

# The RSIM Simulation Environment

September 10, 2003

RSIM (formerly known as Rice Simulator for ILP Multiprocessors) provides a detailed means to quantitatively analyze the behavior and performance of multiprocessor and uniprocessor systems built from modern multiple-issue, dynamically-scheduled processors. RSIM has been extensively tested and is publicly available for non-commercial use. This document covers relevant information for this homework; more extensive documentation is available in the RSIM manual at <http://www.ece.rice.edu/~rsim/Manual/manual.html>

For this homework, the RSIM system is available in the directory `/home/elec526/rsim/` on Owlnet; you are also free to copy this directory hierarchy to any other Rice machine that you choose. Note that this version of RSIM is more recent than the one distributed at the RSIM home page, so you should not use the official RSIM version. This version of RSIM includes support for instruction caches, TLB, and VIS instructions, a fast emulation mode, as well as other features that improve reliability. The current version is maintained by Chris Hughes at UIUC ([rsim@ece.rice.edu](mailto:rsim@ece.rice.edu)).

## 1 RSIM Environment Overview

RSIM simulates the execution of a parallel program on a parallel target system while running on a uniprocessor host. It is a detailed execution-driven simulator and accurately models all the contention in the target system. The RSIM environment consists of two elements: a parallel application to be simulated and the simulator. The simulator consists of three important modules: the processor, the memory system and the network. The processor module simulates the decoding, issue, execution and retirement phases of the pipeline for each instruction. The memory system module simulates the behavior of the memory hierarchy including caches, write buffers, coherence protocol, and the main memory. Finally, the network module is used to simulate the interconnection network using user-defined topologies and routing algorithms.

The base architecture simulated by RSIM is a distributed shared-memory system that uses a release-consistency memory model. The network topology is a 2D-mesh with separate request and reply networks to avoid deadlock. Each processor node in the system contains an ILP processor, a lockup-free primary cache, a write buffer (only if the primary cache is write-through), a lockup-free secondary cache, a directory, a portion of the global shared-memory, and a network interface (see Figure 1).

The user can vary many parameters to study the effect of different system configurations on the execution of the application. The next sections discuss a few important parameters that could be useful for this homework.

## 2 System Parameters

The relevant parameters of the default RSIM configuration are given in Table 1.

## 3 Command-line and Configuration File Options

Each of the listed parameters can be varied by command line or configuration file parameter options to RSIM. The important parameters for this homework are discussed below. The full set of command line and configuration file parameters for RSIM are available in Chapter 3 of the RSIM manual, available from the WWW at <http://www.ece.rice.edu/~rsim/Manual/node28.html>

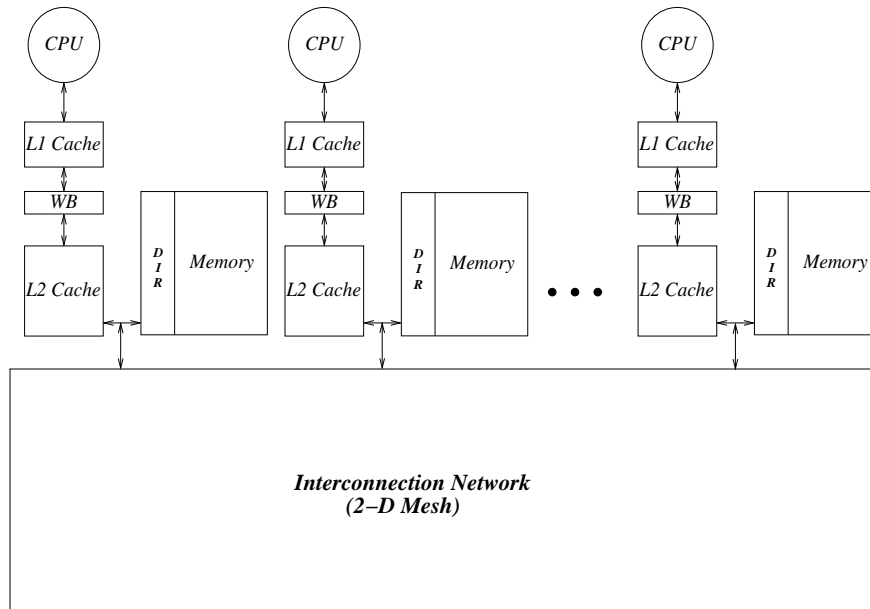


Figure 1: Base Architecture Block Diagram

Some sample configuration files are available in `/home/elec526/rsim/inputs/`. Please note that you can also generate the configuration file by using the javascript utility available at `file:/home/elec526/rsim/inputs/rsimconfig.html`.

The most important command line options for this homework are:

- a <num> Instruction window size
- i <num> Instruction issue width
- H <num, num> Number of MSHR's (cache miss buffers) at L1 and L2 respectively, defaults to 8,8
- f <app> Name of application to simulate.
- D <dirname> Directory in which to dump output and statistics files. Note that RSIM output files can be very large, and so all members of the group must share their disk space and use it judiciously. You could also create a subdirectory in `/tmp` and use that as output space. However, files in `/tmp` may be deleted at any time, and are thus not safe for long-term use.
- S <filename> File name prefix for full application output and statistics dumps. Each test run should use a different name for this option. RSIM produces three files: *filename\_out*, which contains the application standard output; *filename\_err*, which contains the application standard error and the "concise" statistics of RSIM; and *filename\_stat*, which contains the detailed statistics of RSIM.

Important configuration file parameters for this homework could be cache size, cache block size, cache access latency, cache associativity, bus width, memory access time, etc. More information is available at `file:/home/elec526/rsim/inputs/rsimconfig.html`.

<b>ILP Processor</b>	
Processor clock speed	300 MHz
Instruction issue width	4
Instruction window size	64
Functional units	2 ALU, 2 FPU , 2 Load/Store
ALU latency (logic, add, subtract, shift)	1 cycle
ALU latency (multiply)	3 cycles
ALU latency (divide)	9 cycles
ALU repeat rate	1 cycle (fully pipelined)
FPU latency (add, subtract, multiply)	3 cycles
FPU latency (move, negate, absolute value)	1 cycle
FPU latency (conversions)	4 cycles
FPU latency (divide)	10 cycles
FPU latency (square root)	10 cycles
FPU repeat rate (add, move, etc.)	1 cycle (fully-pipelined)
FPU repeat rate (conversions)	2 cycle delay
FPU repeat rate (divide, square root)	6 cycle delay
Memory unit size	32 entries
<b>Network parameters</b>	
Network speed	150MHz
Network width	64 bits
Flit delay (per hop)	2 network cycles
<b>Cache parameters</b>	
Cache line size	64 bytes
L1 cache (on-chip)	Direct mapped, 16 K
L1 request ports	1
L1 hit time	1 cycle
Number of L1 MSHRs	8
L2 cache (off-chip)	4-way associative, 64 K
L2 request ports	1
L2 hit time	8 cycles, pipelined
Number of L2 MSHRs	8
<b>Memory parameters</b>	
Memory access time	18 cycles (60 ns)
Memory transfer bandwidth	16 bytes/cycle
Memory Interleaving	4-way

Table 1: Default hardware parameters for RSIM

## 4 Writing applications for RSIM

This section briefly discusses the issues related to writing applications for RSIM. More detailed information is available at <http://www.ece.rice.edu/~rsim/Manual/node46.html>

For a sample application program and makefile please refer to the quicksort application available at `/home/elec526/rsim/apps/QS`. Note that it is also important to use the `SUNWspr0-5.1` compiler series or older, because later versions use some 64-bit instructions that are not supported. This compiler suite is available on OwlNet, and you should add `/usr/site/SUNWspr0-5.1/bin` to the front of your `PATH` environment variable. (This compiler version may not be available on some research machines, but you can always build application executables on OwlNet and then move them over to other machines, since they are statically linked.)

For locations that are read/written by multiple processors, the simulator has to accurately simulate cache behavior including cache-coherence actions and network latency. Shared memory must be allocated through the `shmalloc` function (which has syntax similar to `malloc`). There is no corresponding free function. Memory allocated through the ordinary `malloc`, as static variables, or on the stack will be private to a node and noncoherent. In order to distribute such locations across the nodes, the function `AssociateAddrNode(startaddr, endaddr, node, description)` should be used. All accesses to locations from `startaddr` to `endaddr-1` will be simulated as

if node was their home node on the target architecture. (Note: AssociateAddrNode has a granularity of a virtual memory page, 4 KB.)

As an example, the following code shows one way to create an array of 4k integers, and distribute it in memory among 4 processors.

```
int *intarray;
intarray = shmalloc(sizeof(*intarray)*4096); /* allocate memory */
for (nodenum=0; nodenum<4; nodenum++) {
    AssociateAddrNode(&intarray[nodenum*1024], /* start address */
                    &intarray[(nodenum+1)*1024], /* ending address */
                    nodenum, /* home node */
                    ``intarray``); /* just a string description */
}
```

**Process Creation.** At the beginning of execution, RSIM starts an application with a single processor in the specified architectural configuration. The application must then use the fork() system call to spawn off new processes, each of which is run on its own processor. The semantics of fork() are identical to those of UNIX fork(). In the context of RSIM, fork() causes the new processor to have its own copy of the application code segment, global and statically allocated variables, private heap, and process stack, but the new processor has the same logical version of the shared portion of the address space. Since RSIM currently does not support multitasking, indeterminate results will arise if more processes are started than the number of processors specified in the configuration file. A simulated process can call getpid() to determine its processor number.

**Synchronization support for parallel applications.** The RSIM applications library supports three types of multiprocessor synchronization primitives - locks, flags (or pauses), and barriers. To use the lock and flag synchronization functions, the application must include the header file locks.h; for barriers, the application must include treebar.h.

For further information, please refer to the manual at <http://www.ece.rice.edu/~rsim/Manual/node46.html>.

## 5 Running RSIM

Arguments to the simulator are given on the command line first, followed by a double dash, followed by arguments to the application being simulated.

For example, if the merge sort application accepted arguments “-p<sub>i</sub>processors<sub>i</sub> -n<sub>i</sub>datasize<sub>i</sub>”, it could be run with the application arguments -p4 -n8192 specifying a run with four processors and an 8192 element array to be sorted. Matrix multiplication would require number of processors, number of rows, and number of columns as its command line arguments. Thus, to simulate Merge sort with 4 processors with a 128 element instruction window and 8 way instruction issue, and to dump simulation outputs in the directory ~/testoutputs with the filename prefix MSa128i8p4, the following command should be used :

```
~elec526/rsim/obj/native/rsim -S MSa128i8p4 -D ~/testoutputs
-o ~elec526/rsim/inputs/sample -a128 -i8
-f <path for the application> -- -p4 -n8192
```

## 6 Analyzing Statistics Generated by RSIM

The statistics files generated by RSIM tend to be large and difficult to interpret in raw format. Additional utility scripts are provided in the /home/elec526/rsim/bin directory to condense the information in these files into a more readable format.

The most important script provided is the pstats script, which gives the statistics for a “phase” of application execution. These statistics give the execution time in cycles, the number of instructions per cycle, and the relative

contribution to cycle count of each type of operation (Busy time, ALU time, FPU time, Read time, Write time, and Multiprocessor synchronization).

Note that in ILP processors, the relative contribution of each execution time component is difficult to gauge precisely, since instructions can overlap with each other and execute out of order. Thus, we breakdown execution time into its components by charging each instruction according to the number of cycles it prevents RSIM from retiring instructions at its peak retirement rate; cycles in which RSIM retires instructions at peak rate are considered *Busy* cycles.

To process one or more files using `pstats`, use the command:

```
~elec526/rsim/bin/pstats file1_err [file2_err file3_err ...] phase
```

where `fileN_err` represents the `_err` files produced by RSIM and `phase` is the number of the critical phase of the application. It is often useful to divide the application into various phases (or parts) to separate the statistics of critical parts from other parts of the application. An application can be divided into various phases by using the following functions.

```
newphase(int phasenum);
/* This function starts a new phase with the phase number indicated by
phasenum. Each application starts in phase 0 by default*/

endphase();
/* This function ends the current phase and prints out statistics for
the current phase. It should be called before a new newphase function */
```

For further details, please refer to the manual at  
<http://www.ece.rice.edu/~rsim/Manual/node50.html>.

The following command invokes `pstats` corresponding to the RSIM command line given in the previous section, assuming that the critical phase number is 1. NOTE: You must have the `/home/elec526/rsim/bin` directory in your `PATH` environment variable before you can use `pstats`.

```
pstats ~/testoutputs/MSa128i8p4_err 1
```