**CSCI 5551 Team Assignment 1**

During the last few weeks we've covered a range of both hardware and software ideas, establishing not only a need for parallel and distributed systems but a framework for creating and judging them. Thus far we've been concentrating on algorithms that can decomposed into binary operations that can be rearranged to exploit some key architectural advantages of parallel / distributed computer systems. Specifically, we have been making frequent usage of the principles of temporal and special locality via SIMD and MIMD architectures. Through the example implementations, we have seen the advantages of a superscalar architecture, pipelining, N.U.M.A., U.M.A. systems and seen what in days’ past was considered a super computer. During our thorough review of the prefix algorithm and the multitude of ways of coding it, we have seen that manual rescheduling can be used to exploit multiple concurrent specialized execution units, preloading work (instructions), and resources (data). These techniques were further reinforced with our reviews of matrix multiplication; through which we have seen that rotating the traditional algorithm allows for an intentional memory access pattern, like: row by row or column by column and the uses of those depending on the implementing-language - row major, column major, or skewed matrix storage in memory. All of this culminated recently into the question of how to judge a good parallel algorithm; and one of the possible frameworks for the major metrics of a ‘good’ algorithm, Amdahl's Law. Amdahl's Law allows for formulation of objective measurements for quality, such as: time to completion, speed up, efficiency, return on investment in more processors, and upper limits for each metric. Finally, Amdahl's Law alludes to a triangle of desirable goals, that for most problems, cannot be maximized for all possible metrics.

This course has generated a range of impressions about the pseudo code, algorithms, algorithm development and metrics for a ‘good’ parallel algorithm. The pseudo code being used so far seemed to be somewhat generic and not concentrate on the legion of required semantics for optimizations specific to a recent generation of processor, such as conditional moves, true / pseudo-hardware threads, CPU hints, and the differences between user space and OS space threading libraries. It also seems that a proper compiler could be smart enough to notice that there is a closer to ‘perfect’ tree evaluation than humans regularly would be able to extract and code at any given time; that being said it does make sense that as programmers we would work to understand the optimal expression and ensure that we phrase our code in a way that is easily mapped / compiled into the nearly ‘perfect tree’ of operations. In addition, one of the reoccurring themes is that it seems like SIMD could be thought of as processing a binary tree of operations a level at a time; whereas MIMD could be seen as either doing a sub-tree at a time (such that you can amortize the cost of fork / join), or a column at a time. In other words, it could be said that SIMD is row parallel and MIMD is column parallel.

During the course a number of questions have come up, including:

* How does transactional memory affect the prefix problem?
* Are there any power / heat considerations with these algorithms?
* Previously, processor architecture transitioned between CISC to RISC; is it possible that processor architecture is now transitioning into a heterogeneous processing style, where the general purpose processor is RISC but the specialized processors are more of a domain specific CISC or using completely disjoint instruction sets?
* How far are modern architectures going into speculative execution?
* Is there a good paper experimenting with (a fairly large work size and utilizing a variety of architectural styles) human made operation rescheduling, hardware made operation rescheduling and compiler made operation rescheduling techniques?
* Is there any architecture distributing general purpose processing power within the 'main memory' cells?