6

EXECUTION DRIVEN SIMULATION OF SHARED MEMORY MULTIPROCESSORS

Bob Boothe

University of Southern Maine, Portland, Maine

1 INTRODUCTION

Execution driven simulation[7] is a technique for building fast instruction level computer simulators. It is applicable when the instruction set of the simulation host machine is the same as, or very similar to, that of the machine being simulated. In this chapter we examine three execution driven simulators designed to study shared memory multiprocessors using a uniprocessor as the simulation host machine.

In building a simulator one can take advantage of the fact that some events are of greater interest than others. For instance a simulator of shared memory multiprocessors is primarily concerned with the load and store instructions which access memory. For other instructions, such as arithmetic and control, the only concern is that they get performed and that their execution time is properly accounted for. The key idea of execution driven simulation is that rather than simulate each individual instruction, the bulk of the instructions can be directly executed by the host computer. Only those instructions requiring special treatment need to be simulated.

The simulation involves two stages: first a preprocessing of the application and then the actual simulation. In the preprocessing stage, the application program is modified by inserting extra instructions that will perform simple operations needed by the simulator and by inserting calls to simulator routines at important events. Figure 1 shows a simple example. Here the register `Rtime` is used to accumulate the execution time (in processor cycles) as the program is executed. Each application instruction is followed by an extra instruction that increments this time register. The load word instruction (`lw`) in this

T. M. Conte et al. (eds.), *Fast Simulation of Computer Architectures*

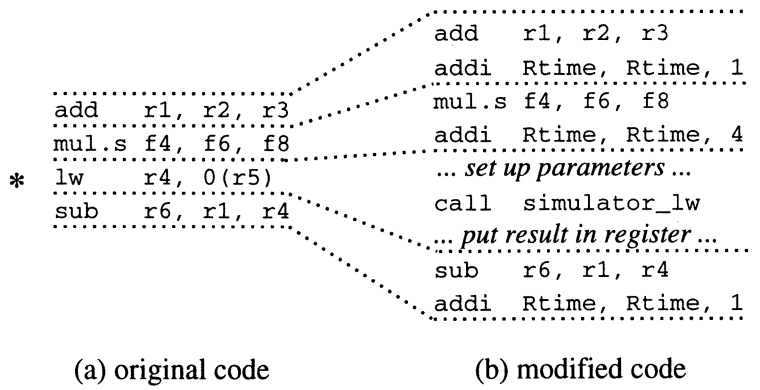

Figure 1 A simple example of code augmentation. The `lw` instruction (marked with the asterisk) causes an event of interest to the simulator, and thus in the modified code it is replaced with a call to the simulator. All examples in this chapter use the MIPS instruction set.

example causes an event of interest to the simulator; it is replaced by a call to a simulator routine. When the modified application is executed, these inserted calls will feed events to the simulator. This general technique is called *execution driven simulation* because it is the execution of the modified application and the inserted calls that drives the simulation process.

The preprocessing stage of modifying the application code is called *augmentation*. It is generally done at the machine language level on either object files or the executable file. In general it is more sophisticated than in the example. For instance, a simple improvement can eliminate most of the time counting instructions. Instead of inserting a time counting instruction after every original instruction, one time counting instruction can be used for an entire basic block. (A basic block is a group of contiguous instructions that is always executed in sequence. The only jumps into the block are to the first instruction. The only jumps out are from the last instruction.) The single time counting instruction for a basic block would update the time counter by the sum of the times of its component instructions.

The two augmentations seen so far, time counting and event call-outs, form the basic mechanism used by an execution driven simulator. Later we will see several other useful augmentations.

Code augmentation is currently an important technique for building other tools besides simulators. Profilers such as `pixie`[12] augment the application code