CSE 471: Computer Design & Organization
Extra Credit Assignment
Due: Thursday, May 31

This assignment gives you the opportunity to contrast and evaluate the sources of performance in chip multiprocessors and multithreaded processors. It should also give you an appreciation for how closely compiler optimization and code scheduling are tied to good computer performance.

All through the assignment I have posed questions for you to think about and whose purpose is to jog and guide your thinking. You don't have to answer each question in your report in a rote fashion, but you should touch on the issues raised.

For this assignment you can work in teams of two.

And remember, this is an extra credit assignment. Don't stress out over it, and don't let it interfere with studying for the final (which is not extra credit, as you undoubtedly know ☺). I hope you have fun with it, and that it stretches you abit.

Background: Tiling

Compiler writers develop machine-dependent compiler optimizations to transform code, so that it executes more efficiently on a particular implementation or a particular class of machines. One such optimization is called *tiling* (or blocking). Tiling partitions an array of data into smaller arrays, called *tiles*, and generates instructions that do the computation that had been specified for the whole array, but one tile at a time.

. Tiling was first developed for use on uniprocessors. In that environment the motivation for using tiling was to reduce the number of misses in the L1 data cache. A brief explanation of tiling and a code transformation example for uniprocessors appears in the 4th edition of the text on pages 303 – 305 and in the 3rd edition, pages 433 – 435.

Part I: Evaluation of Tiling

The purpose of this assignment is to analyze tiling in the context of parallel processors.

First, think about whether tiling is an appropriate optimization for parallel machines, both chip multiprocessors and simultaneous multithreaded processors. Would tiling improve performance in these designs? Why or why not? What aspects of performance would they improve? On the other side of the coin, would tiling have any performance overheads that would need to be overcome for the optimization to be beneficial? Why or why not? What are the sources of these overheads?

It is important that you come to some working hypotheses here before starting on the next part of the assignment.

Part I: Tiling Design

If you decide that tiling will improve performance on a chip multiprocessor, design a tiling algorithm for CMPs, using the matrix multiply program, shown below.

```
// In this code two matrices of size NxN, called A and B, are
multiplied and the results are stored in C.  A scalar variable
temp is used to hold the result, simply to avoid some array
element computation.  It is not needed for algorithm correctness.

for(i = 0; i < N; i++) {
   for (j = 0; j < N; j++) {
      {
      temp = 0;
      for (k = 0; k < N; k++)
         temp = temp + A[i][k] * B[k][j];
      C[i][j] = temp;
      };
```

How does your algorithm for multiprocessors facilitate the advantages you anticipated from tiling? How does it minimize the overheads? Are there any outstanding issues or trade-offs?

Will your current algorithm also work well on an SMT or should a different strategy be employed on each? If you think the latter, design an algorithm for SMT. Now, the same set of questions apply: How does your algorithm for SMTs facilitate the advantages you anticipated from tiling? How does it minimize the overheads? Are there any outstanding issues or trade-offs?

Part III: The report

Turn in the fruits of your labor in the form of a report. You should be able to say everything you want to say in two or three pages, excepting for the algorithms.