PSATSim: An Interactive Graphical Superscalar Architecture Simulator for Power and Performance Analysis

Clint W. Smullen, IV
Clemson University
Dept. of Electrical and Computer Engineering
Clemson, SC 29634
csmulle@clemson.edu

Tarek M. Taha
Clemson University
Dept. of Electrical and Computer Engineering
Clemson, SC 29634
tarek@clemson.edu

Abstract

Two of the most important design issues for modern processors are power and performance. It is important for students in computer organization classes to understand the tradeoff between these two issues. This paper presents PSATSim, a graphical simulator that allows student to configure the design of a speculative out-of-order execution superscalar processor and see the effect of the design on both power and performance. The simulator explicitly shows the relationship between instructions within a processor by visually tagging instructions. The use of a graphical simulator makes it simple for instructors to demonstrate the execution of instructions within these architectures and the interactions among processor components.

1. Introduction

Given the emphasis on speculative out-of-order execution superscalar processors and power consumption in industry, it is important to introduce and cover these topics in computer organization classes. Ideas that students need to understand include the flow of instructions within the pipeline, dynamic scheduling, speculative execution, and the data dependencies between instructions. Students also need to appreciate the power consumption which offsets performance gains within processors. It is difficult to explain these concepts and the interactions among the architecture components without visual aids. Graphical simulations of these architectures allow students to easily grasp the concepts of the architectures by observing the flow of instructions in time. They allow students to explore the impact of different architecture options on the performance and power consumption of a processor.

The majority of existing graphical simulators target simple processor architectures. For instance, WinDLX [4] models the scalar pipelined architecture described in [5]. The IJVM simulator [8] models the stack based JVM described in [6]. Existing superscalar processor simulators with graphical front-ends such as SuperSim [7] and the Simplescalar Visualizer [10] model processors in great detail, but do not show the dependencies between instructions within the datapath that contribute to different instruction flows. At present the only classroom tool for modeling the power consumption of computer architectures is Quilt [2]. However, it does not graphically show the flow of instructions in a processor.

This paper presents PSATSim, a graphical simulator that explicitly shows the relationship between instructions within a superscalar out-of-order execution processor. It indicates both the power consumption and performance of a given architecture configuration. The simulator is based on ideas in the SATSim simulator described by Wolff and Wills [12]. Our simulator provides a wider variety of configuration options, explicitly shows speculative execution, provides a clear layout of the processor components, and provides power consumption values at the end of execution. PSATSim has been designed for ease of use and installation. Color tagging of instruction registers makes it easy to follow the flow of instructions. Incorrectly speculated instructions are indicated on screen, making it possible to see the effect of misspeculation. The simulator is trace driven, with the traces generated using SPEC benchmarks on the SimpleScalar simulator [10]. PSATSim is currently used in undergraduate and graduate courses in computer architecture at Clemson University.

2. Simulator Features

PSATSim models the dynamic power consumption of superscalar processor architectures. It incorporates a speculative execution model which gives relative accuracy to the power values. PSATSim allows students to experiment

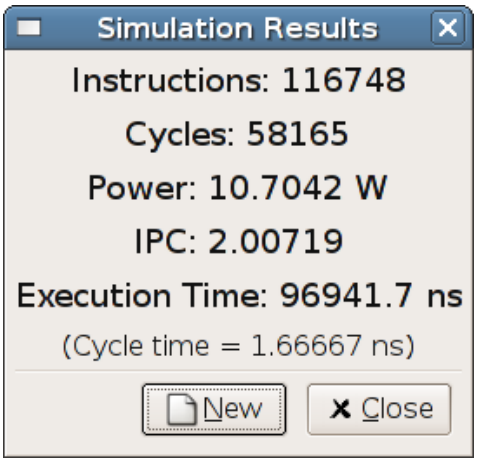


Figure 1. Example Execution Results

with a wide range of architectural parameters and to observe both the power and performance results of the configuration. An example of the results screen from an execution of PSATSim is shown in Figure 1. The following sections will describe the architecture used by PSATSim, the model of speculative execution, and the power model.

2.1. Modeling instruction execution

PSATSim models a superscalar processor with out-of-order speculative execution. Out of order execution allows instructions not waiting on data values to execute ahead of other instructions. Speculative execution allows the processor to speculate on branch outcomes and continue executing instructions, without disrupting the final state of the processor. Figure 2 shows the graphical view of the processor presented by PSATSim. Instructions flow downward in this view. The architecture closely models the superscalar architecture described by Shen and Lipasti [9]. The architecture can be divided into three parts: the in-order front end, the out-of-order core, and the in-order back end (Table 1 shows the components within each part). Users are able to modify the configuration of the processor and view the effects on instruction flow through the simulator. At the end of execution, a results screen is displayed including the power consumption and the execution time. Configuration options are described in detail in section 3.

The three stages in the front end (fetch, decode, and dispatch) have an in-order flow. As instructions enter the processor through the fetch stage, they are serially numbered and color coded to easily identify their progress through the architecture. After moving through the decode and dispatch stages, instructions enter the out-of-order execution core. The branch predictor is not shown in the graphical view as it is modeled statistically. Since the simulator

Table 1. Components within the simulated architecture.

Part of processor	Components that can be configured
Front end	Fetch, decode and dispatch stage widths, branch prediction accuracy, memory hit rate and access time.
Out of order core	Execution units, renaming table, reorder buffer, and reservation stations.
Back end	Commit buffer width.

is trace driven, the outcome of all branch instructions are known a priori. This allows users to try different predictor characteristics and observe its effect on the architecture. The same applies to the memory hierarchy.

In the execution core, instructions are simultaneously entered into the reorder buffer and an available reservation station. They are also renamed through the renaming table and register remapping table (these tables can be minimized to reduce on-screen clutter if needed). The reservation stations indicate the instruction opcode and the source and destination registers. These registers are indicated by the serial number of the instruction producing them. If their value is not available, the registers are also color coded to their producer instruction's color. Once the producer instruction completes execution, the register color is removed to indicate that its value is available. This makes it easy to identify the data dependencies between instructions in the different reservation stations and functional units. Once all of the input values are available and the front pipeline stage on an appropriate associated execution unit is free, the instruction begins execution. The instruction continues to occupy its reservation station until the instruction's execution has completed. An execution unit may be pipelined or have a multi-cycle delay. Each stage in an execution unit pipeline is shown, while multi-cycle delays are only indicated by the multi-cycle residency in the execution unit.

After an instruction finishes execution, it is removed from the reservation station. In addition, the corresponding entries in the renaming table and reorder buffer are uncolored. This makes it easy to determine which instructions are ready to commit from the reorder buffer. The reorder buffer is placed on the left side of the display to provide an easy reference to all the instructions as they flow through the datapath.

Memory access occurs in the last pipeline stage of the memory unit. Therefore, if there is a data cache miss, every memory instruction in the execution unit will be delayed. The memory hierarchy is not visualized in the interface,

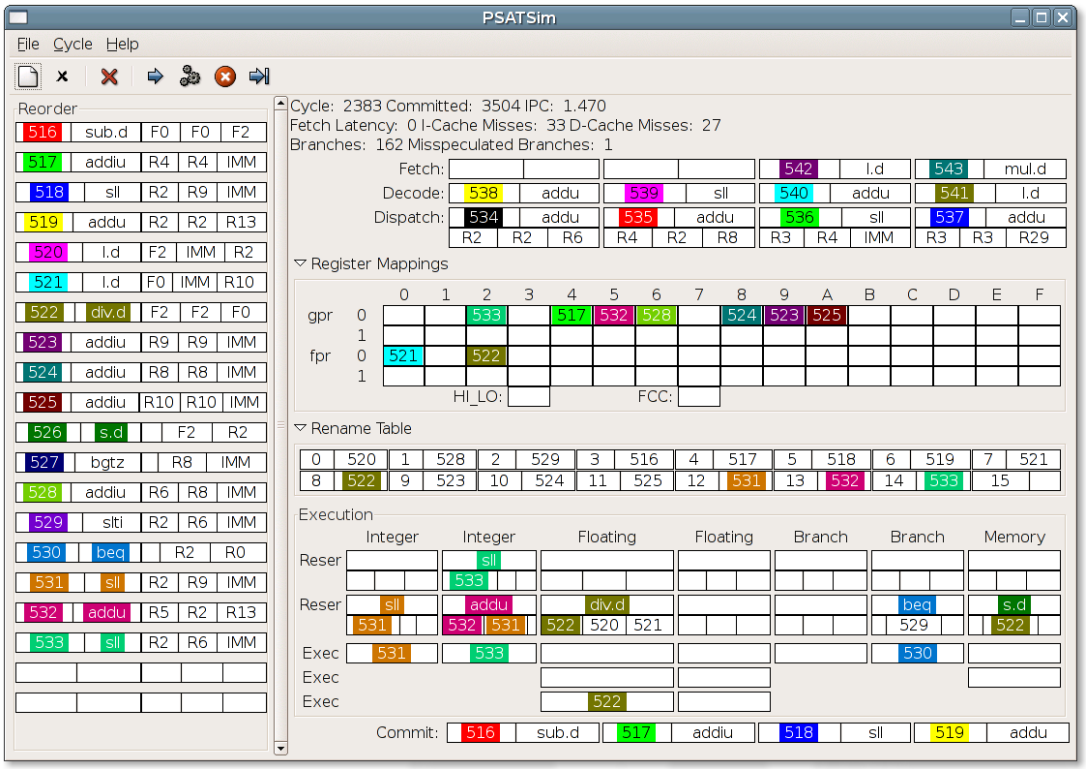


Figure 2. Normal Execution

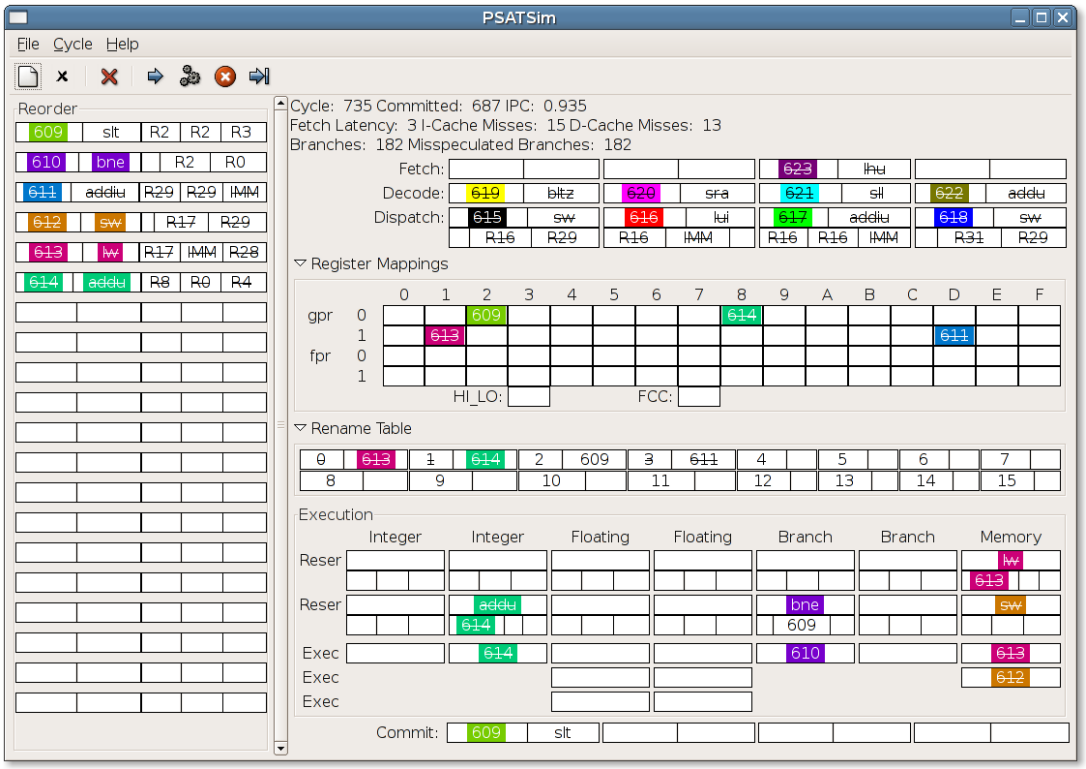


Figure 3. Speculative Execution

though cache misses in each level are listed in the running statistics at the top of the simulator. The statistics at the top of the window also list the current cycle number, the number of instructions committed, the IPC, the number of cycles remaining on a fetch (shown as 'Fetch Latency'), the number of branches encountered, as well as the number of misspeculated branches.

2.2. Modeling speculative execution

PSATSim provides the option of simulating the effect of speculative execution, resulting in significantly more accurate power modeling. Since the simulator is trace driven, the instructions that would be fetched on a branch misprediction are not available in the trace. However, Bhargava et al. [1] show that it is possible to model the effect of misprediction by fetching instructions from the main trace file. This is due to the fact that a large percentage of the instructions fetched during misspeculation are fetched normally at other points during execution. In other words, instructions that would have been executed during a misspeculation are similar to those in the input trace which is already being utilized. PSATSim simulates the effect of misspeculation by loading instructions from the same trace file independent of the normal stream of execution. This provides the same general mix of instructions which would have occurred if the actual program had been executed and accounts for the energy consumed by instructions which, in the end, have no effect on the architectural state of the processor.

If the simulation of misspeculation is enabled, the misspeculated instructions are shown with a strikethrough on the display. Figure 3 shows an example of the simulator in this state. As with any architecture allowing out-of-order speculative execution, it is necessary to commit the instructions in order; in PSATSim, this is accomplished using a reorder buffer. Each cycle, up to the superscalar width of finished instructions in the reorder buffer will be committed, thereby permanently affecting the processor's state. The instructions that will be committed in the current cycle are shown in the bottom row of the display in the 'Commit' row. Though misspeculation visualization may be disabled, the reorder buffer is always used to ensure that instructions are committed in-order.

2.3. Modeling power consumption

PSATSim incorporates the Cacti-based [11] Wattch high-level power modeling system [3]. Though the Wattch power model has been designed for the SimpleScalar architecture, we have adapted it to support the architectural flexibility of PSATSim. Because of the large number of differences between the two simulators' architectures, power values cannot be directly compared between Wattch and PSATSim.

The Cacti system was designed for computing delay-optimal cache structures. The delay comes in the form of the effective resistance and capacitance of the cache. In computing dynamic power, all that is needed is the effective capacitance of a structure. The energy is consumed during switching of the transistor states, that is, in charging and discharging the transistors. The resistance is not needed, since it principally affects the maximum clock speed at which a circuit will operate. The Wattch power model uses Cacti, and its implementation of known semiconductor manufacturing process parameters, to compute the effective capacitances for a wider range of processor structures [3].

The dynamic power is defined by Wattch as: $P_d = C V_{dd}^2 a f$. The exclusion of a model for static power does not significantly impact the accuracy of PSATSim due to the use of 350 nm technology process parameters. The capacitance, C, is generated using Cacti based on the process parameters. The supply voltage, V_{dd}, as well as the clock frequency, f, are generally determined by the technology process used, though many architectural design decisions can significantly reduce the clock frequency. The activity factor, a, represents the average proportion of the structure that is switching each cycle. Modeling the activity factor well is the most difficult part of normalizing the power model to published power figures.

In modeling the effective capacitance for different components, there are three general categories which an architectural structure can fall into: array structures, content-associate memories, and complex logic blocks [3]. The last category is used to model functional units and will be discussed later. The first two categories are interrelated and are used to model the caches, the reorder buffer, the register renaming table, and the architecture register file. Array structures are modeled as sets of decoder, wordline drive, and bitline discharge circuitry, which consume most of the energy. Reservation stations are modeled as content-associative memories, which are similar to array structures, except that they use taglines and matchlines due to their fully-associative nature. The original implementation of Wattch inside the sim-outorder component of SimpleScalar [10] features a register update unit, which combines the features of the reorder buffer, a centralized reservation station, and the dispatch stage into a single unit. PSATSim models the structures separately, since that is much more common in industry. The PSATSim power model for each of these structures is correspondingly independent, resulting in a much wider range of possible configurations than with Wattch implemented in sim-outorder.

Caches are modeled using a hybrid between the two that is built out of array structure equations. This is due to the fact that caches are rarely fully-associative, due to their size and the need for performance. Since the memory hierarchy

is stochastically modeled in PSATSim, the cache size parameters which would actually provide the supplied hit rates is unknown. Therefore, it is necessary to assume certain cache sizings for the purposes of power modeling. Since PSATSim uses the baseline 350 nm technology parameters contained within the Wattch model, the two L1 caches are modeled as direct-mapped 16 kB each, while the L2 cache is modeled as 256 kB four-way set associative. These cache parameters are typical of processors made using 350 nm processes, as specified by Wattch [3].

For structures such as caches, which pre-charge and discharge the bitlines every cycle, regardless of being used, the activity factor is assumed to be 100%. For wide structures, such as the fetch, decode, dispatch, and commit stages, the activity factor can be scaled by the utilization. That is, if only one-half of the decode stage is being used, then the activity factor can be approximately halved. Wattch introduces a number of conditional clocking schemes designed to mimic modern processors. The most important of which models the activity factor as a linear function of the number of units being utilized. Even with no utilization, there is still some power consumed, due to clock circuitry. At this time, PSATSim scales the activity factor for all regular structures linearly from 10% to 100% (this is the same assumption used in Wattch). The 10% minimum is used by Wattch to account for transistors that cannot be disabled by aggressive conditional clocking [3]. If a structure is too complicated to be modeled in this way, then an activity factor of 50% is assumed. This gives an equal likelihood for each gate to be switched.

The selection logic, found with each reservation station, and the dependency check logic, found in the dispatch stage, both feature complex logic which have models which were used within Wattch [3]. Due to the complex nature of functional units, there is no way to construct a high-level model to support a wide range of functional unit types. It is therefore necessary to extract capacitance values for each functional unit type from low-level models for a specific technology process. In PSATSim, these values are stored, along with technology process parameters, within a file that defines the ISA and the capacitance of the circuitry to execute each instruction. At the moment, PSATSim uses extremely aggressive clocking which causes different opcodes that execute on the same functional unit to have different energy consumption.

3. Simulator Configuration

When the simulator is started up, the user is presented with a processor configuration dialog box to tailor the architecture. The configuration is divided into three main parts (through a tabbed interface): general, execution, and memory/branch. Figures 4, 5, and 7 show example screenshots

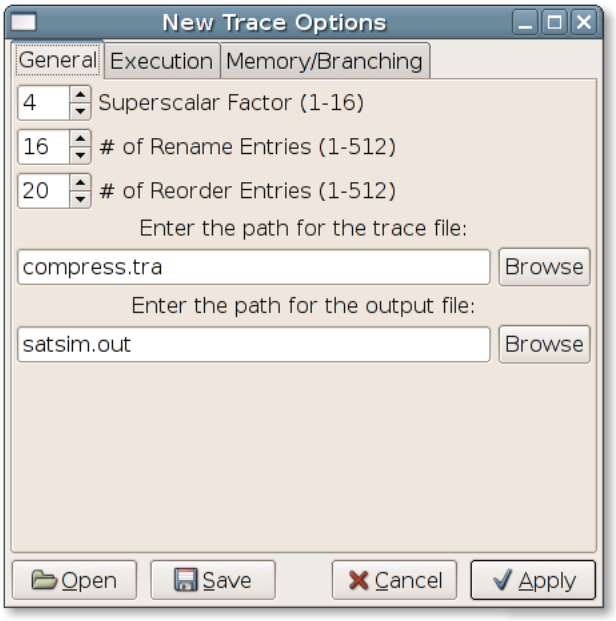


Figure 4. General Settings for a New Trace

of the three tabs. The settings specified can be saved and loaded later. The exact configuration used by the simulator is saved into the output file, including the random-number generator's seed, making it possible to exactly recreate the output. The configurations are stored in XML format, making it simpler to edit and view manually if necessary. The following sections discuss the three configuration tabs and how they relate to the architecture modeled.

3.1. The General Tab

On the general tab (see Figure 4), the superscalar width specifies the bandwidth of the front and back ends of the processor (fetch, decode, dispatch, and commit stages). The number of reorder buffer entries, renaming table entries, the input trace file, and the output results file are also specified here.

3.2. The Execution Tab

On the execution tab (see Figure 5), the user can configure the reservation station and the execution unit configuration. Three reservation station structure types (as defined in Shen and Lipasti [9]) are offered: distributed, centralized, and hybrid. In the first case each functional unit has its own reservation station, while in the centralized case all functional units share a common reservation station buffer (see Figure 6). In the hybrid case, a group of functional units shares a buffer, as in the MIPS R10000 [13].

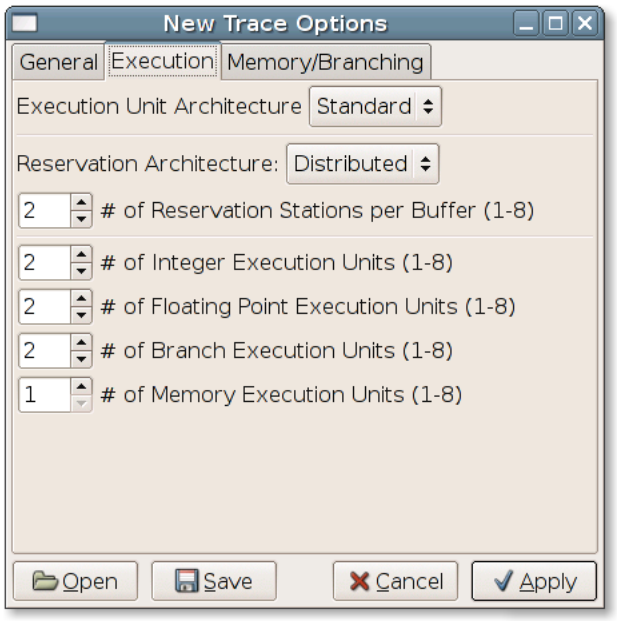
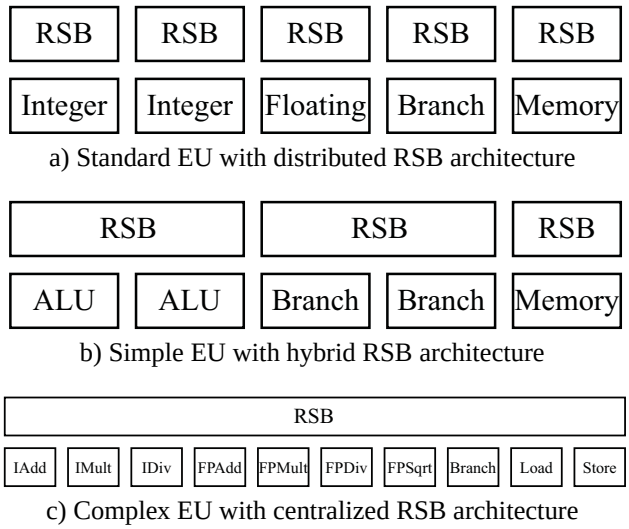


Figure 5. Standard Execution Architecture Settings for a New Trace



a) Standard EU with distributed RSB architecture

b) Simple EU with hybrid RSB architecture

c) Complex EU with centralized RSB architecture

Figure 6. Example EU and RS Configurations

Table 2. Functional Unit Types

Type	Functional Units
Simple	ALU, branch, and memory.
Standard	Integer, floating point, branching, and memory.
Complex	Integer addition / multiplication / divison, floating point addition / multiplication / division / square-root, branching, and load/store.

The execution tab also allows the user to configure the number of functional units. Depending on the details taught in class, students can pick different types of functional unit configurations: simple, standard, complex, and custom (see Table 2). In the simple case, all integer and floating point operations are carried out through a general ALU. In the standard case, separate integer and floating point ALUs are provided, while in the complex case, separate functional units are provided for each sub type of operation (such as floating point divide and floating point multiply).

In the custom case, the user can specify any arbitrary structure of functional units and reservation stations. It allows functional unit latencies and throughputs to be changed, along with the type of instructions executed by the units. In addition the functional units connected to each reservation station can be modified. The custom configuration is specified by hand editing an XML file.

3.3. The Memory/Branch Tab

The branch prediction accuracy and memory properties are defined in the memory/branch tab (see Figure 7). The user can choose to have just system memory, system memory with L1 caches, or system memory with both L1 and L2 caches. The access latency and hit-rate for each cache level is configurable, as is the branch speculation accuracy. A checkbox is provided to disable the visualization of branch misspeculation modeling. Since these components are modeled statistically, students can observe the effect of different branch hit-rates or memory latencies on processor performance.

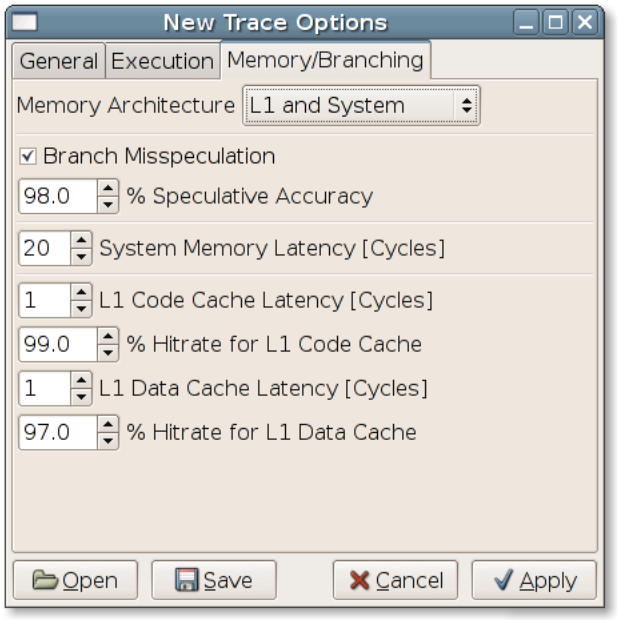


Figure 7. L1 and System Memory Architecture Settings for a New Trace

4. Implementation

PSATSim was implemented in C++. GTK+ v2 was used to provide the GUI functionality. The use of cross-platform libraries enables PSATSim to run under both Windows and Linux environments, as well as some other Unix-type systems, including Solaris and MacOS X. Full inheritance and polymorphism is used, allowing for ease of extension in the future. Each component in the system, be it an entry in the reorder buffer or an execution unit, is an independent object that is linked to other objects. This design allows for custom architectures and simplifies future expansion of the program.

The Windows installer for PSATSim uses the Nullsoft Scriptable Install System and was designed to conform to general norms of modern installers. Options are given to install additional trace files. An installer for GTK+ is included in the installer to help get PSATSim running on machines which do not have GTK+ installed already, which is a high probability for most Windows based machines. The simulator and it's associated trace files can be downloaded from:
`http://www.ces.clemson.edu/~tarek/psatsim/`

5. Use of PSATSim

PSATSim allows the user to explore different configurations of basic superscalar processors with ease. In a course on computer architecture, it can be used in class to visually demonstrate the flow of instructions in a modern processor. A large variety of superscalar architecture configurations can be demonstrated by modifying the configuration file. The color coding of instructions allows students to visualize the interaction between processor components. PSATSim provides the same structure as other simulators, but with a more flexible configuration system, allowing an instructor to both raise and lower the level of complexity to meet the needs of their students. Configurations can be developed and then saved for use later in class or to be given as an experiment to students.

It is important for students to understand how to investigate architectures that are optimized in terms of cost, performance, and power. Projects designed around PSATSim have been used in both undergraduate and graduate level courses at Clemson University to teach students these concepts. In these assignments, students are asked to explore architecture configurations that optimize for power, performance, or both within a given chip area budget (at present an equation for chip area based on architecture configurations is utilized). The assignments can be tailored by having students develop architectures that are targeted to a specific class of applications (such as integer, scientific, multimedia, or a combination of these).

6. Future Work

In its present form, PSATSim does not allow the user to directly alter the power model settings. The inclusion of an interface with which the user could select from a set of technology processes would increase the simulator's flexibility and usefulness. This would also allow for the incorporation of more modern technology processes than the base 350 nm process. Also, the incorporation of leakage characteristics would improve PSATSim's accuracy for more modern processes.

At the moment, the user input cache hit-rates do not impact the power consumption due to the use of a fixed cache organization power model that does not reflect the chosen hit-rates. Similarly, the efficacy of the branch predictor has no impact on the power consumption. Functional modeling of the branch predictor components would provide more accurate power and performance modeling. The incorporation of a model to reconstruct the run-time object code, such as that presented by Bhargava, et. al, would improve the accuracy of energy consumption statistics for misspeculated execution paths [1]. The inclusion of memory addresses into the trace files would facilitate the incorporation of functional cache modeling as well. This would make it possible to directly associate the power consumed by the caches and branch predictors and would significantly improve the accuracy of the power model across the range of input para-

meters.

The implemented power model has not been normalized against published figures, so it is not possible to compare the power values from PSATSim against those of other simulators. Normalization would improve the accuracy of the power model and extend the usefulness of PSATSim as part of a broader power modeling project. Additionally, the ability to easily input the energy values for each instruction type by the user would make it possible for students to incorporate values generated from SPICE models and the like into PSATSim and quickly see the results of low-level changes at a high-level.

7. Conclusion

PSATSim provides a unique tool for use in computer architecture courses. It allows students to try out different architectural parameters and interactively watch execution or quickly jump to the end of the execution and view the resultant power and performance figures. It can be used both to demonstrate the operation of speculative superscalar architectures and to teach students about the trade off between performance and dynamic power consumption.

References

[1] R. Bhargava, L. K. John, and F. Matus. Accurately modeling speculative instruction fetching in trace-driven simulation. In *International Performance, Computer, and Communications Conference*, pages 65–71, 1999.

[2] G. J. Briggs, E. J. Tan, N. A. Nelson, and D. H. Albonesi. QUILT: A gui-based integrated circuit floorplanning environment for computer architecture research and education. In *WCAE*, 2005.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, 2000.

[4] H. Gruenbacher and M. Khosravipour. WinDLX and MIPSim Pipeline Simulators for Teaching Computer Architecture. In *IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS'96)*, page 412, 1996.

[5] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, Third Edition*. Morgan Kaufmann Publishers, San Francisco, 2003.

[6] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1999.

[7] A. Misev and M. Gusev. *Visual simulator for ILP dynamic OOO processor*. St. Cyril & Methodius U., 2004.

[8] R. Ontko and D. Stone. mic1 IJVM Assembler. Developed for A. S. Tanenbaum's *Structured Computer Organization*, available at `http://www.ontko.com/mic1/`, April 2002.

[9] J. P. Shen and M. H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill, 2005.

[10] C. T. Weaver, E. Larson, and T. Austin. *Effective Support of Simulation in Computer Architecture Instruction*. International Symposium on Computer Architecture, 2002.

[11] S. Wilton and N. Jouppi. CACTI: An enhanced cache access and cycle time model, 1996.

[12] M. Wolff and L. Wills. *SATSim: A Superscalar Architecture Trace Simulator: Using Interactive Animation*. Georgia Institute of Technology, 2000.

[13] K. C. Yeager. The MIPS R10000 Superscalar Microprocessor. *IEEE Micro*, 16(2):28–40, April 1996.