Some Implementation Results on Random Polling Dynamic Load Balancing

Peter Sanders
Department of Computer Science
University of Karlsruhe, 76128 Karlsruhe, Germany
Email: sanders@ira.uka.de

Abstract

Using two sample applications, we demonstrate the effectiveness of our portable and reusable library for parallel tree search. On 1024 Transputers we achieve near optimal speedup even for quite small instances of the *Golomb ruler* problem. The *0/1 knapsack problem* is more challenging but it is possible to achieve superlinear speedup compared to the standard sequential depth first algorithm the implementation is based on.

1 The Problem

Many algorithms in operations research and artificial intelligence are based on depth first search in implicitly defined trees. Since these problems are often very time consuming, parallelization is an attractive approach to extend the applicability of these algorithms. Unfortunately, the current status of parallel programming tools and the background of application programmers make an effective parallelization difficult. This problem can be (partially) solved by encapsulating the parallelization into an application independent library. We outline the design and implementation of such a library and demonstrate its effectiveness using the following two examples:

A *Golomb ruler* [1] of length m with n marks at the integer positions $0 = m_1$, m_2, $\ldots$, $m_n = m$ has the property $|\{m_j - m_i : 1 \leq i < j \leq n\}| = n(n-1)/2$, i.e., the ruler can be used to measure a maximum number of distances. For a given n we want to find a Golomb ruler with minimal m. These rulers have applications in interferometry and telematics.

An instance of the *0-1 knapsack problem* is defined by n items with weight w_i and profit p_i and a knapsack of capacity M. We are looking for $x_i \in \{0,1\}$ such that $\sum p_i x_i$ is maximized subject to the constraint $\sum w_i x_i \leq M$. Next to the traveling salesman problem, the knapsack problem might be one of the most extensively studied discrete optimization problems [5].

2 Sequential Algorithms

The Golomb ruler problem is solved using backtracking. On level i of the search tree mark i is placed. Starting from a trivial approach, we introduced a number of heuristics which reduced the sequential execution time by two orders of magnitude. The search tree therefore has a quite irregular shape, but it is not very deep and it remains wide and bushy. Node evaluations are linear in m.

There are two basic approaches to exact solutions of the knapsack problem. Dynamic programming is good if n is not too large and the w_i lie within a small discrete range. In other cases, dynamic programming fails due to its exponential memory requirements. For these cases, variants of depth first branch-and-bound are better. First, the items are sorted by their profit-density (from now on, let w_i, p_i refer to the i-th best item); then depth first branch-and-bound traverses a binary search tree where x_i is determined at level i of the tree. Lower bounds for the branch-and-bound heuristics are based on relaxing the integrality constraints on the x_i. The bounds can be computed quickly (in $O(\log n)$ time) using binary search and some precomputation. The instances we consider have a very deep but thin irregularly shaped search tree.

3 Parallelization

Our library PIGSeL (parallel implicit graph search library) is a layered system of interchangeable modules. The only part which needs to be ported to a new machine is a *messaging interface* supporting asynchronous communication including a simple form of active messages. *Collective operations* like broadcast or reduce are implemented on top of this although they can use native routines when desired. A *load balancing and coordination* module handles the parallelization using a simple interface to a *search engine*. Besides sequential search, the search engine provides functionality for splitting the search space. This is usually done by manipulating an explicitly managed search stack. For coarser grained problems like Golomb rulers even the search engine is generic – the user only has to define *node evaluation and expansion* functionality.

The central aspect of parallelization is load balancing. We currently use random polling [3] – an almost penetrantly simple algorithm. Every PE (processing element) works on a single subproblem at a time. When the subproblem is exhausted, it asks a randomly chosen other PE to split its subproblem. When the requested PE is also idle, another random PE is choosen. It can be shown [10] that for a large class of problems it is unlikely that any PE has to issue more than $O(\log P)$ requests overall (Let P denote the number of PEs). So, if the per PE load is larger than the cost for communicating $O(\log P)$ subproblems, arbitrarily high efficiency can be achieved.

In order to get a scalable implementation, a number of additional aspects have to be considered:

- Each PE is initialized with a subproblem using a single broadcast of the root problem and $\lceil \log_2 P \rceil$ subsequent local splits based on the PE number.

- We use a termination detection algorithm based on embedding a binary tree into the network. Its time consumption is proportional to the network diameter [6]. In contrast, the more popular ring based schemes take time $O(P)$.

- When a new solution is found in branch-and-bound, the new bound needs to be quickly disseminated to all PEs. We do this by indirectly sending the value to PE 0 along a binary reduction tree. Values which are only locally optimal are discarded as soon as possible. Only when the value has reached PE 0 it is broadcasted to all PEs. If locally improved solutions were immediately broadcasted, this would result in severe network contention for applications like the knapsack problem where many suboptimal solutions are found initially.

- For the knapsack problem, the search space is split by evenly dividing open subproblems on all tree levels between the subproblems [8]. The simpler (and often sufficient) approach of only splitting the top open problem would generate very unequal splits most of the time.

4 Results

All measurements we are presenting here were performed on a Parsytech GCel-3/1024;[1] a 32×32 mesh of 30MHz T805 transputers. We use the parallel operating system Cosy [2]. Refer to [11] for measurements with PIGSeL using PVM on a network of workstations.

The algorithm for the Golomb ruler problem is well suited for studying scalability issues. In Figure 1 it is used to verify that the shortest known ruler with 12 respectively 13 marks are indeed optimal. In this mode, the search tree is independent of the order in which subtrees are evaluated. We therefore get smooth, well reproducible speedup curves. (Small fluctuations due to the randomized load balancer have been damped by averaging over 5 measurements). Even for the quite small problem with 12 marks we get a speedup of 578 at a parallel execution time of $0.88s$. For even larger problems we achieve almost perfect speedup. (Compared to the specialized sequential algorithm.) Since there is little communication taking place, large backtracking problems like finding new Golomb rulers can also be successfully parallelized on networks of workstations.

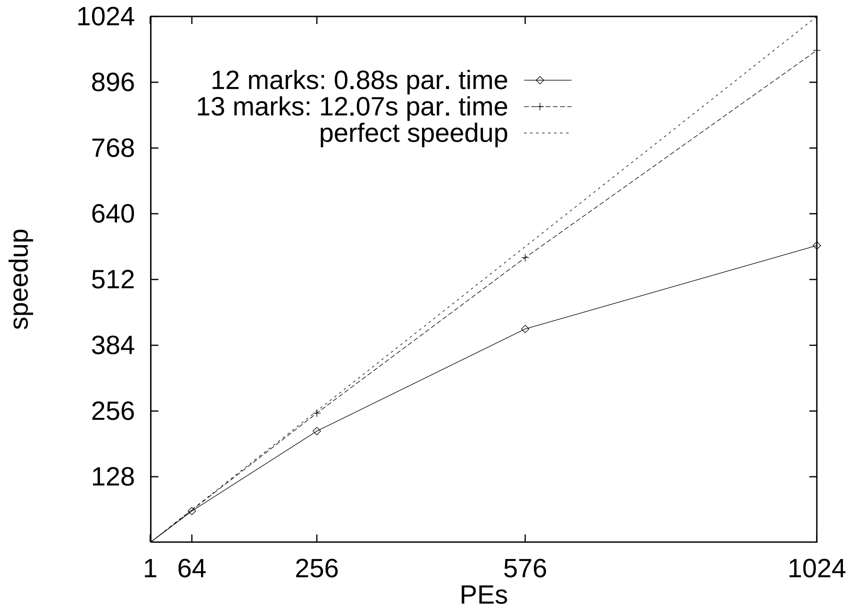

Figure 1: Speedup for Golomb rulers.

[1]We would like to thank the Paderborn Center for Parallel Computing (PC^2) for making this machine available.

The effectivity of parallelizing the knapsack problem is very dependent on the instances considered. One can generate very large trees with very small $n \approx 50$. These are easy to parallelize. On the other hand, in [5] a class of very easy instances with n up to 250000 is considered where sorting the items by profit density is the limiting factor – there is virtually no parallelism in the tree search phase. We have generated 256 random instances with $n = 2000$, $w_i \in [0.01, 1.01]$, $p_i \in [w_i + 0.1, w_i + 0.125]$ using the (32-bit) random number generator of INMOS-C. This statistics was chosen to provide nontrivial but still sequentially tractable problems with large n. The double-logarithmic plot in Figure 2 shows the relation between speedup and sequential execution time. There is a large number of very small problems for which we cannot expect any significant speedup. Beginning at per PE loads of about 10s we start to observe good performance. Very large problems show a considerable superlinear speedup. For these instances the sequential algorithm appears to have run into some kind of "dead end". The parallel algorithm is more robust because it follows multiple search paths at once. The overall parallel execution time for 1024 PEs is 1410 times smaller than the sequential time. This indicates that the traditional pure depth first strategy is not the best choice for a sequential algorithm.[2]

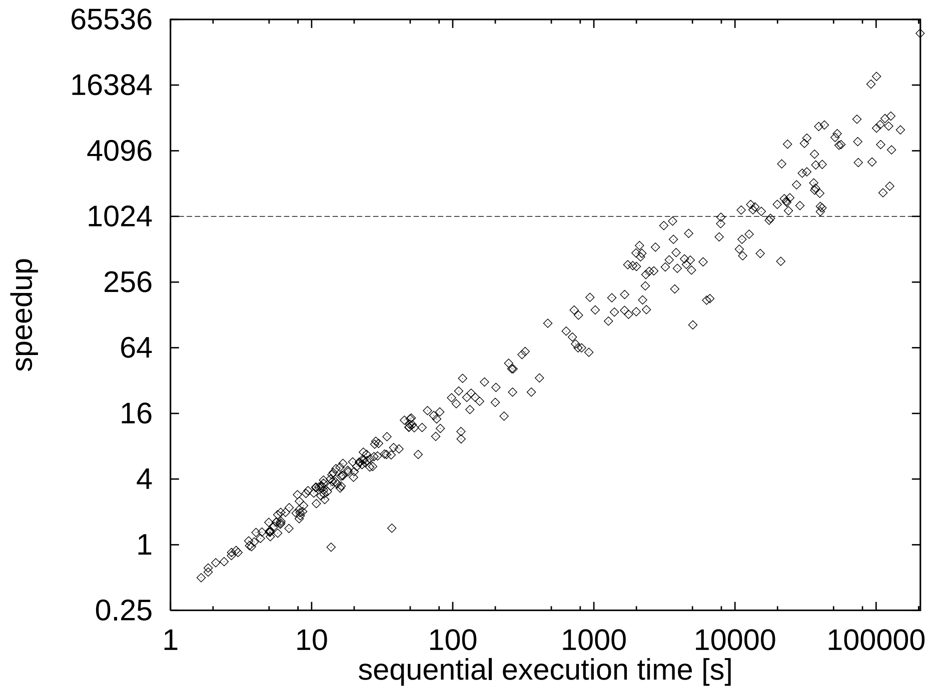

Figure 2: Speedup for 256 instances of the knapsack problem on 1024 PEs.

5 Conclusions

We have demonstrated that it is possible to efficiently exploit large parallel machines for searching irregularly shaped trees using a reusable portable parallel library. The random polling algorithm, which was known to be successful on low diameter networks

[2]But note that traditional best first methods are no alternative here because they have excessive memory requirements and a large management overhead considering the fast node evaluation functions available for the knapsack problem.

[3], is equally effective on high latency machines with software routing. The results for small golomb ruler problems demonstrate an efficient parallel execution time which is at least an order of magnitude smaller than previous results [9, 12]. Much of this effectivity carries over to more challenging problems like the knapsack problem which requires fine-grained depth first branch-and-bound with a deep thin search tree and frequent bound updates. Previous work on parallelizing the knapsack problem [4, 7] did not yield comparable results.

References

[1] G. S. Bloom and S. W. Golomb. Applications of numbered undirected graphs. *Proceedings of the IEEE*, 65(4):562–570, April 1977.

[2] R. Butenuth and S. Gilles. COSY — ein Betriebssystem für hochparallele Computer. In *Transputer Anwender Treffen*, Aachen, 1994.

[3] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms.* Benjamin/Cummings, 1994.

[4] W. Loots and T. H. C. Smith. A parallel algorithm for the 0-1 knapsack problem. *International Journal of Parallel Programming*, 21(5):349–362, 1992.

[5] S. Martello and P. Toth. *Knapsack Problems – Algorithms and Computer Implementations.* Wiley, 1990.

[6] F. Mattern. *Verteilte Basisalgorithmen.* Number 226 in Informatik-Fachberichte. Springer, 1987.

[7] G. P. McKeown, V. J. Rayward-Smith, and S. A. Rush. Parallel branch-and-bound. In *Advances in Parallel Algorithms*, pages 349–362. Blackwell, 1992.

[8] V. N. Rao and V. Kumar. Parallel depth first search. Part I. *International Journal of Parallel Programming*, 16(6):470–499, 1987.

[9] A. Reinefeld and V. Schnecke. Work-load balancing in highly parallel depth-first search. In *Scalable High Performance Computing Conference*, pages 773–780, Knoxville, 1994.

[10] P. Sanders. A detailed analysis of random polling dynamic load balancing. In *International Symposium on Parallel Architectures Algorithms and Networks*, pages 382–389, Kanazawa, Japan, 1994.

[11] P. Sanders. Portable parallele Baumsuchverfahren: Entwurf einer effizienten Bibliothek. In *Transputer Anwender Treffen*, pages 168–177, Aachen, 1994. IOS.

[12] S. Tschöke, M. Räcke, R. Lüling, and B. Monien. Solving the traveling salesman problem with a parallel branch-and-bound algorithm on a 1024 processor network. Technical report, Universität Paderborn, 1994.