Efficient Workload Balancing

Max Böhm, Ewald Speckenmeyer
Universität zu Köln, Institut für Informatik, D–50969 Köln
e–mail: {boehm|speckenmeyer}@informatik.uni–koeln.de

Proposal

In this proposal we consider the satisfiability problem (SAT) for boolean formulas in conjunctive normal form on a transputer–network of up to 1024 processors. The hardware as well as the choice of the SAT–problem are used in order to demonstrate a simple but very effective method of dynamically distributing workload among the processors of a distributed multiprocessor machine. The workload balancing algorithm immediately can be applied for several network topologies like k–dimensional processor arrays, $k \geq 1$, processor trees or hypercubes.

In case of the SAT problem we are essentially interested in solving hard random instances. We have developed a Davis–Putnam based satisfiability solver. The quality of such a solver heavily relies on the ability of predicting, which of the unset literals should be set true next in order to hold the search space as small as possible. By experiments with k–SAT formulas (all clauses have a fixed length k) with varying ratios of clauses and variables we have singled out a strategy essentially computing for every unset literal a vector weighting the occurrences of the literal and its complement in clauses of length $i = 2, 3, \ldots, k$. A literal with highest vector under the lexicographic order will be chosen next. Each step of truth setting a literal, and consequently of its complementary literal, causes an update of the current subformula. This update should be as efficient as possible, i.e. a good data structure for representing formulas is needed. In order to be able to represent any boolean formula in conjunctive normal form, we have used a purely pointer based data structure. This data structure needs linear storage space and allows for an optimal execution of nearly all data structure operations, which have to be performed by our algorithm, see chapter 3. The implementation of this algorithm turned out to be the fastest program among 35 programs in a SAT competition, see [1].

In order to be able to further speed up our algorithm, we have implemented it on a parallel computer, a Transputersystem built up from up to 1024 processors (INMOS T800/4MB). Transputersystems enable the programmer to realize every net–topology with the restriction that every processor is connected with at most four other processors. All experiments were run on 2-dimensional grids of up to 32×32 processors and on linear processor arrays. We want to mention here in advance, that a SUN SPARC 10 processor works at least 10 times faster than a T800 processor. What we are interested in and what we have achieved however, is to speed up our sequential SAT–solver by a factor of nearly $1/N$ by running a copy of it on each of the N processors and distributing the workload between the processors in a convenient way.

As mentioned above, our parallel SAT–solver runs a copy of the sequential SAT–solver on every processor, and the N processors cooperate, when searching for a solution in the space of partial truth assignments, by partitioning this space. That way each processor will be provided with a certain amount of workload represented by its subspace of partial

truth assignments. The most difficult part, which has to be solved when running the SAT–solver on a parallel computer, consists of balancing the workload between the processors in such a way that on the one side idle times of processors should almost always be avoided, on the other side the workload balancing phase should consume as few computing time as possible. This is a nontrivial task. Our parallel algorithm redistributes workload between the processors at certain points of time. This is necessary, because no reliable estimation of workload represented by a partial truth assignment, which has not yet been singled out not to lead to a satisfying truth assignment, is known. We have used as workload estimation the function α^n for a partial truth assignment with n unset variables, for varying values of α between 1.04 and 1.42, depending on the parameters of the class of formulas from which the input formula is chosen.

The workload redistribution phase is activated, if the estimated workload for some processor goes down some limit. Then, in case of a linear processor array, the following steps are performed in order to achieve a uniform mapping of the remaining workload to the processors. In the first step the prefix sums of the workload estimations for all processors are determined for the linear array of N processors. In a second step the last processor, which knows the total amount L of estimated workload and the number N of processors, broadcasts the ratio $\mu = L/N$, i.e. the amount of workload for each processor in case of a uniform workload distribution among the N processors, to all processors. Finally in a third step, each processor p knows due to the knowledge of μ and its rank in the linear processor array, i.e. how many processors are to the left of p, whether workload has to be sent from p to its left (right) neighbor, or whether p has to receive workload from its left (right) neighbor. Thus workload redistribution among the processors now can be performed in at most N–1 steps, where the amount of workload moved between the processors in as small as possible. These three steps can be performed in time $O(N)$. The workload redistribution phase for rectangular grids is achieved by first performing the above workload balancing phase for all linear arrays of processors linked in the first dimension and then for all linear arrays of processors linked in the second dimension. I.e., in case of square grids with N processors the workload distribution phase can be performed in time $O(\sqrt{N})$. The strategy mentioned above can be extended to processor trees, leading to an $O(depth\ of\ tree)$ algorithm or to hypercubes with running–time of $O(log(N))$.

We have run many experiments with our parallel SAT–solver with up to 1024 processors. We want to stress the point, that we have obtained an efficiency close to 1. Obtaining these results is not a trivial task, and it requires a lot of tuning.

At the moment we apply the workload redistribution algorithm in order to solve feedback problems for graphs and we also want to apply it for the factorizing problem of large numbers.

References

[1] Buro M., Kleine Büning H.: *Report on a SAT competition*, EATCS Bulletin, No 49, Feb 1993, 143–151