A Research on Minisat Using Coarse-grained Parallel Technique Based on Multi-core and Multi-platform

*Junjie Zhu[1, a], Jingfei Jiang[1, b], Xiaocheng Luo[1, c], Yong Dou[1, d]

[1]National Laboratory of Parallel and Distributed Processing, National University of Defence Technology, Changsha, 410073, China

[a]email: junjiepiglet@163.com, [b]email:jingfeijiang@126.com

[c]email: luoxiaocheng15@nudt.edu.cn, [d]email:yongdou@nudt.edu.cn

Keywords: Minisat solver; coarse-grained; multi-core; multi-platform

Abstract. In this paper, we make a research on a widely-used SAT solver, Minisat, aiming to improve its performance using coarse-grained parallel method on multi-core and multi-platform. Firstly, we parallel the Minisat by mean of OpenMP and test its performance with different threads by running a test set consisting of 2000 SAT problems on an X86 computer. Besides, a scheduling strategy with time sequence is added to the process and achieves a better speed-up ratio. Then, we move the algorithm to an ARM computer and repeat the same process, finding that the performance of Minisat on X86 is better than that on ARM, but ARM platform has a better scale effect than X86 platform when running at full load and is able to perform better than X86 when they have the same hardware configuration.

Introduction

As the first proved NP-complete problem [1], satisfiability problem (SAT problem for short) has an important theoretical and application value in mathematical logic, computer science, IC design and verification, artificial intelligence, etc. It is the core problem of computer theory and application. The design and implement of efficient algorithms to solve this problem is of great significance. However, there doesn't exist such an algorithm whose computational complexity in the worst case can reach polynomial level, thus the solving speed is a big problem restricting the development of the SAT algorithms.

As one of the most popular SAT solvers, Minisat [2] is based on DPLL [3] algorithm, which is a kind of classic SAT complete algorithms. Minisat is added into some techniques like confliction clause learning and watched literal, achieving a good performance. Nowadays, there are many parallel algorithms of SAT, such as GrADSAT [4], NAGSAT [5], Satz [6] and PSATO [7]. There also exist parallel algorithms of Minisat like PMSat [8]. But most of them are faced to clusters and grids consisting of multiple computers, which means these algorithms are not fittable to reach an ideal performance on one computer. Our paper tries to improve Minisat performance on one computer by applying coarse-grained parallel technique and proper dynamic scheduling strategy.

Nowadays, most of the SAT studies are based on X86 platforms, but there is another efficient platform. The ARM architecture is a 32-bit reduced instruction set (RISC) processor architecture with high performance, low cost and low energy consumption. It is widely used in consumer electronics, industrial control, communication system, network system and military industrial project. However, there are few researches about Minisat on this platform. In our paper, we apply the Minisat coarse-grained parallel algorithm to ARM platform, and compare its performance with that on X86 platform, so that we can know if ARM platform is more suitable to solve SAT problems than X86 platform, which is of great significance to a deep study on SAT.

Methodology

Minisat is a kind of complete SAT solver, it can theoretically solve all SAT problems. Comparing with the incomplete solver, its structure is more complex and the solving process is more complicated. Figure 1 shows the main steps of Minisat solver solving a SAT problem. From Figure 1 we can find that steps of Minisat including variable decision, propogation, conflict judgment, conflict analysis, clause learning, are associated tightly. These steps share a clear order. The latter step needs the data produced by the former steps, which means the algorithm exists strong data dependence and not suitable for being dealt with the fine-grained parallel technique. If we seperate the solving process of one SAT case into several jobs and put them on different threads to complete, chances are that communication cost among threads will occupy plenty of computing resources, reducing the solution efficiency of the algorithm. Therefore, we carry on the coarse-grained parallel technique, that is, we start multiple threads at the same time while each thread at a time dealing with one SAT problem, every thread will not accept a new case until it succeed solving the present one. Considering the fact that most SAT cases require memory of several KB to several MB in the process of being solved, which is far less than the available memory of general computers. Therefore, in this paper, we ignored the memory size restrictions on Minisat. At the same time, in order to avoid the overhead costs of sharing thread parameters and to trade space for time, we define independent data structures and allocate independent memory space for each thread when starting multiple threads. So the threads are independent of each other and share no data exchange.

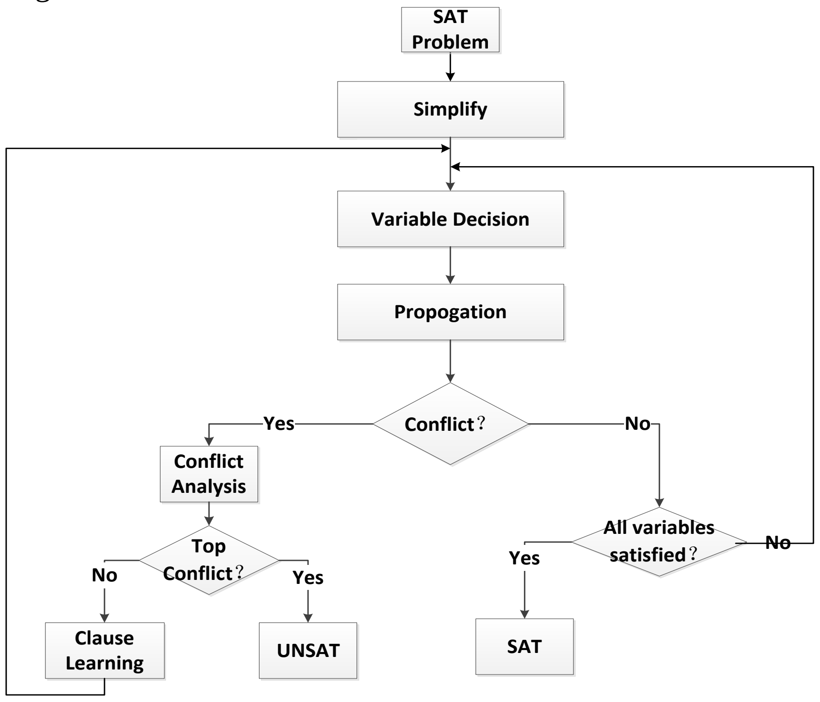

Fig.1. Flow chart of Minisat

As for scheduling strategy, we adopt dynamic scheduling so as to balance the load among threads. This kind of scheduling method dynamically allocates the input SAT cases to threads according to the solving condition of each thread while program is running. What's more, we also apply asynchronous scheduling strategy to make full use of computational resource, that is, each thread will immediatelly deal with the next allocated SAT case rather than wait for other threads. Figure 2 shows the coarse-grained parallel method of Minisat when the solver deals with a set of input SAT problems.

Using this scheduling method, we imagine that one thread is assigned to a set of SAT cases whose solving time in all is equal to that of any other threads, and all threads end at basically the same time, so that threads can run at full capacity in this period of time, making full use of the computing resources of the computer. But things are different when the solver runs in practice. We noticed that threads are not end as the way we imagined. Some threads are still working while others have already finished their jobs.

After analysis, we realize that because the solving time of each input SAT problem varies and these SAT problems are sent into Minisat in an fixed order, such an circumstance will happen:

Thread A have completed the penultimate mission when the system assigns it an easy case. While Thread A is dealing with its last mission, Thread B completes its penultimate mission and system gives it a much more difficult case. So thread B will keep solving this case for a long period of time after Thread A stops, which leads to the different run time of two threads. This explanation is also effective when the number of threads is more than two. The time difference mentioned above can theoretically reach to as much as the solving time of the most difficult SAT problem among the input dataset.

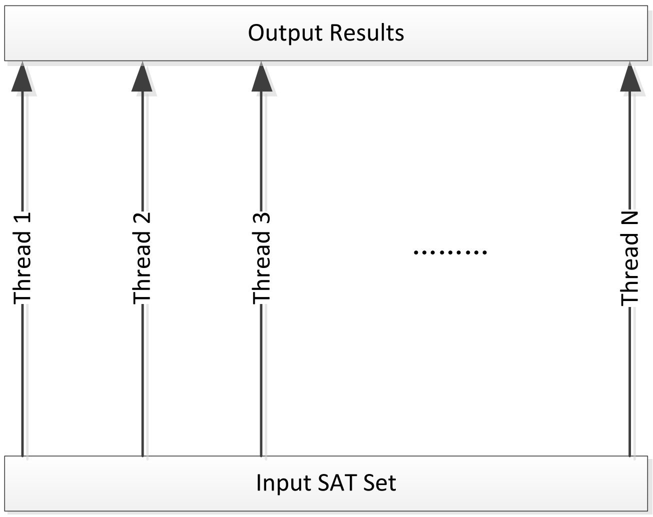

Fig.2. Coarse-grained parallel method of Minisat

We propose a new scheduling algorithm with time sequence in order to solve this problem. Basic idea of this algorithm is that the SAT datasat are ordered by time sequence from large to small according to their practical solving time before sent into Minisat solver, so that cases with a longer solving time will be scheduled prior to those with a shorter solving time. By doing so, each SAT case of thread assigned at last is an easy case, making it possilble to keep run time of each thread basicly equal. Diffrences between the ordinary dynamic scheduling and dynamic scheduling with time sequence are shown in figure 3.

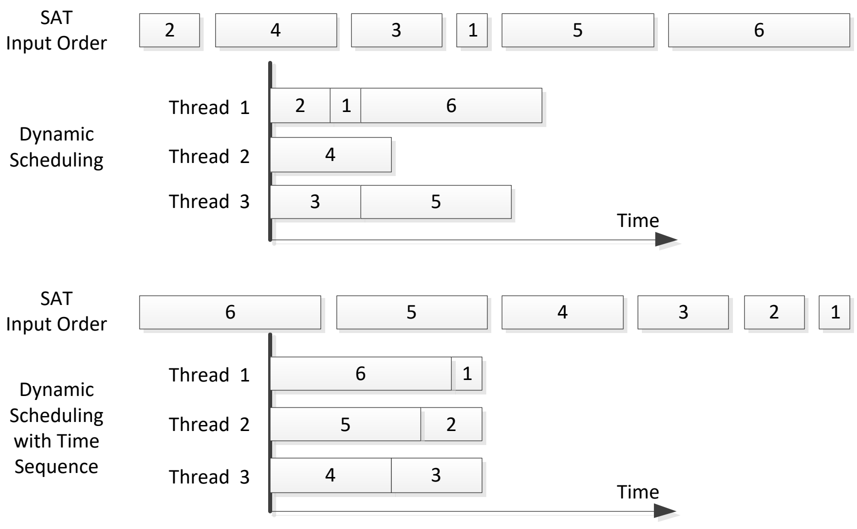

Fig.3. Differences between two scheduling strategies

In order to know a SAT problem's solving time before practically running so as to make the dynamic scheduling strategy with time sequence realizable, we borrow Frank Hutter's time prediction model [9] that is based on random forest. This model can predict the solving time of SAT problems according to their features, making it possible to arrange the input dataset in time sequence from big to small and meet the need of our new scheduling strategy.

Experimental Results and Analysis

We do the experiment on both X86 and ARM platforms. Configuration information of the two platforms are shown in Table 1.

Table 1. Configuration information of X86 and ARM platforms

Platform	X86	ARM
CPU Model	Intel Xeon E5-2650	FT-1500A
Number of CPU	1	2
Cores per CPU	16	8
Hypert-threadings per Core	2	1
Frequency	2.00GHz	1.5GHz
Memory Size	64GB	32GB

Table 2. Testset solving time with different threads on two platforms using two scheduling methods.

Number of thread		1	2	4	8	16	32	64	128
Dynamic Scheduling	X86	4191.3	2444.9	1188.3	576.3	334.8	**237.6**	253.1	270.4
	ARM	8789.9	4481.7	2257.3	1212.4	**700.6**	725.1	751.9	770.7
Dynamic Scheduling with Time Sequence	X86	4191.3	2568.3	1230.3	581.6	310.1	**218.7**	257.0	312.3
	ARM	8789.9	4384.8	2219.8	1143.6	**638.2**	686.1	735.1	778.8

The testdata consists 2000 SAT cases, each of them is able to be solved in less than 100 seconds. During the experiment, we test the two scheduling methods on both platforms. As for each scheduling method, we set different numbers of threads to solve the testdata. The program stops and we record the time used when all of the testset are successfully solved. It's worth noticing that when we test the dynamic scheduling method with time sequence, the time spent on predicting solving time and sorting is not reckoned in.

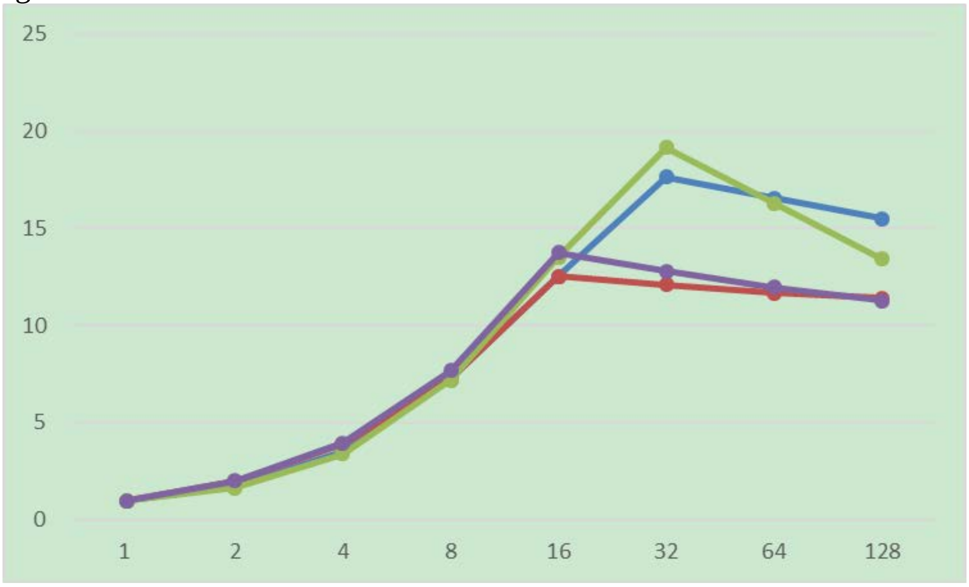

Fig.4. Speed-up ratio of different scheduling methods on different platforms (The horizontal axis represents number of thread while the vertical axis represents speed-up ratio. The green, blue, purple and red line respectively represents dynamic scheduling with time sequence on X86, dynamic scheduling on X86, dynamic scheduling with time sequence on ARM, dynamic scheduling on ARM).

The experiment results are shown in Table 2. Figure 4 shows the speed-up ratio.

After analysis of data from Table 2 and Figure 4, we can come to conclusions:

1. Minisat perfoms best with 32 threads on X86 platform while with 16 threads on ARM

platform, which is in accordance with their own hardware configuration.

2. Dynamic scheduling with time sequence perfoms better than ordinary dynamic scheduling on both platforms. The speed-up ratio increases from 17.64 to 19.16 on X86 and from 12.55 to 13.77 on ARM platform, which is a pretty good perfoamance improvement.
3. ARM platform has a better scale effect than X86 when running with full load. As we can see, Performance of one thread on X86 is about twice as fast as that on ARM, but X86's performance when running with full load is just 3 times as fast as ARM's. This number should be 6 because this X86 computer can support threads twice as many as ARM computer. So ARM computer obtains an extra speedup than X86 by parallelism. Therefore, it is probably that ARM will perform better than X86 if they have the same configuration.

Conclusion

In this paper, we adopt a dynamic scheduling strategy with time sequence into Minisat's coarse-grained reseach and achieve a better speed-up ratio and performance than ordinary dynamic scheduling. What's more, we complete the same test on ARM platform and find that even ARM's overall performance when running Minisat is not as good as X86's, it should perform better when increasing its cores, support threads and cpu performance to X86's level.

Acknowledgement

In this paper, the research was sponsored by the National Key Research and Development Program of China (Project No. 2016YFB0200401) and National Science Foundation of China (Project No. 61303070 and No. 61572515).

References

[1] Cook S A. The complexity of theorem-proving procedures[C]// ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, Usa. DBLP, 1971:151-158.

[2] Een N, Srensson N. An Extensible SAT-solver, Lecture Notes in Computer Science 2919, Springer (2004) 502-518.

[3] Sofroniestokkermans V. DPLL (T): Fast Decision Procedures [J]. 2004.

[4] Chrabakh W, Wolski R. GrADSAT: A Parallel SAT Solver for the Grid[C]// Supercomputing Conference. 2003.

[5] Forman S L, Segre A M. NAGSAT: A Randomized, Complete, Parallel Solver for 3-SAT [J]. Proceedings of Sat, 2002.

[6] Zhang H, BONACINA M.P.A.O.L.A, Hsiang J. PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems [J]. Journal of Symbolic Computation, 1996, 21(4-6):543-560.

[7] Jurkowiak B, Li C M, Utard G. A Parallelization Scheme Based on Work Stealing for a Class of SAT Solvers [J]. Journal of Automated Reasoning, 2005, 34(1):73-101.

[8] Gil L, Flores P, Silveira L M. PMSat: a parallel version of MiniSAT. [J]. Journal on Satisfiability Boolean Modeling & Computation, 2008, 6(6):71-98.

[9] Hutter F, Xu L, Hoos H H, et al. Algorithm runtime prediction: Methods & evaluation[J]. Artificial Intelligence, 2014, 206(206):79-111.