The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures

Anthony-Trung Nguyen[†], Maged Michael[‡], Arun Sharma[†], and Josep Torrellas[†]

[†]Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
Urbana, IL 61081
{anguyen,arsharma,torrella}@csrd.uiuc.edu

[‡]Department of Computer Science
University of Rochester
Rochester, NY 14627
michael@cs.rochester.edu

Abstract

Most publicly-available simulation tools only simulate RISC architectures. These tools cannot capture the instruction mix and memory reference patterns of CISC architectures. In this paper, we present an overview of Augmint, *an execution-driven multiprocessor simulation toolkit that fills this gap by supporting Intel x86 architectures. Augmint also supports trace-driven simulation for uniprocessors as well as multiprocessors, with minor effort on the part of simulator developers. Augmint runs m4-macro-extended C and C++ applications such as those in the SPLASH and SPLASH-2 benchmark suites. Augmint supports a thread-based programming model with shared global address space and private stack space. Augmint supports a simulator interface compatible with that of the MINT simulation toolkit for MIPS architectures, thus allowing the reuse of most architecture simulators written for MINT. Augmint simulations run on x86-based uniprocessor systems under UNIX or Windows NT[2]. The source code of Augmint is publicly available from http://www.csrd.uiuc.edu/iacoma/augmint.*

1 Introduction

Software simulation is the most cost-effective and flexible method to evaluate multiprocessor and memory hierarchy architectures and organizations. Alternative techniques are hardware prototyping and analytical modeling. In comparison to software simulation, hardware prototyping is expensive, inflexible, and requires a long development time. Analytical modeling often fails to capture the complexity of the interaction among complex system components.

There are roughly two main approaches to multiprocessor simulation: *trace-driven* and *execution-driven* simulation. Trace-driven simulation uses traces of instructions and/or memory references of real applications to drive memory simulators. While traces can capture realistic application and system workloads, trace-driven simulation are often unable to model the changes in the behavior of parallel applications caused by differences in timing between the target (simulated) system and the system on which the traces were collected [2, 3]. Furthermore, a trace must contain a large number of memory references to minimize the effect of cold start and variations in workloads on the simulation result, thus requiring a very large storage space. Execution-driven simulations, instead of using traces, interleave the execution of the applications with the simulation of the target systems, thus allowing the target systems to influence the behavior of the applications, yielding more realistic results than trace-driven simulations.

Most publicly-released execution-driven simulation tools support RISC architectures only [1, 4, 10, 12]. RISC-based execution-driven simulations do not capture the instruction mix and memory reference patterns of CISC architectures. Augmint was designed to fill this gap by providing a public simulation toolkit for Intel x86 architectures. Augmint currently runs on x86-based uniprocessor systems under the Solaris [1], Linux, and Windows NT [2] operating systems [7, 8].

In this paper, we present an overview of Augmint. We organize the presentation as follows. In section 2, we discuss the design issues and choices made while developing Augmint, and the similarities and differences between it and other simulation tools. Section 3 presents an overview of the structure of an Augmint simulation and the functionality of its main components. Section 4 presents the requirements and features of user applications that can be simulated using Augmint. In section 5, we describe the main features of Augmint's interface with architecture simulators or trace generators. Section 6 shows that Augmint can also be used to develop trace-driven simulators. We conclude with section 7.

[1] Solaris is a trademark of Sun Microsystems, Inc.
[2] Windows NT is a trademark of Microsoft Corporation.

2 Design Issues

Augmint was developed to meet a number of goals that influenced its design. First, it must support widely available, non-trivial parallel applications such as those in the SPLASH [9] and SPLASH-2 [13] suites. This goal would allow researchers to run applications with Augmint immediately and focus most of the effort on developing architecture simulators. It would also provide additional means for validating the correctness and accuracy of Augmint by comparing its simulation results with those of other simulators. Second, the design and implementation had to be completed quickly to allow more time for developing non-trivial architecture simulators using Augmint.

To accomplish the second goal, some parts of Augmint had to be built from existing simulation tools. The main candidates were Proteus [1], Tango Lite [4], and MINT [11, 12], since they were publicly available. MINT and Tango Lite satisfied the first goal. However, we did not simply port either Tango Lite or MINT to the x86 architecture for several reasons. For Tango Lite, we felt that the interface provided to simulator designers was not intuitive, and the scheduling facility is not general enough to allow the modeling of multiple outstanding memory references from a processor. For example, Dixielite, a simulator of the Stanford DASH project [5], requires a task management module in assembly code to manage tasks in the memory hierarchy. We believed that a simulation tool must provide a clean interface and support all the scheduling requirements of both application threads and simulator activities.

MINT satisfies both requirements. However, applying MINT's approach of interpreting the application code [3] would require a substantial investment in time to understand x86 object file formats and build an interpreter for the x86 instruction set. In addition, it would limit the portability of Augmint to operating systems with identical object file formats. Accordingly, we took the approach of native execution and modified the internals of MINT, while retaining its simulator interface and adaptable internal utilities. To support native execution of simulated applications, we developed a tool that augments application assembly code with instrumentation code that switches context to Augmint and generates interesting events (e.g. at memory reference points) for simulation. The augmentation tool recognizes the complex x86 instruction set and understands instructions with explicit as well as implicit memory references.

In order to support native execution of application threads, Augmint provides a threads package that supports one thread on the host machine for each simulated application thread in addition to a thread for executing the code of Augmint and the architecture simulator. This approach differs from those of MINT and Tango Lite. MINT uses one thread to represent all the components of the simulation system, including interpreting the application code for all application processes. We found that MINT's approach allows easier debugging of architecture simulators. However, we could not use this approach in Augmint because of the difference between MINT's interpretation versus Augmint's native-execution approaches. Augmint's approach allows similar simulator debugging capabilities to those of MINT.

Tango Lite, instead, represents a simulated application thread with one thread that executes the application code, the internals of Tango Lite, and the architecture simulator when the application thread generates events for the simulator. This approach makes it more difficult to support multiple concurrent simulated activities on behalf of simulated application threads (e.g. concurrent memory references under a relaxed memory consistency model).

As for the application programming model, Augmint supports a thread-based model. All application threads share a single address space, and only stack variables are private. This is another difference from MINT which supports a process-based model, where each simulated process has a separate address space. Only data allocated as shared can be accessed by more than one process. This model allows MINT to support many existing UNIX programs. On the other hand, for native execution-based simulation tools like Augmint (and Tango Lite) it is difficult to adopt the process-based model as it requires the tool to distinguish the application address space from that of the simulation tool and the simulator. This would require the tool to understand and accordingly rely on object file formats, and hence reduce its portability to other operating systems.

Finally, Augmint ensures the reproducibility of simulation results by following MINT's approach to memory management. MINT directly manages and controls the location of the address space of the applications, while Tango Lite uses the memory-allocation system calls of the host operating system directly. Reproducibility is important for debugging architecture simulators. Otherwise, multiple simulation runs with identical application and simulation parameters may produce different results because the operating system does not necessarily return the same address for the same allocation sequence of different simulation runs. In addition to easing debugging, repeatability of memory allocation behavior is also desirable when comparing different architectures. It is possible to achieve reproducibility with Tango Lite but not without effort on the part of the simulator developers.

[3] For optimized performance, MINT uses native execution for long code blocks between interesting event generating points in the application code. Otherwise it uses interpretation.

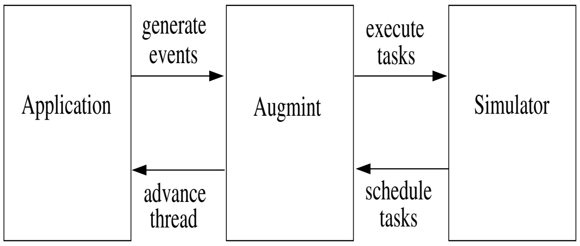

Figure 1. Structure of the application, Augmint, and simulator in the Augmint simulation environment.

3 Simulation Structure

The Augmint tool consists of a compile-time component and a run-time environment. The compile-time component prepares applications for simulation by augmenting them with instrumentation code. In the rest of this paper, Augmint refers to the run-time environment of the Augmint simulation toolkit.

As shown in Figure 1, the Augmint simulation environment is composed of three modules: an augmented application, Augmint, and an architecture simulator. The augmented application consists of the code of the application, augmented with instrumentation code that switches context to the Augmint/simulator thread at specific events such as memory references, synchronization events, and user-defined activities. Augmint handles these events, and typically creates and schedules tasks on a centralized task queue for these events to be simulated by the simulator in simulated time order. Augmint also schedules application threads, and maintains simulated time and the correct interleaving of memory activities. It also handles memory allocation, locks, barriers, and semaphores. The architecture simulator consists of functions that define the actions and delays associated with tasks called by Augmint. The simulator also defines other tasks and can schedule them by calling task management functions provided by Augmint.

The code of the three modules–the augmented application, the simulator, and Augmint–are linked into one executable. In the initial phase of the simulation, the Augmint thread allocates application thread structures, creates shared and private address spaces and other resources, initializes resources in the simulator, and sets up a central task queue. All simulation tasks are scheduled on this task queue and executed in the chronological order of simulated time. Then, Augmint schedules a task that switches context to the main application thread, as the first task in the task queue. When this task is extracted and executed, control is passed to the main application thread which executes the application code until an event is generated. At this point, the main application thread switches context back to the Augmint thread which handles the generated event. Based on this event, a task is created and inserted into the task queue to be executed at a specific simulated cycle. When this task becomes the earliest one on the task queue, Augmint extracts it and executes a function associated with the task. Such function can generate and schedule more tasks on the task queue. When a task completes, it can allow the corresponding application thread to advance by signaling Augmint to switch context to the corresponding application thread. Alternatively, it can signal Augmint to extract the next task in the task queue and execute it. To create a new application thread, the parent thread generates a "create" event and switches context to the Augmint thread. Augmint handles the event by creating a new thread and scheduling a task on the task queue that will eventually context switch to the new thread.

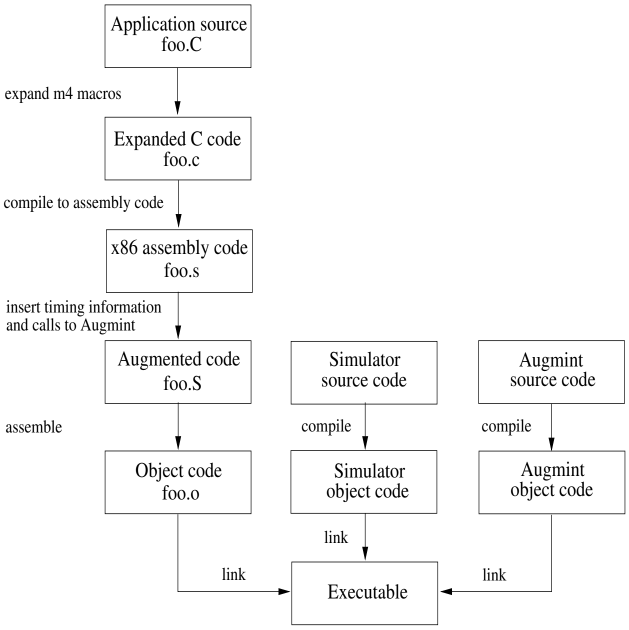

Figure 2. The procedure for generating a simulation executable.

There are several steps involved in building an Augmint-based executable, as shown in Figure 2. The application source code is preprocessed by an m4 macro processor to translate parallel constructs expressed as m4 macros [6] into events and context switches for Augmint. A GNU C compiler compiles the resulting source code into x86 assembly code, which is augmented (by Augmint's x86 augmenter) with instrumentation code that updates the simulated clock at the end of each basic block to account for the execution time of instructions within that basic block. The augmenter also instruments each memory reference with instructions to calculate the data address, size, and value, to generate an event for Augmint, and to switch context to the Augmint thread. Finally, the augmented code is assembled into object code and linked with the object code of Augmint and the simulator to produce a single executable.

4 Simulated Applications

Augmint accepts applications written in C/C++. Memory allocation functions and parallel constructs such as thread creation, locks, barriers, and semaphores are expressed in m4 macros in a fashion similar to applications in the SPLASH [9] and SPLASH-2 [13] suites. For example, a new thread is spawned with the m4 macro CREATE; a lock is acquired with LOCK; a barrier is called with BARRIER; and global shared memory is allocated with G_MALLOC. In addition, m4 environment macros such as MAIN_ENV must be included at the top of the application file to include necessary header files containing function prototypes and data types and structures for the interface with Augmint.

Augmint requires the wrapping of AUG_OFF and AUG_ON around m4 macros that return values, to prevent the augmenter from instrumenting instructions that access Augmint variables. For instance, G_MALLOC returns the address of an allocated block of memory and therefore must be wrapped by AUG_OFF and AUG_ON macros. If there is a contiguous block of G_MALLOC calls as typically found in SPLASH and SPLASH-2 applications, the whole block needs only one pair of AUG_OFF and AUG_ON. The corresponding simulator tasks for G_MALLOC can account for simulated time spent in calling the memory management system.

Augmint provides the simulator writers with a versatile macro, GENERATE_USER_EVENT, which generates user-defined events from the application to the simulator through Augmint. For example, it can be used to simulate prefetch instructions.

5 Interface between Augmint and the Architecture Simulator

As mentioned in section 3, when handling application events, Augmint schedules tasks on the task queue. All of the functions associated with these tasks are defined in Augmint. Some are internal to Augmint, while the others can be redefined in the simulator. The latter functions serve to allow Augmint to pass information about the events to the simulator. The most frequently called of these functions are those that correspond to memory references. In response to every read event reported by the augmented application, Augmint schedules a task that calls the function `sim_read` that is (usually) defined by the simulator. Similarly, Augmint calls `sim_write` on every write event and `sim_user` for every invocation of GENERATE_USER_EVENT.

Augmint provides stubs for the standard task functions (such as `sim_read`, `sim_write`, `sim_user`, `sim_lock_attempt`, etc.) with minimal functionality that allow the application thread to proceed, in case these functions are not defined by the simulator. However, in any non-trivial simulator, the simulator developer is expected to redefine most of these functions with more elaborate functionality. This functionality in most cases requires the capability to create and manipulate simulator-defined tasks (as opposed to the standard Augmint-defined tasks). Augmint provides the simulator with these capabilities in addition to other utilities such as hash tables, queues, and debugging handles as part of the interface between the simulator and Augmint. One of the most commonly used of these task management functions is `sched_task`, which creates a new task and schedules it for execution at some simulated time. Associated with this task is a simulator-defined function that will be invoked when the task is executed.

Another important component of the interface between the simulator and Augmint is the set of return values of task functions, which allow the simulator to block and unblock application threads.

6 Trace-Driven Simulation

Although Augmint is an execution-driven simulation tool, it can be easily used to drive simulators with traces. The simulator developer can implement a simple "application" that reads in instructions or memory references from a trace file, decodes the instructions, and generates events for Augmint. By shifting the trace capability to an application that reads and decodes traces, Augmint allows complete flexibility for supporting any trace file format.

A skeleton sample application that reads instructions from a trace file and generates events for Augmint can have the following structure:

```
AUG_OFF
while (get_next_instruction() != EOF) {
    decode_instruction();
    generate_augmint_event(...);
}
AUG_ON
```

The above loop reads instructions from the trace file using a function `get_next_instruction`. Each instruction is decoded by a function `decode_instruction`, which determines the type of the instruction. For example, if it is a load or store, the address of the data location is calculated and the data is extracted. If it is a branch, the target address is extracted. If the decoded information indicates that the instruction is an event of interest to the simulator, a function `generate_augmint_event` can call Augmint's macro

GENERATE_USER_EVENT to deliver the event to the simulator through Augmint.

The application can spawn multiple threads that execute the loop above to read multiprocessor traces and simulate multiple instruction streams.

The option to run Augmint in a trace-driven mode is particularly useful for users who would like to use their simulators with both applications and traces collected from other systems.

7 Conclusions

This paper describes the main characteristics of Augmint, an execution-driven simulation tool for developing multiprocessor simulators for x86 architectures. Augmint's compile-time component augments the application with instrumentation code to update the simulation clock and generate events for simulation. This augmentation approach gives Augmint the flexibility to run on different operating systems such as Solaris, Linux, and Windows NT. Augmint supports an interface that is compatible with that of MINT, thus it can run most architecture simulators written for MINT. Augmint runs applications from the SPLASH and SPLASH-2 suites and any application written in C/C++ and m4 macros in a fashion similar to SPLASH and SPLASH-2 applications. Augmint supports a thread-based programming model with shared address space and a private stack space per thread. Augmint supports a powerful set of debugging capabilities that facilitates the development of simulators. The source code of Augmint is publicly available from http://www.csrd.uiuc.edu/iacoma/augmint.

Acknowledgements

We are grateful to John Cabarjal and David Archer from Intel Corporation for their support in the development of Augmint.

References

[1] E. A. Brewer, C. N. Bellarocas, A. Colbrook, and W. E. Weihl. Proteus: A High-Performance Parallel-Architecture Simulator. Technical Report LCS/TR-516, MIT Laboratory for Computer Science, September 1991.

[2] S. Goldschmidt. *Simulation of Multiprocessors: Accuracy and Performance*. PhD thesis, Stanford University, June 1993.

[3] S. R. Goldschmidt and J. L. Hennessy. The Accuracy of Trace-Driven Simulations of Multiprocessors. Technical Report CSL-TR-92-546, Stanford University, Computer Systems Laboratory, September 1992.

[4] S. A. Herrod. Tango Lite: A Multiprocessor Simulation Environment. Technical report, Stanford University, Computer Systems Laboratory, November 1993.

[5] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam. The Stanford DASH Multiprocessor. *IEEE Computer*, pages 63–79, March 1992.

[6] E. Lusk, R. Overbeek, et al. *Portable Programs for Parallel Processors*. Holt, Rinehart, and Winston, Inc., New York, NY, 1987.

[7] A. Sharma. Augmint, a Multiprocessor Simulator. Master's thesis, University of Illinois at Urbana-Champaign, 1996.

[8] A. Sharma, A.-T. Nguyen, J. Torrellas, M. Michael, and J. Carbajal. Augmint: a Multiprocessor Simulation Environment for Intel x86 Architectures. Technical Report 1463, University of Illinois at Urbana-Champaign, March 1996.

[9] J. Singh, W.-D. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. In *Computer Architecture News*, March 1992.

[10] A. Srivastava and A. Eustace. ATOM: A System for Building Customized Program Analysis Tools. *SIGPLAN Notices*, 29(6):196–205, June 1994.

[11] J. E. Veenstra and R. Fowler. MINT Tutorial and User Manual. Technical Report 452, University of Rochester, June 1993.

[12] J. E. Veenstra and R. J. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In *Proceedings of the Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, pages 201–207, Durham, NC, January 1994.

[13] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *22nd International Symposium on Computer Architecture*, 1995.