1. Bio

I am Alex Reinking, a first-year graduate student advised by Koushik Sen. I did my undergrad in Connecticut at Yale, where I double-majored in Computer Science and in Mathematics, but I grew up in Minnesota. I’m of half Mexican heritage (on my dad’s side) y entonces hablo un poquito español. I work in programming language theory, and have mathematical background in graph theory, category theory, and algorithmics (especially convex optimization). I get a little obsessive about performance tuning, although I’m not the best at it by any stretch. Ideally, I’d like to get much better at this by taking this class. One of my biggest critiques of my own field is about the lack of emphasis on usability from a performance perspective.

I wrote a SAT solver as part of my program analysis course in undergrad. We were challenged to develop a solver that was as fast as possible, using whatever resources we could find in the literature. I wrote the fastest (correct) solver of over forty in the course, implementing watch lists, conflict-driven clause learning, VSIDS (variable selection), randomized restarts, cache efficient heaps and vectors, and much more. For this reason, I chose SAT solvers as my parallel application.

2. Application: Parallel SAT Solvers

The yearly SAT competition pits SAT solvers against one another to measure their performance. SAT is the classic NP-complete problem, but state-of-the-art solvers are able to solve industrial-strength problems from a variety of fields, from flight scheduling to a myriad of program analysis problems. The most recent winner of the Parallel track of the SAT competition is “Treengling” [1].

The straightforward approach to adapting the classic DPLL backtracking algorithm to the parallel setting is to simply distribute different guesses among available processor cores. However, the frequency of communication proves inefficient in practice. Instead, a portfolio-based approach is taken by all modern parallel solvers, which do not communicate at all. Instead, multiple instances of a parameterized solver begin running in parallel, exploring different branches. As CDCL discovers conflict clauses, modern solvers periodically share this knowledge with alternate instances. Because different assumptions and branches are being explored by each solver, little redundant work is done. Thus, this approach scales with the number of processor cores available, as long as the memory scales concomitantly.

The SAT competition compares the performance of each class of solvers. On similar benchmarks, the parallel solvers take slightly less time to determine satisfiability than the sequential solvers do. In the average case, they appear to perform much better[2]. All parallel solvers are run on computers equipped with “Dual Socket Xeon E5-2690 v3 (Haswell) : 12 cores per socket (24 cores/node), 2.6 GHz processors and 64 GB DDR4-2133 (8 x 8GB dual rank x8 DIMMS) main memory. Hyperthreading is Enabled - 48 threads (logical CPUs) per node. Compute nodes lack a local disk, but users have access to a 32 GB /tmp RAM disk to accelerate IO operations.”[3]
REFERENCES
