GPU4SAT: solving the SAT problem on GPU

Hervé Deleau, Christophe Jaillet, and Michaël Krajecki

CReSTIC SysCom
Université de Reims Champagne-Ardenne
Moulin de la Housse, BP 1039, 51687 Reims cedex 2
{herve.deleau,christophe.jaillet,michael.krajecki}@univ-reims.fr

Abstract. The performance of Graphics Processing Units (GPU) has increased in an impressive way in the last few years. The explosion of their computational capacity fascinates various scientific communities. However, programming style must be adapted to fully exploit this massively parallel architecture, which is not necessarily straightforward.
This article proposes a first GPU approach for solving the well-known SAT problem. The adaptation phases are detailed, including the problem memory representation and the resolution methods. This approach has been compared with the Walksat method on a standard CPU. This enables us to test our representation of the problem and resolution method on various SAT formula. It also leads to a comparative study of the computational capacity of the most recent generations of CPU's and GPU's.

Key words: SAT problem, GPU, parallel programming, SIMD model, CUDA

1 Introduction

While the CPUs evolution declines (Moore's law [4]), the computational capacity of Graphics Processing Units (GPUs) increases. For a few years, GPU manufacturers enable us to use GPUs for scientific computing, thanks to convenient APIs like CUDA (NVIDIA).

This article proposes a first approach for solving the SAT problem on GPU architectures. After a brief presentation of GPUs, the SAT problem is described and different resolution approaches are given. There is still much to be done, but this initial work already gives good results, even compared with the WalkSAT solver [6] on CPU.

2 GPUs presentation

The difference between a CPU and a GPU is not only in their design but also in their use. Whereas CPUs are designed to treat a set of instructions as quickly as possible, GPUs are designed to handle a set of datas into a limited time. GPUs are composed of a large number of common processors that operate synchronously in a SIMD manner (Single Instruction on Multiple Data), and increase their performances requires to multiply the number of these CPUs.

As described on figure 1, a GPU is composed of a multiprocessors set (up to 16 on GeForce 8800 GTX), each of them consisting in a set of processors (8 on G80 series), with a small amount of shared memory. At each clock cycle, a multiprocessor executes the same instruction on a *block* of threads. A *grid* is a set of blocks executed over the multiprocessors, and its is possible to run up to 512 threads/block.

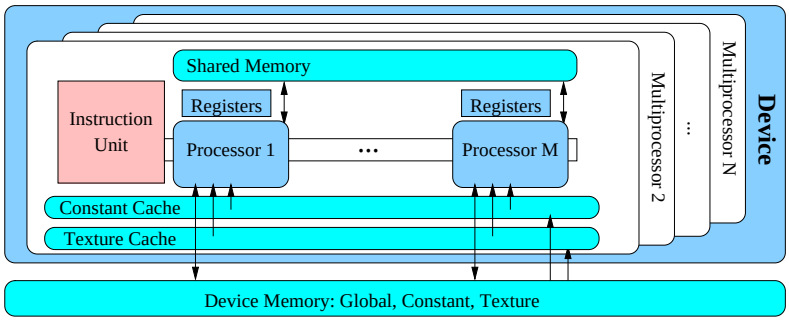

Fig. 1. GPUs hardware model (NVIDIA).

3 SAT definition

The SAT boolean satisfiability problem is a well-know decision problem, reference for all the NP-Complete problems [1]. A SAT instance is a conjunction of literals disjunctions, such as $(x_1 \vee x_2 \vee -x_3) \wedge (x_2 \vee -x_5 \vee -x_6) \wedge (-x_1 \vee x_4 \vee x_6)$, expression consisting of the logical operators AND and OR and NOT on boolean variables. A SAT instance is satisfiable if there is a boolean assignment of the literals such that the expression is true (else it is unsatisfiable).

Solving a SAT problem instance is proving whether it is satisfiable (by exhibiting a solution) or not. There are two approaches for the resolution: the complete methods, generally consisting in tree search (DPLL algorithm [2, 3]); the incomplete ones, such as local search methods, that seek to improve an assignment, looking for a better one in its neighbourhood (GSAT, WalkSat [6]).

4 SAT on GPUs

4.1 Matrix representation

Each SAT problem instance has a natural matricial representation, and testing instanciations consists in a matrixes multiplication, as illustrated by figure 2: a resulting column indicates whether the assigment satisfies all the clauses (it does if it contains no 0).

This representation immediately leads to a complete resolution method, testing all the available instanciations in a wide second matrix. This computation naturally fits with a SIMD scheme on massively parallel architectures, and can

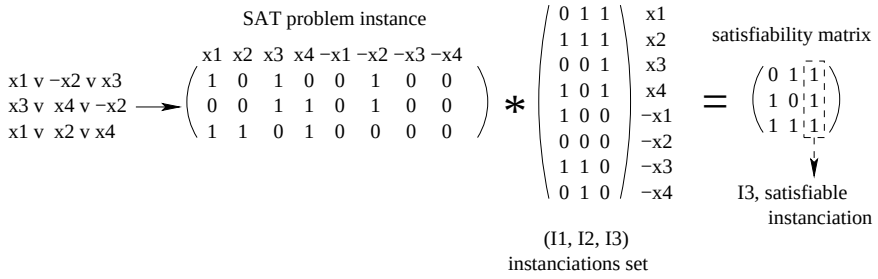

Fig. 2. Solving a SAT problem instance as a matrixes multiplication.

be efficiently implemented on GPUs if it is spread over the processors with a crafty use of memory accesses (the principle is detailed in [5]). Unfortunately the search space exponentially grows with the number of variables and such a complete method can't be applyed, even on small SAT instances.

4.2 Incomplete approaches - GPU4SAT

As it is impossible to test all the instanciations, incomplete methods may be used, based on a heuristic to improve given instances: considerating the number of satisfied clauses as a quality function, local search techniques evaluate the neighbourhood of any instanciation, looking for the best literal to flip. It is simultaneously done on a set of initial assignments distributed over the threads.

It is necessary to get the new conflict matrix after each local search iteration, and this leads to two different approaches: the $MULT$ version computes the product again after each improvment step; the $UPDATE$ one gets it from its previous value, applying the changes only to the concerned clauses.

5 Experimentation

These approaches have been developed using CUDA 1.1, and tested on GeForce 8600 GT and 8400 GS. As SAT reduces to 3-SAT, we limited our experiments to 3-SAT instances (each clause containing 3 literals); we generated them at the threshold (ratio between the number of clauses and variables where the resolution is the most difficult). We only worked on satisfiable SAT problem instances in order to make it possible to compare GPU4SAT to WalkSAT, efficient CPU solver (presented here with the "best" heuristic).

As a first result, le $MULT$ version of GPU4SAT has poor performances because it has to compute the whole conflict matrix at each optimization step. Table 1 even shows, for the $UPDATE$ method on GeForce 8400/8600, the influence on the performance of the GPU number of processors, and of the qualitative peformance of the GPU cards (the ratio of 2 is improved by the processors frequency and by the memory transfer rates). Note that the table presented here gives, with the number of iterations treated and the number of variables flips performed each second:

– LS: average time for local search and conflic computation
– TIME: global computational times observed

Method	LS (ms)	TIME (ms)	NB ITER	FLIPS/s	Success
UPDATE 8400	12.16	32584.2	2611x512	41034	100%
UPDATE 8600	3.46	8312.6	2306x512	142064	100%
Walksat best		313.0	212614	679279	100%

Table 1. GPU4SAT *vs* WalkSAT : 256 variables 3-SAT instances, 1088 clauses

Compared with WalkSAT-best (CPU), the GPU4SAT-UPDATE on 8600 already offers a good flips/s rate (at least 570000 expected with GeForce 8800), but the global computation time and number of iterations performed show that the method still can be improved, in the field of the heuristic used.

6 Conclusion

We have presented here the GPU4SAT a local search method for solving SAT problem instances on GPU architectures. Taking advantage of the massively parallel structure of the GPUs, this resolution method already offers interesting results; GeForce8800 and future NVIDIA G90 series should bring their expected power and improve the results.

This approach can be optimized in different ways, first by taking into account the SIMD execution model of GPU architectures in a more accurate manner, and by using the different types of memory available on GPUs in an optimized way in order to minimize the accesses costs (constant or texture memory may offer interesting transfer rates compared to global memory).
The algorithm can also be improved by using heuristics inspired by the best solvers available at the moment (those used by WalkSAT for example).

References

1. Cook, Stephen A. The complexity of theorem-proving procedures. STOC '71: Proceedings of the third annual ACM symposium on Theory of computing. ACM Press. 151–158 (1971)
2. Davis, M., Putman H. : A Computing Procedure for Quantification Theory. J. ACM. 7, 201–215 (1960)
3. Davis, M. Logemann G., Loveland, D. A machine program for theorem-proving. Commun. ACM. 5, 394–397 (1962)
4. Moore, G. E. : Cramming More Components Onto Integrated Circuits. Proceedings of the IEEE. 86, 82–85 (1998)
5. NVIDIA. CUDA Programming Guide 1.1. `http://www.nvidia.com/object/cuda_develop.html`, 67–72 (2007)
6. Selman, D., Kautz, H.A., Cohen B. : Local Search Strategies for Satisfiability Testing. Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability. Michael Trick and David Stifler Johnson. (1993)