Adaptive Scheduling of Array-Intensive Applications on Mixed-Mode Reconfigurable Multiprocessors*

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*-Array-based applications are normally computation-intensive and many customized designs have been proposed targeting FPGA (Field-Programmable Gate Array) implementations to accelerate their real-time solutions. Most often a time-consuming procedure to convert floating-point representations for a fixed-point implementation is necessary due to insufficient hardware resources. Recent remarkable advances in state-of-the-art FPGAs provide new opportunities. This paper presents a dynamic scheduling strategy that applies novel application characterization and adaptive allocation of processors among the active tasks; we show results for our in-house developed mixed-mode MPoPC (MultipProcessor-on-a-Programmable-Chip) HERA (Heterogeneous Reconfigurable Architecture) machine. The innovative adaptive parallelization technique combined with the unique flexibilities provided by the reconfigurable logic help to achieve high utilization of resources and potentially minimum execution times. Experimental results for parallel Singular Value Decomposition (SVD) on Xilinx Virtex II FPGAs are reported.**

I. INTRODUCTION

Many signal processing algorithms commonly found in a wide range of applications, such as wireless communications, medical imaging and multimedia applications, are abundant in array data structures. The exponential growth of information and the real-time solution of these applications constantly challenge the computing power of digital systems. Traditionally, the dominant platforms have been DSP processors, ASICs and FPGAs. FPGAs have emerged as a low-cost and low-risk alternative to ASICs due to their high flexibility and capability to customize the hardware architecture in order to match the characteristics of algorithms [1]. Most often, only fixed-point computations are supported by these platforms due to insufficient resources, even though the corresponding algorithms utilize floating-point (FP) operations. Hence, significant research has focused on how to transform floating-point into fixed-point representations and vice-versa [2]. However, recent remarkable advances in FPGAs, such as millions of user-accessible logic gates, and plenty of embedded DSP and memory blocks, provide new opportunities for FP digital signal processing. Research results show that the peak FP performance of FPGAs has outnumbered in the last few years that of modern microprocessors and is growing much faster than the latter [3]. With state-of-the-art FPGAs, it is feasible to design and implement parallel FP reconfigurable computing machines [5, 9]. Another notable trend is that fine-grain, application-specific schemes face many challenges whereas coarse grain architectures are more effective on modern FPGAs [4].

HERA [5] is an MPoPC that we have designed and implemented with in-house developed IEEE-754 single-precision FP units on the Annapolis WILDSTAR II-PCI board populated with two Xilinx Virtex II FPGAs [13]. In contrast to conventional FPGA-based designs, MPoPCs are user programmable instead of developer programmable. We have not seen major research effort in this direction. The seminal advantage of FPGAs is that we can customize the hardware architecture based on application characteristics and, hence, achieve a higher utilization of hardware resources compared to fixed hardware (such as microprocessors and ASICs). Based on this motivation, HERA can be dynamically reconfigured to support a variety of independent or cooperating computing modes, such as SIMD (Single-Instruction, Multiple-Data), Multiple-SIMD, and MIMD (Multiple-Instruction, Multiple-Data).

Taking advantage of HERA's mixed-mode parallelism and reconfigurability, we present an adaptive scheduling strategy for array-intensive applications. In our computing model, an array-intensive application is represented by a mixed-mode *task flow graph* consisting of two distinct categories of tasks: SIMD and MIMD. SIMD tasks are the focus of dynamic adaptive scheduling. In contrast to most previous loop scheduling strategies for conventional multiprocessor systems [6-7], which consider just one loop at a time, the data dependences and priorities associated with multiple SIMD tasks (loops) are the main concern of our adaptive scheduling approach. In order to achieve a minimum execution time for the entire application, the number and type of PEs (Processing Elements) assigned to each SIMD task vary dynamically at runtime. Another major advantage of our dynamic scheduling (compared to related work) is its much less scheduling overhead due to the very low communication cost in MPoPCs. Moreover, our HERA system supports run-time reconfiguration provided by FPGAs, which allows to dynamically change PE functionality and reconfigure the system to increase resource utilization and match task

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

characteristics. *Singular Value Decomposition (SVD)* is investigated as being a representative computation-intensive example suitable for evaluating our approach. An FPGA implementation of fixed-point SVD on a systolic array appeared in [8]. To the best of our knowledge, no FP solution has ever been published for FPGAs.

The remainder of this paper is organized as follows. Section II provides an overview of HERA. We briefly introduce application characterization in Section III. The adaptive and dynamic scheduling algorithm is presented in detail in Section IV. The implementation details of the SVD algorithm and performance results are shown in Section V. We conclude the paper in Section VI.

II. HERA: A MIXED-MODE MPOPC

The general organization of our HERA machine with m x n PEs interconnected via a 2-D mesh network is shown in Fig. 1. The key architectural features include:

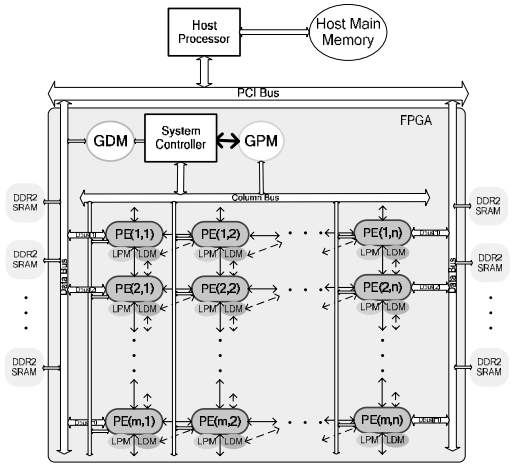

Fig. 1. HERA organization.

- Pipelined PEs supporting IEEE 754 single-precision FP operations;
- PEs programmable and customizable from a library of hardware components;
- 2-D mesh layout with NEWS connections;
- General-purpose instruction set;
- Each column of PEs has a shared bus;
- The dual-ported data memory of each PE is also directly accessible by its immediate south and west neighbors;
- Every PE can be selected by the sequencer using its ID or a mask.

III. APPLICATION CHRACTERIZTION

Any array-intensive application presented at a high level is first analyzed to construct its *Task Flow Graph* (TFG), $G = (S, D)$; it is a *weighted directed acyclic graph* (wDAG). S and D represent the set of nodes and edges, respectively. Each node in this graph represents a task $S_i \in S$, for $i \in [1, n]$. There are two types of tasks: SIMD and MIMD. Associated with each task S_i are its computing mode (SIMD/MIMD), FP computation cost and FP operation types, represented by $\varepsilon(S_i)$, $FP(S_i)$ and $\pi(S_i)$, respectively. The memory requirements of each task are represented by two parameters, $\sigma_c(S_i)$ and $\sigma_d(S_i)$, corresponding to the instruction and data memory, respectively. A directed edge between two tasks S_i and S_j represents a data dependence between them, and its weight $D_{i,j} \in D$ represents the volume of data that goes from task S_i to S_j.

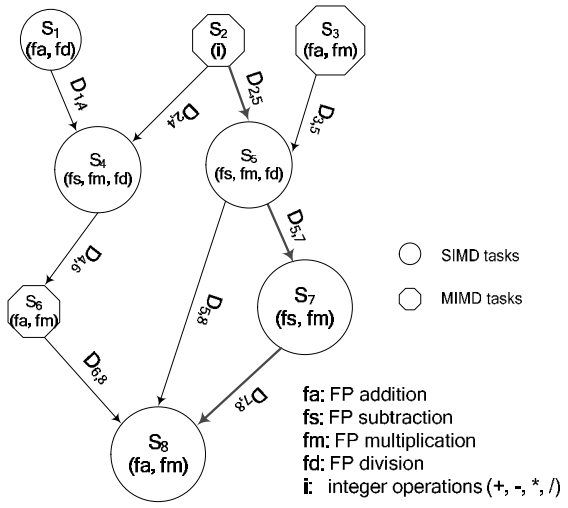

Fig. 2. A typical task flow graph.

Dynamic system configurations for a given application are employed since FP computing cores are very resource expensive. Only the required FP operations are supported by PEs. The functionality of PEs can be changed via hardware reconfiguration at runtime as needed by the tasks. Relevant details can be found in [9].

IV. ADAPTIVE SCHEDULING STRATEGY

The objective of our adaptive scheduling technique is to minimize the execution time of the application by dynamically repartitioning and redistributing active SIMD tasks among available PEs in the system. We use a centrally controlled scheme based on our system architecture. PEs are required to record runtime statistics in specified registers and the *system controller* (SC) collects such information from individual PEs to make assignment decisions.

The target system is modeled as an undirected graph $GT = (P, L)$, where a vertex $P_i \in P$ represents PE_i and an undirected link L_{ij} represents a bi-directional communication channel between PE_i and PE_j, for $i, j \in [1, p]$. Each PE_i is associated with a parameter $v(P_i)$ that records its functionality. The weight $w(L_{ij})$ on each L_{ij} denotes its minimum communication cost. The minimum communication cost is calculated based on the hops between PE_i and PE_j. Whenever there is contention for the route, the task with higher priority will be routed first. Also, communication jobs always have higher priority than computation jobs (i.e., PEs are always forced to route their messages even when they are busy). A priority is assigned to each task in TFG, which dynamically changes as scheduling

proceeds. This is because the dynamic assignment of PEs, dynamic partitioning and migration of tasks to multiple PEs, and the communication pattern and cost in our policy result in changes in the critical path. If two tasks are assigned to the same PE, then the communication is removed. The following is our policy to assign the priority.

Step 1. Find the first critical path (CP_1) in S. For many critical paths, choose one randomly.

Step 2. Assign priority 1 to the entry node in CP_1.

Step 3. Proceed along CP_1 and assign each task the priority of its predecessor plus one.

Step 4. Find the other parents of each task in CP_1 and insert them before this task; determine the priority of multiple parents by their location in subsequent critical paths.

Step 5. Find the next CP in S and repeat Steps 3-4 until all tasks are processed.

In the scheduling procedure, a task is said to be *READY* when all its inputs are available. A *QUALIFIED* PE for a task is defined as a PE that supports all the operation types $\pi(S_i)$ in the task S_i. We define the *average execution time* (AET) for each task S_i on the p_i processor as $AET(S_i, p_i) = (O_i / p_i) + T_c(S_i, p_i)$, where $T_c(S_i, p_i)$ is the communication overhead caused by distributing task S_i to p_i PEs as compared to scheduling on one PE. A distinct advantage of our strategy is that the SC is fully aware of the performance of each PE and system structure. In order to simplify the scheduling algorithm and reduce the runtime scheduling overhead, the following policies are applied.

- At any given time, tasks do not share the same PE.
- Pieces of an MIMD task can only be scheduled to immediate neighbors.
- SIMD tasks have a higher priority than MIMD tasks (the former also can be assigned to immediate neighbors).

A. Loop Partitioning

Loop-based partitioning is the basis of our adaptive parallelization. In our current design, flow dependence is allowed inside an assigned iteration. We distinguish the following cases.

FOR loops without cross-iteration dependence

Assume that the total number of PEs available to the loop is α and the total number of iterations is L. The loop space is split into α groups of size $\lfloor L/\alpha \rfloor$ or $\lceil L/\alpha \rceil$. Each PE gets such a group and the corresponding data set. These loops map naturally to the SIMD mode and no communication is required. *FOR* loops without both flow and cross-iteration dependences are treated the same way.

FOR loops with cross-iteration dependence [10]

We assume that the distance between successive data dependent iterations is w. Fig. 3 shows a simple example with $w = 2$. Let the total iteration space L in the loop be a multiple of w and the loop can be divided into w partitions. The i^{th} partition contains the iterations $l_0 + i + k*w$, where $k \in [0, L/w-1]$ and l_0 is the starting point of the loop index, for $i \in [0, w-1]$. Each partition is then further divided into α groups of size $\lceil L/(\alpha*w) \rceil$ or $\lfloor L/(\alpha*w) \rfloor+1$ of continuous iterations. Each PE gets such a group and the corresponding data set. By distributing data this way, data communication is restricted between two neighboring groups.

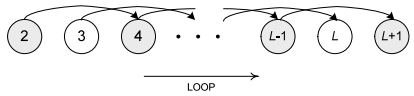

Fig. 3. Cross-iteration dependence example.

Heterogeneous loops

Each iteration in heterogeneous loops has different FP operations. Let f_i be the number of FP operations in the i^{th} iteration and $f_i = a*i + b$. Such a loop can be transformed into a homogeneous loop by combining the i^{th} and $(n-i-1)^{th}$ iterations into a new loop with a constant number of operations [11]. Then the above partitioning techniques can be applied to these loops.

Other loops that can be transformed into *FOR* loops (e.g., *WHILE* loops) are treated similarly.

B. PE Searching

The distance between PEs is one of the two critical parameters to the communication cost. Thus, the order in which we search for one or more candidate PEs is an important step affecting the overall mapping performance. Based on the HERA organization and interconnection network, we propose column-oriented PULSE searching as shown in Fig. 4 (a). The motivations are:

- The column buses in HERA can be used to broadcast instructions in SIMD and M-SIMD.
- In this search pattern, the distance between two adjacent stops is always one.
- One port of the data memory of each PE in HERA is shared with its immediate neighbors to the west and south. By selecting candidate PEs in the PULSE pattern, there is a large chance that we can save on communication time. For example, consider the TFG shown in Fig. 4 (b) and assume that S_1 and S_2 have already been executed by PE_3 and PE_4, respectively. We assume that S_3 has to be mapped to a PE other than PE_3 or PE_4. If S_3 is assigned to PE_8, then the communication distance is only one hop (PE_4 is asked to get the result of S_1 from PE_3, and PE_8 can access both results from S_1 and S_2 in the local memory of PE_4).

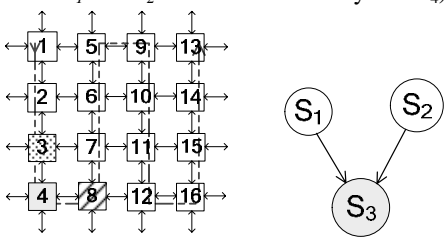

Fig. 4. PE searching path.

Assume that the numbers of PEs assigned to tasks S_i and S_j are p_i and p_j, respectively, and $x = min\{p_i, p_j\}$. The objective function to be minimized in this step is the communication time among tasks:

$$T_{com} = \sum_{i,j=1}^{n} \max\{[D_{ij}/(\lambda * x) + T_{ini}] * H(i,j) + T_{cflit}\}$$

where D_{ij} is the amount of data in bytes communicated between these two groups of PEs, λ is the transfer speed in bytes/second between two immediate neighbors, T_{ini} is the overhead to initialize the transfer, $H(i, j)$ is the number of hops between two communicating PEs and T_{cflict} is the routing delay caused by data collisions. In order to reduce the collision and communication costs, data locality is taken into account when mapping tasks to PEs.

C. Dynamic Adaptive Scheduling Algorithm

The scheduling procedure is as follows.

1. WHILE $Pt \neq \varnothing$ and $S \neq \varnothing$ DO // *Pt is the set of available PEs*
2. {
3. **Find out** CP_1, CP_2, CP_3, …, until all tasks are included;
4. **Calculate** the priority of each task; **sort** S in the decreasing order of priority;
5. **Pick** the top task $S_i \in S$;
6. **IF** $\varepsilon(S_i)$ = *MIMD* and *READY* **THEN**
7. { $\forall$ $PE_i \in Pt$, i=1,…, pt,
8. **Search** for the PEs that hold the inputs of S_i;
9. **IF** ($p_i \neq 0$) **THEN** // p_i : qualified PEs in Pt for S_i
10. {**Find** q PEs in these p_i PEs with the minimum $AET(S_i, q)$ and **assign** these q PEs to S_i;
11. $\forall$ PE_i, i=1,…, q, **set** $cm(PE_i)$=*MIMD*;}
12. **ELSE**
13. {**Reconfigure** one or more PEs into a qualified PE for S_i;
14. **Assign** this PE to S_i;
15. **Set** $cm(PE)$=*MIMD*; }
16. **Delete** S_i from S; }
17. **ELSEIF** $\varepsilon(S_i)$ = *MIMD* and not *READY* **THEN**
18. { **Pick** the next task in S;
19. **GOTO** 6;}
20. **ELSEIF** $\varepsilon(S_i)$ = *SIMD* and *READY* **THEN**
21. { **Assign** all qualified PEs $\in$ *Pt to* S_i;
22. $\forall$ qualified $PE_i \in Pt$, **Set** $cm(PE_i)$=*SIMD*;
23. **Delete** S_i from S; }
24. **ELSEIF** $\varepsilon(S_i)$ = *SIMD* and not *READY* **THEN**
25. { $\forall$ S_k , $k \in [1, m]$, that hold the inputs of S_i,
Add the qualified PEs $\in Pt$ to these m tasks and **repartition** the tasks so as to
$AET'(S_1, p_1)$=$AET'(S_2, p_2)$=…=$AET'(S_k, p_k)$,
where $AET'(S_k, p_k) = (O'_k / p_k) + T_r(p_k)$, and O'_k is the remaining number of operations for task S_k and $k \in [1, m]$;
Set $cm(PE_k) = \varepsilon(S_k)$, if PE_k is assigned to process S_k ;}
26. **WHILE** (not all the m tasks are finished) *//Some other PEs may become idle during waiting*
27. {**Reconfigure** unqualified PEs for S_i into qualified PEs; //Partial hardware reconfiguration
28. **GOTO** 25; }
29. }
30. }

V. Application Study

Singular Value Decomposition (SVD) has been chosen as a computation-intensive algorithm to evaluate the performance. This algorithm is abundant in nested loops and for square matrices it requires more than 20 times the number of FP operations in LU factorization. Given a matrix $A \in R^{M \times N}$, SVD factorizes A in the form $A = U\sigma V^T$; $U = [u_1, \ldots, u_M]$ is an M x M orthogonal matrix, $V = [v_1, \ldots, v_N]$ is an N x N orthogonal matrix, and $\sigma = diag(\sigma_1, \ldots, \sigma_r)$ is a diagonal matrix with $r = min(M, N)$ and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$. The σ_i*'s* are known as the *singular values* of A, and vectors u_i and v_i are the i^{th} *left singular* and *right singular vector*, respectively. The algorithm studied here is based on Golub-Kahan [12], which is one of the most commonly employed implementations. In our experiment, a modified sequential description of the SVD algorithm at http://www.netlib.org/ was analyzed and then divided into the tasks summarized in Table 1.

In-house developed single-precision FP units were chosen and the Annapolis WILDSTAR II-PCI board with two XC2V6000-5 FPGAs [13] was used. The system was clocked at 125MHz. We first evaluated the effect of runtime reconfiguration (RTR). The chosen numbers of PEs for the four steps in Table 1 were 36, 42, 42 and 42, respectively. Five randomly generated dense matrices were used; the execution times are shown in Fig. 5. The results prove that partial dynamic RTR system reconfiguration can improve the performance significantly; the speedup increases with the matrix size. Fig. 6 shows a performance comparison of our adaptive scheduling with an alternative dynamic scheduling where all the available PEs are assigned to each task when it is ready (fixed through the task lifetime). Our adaptive scheduling, enforced by RTR, performs much better than the naive scheduling strategy. It was observed during the experiments that adaptive scheduling greatly reduces the effect of data dependences and the idle times of PEs. It effectively shortens the critical path, which largely determines the execution time of the entire application. An increase in the number of PEs improves dramatically the execution of the largest SIMD tasks (three-nested loops), and the overheads of loop partitioning and scheduling become less significant for larger matrix sizes.

VI Conclusions

Our MPoPCs based on multimillion-gate FPGAs provide low-cost, low-risk and high-performance computing platforms for floating-point array-intensive applications. Our dynamic adaptive scheduling strategy takes advantage of HERA's mixed-mode parallelism. The good performance results with the SVD algorithm prove the effectiveness of our dynamic approach and suggest more benefits for increased matrix size compared to fixed PE allocations.

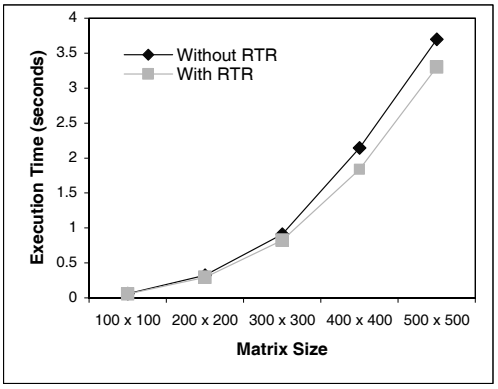

Fig. 5. Execution times with and without partial runtime reconfiguration (RTR).

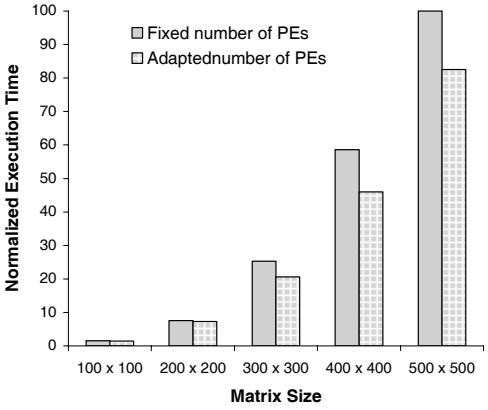

Fig. 6. Normalized execution times for our strategy and naïve dynamic scheduling.

Table 1. Task Information for SVD

Step	Tasks	Function Description	Comp. Mode	Comp. Cost	Comm. Vol.	FP Op. Types
Householder reduction ($i = 1, \ldots, N$)	1	Calculate s	SIMD	$4M$	1	+, /, *
	2	Calculate g, h, and A_{ii}	MIMD	4	1	Sq., /, -, *
	3	Update A	SIMD	$4MN$	MN	+, *, /
	4	Scale A	SIMD	$2M$	M	+, *
	5	Calculate w	MIMD	1	1	*
	6	Calculate s	SIMD	$3(N-i-1)$	$N-i-1$	+, *, /
	7	Calculate g, h, and A_{il}	MIMD	4	1	Sq., /, -, *
	8	Calculate rv1	MIMD	N	N	/
	9	Update A	SIMD	$(M-i-1)(N-i-1)^2$	MN	*, +
	10	Scale A	SIMD	$N-i-1$	N-i-1	*, +
	11	Find $max\ \{anorm, \lvert w\rvert + \lvert rv1\rvert\}$	MIMD	$2^{\log N}$	1	+
Accumulation of right-hand transformations ($i = 1, \ldots, N$)	12	Calculate v	SIMD	$2(N-i-1)$	$N-i-1$	/
	13	Calculate s and v	SIMD	$4(N-i-1)^2$	$(N-i-1)^2$	+, *
Accumulation left-hand transformation ($i = 1, \ldots, min(M,N)$)	14	Update A with s, f conditionally	M-SIMD	$2(m-i-1)$[n-i+1+2($M-i$)] or ($M-i$)	$2MN$	+, *, /
Diagonalization of bidiagonal form ($i = 1, \ldots, N$)	15	Check *flag*	MIMD	$N-i$	$N-i$	None
	16	Update A	SIMD	$6i(M-1)$	MN	+, -, *, /
	17	Make singular values nonnegative	SIMD	N	N	*
	18	Calculate f	MIMD	20	4	+, -, *, /
	19	QR transformation	SIMD	$12(i-1)N$	$2MN$	+, -, *, /

REFERENCES

1. R. Tessier and W. Burleson, "Reconfigurable computing and digital signal processing: a survey," *Journal VLSI Signal Processing*, May/June 2001.
2. S. Roy and P. Banerjee, "An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design," *IEEE Trans. Computers*, vol. 54, pp. 886-896, July 2005.
3. K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," *12th ACM/SIGDA FPGA*, 2004.
4. C. Ebeling, C. Fisher, G. Xing, M. Shen and H, Liu, "Implementing an OFDM receiver on the RaPiD reconfigurable architecture," *IEEE Trans. Computers*, vol. 53, pp. 1436-1448, Nov. 2004.
5. X. Wang and S. G. Ziavras, "Exploiting mixed-mode parallelism for matrix operations on the HERA architecture through reconfiguration," *IEE Proceedings, Computers & Digital Techniques,* accepted for publication.
6. S. F. Hummel, E. Schonberg and L. E. Flynn, "Factoring, a scheme for scheduling parallel loops," C*ommu. ACM*, vol. 35, Aug. 1992.
7. C. D. Polychronopoulos and D. Kuck, "Guided self-scheduling: a practical scheduling scheme for parallel supercomputers," *IEEE Trans. Computers*, vol. 36, pp. 1425-1439, Dec. 1987.
8. A. Ahmedsaid, et. al., "Improved SVD systolic array and implementation on FPGA," *IEEE FPT*, 2003.
9. X. Wang and S. G. Ziavras, "A framework for dynamic resource management and scheduling on reconfigurable mixed-mode multiprocessors," *IEEE Intern. Conf. on Field-Programmable Technology (FPT'05)*, Singapore, Dec. 11-14, 2005.
10. Ding-Kai Chen, J. Torrellas and Pen-Chung Yew, "An efficient algorithm for the run-time parallelization of DOACROSS loops," *Supercomputing*, pp. 518–527, Nov. 1994.
11. M. Cierniak, W. Li and M. J. Zaki, "Loop scheduling for heterogeneity," *IEEE Intern. Symp. High-Performance Distributed Computing,* pp. 78-85, 1995.
12. G. H. Golub and W. Kahan, "Calculating the singular values and pseudoinverse of a matrix," *SIAMJ Numer. Anal.,* vol. 2, 1965.
13. Annapolis Microsystems, Inc, http://www.annapmicro.com.