Implementation of a 32-bit RISC Processor for the Data-Intensive Architecture Processing-In-Memory Chip

Jeffrey Draper, Jeff Sondeen, Sumit Mediratta, Ihn Kim
University of Southern California Information Sciences Institute
draper@isi.edu, sondeen@isi.edu, sumitm@isi.edu, ihnk@usc.edu

Abstract

The Data-Intensive Architecture (DIVA) system employs Processing-In-Memory (PIM) chips as smart-memory coprocessors to a microprocessor. This architecture exploits inherent memory bandwidth both on chip and across the system to target several classes of bandwidth-limited applications, including multimedia applications and pointer-based and sparse-matrix computations. The DIVA project is building a prototype workstation-class system using PIM chips in place of standard DRAMs to demonstrate these concepts. We have recently completed initial testing of the first version of the prototype PIM device.

A key component of this architecture is the scalar processor that coordinates all activity within a PIM node. Since such a component is present in each PIM node, we exploit parallelism to achieve significant speedups rather than relying on costly, high-performance processor design. The resulting scalar processor is then an in-order 32-bit RISC microcontroller that is extremely area-efficient. This paper details the design and implementation of this scalar processor in TSMC 0.18μm technology. In conjunction with other publications, this paper demonstrates that impressive gains can be achieved with very little "smart" logic added to memory devices.

1 Introduction

The increasing gap between processor and memory speeds is a well-known problem in computer architecture, with peak processor performance increasing at a rate of 50–60% per year while memory access times improve at merely 5–7%. Furthermore, techniques designed to hide memory latency, such as multithreading and prefetching, actually increase the memory bandwidth requirements [2]. A recent VLSI technology trend, embedded DRAM, offers a promising solution to bridging the processor-memory gap [9]. One application of this technology integrates logic with high-density memory in a processing-in-memory (PIM) chip. Because PIM internal processors can be directly connected to the memory banks, the memory bandwidth is dramatically increased (with hundreds of gigabit/second aggregate bandwidth available on a chip—up to 2 orders of magnitude over conventional DRAM systems). Latency to on-chip logic is also reduced, down to as little as one half that of a conventional memory system, because internal memory accesses avoid the delays associated with communicating off chip.

The Data-Intensive Architecture (DIVA) project leverages PIM technology to replace or augment the memory system of a conventional workstation with "smart memories" capable

of very large amounts of processing. System bandwidth limitations are thus overcome in three ways: (1) tight coupling of a single PIM processor with an on-chip memory bank; (2) distributing multiple processor-memory nodes per PIM chip; and, (3) utilizing a separate chip-to-chip interconnect that allows PIM chips to communicate without interfering with host memory bus traffic. Although suitable as a general-purpose computing platform, DIVA specifically targets two important classes of applications that are severely performance limited by the processor-memory bottlenecks in conventional systems: multimedia processing and applications with irregular data accesses. Multimedia applications tend to have little temporal reuse [12] but often exhibit spatial locality and both fine-grain and coarse-grain parallelism. DIVA PIMs exploit spatial locality and fine-grain parallelism by accessing and operating upon multiple words of data at a time and exploit coarse-grain parallelism by spreading independent computations across PIM nodes. Applications with irregular data accesses, such as sparse-matrix and pointer-based computations, perform poorly on conventional architectures because they tend to lack spatial locality and thus make poor use of caches. As a result, their execution is dominated by memory stalls [3]. DIVA accelerates such applications by eliminating much of the traffic between a host processor and memory; simple operations and dereferencing can be done mostly within PIM memories.

Performance evaluation of many applications has shown that a DIVA platform provides significant speedups. These results as well as thorough descriptions of system architecture issues have appeared in previous papers [5, 6, 7]. Also included in previous publications are comparisons to other PIM architectures as well as conventional architectures. This paper focuses on the microarchitecture design and implementation of the scalar processor, or microcontroller, that coordinates all activity on a DIVA PIM node. Due to area constraints, the design goal was a relatively simple processor with a coherent, well-designed instruction set, for which a gcc-like compiler is being adapted. The resulting scalar processor is a RISC processor that supports single-issue, in-order execution, with 32-bit instructions and 32-bit addresses. Its novelty lies in the special-purpose functions it supports to interface to other crucial components of the DIVA design. The processor was fabricated as part of a DIVA prototype chip in TSMC 0.18μm technology and is currently in test. The remainder of the paper is organized as follows. Sections 2 and 3 present an overview of the DIVA system architecture and microarchitecture, to put the scalar processor design into its proper context. Section 4 describes the scalar processor microarchitecture in detail. Section 5 presents details of the fabrication and testing of the scalar processor as part of a PIM chip, and Section 6 concludes the paper.

2 System architecture overview

A driving principle of the DIVA system architecture is efficient use of PIM technology while requiring a smooth migration path for software. This principle demands integration of PIM features into conventional systems as seamlessly as possible. As a result, DIVA chips are designed to resemble commercial DRAMs, enabling PIM memory to be accessed by host software as if it were conventional memory. In Figure 1, we show a small set of PIMs connected to a single host processor through conventional memory control logic.

Spawning computation, gathering results, synchronizing activity, or simply accessing non-local data is accomplished via parcels. A parcel is closely related to an active message as it is a relatively lightweight communication mechanism containing a reference to

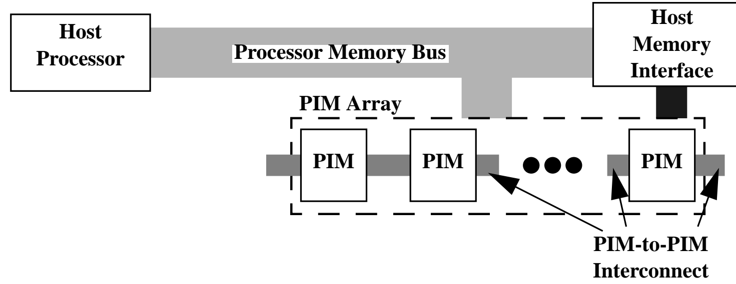

Figure 1. DIVA system architecture

a function to be invoked when the parcel is received [14]. Parcels are distinguished from active messages in that the destination of a parcel is an object in memory, not a specific processor. From a programmer's view, parcels, together with the global address space supported in DIVA, provide a compromise between the ease of programming a shared-memory system and the architectural simplicity of pure message passing. Parcels are transmitted through a separate PIM-to-PIM interconnect to enable communication without interfering with host-memory traffic, as shown in Figure 1. Details of this interconnect may be found in [11], and more details of the system architecture may be found in [5, 6, 7].

3 Microarchitecture overview

Each DIVA PIM chip is a VLSI memory device augmented with general-purpose computing and networking/communication hardware. Although a PIM may consist of multiple nodes, each of which are primarily comprised of a few megabytes of memory and a node processor, Figure 2a shows a PIM with a single node, which reflects the focus of the initial

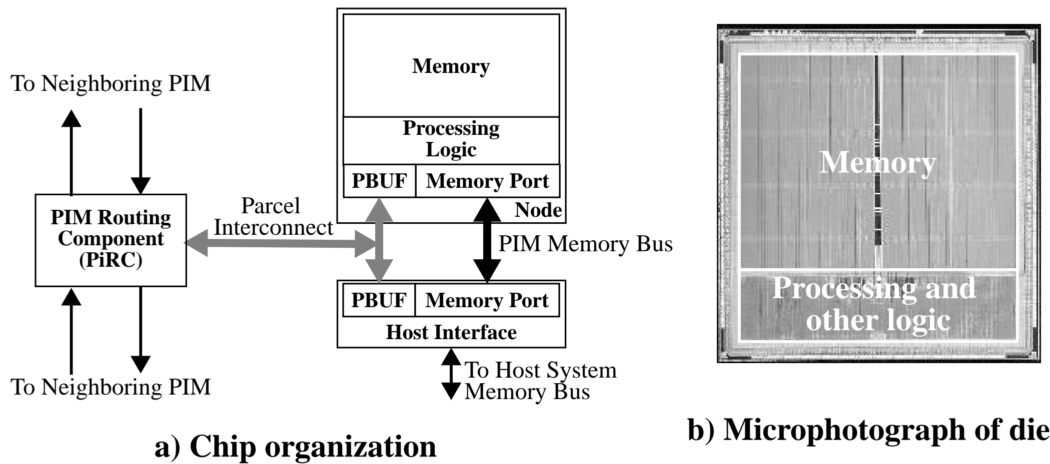

Figure 2. DIVA PIM chip

research that is being conducted. Nodes on a PIM chip share a single PIM Routing Component (PiRC) and a host interface. The PiRC is responsible for routing parcels between on-chip parcel buffers and neighboring off-chip PiRCs. The host interface implements the

JEDEC standard SDRAM protocol [10] so that memory accesses as well as parcel activity initiated by the host appear as conventional memory accesses from the host perspective.

Figure 2a also shows two interconnects that span a PIM chip for information flow between nodes, the host interface, and the PiRC. Each interconnect is distinguished by the type of information it carries. The PIM memory bus is used for conventional memory accesses from the host processor. The parcel interconnect allows parcels to transit between the host interface, the nodes, and the PiRC. Within the host interface, a parcel buffer (PBUF) is a buffer that is memory-mapped into the host processor's address space, permitting application-level communication through parcels. Each PIM node also has a PBUF, memory-mapped into the node's local address space.

Figure 3 shows the major control and data connections within a node. The DIVA PIM node processing logic supports single-issue, in-order execution, with 32-bit instructions and 32-bit addresses. There are two datapaths whose actions are coordinated by a single execution control unit: a 32-bit scalar datapath that performs operations similar to those of standard 32-bit integer units, and a 256-bit WideWord datapath that performs fine-grain parallel operations on 8-, 16-, or 32-bit operands. Both datapaths execute from a single instruction stream under the control of a single 5-stage DLX-like pipeline [8]. The

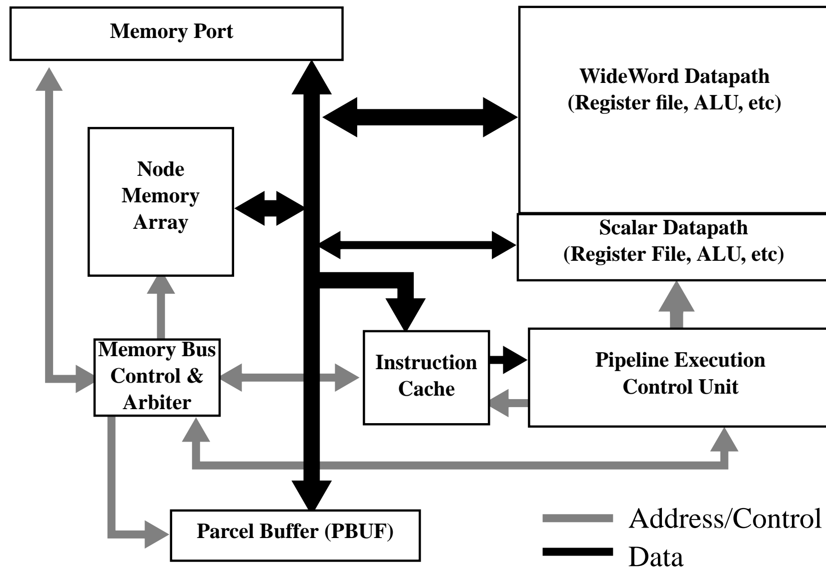

Figure 3. DIVA PIM node architecture

instruction set has been designed so both datapaths can, for the most part, use the same opcodes and condition codes, generating a large functional overlap. Each datapath has its own independent general-purpose register file, 32 32-bit registers for the scalar datapath and 32 256-bit registers for the WideWord datapath, but special instructions permit direct transfers between datapaths without going through memory. Although not supported in the initial DIVA prototype, floating-point extensions to the WideWord unit will be provided in future systems. In addition to the execution unit and associated datapaths, each DIVA PIM node contains other essential components of note. Descriptions of these components as well as the WideWord datapath will appear in future publications.

4 Microarchitecture details of the DIVA scalar processor

The combination of the execution control unit and scalar datapath is for the most part a standard RISC processor and serves as the DIVA scalar processor, or microcontroller. It coordinates all activity within a DIVA PIM node, including SIMD-like operations in the WideWord datapath, interactions between the scalar and WideWord datapaths, and parcel communication. To avoid synchronization overhead and compiler issues associated with coprocessor designs and also design complexity associated with superscalar interlocks, the DIVA scalar processor was designed to be tightly integrated with other subcomponents, as described in the previous section. This characteristic led to a custom design rather than augmenting an off-the-shelf embedded IP core. This section describes the microarchitecture of the DIVA scalar processor by first presenting an overview of the instruction set architecture, followed by a description of the pipeline and discussion of special features.

4.1 Instruction set architecture overview

Much like the DLX architecture [8], most DIVA scalar instructions use a three-operand format to specify two source registers and a destination register, as shown in Figure 4. For these types of instructions, the opcode generally denotes a class of operations, such as arithmetic, and the function denotes a specific operation, such as add. The **C** bit indicates whether the operation performed by the instruction execution updates condition codes. In lieu of a second source register, a 16-bit immediate value may be specified. The scalar instruction set includes the typical arithmetic functions add, subtract, multiply, and divide; logical functions AND, OR, NOT, and XOR; and logical/arithmetic shift operations. In addition, there are a number of special instructions, described in Section 4.3. Load/store instructions adhere to the immediate format, where the address for the memory operation is formed by the addition of an immediate value to the contents of **rA**, which serves as a base address. The DIVA scalar processor does not support a base-plus-register addressing mode because it requires an extra read port on the register file for store operations.

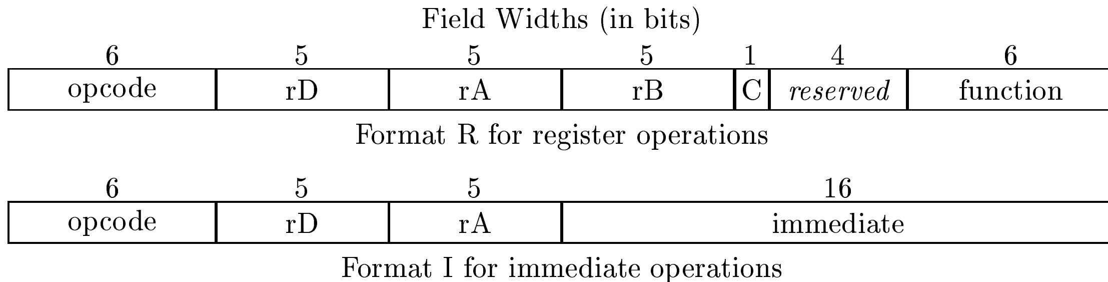

Figure 4. DIVA scalar arithmetic/logical instruction formats

Branch instructions use a different format (not shown due to page constraints). The branch target address may be PC-relative, useful for relocatable code, or calculated using a base register combined with an offset, useful with table-based branch targets. In both formats, the offset is in units of instruction words, or 4 bytes. By specifying the offset in instruction words, rather than bytes, a larger branch window results. To support function calls, the branch instruction format also includes a bit for specifying linkage, that is, whether a return instruction address should be saved in R31. The branch format also includes a

3-bit condition field to specify one of eight branch conditions: always, equal, not equal, less than, less than or equal, greater than, greater than or equal, or overflow.

4.2 Pipeline description and associated Hazards

A more detailed depiction of the pipeline execution control unit and scalar datapath are given in Figure 5. The pipeline is a standard DLX-like 5-stage pipeline [8], with the following stages: (1) instruction fetch; (2) decode and register read; (3) execute; (4) memory; and, (5) writeback. The pipeline controller contains the necessary logic to handle data, control, and structural hazards. Data hazards occur when there are read-after-write register dependences between instructions that co-exist in the pipeline. The controller and datapath contain the necessary forwarding, or bypass, logic to allow pipeline execution to proceed without stalling in most data dependence cases. The only exception to this generality involves the load instruction, where a "bubble" is inserted between the load instruction and an immediately following instruction that uses the load target register as one of its source operands. This hazard is handled with hardware interlocks, rather than exposing it, to be compatible with a previously developed compiler.

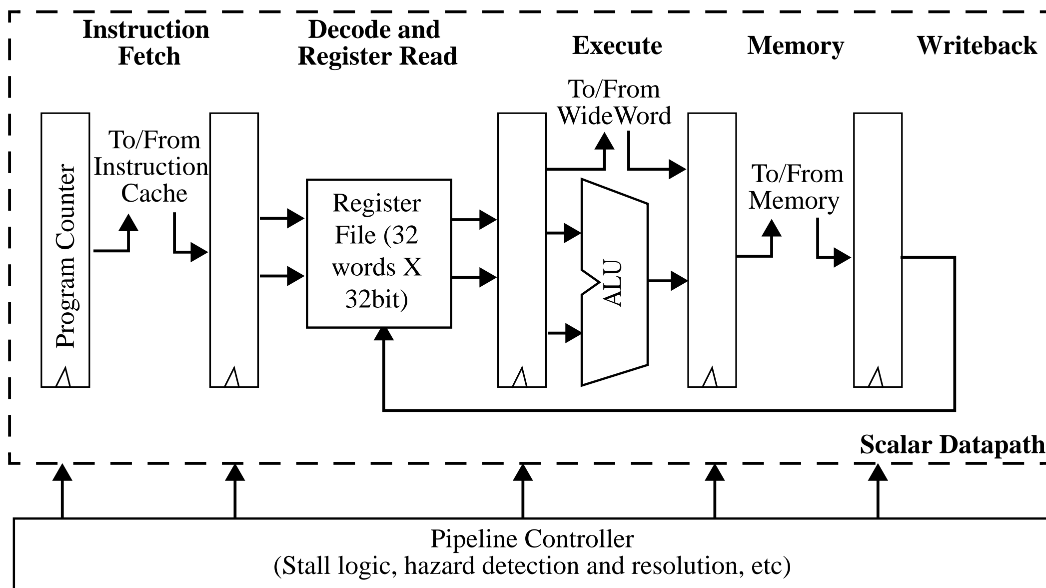

Figure 5. DIVA scalar processor pipeline description

Control hazards occur for branch instructions. Unlike the DLX architecture [8], which uses explicit comparison instructions and testing of a general-purpose register value for branching decisions, the DIVA design incorporates condition codes that may be updated by most instructions. Although a slightly more complex design, this scheme obviates the need for several comparison instructions in the instruction set and also requires one fewer instruction execution in every comparison/branch sequence. The condition codes used for branching decisions are: **EQ** - set if the result is zero, **LT** - set if the result is negative, **GT** - set if the result is positive, and **OV** - set if the operation overflows. Unlike the load data dependence hazard, which is not exposed to the compiler, the DIVA pipeline design imposes a 1-delay slot branch, so that the instruction following a branch instruction is

always executed. Since branches are always resolved within the second stage of the pipeline, no stalls occur with branch instructions. The delayed branch was selected because it was compatible with a previously developed compiler.

Since the general-purpose register file contains 2 read ports and 1 write port, it may sustain two operand reads and 1 result write every clock cycle; thus, the register file design introduces no structural hazards. The only structural hazard that impacts the pipeline operation is the node memory. Pipeline stalls occur when there is an instruction cache miss. The pipeline will resume once the cache fill memory request has been satisfied. Likewise, since there is no data cache, stalls occur any time a load/store instruction reaches the memory stage of the pipeline until the memory operation is completed.

4.3 Special features

The novelty of the DIVA scalar processor lies in the special features that support DIVA-specific functions. Although by no means exhaustive, this section highlights some of the more notable capabilities.

4.3.1 Run-time kernel support

The execution control unit supports supervisor and user modes of processing and also maintains a number of special-purpose and protected registers for support of exception handling, address translation, and general OS services. Exceptions, arising from execution of node instructions, and interrupts, from other sources such as an internal timer or external component like the PBUF, are handled by a common mechanism. The exception handling scheme for DIVA has a modest hardware requirement, exporting much of the complexity to software, to maintain a flexible implementation platform. It provides an integrated mechanism for handling hardware and software exception sources and a flexible priority assignment scheme that minimizes the amount of time that exception recognition is disabled. While the hardware design allows traditional stack-based exception handlers, it also supports a non-recursive dispatching scheme that uses DIVA hardware features to allow preemption of lower-priority exception handlers.

The impact of run-time kernel support on the scalar processor design is the addition of a modest number of special-purpose and protected (or supervisor-level) registers and a non-negligible amount of complexity added to the pipeline control for entering/exiting exception handling modes cleanly. When an exception is detected by the scalar processor control unit, the logic performs a number of tasks within a single clock cycle to prepare the processor for entering an exception handler in the next clock cyle. Those tasks include:

- determining which exception to handle by prioritizing among simultaneously occurring exceptions,
- setting up shadow registers to capture critical state information, such as the processor status word register, the instruction address of the faulting instruction, the memory address if the exception is an address fault, etc,
- configuring the program counter logic to load an exception handler address on the next clock cycle, and
- setting up the processor status word register to enter supervisor mode with exception handling temporarily disabled.

Once invoked, the exception handler first stores other pieces of user state and interrogates various pieces of state hardware to determine how to proceed. Once the exception handler routine has completed, it restores user state and then executes a return-from-exception instruction, which copies the shadow register contents back into various state registers to resume processing at the point before the exception was encountered. If it is impossible to resume previous processing due to a fatal exception, the run-time kernel exception handler may choose to terminate the offending process.

4.3.2 Interaction with the WideWord datapath

There are a number of features in the scalar processor design involving communication with the WideWord datapath that greatly enhance performance. The path to/from the WideWord datapath in the execute stage of the pipeline, shown in Figure 5, facilitates the exchange of data between the scalar and WideWord datapaths without going through memory. This capability distinguishes DIVA from other architectures containing vector units, such as AltiVec [1]. This path also allows scalar register values to be used as specifiers for WideWord functions, such as indices for selecting subfields within WideWords and indices into permutation look-up tables [4]. Instead of requiring an immediate value within a WideWord instruction for specifying such indices, this register-based indexing capability enables more intelligent, efficient code design.

There are also a couple of instructions that are especially useful for enabling efficient data mining operations. ELO, encode leftmost one, and CLO, clear leftmost one, are instructions that generate a 5-bit index corresponding to the bit position of the leftmost one in a 32-bit value and clear the leftmost one in a 32-bit value, respectively. These instructions are especially useful for examining the 32-bit WideWord condition code register values, which may be transferred to scalar general-purpose registers to perform such tests. For instance, with this capability, finding and processing data items that match a specified key are accomplished in much fewer instructions than a sequence of bit masking and shifting involved in 32 bit tests, which is required with conventional processor architectures.

There are some variations of the branch/call instructions that also interact with the WideWord datapath. The **BA** (branch on all) instruction specifies that a branch is to be taken if the status of condition codes within every subfield of the WideWord datapath matches the condition specified in the BA instruction. The **BN** (branch on none) instruction specifies that a branch is to be taken if the status of condition codes within no subfield of the WideWord datapath matches the condition specified in the BN instruction. With proper code structuring around these instructions, inverse forms of these branches, such as branch on any or branch on not all, can also be effected.

4.3.3 Miscellaneous instructions

There are also several other miscellaneous instructions that add some complexity to the processor design. The probe instruction allows a user to interrogate the address translation logic to see if a global address is locally mapped. This capability allows users who wish to optimize code for performance to avoid slow, overhead-laden address translation exceptions. Also, an instruction cache invalidate instruction allows the supervisor kernel to evict user code from the cache without invalidating the entire cache and is useful in process termination cleanup procedures. Lastly, there are versions of load/store instructions that "lock" memory operations, which are useful for implementing synchronization functions, such as semaphores or barriers.

5 Implementation and testing of the DIVA scalar processor

The specification of the DIVA scalar processor required on the order of 10,000 lines of VHDL code, consisting of a mix of RTL-level behavioral and gate-level structural code. A preliminary, unoptimized stand-alone layout of the scalar processor consisted of 23,000 standard cells (approximately 200,000 transistors) and occupied 1 sq mm in 0.18μm technology. It was projected to operate at 400MHz while dissipating 80mW.

Although the scalar processor is suitable for stand-alone embedded implementations, the DIVA project employs it as part of a tightly integrated node design, as discussed in Section 3. The scalar processor VHDL specification was included as part of the DIVA PIM prototype specification, which was synthesized as a "sea of gates" using Synopsys Design Analyzer. The entire chip was placed and routed with Cadence Silicon Ensemble, and physical verification, such as DRC and LVS, was performed with Mentor Calibre. The intellectual property building blocks used in the chip include Virage Logic SRAM, a NurLogic PLL clock multiplier, and Artisan standard cells, pads, and register files.

The first DIVA PIM prototype, shown in Figure 2b, is a single-node implementation of the DIVA PIM chip architecture and is currently in test. Due to challenges in gaining access to embedded DRAM fabrication lines in a timely fashion, this first prototype is SRAM-based. This chip implements all features of the DIVA PIM architecture except address translation and floating-point capabilities. A second version of a PIM chip, which not only integrates these functions but achieves a faster clock rate, is due to tape out in the second half of 2002. The chip shown in Figure 2b was fabricated through MOSIS in TSMC 0.18μm technology, and the silicon die measures 9.8mm on a side. It contains approximately 2 million logic transistors in addition to the 53 million transistors that implement 8 Mbits of SRAM. The chip also contains 352 pads, 240 signal I/O, and is packaged in a 35mm TBGA. The chip is estimated to dissipate 2.5W at 100MHz.

The chip is being tested with the use of an HP 16702A logic analysis mainframe. Pattern generator modules apply test vectors to the inputs of the chip, and timing/state capture modules sense the outputs of the chip. The chip is currently being tested for functionality at a testbench speed of 80MHz. Although exhaustive testing has not yet been completed, the chip is running a demonstration application of matrix transpose that exercises all major control and datapaths within the scalar processor, including many of the special features highlighted in Section 4.3. Even in this limited test setup, the chip is performing 640 MOPS while dissipating only 800mW. We estimate that the scalar processor is contributing only 80mW to this power measure. Also, though at-speed testing has not been completed yet, we do not anticipate this prototype to operate much beyond 100MHz due to critical path limitations in the WideWord datapath. If implemented and optimized separately as an embedded microcontroller, we expect the scalar processor to easily operate above 500MHz.

6 Conclusion

This paper has presented the design and implementation of the scalar PIM processor used in the DIVA system, an integrated hardware and software architecture for exploiting the bandwidth of PIM-based systems. Although the core of the scalar processor design is much like a standard 32-bit RISC processor, it has a number of special features that make it well-suited to serving as a PIM node microcontroller. A working implementation of this

architecture, based on TSMC 0.18μm technology, has proven the validity of the design. The resulting workstation system architecture that incorporates PIMS using this processor is projected to achieve speedups ranging from 8.8 to 38.3 over conventional workstations for a number of applications [5, 6]. These results demonstrate that by sacrificing a small amount of area for processing logic on memory chips, PIM-based systems are a viable method for combatting the memory wall problem.

Acknowledgments

The authors would like to acknowledge the support of the DIVA project team and DARPA (Contract No. F30602-98-2-0180).

References

[1] http://www.altivec.org.

[2] D. Burger, J. Goodman and A. Kagi, "Memory Bandwidth Limitations of Future Microprocessors," *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1996.

[3] J.B. Carter, et al, "Impulse: Building a Smarter Memory Controller", *Proceedings of the Fifth International Symposium on High Performance Computer Architecture*, pp. 70-79, January 1999.

[4] J. Chame, M. Hall and J. Shin, "Code Transformations for Exploiting Bandwidth in PIM-Based Systems," *Proceedings of the ISCA Workshop on Solving the Memory Wall Problem*, June 2000.

[5] J. Draper, et al, "The Architecture of the DIVA Processing-in-Memory Chip", to appear at the International Conference on Supercomputing, June 2002.

[6] Mary Hall, et al, "Mapping Irregular Application to DIVA, a PIM-based Data-Intensive Architecture," *Supercomputing*, November 1999.

[7] M. Hall and C. Steele, "Memory Management in a PIM-Based Architecture," *Proceedings of the Workshop on Intelligent Memory Systems*, October 2000.

[8] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, Second Edition, 1996.

[9] S. Iyer and H. Kalter, "Embedded DRAM technology," *IEEE Spectrum,* April 1999, pp. 56-64.

[10] http://www.jedec.org.

[11] C. Kang and J. Draper, "A Fast, Simple Router for the Data-Intensive Architecture (DIVA) System," *Proceedings of the IEEE Midwest Symposium on Circuits and Systems*, August 2000.

[12] P. Ranganathan, S. Adve and N. Jouppi, "Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions," *Proceedings of the International Symposium on Computer Architecture*, May 1999.

[13] A. Saulsbury, T. Wilkinson, J. Carter and A. Landin, "An Argument for Simple COMA", *Proceedings of the Symposium on High-Performance Computer Architecture*, January 1995.

[14] T. von Eicken, D. Culler, S. C. Goldstein and K. Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation", *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.