LSU EE 7700-1 Homework 1 Due: 3 March 2010

To complete this assignment follow the setup instructions from the course Web page. The setup instructions bring you to the point where you can compile the cpu-only examples but for this assignment that won't be necessary. (The setup will facilitate the use of Emacs to view the source and assembler code.)

In this assignment the PSE (pronounced see) visualization program will be used to analyze the execution of a SPARC v8plus program (our lighting demo).

For instructions on using PSE visit the EE 4720 procedures page, `http://www.ece.lsu.edu/ee4720/proc.html`*, and look for the PSE instructions.*

For documentation on the SPARC instructions see `http://www.ece.lsu.edu/ee4720/doc/JPS1-R1.0.4-Common-pub.pdf`*.*

Datasets, source code, and assembly code for the timing analysis versions of the demo-4-lighting.cc have been placed in `/home/faculty/koppel/pub/gpm/2010/hw01` on the ECE Linux/Sun filesystem.

To view a dataset you could issue a command like.
`pse /home/classes/ee7700/com/2010/hw01/demo-4-lighting-normal.ds &`.

To save typing you might create a symbolic link to the dataset directory and use that:

```
 cd ~                                             # Or place link elsewhere.
 ln -s /home/faculty/koppel/pub/gpm/2010/hw01   # Do this just once.
 pse hw01/demo-4-lighting-normal.ds &            # Run PSE in background.
 emacs hw01/demo-4-lighting.cc  &                # View source using Emacs.
```

The dataset `demo-4-lighting-normal.ds` shows execution on a four-way superscalar CPU with a 64-entry reorder buffer, two floating-point units, and predict one block (branch) per cycle and fetch two non-contiguous regions per cycle. The processor roughly matches the Fujitsu SPARC 64 VI.

The source code and assembler are also in the hw01 directory.

Problem 1: *Note: This problem is here to check out the software setup and for familiarization. It won't be graded.* The code demonstrating 3D rendering, demo-4-lighting.cc, is in directory /home/faculty/koppel/pub/gpm/2010/hw01(on the ECE Linux/Sun systems), along with the code in SPARC assembly language (demo-4-lighting.s) and a record of its execution on a simulated computer, in file demo-4-lighting-normal.ds.

Load the source and assembler code into the editor of your choice, and display the dataset using PSE. If there are any difficulties with this step ask questions.

In PSE, view a segment PED graph (the one showing instruction execution and assembly code). You should be able to see assembler on the right-hand side.

(*a*) Find a source code line reference in the assembler file, such as demo-4-lighting.cc:712. Then find the corresponding line in the source code file (just jump to the appropriate line number) and in the assembler file (search for "! 712" for example). Use your favorite editor, Emacs with the class setup is recommended.

Problem 2: The dataset file records execution over several frames. Using PSE find the fraction of time (for a frame) spent in each section.

Prepare a table with a row for each section (make up a name, perhaps using comments in the source file). Each row should have:

- A section name.

- The number of cycles in the section.
- The percentage of time for that frame needed for the section.
- The execution rate during the section (in insn per cycle).
- A quick judgment on why the execution was below the peak (or "at peak" if execution was 4 IPC). Possible reasons are: branch misprediction, cache misses, low local ILP.

Problem 3: Locate the code that transforms normals and coordinates to eye space, and determine the following for each loop (coordinate loop and normal loop):

(*a*) Find the average number of cycles per vertex and the total number of instructions per vertex. (Per vertex means per coordinate for the coordinate loop and per normal for the normal loop.)

(*b*) Provide the number of instructions in each category below:

- Graphical Computation: Floating-Point (used to operate on coordinate or normal).
- Graphical Data Movement: Load or Store of vertex-related data.
- Overhead: Instructions needed to compute addresses, determine whether at end of list of vertices, and other non-graphical tasks.

Problem 4: Continue considering the coordinate and normal transformation code.

(*a*) Find an example of where this code (either the coordinate or normal transformation) can be improved. Show the existing code and show your improved version. *Hint: There is at least one example that does not require in-depth knowledge of how the transformation or vertex list code works.*

```
! Coordinate transform code.

O  sll     %i1,2,%l6      X Eliminate, these.
O  sub     %l6,%i1,%o3    X Instead just add size of vertex
O  sll     %o3,2,%i0      X to vertex pointer and compare to
O  sub     %i0,%i1,%o1    X pre-computed end address.
O  sll     %o1,2,%g3      X
O  addcc   %l7,%g3,%l6
O  be,pn   %icc,.L77003469
O  add     %i1,1,%l3
O  prefetch        [%l6+512],2
O  prefetch        [%l6+524],2  X Eliminate (one sufficient)
O  prefetch        [%l6+512],2  X Eliminate
O  prefetch        [%l6+520],2  X Eliminate
D  ld      [%fp-2164],%f31
O  st      %l3,[%fp-332]        X Eliminate, sufficent regs.
D  ld      [%l6+4],%f30
D  ld      [%g3+%l7],%f5
D  ld      [%l6+8],%f6
D  ld      [%l6+12],%f7
G  fmuls   %f30,%f23,%f1
G  fmuls   %f30,%f19,%f2
G  fmuls   %f30,%f20,%f4
```

```
G  fmuls   %f30,%f21,%f0
G  fmadds  %f5,%f31,%f1,%f24
G  fmadds  %f10,%f5,%f2,%f25
G  fmadds  %f13,%f5,%f4,%f3
G  fmadds  %f16,%f5,%f0,%f26
G  fmadds  %f17,%f6,%f24,%f27
G  fmadds  %f9,%f6,%f25,%f29
G  fmadds  %f12,%f6,%f3,%f31
G  fmadds  %f15,%f6,%f26,%f4
G  fmadds  %f18,%f7,%f27,%f28
D  st      %f28,[%g3+%l7]
D  ld      [%l6],%f3
G  fmadds  %f8,%f7,%f29,%f30
D  st      %f30,[%l6+12]           CX Unnecessary reg movement.
D  ld      [%l6+12],%f1            CX
G  fmadds  %f11,%f7,%f31,%f2
G  fmadds  %f14,%f7,%f4,%f0
D  st      %f0,[%l6+4]             X Unnecessary register movement.
D  st      %f2,[%l6+8]             X
D  ld      [%l6+4],%f24            X
D  ld      [%l6+8],%f5             X
G  fdivs   %f22,%f1,%f6
G  fmuls   %f5,%f6,%f26
G  fmuls   %f24,%f6,%f25
D  st      %f25,[%l6+4]
D  st      %f26,[%l6+8]
G  fmuls   %f3,%f6,%f27
D  st      %f27,[%g3+%l7]
D  st      %f22,[%l6+12]
O  ld      [%fp-332],%g3          X Have enough regs, no
O  ld      [%fp-340],%g4          X aliasing.
O  ld      [%fp-348],%l7          X
O  cmp     %g3,%g4
O  bne,a,pt        %icc,.L900001151
O  ld      [%fp-332],%i1          X Have enough regs.
```

(*b*) Estimate the speedup of your code (the average cycles per vertex before and after your change).

Problem 5: Consider just the code that transforms normals to eye space. The execution rate for this code (and other code too) would be improved by a larger ROB.

(*a*) Estimate the minimum size of the ROB needed for full-speed execution by measuring the computation critical path: a sequence of instructions in which successive instructions wait for the previous one. (These stand out and can be verified using PSE's ability to highlight dependencies.) Note that if the critical path is C cycles then during its execution the CPU would fetch $4C$ instructions and the ROB would approximately need space to hold them. Use this fact to estimate the minimum size needed.

(*b*) Explain why the modified code below might improve performance on this system. Optional: Estimate by how much the modified code will improve performance based upon critical path measurements above.

```
// Before
while ( pVertex* const v = vtx_list.iterate() )
  {
    v->normal *= nte;
    v->normal.normalize();
  }

// After
while ( pVertex* const v = vtx_list.iterate() )
  v->normal *= nte;
while ( pVertex* const v = vtx_list.iterate() )
  v->normal.normalize();
```