DIMACS TECHNICAL REPORT 94-20
April 1994

DIMACS Workshop on Parallel Processing of Discrete Optimization Problems

April 28–29, 1994

Organizers
P.M. Pardalos [1] (Department of Industrial and Systems Engineering, University of Florida)
M.G.C. Resende (AT &T Bell Laboratories, Murray Hill)
K.G. Ramakrishnan (AT &T Bell Laboratories, Murray Hill)

[1]DIMACS Visitor

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories, and Belcore.
DIMACS is an NSF Science and Technology Center, funded under contract STC-91-19999; and also received support from the New Jersey Commission on Science and Technology.

ABSTRACT

In the context of the 1993-1994 DIMACS special year on Parallel Computing, a two-day workshop entitled "Parallel Processing of Discrete Optimization Problems" was held on April 28–29. The workshop featured about twenty invited speakers from Europe and North America. This report containts the abstracts of the talks. A DIMACS volume with complete refereed papers from the workshop will be also published.

The workshop was sponsored by DIMACS with funds from National Science Foundation and The New Jersey Commission on Science and Technology.

Introduction

Discrete optimization problems (DOPs) arise in various applications such as planning, scheduling, computer aided design, robotics, game playing and constraint directed reasoning. Often, a DOP is formulated in terms of finding a (least cost) solution path in a graph from an initial node to a goal node and solved by graph/tree search methods such as branch-and-bound and dynamic programming. Availability of parallel computers has created substantial interest in exploring the use of parallel processing for solving discrete optimization problems (se some references below).

In this workshop twenty invited speakers presented their work on parallel algorithms for solving discrete optimization problems.

Formally, a DOP can be stated as follows : *Given a finite discrete set X and a function $f(x)$ defined on the elements of X, find an optimal element x_{opt}, such that, $f(x_{opt}) = min\{f(x)|x\epsilon X\}$.* In certain problems, the aim is to find any member of a solution set $S \subset X$. These problems can also be easily stated in the above format by making $f(x) = 1$ for all $x\epsilon S$, and $f(x) = 0$ for all other elements of X.

In most problems of practical interest, the set X is quite large. Consequently, exhaustive enumeration of elements in X to determine x_{opt} is not feasible. Often, elements of X can be viewed as paths in graphs/trees, the cost function can be defined in terms of the cost of the arcs, and the DOP can be formulated in terms of finding a (least cost) solution path in the graph from an initial node to a goal node. Branch and bound and dynamic programming methods use the structure of these graphs to solve DOPs without searching X exhaustively.

Given that many classes of DOP are NP-hard, one may argue that there is no point in applying parallel processing to these problems, as the worst-case run time can never be reduced to a polynomial unless we have an exponential number of processors. However, the average time complexity of heuristic search algorithms for many problems is polynomial. Also, there are some heuristic search algorithms which find suboptimal solutions in polynomial time (e.g., for certain problems, approximate branch-and-bound algorithms are known to run in polynomial time). In these cases, parallel processing can significantly increase the size of solvable problems. Some applications using search algorithms (e.g. robot motion planning, task scheduling) require real time solutions. For these applications, parallel processing is perhaps the only way to obtain acceptable performance. For some problems, optimal solutions are highly desirable, and can be obtained for moderate sized instances in a reasonable amount of time using parallel search techniques (e.g. VLSI floor-plan optimization).

Parallel computers having thousands of processing elements are now commercially available. The cost of these machines is similar to that of large mainframes, but they offer significantly more raw computing power. Due to advances in VLSI technology and economies of scale, the cost of these machines is expected to go down drastically over the next decade. It may be possible to construct computers comprising thousands to millions of processing elements at costs ranging from those of high-end workstations to large mainframes. This technology has created substantial interest in exploring the use of parallel processing for search based applications.

The talks in this workshop have covered a wide spectrum of algorithms and applications in parallel processing of discrete optimization and related problems.

References

[1] Grama Y. Ananth, Vipin Kumar and P. M. Pardalos. Parallel Processing of Discrete Optimization Problems. In *Encyclopedia of Microcomputers* Vol. 13 (1994), pp. 129-147, Marcel Dekker, Inc. New York (1993).

[2] Vipin Kumar and V. N. Rao. Scalable parallel formulations of depth-first search. In Vipin Kumar, P. S. Gopalakrishnan, and Laveen Kanal, editors, *Parallel Algorithms for Machine Intelligence and Vision*. Springer-Verlag, New York, 1990.

[3] T.A. Feo, M.G.C. Resende and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 1994.

[4] Vipin Kumar and V. Nageshwara Rao. Parallel depth-first search, part II: Analysis. *International Journal of Parallel Programming*, 16 (6):501–519, 1987.

[5] T. H. Lai and Sartaj Sahni. Anomalies in parallel branch and bound algorithms. *Communications of the ACM*, pages 594–602, 1984.

[6] Y. Li and P.M. Pardalos. Parallel algorithms for the quadratic assignment problem. In P.M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 177–189. North-Holland, 1992.

[7] V. Nageshwara Rao and Vipin Kumar. Parallel depth-first search, part I: Implementation. *International Journal of Parallel Programming*, 16 (6):479–499, 1987.

[8] P.M. Pardalos and J.Crouse. A parallel algorithm for the quadratic assignment problem. In *Proceedings of the Supercomputing 1989 Conference*, pages 351–360, ACM Press, (1989).

[9] P.M. Pardalos and X.Li. Parallel branch and bound algorithms for combinatorial optimization. *Supercomputer*, 39:23–30, 1990.

[10] P.M. Pardalos (Editor). Advances in Optimization and Parallel Computing. North-Holland, 1992.

[11] P.M. Pardalos, A.T. Phillips and J.B. Rosen. Topics in Parallel Computing in Mathematical Programming. Science Press (1993).

[12] M.G.C. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite networks. *SIAM Journal on Optimization*, 3:516–537, 1994.

SCHEDULE and ABSTRACTS

PARALLEL PROCESSING OF DISCRETE OPTIMIZATION PROBLEMS

April 28-29, 1994

DIMACS

Organized by P.M. Pardalos, M.G.C. Resende, and K.G. Ramakrishnan

Thursday, April 28, 1994

9:00 -9:10	**Welcome from the DIMACS Director and the Organizers**
Session T.A	Chairman: P. Pardalos
9:10	David Applegate, R. Bixby, V. Chvatal and W. Cook **Solving Large TSPs in Parallel**
9:45	Jonathan Eckstein **Parallel Branch-and-Bound Techniques for General Mixed Integer Programming on the CM-5**
10:20	Jun Gu **Parallel Processing for Satisfiability (SAT) Problems**
10:55-11:15	**Break**
Session T.B	Chairman: M.G.C. Resende
11:15	Gerasoulis Apostolos **A Multistage Approach to Scheduling Task Graphs and its Impact to Parallel Numerical Computing**
11:50	Christian Schütt and Jens Clausen **Parallel Algorithms for the Assignment Problem - An Experimental Evaluation of 3 Distributed Algorithms**
12:25-2:15	**Lunch**

Session T.C	Chairman: K.G. Ramakrishnan
2:15	Per S. Laursen **Experiences with a Synchronous Parallel Branch and Bound Algorithm**
2:50	Reinhard Lüling, Burkhard Monien, Stefan Tschöke **Load Balancing for Distributed Branch & Bound Algorithms: Experiments and Theory**
3:25	George Karypis, Anshul Gupta, and Vipin Kumar **Parallel Formulation of Interior Point Algorithms**
4:00-4:20	**Break**
Session T.D	Chairman: P.M. Pardalos
4:20	Costas Busch and Marios Mavronicolas **Proving Correctness of Balancing Networks**
4:55	R. Corrêa and A. Ferreira **On (A)synchronous Parallel Branch-and-Bound**
5:30	Van-Dat Cung and Bertrand Le Cun **A Suitable Data Structure for Parallel A***
7:00	**Banquet**

Friday, April 29, 1994

Session F.A	Chairman: M.G.C. Resende
9:00	Catherine Roucairol **Parallelism and Irregular Data Structures**
9:35	J. MacGregor Smith and Kui Xu **Serial & Parallel Algorithms for Quadratic Set Packing Problems**
10:10	Robert Meyer **Parallel Genetic Algorithms in Network Optimization**
10:45-11:05	**Break**
Session F.B	Chairman: K.G. Ramakrishnan
11:05	Shantanu Dutt and Nihar R. Mahapatra **New Anticipatory and Scalable Load Balancing Strategies for Parallel Best First Search**
11:40	Santosh Pande and Kleanthis Psarris **Scheduling Acyclic Task Graphs on Distributed Memory Parallel Architectures**
12:15	Alexander Reinefeld **Scalability of Massively Parallel Depth-First Search**
12:50-2:30	**Lunch**

Session F.C	Chairman: P. Pardalos
2:30	Andrea Clementi, Ludek Kucera and José Rolim **A Note on Parallel Randomized Algorithms for Searching Problems**
3:05	Jaideep Srivastava, Thomas M. Niccum and Bhaskar Himatsingka **Query Optimization and Processing in Parallel Databases**
3:40-4:00	**Break**
Session F.D	Chairman: M.G.C. Resende
4:00	Theodore Johnson **Asynchronous Distributed Data Structures: Lists and Priority Locks**
4:35	Boleslaw Mikolajczak **Parallel Distributed Algorithm to Compute all Homomorphisms of Finite Atomata**
5:10	Zhijun Wu **ϵ-Optimal Solutions to Nonlinear 0-1 Integer Programs Using the Subgradient Algorithm**
5:45	**Closing Remarks from the Organizers**
7:00	**Dinner**

Solving Large TSPs in Parallel

David Applegate[2]
R. Bixby[3]
V. Chvatal[4]
W. Cook[5]

The Traveling Salesman Problem is one of the classic NP-complete combinatorial optimization problems. I will discuss some of the techniques we used to solve some large (up to 4461 city) problems in parallel on a bunch of workstations.

[2]AT&T Bell Laboratories, Murray Hill, NJ 07974, david@research.att.com
[3]Rice University, bixby@rice.edu
[4]Rutgers University, chvatal@cs.rutgers.edu
[5]Bellcore, bico@bellcore.com

A Suitable Data Structure for Parallel A*

Van-Dat Cung[6]
Bertrand Le Cun[7]

We present a new parallel implementation of the heuristic state space search A* algorithm. We show the efficiency of a new double criteria data structure called *treap*, instead of usual priority queues (heaps). This data structure allows operations such as *Insert*, *DeleteMin* according to one criterion and *Search* on the other criterion. These operations are essential in the A* algorithm.

Furthermore, we give concurrent algorithm of the treap within a shared memory environment. Results on the 15 puzzle are presented; they have been obtained on two machines, with virtual or not shared memory, the KSR1 and the Sequent Balance 8000.

[6]Laboratorie PRiSM, Université de Versailles, F-78000 Versailles, France
[7]Laboratorie PRiSM, Université de Versailles, F-78000 Versailles, France

On (A)synchronous Parallel Branch-and-Bound[8]

R. Corrêa[9]
A. Ferreira[10]

We theoretically compare the efficiency of two versions, sequential and parallel (synchronous), of the method called *branch-and-bound* used for searching for an optimal solution in the scope of combinatorial optimization. The sequential version consists of successive decompositions of the original problem in smaller disjoint subproblems and its parallel version consists of a high level parallelization that assumes a shared memory model of parallel computation. Its principle consists of decomposing several subproblems in parallel at each iteration in a synchronous way, and its efficiency is usually defined with respect to its sequential counterpart.

In this work, we propose to measure the *effectiveness* in a parallel execution of a *branch-and-bound* algorithm as the rate of processor utilization for doing useful work, e.g., in iterations in where an antecedent of an optimal solution is decomposed. This measure takes into account the two factors involved in the efficiency of parallel *branch-and-bound* algorithms, namely, the *parallel work of the processors* and the *quality of the heuristic selection function.*

The goal of this study is twofold. First a question refering to the existence of inherently sequential *branch-and-bound* algorithms, was posed and partially answered by Clausen[11]. We shall show that, for each instance of a problem, the *effectiveness* does decrease beyond a determined number of processors. Futhermore, we shall answer affirmatively the following question: can the parallel algorithm be more effective than the sequential one?

The *knapsack problem* is used as an application and some simulation results are presented to validate and illustrate the use of the new efficiency measures.

[8]This work was partially supported by DRET and the project Stratagème of the French CNRS.

[9]Partially supported by a CNPq (Brazil) fellowship, grant 201421/92-5 (BC). LMC – IMAG/INPG, 46, av. Félix Viallet, 38031 Grenoble Cédex, France, correa@aquilon.imag.fr

[10]CNRS - Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, 46, Allée d'Italie, 69364 Lyon Cédex 07 - France, ferreira@lip.ens-lyon.fr

[11]J. Clausen, Do inherently sequential branch-and-bound algorithms exist? *Parallel Processing Letters*, to appear

Scalability of Massively Parallel Depth-First Search

Alexander Reinefeld[12]

We compare the scalability of two generic schemes for heuristic depth-first search (DFS) on highly parallel MIMD systems. The first one employs a task attraction scheme where the work packets are generated on demand by splitting the donors' stack [1]. Analytical and empirical investigations show that this scheme is only suitable for moderately parallel systems with, e.g., ≤ 256 processors. The recursive stack-splitting generates successively smaller work packets of unpredictable sizes which might be too small to justify shipment to another processor. In its iterative-deepening version (PIDA*) the communication overhead increases very much over the iterations.

Our second scheme also employs a task attraction scheme, but uses pre-computed work packets instead. This *fixed-packet scheme* [2] starts with a good initial work distribution that is generated by a redundant breadth-first search of the first few tree levels on all processors. Later on, work packets of fixed sizes are shipped to idle processors on demand. Experiments with the 15-puzzle and floorplan-optimization problems on a $32 \times 32 = 1024$ MIMD system indicate that the fixed-packet scheme is better scalable than the stack-splitting method.

Analytical results show that the amount of work that is needed to achieve constant efficiency with increasing system size p needs to grow with order $O(\log p \cdot p^2)$ for the stack-splitting scheme, while the fixed-packet approach needs a work increase of $O(p^2)$.

The iterative-deepening variant of the fixed-packet scheme, *AIDA**, continuously improves the global load balance by changing ownerships of the transferred work packets. As a consequence, the communication overhead decreases in the course of the search process.

Unlike PIDA*, AIDA* is portable in the sense that existing sequential search routines can be linked to AIDA*'s communication routines for implementation on a large scale MIMD system.

References

[1] V. Kumar and V.N. Rao. *Scalable parallel formulations of depth-first search.* In: Kumar, Gopalakrishnan, Kanal (eds.), Parallel Algorithms for Machine Intelligence and Vision, Springer-Verlag (1990), 1 – 41.

[2] A. Reinefeld and V. Schnecke. *AIDA* – Asynchronous Parallel IDA*.* 10^{th} Canadian Conf. on Art. Intell. AI'94, (May 1994), Banff, Canada.

[12] Paderborn Center for Parallel Computing, D-33095 Paderborn, Germany, ar@uni-paderborn.de

Parallel Distributed Algorithm to Compute all Homomorphisms of Finite Automata

Boleslaw Mikolajczak[13]

The problem of finding (generalized) homomorphisms between two automata is NP-complete. Similarly, the problem of finding the set of all (generalized) homomorphisms is also NP-complete. On the other hand, homomorphisms are very important in modeling and decomposition of finite state systems in different areas of computer science. Analysis of two major known sequential algorithms (decomposition into primaries due to Bavel and decomposition into autonomous factors due to Grzymala-Busse and Mikolajczak [1, 2]) is first presented from parallelization point of view. The sequential algorithm based on functional decomposition of automata into autonomous factors has been chosen for parallelization. Two versions of explicit parallelization are considered: data parallelism and process parallelism. As a result, process parallelism paradigm has been selected for detailed analysis and distributed implementation using transputer-based reconfigurable architecture. This parallelism adheres to the Hoare's CSP model.

References

[1] X. Hua. *Implementation of a parallel algorithm of automata homomorphisms problem on reconfigurable transputer architecture.* Master Research Project, University of Massachusetts Dartmouth, Computer and Information Science Department, January 1994.

[2] B. Mikolajczak. (Ed.), *Algebraic and Structural Automata Theory.* Annals of Discrete Mathematics, North-Holland Publishing Company, vol.44 (1991).

[13] Computer and Information Science Department, University of Massachusetts Dartmouth, N.Dartmouth, MA 02747, bolek@cis.umassd.edu

Parallel Algorithms for the Assignment Problem – An Experimental Evaluation of Three Distributed Algorithms

Christian Schütt[14]
Jens Clausen[15]

Parallel processing is one way of increasing the computational power available when solving large combinatorial problems. Parallel systems consisting of a number of tigthly coupled powerful processors are used extensively. In such systems, no shared memory is available for data sharing.

When used in the implementation of search-based algorithms such as Branch-and-Bound, the lack of shared memory resprsents no significant problem. Implementing polynomially bounded algorithms, however presents difficulties with regards to processors utilization. It is difficult to design algorithms, which scale well when the number of processors in the system is increased and at the same time are comparable with the fastest sequential algorithms with respect to measured running times.

In this work we have studied three different distributed algorithms for the assignment problem: one based on the traditional Hungarian method [3], one based on the shortest augmenting path variant hereof [1], and one based on the auction algorithm [2]. The algorithms have been implemented on a distributed system with 16 i860-processors each with 16 MB memory. Experiments have been made with a range of problems, the largest being dense problems with cost matrix of dimension 1500 and cost coefficient range from 1 to 1000000. We present the computational results obtained and discuss the difficulties in the algorithmic design caused by the lack of shared memory.

References

[1] E.Balas, D.Miller, J.Pekny, and P.Toth, "A Parallel Shortest Augmenting Path Algorithm for the Assignment Problem", J. ACM 38, 1991.

[2] D.P.Bertsekas and D.A.Castanon, "Parallel Synchronous and Asynchronous Immplementatiuons of the Auction Algorithm", Parallel Computations 17, 1991.

[3] S.Storøy and T.Sørevik, "A SIMD, Fine-grained, Parallel Algorithm for the Dense Linear Assignment Problem, unpublished manuscript, 1992.

[14]DIKU, Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK 2100 Copenhagen Ø Denmark

[15]DIKU, Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK 2100 Copenhagen Ø Denmark, clausen@diku.dk

New Anticipatory and Scalable Load Balancing Strategies for Parallel Best First Search

Shantanu Dutt [16]
Nihar R. Mahapatra [17]

In this talk, we present new load balancing strategies for scalable high-performance parallel best-first search (BFS) algorithms suitable for distributed-memory machines. In parallel BFS, inefficiencies such as processor starvation and search of non-essential spaces (search spaces not explored by the sequential algorithm) grow with the number of processors P used, thus restricting its scalability. To alleviate this effect, we propose a novel parallel startup phase and an efficient dynamic load balancing strategy called the quality equalizing (QE) strategy. Our new parallel startup scheme executes optimally in $\Theta(\log P)$ time and, in addition, achieves good initial load balance. The QE strategy possesses certain unique quantitative and qualitative load balancing properties that allow it to significantly reduce starvation and non-essential work. Furthermore, it enables processors to request work ahead of time by "anticipating" that they will run out of work soon, and thereby compensates for latency in work transfer. Consequently, we obtain a highly scalable parallel BFS algorithm with an almost-linear speedup. The startup and load balancing schemes were employed in parallel BFS algorithms to solve the Traveling Salesman Problem on an nCUBE2 hypercube multicomputer. The QE strategy yields speedup improvements of about 20-185% and 15-120% at low and intermediate work densities (the ratio of the problem size to P), respectively, over three well known load balancing methods—the round-robin (RR), the random communication (RC) and the neighborhood averaging (NA) strategies. The average speedup observed on 1024 processors (for a particular problem size) is about 985, representing an impressive efficiency of 0.96. Finally, we analyze and empirically evaluate the scalability of parallel BFS algorithms in terms of the isoefficiency metric. Our analysis gives (1) a $\Theta(P \log P)$ lower bound on the isoefficiency function of any parallel BFS algorithm, and (2) a general expression for the upper bound on the isoefficiency function of our parallel BFS algorithm using the QE strategy on any topology—for the hypercube and 2-D mesh architectures the upper bounds on the isoefficiency function are found to be $\Theta(P \log^2 P)$ and $\Theta(P\sqrt{P})$, respectively. Experimental results validate our analysis, and show that parallel BFS using the QE load balancing strategy has better scalability than when using the RR, RC or NA strategies. We also find that the QE strategy is equally effective across different node-cost distributions of the search space; it should thus prove very useful to many applications.

[16] University of Minnesota, Department of Electrical Engineering, 200 Union St. SE, Minneapolis, MN 55455
[17] University of Minnesota, Department of Electrical Engineering, 200 Union St. SE, Minneapolis, MN 55455

Parallel Branch-and-Bound Techniques for General Mixed Integer Programming on the CM-5

Jonathan Eckstein[18]

This talk discusses the implementation of a classical general mixed integer programming (MIP) branch-and-bound method on the CM-5 family of distributed-memory MIMD multiprocessors. Computational results indicate that efficient parallel solution of many "real-world" MIP's is now a reality. The most interesting design issues relate to the degree of centralization in search control and data storage; the code is capable of centralized "master-slave," fully decentralized, or partially centralized decision making. A "quasi-distributed" active-message storage scheme for subproblem nodes makes fully or partially centralized control of the search process practical even on fairly large configurations. For distributed search control, the implementation uses an asynchronous variant of a traditionally "SIMD" load-balancing algorithm, in conjunction with the randomized scheme of Karp and Zhang. A distributed scheme for storing global heuristic "pseudo-cost" information is also presented.

[18]Mathematical Sciences Research Group, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, `eckstein@think.com`

A Multistage Approach to Scheduling Task Graphs and its Impact to Parallel Numerical Computing [19]

Apostolos Gerasoulis[20]

The scheduling of parallel computation has received considerable attention recently. Many new algorithms have been proposed, especially for static scheduling. In this talk we discuss a multistage approach to scheduling with communication. In the first step, we schedule without any limitations on the number of processors and assume a unit distance between processors. We call this step the clustering step and a very efficient algorithm, the Dominant Sequence Clustering, is introduced. In the second step, if the clusters from the first step are greater than the available number processors, a load balancing heuristic is used, assuming again unit distance between processors. In the third step we take processor distances into consideration and the clusters are mapped to the actual physical architecture so that the total communication volume between clusters is minimized.

A complete software system named PYRROS has been built, based on this multistage scheduling approach, to map task computation and generate parallel code. We have tested the system on a large class of structured and unstructured problems on message passing architectures such as nCUBE-2 and INTEL/i860. The applications range from dense matrix solvers, such as Gaussian Elimination, to sparse matrix solvers and particle simulations using hierarchical fast methods. Major improvements in performance have been observed by using scheduling. Other researchers using our algorithms have also observed "unprecedented" speedups for sparse matrix computations on the CM5.

Papers related to PYRROS scheduling algorithms and theory can be obtained via anonymous ftp from cs.rutgers.edu in the directory pub/gerasoulis/reports. PYRROS is also available as an experimental tool for the NCUBE-2S and INTEL/i860 machines.

[19] The work presented here was in part supported by ARPA contract DABT-63-93-C-0064, by the Office of Naval research under grant N000149310114, by a startup fund from University of California at Santa Barbara and by NSF grant DMS-8706122. The content of the information herein does not necessarily reflect the position of the Government and official endorsement should not be inferred. Joint work with Tao Yang, Department of Computer Science, University of California at Santa Barbara (`tyang@cs.ucsb.edu`), and Jia Jiao, Department of Computer Science, Rutgers University (`jjiao@cs.rutgers.edu`).

[20] Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, Email: gerasoulis@cs.rutgers.edu

Parallel Processing for Satisfiability (SAT) Problem[21]

Jun Gu[22]

The satisfiability problem (SAT) is a fundamental problem in mathematical logic, constraint satisfaction, VLSI engineering, and computing theory. Methods to solve the satisfiability problem play an important role in the development of computing theory and systems. Since the past decade, many parallel processing techniques have been developed to solve the satisfiability problem. The goal of this talk is to expose the rapidly growing field, to discuss tradeoff and limitations, and to describe the state-of-the-art techniques, both algorithms and special-purpose architectures, for parallel processing of the satisfiability problem.

[21]This work was supported in part by 1987 and 1988 IEEE/ACM Academic Scholarship Awards and is presently supported in part by NSERC Strategic Grant MEF0045793, NSERC Research Grant OGP0046423, and 1993 ACM Design Automation Scholarship Awards.

[22]Dept. of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta T2N 1N4, Canada, `gu@enel.ucalgary.ca`, (403) 220-4875 (o) (403) 282-6855 (fax)

Parallel Formulation of Interior Point Algorithms

George Karypis[23]
Anshul Gupta[24]
Vipin Kumar[25]

In recent years, interior point algorithms have become successful in solving medium- to large-size linear programming problems. In this paper we describe a highly parallel formulation of the interior point algorithm. A key component of the interior point algorithm is the solution of a sparse system of linear equations using Cholesky factorization. The performance of parallel Cholesky factorization is determined by (a) the communication overhead incurred by the algorithm, and (b) the load imbalance among the processors. In our parallel dual affine algorithm we use our recently developed parallel multifrontal algorithm that has the smallest communication overhead over all parallel algorithms for Cholesky developed to date. The computation imbalance depends on the elimination tree associated with the factorization ordering algorithm. To balance the computation, we implemented and evaluated four different ordering algorithms. Among these algorithms, Kernighan-Lin and spectral nested dissection yields the most balanced elimination trees and greatly increases the amount of parallelism that can be exploited. Our preliminary implementation achieves a speedup as high as 108 on 256-processor nCUBE 2 on moderate-size problems.

[23] karypis@cs.umn.edu, Computer Science Dept, University of Minnesota, Minneapolis
[24] agupta@cs.umn.edu, Computer Science Dept, University of Minnesota, Minneapolis
[25] kumar@cs.umn.edu, Computer Science Dept, University of Minnesota, Minneapolis

Proving Correctness of Balancing Networks

Costas Busch[26]
Marios Mavronicolas[27]

Balancing networks have been recently proposed by Aspnes, Herlihy and Shavit [2] as a new class of distributed, low-contention data structures useful for solving a variety of multi-processor coordination problems that can be expressed as *balancing problems*. In a recent work [3], the present authors developed a mathematical theory of the combinatorial structure of balancing networks.

In this work, a paradigmatic methodology for showing correctness of balancing networks is developed as a direct consequence of this combinatorial theory. This methodology is applied to yield uniform and transparent correctness proofs for most constructions of balancing networks known so far [1, 2, 4]; these proofs significantly improve upon previous ones in terms of length and simplicity. This new use of the combinatorial theory presented in [3], along with its previous uses there in deriving impossibility results and designing verification algorithms for balancing networks, strengthens the evidence that our theory provides the right mathematical framework for a uniform study of balancing networks.

References

[1] E. Aharonson and H. Attiya, "Counting Networks with Arbitrary Fan-Out," in *Proceedings of the 3rd Annual ACM–SIAM Symposium on Discrete Algorithms,* pp. 104–113, January 1992.

[2] J. Aspnes, M. Herlihy and N. Shavit, "Counting Networks and Multi-Processor Coordination," in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing,* pp. 348–358, May 1991.

[3] C. Busch and M. Mavronicolas, "A Combinatorial Treatment of Balancing Networks," Technical Report, Department of Computer Science, University of Cyprus, April 1994. Submitted for publication.

[4] N. Hardavellas, D. Karakos and M. Mavronicolas, "Notes on Sorting and Counting Networks," in *Proceedings of the 7th International Workshop on Distributed Algorithms*

[26] Department of Computer Science, University of Crete, and Institute of Computer Science, FORTH, Heraklion 71110, Greece. E-mail address: mpous@csi.forth.gr

[27] Department of Computer Science, University of Cyprus, Nicosia, Cyprus. E-mail address: mavronic@csi.forth.gr

(WDAG-93), Lecture Notes in Computer Science, Vol. 725, Springer-Verlag, pp. 234–248, September 1993.

Scheduling Acyclic Task Graphs on Distributed Memory Parallel Architectures

Santosh Pande[28]
Kleanthis Psarris[29]

Scheduling directed acyclic task graphs on distributed memory parallel architectures with non-zero communication costs is shown to be an NP-hard problem.

In this work we propose a heuristic algorithm to solve a variant of the problem by investigating the scalability of the schedule length with the available number of processors. We introduce the concept of the threshold of a task which quantifies a trade-off between the schedule-length and the degree of parallelism. The algorithm minimizes either the schedule length or the number of required processors. The schedule generated by the algorithm satisfies:

A feasibility condition which guarantees that the schedule delay of a task from its earliest start time is below the threshold, and

An optimality condition which uses a merit function to decide the best task – processor match for a set of tasks competing for a given processor.

The issue of regeneration of the schedule for a set of distributed memory architectures with different communication costs is also addressed. It is shown that under certain conditions, schedule on the whole or at least part of the graph can be reused for a different architecture by meeting both the *feasibility* and the *optimality* conditions stated above.

[28] Computer Science Department, Ohio University, Athens, OH–45701, (614)–593–1251, pande@ace.cs.ohiou.edu
[29] Computer Science Department, Ohio University, Athens, OH–45701, (614)–593–1249, psarris@ace.cs.ohiou.edu

Load Balancing for Distributed Branch & Bound Algorithms: Experiments and Theory

Reinhard Lüling[30]
Burkhard Monien[31]
Stefan Tschöke[32]

Introduction The load balancing problem is one of the most important questions in the area of parallel and distributed computing and has therefore been studied theoretically and experimentally for a long time. Branch & Bound applications have often been used as benchmark applications for the experimental investigation of load balancing strategies. This is due to the fact that branch & bound principles are used for a large number of real-world applications and that these type of tree-structured computations seem to be hard to parallelize on large scale parallel machines.

Within this paper we will present experimental results on a massively parallel Transputer system, containing 1024 processors and will give some theoretical foundations for the used load balancing strategy.

Experimental Results In contrast to the work presented in previous papers which mainly focus on small or medium sized parallel machines, we present a distributed branch & bound principle which is suitable for large scale parallel machines. The efficiency is demonstrated by solving the weighted vertex cover problem and the symmetric traveling salesman problem on the 1024 node Transputer system of the "Paderborn Center for Parallel Computing, PC^2". Both problems have been studied intensively in the past and therefore a large set of efficient and effective bounding and branching algorithms are available. For a detailed description of the branching and bounding procedures see [1] and [3].

To solve best-first branch & bound problems efficiently on large scale parallel machines one has to minimize idle times, search overhead and control overhead. Methods to solve these three different problems are in some way contradicting, so one has to find good compromises to solve the problems sufficiently.

The aim of our load balancing method is to keep the load of neighbored processors on a nearly

[30]Department of Mathematics and Computer Science, University of Paderborn, Germany, rl@uni-paderborn.de
[31]Department of Mathematics and Computer Science, University of Paderborn, Germany, bm@uni-paderborn.de
[32]Department of Mathematics and Computer Science, University of Paderborn, Germany, sts@uni-paderborn.de

equal level. For this purpose, each processor "knows" the load of his neighbors in the network. In fact a neighbored processor is only informed about load changes which increase a given threshold. It sends some unexplored nodes to a neighbor if its own load is large relatively to that of its neighbors and sends its local load value for work request if it is assumed that it is small. The exchange of load values and the initiation of a load balancing action depends on the actual load change of a processor. All activities are only performed if the local load of a processor increases of decreases by a factor of Δ. This ensures that unnecessary load balancing actions are avoided.

This strategy will guarantee that a processor and its neighbors have nearly the same load and therefore lead to maximal processor saturation and an approximatively equal load level in the whole network. If the load is defined in an appropriate way and processors always send their best (minimal bound) available unexplored nodes, all processors work on nodes in the search tree which have a nearly equal bound. Therefore the search overhead is minimized if the load difference Δ of neighbored processors is small enough. To avoid thrashing situations that are likely to occur if Δ is small and the load situation changes dramatically, we introduce a controller process to modify Δ by feedback of the load balancing activity.

Results of this strategy on up to 1024 processor show that large speedups (up to about 930) can be reached, although the parallel computation time is very small (about 1.5 minutes). This demonstrates the effectivity of the proposed load balancing principle.

Theoretical Results To analyse the load balancing algorithm used for the practical experiments we modify it in a way, that a new load balancing phase is initiated if the number of load packets generated by a processor decreases or increases by a factor $f > 1$. Whenever a processor initiates a load balancing phase, a random neighborhood, containing δ processors is chosen and the load of all $\delta + 1$ processors is equalized. We will show that for a large number of parameters f and δ this algorithm can balance the workload in such a way that the difference between the expected load of any two processors is bounded by a constant independent of the network size and the generation and consumption of workload, thus always achieving a good workload balance. This shows that a purely local load balancing algorithm, working independently of network size and only initiated by local load changes, can balance the workload evenly throughout any network. To analyze the load differences between any two processors, a technique based on the simple model of only one processor able to generate new load packets (one-processor-producer-model) is used. This will be extended to the one-processor-producer-consumer-model and finally to the generation and consumption of load packets by any processor. This technique can also be used for the analysis of other algorithms.

Since the factor between the expected load of any two processors is important for the practicability of an algorithm, we will present the exact values and show that they can be scaled by the parameters of the load balancing algorithm allowing any balancing quality.

The quality of the expected values is proven by analyzing their variation. It is found that variation is small in general and can also be scaled by the load balancing parameters. Some bounds of the expected costs end the analysis.

References

[1] R. Lüling, B. Monien, *Load Balancing for Distributed Branch & Bound Algorithms*, Proc of Int. Parallel Processing Symposium (IPPS), 1992, pp. 543-549

[2] R. Lüling, B. Monien *A Dynamic Distributed Load Balancing Strategy with Provable Good Performance*, Proc. of 5th ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1993, pp. 164-173

[3] R. Lüling, B. Monien, M. Räcke, S. Tschöke, *Efficient Parallelization of a Branch & Bound Algorithm for the Symmetric Traveling Salesman Problem*, European Workshop on Parallel Computing (EWPC), 1992, Barcelona

[4] R. Lüling, B. Monien, F. Ramme, *Load Balancing in Large Networks : A Comparative Study* Proc. of IEEE Symposium on Parallel and Distributed Processing, 1991, pp. 686-689

A Note on Parallel Randomized Algorithms for Searching Problems

Andrea Clementi[33]
Ludek Kucera[34]
José Rolim[35]

Searching plays an important role in combinatorial optimization problems. Many of searching strategies are based on Breadth First Search (BFS). Optimal sequential algorithms are known for BFS, but since this problem is P-complete efficient *parallel* algorithms are unlikely to exist. Indeed the best known polylog-time PRAM-algorithm for BFS uses $M(n)$ processors where $M(n)$ is the minimum number of processors in order to multiply an $n \times n$ matrix.

In this work, we study alternative *randomized parallel* PRAM-algorithm working well in average for BFS. We study several algorithms for dense and sparse graphs. Furthermore, we show statistical results of the performances of this algorithm on the basis of a large number of tests executed on randomly generated instances. Such tests support our theoretical claim that the algorithms work well in average.

[33] Centre Universitaire d'Informatique, Université de Genève, 24 rue Général Dufour, CH 1211 Genève 4, Switzerland
[34] Centre Universitaire d'Informatique, Université de Genève, 24 rue Général Dufour, CH 1211 Genève 4, Switzerland
[35] Centre Universitaire d'Informatique, Université de Genève, 24 rue Général Dufour, CH 1211 Genève 4, Switzerland

Parallelism and Irregular Data Structures

Catherine Roucairol[36]

The aim of this talk is to point out the difficulties coming from the parallelization of algorithms working on irregular data structures. Parallel exact and approximate methods for solving difficult discrete optimization problems typicaly work on that kind of data structures. We will also briefly describe the French project STRATAGEME: a methodology for a massively parallel processing of ununstructured problems in Linear algebra, Image processing, Optimization.

[36] PRiSM, 45 Avenue des Etats-Unis, 78000 VERSAILLES, France, 39-25-40-88 (tel), 39-25-40-57 (fax), `Catherine.Roucairol@prism.uvsq.fr`

Parallel Genetic Algorithms in Network Optimization

Robert R. Meyer[37]

We present parallel genetic algorithms for several classes of *nonconvex multicommodity flow problems* arising from applications in parallel database design and domain decomposition for computer vision and the solution of partial differential equations. These algorithms take advantage of a common geometric theory for these problems, and have proven to be very fast and effective, even for problems with more than one million fixed-charge terms.

The formulation of these combinatorial problems assumes that tasks corresponding to the cells of a given D-dimensional hyperrectangular grid are to be assigned in a load-balanced manner to the processors of a parallel computer. Different objective functions are obtained by modelling various measures of communication required to coordinate the tasks (for example, centralized or neighborhood or toroidal communication). Extensions of the genetic algorithm approach to other types of fixed-charge networks will also be presented.

References

R.J. Chen, R.R. Meyer, and J. Yackel, "A Genetic Algorithm for Diversity Minimization and Its Parallel Implementation", in *Genetic Algorithms: Proceedings of the Fifth International Conference*, 1993.

S. Ghandeharizadeh, R.R. Meyer, G. Schultz, and J. Yackel, "Optimal Balanced Partitions and a Parallel Database Application", *ORSA J. on Computing*, 5, 151-167, 1993.

J. Yackel and R.R. Meyer, "Optimal Tilings for Parallel Database Design", in *Advances in Optimization and Parallel Computing*, P. M. Pardalos (Editor), North Holland, Amsterdam 1992, 293-309.

[37]Computer Sciences Department, University of Wisconsin-Madison, Madison, Wisconsin 53706 USA, rrm@cs.wisc.edu

Serial & Parallel Algorithms for Quadratic Set Packing Problems

J. MacGregor Smith[38]
Kui Xu[39]

The general focus of this paper is on using Vectorization and Parallel Fortran Language extensions for solving Quadratic Set Packing (QSP) Problems. These engineering optimization problems which have the Quadratic Assignment Problem (QAP) as a fundamental foundation are notorious **NP**–Complete problems. The paper includes a mathematical programming formulation of the problem, a description of the underlying branch and bound algorithm, a profile analysis, a description of the Vectorization and Parallel Fortran Language extensions and its implementation under the VM operating system on the IBM 3090 multiprocessor system. In addition an analysis of parallelization strategies for the problems and a discussion of the performance resulting from the vectorization and parallelization of the algorithm are included. It is shown that problems with tree searching operations can be efficiently parallelized within the parallel tasks, obtaining significant speed-ups on the IBM 3090 multi-processor system.

[38]Department of Industrial Engineering and Operations Research, University of Massachusetts, Amherst, Massachusetts 01003, jmsmith@.ecs.umass.edu
[39]Department of Industrial Engineering and Operations Research, University of Massachusetts, Amherst, Massachusetts 01003

Query Optimization and Processing in Parallel Databases

Jaideep Srivastava[40]
Thomas M. Niccum[41]
Bhaskar Himatsingka[42]

Selecting the best plan for executing a given query is the problem of *query optimization.* The focus of query optimization for sequential machines has been on finding query plans which involve the least amount of work, since response time is equivalent to work done in a uniprocessor environment. With the advent of parallel computers and their application to data management, this is no longer true. It may not be the case that the best sequential plan will result in the best parallel plan, since sequential dependencies in certain plans make them inherently less parallelizable. It is thus possible to reduce the response time of a query by selecting a plan which may do more work but is also more parallelizable. I/O has traditionally been a bottleneck for query processing, and the difference in I/O and CPU speeds is increasing with modern technology. Since join processing is very CPU intensive, maximizing overlap between I/O, and CPU resource utlization is desirable. Researchers have looked at *Asynchronous I/O* as a tool to achieve this effect. In this paper we study the problem of parallel join execution on a shared nothing architecture with support for asynchronous I/O, and asynchronous message passing. We examine the tradeoffs among different query execution models and propose a viable alternative. An analytical cost model for the proposed query execution model is developed. Tradeoffs among different query plan structures are explored, and query domains where one structure does better than the other are categorized. Based on this we propose a *heuristic approach* which tries to achieve the best of all worlds, and gives a stable, and good average performance across most domains.

[40]Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
[41]Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
[42]Department of Computer Science, University of Minnesota, Minneapolis, MN 55455

Experiences with a Synchronous Parallel Branch and Bound Algorithm

Per S. Laursen[43]

Implementation of parallel Branch and Bound algorithms employing dynamic load balancing can often be a tedious task. Most approaches require administration of asynchronous communication, and may also employ elaborate protocols for exchange of unfathomed subproblems. In addition hereto, it may take substantial experimentation to tune such algorithms to optimal performance. This may be unimportant from a purely scientific point of view, but from a practical perspective, simpler approaches may be attractive, even though they may exhibit less impressive run–time performance.

We present a simple version of parallel Branch and Bound for MIMD–type parallel computers. The algorithm does employ dynamic load balancing, but certain restrictions are imposed on the communication, rendering it "predictable" to such a degree that most of the usual constructs for communication management can be omitted. Most importantly, the communication is restricted to be strictly local and synchronous.

The resulting algorithm was implemented on a network of 16 Transputers, and applied to test problems from three classes of NP–hard combinatorial optimization problems; Quadratic Assignment, Graph Partitioning and Weighted Vertex Cover. In all three cases, processor utilizations in the range from 0.80 to above 0.95 were observed, indicating that this synchronous algorithm is competitive with previous approaches, even when regarding performance alone. The algorithm was fairly easy to implement; a fact we also consider a positive distinction.

[43]DIKU – Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark, svalle@diku.dk

Asynchronous Distributed Data Structures: Lists and Priority Locks

Theodore Johnson[44]

Distributed data structures are needed for efficient and scalable computations on message passing processors. Since lists are a key part of many useful data structures, we investigate issues in implementing asynchronous distributed lists. As an application of distributed lists, we present a distributed priority lock.

Li and Hudak have presented a distributed synchronization algorithm that maintains a distributed dynamic tree, with the root containing the synchronization token. Since the tree is dynamic, only the processors that actively compete for the token are involved in the synchronization. In addition, Li and Hudak use path compression to guarantee an amortized $O(\log n)$ messages per token request, where n is the number of processors.

A problem with the Li and Hudak algorithm is that it requires $O(n)$ storage per processor, to store the addresses of blocked processes. By using distributed lists, we reduce the storage requirement to $O(\log n)$ bits per processor. The distributed lists allow us to block processes externally instead of internally, so all that needs to be remembered is the head of the list of blocked processors. Once distributed lists are used, it is easy to extend the algorithm to handle priority synchronization.

We will discuss some extensions to the distributed priority lock. Also, we will discuss the application of distributed lists to a distributed dynamic sub-cube allocation algorithm for hypercube processors, based on buddy allocation.

[44]Dept. of CIS, University of Florida, Gainesville, Fl 32611-2024, ted@cis.ufl.edu

ε-Optimal Solutions to Nonlinear 0-1 Integer Programs Using the Subgradient Algorithm

Zhijun Wu[45]

A subgradient algorithm for nonlinear 0-1 integer programming is presented. Given a nonlinear 0-1 integer program, the algorithm generates a sequence of piecewise linear approximations to the objective function, and searches for a solution by successively solving a set of corresponding linear subproblems. We show that the algorithm can achieve an ε-optimal solution to a given problem for an arbitrarily small ε. Application to quadratic 0-1 integer programming is demonstrated. Parallelism can be exploited for both piecewise linear approximations and linear subproblem solves.

[45] Advanced Computing Research Institute, Cornell University

Improved parallel algorithms via approximations of probability distributions

Aravind Srinivasan[46]
Suresh Chari
Pankaj Rohatgi

We present two new techniques for approximating probability distributions. The first is a new method for constructing the small-bias probability spaces introduced by Naor & Naor, which leads to improved parallel algorithms for many problems which can be solved deterministically using small-bias spaces, such as finding large cuts in graphs, set cover, set discrepancy etc. The second is an explicit construction of small probability spaces approximating general independent distributions, which are better than the constructions of Even, Goldreich, Luby, Nisan & Velickovic.

[46]DIMACS and Institute for Advanced Study

ε-Optimal Solutions to Nonlinear 0-1 Integer Programs Using the Subgradient Algorithm

Zhijun Wu[47]

A subgradient algorithm for nonlinear 0-1 integer programming is presented. Given a nonlinear 0-1 integer program, the algorithm generates a sequence of piecewise linear approximations to the objective function, and searches for a solution by successively solving a set of corresponding linear subproblems. We show that the algorithm can achieve an ε-optimal solution to a given problem for an arbitrarily small ε. Application to quadratic 0-1 integer programming is demonstrated. Parallelism can be exploited for both piecewise linear approximations and linear subproblem solves.

[47]Advanced Computing Research Institute, Cornell University