Processor Simulator Elucidator (PSE)

Preliminary Documentation

20 February 2008, 21:07:59 CST

1. Introduction

Processor Simulation Elucidator (PSE, pronounced see, which is its old name) is a viewer for *dataset* files produced by CPU simulators. Currently (20 Feb 2008) it only works with the extensively modified version of RSIM 1.0 used in my research.

This PSE documentation is for those familiar with the workings of dynamically scheduled superscalar processors.

1.1. Running a Simulation

The first step in using PSE is for someone, not necessarily the PSE user, to run a simulation with a compatible CPU simulator. Such a simulator will write a *dataset* file which contains detailed information on how instructions execute, as well as other information, for example, the reorder buffer size, the time the simulation was run, etc.

1.2. Loading a Dataset

PSE itself is run after a simulation is complete (or during, for the impatient) to view the dataset file. A dataset file is loaded by specifying it as a command-line argument or by using the Open option of the File menu.

When PSE loads a dataset file it also tries to find the executable of the benchmark (program) that the simulated computer ran. If it's successful then it can display assembly-language instructions, if it can't find the file or if it suspects it has been re-compiled it will pop up a warning.

1.3. Viewing Simulation Results

Simulation data is shown by three different plots and by a table of data. When a dataset is first loaded an *overview plot* is displayed; this shows the value of important quantities, such as IPC, over the execution of the program, see Section 2 [next page] for details. Clicking on a point in the overview window will switch to a *PED (pipeline execution diagram) plot* which shows cycle-level detail of instructions' execution along side assembly code for those instructions; see Section 3 [p. 3]. From the PED plot pressing the PED button switches to a *ROB (reorder buffer) plot* which shows essentially the same information as the PED plot but rearranged to emphasize overall execution efficiency; see Section 4 [p. 6]. (Pressing PED again returns to the PED plot.) From any plot window pressing the tool (a wrench) brings up the dataset variables window, where many details about the simulation can be found, see Section 5 [p. 6].

1.4. Completeness and Polish

Features are added to PSE as they are needed based upon the preferences of the users and available time to implement them. Natural features are often left unimplemented (such as the ability to load multiple datasets) when they are easy to do without. Suggestions and volunteers are welcome!

2. Segments and The Overview Window

The overview window, displayed after a dataset is loaded, provides an overview of program execution, showing important quantities (such as IPC) throughout execution. The quantities are shown for periods of time called *segments*, the quantities themselves come from *overview series*.

2.1. Segments (of execution time)

It is possible to collect detailed instruction information for the entire run of a program (running on the simulated processor), however for many programs the resulting dataset file would be much too large. What PSE currently does is collect detailed information during short time spans, called *segments*. Between the segments no information is collected. For example, making the segment duration 1000 cycles and the segment period (the gap between segment starts) 10000 cycles will reduce the dataset file by nearly a factor of 10 (over collecting information every cycle). (The size of a segment and the gap is determined when running the simulation, not when running PSE.)

2.2. Overview Window Features

When PSE is started it displays the IPC (execution rate in instructions per cycle) and possibly other information in an *overview window*. The IPC is shown as black dots, the horizontal position (x-axis) indicates time (or more precisely, the segment number to which it belongs) and its vertical position indicates the IPC during that segment. There is currently no scale, but the exact value of IPC (or other quantities) can be found by placing the mouse pointer over the corresponding dot and looking at the key.

Clicking on a dot will switch to a *segment plot*, which can be either a *PED (pipeline execution diagram) plot* or a *ROB (reorder buffer) plot*. Those plots are described in Section 3 [next page] and Section 4 [p. 6]. Pressing the tool button will show *dataset variables*, which provide information about the simulation, see Section 5 [p. 6]. A search box is also present, see Section 6 [p. 7].

The *status line* (at the bottom) shows the time (in cycles) of the beginning and end of the segment, labeled *time*. *Insn* provides the count of how many instructions committed (were completely executed so far) at the beginning and end of a segment. *Tag* shows the tag numbers assigned by the simulator which is usually the number of instructions decoded. When few instructions are squashed the tag number will be only slightly larger than insn, if lots of instructions are squashed (perhaps due to poor branch prediction) that tag number will be much higher. The tag number should never be lower than the insn number.

The *Top Insn* label shows the most-frequently executed instruction in the segment, showing its address and possibly the procedure it is in and a source file and line number. The *Top Call* line shows the most frequently executed `call` instruction.

The overview window also displays a segment's *rank*; the segment with rank 1 has the lowest IPC, the highest-numbered rank is the fastest.

The Sort button toggles the sorting of the segments. Initially they are in chronological (execution time) order. Pressing Sort switches that to rank order. Currently sort order can only be chronological or by IPC, but maybe one day it will be possible to sort by the value of any overview series (say branch prediction accuracy).

The overview window has a line reading "Data Size. . .." This is for debugging purposes, ignore it. (It's the amount by which the segment information has been compressed.)

2.3. Overview Series

An overview series is a list of data values, one per segment, measuring some quantity of interest,

such as IPC. They are shown on the overview plot as colored dots. A key showing a description and values can be shown or removed using Overview Plot Values Pane entry in the View menu; this is available in both the overview and segment windows.

The key shows the current value and may also show a count of the number of events that value is based on. An event for IPC is an instruction, an event for the branch prediction ratio is the execution of a branch, an event for the L2 hit rate is an L1 miss. For some series the event is clearly labeled.

When the check box next to a key item (such as IPC) is checked the dots will be connected, this might be used to emphasize values of importance. Clicking the ex removes the dot.

To see a list of all overview series click the plus on the left side of the key or select Overview Plot Series Choose in the View menu; this should pop up a window. Clicking D check boxes will display the corresponding series, clicking C will connect the dots. The Scale Min and Scale Max blocks show how the corresponding series are currently scaled, those can be changed.

Changes take effect immediately and are saved when you exit PSE.

3. PED Plot

The PED plot shows a pipeline execution diagram of a segment. Each instruction is shown as a horizontal bar, at the default zoom the bars blend together. (Press `w` to get a zoomed-in view in which the bars are separated, then press `l` to restore the previous view.) Colors indicate instruction state, the state name appears just above the plot on the left-hand side. (States are described in Section 10 [p. 8].)

The upper right side of the window shows a disassembled instruction (if the executable was found) and its location and a pane to the right, called the *assembly pane*, shows assembler code. See the assembler pane section for more information.

The status line, at the bottom, shows the time (in cycles), the tag (an instruction number assigned consecutively to decoded instructions), and the address of the instruction.

A small swarm of visual guides follow the pointer as it moves over instructions. Some of this swarm is used to indicate information such as dependencies, see Section 3.2 [p. 5].

3.1. PED Plot Display and Navigation

3.1.1. Instruction Cursor & the Pointed Instruction

The instruction under the mouse pointer will be called the *pointed instruction*, in the plot area it will have white *markers* on either side (shaped like pointy bullets). There is also an *instruction cursor*, the instruction it is over is called the *cursor instruction*. The cursor instruction is highlighted with pale yellow markers.

Initially the instruction cursor follows the pointer so the pointed instruction is also the cursor instruction. If the left mouse button is clicked over an instruction the instruction cursor will be locked there (a button labelled Lock will be depressed above the assembly pane). Clicking the same instruction again will unlock the cursor, clicking a different instruction will move the cursor.

Pressing the `shift` key will temporarily move the instruction cursor to the pointed instruction (when the key is released the previous cursor location will be restored).

In addition to clicking, the instruction cursor can be moved by pressing `n` or `p`, using the mouse wheel, or using the up and down buttons above the assembly area. Holding a button or key will cause the cursor to move at the rate of 10 instructions per second.

In the assembly area the cursor instruction (if visible) has a yellow background and the pointed

instruction, if different than the cursor instruction, has a white background.

3.1.2. ROB Occupancy Indicator

In the plot area the mouse pointer will be followed by a pair of vertical lines (really just a single skinny rectangle); this outlines instructions currently in the reorder buffer; the color indicates the amount of empty space in the ROB from red (full) to green (less than half full).

3.1.3. Assembly Box Positioning

Also following the mouse pointer (at least initially) is a light blue box, the *assembly box*; it surrounds the same instructions that are displayed in the assembly pane. Single clicking an empty point inside this box will lock it in place and as a result the instructions in the assembly pane won't change when the pointer is moved. It can be unlocked by pressing the Unlock button or by clicking any empty point. Beware, it's easy to lock the assembly box without realizing it. (If it's locked Unlock is visible.)

The behavior of the instruction cursor and assembly box can at times be annoying, even to the program's author. Suggestions for improvement are welcome.

3.1.4. Clipping and Wrapping

By default, instructions wrap from bottom to top. That is, the instruction following the one at the bottom of the plot area appears at the top. This wrapping can be turned off by pressing the Clip button, *clipping* the PED plot. When clipping is turned on PSE will choose which diagonal band of instructions to display, and will sometimes make a bad choice. Clipping is turned on automatically by certain zoom operations.

The plot can be scrolled using the left and right arrow keys and the home and end buttons. By default when the arrow keys are pressed the scrolling will be animated, that is, it will appear to accelerate and decelerate. The window can also be dragged by moving the pointer while pressing button 1 (the left button).

3.1.5. Zooming

Different ways of zooming are discussed below, most are accessible from the Zoom menu. When zooming with clip mode off (the Clip button is not pressed) instructions may start to overlap each other in the PED plot, making it difficult to read. If that happens turn clip mode on (which is tricky because it may remove the instructions you were looking at) or zoom by double-clicking or dragging (see below).

Any single zoom operation can be reversed by pressing `l`. Most zoom operations are listed in the View menu, the accelerators (keyboard equivalents, shortcuts) are described below.

Pressing `z` and `u` will zoom in and out, respectively. Holding down the right mouse button and rotating the mouse wheel will also change the zoom. Pressing `0` zooms out as far as possible (an instruction is one pixel high and a cycle is one pixel wide). (Or is it two pixels?) Pressing `1` (digit one) zooms out so that an instruction is one cycle high and a cycle is n cycles wide for an n-way superscalar processor.

Pressing `w` or double-clicking will zoom so that the PED plot shows exactly (almost) the instructions shown in the assembly pane. This turns on clip mode automatically.l

The mouse can be used to specify an area to zoom into by holding shift and the left mouse button and dragging out a rectangle. This turns on clip mode automatically.

Each register referenced (source or destination) by the cursor instruction will be shown in a different color, for loads and stores the address is shown in blue. The source registers of direct dependent following instructions will be shown bold and in a color matching a destination of the

cursor instruction. If a register color matches a cursor instruction destination but is not bold then the dependence is indirect, that is, the register value was computed by a chain of instructions starting with the cursor instruction.

Similarly, the destination of a preceding instruction that matches a source of the cursor instruction will be show bold and in a matching color. Indirect dependencies are shown non-bold.

If a dependence is through memory (a store followed by a load) then the color is blue and the register will be show in a slanted font.

The assembler for some instructions omit certain registers, for example the integer condition code register is not shown in `subcc` or in `bgt`. Such dependencies are still tracked.

PSE treats the destination of a conditional move instruction (such as, `movg`) as both a source and a destination.

Dependence information is also shown in the plot area by markers of the same color as the registers.

Dependence highlighting can be partially or completely turned off using the context menu or the highlight menu.

3.1.6. Same Static Instruction Highlighting

The address of the cursor instruction is shown in green bold, the address of other dynamic instances of the same static instruction (same address) will be shown in green but not bold. In the plot area these instructions will have green markers.

This feature is useful for finding the extent of loop bodies.

3.1.7. Dead Instruction Highlighting

An instruction is dead if it writes a register or memory location that's not used before being overwritten. Such instructions are shown completely crossed out in the assembly area and exed out or dark gray (depending on zoom level) in the plot area. (Deadness is based only on activity in the current segment. A dead instruction won't be highlighted if its register or memory location is overwritten outside the segment.)

An instruction is indirectly dead if it writes a register that is used only by dead or indirectly dead instructions. In the assembly area indirectly dead instructions have their addresses crossed out, in the plot area they are exed out (with fewer exes than dead instructions).

This feature can be disabled using the context menu or the Highlight menu.

3.2. Annotations

Annotations are pieces of information associated with either an instruction, a group of instructions, a snip (if you don't know what that is just ignore it), or a cycle. They provide information about the instruction (or group, etc.); the information can be displayed in three places: near the instruction in the plot area, near the instruction in the assembly pane, or in the annotation message pane (located just above the plot area).

In some runs annotations indicate branch and predictor history for each branch, and cache outcome history for each load. Annotations are easy to add to the simulation so a dataset might be enriched (or cluttered) with dozens of them.

To see a list of available annotations select Annotation Preferences from the View menu. For a description of a particular annotation hover the mouse over the ? and wait for the tooltip. The checkboxes to the left control annotation display. The rightmost checkboxes control an individual annotation, the ones further left control groups. The check box labeled Plot-Area Instruction Annotations will turn off all annotations in the plot area, but not those in the assembly or message

area.

The letters under Area show where the annotation appears. The fun way to find an annotation is to hover the mouse over the annotation name. If the annotation is present it will flash, be it in the plot area, message area, or assembly pane. The less fun way to find annotations is to look at the letters. If the letter is lower case then the annotation does not affect the respective area. For example, "p A m" means the annotation only appears in the assembly pane. If the letter is gray-ed out it means the annotation is not currently visible (perhaps because no visible instruction carries the annotation). A background of white means the pointer is over an instruction carrying the respective annotation, a background of yellow means the instruction cursor is at an annotated instruction.

4. Reorder Buffer (ROB) Window

Pressing the PED button in the PED plot will transform the plot into a *reorder buffer (ROB) plot*. The ROB plot shows the same information as the PED plot, but reorganized to emphasize the number of in-flight instructions and fetch performance.

The program itself appears in a assembly pane on the right-hand side (if the executable was found). The instruction under the pointer is shown in bold, its address is shown in green as is the addresses of other copies of the same static instruction (making it easy to identify loops). The assembly pane can be toggled on and off by pressing the DIS button.

By default the vertical scale of the reorder buffer window is based on the size of the reorder buffer. With this scaling when the ROB is half full instructions will reach halfway to the top of the window. If the DP button is pressed the ROB plot is scaled so instructions always reach the top, perhaps making it easy to see detail but also making it difficult to determine how full the ROB is.

Pressing the Overview button returns to the overview window. Pressing the PED button swtiches to a PED plot.

5. Dataset Variables

Pressing the tool or selecting Data from the View menu will pop up the *Dataset Variables* window. This window shows all kinds of information about the simulation run. Each piece of information is associated with a dataset variable, the dataset variable `Report` shows the human readable text output of the simulator.

Dataset variables are arranged in tree form; there is no easy way to search them from the GUI. The value of a dataset variable can also be determined from the command line using the -v option followed by the variable name. For example,

```
[nested.ece.lsu.edu] % pse demo-4-gcc43-4way.ds -v time_ue
 time_ue = 18 Feb 2008 10:14:05 CST
```

All values can be retrieved by using - for the variable name, but expect a lot of output so this option is best used with grep or by piping the output to a file:

```
pse demo-4-gcc43-4way.ds -v - > allvars.txt
```

In the dataset variables window variables describing specifications of the simulated system are under `Configuration`, for example, `bp\_method`, shows the branch predictor used and `bp\_ghr\_length` shows the length of the global history register (GHR).

Information about where the simulation was run is shown under `Environment`. For example, `time\_ue`, shows the calendar time that the simulation was run.

Variables describing how the simulation was run are under `Simulation`.

Variables describing the results of simulation are shown under `Data`.

6. Search

The overview and segment plots display a search block. It is under the main toolbar, a tooltip starting "Search Target" will appear when the mouse pointer is hovered over the text field. The feature can be used to search for an instruction by address, and instruction by tag, or to move to a particular cycle.

To search for an instruction put its address in hexadecimal prefixed by `0x`, for example, `0x12344`. If shown in red, the address is invalid, if shown in gray no instruction is found, if shown in blue matches were found. Pressing enter will jump to the first match.

The instruction is highlighted in the PED plot's plot area with a blue marker and the instruction's address is shown in blue in the assembly pane. In the segment window buttons on either side of the search block will move the instruction cursor to the first, previous, next, or last match. The button on the far left erases the search field. The tab key can also be used.

To search by tag enter the tag (in decimal). To move to a particular time (cycle) enter the cycle number preceded with a `c`.

7. Preferences File

When PSE starts it looks for a file named `.pse\_prefs` in the current directory and if found loads preferences from that file. When PSE exits it will write that file, updating values changed. Preferences include the list of overview series to display, whether clip mode is on, etc.

Some preferences can be set using the GUI, others just have to be set by hand-editing the preferences file before PSE is started.

If, when exiting PSE it finds that the preferences file has changed since it started it will ask whether you want to overwrite the preferences. There is no good way to merge preferences set during overlapping PSE sessions.

8. Assembly Pane

The assembly pane shows assembly code for instructions in the plot area, those instructions will be outlined with a blue or black box. It can be hidden or shown by selecting Disasembly Pane from the View menu. Its width can be changed by dragging the grippy. The assembly pane is only shown if PSE was able to find the benchmark used to generate the dataset.

In addition to the assembly pane, a location and assembly code for the cursor or pointed instruction will also be shown below the search toolbar, it will look something like:

```
.LLM1160  render_light+2259  stl_deque.h:195
add  %l7, %g1, %l6
```

The `add`, of course, is the instruction; the text above it is the location of an instruction; the same find of location information is used in the assembly window.

The location includes several items: an assembly line number (`.LLM1160` above), which is based on certain assumptions which don't always hold. A procedure name and offset (`render\_light+2259`, meaning the 2259th instruction in the render light procedure), and a source file and line number (`stl\_deque.h:195`).

The source file and line number are only available if the benchmark was compiled with the debug option. If the pointer is out of the plot window or the executable cannot be found the first

instruction in the segment is shown.

9. Creating Image Files

The printer button pops up a dialog for writing an image file version of the plot window, the image file can be in encapsulated PostScript (eps), pdf, or png format. To select a format type a file name with the appropriate extension.

10. Instruction States

The colors shown in PED and reorder buffer plots indicate what are called *instruction states*, but might more accurately be called instruction *events* or instruction *information* since the states don't necessarily correspond to any processor-maintained state. The state descriptions below are for a personal version of RSIM 1 at the time of this writing. Most states are defined (name and color) in the dataset file, so the description below documents the simulator, not PSE itself.

Roughly, the states show what part of the processor an instruction is in, what it is waiting for, what it is doing, or what has happened to it. Limitation: states related to load and store instructions that are in flight at the beginning of a segment are not fully shown.

The color used for a state can shown tinted, as though colored plastic were placed over a part of the screen. The most common use of tinting is to show instructions that will be squashed, they are tinted red and the state name has "(Doomed)" appended to it. In pre-execution simulations pre-executed instructions are shown tinted blue.

10.1. Typical Non-Memory Instruction States

An instruction with no dependencies and which is at the head of the reorder buffer when it completes will be in flight for the minimum amount of time; the following states will be displayed for such an instruction: **Decode**→ **Pre-Ready**→ **Execute**→ **Commit**. Many instructions will have to wait for operands before executing and wait again to commit: **Decode**→ **Pre-Ready**→ **Dep Wait**→ **Execute**→ **Complete**→ **Commit**. Some instructions are unfortunate enough to follow a mispredicted branch or an instruction that raises an exception: **Decode**→ **Pre-Ready**→ **Dep Wait**→ **Execute**→ **Complete**→ **Squash**. In this case, all of the states before squash are marked "Doomed."

10.2. List of Non-Memory Instruction States

Predict

The instruction address has been predicted. In processors that predict blocks this can happen far in advance when a block has many instructions. (A *basic block* is a group of instructions with a single entry [can't jump into the middle] and a single exit [can't jump out of the middle]. The block predictor predicts basic blocks, which in part, might mean predicting a branch at the end of the block.)

Fetch

The address is being fetched from the instruction cache.

Decode

The first cycle at which the instruction is decoded. Currently, this is the first cycle at which an instruction is displayed. (Yes, fetch is on the to-do list.)

Pre-Ready

This state covers two things. Early pre-ready states are for decoding and register renaming. Later ones are for selection by the scheduler, reading of registers, and dispatch to a functional unit. The later pre-ready states are always shown, even if an instruction has to wait for operands.

Dep Wait
(Dependency Wait) Instruction is waiting for an operand.

FU Wait
(Functional Unit Wait) Instruction is ready to execute but must wait for a functional unit to become available.

Execute
Instruction is executing. For memory instructions execution means computing an effective address, there are separate states for bypass or cache usage.

Complete
Instruction has executed and it waiting to commit.

Commit
The instruction is being committed.

10.3. Typical States for Loads and Stores

Loads and stores use the states listed above plus additional states listed below. Both loads and stores must compute an address. Loads sometimes get their data in the load/store queue but other times must go to the cache. Stores always go to the cache, but they do so starting just before they commit and the cache access can finish after they commit. (Meaning the cache must be able to complete the store, there can be no chance of an exception.)

Typical states displayed for an instruction are listed below, followed by a description of just what the states mean.

A typical set of states for a load finding data in the load/store queue: **Decode→ Pre-Ready→ Dep Wait→ Execute→ Load Wait→ Execute→ Complete→ Commit**. The first execute is for the address, the second it for checking the load/store queue.

A typical set of states for a load finding data in the L1 cache: **Decode→ Pre-Ready→ Dep Wait→ Execute→ L1 Start→ Complete→ Commit**. (Unlike the previous case, there is no preceding store with an unresolved address.)

A typical set of states for a load finding data in the L2 cache: **Decode→ Pre-Ready→ Dep Wait→ Execute→ Load Wait→ L1 Start→ L1 Miss→ Complete→ Commit**. If some other load accessed the same line the states would be: **Decode→ Pre-Ready→ Dep Wait→ Execute→ Load Wait→ L1 Start→ L1 Coalesce→ Complete→ Commit**.

A typical set of states for a load finding data in memory: **Decode→ Pre-Ready→ Dep Wait→ Execute→ Load Wait→ L1 Start→ L1 Miss→ L2 Miss→ Memory Access→ Memory Complete→ Commit**.

A typical set of states for a store: **Decode→ Pre-Ready→ Store Wait Address & Data→ Execute→ Store Wait Data→ Store Ready→ L1 Start→ Commit→ Cache Done**.

10.4. List of Load and Store Instruction States

Store Wait Address and Data
A store is waiting for operands needed to compute its address and the data to store.

Wait Address
A memory instruction is waiting for operands needed for its address.

Store Wait Data
Store is waiting for its data.

Wait LSU
A memory instruction is waiting for space in the load/store unit.

Load Wait
A load is waiting for a preceding store.

Store Ready
A store is ready to execute. (Stores execute when all preceding instructions have committed, cache operations continue even after the store itself committed.)

Cache L1 Start
The load store unit has sent a load or store request to the cache. (There is no L2 Start state.)

Cache L1 Wait
The queue leading to the L1 cache was found full.

L1 Coalesce
A load or store has accessed a line that is not in the L1 cache and some other load or store has accessed the same line. (Sort of like finding out the library does not have the May 1950 issue of Time magazine you need and that ordinarily it would take a week to get it but, lucky for you, four days ago someone asked for the June 1950 issue. It will arrive in three days along with all the other 1950 issues of time magazine.)

L1 No Mushers
The limit to the number of misses the cache can simultaneous handle was reached and so the load or store must wait. (The term musher is only used here, nowhere else.)

L1 Miss
The load or store encountered a level 1 cache miss and is waiting for the L2 cache to respond. The duration of this state is the L2 cache latency plus any waiting time before the L2 access started.

L2 Miss
The load or store encountered a level 2 cache miss and is waiting for memory to respond. The duration of this state is the time needed for a request to reach a memory device.

Memory Access
A memory read or write has started. The duration of this state is the memory access latency.

Memory Complete
A memory read or write has completed. A load will complete after the data is sent back to the processor.

Cache Done
The load store unit receives the data from a load. This state is often not visible because it is obscured by some other state.