Project Milestone #2

John Dickerson (jpdicker@cs.cmu.edu)
15-780: Graduate Artificial Intelligence
April 19, 2011

"Scaling Today's SAT Solver" Proposal Recap

I am interested in combining and parallelizing multiple search techniques. For my class project, I am combining portfolio-based DPLL SAT solving with Hyperresolution. Specifically, I am building on top of a system started by Erik Zawadzki and myself; I am incorporating our highly parallel DPLL+Resolution code into the relatively untried portfolio-based parallel solver called ManySAT [7], which in turn is built on the highly successful, vested, and quite fast, MiniSAT v2.2 [5].

Progress Toward Original Project Plan

I provide the following plan, with 75%, 100%, and 125% cutoffs corresponding to how much I will have done by the May poster session given an overestimation, correct estimation, or underestimation of the amount of work this project will require.

75% Plan

1. *Become familiar with the ManySAT system and parallel SAT solving literature.*

   Regarding the ManySAT code, I actually hit a major roadblock until quite recently; specifically, the ManySAT folks removed their source code before and during the SAT-2011 (Theory and Applications of Satisfiability Testing) submission period. Their new code only popped up within the last week. On the plus side, the newest ManySAT code base is extremely similar to the most recent MiniSAT project with which I am intimately familiar. The major additions are some hooks into the actual base solver code (to define a portfolio of four slightly different DPLL-based solvers, as well as to permit basic message passing capabilities), and then a new file that controls some of the communication between each independent DPLL run (presumably on independent cores or CPUs). The ManySAT solver did not touch the packaged SatElite—produced by the authors of MiniSAT—preprocessor, so I am still aware of its capabilities and code base.

   On the parallel SAT literature side, I am confident in my knowledge of state-of-the-art parallel SAT solving. The big players are still SATzilla-type [8] solvers in the portfolio-based realm, and ManySAT in the parallel realm. MiniSAT is still an extremely strong contender in serial SAT solving, so ManySAT's core remains a safe foundation upon which to build a highly parallelized SAT solver.

   Regarding combining different SAT solving techniques *a la* DPLL and Hyperresolution, the field remains relatively unexplored. Most recent literature involving non-DPLL techniques in SAT fall into one of two categories: preprocessing or theory.

   Preprocessing. With preprocessing, Bacchus experiments with using both binary resolution and limited hyperresolution to cull redundant clauses before instituting the standard DPLL search [2]. Other similar examples exist [3]. One key piece of information I took away from this literature search is that people have avoided (hyper)resolution because of its incredibly high computational cost. Although theoretically more powerful, when push comes to shove standard—and highly studied, hacked, and implemented—DPLL tends to beat other methods. This bodes well for our goal of offloading the hard Hyperresolution work to non-critical cores in the hopes of extracting valuable information to guide the fast, relatively uninformed DPLL search.

Theory. There is a wide body of literature studying the computational power of various search and proving techniques. For example, within the last decade it was shown that there exists a non-trivial separation between regular and general resolution techniques [1]. Similar results exist between various types of resolution and standard DPLL (see, for example, [6]). Recent purely theoretical results have shown some of the promise of augmenting DPLL with "simple" improvements that increase the power of standard DPLL exponentially [4]. Again, this bodes well for our approach; however, none of the recent results have been implemented, and all were discussed in a purely serial format.

2. *Hook the ManySAT codebase into my current parallel solver; ensure that it runs with ManySAT's DPLL solver running in single-threaded mode.*

   As described above, I was only able to get my hands on the ManySAT codebase this week. Before this time, I spent most of my time getting the code base written by myself and Erik Zawadzki working (correctly) on the world's largest shared memory supercomputer[1]. Still, since receiving—and learning—the ManySAT code, I have begun merging the pure DPLL+Resolution code into the ManySAT framework. I do not foresee this process taking much longer than another week, at least to obtain a single-threaded, correct combined solver.

3. *Get ManySAT working with multiple DPLL searches in conjunction with my parallel solver.*

   Moving the ManySAT+Resolution code over to a highly parallel system will involve at least two major hurdles. First, the code I have for DPLL+Resolution is written with a different threading structure and set of threading libraries than ManySAT (Intel TBB[2] vs. pthreads). Intel TBB, on which the DPLL+Resolution code is currently built, provides a number of threadsafe STL-like data structures—specifically queues and hash maps—that are currently used by the Hyperresolution nodes. Thankfully, the base threading (launching, managing, joining, et cetera) is quite similar to pthreads. Hopefully, it will not be too much of an issue to get the ManySAT code to compile and correctly run with the TBB libraries.

   The second hurdle will be in getting the ManySAT suite to run on a supercomputer. Currently, ManySAT has only been tested on one multicore CPU, with either four or eight cores[3]. The supercomputer I am testing on is ideal for this situation in that is a shared memory machine—theoretically, in the eyes of the solver, the memory is one giant addressable space and each core can communicate with each other core. Still, problems will arise. Along these lines, setting up a testing suite on the supercomputer will be a bit of a hassle.

4. *Explore the tradeoffs between allocating more ManySAT cores versus more Hyperresolution cores. How does this affect the node count of the final search tree(s) versus the speed of the actual solver? Is there some obvious balance?*

   A response to these questions will arise once the ManySAT code is hooked into our DPLL+Hyperresolution framework, and a reasonable testing suite has been built on our resident supercomputer.

100% Plan

5. *Create and intensively test heuristic methods for determining the level of communication between ManySAT-to-ManySAT cores, ManySAT-to-Hyperresolution, and Hyperresolution-to-Hyperresolution cores.*

   While I have not had the opportunity to test my DPLL+Resolution solver alongside the ManySAT solver, I have been able to perform extremely preliminary tests of the Hyperresolution-to-Hyperresolution form. Barring some initial technical difficulties (hardware outages and code bugs), performance on a purely Hyperresolution-to-Hyperresolution is still fairly poor. I believe—and some rudimentary analytics seem to confirm—that the problem falls into two camps.

[1]`http://www.psc.edu/machines/sgi/uv/blacklight.php`
[2]`http://threadingbuildingblocks.org/`
[3]This is also how the current SAT competitions run their parallel solver tracks.

- Extreme communication overhead (how many resolved clauses do I send, to whom do I send them, how frequently should I initiate contact, how frequently should I accept others' clauses, et cetera?) These are the kind of problems that the Master-Slave paradigm provided by the ManySAT framework will hopefully help to alleviate. Still, significant experimentation will be required to determine a good adaptive strategy to handling the communication overhead problem.
- Timely clause production. I've found that, given a wholly unguided hyperresolution process, many of the clauses produced by my Hyperresolution nodes do not "fit in" to the current DPLL tree search. State-of-the-art DPLL takes temporal and spatial locality into account in the way it learns (and forgets) clauses; as such, when one of my Hyperresolution nodes returns a new clause—however strong it might be—it is often not fully realized by the DPLL search. This is due to the fact that, since the tree search fed the Hyperresolution node's database the resolvant's parents, it has potentially restarted or otherwise moved into a completely different part of the search space. Again, hopefully ManySAT's notion of "clause goodness" will help alleviate this problem.

125% Plan

I have not gotten to any of the goals in the 125% plan yet. Thankfully, none of my preliminary results invalidate these optimistic goals' directions, so I still plan to pursue (and hopefully publish) them.

References

[1] M. Alekhnovich, J. Johannsen, T. Pitassi, and A. Urquhart, *An exponential separation between regular and general resolution*, Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM, 2002, pp. 448–456.

[2] F. Bacchus, *Exploring the computational tradeoff of more reasoning and less searching*, Proceedings of the Fifth International Conference on Theory and Applications of Satisfiability Testing (SATâĂŹ02), Citeseer, 2002, pp. 7–16.

[3] F. Bacchus and J. Winter, *Effective preprocessing with hyper-resolution and equality reduction*, Theory and Applications of Satisfiability Testing, Springer, 2004, pp. 321–322.

[4] P. Beame, R. Impagliazzo, T. Pitassi, and N. Segerlind, *Formula caching in DPLL*, ACM Transactions on Computation Theory (TOCT) **1** (2010), no. 3, 1–33.

[5] N. Eén and N. Sörensson, *An extensible SAT-solver*, Theory and Applications of Satisfiability Testing, Springer, 2004, pp. 333–336.

[6] A. Haken, *The intractability of resolution*, Theoretical Computer Science **39** (1985), 297–308.

[7] Y. Hamadi, S. Jabbour, and L. Sais, *ManySAT: a parallel SAT solver*, Journal on Satisfiability, Boolean Modeling and Computation **6** (2009), 245–262.

[8] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown, *SATzilla: portfolio-based algorithm selection for SAT*, Journal of Artificial Intelligence Research **32** (2008), no. 1, 565–606.