Homework #4

Due Thursday, 20 September 2007.

Show your work, and be sure to use the more complex version of Ahmdahl's law in each problem, no credit if you don't solve the problem by plugging into this formula:

Overall Speedup = 1 / ((1 – f) + (f / Speedup))

Here f is the fraction of execution time that an optimization applies to and "Speedup" is the speedup that applies to that fraction, as opposed to the overall speedup for the whole system.

1. Suppose multiplies are 30% of the instruction mix for the benchmark you're targeting. Suppose all other instructions besides multiply take a single cycle to complete. By adding a multiplier to your ALU you increase the clock cycle time of all instructions by a factor of 1.1, but you decrease the CPI of multiply instructions from an average of 10 to an average of 7. What is the overall speedup of adding a multiplier (note that if the speedup is less than 1.0 then it's a slow-down and we would probably opt not to add a multiplier)?

2. A network filtering hardware company claims that they get a 3x speedup overall in terms of the number of packets processed per second, and to have done so by implementing an optimization that applies to one type of network packet, which you know to constitute one fourth of typical Internet packets. Is this possible? What is the maximum speedup they could have possibly achieved?

3. You want to parallelize a scientific application you're working on and execute it on a cluster of ten machines (assume one, single-threaded processor each). To do this, you have to use a different algorithm that must execute 4 times as many floating point instructions as the optimized serial version of your code (but uses linear algebra formulas that can be parallelized). Using the number of floating point operations as a performance measure and ignoring communication and synchronization costs, what fraction of your code must be able to run in parellel to at least break even? (Sometimes you have to do precomputations and postcomputations that can't be parallelized, even for a parallel algorithm).