The Scalability of Parallel SAT Solvers

Roberto Asín Achá[1], Juan Olate[2], and Leo Ferres[2]

[1] Departamento de Ingeniería Civil Informática
Facultad de Ingeniería
Universidad Católica de la Santísima Concepción, Chile
rasin@ucsc.cl
[2] Department of Computer Science
Faculty of Engineering
Universidad de Concepción, Chile
{lferres|juanolate}@udec.cl

In this paper, we study issues of scalability of parallel solvers of the satisfiability (SAT) problem on hierarchical-memory multicore (SMP) systems.

Since at least 2009, parallel SAT solvers (henceforth, pSATs) have been performing at the top of the SAT Competition (in 2011, all three wall-clock time winners of the competition are parallel solvers). Also in 2011, pSATs and sequential SAT solvers are grouped into a single competition track[3], which signals the widespread interest in pSATs by the research and industrial communities. This appeal stems in part because of the inherent properties of parallel algorithms, but also because for certain practical applications such as formal verification of pipelined microprocessors, automatic test pattern generation, dependencies between components, among many others, *wall-clock speed* is of the utmost importance: a SAT solver running slower than the production line is of little use.

Meanwhile, instead of increasing clock performance, chip manufacturers are investing heavily on multicore architectures to improve performance and lower power consumption (AMD released the 8-core Opteron 3260 EE in late 2011, and Intel will do the same with the Xeon E5-2650, and its low power version, the Xeon E5-2650L early this year). As Herb Sutter once put it, "the free lunch is over" [1], and it effectively means that software in general will not be getting any faster as years go by by simply relying on faster processors, but by relying on how software scales in over multicore systems.

Finally, modern memory architectures are not flat Processor↔RAM architectures, but a hierarchy of faster-but-smaller to slow-but-large memories with latencies varying from $0.5ns$ access, $32Kb$ memories such as the L1 cache, to tens of nanoseconds, megabyte-large memories like the L3 cache, to gigabyte, $100ns$ access memory such as main memory. Hierarchical memory architectures have a strong impact on the performance of sequential software (e.g., row scanning arrays in row-major representation, memory transfers may be in the order of the input divided by the size of the cache line, while memory transfers for column scanning is in the order of the square of the input). For multicore systems, besides the problem of data representation, there is the problem of false sharing: if two threads write on different words of the same cache line, then the cache line

[3] http://www.satcompetition.org/

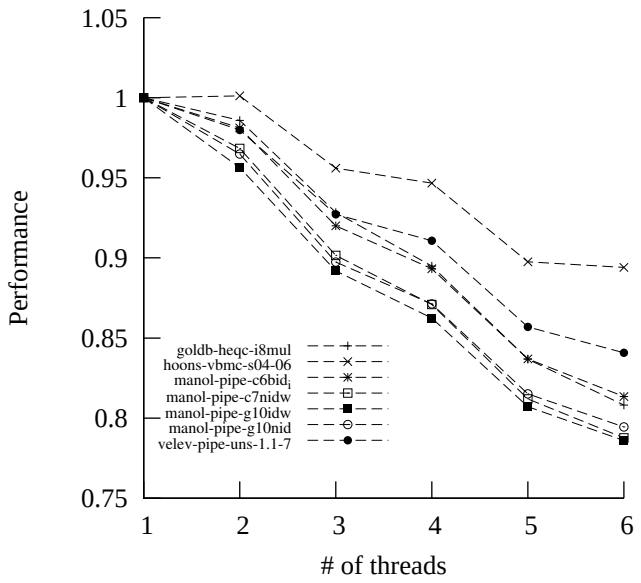

Fig. 1. Performance decay of `plingeling` when run over many cores

in one or the other processor becomes "dirty" and a round trip to RAM ensues, wasting valuable time due to latency.

Thus, given the three arguments above: how *do* pSATs scale in hierarchical-memory multicore architectures? Our case study is the winner of the 2011 SAT Competition, `plingeling`, a portfolio-approach SAT solver [2]. Portfolio-approach solvers such as `plingeling` run multiple SAT solvers in parallel, taking advantage of the fact that different solvers perform better for different problems. The first experiment tested `plingeling` on a 6-core multicore machine, varying the number of threads. In Figure 1 we can see how `plingeling`'s performance tends to decay sharply (up to 30%) when several instances are executed in the same processor, even when they do not, in principle, share any software resources other than the common process address space. We would thus expect that all instances would perform similarly at around 1, plus or minus a small fraction. This is in fact what happens when `plingeling` is run in two different cores of two different chips, even in the same machine.

In order to find the culprit of the performance decay, we have developed a simple portfolio-based SAT solver that allows us to replicate the behavior of `plingeling`, and then also experiment with alternative scenarios in order to take measures towards the improvement of its performance.

1 The AzuDICI SAT Solver

References

1. Sutter, H.: The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobbs Journal **30** (2005) 202–210
2. Biere, A.: Lingeling and friends at the SAT Competition 2011. Technical Report 11-1, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria (2011)