UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC171 Parallel Computer Architectures Spring 2009

Homework 2 Due Wednesday 13 May 12:00 pm [turn in to homework box NOT IN CLASS]

The homework is to choose from the following book problems and to do Problem 1 (below) in its entirety. I will distribute solutions to all problems in class on Wednesday 13 May.

Homework problems (from H&P, and its accompanying CD):

- Do 4 of the following 5 problems:
  - 4.1. Cache coherence.
  - 4.7. Synchronization primitives.
  - 4.8. Cache coherence with snooping protocol.
  - 4.16, 4.17. Distributed shared memory.
- Do E.1 (on your CD). Bandwidths.
- Do 1 of the following 2 problems:
  - E.10, E.11. Switching networks. For E.11, pick any 3 of the 6.

1. **You must do this problem.** This is a design problem with a significant design component. It is derived from a paper by Mark Hill and Michael Marty at the University of Wisconsin, "Amdahl's Law in the Multicore Era", which appeared in IEEE Computer in July 2008. You will independently derive and analyze some results from this paper. **You may not read this paper, visit its webpage, look at any talk slides about this paper, or read any commentary on this paper while doing this homework problem.** No exposure to this paper at all! Doing so is considered a violation of the honor code. You will learn a lot from this analysis and I want to make sure you do it independently.

   Hill and Marty wanted to use Amdahl's Law to analyze the possible performance of multicore chips under three design scenarios: symmetric cores (in which all cores on the multicore chip are identical); asymmetric cores (in which all cores on the multicore chip may vary in performance and/or resources); and dynamic techniques (in which several cores can work together to speed up sequential parts of a program).

   Recall Amdahl's Law, which calculates the overall speedup of a program when part of that program is sped up: $\text{speedup}_{\text{overall}} = \frac{1}{(1-f)+f/S}$, where f is the fraction of the program to be sped up and S is the speedup of that fraction f.

   Hill and Marty proposed the following hardware cost model. They assume that "a multicore chip of given size and technology generation can contain at most n *base core equivalents (BCEs)*, where a single BCE implements the baseline core". The limit n might be a limit in size, power, cost, or a combination of those factors. They also assume that an architect can increase the performance of a core by giving it more resources. If a single core of size 1 BCE has performance

1, then a core made of n BCEs has performance $P(n)$. They define $P(n) = \sqrt{n}$. In other words, if you make a core 4 times as large, it has twice the performance of the original core; if you make a core 9 times as large, it has three times the performance. The number of BCEs per core is designated as r.

(a) **Symmetric Multicore Chips**. In a symmetric multicore processor, we assume that f is the fraction of the program that can be parallelized. Sequential code is run on one core (with performance $P(r)$). Parallel code is run on all cores (with performance $P(r) \cdot n/r$). Make sure you understand this paragraph well—and that you could derive it yourself—before proceeding. You will now need to derive the expression for speedup as a function of f, n, and r.

Do the following:

For $f = 0.5, 0.9, 0.975, 0.99,$ and 0.999, and for both $n = 16$ and $n = 256$ (you will draw one plot for each n, so two plots total with five lines each), plot the speedup on a program with a fraction of parallel code f over a single core as a function of r. For example, one point on the x axis of this plot will be $r = 1$, which will create n identical cores. Another will be $r = n$, which will create one single core of size n. You will turn in these plots.

With those results, answer the following questions:

- Is it important for programmers on symmetric multicore chips to maximize f?
- Is it ever useful to use cores that are larger than 1 BCE (should r ever be greater than 1)? Why or why not?
- As we move to larger chips (more BCEs), is it more or less important to build cores that are larger than 1 BCE?

(b) **Asymmetric Multicore Chips**. Another design strategy for a multicore processor is to build one large core (of size r) and many $(n - r)$ small cores of size 1. For sequential code, only the large core is running (with performance $P(r)$); for parallel code, the large core gets performance $P(r)$ and the small cores each get $P(1) = 1$.

Do the following:

For $f = 0.5, 0.9, 0.975, 0.99,$ and 0.999, and for both $n = 16$ and $n = 256$, plot the speedup on a program with a fraction of parallel code f over a single core as a function of r.

With those results, answer the following questions:

- Does this design strategy make sense? Can we get better speedups with one large core and many smaller cores? Is the performance ever worse than the symmetric case?
- As we move to larger chips (more BCEs), are asymmetric cores a better or worse idea (compared to smaller chips), and is better to make a larger single core or a smaller single core?

(c) **Dynamic Multicore Chips**. A third design strategy for a multicore processor is to build uniform size-1 cores (as in symmetric multicore chips), but allow r cores to combine together with performance $P(r)$ on all sequential code.

Do the following:

For $f = 0.5, 0.9, 0.975, 0.99,$ and 0.999, and for both $n = 16$ and $n = 256$, plot the speedup on a program with a fraction of parallel code f over a single core as a function of r.

With those results, answer the following question:

- Does this design strategy make sense compared to the asymmetric case? Is the performance always better, always worse, or sometimes better and sometimes worse?