Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

DEVELOPING A SIMULATOR FOR THE USC ORTHOGONAL MULTIPROCESSOR

Sharad Mehrotra

Department of EE-Systems
University of Southern California
Los Angeles, California 90089-1115

ABSTRACT

This paper describes the development of a CSIM-based simulator during the RISC-based design of a prototype multiprocessor, the *Orthogonal Multiprocessor* (OMP). The prototype machine is to be built with Intel *i*860 microprocessors and parallel memory modules that are 2-D interleaved and orthogonally accessed using custom-designed spanning buses. Initial application areas for the machine are image processing, computer vision and neural network simulation. After briefly describing the machine architecture and the simulator, we discuss the interplay between simulation and design, and show specific cases where simulation results have impacted the design. The simulator has been used to project machine performance and evaluate design choices.

1. INTRODUCTION

The advent of RISC processors and the increasing use of parallel processing software [Hwang and DeGroot 1989] have created a significant impact on the computer industry. This paper reports the interplay between design and simulation during the development of an *Orthogonal Multiprocessor* (OMP) system using state-of-the-art 64-bit RISC microprocessors, a conflict-free memory organization, and multi-dimensional spanning buses. This OMP architecture was developed at the University of Southern California [Hwang et al. 1989] during the past four years. The detailed design of a 16-processor OMP prototype is currently underway using Intel i860 CPU chips and funding from the US National Science Foundation.

We have developed a C-language based OMP simulator [Mehrotra 1990, Cheng 1990], using the CSIM simulation language (CSIM has been copyrighted by the Microelectronics and Computer Technology Corporation) developed at MCC [Schwetman 1986]. The simulator has been used to investigate architectural design choices. It is also being used to develop and evaluate the performance of parallel algorithms for the OMP. Simulated performance results for matrix multiplication, orthogonal sorting, and 2-D FFT on an OMP with 16 RISC processors were reported in [Mehrotra et al. 1990].

The rest of this paper is organized as follows. In Section 2 we introduce the architecture of the Orthogonal Multiprocessor. In Section 3 we provide a brief description of the design of the OMP simulator. In the next two sections, we describe how simulation results have validated the design, and how we are constantly refining the simulator to produce more accurate performance data. Finally, in Section 6 we summarize the current status of the simulator and outline future tasks.

2. THE OMP ARCHITECTURE

The global OMP architecture is briefly discussed in this section. Further details can be found in [Hwang et al. 1990]. As illustrated in Fig. 1, the prototype OMP will consist of 16 i860 processors and 256 partially shared *memory modules*, that are interconnected by a mesh of spanning buses. Each processor uses a dedicated pair of spanning buses to access a row of 16 interleaved memory modules, called a *row access*, or to access a column of 16 interleaved memory modules, called a *column access*. The two access modes are mutually exclusive so that at any given time the orthogonal memory is accessed by rows or by columns, but not in a mixed access mode. Each memory module, $M_{ij}(i \neq j)$, is shared by two processors, P_i and P_j, but only one of the two processors can access the memory module at a time. The collection of all 256 memory modules is called *orthogonal access memory* (OAM).

The OMP architecture efficiently supports the *Single Program, Multiple Data* (SPMD) programming style [Karp 1987] using partially shared memory. SPMD operations imply that a large program is partitioned into multiple subprograms for parallel execution by the i860 processors. The divided subprograms may require synchronization at certain points of time, but they do not have to be executed in lock step at the instruction level as in an SIMD machine. Instead, SPMD can be considered as a special class of MIMD operation.

Figure 2 illustrates the memory hierarchy and access latencies viewed from each processor, P_i for i = 0, 1, 2,..., 15. Each processor sees four different levels of memory; namely the *internal cache* (8K bytes of data and 4K bytes of instruction), *local memory*, the *Vector Register Windows* (VRWs), and a subspace of the OAM, involving 31 modules on 7 memory boards. It should be noted that the diagonal memory module, M_{ii}, is only accessed by processor P_i as a private memory and participates in both row and column accesses. Eight spanning buses, HB_i and VB_i, for i=0, 1, 2, 3 are built on the memory backplane forming the *spanning bus network* shown in the center of Fig. 1 and Fig. 2. The access latencies of the four memory levels are labelled as t_1, t_2, t_3, t_4 in Fig. 2. The local memory is used to hold programs, scalar data sets, and local variables. The hierarchy, $t_2 < t_4$, forms a 2-level path to access the orthogonal memory, which holds vector data sets either by rows or by columns. Thus, the OAM supported by the VRWs acts as a form of *vector memory*. Further details of the memory hierarchy and its management may be found in [Panda and Hwang 1990].

3. THE OMP SIMULATOR

The OMP simulator is *process-oriented* and *algorithm-driven*. In a process-oriented simulator, the programmer defines *processes*

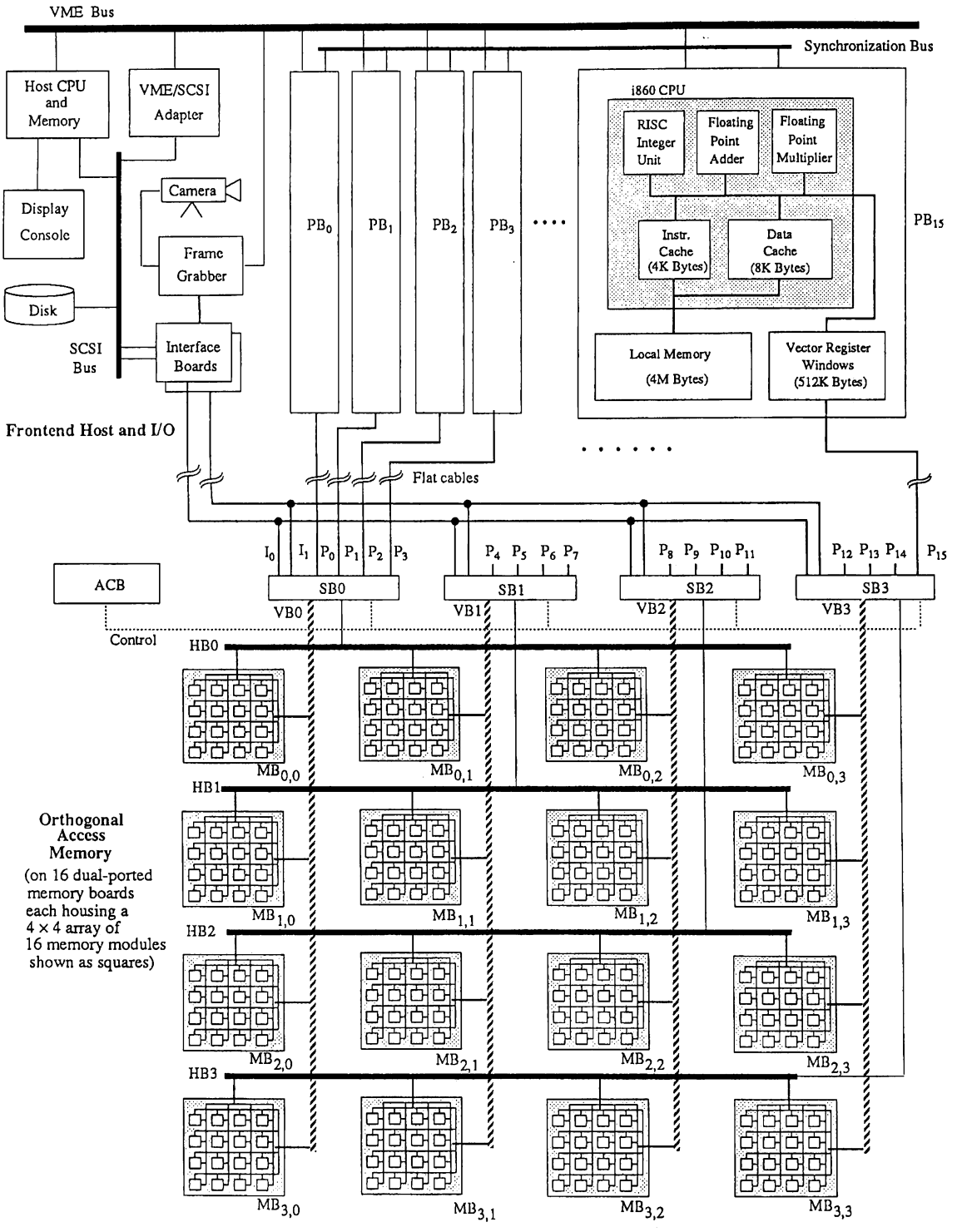

Figure 1. Board Level Design of the Orthogonal Multiprocessor with Sixteen i860 Processors and 256 Memory Modules Interconnected by 8 Spanning Buses (PB_i: Processor Boards, $MB_{i,j}$ Memory Boards, SBi: Switch Boards, ACB: Access Control Board, HBi: Horizontal Bus, VBi: Vertical Bus)

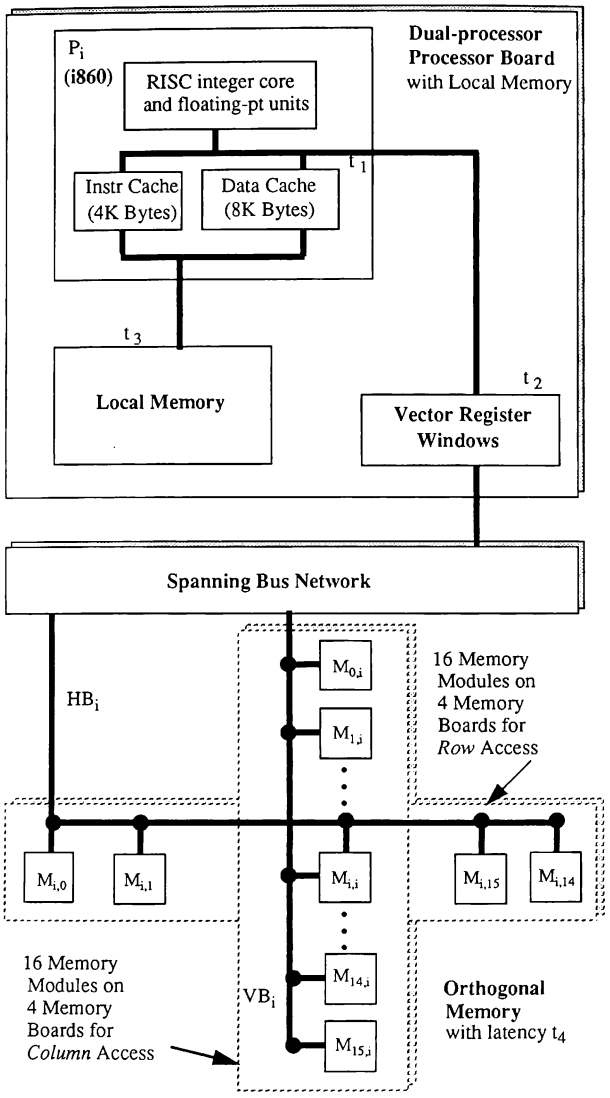

Figure 2. Memory Hierarchy and Access Latencies from each Processor P_i for i=0,1,..,15 in the Prototype Design

that use the modeled system's resources. A process represents a complete computing activity that is scheduled for execution on the simulation engine. During the simulation run, interacting processes communicate and synchronize with each other as they use system resources [MacDougall 1976]. In an algorithm-driven simulator, programs are used to drive the simulation, and performance measures are gathered during the program run. Users of the OMP simulator write parallel algorithms in terms of processes by using the C programming language augmented with *simulation directives* from the CSIM simulation kernel [Schwetman 1989]. The design of our simulator has been influenced by the ASPOL-based simulator for the ETA[10] supercomputer [ETA Systems 1986] and the SPAWN simulator [Dubois et al. 1986]. One of our goals has been to retain the advantages of algorithm-driven simulation, while modeling the machine design as accurately as possible. The simulator models the backend machine shown in Fig. 1. We do not simulate the frontend host. The only dynamic interaction between the backend machine and the host occurs when program and data are transferred between the two machines. Since this interaction is infrequent we don't capture it in the simulator.

Figure 3 depicts the design of the OMP simulator. The simulator consists of three modules: the *user developed algorithm*, an OMP *architecture specification file*, and the CSIM simulation kernel. The architecture specification file defines components of OMP in terms of CSIM facilities, events and C data structures. Facility definitions include the VME Bus, the processors (i860s), the orthogonal memory modules, and the spanning bus network. Event definitions include row-access and column-access flags for each processor, and some other definitions for housekeeping. Certain operating system features that will be available on the machine are also modeled in the simulator. These include data structures and CSIM event declarations for achieving synchronization between processes (parallel *program segments*).

The OMP simulator is *deterministic*. All system delays included in the simulator are explicit values used in the actual hardware design and have been calculated from the 40MHz *i*860 data sheet and vendor parts catalogs. Table 1 shows some of the parameters used in the simulator. It is impossible to model all the hardware accurately in a process-oriented multiprocessor simulator. Judicious approximations have to made in developing the model so that the results are meaningful. We have identified important OMP parameters and calculated their values.

Table 1. Hardware Timings for 40MHz i860-based Orthogonal Multiprocessor

Time	Description
150 nsec	Local memory access time
825 nsec	OAM vector access latency
300 nsec	Time to access successive vector elements from OAM, after the initial element has been fetched
275 nsec	Synchronization bus access time
665 nsec	Time to read and modify memory location across Synchronization bus
240 nsec	Time to broadcast data on Synchronization bus
30 nsec	*i*860 RISC operation time
75 nsec	Time to access an element in VRW

The simulation directives in the OMP simulator are grouped into two sets: one set is supported by the OMP-C compiler as a set of *language extensions*, while the other is used only for simulation. Table 2 shows some of the directives. Language extensions are needed to simplify the management of OMP hardware, to efficiently allocate data structures in the orthogonal memory, to synchronize parallel program segments, and to facilitate communication between the host and the backend machine. All directives are implemented as parameterized *macros* of CSIM statements. By encapsulating CSIM statements in macros we have provided the user with an abstraction that eases simulator program development.

The OMP simulator time slices the execution of parallel code segments on a sequential workstation. Since the prototype OMP is being constructed with *i*860 chips, we need to estimate the execution time of each code segment on an *i*860 processor, and appropriately advance simulation time for the CSIM process representing that segment. We use an instruction counting technique [Weinberger 1984] to estimate the *i*860 cycles being simulated.

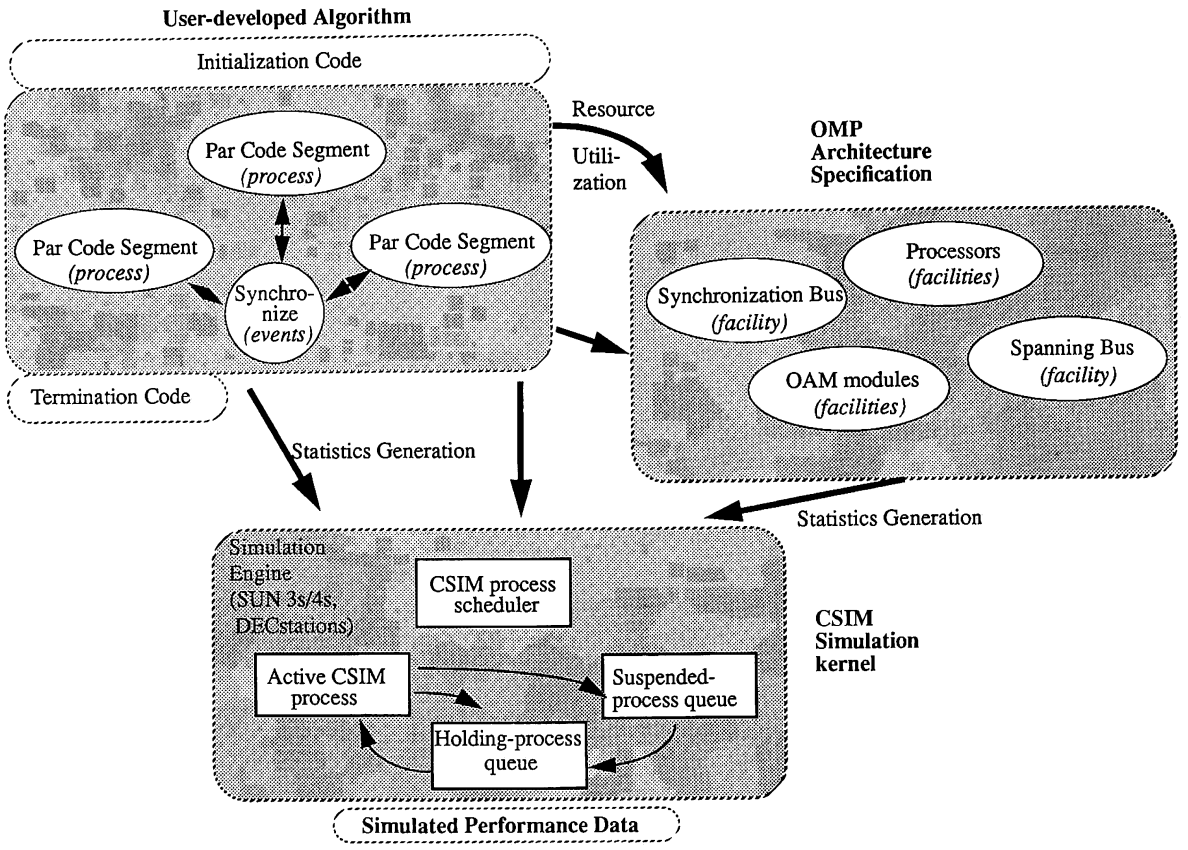

Figure 3. Organization of the Algorithm-Driven Orthogonal Multiprocessor Simulator

First, a user developed program is compiled to SPARC assembly language using the C compiler on a SUN-4 machine (SPARC and SUN-4 are trademarks of Sun Microsystems, Inc.). This assembly code is analyzed and the number of instructions in each *basic-block* of code is counted. We assume that there is a one-to-one correspondence between the SPARC assembly code and *i*860 assembly code. The cumulative instruction count for each basic-block is then inserted back to the source program as a call to the CSIM *hold* subroutine [Schwetman 1989] to expend the correct amount of simulated time. The modified source program is then recompiled with the CSIM library for building an executable image. An error would be introduced in the instruction count if the extra code generated by the simulation directives was included. This is because simulation directives representing language extensions will actually be translated into a much *smaller* number of instructions by the OMP-C compiler than the count derived from a translation of the CSIM macro. Simulation specific directives will not be translated at all! To avoid this inaccuracy, all CSIM directives are enclosed with *markers* so that instruction counting is not performed for the assembler stream in between the "clock_off" and the "clock_on" markers. Directives representing language extensions have been analyzed ahead of time and instruction counts assigned to them. Whenever any of these directives is invoked, the instruction count for that processor is incremented by a fixed amount. Thus, the final instruction count for a processor at the end of the simulation run is proportional to its *actual* computational activity.

To keep simulation times acceptable, we have made several simplifying assumptions in our simulator model for the processors used in the OMP. First, we assume that each processor can issue an instruction every clock cycle. This assumption has been made based on the 4K byte size of the instruction cache, its read-through organization, and an assumed hit rate of over 95%. Second, when floating-point operations are used, it is assumed that they can be overlapped with integer operations inside the *i*860. Third, it is assumed that the *i*860 data cache always holds all the local data. It has been our experience up to now that typical OMP programs manipulate large vector instead of scalar data sets. Vector data is stored in the orthogonal memory, accesses to which bypass the *i*860's internal data cache. Each processor will allow execution of multiple threads of code that will be managed by the operating system. This overhead has not yet been factored into the simulator.

4. DESIGN VALIDATION

The simulator has been used to verify our hardware design decisions. As mentioned earlier, system hardware has been modeled as accurately as possible. For example, the interface between the processor board and the OAM is the spanning bus network. This backplane includes logic for switching and buffering address and data information. Consequently, it contributes to the latency t_4 depicted in Fig. 2. Also included in t_4 are cable delays in connecting processor boards and the memory backplane. Other system delays have been calculated in the same way and recorded in

Table 2. Sample Simulation Directives used in the Orthogonal Multiprocessor Simulator

Directive		Remarks
Language Extension	TSETCNT(str-idx,N,pid)	Initialize synch count for process pid to N if no other process has done so (using counter indexed by str-idx)
	SYNCH(str-idx,pid)	Wait at synch point (using counter indexed by str-idx) until all processes are there
	SET_ROWACC(pid)	Process pid declares it is in row-access mode
	CLR_COL(pid)	Process pid declares it is leaving col-access mode
	PIPE_IN(pid)	Process pid to fetch interleaved row or column of data from OAM
	PIPE_OUT(pid)	Process pid to write interleaved row or column of data to OAM
	BROADCAST(pid)	Process pid to write scalar data item in pipelined mode to row or column of OAM
	CRT_TSK(tsk-nam)	Create code thread tsk-nam on processor
	END_TSK(pid)	Terminate code thread pid
Simulation Specific	_ACC_BUF(pid)	Indicate vector register window access by process pid
	_ACC_LM(pid)	Indicate local memory access by process pid
	_SIM_INIT(mname)	Initiate simulation with model name mname
	_PROC_STATS(pid)	Report statistics for process pid
	_SIM_END	End simulation

the OMP architecture specification file. Examples include local-memory access time, vector-register-window access time, and the VME-bus access time. Care has been taken to account for cases where multiple accesses are made to the orthogonal memory, such as in pipelined read and pipelined write accesses.

As of this writing, there have been at least two major design decisions that have been changed due to simulation results. These are the structure of the interprocessor bus, and the data buffering strategy for accessing the OAM. The initial design of the machine used a *single* bus for interprocessor communication and synchronization. Code threads running on the multiple processors were synchronized by a *barrier synchronization*. Simulation runs have revealed however, that this method is extremely costly if synchronization is done frequently. In the worst case, a processor arriving at the barrier has to wait for all other processors before it can execute code beyond the barrier. The overhead for synchronization can therefore become unacceptable. With the insight derived from simulation, a design decision has been made to implement a *hardwired* synchronization mechanism to augment the barrier technique [Hwang et al. 1990].

Initially it was decided to expand the *i*860 processor's internal instruction and data cache with *external* cache. The idea was that this external cache would buffer accesses to data from local memory and OAM. There was no need for maintaining cache-coherence amongst the multiple processors because each processor was operating on a physically *exclusive* region of OAM in either of its access modes. However, cache *flushing* was still required when processors switched from row-access to column-access mode or vice versa. Simulation proved cache flushing to be a major overhead. The new design uses vector register windows for buffering OAM data [Panda and Hwang 1990], and *disables* the internal caches during vector accesses. The *i*860's internal caches are activated when accessing local memory.

5. SIMULATOR REFINEMENT

The first version of the simulator declared the OAM as an array of CSIM *facilities* [Schwetman 1989], representing the OAM as a set of rows and columns without distinguishing amongst memory modules in a row or column. During execution therefore, statistics were only collected for accessing the *entire* row or column. After analyzing some preliminary data, it was decided to refine the description of the OAM down to the individual memory module. Accordingly, the OAM was redefined as an array of $N \times N$ CSIM facilities, where N is the number of processors in the system. Once this change was implemented it became possible to evaluate access patterns to single modules. With this change, it was seen that non-uniform OAM module accesses were proving to be a big penalty, so a decision was made to limit OAM accesses to only vector reads and writes. The original architectural specification had intended OAM access to be restricted as such, but the issue was reopened for discussion during the detailed design. With simulation results at hand, the designers were able to concentrate on optimizing the vector access mode.

After the external cache idea for the processor board was dropped, it was decided to offer a data buffer board to hide the access latency of OAM data. Initially we did not have a firm idea about how to structure such a buffer, except that it would have to be optimized for fast data accesses from the processor. The initial simulation model of the buffer therefore was simply as a fast on-board buffer. As the design has progressed, the on-board data buffer idea has been refined to the vector register windows concept. Although vector register windows are still composed of the *same* data buffers that were modeled earlier, they are enhanced by on-the-fly data permutation capabilities, and supported by an index memory for storing predefined data manipulation templates [Panda and Hwang 1990]. We are planning to modify our simplistic data buffer model to reflect the new design.

In modeling the spanning bus network for the initial release of the simulator, we took a *worst case* modeling approach. It was assumed that any OAM request passing thru the spanning bus network would experience its worst case delay. This delay calculation included the electrical delays in traversing the backplane and associated cables, as well as contention for the buses, HB_i and VB_i, i=0, 1, 2, 3. Bus contention occurs because each of the horizontal and vertical buses is shared by up to four processors. With the design firming up, a more detailed description of the backplane has emerged, so we are updating the simulation model.

Our new model will declare the spanning bus network as a collection of eight CSIM facilities. Since each bus will actually be time-sliced amongst the competing processors by a round-robin scheduling algorithm, the CSIM facility representing each bus will also be managed using round-robin scheduling. With this refinement we will be able to measure the bus contention as it *really* occurs for application algorithms.

The current timing model in the simulator assumes that one *i*860 instruction is equivalent to one SPARC instruction. This is only a first step to get us going. Our laboratory now has an Intel *i*860 development system that contains an *i*860 processor and a software simulator for the processor. We plan to benchmark program execution times on the development system and compare them against the running times derived from the simulator for the single CPU case. Any resulting discrepancy will be reduced by *scaling* the simulator results to correspond to the development system data. The scaled times will then be used to estimate execution times for multiprocessor runs.

6. CONCLUSIONS

We have reported our experiences in developing a multiprocessor simulator during the design of OMP. Using the simulator, we have been able to validate some critical design parameters, such as the local memory and OAM access latencies, the interprocessor synchronization overhead allowed, and the VRWs configuration. We have also been able to make some projections about the *effective processing rate* of the OMP [Hwang et al. 1990].

Several extensions to the OMP simulator are planned. The simulator produces a lot of information at run-time. Using an MCC developed graphical tool we have been able to display some of the collected statistics, but the tool needs to be enhanced. The calibration of timing estimates produced by the simulator has to be performed. In addition, we are looking at ways of speeding up the simulation runs.

ACKNOWLEDGMENTS

We would like to thank the scientific input provided by individuals working in the project. We appreciate the interaction with Viji Balan, Rama Chellappa, Chien-Ming Cheng, Michel Dubois, Brian Finnerty, Navid Haddadi, Ellis Horowitz, Fon-Jein Hsieh, Kai Hwang, Sugih Jamin, Kim Korner, Jin-Cheng Liu, Mark Lytwyn, Weihua Mao, Hemraj Nair, D.K. Panda, Santosh Rao, and Shisheng Shang during group meetings and technical discussions. We appreciate the help from Herb Schwetman of MCC, Mac MacDougall of Apple Computer, and Cliff Arnold of Control Data Corporation. We acknowledge the financial support from NSF under grant No. MIP-89-04172 and from USC's School of Engineering in terms of cost sharing and laboratory renovation.

REFERENCES

Cheng, C.M. (1990), "Programmer's Guide to the USC Orthogonal Multiprocessor Simulator," Technical Report, Department of EE-Systems, University of Southern California, Los Angeles, CA.

Dubois, M., M. Balakrishnan, F.A. Briggs and I. Patil (1986), "Trace-driven Simulations of Parallel and Distributed Algorithms in Multiprocessors," In *Proceedings of the International Conference on Parallel Processing*, K. Hwang, S.M. Jacobs, and E.E. Swartzlander, Eds. IEEE Computer Society Press, Los Alamitos, CA, 909-917.

ETA Systems Inc. (1986), *ETA-10 Multiprocessing Simulator User's Guide*, Publication 1076, ETA Systems Inc., MN.

Hwang, K., and D. DeGroot, Eds. (1989), *Parallel Processing for Supercomputers and Artificial Intelligence*, McGraw Hill, New York, NY.

Hwang, K., M. Dubois, D.K. Panda, S. Rao, S. Shang, A. Uresin, W. Mao, H. Nair, M. Lytwyn, F. Hsieh, J. Liu, S. Mehrotra, and C.M. Cheng (1990), "OMP: A RISC-based Multiprocessor using Orthogonal-Access Memories and Multiple Spanning Buses," In *Proceedings of the 1990 ACM International Conference on Supercomputing*, ACM Press, Baltimore, MD, 7-22.

Hwang, K., P.S. Tseng, and D. Kim (1989), "An Orthogonal Multiprocessor for Parallel Scientific Computations," *IEEE Transactions on Computers C-38, 1, 47-61.*

Karp, A.H. (1987), "Programming for Parallelism," *Computer 20, 5, 43-57.*

MacDougall, M.H. (1976), "System Level Simulation," In *Digital System Design Automation: Languages, Simulation and Data Base*, M.A. Breuer, Ed. Computer Science Press, Rockville, MD, 1-115.

Mehrotra, S. (1990), "Simulator Manual for the USC Orthogonal Multiprocessor," Technical Report, Department of EE-Systems, University of Southern California, Los Angeles, CA.

Mehrotra, S., C.M. Cheng, M. Dubois, K. Hwang, and D.K. Panda (1990), "Algorithm Driven Simulation and Projected Performance of the USC Orthogonal Multiprocessor," In *Proceedings of the International Conference on Parallel Processing*, IEEE Press, Los Alamitos, CA.

Panda, D.K. and K. Hwang (1990), "Reconfigurable Vector Register Windows for Fast Matrix Computation on the Orthogonal Multiprocessor," In *Proceedings of the International Conference on Application Specific Array Processors*, IEEE Computer Society Press, Los Alamitos, CA.

Schwetman, H.D. (1986), "CSIM: A C-based, Process-Oriented Simulation Language," In *Proceedings of the 1986 Winter Simulation Conference*, J. Wilson, J. Henriksen, and S. Roberts, Eds. SCSI, San Diego, CA, 387-396.

Schwetman, H.D. (1989), "CSIM Reference Manual (Revision 13)," Technical Report ACA-ST-252-87, Microelectronics and Computer Technology Corp., Austin, TX.

Weinberger, P.J. (1984), "Cheap Dynamic Instruction Counting," *AT&T Bell Laboratories Technical Journal 63*, 8, 1815-1826.