Salt & Alef: Unlocking the Power of Boolean Satisfiability

James R. Ezick, Samuel B. Luckenbill, Donald D. Nguyen, Peter Szilagyi, Richard A. Lethin
{ezick, sbl, nguyen, szilagyi, lethin}@reservoir.com
Reservoir Labs, Inc.

Introduction

Solvers for the Boolean satisfiability problem (SAT) are an enabling technology for a diverse set of applications relevant to the HPEC community. These applications include formal verification, analysis of numerical precision for embedded pipelines, and cognitive reasoning. However, solver performance, measured in terms of speed and maximum problem size, is a limiting factor to the application of SAT to real-world problems. We are developing a constraint language, translation tool, and parallel SAT solver to significantly mitigate the impact of these limitations.

Salt is a universal language and translation tool for constraint systems. With Salt, we can optimize the representation of real-world constraint systems to the characteristics of a target solver, thus realizing the solver's full potential. Alef is a parallel SAT solver that distributes both the storage and computational requirements of a SAT instance across multiple nodes. Our analysis shows that we are likely to achieve a consistent performance improvement over existing solvers in terms of speed and allowable problem size. We expect to demonstrate the performance of the complete pipeline running on a Cray XD1 [1] in the near future. These advances are making it possible for us to reexamine previously intractable verification, numerical precision, and cognitive problems.

The Boolean Satisfiability Problem

Given a propositional logic formula, Boolean satisfiability is the problem of finding an assignment to the variables of the formula such that the formula evaluates to true. Formulas for which such an assignment exists are *satisfiable*. In the general case, finding a satisfying assignment is an NP-complete problem. However, modern solvers use a range of heuristics and optimizations based on logical inference that allow them to solve some real-world instances with more than one million variables [2][3].

In addition to the generality of the approach and the existence of highly tuned solvers, SAT is an attractive formulation for embedded pipeline analysis and cognitive reasoning because of the one-to-one correspondence that can be encoded between individual signals and individual propositional variables. For embedded pipeline analysis, this correspondence allows us to analyze the implementation of computer arithmetic as opposed to a mathematical abstraction. For cognitive problems, individual variables concisely capture atomic facts that can be assumed, learned, or refuted.

This work was produced with US Government support, under DARPA contract W31P4Q-04-C-R257. The US Government has certain rights to this work.

Closing the Programming Gap

Most solvers require that SAT problem instances be specified in conjunctive normal form (CNF). CNF is a representation of a formula as a conjunction of disjunctive clauses of (possibly negated) individual variables. While CNF provides algorithmic advantages to most solvers, it suffers from two substantial drawbacks:

1. It is difficult to generate CNF directly from a problem instance without intermediate translation.
2. It obscures known higher-order relationships between constraints.

While CNF is a normal form, it is not canonical. Previous work has shown that the choice of encoding can make a substantial difference in the performance of the solver [4]. Exacerbating the problem, we have demonstrated that the best known choice of encoding often varies with the choice of solver. These realities point to the existence of a significant gap separating the needs of people generating real-world constraint problem instances from the impressive research that has gone into optimizing SAT solvers.

Salt closes this gap. Salt is a constraint logic language and translator for SAT applications that translates Salt input files into CNF files suitable for processing by a SAT solver. The Salt language provides a simple and intuitive syntax for representing propositional logic, arbitrary precision fixed point arithmetic, and set theoretic constraints, while the Salt translator provides a uniform, optimized way of translating those constraints into CNF. Through a set of command line arguments and translation parameters, Salt can be tuned to produce superior performance on a range of solvers. Figure 1 illustrates the variability one such parameter, *capture threshold*, induces on the minimum, maximum, and mean solution times for a suite of prime factorization SAT instances.

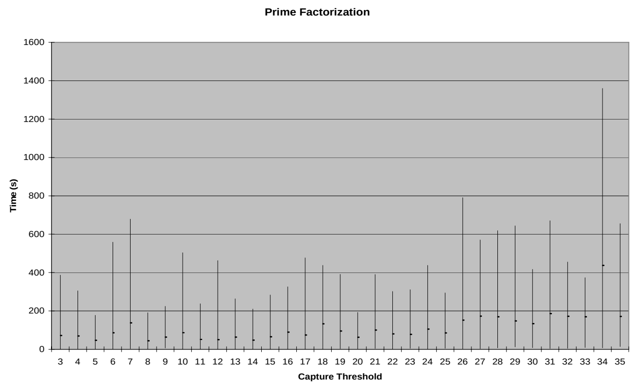

Figure 1: Effect of Capture Threshold on Solution Time

The design of Salt is motivated by the well-understood relationship between applications, programming languages, compilers, and hardware. In that relationship, a programming language provides a common means of expression for a range of applications and the compiler serves as a platform for translating the programming language into hardware instructions. Strong compilers can optimize programming language code and possibly retarget the same source code to a range of hardware. With the Salt language, we have developed an application-independent syntax for expressing logical constraints. With the Salt translator, we have created a platform for analyzing those constraints at a higher level and choosing a representation tuned to the heuristics of the target solver. We have demonstrated that Salt is essential to achieving peak performance from an off-the-shelf SAT solver.

Extending Satisfiability to Multiprocessors

Most modern SAT solvers are based on the Davis-Putnam-Loveland-Logemann (DPLL) algorithm [5]. At each step in the DPLL algorithm, the objective is to extend a partial assignment of the variables to include an additional variable assignment. Each new assignment initiates a sequence of additional assignments dictated by logical implication. That sequence may reveal a conflicting assignment, causing the search algorithm to backtrack. The process of discovering logical implications is known as Boolean constraint propagation (BCP). DPLL-based SAT solvers spend 80% or more of their time performing BCP [2], making BCP an ideal candidate for parallelization.

Additionally, the size of real-world problems can be quite large; it is not atypical for a CNF encoding to require storage on the order of gigabytes. These massive storage requirements limit the applicability of standard SAT solvers to some real-world problem instances.

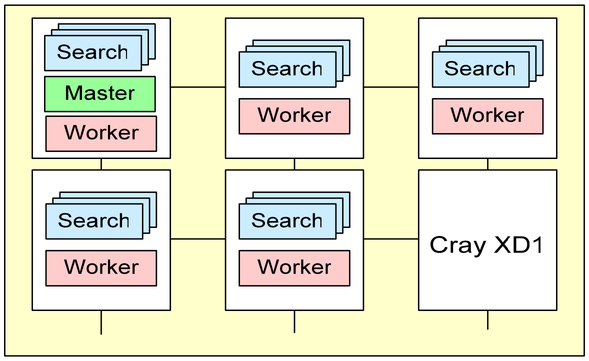

Figure 2: Example Distribution of Threads for Alef Parallel SAT Solver on HPC Hardware

Alef's parallel implementation is broken into three thread types: master, worker, and search. Each thread runs an autonomous message-driven algorithm. The master thread is in charge of distributing work to the search threads, coordinating load balancing, and detecting instances that are not satisfiable. Search threads are in charge of making decisions, coordinating BCP and conflict resolution, and ultimately finding a satisfying variable assignment. Worker threads perform BCP on behalf of the search and master threads, and optionally, perform distributed conflict resolution. Figure 2 illustrates a possible distribution of these threads over the nodes of a Cray XD1.

This organization was partially motivated by the design of an application-specific processor that exploits data parallelism in BCP, which showed speedups between 20x and 60x [6]. While extremely low on-chip latencies allowed the processor's designers to randomly distribute clauses across the chip, the higher message latencies of multi-processor machines require an intelligent partitioning of the clauses. However, partitioning clauses can reduce available data parallelism, as each processor can only handle one clause from a partition at a time. Another parallel solver partitions clauses and distributes them across the nodes of a cluster [7]. While it can attempt larger SAT instances, its performance is lower than sequential solvers due to the combination of high message latency on the cluster and reduced available data parallelism.

To increase available data parallelism, Alef runs multiple search threads, each working on a different area of the search space. Additionally, the message latency of the XD1 is up to two orders of magnitude less than the latency of a cluster with Gigabit Ethernet. Given the low communication latency of the XD1 and Alef's algorithmic improvements, we anticipate a significant performance improvement over sequential solvers in addition to the ability to attempt larger SAT instances.

Conclusion

With Salt and Alef, we are building an advanced tool chain that will enable more extensive application of Boolean satisfiability solving. Our tools make SAT techniques easier to apply and more effective to use on problems in formal verification, analysis of numerical precision for embedded pipelines, and cognitive reasoning. We are currently conducting experiments with a full-featured version of Salt and expect to have experimental results from Alef in the near future.

References

[1] Cray, Inc., "Cray XD1 Datasheet," 2005, Available at: http://www.cray.com/products/xd1/index.html.

[2] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *DAC'01*, Las Vegas, NV, June 2001.

[3] "Boolean Satisfiability Research Group at Princeton," [Online Document], Apr 2006, Available at: http://www.princeton.edu/~chaff/zchaff.html.

[4] N. Een, A. Biere. "Effective Preprocessing in SAT through Variable and Clause Elimination," *SAT'05*, June 2005.

[5] M Davis, G. Logemann, and D. Loveland. "A Machine Program for Theorem Proving," *CACM*, 5(7), 1962.

[6] Y. Zhao, "Accelerating Boolean Satisfiability through Application Specific Processing," *Ph.D. Thesis*, Department of Electrical Engineering, Princeton University, Oct. 2001.

[7] M. Ganai, et al., "Efficient Distributed SAT and SAT-based Distributed Bounded Model Checking," *CHARME'03*, Oct. 2003.