UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,

Petitioner

v.

FG SRC LLC,

Patent Owner

CASE NO.: 2020-01449
PATENT NO. 7,149,867

DECLARATION OF EILEEN D. MCCARRIER

Mail Stop **PATENT BOARD**
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450

Alexandria, VA 22313-1450

I, Eileen D. McCarrier, declare as follows:

1. I am currently Manager of Research Services at Pillsbury Winthrop Shaw Pittman LLP, where I have worked as a reference librarian and manager of research services for 26 years.

2. I make this declaration based on my own personal knowledge, including my knowledge of library science practices and the evidence cited herein.

3. I earned a Master of Arts degree in Library Science from the University of Wisconsin-Madison in 1980, and I have worked as a law librarian for 40 years.

4. I have prepared this declaration in connection with the above-captioned *inter partes* review ("IPR") proceeding for which I obtained library copies of previously filed Exhibits 1003 and 1004. Each of these articles was published by the Institute of Electrical and Electronics Engineers, Inc. ("IEEE"), a widely recognized publisher of technical papers spanning a broad range of technologies including electronics, electrical engineering, telecommunications, computing, and more. The IEEE publishes thousands of conference papers every year, including by making them publicly available via its Xplore digital library. The IEEE's collection of publications is recognized within the reference library field as an authoritative source of consolidated published papers in electrical engineering, computer science, and related fields.

5. The primary holders of the original printed versions of IEEE conference proceedings from the 1996–2000 era are generally university libraries across the United States. However, at the time of the drafting of this declaration, safety protocols relating to COVID-19 continue to restrict the ability of members of the public to physically access many of these institutions. To accommodate these safety procedures, the library copies described below were physically located by library staff, digitally scanned, and then emailed directly to me.

Library copies of Zhang, Exhibit 1003

6. X. Zhang et al., Architectural Adaptation for Application-Specific Locality Optimizations, IEEE (1997), published in the Proceedings of the International Conference on Computer Design - VLSI in Computers and Processors (IEEE, October 12–15, 1997), 150–156 ("Zhang") was filed as Exhibit 1003 with the petition in the above-captioned IPR proceeding.

7. A true and correct copy of Zhang obtained from the Library of the Missouri University of Science and Technology, Rolla, Missouri, part of the University of Missouri system, is attached as Appendix EDM01. I personally received this copy as a pdf from the interlibrary loan department of C. L. Wilson Library.

Library copies of Gupta, Exhibit 1004

8. R. Gupta, Architectural Adaptation in AMRM Machines, Proceedings of the IEEE Computer Society Workshop on VLSI 2000 (IEEE, April 27–28, 2000), 75–79 ("Gupta") was filed as Exhibit 1004 with the above-captioned IPR petition.

9. A true and correct copy of Gupta obtained from the Georgia Tech Library, Georgia Institute of Technology is attached as Appendix EDM02. I personally received this copy as a pdf from the library's Interlibrary Loan Office.

I declare under penalty of perjury that the foregoing is true and correct.

Date: March 31, 2021

Eileen D. McCarrier

Appendix EDM01

Proceedings

International Conference on Computer Design

VLSI in Computers and Processors

Proceedings

International Conference on Computer Design

VLSI in Computers and Processors

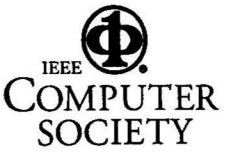
October 12-15, 1997

Austin, Texas

Sponsored by

IEEE Computer Society Technical Committee on Design Automation

IEEE Circuits and Systems Society



Los Alamitos, California

Washington • Brussels • Tokyo

IEEE Computer Society Order Number PR08026
ISBN 0-8186-8026-X
ISBN 0-8186-8207-8 (case)
ISBN 0-8186-8208-6 (microfiche)
IEEE Order Plan Catalog Number 97CB36149
ISSN 1063-6404

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1-714-821-8380
Fax: + 1-714-821-4641
E-mail: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1-908-981-1393
Fax: + 1-908-981-9667
mis.custserv@computer.org

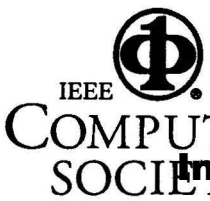
IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: + 32-2-770-2198
Fax: + 32-2-770-8505
euro.ofc@computer.org

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: + 81-3-3408-3118
Fax: + 81-3-3408-3553
tokyo.ofc@computer.org

Editorial production by Ian Torwick

Cover design by Joseph Daigle/Studio Productions

Printed in the United States of America by Technical Communication Services



11-25-97
ah

Table of Contents

ICCD '97 Conference Program

1997 Technical Program

Concurrent Sessions 3.2

Concurrent Session 3.3

Architectural Adaptation for Application-Specific Locality Optimizations

Xingbin Zhang* Ali Dasdan* Martin Schulz† Rajesh K. Gupta‡ Andrew A. Chien*

*Department of Computer Science
University of Illinois at Urbana-Champaign
{zhang,dasdan,achien}@cs.uiuc.edu

†Institut für Informatik
Technische Universität München
schulzm@informatik.tu-muenchen.de

‡Information and Computer Science, University of California at Irvine
rgupta@ics.uci.edu

Abstract

We propose a machine architecture that integrates programmable logic into key components of the system with the goal of customizing architectural mechanisms and policies to match an application. This approach presents an improvement over traditional approach of exploiting programmable logic as a separate co-processor by preserving machine usability through software and over traditional computer architecture by providing application-specific hardware assists. We present two case studies of architectural customization to enhance latency tolerance and efficiently utilize network bisection on multiprocessors for sparse matrix computations. We demonstrate that application-specific hardware assists and policies can provide substantial improvements in performance on a per application basis. Based on these preliminary results, we propose that an application-driven machine customization provides a promising approach to achieve high performance and combat performance fragility.

1 Introduction

Technology projections for the coming decade [1] point out that system performance is going to be increasingly dominated by intra-chip interconnect delay. This presents a unique opportunity for programmable logic as the interconnect dominance reduces the contribution of per stage logic complexity on performance and the marginal costs of adding switching logic in the interconnect. However, the traditional *co-processing* architecture of exploiting programmable logic as a specialized functional unit to deliver a specific application suffers from the problem of machine retargetability. A system generated using this approach typically can not be retargeted to another application without repartitioning hardware and software functionality and reimplementing the co-processing hardware. This retargetability problem is an obstacle toward exploiting programmable logic for general purpose computing.

We propose a machine architecture that integrates programmable logic into key components of the system with the goal of customizing architectural mechanisms and policies to match an application. We base our design on the premise that communication is already critical and getting increasingly so [17], and flexible interconnects can be used to replace static wires at competitive performance [6, 9, 20]. Our approach presents an improvement over co-processing by preserving machine usability through software and over traditional computer architecture by providing application-specific hardware assists. The goal of application-specific hardware assists is to overcome the rigid architectural choices in modern computer systems that do not work well across different applications and often cause substantial performance fragility. Because performance fragility is especially apparent on memory performance on systems with deep memory hierarchies, we present two case studies of architectural customization to enhance latency tolerance and efficiently utilize network bisection on multiprocessors. Using sparse matrix computations as examples, our results show that customization for application-specific optimizations can bring significant performance improvement (10X reduction in miss rates, 100X reduction in data traffic), and that an application-driven machine customization provides a promising approach to achieve robust, high performance.

The rest of the paper is organized as follows. Section 2 presents our analyses of the technology trends. Section 3 describes our proposed architecture and the project context. We describe our case studies in Section 4 and discuss related work in Section 5. Finally, we conclude with future directions in Section 6.

1063-6404/97 $10.00 © 1997 IEEE

2 Background

Technology projections for the coming decade point out a unique opportunity of programmable logic. However, the traditional co-processing approach of exploiting programmable logic suffers from the problem of machine retargetability, which limits its use for general purpose applications.

2.1 Key Technology Trends

The basis for architectural adaptation is in the key trends in the semiconductor technology. In particular, the difference in scaling of switching logic speed and interconnect delays points out increasing opportunities for programmable logic circuits in the coming decade. Projections by the Semiconductor Industry Association (SIA) [1] show that on-chip system performance is going to be increasingly dominated by interconnect delays. Due to these interconnect delays, the on-chip clock periods will be limited to about 1 nanosecond, which is well above the projections based on channel length scaling [1]. Meanwhile, the unit gate delay (inverter with fanout of two) scales down to 20 pico-seconds. Thus, modern day control logic consisting of 7-8 logic stages per cycle would form less than 20% of the total cycle time. This clearly challenges the fundamental design trade-off today that tries to simplify the amount of logic per stage in the interest of reducing the cycle time [14]. In addition, this points to a sharply reduced marginal cost of per stage logic complexity on the circuit-level performance.

The decreasing delay penalty for (re)programmable logic blocks compared to interconnect delays also makes the incorporation of small programmable logic blocks attractive even in custom data paths. Because the interconnect delays scale down much more slowly than transistor switching delays, in the year 2007, the delay of the average length inter-connect (assuming an average interconnect length of 1000X the pitch) would correspond to approximately three gate delays (see [5] for a detailed analysis). This is in contrast to less than half the gate delay of the average interconnect in current process technology. This implies that due to purely electrical reasons, it would be preferred to include at least one inter-connect buffer in a cycle time. This buffer gate when combined with a weak-feedback device would form the core of a storage element that presents less than 50% switching delay overhead (from 20ps to 30ps), making it performance competitive to replace static wires with flexible interconnect.

In view of these technology trends and advances in circuit modeling using hardware description languages (HDLs) such as Verilog and VHDL, the process of hardware design is increasingly a language-level activity, supported by compilation and synthesis tools [11, 12]. With these CAD and synthesis capabilities, programmable logic circuit blocks are beginning to be used in improving system performance.

2.2 Co-processing

The most common architecture in embedded computing systems to exploit programmable logic can be characterized as one of *co-processing*, i.e., a processor working in conjunction with dedicated hardware assists to deliver a specific application. The hardware assists are built using programmable circuit blocks for easy interpretation with the predesigned CPU. Figure 1 shows the schematic of a co-processing architecture, where the co-processing hardware may be operated under direct control of the processor which stalls while the dedicated hardware is operational [10], or the co-processing may be done concurrently with software [13]. However, a system generated using this approach typically can not be retargeted to another application without repartitioning hardware and software functionality and reimplementing the co-processing hardware even if the macro-level organization of the system components remains unaffected. This presents an obstacle of exploiting programmable logic for general-purpose computing even though technology trends make it possible to do so.

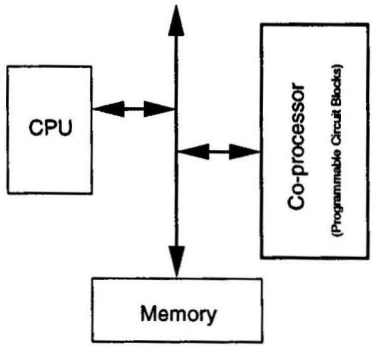


Figure 1. A co-processing Architecture

3 Architectural Adaptation

We propose an architecture that integrates small blocks of programmable logic into key elements of a baseline architecture, including processing elements, components of the memory hierarchy, and the scalable interconnect, to provide *architectural adaptation* - the customization of architectural mechanisms and policies to match an application. Figure 2 shows our architecture. Architectural adaptation can be used in the bindings, mechanisms, and policies on the interaction of processing, memory, and communication resources while keeping the macro-level organization the

same and thus preserving the programming model for developing applications. Depending upon the hardware technology used and the support available from the runtime environment, this adaptation can be done statically or at runtime.

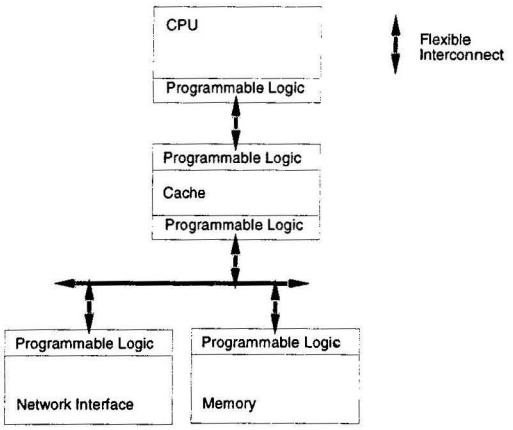


Figure 2. An Architecture for Adaptation

Architectural adaptation provides the mechanisms for application-specific hardware assists to overcome the rigid architectural choices in modern computer systems that do not work well across different applications and often cause substantial performance fragility. In particular, the integration of programmable logic with memory components enables application-specific locality optimizations. These optimizations can be designed to overcome long latency and limited transfer bandwidth in the memory hierarchy. In addition, because the entire application remains in software while the underlying hardware is adapted for system performance, our approach improves over co-processing architectures by preserving machine usability through software. The main disadvantage of our approach is the potential increase on system design and verification time due to the addition of programmable logic. We believe that the advances in design technology will address the increase of logic complexity.

3.1 Project Context

Our study is in the context of the MORPH [5] project, a NSF point design study for Petaflops architectures in the year 2007 technology window. The key elements of the MORPH (**M**ultiprocess**O**r with **R**econfigurable **P**arallel **H**ardware) architecture consists of processing and memory elements embedded in a scalable interconnect. With a small amount of programmable logic integrated with key elements of the system, the proposed MORPH architecture aims to exploit architectural customization for a broad range of purposes such as:

- control over computing node granularity (processor-memory association)
- interleaving (address-physical memory element mapping)
- cache policies (consistency model, coherence protocol, object method protocols)
- cache organization (block size or objects)
- behavior monitoring and adaptation

As an example of its flexibility, MORPH could be used to implement either a cache-coherent machine, a non-cache coherent machine, or even clusters of cache coherent machines connected by put/get or message passing. In this paper, we focus on architectural adaptation in the memory system for locality optimizations such as latency tolerance.

4 Case Studies

We present two case studies of architectural adaptation for application-specific locality optimizations. On modern architectures with deep memory hierarchies, data transfer bandwidth and access latency differentials across levels of memory hierarchies can span several orders of magnitude, making locality optimizations critical for performance. Although compiler optimizations can be effective for regular applications such as dense matrix multiply, optimizations for irregular applications can greatly benefit from architectural support. However, numerous studies have shown that no fixed architectural policies or mechanisms, e.g., cache organization, work well for all applications, causing performance fragility across different applications. We present two case studies of architectural adaptation using application-specific knowledge to enhance latency tolerance and efficiently utilize network bisection on multiprocessors.

4.1 Experimental Methodology

As our application examples, we use the sparse matrix library `SPARSE` developed by Kundert and Sangiovanni-Vincentelli (version 1.3 available from `http://www.netlib.org/sparse/`), and an additional sparse matrix multiply routine that we wrote. This library implements an efficient storage scheme for sparse matrices using row and column linked lists of matrix elements as shown in Figure 3. Only nonzero elements are represented, and elements in each row and column are connected by singly linked lists via the `nextRow` and `nextCol` fields. Space for elements, which is 40 bytes per matrix element, are allocated dynamically in blocks of elements for efficiency. There are also several one dimensional arrays for storing the root pointers for row and column lists.

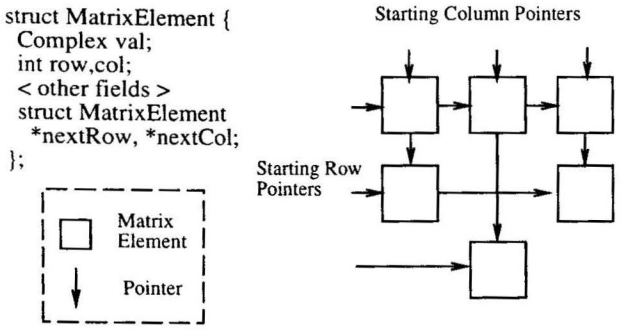


Figure 3. Sparse Library Data Structures

We perform cycle-based system-level simulation using a program-driven simulator based on MINT [22] that interprets program binaries and models configurable logic blocks behaviorly. The details of our simulation environment are presented in [4]. Table 1 shows the simulation parameters used. We report our empirical results for current day computer technologies and then use derivative metrics (such as miss rate) to extrapolate potential benefits for future computer systems which will exhibit much higher clock rates and memory sizes. We also manually translated the C routines modeling the customizable logic blocks into HardwareC [18] to evaluate their hardware cost in terms of space and cycle delays. (However, our recent work is focused on automatic translation of these routines to synthesizable blocks [19].)

	L1 Cache	L2 Cache
Line Size	32B or 64B	32B or 64B
Associativity	1	2
Cache Size	32KB	512KB
Write Policy	Write back + Write allocate	Write back + Write allocate
Replacement Policy	Random	Random
Transfer Rate	(L1-L2) 16B/5 cycles	(L2-Mem) 8B/15 cycles

Table 1. Simulation Parameters

4.2 Architectural Adaptation for Latency Tolerance

Our first case study uses architectural adaptation for prefetching. As the gap between processor and memory speed widens, prefetching is becoming increasingly important to tolerate the memory access latency. However, oblivious prefetching can degrade a program's performance by saturating bandwidth. We show two example prefetching schemes that aggressively exploit application access pattern information.

Figure 4 shows the prefetcher implementation using programmable logic integrated with the L1 cache. The prefetcher requires two pieces of application-specific information: the address ranges and the memory layout of the target data structures. The address range is needed to indicate memory bounds where prefetching is likely to be useful. This is application dependent, which we determined by inspecting the application program, but can easily be supplied by the compiler. The program sets up the required information and can enable or disable prefetching at any point of the program. Once the prefetcher is enabled, however, it determines what and when to prefetch by checking the virtual addresses of cache lookups to check whether a matrix element is being accessed.

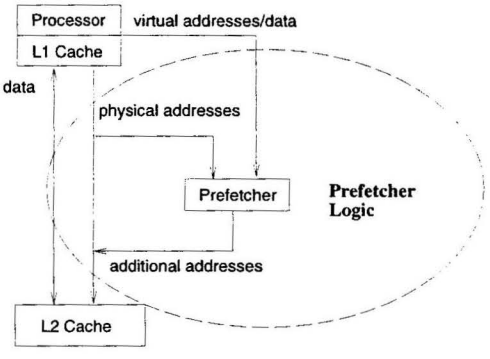


Figure 4. Organizations of Prefetcher Logic

The first prefetching example targets records spanning multiple cache lines and for our example, prefetches all fields of a matrix element structure whenever some field of the element is accessed. The pseudocode of this prefetching scheme for the sparse matrix example is shown below, assuming a cache line size of 32 bytes, a matrix element padded to 64 bytes, and a single matrix storage block aligned at 64-byte boundary. Prefetching is triggered only by read misses. Because each matrix element spans two cache lines, the prefetcher generates an additional L2 cache lookup address from the given physical address (assuming a lock-up free L2 cache) that prefetches the other cache line not yet referenced.

```
/* Prefetch only if vAddr refers to the matrix */
GroupPrefetch(vAddr,pAddr,startBlock,endBlock) {
 if (startBlock <= vAddr && vAddr < endBlock) {
  /* Determine the prefetch address */
  if (pAddr & 0x20) ptrLoc = pAddr - 0x20;
    else ptrLoc = pAddr + 0x20;
  <Initiate transfer of line at ptrLoc to L1 cache>
}}
```

The second prefetching example targets pointer fields that are likely to be traversed when their parent structures are accessed. For example, in a sparse matrix-vector mul-

tiply, the record pointed to by the nextRow field is accessed close in time with the current matrix element. The pseudocode below shows the prefetcher code for prefetching the row pointer, assuming a cache line size of 64 bytes. Again prefetching is triggered only by read misses, and the prefetcher generates an additional address after the initial cache miss is satisfied using the nextRow pointer value (whose offset is hardwired at setup time) embedded in the data returned by the L2 cache.[1]

```
/* Prefetch only if vAddr refers to the matrix */
PointerPrefetch(data,vAddr,startBlock,endBlock) {
 if (startBlock <= vAddr && vAddr < endBlock) {
  /* Get row pointer from returned cache line */
  ptrLoc = data [24];  /* row ptr offset = 24 */
  <Initiate transfer of elt at ptrLoc to L1 cache>
}}
```

Our prefetching examples are similar to the prefetching schemes proposed in [23], where they are shown to benefit various irregular applications. However, unlike [23], using architectural customization enables more flexible prefetching policies, e.g., multiple level prefetch, according to the application access pattern.

4.3 Architectural Adaptation for Bandwidth Reduction

Our second case study uses a sparse matrix-matrix multiply routine as an example to show architectural adaptation to improve data reuse and reduce data traffic between the memory unit and the processor. The architectural customization aims to send only used fields of matrix elements during a given computation to reduce bandwidth requirement using dynamic scatter and gather. Our scheme contains two units of logic, an address translation logic and a gather logic, shown in Figure 5.

The two main ideas are prefetching of whole rows or columns using pointer chasing in the memory module and packing/gathering of only the used fields of the matrix element structure. When the root pointer of a column or row is accessed, the gather logic in the main memory module chases the row or column pointer to retrieve different matrix elements and forwards them directly to the cache. The cache, in order to avoid conflict misses, is split into two parts: one small part acting as a standard cache for other requests and one part for the prefetched matrix elements only. The latter part has an application-specific management policy, and can be distinguished by mapping it to a reserved address space. The gather logic in pseudocode is shown below.

[1] As pointed in [23], the implementation of this prefetching scheme is complicated by the need to translate the virtual pointer address to physical address. We assume that the prefetcher logic can also access the TLB structure. An alternative implementation is to place the prefetcher logic in memory and forward the data of the next record to the upper memory hierarchy. This requires an additional group translation table [23] for address translation.

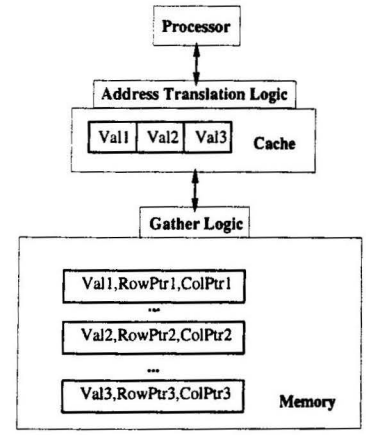


Figure 5. Scatter and Gather Logic

```
/* Row gather: pAddr is the start of a row */
Gather(pAddr) {
 chaseAddr = pAddr;
 while(chaseAddr) {
  forward chaseAddr->val
  forward chaseAddr->row
  chaseAddr=virtual-to-physical(chaseAddr->nextRow)
}}
```

Because the data gathering changes the storage mapping of matrix elements, in order not to change the program code, a translate logic in the cache is required to present "virtual" linked list structures to the processor. When the processor accesses the start of a row or column linked list, a prefetch for the entire row or column is initiated. Because the target location in the cache for the linked list is known, instead of returning the actual pointer to the first element, the translate logic returns an address in the reserved address space corresponding to the location of the first element in the explicitly managed cache region. In addition, when the processor accesses the next pointer field, the request is also detected by the translate logic, and an address is synthesized dynamically to access the next element in this cache region. The translate logic in pseudocode is shown below.

```
Translate(vAddr, pAddr, newPAddr) {
 /* check if accessing start of a row */
 if(startRowRoot<=vAddr && vAddr<=endRowRoot)
   <Initiate prefetch and return row location>
 /* Similarly for column roots */
 ...
 /* Accessing packed rows */
 if(startPackedRows<=pAddr && pAddr<=endPackedRows){
  off = pAddr & 63; /* get field offset */
  if( off == 24 )   /* row ptr. at offset 24 */
   return pAddr + 64; /* synthesize next addr. */
  else { /* Two fields at off1 and 2 are packed
            at new_off1 and 2 of new layout */
   if(off < off_2) new_off = off-(off1-new_off1);
     else new_off = off-(off2-new_off2);
```

```
  return (pAddr>>6) * PACKED_SIZE + new_off; }}
/* Similarly for packed columns */
}
```

As in the case of dense matrix-matrix multiply, blocking of the matrices in the cache is essential to achieve good performance. However, software blocking for sparse matrix-matrix multiply is not as effective as blocking in the dense case because the additional fields in a matrix element structure reduce the number of rows or columns that can fit in the cache, reducing reuse, and because elements are scattered in memory, increasing conflict misses. Figures 6 and 7 compare the miss rates and total data volume transfered between the memory and processor for the straightforward implementation, a pure software blocking scheme, customized scatter and gather, and customized logic with cache bypass for the result matrix (for two 460x460 matrices with 21660 non-zero elements each). The figures show that the hardware assists provided by customization significantly improve the software blocking by reducing row and column footprints to improve reuse. The final results show a 10X in read miss rates and 100X reduction in data volume compared to the straightforward implementation. The implementation costs of these units are less than 1000 in LSI logic 10K family gates and around 1500 in FPGA CLBs, requiring delays of about 3 cycles.

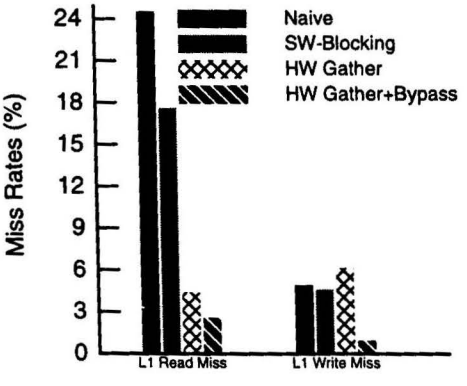


Figure 6. Cache miss rates of different implementations of sparse matrix-matrix multiply.

5 Discussion and Related Work

Traditional computer architectures are designed for best machine performance averaged across applications. Due to the static nature of such architectures, such machines are limited in exploiting an application characteristics unless it is common for a large number of applications. Since no single machine organization fits all applications, the delivered performance is often only a small fraction of the peak machine performance (frequently less than a tenth [5]). Therefore, we believe that there are significant opportunities for application-specific architectural adaptation. In this paper, we have demonstrated mechanisms for latency hiding and required bandwidth reduction that leverage small hardware support as well as do not change the programming model. Following the same methodology, we can build such assists for other applications as well. Among other examples that applications can benefit from are mechanisms for recognition of working set size for a given application that can be used to alter cache update policies or even use a synthesized small victim cache [16], and mechanisms for monitoring access patterns and conflicts in the caches or memory banks and reconfiguring the assists according to these patterns and conflicts. In summary, we expect an adaptable machine to have a large number of application-specific assists that alter architectural mechanisms and policies in view of application characteristics. The application developer with the help of compilation tools selects appropriate hardware assists to customize the machine to match the application without having to repartition the system functionality or rewrite the application.

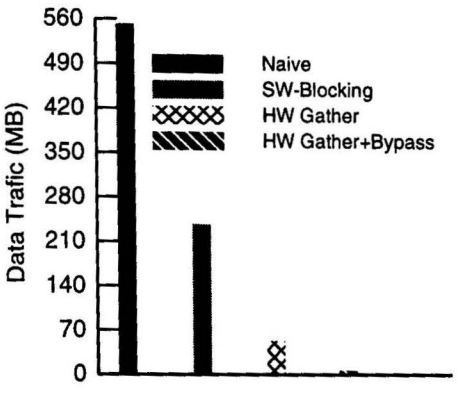


Figure 7. Data traffic volume of different schemes. (Total size of non-zeros, 1.35 MB)

There are other approaches for locality optimizations. Researchers have proposed processor-in-memory (PIM) [2, 3, 7, 8, 21] as a solution for solving the latency and bandwidth limitations of the memory hierarchy. We rely on more traditional processor-memory structures with customizable components, which we believe will yield a machine with more accessible performance than an organization in which processors are accessing primarily their local on-chip memory, particularly for irregular applications. In addition, by adding a small amount of programmable logic to the memory units, we can yield some benefits of having computational elements within the memory. Researchers have also proposed various specific architectural mechanisms for locality optimizations, for instance, group prefetching [23]

and informing memory operations [15], with mechanism-specific implementations. As shown in our case studies, we believe that integrating programmable logic into memory components provide a flexible implementation framework for these mechanisms.

6 Conclusions and Future Directions

The increasing dominance of interconnect delays on system performance makes it practical to integrate programmable logic into all key system components, enabling them to be customized to specific applications. As illustrated by two case studies, such architectural adaptation can provide flexible mechanisms for application-specific locality optimizations to combat the increasing gap between processor and memory speed. In addition, system co-design using this approach presents a way to utilize application-specific hardware much more effectively than co-processing architectures, where part of an application is implemented in hardware. Although not explored in this study, architectural adaptations in processor functional units might also be a promising approach to improve instruction throughput. As future directions, we are studying compilation tools in hardware/software co-design for the architectural assists, IC processing issues for implementing customizable system components, such as cache and scalable interconnects, as well as more, larger application studies.

Acknowledgments

The authors gratefully acknowledge support from National Science Foundation award ASC-96-34947. The authors are also supported by an NSF Young Investigator Award CCR-94-57809 and NSF CAREER Award MIP-95-01615.

References

[1] Seminconductor technology workshop conclusions. Roadmap, Semiconductor Industry Association, 1993.

[2] BORKAR, S., COHN, R., COX, G., GLEASON, S., GROSS, T., KUNG, H. T., LAM, M., MOORE, B., PETERSON, C., PIEPER, J., RANKIN, L., TSENG, P. S., SUTTON, J., URBANSKI, J., AND WEBB, J. iWarp: An integrated solution to high-speed parallel computing. In *Proc. IEEE Supercomputing '88* (1988), pp. 330–341. Orlando, Florida.

[3] BORKAR, S., COHN, R., COX, G., GROSS, T., KUNG, H. T., LAM, M., LEVINE, M., MOORE, B., MOORE, W., PETERSON, C., SUSMAN, J., SUTTON, J., URBANSKI, J., AND WEBB, J. Supporting systolic and memory communication in iWarp. In *Proc. 17th IEEE Int. Symp. on Comp. Arch.* (1990), pp. 70–81.

[4] CHIEN, A., DASDAN, A., GUPTA, R., AND ZHANG, B. Rapid Architectural Design and Validation Using Program-Driven Simulations. In *Symp. on High-level Design, Validation and Test (HLDVT)* (1996).

[5] CHIEN, A. A., AND GUPTA, R. K. MORPH: A system architecture for robust high performance using customization. In *Frontiers of Massive-Parallelism* (1996). Annapolis, Maryland.

[6] CHOI, L., AND CHIEN, A. A. The design and performance evaluation of the DI-multicomputer. *J. Parallel and Distributed Comput.* 36:119–143, 1996.

[7] DALLY, W., CHIEN, A., FISKE, J., FYLER, G., HORWAT, W., KEEN, J., LETHIN, R., NOAKES, M., NUTH, P., AND WILLS, D. The message driven processor: an integrated multicomputer processing element. In *Proc. IEEE Int. Conf. on Comp. Design: VLSI in Computers and Processors* (1992), pp. 416–19.

[8] DALLY, W. J., FISKE, J. A. S., KEEN, J. S., LETHIN, R. A., NOAKES, M. D., NUTH, P. R., DAVISON, R. E., AND FYLER, G. A. The message-driven processor. *IEEE Micro* (April 1992), pp. 23–39.

[9] DALLY, W. J., AND SONG, P. Design of a self-timed VLSI multicomputer communication controller. In *Proc. IEEE Int. Conf. on Comp. Design* (1987), pp. 230–4.

[10] ERNST, R., HENKEL, J., AND BENNER, T. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Design & Test of Computers* (Dec. 1993), pp. 64–75.

[11] GAJSKI, D., DUTT, N., WU, C. H., AND LIN, Y. L. *High-level Synthesis: Introduction to chip and system design.* Kluwer, 1992.

[12] GUPTA, R. K. *Co-Synthesis of Hardware and Software for Digital Embedded Systems.* Kluwer, 1995.

[13] GUPTA, R. K., AND MICHELI, G. D. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design & Test of Computers* (Sept. 1993), pp. 29–41.

[14] HENNESSY, J. L., AND JOUPPI, N. P. Computer technology and architecture: An evolving interaction. *IEEE Computer* (Sep 1991), pp. 18–29.

[15] HOROWITZ, M., MARTONOSI, M., MOWRY, T. C., AND SMITH, M. D. Informing memory operations: Providing memory performance feedback in modern processors. In *Proc. 23rd IEEE Int. Symp. on Comp. Arch.* (1996).

[16] JOUPPI, N. P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. the Int. Symp. on Comp. Arch.* (1990), pp. 364–73.

[17] KOGGE, P. M. Summary of the architecture group findings. In *PetaFlops Architecture Workshop (PAWS)* (Apr. 1996).

[18] KU, D., AND MICHELI, G. D. HardwareC - A Language for Hardware Design (version 2.0). CSL Tech. Rep. CSL-TR-90-419, Stanford Univ., Apr. 1990.

[19] LIAO, S. Y., TJIANG, S., AND GUPTA, R. K. An Efficient Implementation of Reactivity in Modeling Hardware in the CSYN Synthesis and Simulation Environment. In *Proc. Design Automation Conf.* (1997).

[20] SEITZ, C. L. Let's route packets instead of wires. In *Proc. 6th MIT Conf. on Advanced Research in VLSI* (1990), W. J. Dally, Ed., MIT Press, pp. 133–37.

[21] SPERTUS, E., GOLDSTEIN, S. C., SCHAUSER, K. E., VON EICKEN, T., CULLER, D. E., AND DALLY, W. J. Evaluation of mechanisms for fine-grained parallel programs in the J-Machine and the CM-5. In *Proc. 20th IEEE Int. Symp. on Comp. Arch.* (1993), pp. 302–313.

[22] VEENSTRA, J. E., AND FOWLER, R. J. MINT: A front end for efficient simulation of shared-memoy multiprocessors. In *Proc. 2nd Int. Workshop on Modeling, Analysis, and Simulation of Comp. and Telecommunication Syst. (MASCOTS)* (Jan. 1994), pp. 201–207.

[23] ZHANG, Z., AND TORRELLAS, J. Speeding up irregular applications in shared-memory multiprocessors: Memory binding and group prefetching. In *Proc. 22th IEEE Int. Symp. on Comp. Arch.* (1995).

Author Index

NOTES

NOTES

IEEE COMPUTER SOCIETY

Press Activities Board

IEEE Computer Society Publications

The world-renowned Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available in two formats: 100 percent original material by authors preeminent in their field who focus on relevant topics and cutting-edge research, and reprint collections consisting of carefully selected groups of previously published papers with accompanying original introductory and explanatory text.

Submission of proposals: For guidelines and information on Computer Society books, send e-mail to cs.books@computer.org or write to the Acquisitions Editor, IEEE Computer Society, P.O. Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314. Telephone +1 714-821-8380. FAX +1 714-761-1784.

IEEE Computer Society Proceedings

The Computer Society also produces and actively promotes the proceedings of more than 130 acclaimed international conferences each year in multimedia formats that include hard and softcover books, CD-ROMs, videos, and on-line publications.

For information on Computer Society proceedings, send e-mail to cs.books@computer.org or write to Proceedings, IEEE Computer Society, P.O. Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314. Telephone +1 714-821-8380. FAX +1 714-761-1784.

Additional information regarding the Computer Society, conferences and proceedings, CD-ROMs, videos, and books can also be accessed from our web site at http://computer.org/cspress

4/15/97

Appendix EDM02

Architectural Adaptation in AMRM Machines

Rajesh Gupta

Information and Computer Science
University of California, Irvine
Irvine, CA 92697
rgupta@ics.uci.edu

Abstract

Application adaptive *architectures use architectural mechanisms and policies to achieve system level performance goals. The AMRM project at UC Irvine focuses on adaptation of the memory hierarchy and its role in latency and bandwidth management. This paper describes the architectural principles and first implementation of the AMRM machine proof-of-concept prototype.*

1. Introduction

Modern computer system architectures represent design tradeoffs and optimizations involving a large number of variables in a very large design space. Even when successfully implemented for high performance, which is benchmarked against a set of representative applications, the performance optimization is only in an average sense. Indeed, the performance variation across applications and against changing data set even in a given application can easily be by an order of magnitude [1]. In other words, delivered performance can be less than one tenth of the system performance that the underlying hardware is capable of.

A primary reason of this fragility in performance is that rigid architectural choices related to organization of major system blocks (CPU, cache, memory, IO) do not work well across different applications.

Architectural Adaptivity provides an attractive means to ensure robust high performance. Architectural adaptation refers to the capability of a machine to support multiple architectural mechanisms and policies that can be tailored to application and/or data needs [2]. There are a number of places where architectural adaptivity can be used, for instance, in tailoring the interaction of processing with I/O, customization of CPU elements (e.g., splittable ALU resources) etc.

In view of the microelectronic technology trends that emphasize increasing importance of communication at all levels, from network interfaces to on-chip interconnection fabrics, communication represents the focus of our studies in architectural adaptation. In this context, memory system latency and bandwidth issues are key determining factors in performance of high performance machines because these can provide a constant multiplier on the achievable system performance [1]. Further, this multiplier decreases as the memory latency fails to improve as fast as processor clock speeds.

Consider a hypothetical machine with processing elements running at 2 GHz with eight-way superscalar pipelines. Assuming a typical 1 microsecond round-trip latency for a cache miss, this corresponds to about 16K instructions, with an average 30% or 4800 instructions being load/store. For a single-thread execution a miss rates as low as 0.02% reduces computing efficiency by as much as 50%. This points to a need for very low miss rates to ensure that high-throughput CPUs can be kept busy. A similar analysis of the bisection bandwidth concludes that active bandwidth management is required to reduce the need for communication and to increase the number of operations before a communication is necessary.

2. The AMRM Project

The Adaptive Memory Reconfiguration Management, or the AMRM, project at the University of California, Irvine aims to find ways to improve the memory system performance of a computing system. The basic system architecture reflects the view that communication is already critical and getting increasingly so [3], and flexible

0-7695-0534-1/00 $10.00 © 2000 IEEE

interconnects can be used to replace static wires at competitive performance in interconnect dominated microelectronic technologies [4,5,6]. The AMRM machine uses reconfigurable logic blocks integrated with the system core to control policies, interactions, and interconnections of memory to processing [7]. The basic machine architecture supports application-specific cache organization and policies, hardware-assisted blocking, prefetching and dynamic cache structures (such as stream, victim caches, stride prediction and miss history buffers) that optimize the movement and placement of application data through the memory hierarchy. Depending upon the hardware technology used and the support available from the runtime environment this adaptation can be done statically or at run-time. In the following section we describe a specific mechanism for latency management that is shown to provide significant performance boost for the class of applications characterized by frequent accesses to linked data structures scattered in the physical memory. This includes algorithms that operate on sparse matrices and linked trees.

2.1 Adaptation for Latency Management

Latency management refers to techniques for hiding long latencies of memory accesses by useful computation. Most common technique for latency hiding is by prefetching of data to the CPU. Prefetching is combined with smaller and faster (cache) memory elements that attempt to prefetch application context(s) rather than single data elements. The pointer-based accesses to data items in memory hierarchies typically yield poor results because the indirection introduces main memory and memory hierarchy latencies into the innermost computational loop. Techniques such as software prefetching (loop unrolling and hoisting of loads) do not adequately solve the problem, as prefetching at the processor leaves multi-level memory hierarchy latency in the critical path. Purely hardware prefetching [8] is also often ineffective because the address references generated by an application may contain no particular address structure. We use an application-specific prefetching scheme that resides in dedicated hardware at arbitrary levels of the memory hierarchy, in all of them, or to bypass them completely. This hardware performs application-specific prefetching, based on the address ranges of data structures used. When there is a reference to an address inside in this range, the prefetch hardware will prefetch the "next" element pointed to by the current element. The pointer field for the next element can be changed at runtime.

This prefetch hardware is combined, for some applications, with address translation and compaction hardware in the memory controller that works well with data structures that do not quite fit into a single cache line. The address translation is done transparently from the application using hardware assist *translate* in the cache controller and a corresponding hardware assist *gather* in the memory controller. Simulation results using this prefetch hardware show a 10X reduction in read miss rates and 100X reduction in data volume reduction for sparse matrix multiply operations [9].

3. The AMRM System Prototype

While AMRM simulation results continue to provide valuable insights into the space of architectural mechanisms and their effectiveness [7][9][10], a system implementation is needed to bring together different parts of the AMRM project (including compiler and runtime system algorithms to support adaptivity). The AMRM system prototype is divided into two phases. First phase consists of implementation of a board-level prototype; followed by a second phase single-chip implementation of the cache memory system. At the time of this writing, the first phase of the project prototype implementation has recently completed. The rest of this paper describes the system design and implementation of the Phase I prototype and its relationship to the ongoing second phase ASIC prototype.

The AMRM phase I prototype board is designed to serve two purposes. It can simulate a range of memory hierarchies for applications running on a host processor. The board supports configurability of the cache memory via an on-board FPGA-based memory controller. The board is also designed to be used, in future, as a complete system platform, with on-board memory serving as main memory, via a mezzanine card containing the Phase II AMRM ASIC implementation.

Whereas the goal of the AMRM prototype is to build an adaptive cache memory system, in general, the memory hierarchy performance cannot be de-coupled from the processor instruction set architecture, implementation, and compiler implementation. (This is particularly true of the CPU-L1 path that is often pipelined using non-blocking caches.) It would, therefore, be desirable to evaluate any proposed changes to the memory hierarchy for several processor architectures rather than being tied to one specific processor type and its

software. In order to be able to evaluate the effect with different processor architectures as well as to circumvent the implementation difficulties, our Phase I implementation attaches an additional memory hierarchy to a system through a standard peripheral bus. Thus, the board provides a PCI interface that allows a host processor to use the board as a part of its memory hierarchy. Applications running on the host processor are instrumented automatically using the AMRM compiler to use the memory on the AMRM board. Thus direct program execution can proceed on the host processor while the extra memory hierarchy is being exercised.

3.1 AMRM System Goals

One goal of the AMRM prototype system is that it be adaptable to many different memory hierarchy architectures. Another goal of the AMRM system is that it be useful for running real time program execution or even memory simulations. The latter is accomplished by making the AMRM memory available to the user and converting user program to access this memory "directly". The former is accomplished through the use of a sequence of address/command type requests "run" through various memory system configurations. The AMRM system is to be fast enough to support extensive execution or simulation.

A CPU interfaces to the reconfigurable AMRM memory system through the PCI bus. AMRM accepts CPU PCI requests for memory operations, issues them to the attached memory system, and sends back the data for memory read operations as well as memory access time information.

3.2 AMRM prototype architecture

Figure 1 shows the main components of the AMRM prototype board. It consists of a general 3-level memory hierarchy plus support for the AMRM ASIC chip implementing architectural assists with in the CPU-L1 datapath. The host interface is managed by a Motorola PLX 9080 processor. The FPGAs on the board contain controllers for the SRAM, DRAM and L1 cache. A 1 MB SRAM is used for tag and data store for the L1 cache. A total of 512 MB of DRAM is provided to implement part of the cache hierarchy (and also to serve as main memory by reloading the memory controller into the FPGA.)

The board implementation necessarily hard-wires certain parameters of the memory hierarchy. This includes the board's clock. In order to perform detailed and accurate simulation of diverse memory hierarchy configurations at any clock speed, a hardware "virtual" clock has been implemented as part of the performance monitoring hardware. Performance monitoring hardware primarily includes various event counters, which are memory-mapped and readable from the host processor. The "virtual clock" emulates a target system's clock: the clock rate is determined by the target system's memory hierarchy design and technology parameters. For example, the delay for an L1 cache hit, miss fetch etc in terms of virtual clocks can be configured by the host to emulate a given target cache design. Thus, the use of virtual clock allows us to simplify the hardware implementation. For instance, the tag and data stores of L1 cache can be a single RAM while the timing may reflect a design with two separate RAM's.

3.3 Command Interface to the AMRM Board

The memory hierarchy on the AMRM board can be used by an application running on the host processor by writing commands to specific addresses in the PCI address space. Each command consists of a set of four words that specify the operation (e.g., memory read/write, register read/write), the address of the location to access and data in case of a write. For read commands, a read response is generated and data is written into the host's memory.

For debugging purposes and to enable the cache to be flushed by the host, there are commands to access memory banks directly, i.e., without going through the caches. Commands are also available to read/write the status, configuration registers and performance counters.

The onboard command processors reads a command and launches its execution in the AMRM board. Data is read from the cache and sent back to the processor if it hits in the cache. It takes m Virtual Clock cycles. Otherwise it is requested from the next level in the hierarchy. Writes take n virtual clocks. Upon load command completion the data can be written into the system memory for access by the host processor. Both parameters n and m can be programmed under compiler control.

3.4 Virtual Clock System

The virtual clock system consists of a master clock counter, *Vtime*, the virtual clock signal, *Vclk*,

Ready inputs, and associate virtual clock generation logic. The *Ready* input from each major memory hierarchy module specifies that this unit has completed the current virtual clock cycle activities. A new *Vclk* edge/period is generated when all *Ready* inputs reach 1. The *Vtime* can be read out by the host processor to determine the current virtual time. It can also be automatically supplied to the CPU via host memory (as opposed to the AMRM board memory).

Each major unit in the memory hierarchy is designed to generate *Ready* and wait for the *Vclk*, when appropriate. In most cases the designs actually use *AutoReady* counters local to each unit which can be loaded with a programmable number of cycles. An *AutoReady* counter generates *Ready* using the *Vclk* while its output is non-zero. An idle unit not processing any requests also outputs a *Ready* signal every *Vclk*. For instance, Consider the AMRM Read command addressed to the on-board memory hierarchy. It includes a delay (ΔT) from the previous memory access. This delay is used to advance the virtual clock forward before starting the new access. This is accomplished by loading it into the *AutoReady* counter. This allows other memory hierarchy activity to proceed in parallel with CPU computation. For instance, a prefetch unit may be accessing memory during the ΔT-cycle delay.

3.5 AMRM chip functionality

The AMRM board provides for incorporation of an AMRM chip that uses an ASIC implementation of the AMRM cache assist mechanisms. It is positioned between L1 cache and the rest of the system and can be accessed in parallel with the L2 cache. It can thus accept and supply data coming from or going to the L1 cache. For instance, it may contain a write buffer or a prefetch unit to access L2. It also has access to the memory interface and thus can, for instance, prefetch from memory. The AMRM ASIC design is currently in progress. This chip will include a processor core with adaptive memory hierarchy. When plugged into the AMRM board, the ASIC will use onboard DRAM as main memory by simply reconfiguring the memory controller in FPGA1.

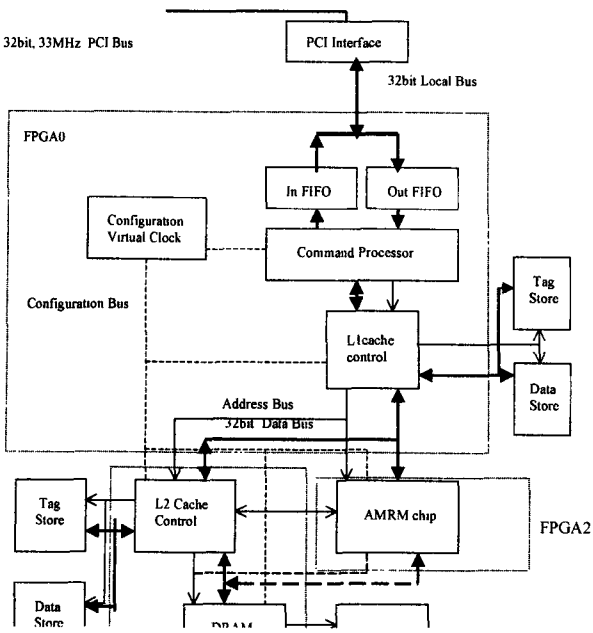


Figure 1: AMRM Phase I Prototype Board

4. Summary

Traditional computer system architectures are designed for best machine performance averaged across applications. Due to the static nature of these architectures, such machines are limited in exploiting application characteristics unless these are common for a large number of applications. Unfortunately, a number of studies have shown that no single machine organization fits all applications, therefore, often the delivered performance is only a small fraction of the peak machine performance. Therefore, we believe that there are significant opportunities for application-specific architectural adaptation. The focus of the AMRM project is on architectural adaptations that close the gap between processor and memory speed by intelligent placement of data through the memory hierarchy. Our current work has demonstrated performance gains due to adaptive cache organizations and cache prefetch assists. In future, we envision adaptive machines that provide a menu of application-specific assists that alter architectural mechanisms and policies in view of the application characteristics. The application developer, with the help of compilation tools, selects appropriate hardware assists to customize the machine to match application needs without having to rewrite the application.

5. Acknowledgement

The AMRM project was conceived through the joint effort of Andrew Chien, Alex Nicolau and Alex Veidenbaum. The team includes Prashant Arora, Chun Chang, Dan Nicolaescu, Rajesh Satapathy, Weiyu Tang, Xiao Mei Ji. The project is sponsored by DARPA under contract DABT63-98-C-0045.

6. References

[1] P. M. Kogge, "Summary of the architecture group findings," In *PetaFlops Architecture Workshop (PAWS)*, April 1996.

[2] A. A. Chien, R. K. Gupta, "MORPH: A System Architecture for Robust High Performance Using Customization," In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'96)*, October 1996.

[3] W. Wulf, S. McKee, "Hitting the memory wall: Implication of the obvious," In *Computer Architecture News*, 1995.

[4] C. L. Seitz, "Let's route packets instead of wires," In *Proceedings of the 6th MIT Conference on Advanced Research in VLSI*, W. J. Dally, editor, MIT Press, pp. 133-37, 1996.

[5] W. J. Dally, P. Song, "Design of a self-timed VLSI multicomputer communication controller," In *Proceedings of ICCD*, 1987.

[6] R. K. Gupta, "Analysis of Technology Trends: A Case of Architectural Adaptation in Custom Datapaths," In *Proceedings of SPIE session on Configurable Computing*, Boston, November 1998.

[7] AMRM Website: www.ics.uci.edu/~amrm.

[8] D. F. Zucker, M.J. Flynn, R. B. Lee, "A comparison of hardware prefetching techniques for multimedia benchmarks," *Technical report CSL-TR-95-683*, Computer Systems Laboratory, Stanford University, Stanford University, CA 94305, December 1995.

[9] X. Zhang, *et al*, "Architectural adaptation for memory latency and bandwidth optimization," In *Proceedings of ICCD*, October, 1997.

[10] A. Veidenbaum, *et al*, "Adapting Cache Line Size to Application Behavior," In *Proceedings of International Conference on Supercomputing (ICS)*, June 1999.

Proceedings

IEEE Computer Society Workshop on VLSI 2000

System Design for a System-on-Chip Era

27-28 April 2000 Orlando, Florida

Edited by
Asim Smailagic, Robert Brodersen and Hugo De Man

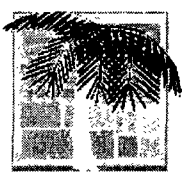
TER
TY

Sponsored by
IEEE Computer Society Technical Committee on VLSI

Proceedings

IEEE Computer Society Workshop on VLSI 2000

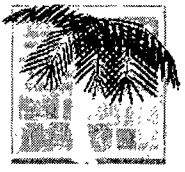
System Design for a System-on-Chip Era



Proceedings

IEEE Computer Society Workshop on VLSI 2000

System Design for a System-on-Chip Era

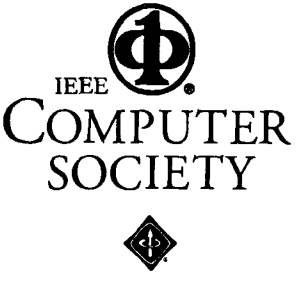


27–28 April 2000

Orlando, Florida

Edited by
Asim Smailagic, Robert Brodersen, and Hugo De Man

Sponsored by the
IEEE Computer Society Technical Committee on VLSI



Los Alamitos, California

Washington • Brussels • Tokyo

IEEE Computer Society Order Number PR00534
ISBN 0-7695-0534-1
ISBN 0-7695-0536-8(microfiche)
Library of Congress Number 99-069215

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1-714-821-8380
Fax: + 1-714-821-4641
E-mail: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1-732-981-0060
Fax: + 1-732-981-9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81-3-3408-3118
Fax: + 81-3-3408-3553
tokyo.ofc@computer.org

Editorial production by Anne Rawlinson

Cover art production by Joe Daigle/Studio Productions

Printed in the United States of America by The Printing House

Table of Contents

System Level Design Examples

Timing Issues in System Design

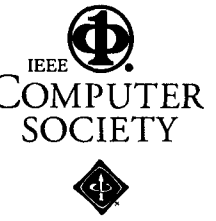
Software System Design and Design Environment

Analysis and Synthesis of Asynchronous Circuits

Advances in Multiplier Design

Issues in System Design

Author Index



Press Activities Board

IEEE Computer Society Publications

The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available from most retail outlets. Visit the Online Catalog, *http://computer.org*, for a list of products.

IEEE Computer Society Proceedings

The IEEE Computer Society also produces and actively promotes the proceedings of more than 141 acclaimed international conferences each year in multimedia formats that include hard and softcover books, CD-ROMs, videos, and on-line publications.

For information on the IEEE Computer Society proceedings, send e-mail to *cs.books@computer.org* or write to Proceedings, IEEE Computer Society, P.O. Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314. Telephone +1 714-821-8380. FAX +1 714-761-1784.

Additional information regarding the Computer Society, conferences and proceedings, CD-ROMs, videos, and books can also be accessed from our web site at *http://computer.org/cspress*

Revised 9 November 1999

Message from the General Chairs

Welcome to Orlando, to WVLSI '2000. We hope and trust that those of you attending the workshop find it to be both enjoyable and a productive use of your time. WVLSI has become a regular annual forum for researchers to exchange ideas in the area of VLSI and system level design, in particular.

This workshop has been successful over the past decade due to the many members of the VLSI community who have volunteered their efforts. In particular, we would like to thank the Program Co-chairs Asim Smailagic, Robert Broderson and Hugo De Man for having put together a program of high technical excellence.

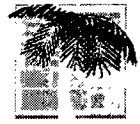
Srinivas Katkoori and Vamsi Krishna helped in coordinating the publicity for the workshop and earn our thanks for their excellent job. We appreciate the help from the past General Co-Chairs, Nagarajan Ranganathan and Anantha Chandrakhasan, in organizing the conference. Crucial administrative help came from the members of the IEEE Computer Society, in particular, Anne Marie Kelly, Mary-Kate Rada and Maggie Johnson.

Welcome once again and enjoy the workshop program.

Vijaykrishnan Narayanan
The Pennsylvania State University, USA

Mary Jane Irwin
The Pennsylvania State University, USA

Message from the Technical Program Chairs



It is our distinct pleasure to welcome you to the IEEE Computer Society Annual Workshop on VLSI in Orlando, FL.

This Workshop explores emerging trends and novel concepts in the area of VLSI. The theme of the Workshop is System Design for a System-on-Chip Era. System Level Design has been identified as a dominant research theme for the next decade. System Design has been gaining significance and momentum recently due to the emergence of system-on-a-chip designs. New visionary approaches at the system design level are needed to exploit the great opportunities created by the continuous advances in technology and miniaturization of the semiconductor devices.

System design is converging on a paradigm which includes general purpose commodity chips (i.e. processors, memories, DSP) and full custom mixed analogy and digital application specific integrated circuits (ASICs) integrated via programmable gate arrays on custom printed circuit boards or complete silicon boards, System-on-a-Chip. These hardware systems will be driven by custom, real time software that utilizes the latest software design paradigms (i.e. object oriented languages, client-server architecture, browser interfaces) and wireless communications to provide users with unique functionality. To be effective, these systems must be optimized taking into account a variety of constraints including complexity, power consumption, heat dissipation, mechanical packaging, ergonomics, and design effort. Also, future system design methodologies are an important topic at the Workshop.

We are glad to have a number of leading scientists and distinguished speakers on the workshop program, providing an unique opportunity for the attendees to hear the recent research results in this technical area. It is the face to face meetings with each other that attendees will probably value most, which is why we have tried to maintain a schedule permitting such interactions.

We would like to acknowledge the effort and help from the program committee members, and thank the authors and invited speakers for their contributions to an outstanding technical program. We gratefully acknowledge a diligent work of Anne Rawlinson, of the IEEE Computer Society Press, on the workshop proceedings.

It is our sincere hope that each attendee will benefit greatly from participating in this conference, and will find these proceedings to be a valuable source of information for your future work.

Asim Smailagic
Carnegie Mellon University

Robert Brodersen
University of California at Berkeley

Hugo De Man
IMEC, Belgium

Workshop Committees

General Chairs

Vijaykrishnan Narayanan
The Pennsylvania State University, USA

Mary Jane Irwin
The Pennsylvania State University, USA

Technical Program Chairs

Asim Smailagic
Carnegie Mellon University, USA

Robert Brodersen
University of California, Berkeley, USA

Hugo De Man
IMEC, Belgium

Program Committee

Gaetano Borriello, *University of Washington, USA*
Don Bouldin, *University of Tennessee, USA*
Francky Catthoor, *IMEC, Belgium*
Anantha Chandrakasan, *Massachusetts Institute of Technology, USA*
Mike Connors, *Intel Corporation, USA*
Rolf Ernst, *University of Braunschweig, Germany*
Georges Gielen, *Katholieke Universiteit Leuven, Belgium*
Mike Gordon, *Cambridge University, UK*
Rajesh Gupta, *University of California, Irvine, USA*
Sorin Huss, *Darmstadt University of Technology, Germany*
Fritz Prinz, *Stanford University, USA*
Teresa Meng, *Stanford University, USA*

Hidetoshi Onodern, *Kyoto University, Japan*
Jef Van Meerbergen, *Philips, Netherlands*
Pierre Paulin, *SGs-Thomson, Grenoble, France*
Jan Rabaey, *University of California, Berkeley, USA*
Zoran Salcic, *University of Auckland, New Zealand*
Dan Siewiorek, *Carnegie Mellon University, USA*
Mani Srivastava, *UCLA, USA*
Diederik Verkest, *IMEC Belgium*
Vijaykrishnan Narayanan, *The Pennsylvania State University, USA*
Jacob White, *Massachusetts Institute of Technology, USA*
Wayne Wolf, *Princeton University, USA*

Publicity Chairs

Srinivas Katkoori
University of South Florida, Tampa, USA

Vamsi Krishna
HP Labs, USA

Registration Chair

Srinivas Aruru
Intel Corporation, USA

Steering Committee

A. Mukherjee
University of Central Florida, Orlando, USA

D. W. Bouldin
University of Tennessee, Knoxville, USA

N. Ranganathan
University of South Florida, Tampa, USA

P. A. Subramanyam
AT&T Bell Labs, Murray Hill, USA

J. A. B. Fortes
Purdue University, West Lafayette, USA