SOME RESEARCH TOPICS FOR STUDENTS

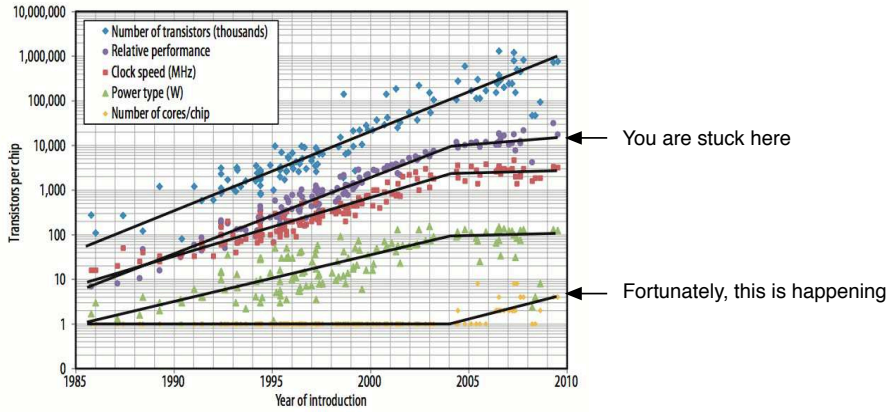
Quentin F. Stout
QStout@umich.edu

Here's a quick overview of some research topics I'm interested in. I've also worked with many students that initiated their own project. If you have something interesting to pursue come and discuss it. I've emphasized PhD thesis topics, but for some of the areas there are undergraduate research projects available. Most of our research involves parallelism, though some is serial. Most, *but not all*, of my students:

- design algorithms for various abstract models of parallelism (with myriad tiny processors or agents)
- and/or design algorithms for real computers (machine learning, multidisciplinary science, etc.)
- and/or model and analyze some aspect of parallelism or cooperative behavior

Former students have gone on to careers at universities, Google, IBM, Intel, Microsoft, think tanks, Silicon Valley startups, national research labs, etc. Two won awards for *Best Thesis* within the entire university.

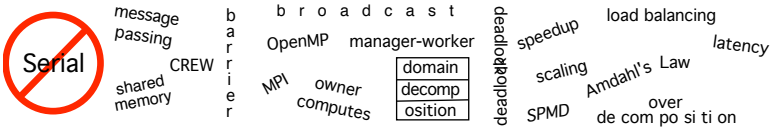
Many computational/learning/social processes are inherently parallel: your brain, internet traffic, evolution, the stock market, etc. Another important reason to study parallelism: *computers aren't getting any faster* (due to pesky limitations imposed by physics), but problems keep getting larger and harder.



For example, Facebook has more than a billion users, and probably $> 10^{11}$ messages have been left on users' walls. Suppose you want to find the strongly connected components of the directed graph with edges given by who wrote a message on whose wall. This takes at least 15 minutes using an optimized implementation of Tarjan's algorithm. Finding strongly connected components is often the first step in much harder graph problems. Perhaps the harder problem takes $100\times$ as long — do you want to wait a day for its solution?

Can the 15 minutes be reduced to 15 seconds, and the day to less than 1/2 hour? Not on a serial computer: Tarjan's serial algorithm is optimal in O-notation, and the high-order coefficients are close to absolutely optimal. 15 seconds should be possible on a parallel computer. However, you need a new algorithm since Tarjan's uses depth-first search, which can't be parallelized.

Laptops have graphics processors (GPUs) with ≈ 100 processors, IBM's Watson computer won Jeopardy by using $\approx$ 3,000 cores, supercomputers have $> 10^6$ cores, etc. However, for many problems there are no efficient parallel programs, so there are many open research problems. Here's a whimsical explanation of parallel computing: http://web.eecs.umich.edu/~qstout/parallel.html



PARALLEL ALGORITHMS (LEARNING, GRAPHS, GEOMETRY, OPTIMIZATION, ETC.): Choose your favorite problem: deep learning, convex hull, selecting an optimal set of friends for a party, etc. You might know (perhaps invented) an optimal serial algorithm, or it isn't optimal but it's the fastest known.

What if the serial algorithm isn't fast enough for the large problems you want to solve? Parallelism can give dramatic improvements, but to create good parallel algorithms you need to know the mathematics and approaches used in serial algorithms, and then figure out additional approaches and properties. For example, space-filling curves are used far more often in parallel algorithms than in serial ones, and many results about hypercube graphs were motivated by work on hypercube computers.

We often look at abstract models based on computing in 2- or 3-dimensional space, motivated by moving electrons on a chip, blood transporting molecules, ant colonies, cellular automata, etc. How can many small entities, each having only a fraction of the data and only communicating with nearby computers, cooperate to efficiently solve the problem? You need algorithms that *think globally, act locally*. Yujie An is working on such a model, showing that adding a small amount of shared memory can give significant speedups.

Faster than quantum, and something we can actually build: A 2-dimensional grid of n very simple processors can sort n items in $\Theta(\sqrt{n})$ time, and a 3-d grid in $\Theta(\sqrt[3]{n})$ time. Similar times hold for finding a minimal spanning tree, intersecting polygons, etc. Lower bound proofs show that these times are impossible on a quantum computer with the same number of qubits.

Minimizing Energy: Power and energy are often the most important constraints on systems ranging from sensors to cellphones to supercomputers. Tim Lewis is looking at sensor networks where the sensors are cooperating, but at any given time almost all of them are in a sleep state to conserve energy. Coordinating a bunch of sleepy processors, where none of them knows all that has happened so far, is rather challenging.

A Simple Example: This illustrates differences between serial, parallel, and energy-minimizing parallel algorithms http://web.eecs.umich.edu/~qstout/maze_algorithms.pdf

Randomized Algorithms for Graphs, Social Graphs, Geometry: Phil MacKenzie won *Best Thesis* for the following work: given $n/\log^* n$ PRAM processors and the adjacency matrix of a random graph of n vertices, his algorithm finds a Hamiltonian cycle, or proves none exists, in $\Theta(\log^* n)$ expected time. $\log^*$ ("log star") is just barely more than constant. Each processor looks at a tiny fraction of the matrix, yet cooperatively they can solve an NP-complete problem faster, in expected time, than they can count. The paper is rather technical: http://web.eecs.umich.edu/~qstout/pap/SPAA93ultra.pdf

Social graphs are quite different than general graphs, and I'm interested in algorithms for random models of both. I'm also interested in algorithms for random points in 2- or 3-dimensional space. For some problems the input might be semi-random, where an adversary is allowed to alter some of the random values. For non-random input, another area of interest is in using randomized algorithms to reduce the energy required.

MACHINE LEARNING AND WORKING IN A GROUP: Several students have worked on projects motivated by term projects they did in my parallel computing class. Amy Netsky is currently working on neural nets for "deep learning", showing that the majority of the connections between neurons can be removed and yet the system learns just as well. She observed this experimentally, and is now working on the mathematics that proves it. Reiko Tanese created a parallel version of the genetic algorithm used for optimization, and surprisingly it showed superlinear speedup. We ended up formalizing this as the "island model" of genetic algorithms, where island populations evolve on their own, but occasionally individuals migrate to other islands. This work has been cited > 1000 times. For both of these projects, initially the changes were made to make them run faster on a parallel computer, and then we noticed that they did better than you would expect.

Julia Lipman analyzed a setting where processors (computers, people) are trying to get tasks done. Each does part of a task and then passes it on, but they get randomly interrupted. She analytically determined the exact rate at which tasks would get completed, modeling it as a directed graph with random delays at the vertices. This

mathematical thesis was motivated by Ted Tabe's timing measurements of communication on our parallel computer. They didn't quite match what we expected, and the cause was random operating system interrupts. Here's Julia's paper: http://web.eecs.umich.edu/~qstout/pap/SICOMPLocalSynch.pdf and Ted's: http://web.eecs.umich.edu/~qstout/pap/IF95ibm.pdf

USING SUPERCOMPUTERS FOR COMPUTATIONAL SCIENCE: This often involves large multidisciplinary teams. Projects have included a NASA National Grand Challenge team for Sun-Earth modeling and a DOE Center of Excellence in Predictive Science for improving computer simulation methodology. NOAA (the national weather forecasting agency) uses our Earth Systems Modeling Framework. All US weather forecasts are based on NOAA's simulations. Here's a movie our space modeling group helped create for the American Museum of Natural History http://csem.engin.umich.edu/gallery/CSEM/SolarStorms_S.mp4. I think they could have picked better music, but we didn't have any input on this aspect.

PARTY GAMES: Parallel algorithms give a variety of party games. For example, use $n/2$ seesaws and Bitonic sort to sort n partiers by weight, taking only $\log_2 n \cdot (1 + \log_2 n)/2$ rounds of seesaws. Or run a parallel algorithm for the stable marriage problem (the host gets to choose whether it is male or female optimal). Actually, parallelizing the stable marriage problem is an interesting research question. If you take my parallel computing course you'll be a data item, and a processor, in a parallel quicksort, and you'll need a coin for the load-balancing/synchronization example. I used to encourage people to wear sneakers for the post-apocaplyptic relabeling algorithm, but the race conditions caused too many collisions.

PROJECTS STUDENTS INITIATE: I'll also listen to ideas about quite different projects. Several students worked on projects that they initiated, where they convinced me it was an interesting problem to pursue. Not all of the projects involve parallelism. For example, I co-advised a team that used machine learning to improve the production rate of Boeing's 787 Dreamliner.

CONTACT: The easiest way to contact me is via email: QStout@umich.edu
If you are within audio range, "Hi" is quite effective. Feel free to drop by my office: 3605 BBB.