom datapaths to speed up the application-specific computations at runtime. 2) We employ a memory switching scheme to overlap computations with communications as much as possible at each level. We also investigate the conditions for completely overlapping computations with communications. This technique overcomes the FPGA interrupt overhead and the PCI bus bandwidth limitation. Thus, it is possible to bring together the computing power of the workstation and FPGAs synthetically.

Many research projects study MM for reconfigurable systems [4, 5, 6]. [4] proposed scalable and modular algorithms for FPGA-based MM. Their algorithms can achieve sustainable high performance with an area-speed optimized floating point unit (FPU). Yet the authors point out that the proposed algorithms still incur high configuration overhead, large-sized configuration files and lack of flexibility. In [5], the authors introduce a parallel block algorithm for MM with impressive results. Though their design is based on a host-FPGA architecture and pipelined operation control is employed, the interrupt overheads between the FPGAs and the host are not taken into consideration. A linear array of PEs is built up to run in the SPMD mode in [5]. [6] investigates sustainable

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

floating-point BLAS performance on a commodity processor, FPGAs and a reconfigurable computing platform. They concluded that FPGAs can achieve higher performance with less memory capacity and bandwidth than the commodity processor.

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

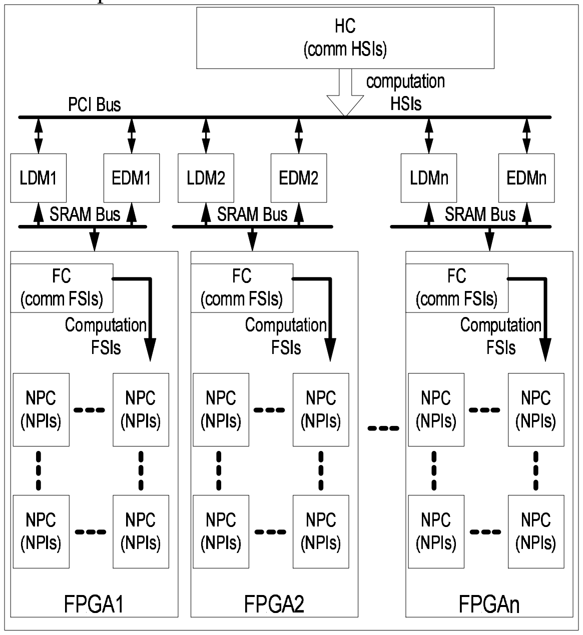
2. Multi-Layered H-SIMD Machine

2.1 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 Figure 1. The HC runs the coarse-grain host SIMD instructions (HSIs) which are classified into host-FPGA communication HSIs and time-consuming computation HSIs. The HC executes the communication HSIs only and issues computation HSIs for FCs to execute. 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 computation FSIs to the nano-processor array. The NPCs finally decode the received computation FSIs into fine-grain nano-processor instructions (NPIs) and sequence them for execution. Due to different execution levels between computation instructions and communication instructions, the H-SIMD machine configures one FPGA as the master which sends an interrupt signal back to the HC once the previously executed computation HSI has been done. 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 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 with a 133MHz PCI bus is about 1.5 ms. This penalty is intolerable in high-performance computing because, for example, 60x60 floating-point matrix multiplication takes about 1.3 ms on a single MAC running at 160 MHz (which is within the range of current FPGA technology). Thus, a design objective of the H-SIMD machine is to reduce the interrupt overheads. A memory switching scheme is used in [5, 7]. But the authors do not specify the conditions to fully overlap computations with communications. Our data prefetching scheme involves memory switching which is designed for the H-SIMD machine to overlap communications with computations as much as possible.

The HC-level memory switching scheme is shown in Figure 2. 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 the EDMs and LDMs 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 current computation HSI, they will switch between EDM and LDM to begin a new iteration. FC is a finite-state machine responsible for the execution of the computation HSI. FCs have access to the NP array over a modified LAD (M-LAD) bus. The LAD bus was originally developed by the Annapolis company and used for on-chip memory references [8]. 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 for the EDM banks. The FC is pipelined and sequentially traverses the states of LL (Loading LRFs), IF (Instruction Fetch), ID (Instruction Decode) and EX (execution). The transition condition from EX to LL is triggered by the master NP's 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-SIMD's performance.



EDM: execution data memory;
LDM: loaded data memory
Figure 1. H-SIMD machine architecture.

The nano-processor array forms the customized execution units of 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) shown in Figure 3. Both

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 communication FSIs at the same time. The computation results are accumulated in the ARFs. Our multiply-accumulator (MAC) consists of commercial Quixilica FPU IPs [9] to reduce the development cycle. Each NP can finish two floating-point operations in one cycle.

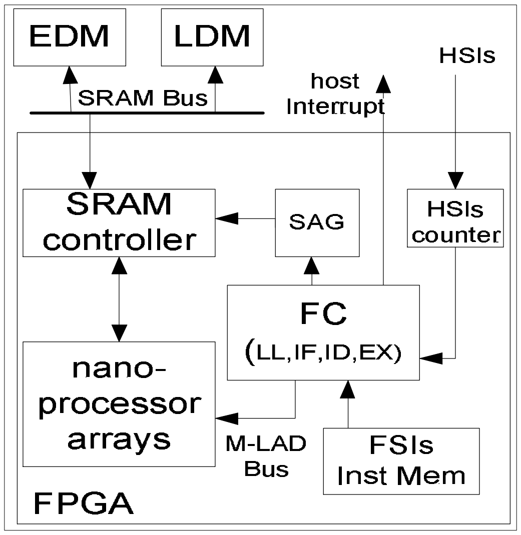


Figure 2. HC-level memory switching.

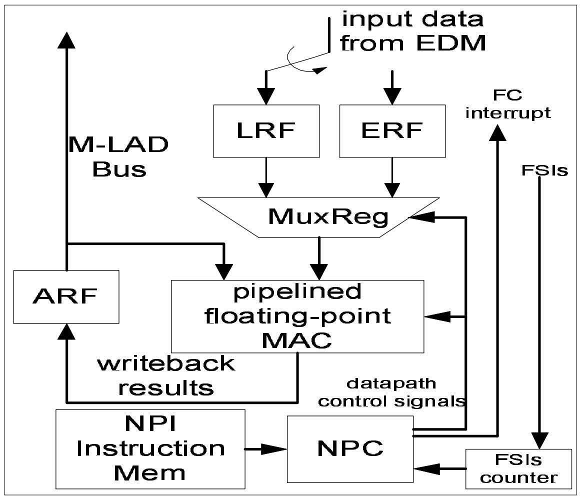


Figure 3. Nano-processor datapath and control unit.

2.2 HISA: Instruction Set Architecture for MM

Similar to the approach for PC clusters in [11], we suggest here that an effective instruction set architecture (ISA) be developed at each layer for each application domain. HC is programmed by host API functions of the FPGA board. They support to open the board, configure the FPGAs, reference the on-board/on-chip memory resources and handle interrupts [12]. We present here the tailoring of HSIs for the block-based MM algorithm. We assume the problem C=A*B, where A, B, and C are N x N square matrices. When N becomes large, block matrix multiplication is used to divide the matrix into smaller blocks to exploit data reusability. Due to limited space here, please refer to [10] for more details about block 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 to the FPGAs and NPs sequentially. This simplifies the design of the hierarchical architecture and eliminates the need for FPGA-FPGA and NP-NP communications at the expense of memory reference time. According to 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 HSIs. We have three HSIs here: 1) ***host_matrix_load(i, S_{LDM}, N_h)***: Through the PCI bus, this HSI will load an N_h x N_h block matrix to the LDM of FPGA i with the starting address ***S_{LDM}*** in the host memory (host-based DMA control is applied). 2) ***host_matrix_store(i, S_{LDM}, N_h)***: The computation results in the LDM of FPGA i can be retrieved by the host through the PCI bus when the computation is done. ***Host_matrix_load/store*** are communication HSIs executed on the host. 3) ***host_matrix_mul_accum(H_A, H_B, H_C, N_h)***: For matrix multiplication of size N_hxN_h, H_A, H_B and H_C are the starting addresses of source matrix A, source matrix B and product accumulation matrix C, respectively. This computation HSI is coded in 32 bits, issued by the HC and executed by the FCs.

The FC is in charge of executing the computation HSI, i.e., decomposing ***host_matrix_mul_accum*** for size N_h x N_h into FSIs for size N_f x N_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 the FSIs and stored into the FC instruction memory. The FSIs are 32-bit instructions with mnemonics as follows: 1) ***FPGA_matrix_load(i, S_{LRF}, N_f)***: the FC will execute this instruction by loading the LRF of NP i with a matrix of size N_fxN_f. ***S_{LRF}*** is the starting address in the EDM. 2) ***FPGA_matrix_store(i, S_{ARF}, N_f)***: The NP computation results are stored into the ARF and retrieved into the FPGA's EDM at starting address ***S_{ARF}*** when the accumulation of the partial products is done. ***FPGA_matrix_load/store*** are communication FSIs executed by the FCs. 3) ***FPGA_matrix_mul_accum(F_a, F_b, F_c, N_f)***: For matrix multiplication of size N_fxN_f, ***F_a***, ***F_b*** and ***F_c*** are the starting addresses of source matrix a, source matrix b and product accumulation matrix c, respectively. This computation FSI is issued by the FCs and executed by the NPCs.

The NPIs are designed for the execution of the computation FSI ***FPGA_matrix_mul_accum.*** The code for ***FPGA_matrix_mul_accum*** is pre-programmed by the 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 the function $R_c=R_a*R_b+R_c$. The NPI code for the computation FSIs needs to be scheduled carefully to avoid data hazards. They occur when operands are delayed in the addition pipeline with latency L_{adder}. Thus, the condition to avoid data hazards is $N_f^2 > L_{adder}$.

3. Analysis of Task Partitioning in Matrix Multiplication

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; the SRAM bus of bandwidth 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 the HSI computation instruction ***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 (***host_matrix_load/store***) 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 should hold for matrix multiplication:

$T_{host\_compute} > t * N_h^3/p$,

$T_{host\_i/o}<T_{HSI\_comm}*q+T_{fpga\_int}=4*b*N_h^2/B_{pci}*q+T_{fpga\_int}$,

where t is the nano-processor cycle time and b is the width in bits of each I/O reference. Simulation results in Figure 4 show that the HSI computation and I/O communication times vary with N_h, p and q, for b=64 and t=7 ns. With increases in the block size for the 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 the issued HSI in parallel.

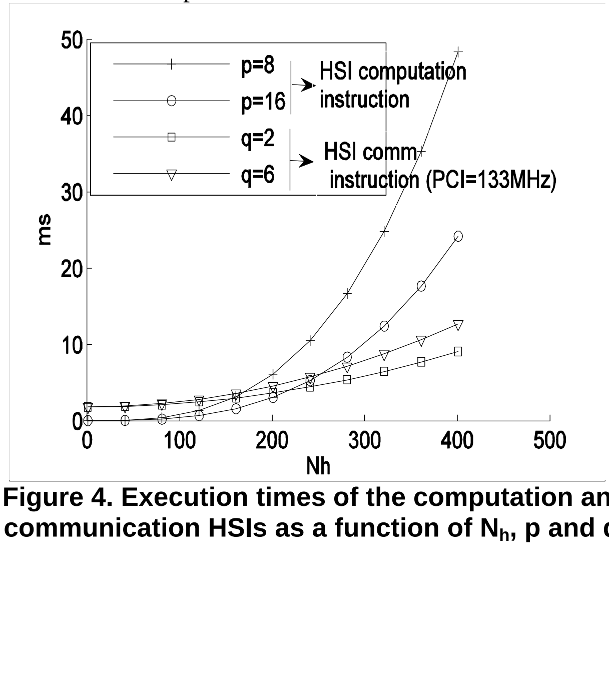
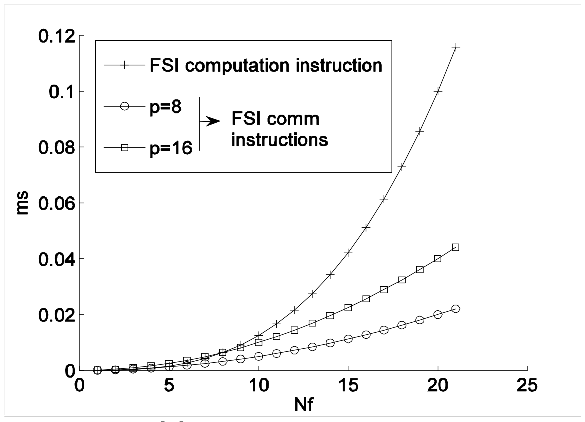
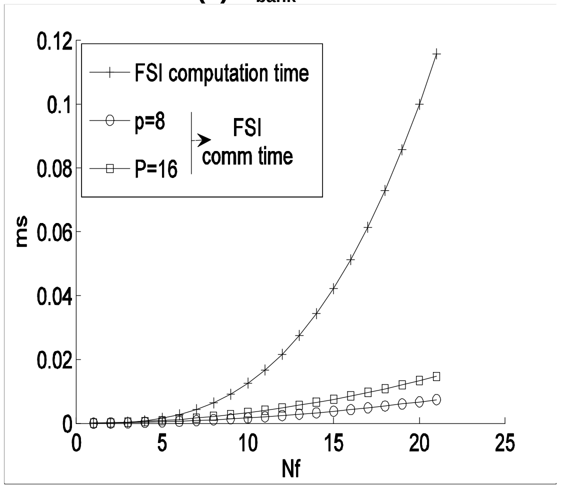


Figure 4. Execution times of the computation and communication HSIs as a function of N_h, p and q.



(a) N_{bank}=2



(b) N_{bank}=6

Figure 5. Execution times of the computation and communication FSIs as a function of N_f, p, and N_{bank}.

For FC-level N_f x N_f block MM, we tweak N_f to overlap the execution time $T_{FPGA\_compute}$ of the FSI computation instruction ***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 the NP interrupt overheads T_{NP_int}. The following upper/lower bounds should hold:

$T_{FPGA\_compute} > t*N_f^3$,

$T_{FPGA\_i/o}<T_{NP\_i/o}*p+T_{NP\_int}=4*b*N_f^2/(B_{sram}*N_{bank})*p + T_{NP\_int}$

N_{bank} is the number of available SRAM banks for each FPGA. Simulation results in Figure 5 show that the computation FSI takes more execution time than the communication FSIs with an increase in N_f. More SRAM banks can provide a higher aggregate bandwidth to reduce the execution times of the communication FSIs. By using the above analysis of the execution time, we explore the design space for the lower bound on N_h and N_f, respectively. On the other hand, the capacity of the off-chip and on-chip memories defines the upper bounds on N_h and N_f. For each FPGA for MM operations: $4*r*N_h^2*b<C_{sram}*N_{bank}$ and $4*r*N_f^2*b <C_{on\text{-}chip}$, where C_{sram} represents the capacity of one on-board SRAM bank; $C_{on\text{-}chip}$ represents the on-chip memory capacity of one FPGA; r stands for the redundancy

of the memory system, so r=2 for our memory switching scheme. In summary, N_h are 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 the Annapolis Wildstar II PCI board containing two Xilinx Virtex-II 6000 FPGAs [8]. We use the Quixilica FPU [9] to build up the NP's floating point MAC. Table 1 gives the characteristics of the Quixilica FPU and MAC for the 64-bit IEEE double-precision format. In our design environment, ModelSim5.8 and ISE6.2 are enlisted as development tools. The Virtex-II 6000 can hold up to 16 NPs running at 148MHz. Broadcasting the FSIs to the nano-processor array is pipelined so that the critical path lies in the MAC datapath. The 1024x1024 MM operation is tested. The block size N_f of FSIs is 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 Figure 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 the performance penalty. When N_h is set to 128, the computation time of the coarse-grain HSI does not increase long enough to overlap the sum of the host interrupt overhead and the PCI sequential reference overhead. If N_h is set to 512, there is a 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 Peak=2*p*q*freq, where all the nano-processors work in parallel. We can sustain 9.1 GFLOPS, which is 95% of the peak performance. The execution overhead on the H-SIMD machine comes from LDM and LRF initialization, and nano-processor interrupts to the FCs.

Table 1. Characteristics of the Quixilica FPU and H-SIMD MAC.

	fpAdder	fpMultiplier	MAC
Pipeline Stages	12	11	24
Slice usage	815	923	1802
Clock speed (MHz)	153	150	148

For an arbitrary size of square MM operations, a padding technique is employed to align the size of the input matrices to multiples of N_f because ***FPGA_matrix_mul_accum*** works on N_f x N_f matrices. N_f is set to 8 during our 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. The padded zeros will definitely increase the H-SIMD's computation overhead and lower its performance. Table 2 presents 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 the large matrix (N>512), the H-SIMD machine with two FPGAs can achieve about 8.9 GFLOPS on average. In fact, the H-SIMD machine can be built with multiple FPGAs because no inter-FPGA communications are needed. We show the relationship between the execution time of 2048x2048 MM and the number q of FPGAs in Figure 7. There exists a saturation point, beyond which the number of FPGAs does not affect the performance significantly. For the case study of block matrix multiplication, seven Virtex II 6000 FPGAs can be enlisted to achieve 31.85 GFLOPS for 64-bit IEEE format floating point MM.

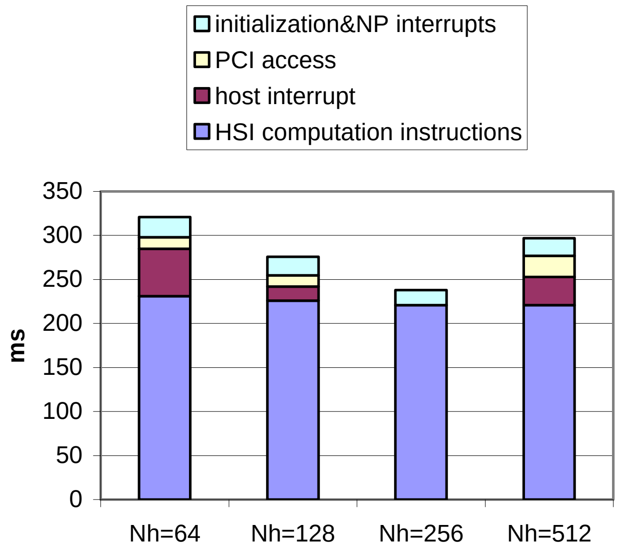


Figure 6. 1024x1024 MM execution time as a function of N_h.

Table 3 compares the performance of our H-SIMD machine with that of previous works on FPGA-based floating-point matrix multiplication [4][5]. Their designs were implemented on Virtex II Pro125 FPGAS (55,616 slices) as opposed to our Virtex II 6000 (33,792 slices). We scale the H-SIMD performance to match the Virtex II Pro125. We estimate that 26 NPs can fit into one Virtex II Pro125 running at 180MHz and can achieve a peak performance of 9.36GFLOPS. The H-SIMD running frequency can be further increased if optimized MACs are used. [4] [5] presented systolic algorithms to achieve 8.3 GFLOPS and 15.6 GFLOPS on a single Xilinx Virtex II Pro XC2VP125, respectively. However, the H-SIMD machine can be used as a computing accelerator for the workstation. The systolic approach does not fit well into this paradigm because of the interrupt overhead, the FPGA configuration overheads and the large size of configuration files.

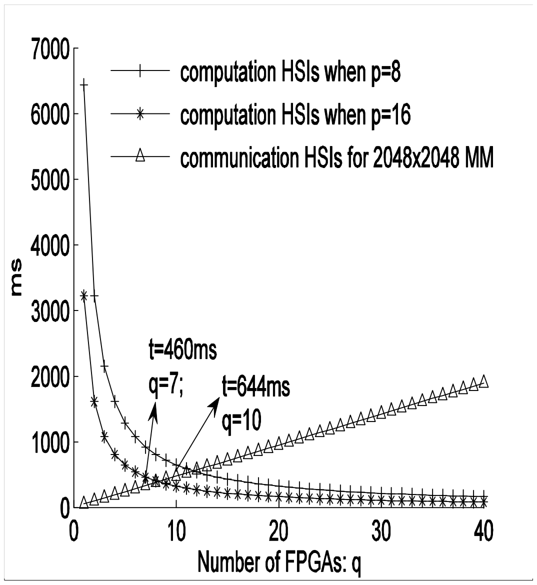


Figure 7. Execution time vs. number of FPGAs (2048x2048 MM).

Table 2. Execution time of MM for various test cases.

Matrix size	H-SIMD machine(ms)	GFLOPS
200	7	2.28
397	18	6.952
601	47	8.683
999	225	8.849
2001	1720	9.039
3999	13882	9.027

Table 3. Performance comparison between H-SIMD and other works.

	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
configuration file size (MB/100 cases)	5	500	500

5. Conclusions

Our multi-layered H-SIMD machine paired with an appropriate multi-layered HISA software approach is effective for data parallel applications. To yield high performance, task partitioning is carried out at different granularity levels for the host, FPGAs and nano-processors. If the parameters of the H-SIMD machine 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 and its high performance was demonstrated. More current FPGAs, e.g, the XC2VP125, could improve the performance tremendously.

References:

[1] R. Tessier and W. Burleson, “Reconfigurable Computing for Digital Signal Processing: A Survery,” *Journal of VSLI Signal Processing*, 28, 2001, pp. 7-27.

[2] M. J. Wirthlin, B. L. Hutchings, K. L. Gilson, “The Nano Processor: a Low Resource Reconfigurable Processor,” *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, April 1994, pp. 23-30.

[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 Transactions on VLSI Systems*, Vol. 9, No. 1, Feb. 2001.

[4] L. Zhuo and V. K. Prasanna, “Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on FPGAs,” *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS’04)*, April, 2004.

[5] Y. Dou, S. Vassiliadis, G. K. Kuzmanov and G. N. Gaydadjiev, “64-bit Floating-Point FPGA Matrix Multiplication,” *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on FPGAs*, Feb. 2005.

[6] K. D. Underwood and K. S. Hemmert, “Closing the Gap: CPU and FPGA Trends in Sustainable Floating-point BLAS Performance,” *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing machines (FCCM 2004)*, April 2004.

[7] H. Singh, M. H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh and E. M. Chaves Filho, “MorphoSys: an Integrated Reconfigurable System for Data-parallel and Computation-intensive Applications,” *IEEE Transactions on Computers*,Volume 49, Issue 5, May 2000.

[8] Annapolis Microsystems, Inc., “Wildstar II Hardware Reference Manual,” Annapolis Microsystems, Inc, Annapolis, MD, 2004.

[9] QinetiQ Ltd., “Quixilica Floating Point FPGA Cores Datasheet,” QinetiQ Ltd, 2004.

[10] Robert Schreiber, Numerical Algorithms for Modern Parallel Computer Architectures, pp. 197-208, Springer-Verlag, New York, NY, 1988.

[11] D. Jin and S. Ziavras, “A Super-Programming Approach for Mining Association Rules in Parallel on PC Clusters,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 9, Sept. 2004.

[12] Annapolis Microsystems, Inc., “Wildstar II Software Reference Manual,” Annapolis Microsystems, Inc, Annapolis, MD, 2004.