DIMACS TECHNICAL REPORT 93-90

December, 1993

Workshop on
Parallel Algorithms: From Solving Combinatorial Problems to Solving Grand Challenge Problems
November 17-19, 1993

Organizers:

James Flanagan (CAIP-Rutgers)
Yossi Matias (AT&T Bell Laboratories)[1]
Vijaya Ramachandran (U. Texas at Austin)

[1]DIMACS permanent member

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC-91-19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

In the context of the 1993-94 DIMACS Special Year on Massively Parallel Computation, a three-day workshop entitled "Parallel Algorithms: From Solving Combinatorial Problems to Solving Grand Challenge Problems" was held on November 17-19, 1994. The workshop focused on the general area of parallel algorithms. The scope included the study of basic problems in parallel computation on the one hand, and the relevance of parallel computation to various applications, including the so-called Grand Challenge Problems, on the other hand.

The workshop featured 38 speakers, which presented invited presentations, as well as contributed talks.

The workshop took place at the DIMACS Center on the Rutgers University campus in Piscataway, New Jersey. DIMACS is the National Science Foundation Science and Technology Center for Discrete Mathematics and Theoretical Computer Science. Co-organizers for this workshop were:

James Flanagan (CAIP-Rutgers)
Yossi Matias (AT&T Bell Laboratories)
Vijaya Ramachandran (U. Texas at Austin)

This report contains the set of abstracts for the talks presented, and serves as the workshop proceedings.

This workshop is part of the DIMACS Special Year in Massively Parallel Computation, which receives significant support from the New Jersey Commission on Science and Technology as part of its commitment to promote excellence in High Performance Computing and Communication in the State of New Jersey.

Workshop Program

Wednesday, November 17, 1993

8:00	Breakfast
9:20	Diane Souvaine, DIMACS & Rutgers *Welcoming remarks from DIMACS*
9:30	Gary Miller, Carnegie-Mellon *Numeric and Combinatorial Aspects to Parallel Scientific Computation*
10:00	Break
10:30	Richard Cole, NYU *2-D Pattern Matching*
11:00	Uzi Vishkin, Maryland & Tel Aviv *Efficient Labeling of Substrings*
11:30	Pierre Kelsen, U. British Columbia *Constant Time Parallel Indexing of Points in a Triangle*
12:00	Pangfeng Liu, DIMACS *Experiences with Parallel N-body Simulations*
12:30	Lunch
2:00	Victor Pan, CUNY *Efficient Parallel computations in Linear Algebra with Applications*
2:30	Roland Wunderling, Berlin *On the Impact of Communication Latencies on Distributed Sparse LU Factorization*
2:50	Ian Parberry, North Texas U. *Algorithms for Touring Knights*
3:20	Break
3:50	Phil Klein, Brown *A Linear-Processor Polylog-Time Parallel Algorithm for Shortest Paths in Planar Graphs*
4:20	Edith Cohen, AT&T Bell Laboratories *Polylog-Time and Near-Linear Work Approximation Scheme for Undirected Shortest Paths*
4:50	Lin Chen, USC *Graph Isomorphism and Identification Matrices: Parallel Algorithms*
5:30	Wine and Cheese Reception

Thursday, November 18, 1993

8:00 Breakfast

8:50 John Board, Duke
Algorithms for Multipole-Accelerated Force Calculation in Molecular Dynamics

9:30 Vijaya Ramachandran, U. Texas at Austin
Parallel Graph Algorithms: Theory and Implementation

10:00 Break

10:30 Zvi Kedem, NYU
Towards High-Performance Fault-Tolerant Distributed Processing

11:00 Torben Hagerup, Max Planck Institute
Fast Deterministic Compaction and its Applications

11:30 Phil Gibbons, AT&T Bell Laboratories
Efficient Low Contention Parallel Algorithms

12:00 Paul Spirakis, Patras
Paradigms for Fast Parallel Approximations for Problems that are Hard to Parallelize

12:30 Lunch

2:00 Olof Widlund, NYU
Some Recent Results on Schwarz Type Domain Decomposition Algorithms

2:40 Jan Prins, U. North Carolina at CH
The Proteus System for the Development of Parallel Algorithms

3:10 Yuefan Deng, SUNY at Stony Brook
Analysis and Prediction on Parallel Processors of Protein Binding on DNA: Pattern Recognition of Hydrogen Bonds

3:40 Break

4:10 Rajeev Raman, Maryland
Optimal Randomized Parallel Algorithms for Computing the Row Maxima of a Totally Monotone Matrix

4:40 Teresa Przytycka, Odense
Trade-offs in Parallel Computation of Huffman Tree and Concave Least Weight Subsequence

5:10 Vijay Vazirani, IIT Delhi & DIMACS
A Primal-dual RNC Approximation Algorithm for (multi)-Set (multi)-Cover and Covering Integer Programs

5:40 Wine and Cheese Reception

Friday, November 19, 1993

8:00	Breakfast
8:50	Mike Goodrich, Johns Hopkins *Parallel Methods for Computational Geometry*
9:20	Yossi Matias, AT&T Bell Laboratories *Highly Parallel Randomized Algorithms - Some Recent Results*
9:50	Dina Kravets, NJIT *Optimal Hypercube Algorithm for the All-Nearest Smaller Values Problem*
10:10	Break
10:40	Mike Atallah, Purdue *Optimal Parallel Hypercube Algorithms for Polygon Problems*
11:10	Ernst Mayr, Munich *Optimal Tree Contraction on the Hypercube and Related Network*
11:40	David Haglin, Mankato State U. *Evaluating Parallel Approximation Algorithms*
12:00	Jesper Traff, Copenhagen *A Distributed Implementation of an Algorithm for the Maximum Flow Problem*
12:20	Lunch
1:50	Joseph JáJá, Maryland *Efficient Parallel Algorithms for Image Processing*
2:20	Rainer Feldmann, Paderborn *Game Tree Search on Massively Parallel Systems*
2:50	Stefan Tschoke, Paderborn *Efficient Parallelization of a Branch & Bound Algorithm* *for the Symmetric Traveling Salesman Problem*
3:10	Erik Tarnvik, Umea, Sweden *Solving the 0-1 Knapsack Problem on a Distributed Memory Multicomputer*
3:30	Break
4:00	Aravind Srinivasan, Institute for Advanced Study & DIMACS *Improved Parallel Algorithms via Approximating Probability Distributions*
4:30	Per Laursen, Copenhagen *Parallel Simulated Annealing Using Selection and Migration –* *an Approach Inspired by Genetic Algorithms*
4:50	Zvi Galil, Columbia U. & Tel Aviv *From the CRCW-PRAM to the HCUBE via the CREW-PRAM and the* *EREW-PRAM or In the Defense of the PRAM*
5:20	Workshop ends

Numeric and Combinatorial Aspects to Parallel Scientific Computation

Gary L Miller
CMU
glmiller@cs.cmu.edu

The simulation of 3-dimensional problems comprise many of the "grand challenge problems". Problems on the list include climate forecasting and turbulence. These problems require computer simulation of problems over highly unstructured domains. The size of a regular mesh may be exponentially larger than the corresponding quality unstructured mesh for the same problem. For the last several years we have been developing algorithms which will assist researchers in making these simulations on the next generation of parallel machines.

In particular, we have made progress on three major aspects of the problem: formulatization/meshing, partitioning, and solving. We have shown that d-dimensional meshes and their corresponding combinatorial-graphs of size n that arise from the finite element method can be partitioned into two roughly equal size pieces with a boundary of size at most $O(n^{(d-1)/d})$. These results have at least four application in scientific computation: (1) for efficient direct methods to solve the associated linear system. (2) for the placement of these systems on machines with sublinear bisection width such as fat-tree or mesh-connected machines. (3) for the design and construction of meshes. (4) for the construction of preconditioners which are used in iterative system solvers.

The running time of the geometric separator algorithm has been greatly improved by the use of Second Moments. We will discuss our new analysis of the moment method.

It seems that one of the main stumbling blocks to complete automation of finite element method is mesh generation in 3-dimensions. We have been experimenting with an unstructured finite difference like method. We have been using variation of a method called Poisson Darts. This is a randomized method for placing points down in a domain which has previously been used in numeric integration and ray-tracing algorithms. The advantage with this unstructured finite difference method is that we do not need to generate unstructured meshes as used in the finite element method. We will show some numbers from preliminary experimentations with this method. They indicate that this method is competitive with finite element. Our methods should lend itself to problems which need a moving or changing mesh.

We have also begun analyzing and experimenting with a combinatorial approach to finding preconditioner to improve convergence rates for iterative linear system solvers. In particular we have been considering weighted trees as preconditioners. We are using the separator structure to guide the construction.

Joint work with Stan Eisenstat, Steve Guattery, Keith Gremban, Dafna Talmor, Shang-Hua Teng, Bill Thurston, and Steven Vavasis.

2-D Pattern Matching

Richard Cole
New York University
cole@cs.nyu.edu

The input for the 2-D pattern matching problem comprises a preprocessed square text of size n x n and a square pattern of size m x m, both drawn over the same alphabet Σ. The problem is to find every occurrence of the pattern in the text.

In this talk a linear work constant time PRAM algorithm is described. Algorithms for the preprocessing of the text are also outlined; the latter also perform linear work but run in $\Theta(\log\log m)$ time, which is the best possible.

Joint work with Crochemore, Galil, Gasieniec, Hariharan, Muthukrishnan, Park, Rytter.

Efficient labeling of substrings

Uzi Vishkin
University of Maryland & Tel Aviv University
vishkin@umiacs.umd.edu

Suffix trees are the main data-structure in string matching algorithmics. There are several serial algorithms for suffix tree construction which run in linear time, but the number of operations in the only parallel algorithm available, due to Apostolico, Iliopoulos, Landau, Schieber and Vishkin, is proportional to $n \log n$. The algorithm is based on labeling substrings, similar to a classical serial algorithm, with the same operations bound, by Karp, Miller and Rosenberg. We show how to break symmetries that occur in the process of assigning labels using the Deterministic Coin Tossing (DCT) technique, by Cole and Vishkin, and thereby reduce the number of labeled substrings to linear.

Joint work with Suleyman Cenk Sahinalp.

Constant Time Parallel Indexing of Points in a Triangle

Pierre Kelsen
University of British Columbia
kelsen@cs.ubc.ca

Consider a triangle whose three vertices are points on the unit grid. Let k denote the number of grid points in the triangle. We describe an *indexing* of the triangle: a bijective mapping from $\{0,\ldots,k-1\}$ to the grid points in the triangle. Computing such a mapping is a fundamental subroutine in fine-grained parallel computation arising in graphics applications such as ray-tracing. We describe a very fast indexing algorithm: after a preprocessing phase requiring time proportional to the number of bits in the vertices of the triangle, a grid point in the triangle can be computed in constant time from its index. The method requires only constant space.

Joint work with Simon Kahan.

Experiences with Parallel N-body simulations[1]

Pangfeng Liu
DIMACS
pangfeng@bellcore.com

N-body simulations are used to study a variety of physical phenomena, the evolution of galaxies for example. While these methods appear to be "embarrassingly parallel," large scale simulations involve irregular and adaptive structures which have been considered inconvenient to implement efficiently.

This talk will describe our ongoing experience on the TMC CM-5 with astrophysical simulations involving millions of bodies. We will describe our programming abstractions, experimental results, and techniques for implementing parallel dynamic adaptive computations. The talk will be self-contained.

Joint work with Sandeep Bhatt (Bellcore).

[1]The work was supported by Thinking Machines Corporation, and grants from ONR, DARPA and NSF.

Efficient Parallel computations in Linear Algebra with Applications

Victor Pan
City University of New York
vpan@lcvax.lehman.cuny.edu

We review some recent progress in designing parallel algorithms for linear algebra computations, focusing on processor efficiency of fast algorithms. Besides the major topic of solving linear systems of equations, computing matrix inverse and determinant, we consider computation of matrix rank and maximal linearly independent subset of a vector set, which has many applications to combinatorial computations.

On the Impact of Communication Latencies on Distributed Sparse LU Factorization

Roland Wunderling
Konrad Zuse-Zentrum fuer Informationstechnik Berlin
wunderling@sc.zib-berlin.de

Solving sparse linear systems Ax=b is a task of pivotal importance, not only in linear programming. In state of the art simplex codes, this task is solved by computing the factorization A=LU, where L and U are lower and upper triangular matrices respectively, and then solving the systems Ly=b and Ux=y by forward and backward substitution.

This talk will deal with parallelizing the factorization part. This has been done efficiently on a network of transputers. However, most current commercially available parallel computers have a ratio of communication latency to time per floating point operation magnitudes larger then 1, making such an algorithm merly impossible.

In this talk, I will show, how the problem of high latencies may be attacked by latency hiding through asynchronous communication, and latency minimizing through exploiting parallelism only at a coarser granularity and message fusion.

The talk will be divided into 5 parts. The first part will briefly sketch our sequential algorithm. In the second part, parallelization opportunities will be shown and a brief survey on their exploitation will be given. After that, our parallel factorization algorithm will be described, with special emphasis on the design decisions due to communication latencies. These include, among others, data distribution and load balancing. Part four is devoted to a performance model for the algorithm, which is used to explicitly show the impact of latencies. In section five, our experimental results on an iPSC/860 are reported. Some conclusions will end the talk.

Algorithms for Touring Knights

Ian Parberry[1]
Department of Computer Sciences
University of North Texas
P.O. Box 13886
Denton, TX 76203-3886
ian@ponder.csci.unt.edu

The effectiveness and efficiency of three algorithms for constructing knight's tours on an $n \times n$ board (for even $n \geq 6$) are compared and contrasted using a combination of experimental and theoretical techniques. The first algorithm is a Hopfield-style neural network recently proposed by Takefuji and Lee. The second is a random walk algorithm that combines two classical techniques due to Euler in 1759 and Warnsdorff in 1823. The third is a new divide-and-conquer algorithm that constructs a knight's tour for all even $n \geq 6$, variants of which can be used to construct quadrisected knight's tours for all even $n \geq 8$, tours that are invariant under a 180 degree rotation for all even $n \geq 8$, and tours that are invariant under a 90 degree rotation for all $n \geq 10$ congruent to 2 modulo 4. The neural network appears to take exponential time. The random walk algorithm also appears to run in exponential time, but is practical for a significantly larger range of n. The divide-and-conquer algorithm runs in $O(n^2)$ (i.e. linear) time on a sequential processor, and is particularly amenable to implementation on many different types of parallel computer: in time $O(n^2/p)$ by a bounded degree network with p processors for all $p = O(n^2/\log n)$, in time $O(n^2/p^2)$ by a $p \times p$ mesh for all $p \leq n^{2/3}$, in $O(1)$ time on an $n \times n$ mesh with multiple CREW buses, and in $O(1)$ time on a CREW PRAM with $O(n^2)$ processors. The sequential algorithm is very simple to implement and can be used to generate tours for reasonably large values of n in a matter of seconds. It can also be used to prove a lower bound of $\Omega(1.1^{n^2})$ on the number of knight's tours on an $n \times n$ board for all even $n \geq 6$, and a stronger lower bound of $\Omega(1.248^{n^2})$ when n is of the form 6×2^k for some $k \in \mathbb{N}$.

[1]Research supported by the National Science Foundation under grant number CCR-9302917, and by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant number F49620-93-1-0100.

A Linear-Processor Polylog-Time Algorithm for Shortest-Paths in Planar Graphs

Philip Klein
Dept. of Computer Science
Brown University
Providence, RI 02912-1910
klein@cs.brown.edu

Computing single-source shortest paths with nonnegative lengths, in addition to being a fundamental problem in its own right, arises frequently in solving other problems. The sequential algorithms for this problem are quite efficient; they require only slightly more than linear time. Unfortunately, we don't know how to solve this problem as efficiently in parallel. Polylog-time algorithms using a polynomial number of processors are known, but the processor bounds are large.

We give an algorithm requiring polylog time and a linear number of processors to solve single-source shortest paths in directed planar graphs. More generally, the algorithm works for any graph provided with a decomposition tree constructed using size-$\sqrt{n}$ separators. Our method also yields a polylog-time algorithm requiring m^2 processors for arbitrary graphs with m edges.

Joint work with Sairam Subramanian.

Polylog-Time and Near-Linear Work Approximation Scheme for Undirected Shortest Paths

Edith Cohen
AT&T Bell Laboratories
Murray Hill, NJ 07974
edith@research.att.com

Shortest paths computations constitute one of the most fundamental network problems. Nonetheless, known parallel shortest-paths algorithms are generally inefficient: they perform significantly more work (product of time and processors) than their sequential counterparts. This gap, known in the literature as the "transitive closure bottleneck," poses a long-standing open problem. Our main result is an $O(mn^{\epsilon_0} + s(m + n^{1+\epsilon_0}))$ work polylog-time randomized algorithm that computes paths within $(1 + O(1/\text{ polylog } n))$ of shortest from s source nodes to all other nodes in weighted undirected networks with n nodes and m edges (for any fixed $\epsilon_0 > 0$). This work bound nearly matches the $\tilde{O}(sm)$ sequential time. In contrast, previous polylog-time algorithms required $\min\{\tilde{O}(n^3), \tilde{O}(m^2)\}$ work (even when $s = 1$), and previous near-linear work algorithms required near-$O(n)$ time. Another result is faster shortest-paths algorithms if accurate distances are required only between "distant" vertices: We obtain an $O((m + sn)n^{\epsilon_0})$ time algorithm that computes paths of weight $(1 + O(1/\text{ polylog } n))\text{dist} + O(w_{\max} \text{ polylog } n)$, where dist is the corresponding distance and $w_{\max}$ is the maximum edge weight. Our chief instrument, which is of independent interest, are efficient constructions of sparse *hop sets*. A (d, ϵ)-hop set of a network $G = (V, E)$ is a set E^* of new weighted edges such that minimum-weight d-edge paths in $(V, E \cup E*)$ have weight within $(1 + \epsilon)$ of the respective distances in G. We construct hop sets where $\epsilon = O(1/\text{ polylog } n)$ and $d = O(\text{polylog } n)$.

Graph Isomorphism and Identification Matrices: Parallel Algorithms

Lin Chen
Univ. Southern California
linchen@flash.usc.edu

Let M_1 and M_2 be two matrices representing, respectively, two graphs G_1 and G_2 Of a certain class $\mathcal{C}$, according to a certain relation $\mathcal{R}$. Suppose G_1 and G_2 are isomorphic if and only if there exist two permutation matrices P_1 and P_2 such that $M_1 = P_1M_2P_2$ Then the matrices representing the graphs are said to be *identification matrices* for $\mathcal{C}$, with respect to $\mathcal{R}$. Here we exhibit part of the use of identification matrices in studying the graph isomorphism problem, a well-known long-standing open problem. We show that, given two graphs in the form of a certain identification matrix, isomorphism can be tested efficiently in parallel if at least one matrix satisfies the circular 1's property, and more efficiently in parallel if at least one matrix satisfies the consecutive 1's property. Graphs which have identification matrices satisfying the circular 1's property include, not exclusively, Θ circular arc graphs and Γ circular arc graphs. The result presented here substantially broadens the class of graphs for which there are known efficient parallel isomorphism testing algorithms.

Scalable Implementations of Multipole-Accelerated Algorithms for Molecular Dynamics

John A. Board, Jr.
Duke University
Department of Electrical Engineering
Box 90291, Durham NC 27708
jab@ee.duke.edu

We consider efficient, scalable solutions to the long-range force computation problem in molecular dynamics (MD) simulation. Though a large variety of forces act between the individual atoms in a molecular dynamics system, the Coulomb force is perhaps the most troublesome to compute. Straightforward implementation of a Coulomb solver leads to an $\mathcal{O}(N^2)$ summation over all pairwise combinations of the N atoms in a system; this quadratic complexity limits the size of systems that can be simulated to a few tens of thousands of atoms even when special-purpose parallel hardware is employed. Practical MD codes usually truncate the Coulomb interaction at some moderate interaction radius to limit the cost of the Coulomb force evaluation; this can potentially alter the dynamics of the simulated system. Our work permits inclusion of all pair interactions (i.e. no truncation) at a runtime cost which grows linearly in the size of the system. Our programs are based on the Fast Multipole Algorithm (FMA) of Greengard and Rokhlln and on similar multipole-accelerated algorithms. We have implemented the FMA and related multipole codes on a variety of uniprocessors and parallel and distributed computer systems. The serial FMA reduces the complexity of the N-body problem of electrostatics and gravitation to order N while preserving controllable accuracy in the computation. The parallel FMA allows further reductions in simulation times for N-body problems; for uniform distributions of particles in cubic simulation volumes, nearly linear speedup is obtained on small ($\leq$ 32 processors) parallel systems. The approach taken scales easily to 64 processors and has been tested on up to 32 processors. Additional refinements to the multipole methods allow for the multipole series manipulations to be performed in the Fourier domain, further reducing the complexity of our force computations.

Joint work with William J. Blanke, Daniel C. Gray, Ziyad S. Hakura, William D. Elliott (Duke) and James F. Leathrum, Jr. (Old Dominion).

Parallel Graph Algorithms: Theory and Implementation

Vijaya Ramachandran
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
vlr@cs.utexas.edu

The design and analysis of efficient parallel graph algorithms has been an area of extensive study over the past decade. In this talk we will describe a library of parallel graph algorithms that we have implemented, based on PRAM algorithms designed by us and other researchers. The library was implemented on the Maspar MP-1, which is a massively parallel SIMD machine with 16,384 processors, and we provide performance data on our code. Our library is modular and currently contains code for finding connected components, minimum spanning forest, ear decomposition, cut-edges, strong orientation and open car decomposition of an undirected graph. These algorithms work by calling a collection of basic parallel primitives such as prefix sums, list ranking and Euler tour on trees. Due to the modular nature of our code it can be extended in a fairly straightforward way to include other parallel graph algorithms based on ear decomposition or open ear decomposition, or other parallel algorithms that build on the basic parallel primitives we have implemented. Our experience with this implementation suggests that, overall, the PRAM is a good model for developing parallel algorithms.

Joint work with Tsan-sheng Hsu and Nate Dean.

Towards High-Performance Fault-Tolerant Distributed Processing

Zvi Kedem
New York University
kedem@cs.nyu.edu

Methods for producing robust programs executing correctly and efficiently on fault-prone asynchronous computing systems are presented. The methods can be used to automatically transform given application programs written for ideal perfect synchronous machines into robust programs executing dependably on realistic machines. The robust programs will contain embedded software modules. During the execution, these modules will monitor the computation's progress and by employing thread scheduling algorithms will dynamically adapt the computation to the changing availability of resources.

Fast Deterministic Compaction and its Applications

Torben Hagerup
Max Planck Institute for Computer Science
Germany
torben@mpi-sb.mpg.de

The lower bound of Beame and Håstad seemed to quench all hope of really fast parallel algorithms for such fundamental problems as compaction. Although seldom expressed in so many words, we were effectively operating with the notion of a "parity bottleneck" of $\Theta(\log n/\log\log n)$. It therefore came as a big surprise when Matias and Vishkin demonstrated that approximate compaction can be carried out in $O(\log^* n)$ expected time on a randomized CRCW PRAM. Later it was discovered that approximate compaction can in fact be done in $o(\log n/\log\log n)$ time even without the use of randomization. The talk describes the latter result and some of its applications, trying to cover as many ideas and as few details as possible.

Efficient Low Contention Parallel Algorithms

Phillip B. Gibbons
AT&T Bell Laboratories
Murray Hill, NJ 07974
gibbons@research.att.com

We recently introduced the queue-read, queue-write (QRQW) PRAM model [1, 2], which permits concurrent reading and writing, but at a cost proportional to the number of readers/writers to a memory location in a given step. The QRQW reflects the contention properties of most parallel machines more accurately than either the well-studied EREW or CRCW models. Indeed, in machines with non-combining networks, concurrent reads or writes to a location queue up and are serviced one-at-a-time. The QRQW PRAM is strictly more powerful than the EREW PRAM, yet it can be as efficiently emulated with only *logarithmic* slowdown on Valiant's BSP model [3] and on hypercube-type non-combining networks. In contrast, efficient emulations for the CRCW PRAM on such networks are only known with *polynomial* slowdown.

The QRQW PRAM is the first formal complexity model for the design and analysis of low-contention parallel algorithms. We study the impact of the QRQW rules on algorithm design, devising new techniques for low-contention algorithms. Our results include fast and efficient algorithms for computing the OR, leader election, linear compaction, multiple compaction, integer sorting, CRCW simulation, and random permutation, as well as several lower bounds. For example, we have obtained the following results:

- An $\Omega(\sqrt{\log n})$ time separation between the QRQW PRAM and the weaker EREW PRAM.
- An $O(\sqrt{\log n})$ time, linear work w.h.p. QRQW algorithm for generating a random cyclic permutation; this contrasts with the best known $O(\log n)$ time, super-linear work EREW algorithm.
- An $O(\log n)$ time, linear work w.h.p. QRQW algorithm for the multiple compaction problem; in contrast, no $o(n \log n)$ work algorithm is known for the EREW.

Our results demonstrate the advantage of a low-contention model (QRQW) over a zero-contention model (EREW), particularly for randomized algorithms.

Joint work with Yossi Matias (AT&T Bell Laboratories) and Vijaya Ramachandran (University of Texas, Austin).

References

[1] P. B. Gibbons, Y. Matias, and V. Ramachandran. QRQW: Accounting for concurrency in PRAMs and Asynchronous PRAMs. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, March 1993. Revised version.

[2] P. B. Gibbons, Y. Matias, and V. Ramachandran. The QRQW PRAM: Accounting for contention in parallel algorithms. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, January 1994. To appear.

[3] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103-111, 1990.

Paradigms for fast parallel approximations for problems that are hard to parallelise

Paul G. Spirakis
Computer Technology Institute
Patras University
P.O.B. 1122, 26110 Patras
Greece
spirakis@grpatvx1.bitnet

We present here recent and new results on fast parallel approximations to problems that are hard to parallelise. We discuss five paradigms for efficient parallel approximations. (1) The dense graph properties paradigm where a particular algorithm that eliminates vertices of low degrees is shown to produce in NC constant ratio approximations for any graph problem satisfying a certain property of a certain class of extremal graph properties. (2) By suitably modifying the L-reductions of the class Max-SNP to be log-space , we show that any problem in SNP admits constant ratio NC approximations. (3) We discuss the primal-dual method of Luby and Nisan which shows that positive LP can be approximated in NC. (4) We present the scaling down method for problems whose hardness is due to sizes of numbers. We show that general Max Flow admits a fully RNC approximation scheme and an NC approximation scheme. We show the same for Maximum Matching. (5) We overview the method of decomposing in NC a planar graph into k-outerplanar components (for fixed k) and discuss the constant ratio approximations obtained for planar MIS and other planar graph problems by this decomposition.

Joint work with J. Diaz , M. Serna , J. Toran and L. Kirousis.

Some Recent Results on Schwarz Type Domain Decomposition Algorithms

Olof B. Widlund
Courant Institute of Mathematical Sciences
New York, New York
widlund@cs.nyu.edu

Domain decomposition methods appear to offer the best promise for the efficient parallel solution of the large linear systems of equations that arise in finite element and finite difference discretizations of problems of continuum mechanics. There is growing experimental evidence that some of these algorithms map quite well onto relatively loosely coupled parallel computing systems.

A domain decomposition methods was considered by Hermann Amandeus Schwarz as early as 1869, although for entirely different reasons. The classical Schwarz alternating method can be described in terms of subspaces, which are directly related to the subdomains into which the given region has been subdivided, and projections onto these subspaces. Many other domain decomposition methods for elliptic problems can be placed in a framework, which is closely related to this interpretation. The resulting theory has recently proven very successful in the systematic study of other iterative methods for partial differential equations such as multilevel and multigrid algorithms.

As an introduction, the classical block Jacobi-conjugate gradient preconditioner for elliptic problems is considered. If the blocks correspond to subregions of diameter H, and that of the original region is on the order of 1, then it is easily shown that the number of iterations must be of order $1/H$. This corresponds to a condition number of order $1/H^2$. This simple preconditioner can be improved, and the condition number decreased to $C(1 + H/h)$, by augmenting it by a simple coarse problem. Here, h is the mesh size of the finite element discretization of the given elliptic problem. It is also shown how the rate of convergence can be greatly enhanced by introducing overlap between the subregions.

This will be followed by a discussion of some numerical experiments that show that these simple algorithms are quite useful even for very difficult elliptic problems.

If time allows, there will also be a short discussion of a different type of domain decomposition algorithms, in particular one due to Barry Smith.

The Proteus System for the Development of Parallel Algorithms

Jan F. Prins
University of North Carolina at Chapel Hill
prins@cs.unc.edu

Fundamental variations in the organization and performance characteristics of current parallel computers give rise to a complex design space in which significant trade-offs exist in the development of practical parallel algorithms. Current parallel programming languages are too low-level and often too architecture-specific to support extensive exploration of this design space. On the other hand, highly abstracted models of parallel computation are often too high-level to express and analyze the performance implications of design changes.

The Proteus system consists of a high-level parallel programming language named Proteus and a program transformation system that translates suitably restricted Proteus programs to low-level codes directly executable on parallel machines. Using the Proteus language, designs for parallel programs can be concisely expressed and refined over a broad spectrum of detail, and, through the use of a Proteus interpreter implementing the parallel semantics sequentially, executed and evaluated. Promising designs can be further refined and translated to executable code using the transformation engine. Taken together, these components support an exploratory or prototyping model of parallel program development.

In this talk, the Proteus language and the translation of data-parallel Proteus programs to a low-level vector-model programming notation will be described. The exploration of decomposition strategies for the 3D adaptive fast-multipole algorithm for N-body calculations will be used to illustrate the development process supported by the Proteus system.

(The papers below are available via anonymous ftp to cs.duke.edu and are located in directory /pub/proteus/reports.)

Joint work with Peter Mills, Lars Nyland and John Reif.

References

[1] Lars Nyland, Jan Prins, and John Reif. A Data-Parallel Implementation of the Adaptive Fast Multipole Algorithms. *In Proc. 2nd Symp. on Issues and Obstacles in the Practical*

Implementation of Parallel Algorithms and the Use of Parallel Machines (DAGS93), Dartmouth College, Hanover, NH, June 1993.

[2] Jan Prins and D. Palmer. Transforming High-Level Data-Parallel Programs into Vector Operations. In *Proc. 4th ACM Symp. on Principles and Practice of Parallel Programming*, San Diego, Ca, May 1993.

[3] Peter Mills, Jan Prins, and John Reif. Rate Control as a Language Construct for Parallel and Distributed Programming. In *Proc. of IEEE Workshop on Parallel and Distributed Real-Time Systems (IPPS'93)*, Newport Beach, CA, April 1993.

[4] Peter Mills, Lars Nyland, Jan Prins, and John Reif. Prototyping High-performance Parallel Computing Applications in Proteus. In *Proc. 1992 DARPA Software Technology Conference*, Meridian Press, 1992.

Analysis and Prediction on Parallel Processors of Protein Binding on DNA: Pattern Recognition of Hydrogen Bonds

Yuefan Deng
Center for Scientific Computing
SUNY at Stony Brook
Stony Brook, NY 11794-3600
Yuefan.Deng@sunysb.edu

The theme of this presentation is specificity, or DNA sequence selection, for protein binding on DNA. We introduce a simplified model for the binding energy based on hydrogen bonds alone. Based on this model, we present an optimized algorithm for geometric pattern recognition. The algorithm is implemented for use on parallel supercomputers. All applicable DNA-protein complexes in the BNL data bank with full atomic coordinates are analyzed by this algorithm, with satisfactory results in all cases.

We conclude that the algorithm may be a useful screening tool for assessing DNA pattern selection by DNA-binding proteins and for efficient solution of the docking problem. We also conclude that hydrogen bonds appear to play significant role in binding specificity in all cases analyzed. The algorithm is also useful as a solution or preprocessing partial solution to the docking problem, as it provides a rapid estimation of the "best" relative alignment of two molecules in a complex.

Energy minimization, even in this simplified model based on the geometry of hydrogen bonds, is complicated by an enormous number of local minima. These are understood combinatorially and screened efficiently using a geometric approach to match patterns approximately, based on a square well potential. The second part of the algorithm is a closed form solution for minimization based on a quadratic potential. A Monte Carlo method using a modified Lennard-Jones potential, as a third step, is used to prioritize further the best ranked approximately matched patterns.

Joint work with G. Campbell, M. Eisenberg, J. Glimm, Y. Wang, and Q. Yu.

Optimal Randomized Parallel Algorithms for Computing the Row Maxima of a Totally Monotone Matrix

Rajeev Raman
University of Maryland
raman@umiacs.umd.edu

We consider the problem of finding the row maxima of a $n \times n$ *totally monotone* matrix, a problem with many applications to geometric and combinatorial problems. Although this problem can be solved in $O(n)$ time sequentially, all known poly-log time parallel algorithms for this problem have a time-processor product of $O(n \log n / \log \log n)$ or worse. Finding work-optimal parallel algorithms for this problem has been a relatively long-standing open problem.

We give randomized algorithms for this problem which run in $O(\log n)$ time and $O(\log \log n)$ time on the EREW and CRCW PRAMs respectively, and have a time-processor product of $O(n)$. The run-times are the best possible, even for randomized parallel algorithms.

Joint work with Uzi Vishkin.

Trade-offs in Parallel Computation of Huffman Tree and Concave Least Weight Subsequence

Teresa M. Przytycka
Department of Math. and Computer Science
Odense University
DK-5230 Odense M, Denmark
przytyck@imada.ou.dk

During the last decade several important problems have been placed in the NC class. However, for many such problems the work function of the corresponding NC algorithm is greater by a polynomial factor than the time complexity of the best known sequential algorithm for the problem. This makes such algorithms unattractive for small (which may even mean linear) number of processors. This observation is also true for the previously known algorithms for the Concave Least Weight Subsequence (CLWS) problem - a problem to which many other interesting problems (including the classic Huffman tree problem) can be reduced.

One way of dealing with the above problem is to search for an almost optimal solution rather than for the optimal one. This leads to the familiar trade-off: accuracy versus total work. Such a trad-off was examined in [1] where a family of parallel NC algorithms that construct almost optimal solutions to the Huffman Tree problem was presented. Among other, there woas given an $O(k \log n \log^* n)$-time n processors algorithm that approximates k the Huffman tree with error bounded by $1/n^k$.

An alternative approach is to develop algorithms, that exhibit a certain work-time trade-off: given a parameter p the algorithm performs $O(w(n,p))$ work, in $t(n,p)$ time, where the $t(n,1) = \tilde{O}(w(n,1))$, and for a given $n, t(n,p)$ decreases with the value of p while $w(n,p)$ can increase with the value of p. The first step in this direction was taken in [2] where an $O(\sqrt{n} \log n)$-time n-processor algorithm to the Concave Least Weight Subsequence problem was given. In this talk, we give parallel algorithms for Huffman Tree problem and the CLWS problem which exhibit the work-time trade-off defined above. Namely, we show an algorithm which a parameter p, solves the CLWS problem in $O(\frac{n}{\sqrt{p}} \log^2 n)$ time using p processors. By the reduction of the Huffman Tree problem to the CLWS problem, we obtain the same complexity bounds for the Huffman Tree problem. However, as we show, for the later problem there exists a simpler (and, in fact, slightly better) algorithm that exhibits a similar trade-off: Namely, for a given parameter $p, p \geq 1$, the algorithm runs in $O(\frac{n}{\sqrt{p \log(p+1)}}(\log^2 p + \log(n - \sqrt{p \log p})))$ time using p processors. In particular, for $p = 1$ our algorithm reduces to the classic sequential algorithm.

Joint work with Christos Levcopoulos (Lund University).

References

[1] D.G. Kirkpatrick and T.M. Przytycka. Parallel construction of binary trees with almost optimal weighted path length, *Proc. 2nd Symp. of Parallel Algorithms and Architectures* (1990).

[2] L.L. Larmore and T.M. Przytycka, Parallel construction of trees with optimal weighted path length, *Proc. 3st Symp. on Parallel Algorithms and Architectures* (1991), pp. 71-81.

[3] Christos Levcopoulos and T.M. Przytycka, A work-time trade-off in parallel computation of Huffman Tree and Concave Least Weight Subsequence, manuscript.

A Primal-dual RNC Approximation Algorithm for (multi)-Set (multi)-Cover

Vijay Vazirani
IIT Delhi & DIMACS
vijay@cs.princeton.edu

We build on the classical greedy sequential set cover algorithm, in the spirit of the primal-dual schema, to obtain simple parallel approximation algorithms for the set cover problem and its generalizations. Our algorithms use randomization, and our randomized voting lemmas may be of independent interest. Fast parallel approximation algorithms were known before for set cover though not for any of its generalizations.

Joint work with Sridhar Rajagopalan.

Parallel Methods for Computational Geometry

Michael T. Goodrich
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
goodrich@cs.jhu.edu

In this talk I will survey general techniques for solving computational geometry problems in parallel, as well as focusing on parallel methods for solving specific problems. Techniques discussed will include geometric divide-and-conquer, parallel data structures, and derandomization. Specific problems addressed will include convex hulls in 2- and 3-dimensions, segment intersection, and some problems from computer graphics.

Highly Parallel Randomized Algorithms – Some Recent Results

Yossi Matias
AT&T Bell Laboratories
Murray Hill, NJ 07974
matias@research.att.com

Recent years have seen a dramatic increase in the number of nearly-constant time parallel algorithms discovered. Paradigms and techniques have been developed to exploit the power of randomization and obtain $O(\log^* n)$ time (w.h.p.) randomized algorithms for quite a few problems, including hashing, generating random permutations, linear approximate compaction, load balancing and interval allocation. The last three problems can be viewed as much relaxed versions of the prefix sums computation-a fundamental problem in parallel computation that cannot be solved faster than $\Theta(\log n/\log\log n)$ time, even when using polynomial number of processors. Recently it was discovered [1] that the prefix sums can be closely approximated in $O(\log^* n)$ time w.h.p., using $n/\log^* n$ processors, which is optimal in terms of both work and running time. The approximate prefix sums are guaranteed to come within a factor of $(1+\epsilon)$ of the values of the true sums in a "consistent fashion", where ϵ is $o(1)$. Some of the ideas behind this algorithm will be discussed.

The log-star randomized algorithms that have been developed for the problems mentioned above run on most of the popular CRCW PRAM models which allow conflict in concurrent write. However, they are not known to run on the COMMON CRCW PRAM, one of the well studied CRCW models in the literature on parallel algorithms. The COMMON model requires processors writing simultaneously into the same memory cell to write the same value only. This restriction, which forbids conflicts in concurrent write, makes it difficult to claim resources at random-an operation which is at the heart of the fast randomized algorithms. We show [2] that in many cases this inherent difficulty can be circumvented and we present $O(\log^* n)$ time (w.h.p.), $n/\log^* n$-processor randomized algorithms on COMMON for the problems of linear approximate compaction, load balancing, and approximate prefix sums.

Joint works with 0. Berkman (King's College), P. Gibbons (AT&T Bell Laboratories), M. Goodrich (Johns Hopkins U.), and U. Vishkin (U. Maryland & Tel Aviv U.).

References

[1] M. T. Goodrich, Y. Matias, and U. Vishkin. Optimal parallel approximation algorithms for prefix sums and integer sorting. In *Proc. of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 1994. To appear.

[2] O. Berkman, P. B. Gibbons, and Y. Matias. On the power of randomization for the Common PRAM (hopping onto the log-star wagon). Manuscript, Nov. 1993.

Optimal Hypercube Algorithm for the All-Nearest Smaller Values Problem

Dina Kravets
New Jersey Institute of Technology
dina@lilac.njit.edu

Given a sequence of n elements, the *All-Nearest Smaller Values* (ANSV) problem is to find, for each element x in the sequence, the nearest element to the left (right) of x that is smaller than x, or to report that no such element exists. Berkman, Schieber, and Vishkin give a CREW-PRAM algorithm for this problem that takes $O(\lg n)$ time using $n/\lg n$ processors. We present a hypercube algorithm for the ANSV problem that runs in $O(\lg n)$ time using n processors. Our algorithm belongs to the class of so-called "normal" hypercube algorithms, and thus achieves the same processor/time bounds on any of the bounded-degree variants of the hypercube (e.g., the butterfly, cube-connected cycles, and shuffle-exchange). We prove that $\Omega(n)$ processors are necessary for any normal hypercube algorithm to solve the ANSV problem in $O(\lg n)$ time. Under a stronger model of hypercube computation, in which each processor is allowed to use all of its edges in a single time-step, we demonstrate that the ANSV problem can be solved in $O(\lg n)$ time using $n/\lg n$ processors. We use our ANSV algorithm to give the first logarithmic-time hypercube algorithm for triangulating a monotone polygon, a routine used in the parallel algorithms for triangulating simple polygons. We also obtain a logarithmic-time hypercube algorithm for constructing a Cartesian tree.

Joint work with Greg Plaxton.

Optimal Parallel Hypercube Algorithms for Polygon Problems

Mikhail J. Atallah
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
mja@cs.purdue.edu

We present parallel techniques on hypercubes for solving optimally a class of polygon problems. We thus obtain optimal $O(\log n)$-time, n-processor hypercube algorithms for the problems of computing the portions of an n-vertex simple polygonal chain C that are visible from a given source point, computing the convex hull of C, testing an n-vertex simple polygon P for monotonicity, and other related problems as well. Previously it was not known how to achieve these complexity bounds on hypercubes, one of the main difficulties being that there is no known optimal sorting hypercube algorithm that achieves these bounds. In fact these are the first optimal geometric hypercube algorithms that do not assume that the input is given already sorted by x or y coordinates. The hypercube model we use is the standard one, with $O(1)$ local memory per processor, and with one-port communication.

Joint work with Danny Z. Chen (University of Notre Dame).

Optimal Tree Contraction on the Hypercube and Related Networks

Ernst W. Mayr
Institut für Informatik
Technische Universität, München
mayr@informatik.tu-muenchen.de

An optimal tree contraction algorithm for the boolean hypercube and the constant degree hypercubic networks, such as the shuffle exchange or the butterfly network, is presented. The algorithm is based on novel routing techniques and, for certain small subtrees, simulates optimal PRAM algorithms. For trees of size n, stored on a p processor hypercube in in-order, the running time of the algorithm is $O(\frac{n}{p}\log p)$. The resulting speed-up of $O(p/\log p)$ is optimal due to logarithmic communication overhead, as shown by a corresponding lower bound.

Joint work with Ralph Werchner, Fachbereich Informatik, J.W. Goethe-Universität, Frankfurt am Main, werchner@informatik.uni-frankfurt.de

Evaluating Parallel Approximation Algorithms

David J. Haglin
Computer and Information Sciences Department
Mankato State University
Mankato, MN 56002
Voice: (507) 389-2968
FAX: (507) 389-6376
haglin@epsilon.cs.mankato.msus.edu

Polynomial-time approximation algorithms have been widely studied, especially for $\mathcal{NP}$-complete problems. Many complexity classes have been developed that are based on how easily a problem can be approximated. There are problems that do not have $(1-\epsilon)$- approximation algorithms, for some fixed ratio ϵ, unless $\mathcal{P} = \mathcal{NP}$. Other problems have $(1-\epsilon)$-approximation algorithms, for $O < (1-\epsilon) \leq r$, but not for $(1-\epsilon) > r$ (unless $\mathcal{P} = \mathcal{NP}$). Here r is the $\mathcal{P}$ - *optimal* value for the problem. Still other problems have ϵ-approximation schemes.

Bringing these ideas to the parallel processing realm requires some additional concepts to fully capture the essence of approximation algorithms. In the serial realm, an approximation scheme trades running time for relative error. In the parallel realm, there is more than one resource that may be increased to obtain a closer approximation: running time and number of processors. We define a *two-way* trade-off as an algorithm that uses more running time and processors to find a closer approximation. A *three-way* trade-off is an algorithm that uses either more running time or more processors to obtain a closer approximation.

We investigate the deterministic maximum matching problem which exhibits an interesting phenomenon showing the different nature between the serial and parallel approximation realms. We present some matching results for bounded degree graphs that improve upon recent approximation schemes. We show how to find a *maximal* matching in a bipartite graph with highest degree Δ in $O(\Delta)$ time using $O(n)$ processors, which leads to an $O(1/(1-\epsilon)^2 \cdot \log n)$ time, $O(n)$ processor algorithm for finding a matching whose relative error is ϵ. This observation applied to bounded degree (general) graphs can improve upon the resource trade-off of a recent $\mathcal{NC}$-approximation scheme for general graphs.

Distributed Implementation of an Algorithm for the Maximum Flow Problem

Jesper Larsson Träff
DIKU - Department of Computer Science
University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen Ø
Denmark
traff@rimfaxe.diku.dk

We discuss an implementation of the maximum flow algorithm of Shiloach and Vishkin[1] on a distributed system without shared memory. This algorithm is a clever variation of Karzanov's algorithm, and possesses features which make it an interesting candidate for distributed implementation. First, it is possible to arrange for many processors to perform flow operations (flow push and return) on different vertices simultaneously. Second, in contrast to Karzanov's algorithm, flow returns may be performed from vertices at different levels in the layered network simultaneously, which eliminates the need for expensive global synchronization of the flow return operations.

The talk will outline the sequential algorithm and concentrate on implementation issues which are relevant regardless of the specifics of the distributed system at hand: distribution of data structures (graph, queue and stacks) and reduction of communication volume. For each layered network the algorithm proceeds in phases, in each balancing (flow push followed by flow return) a set of vertices which have flow excess. Each phase corresponds to a pass over a local queue, after which updates must be exchanged between all processors. With a *static* distribution of the graph empirical speed-up can be expected, although it is not possible to guarantee speed-up in the worst case. A new heuristic for "increasing parallelism" by creating more vertices which will appear in the queue during each phase will also be presented: for each vertex in the layered network the heuristic maintains a *potential flow* which is an upper bound on the flow that the vertex can transport to the sink. Potential flows serve to make pushes more precise, in particular to eliminate (some) pushes that cannot give rise to flow augmentations, thus reducing the number of flow return operations and as a consequence the number of phases.

Results from (preliminary) experiments with the distributed algorithm on a small transputer system without as well as with the potential flow heuristic will be presented and discussed. While it is possible to achieve speed-up for smaller, dense graphs, a major problem in the basic implementation is the large fraction of the total time spent on synchronization and exchange of messages. In order to achieve speed-up when going beyond a few processors, means have to be found to hide synchronization time, possibly by making the algorithm more asynchronous. Furthermore, if the overall number of phases could be reduced, the time spent on synchronization would decrease. Whereas the first point is mainly implement ational, the *potential flow heuristic* attacks the latter problem. Indeed, with this heuristic speed-up of about 4 on 16 processors has been consistently achieved for the types of graphs used.

[1] Yossi Shiloach and Uzi Vishkin. An $O(n^2 \log n)$ parallel MAX-FLOW algorithm. *Journal of Algorithms*, 3:128-146,1982.

Preliminary experimental results

Below results from experiments with two different graphs are shown in order to illustrate problems and promises of the implementation. The implementation has been done in OCCAM, experimentation on a 16 processor transputer system (T800) with randomly generated graphs[2] with the following characteristics:

n	degree	layers	phases	vertices	$\|f\|$
200	99	4	62/16	1209/1077	469689
2000	8-12	5	9937/81	27980/18007	50885

All edges have randomly generated integer costs in the interval [1, 10000]. Column "layers" lists the number of layered networks that have to be constructed to find a maximum flow, and "phases" the total number of phases required without/with the potential flow heuristic. For the larger, sparse graph the difference is dramatic, and very significant since each phase entails a synchronization step for the distributed algorithm. Column "vertices" lists the total number of unbalanced vertices considered without/with the heuristic.

The next tables list the actual running times achieved with 1, 2, 4, 8, and 16 processors. Column "t" lists the total time spent without/with the potential flow heuristic, "s" the time spent on synchronization. As can be seen, keeping the number of phases as low as possible is essential for achieving speed-up.

proc.	1	2		4	
n	t	t	s	t	s
200	9.5/15.2	4.2/5.9	0.6/0.7	3.7/3.6	1.3/0.7
2000	28.2/28.1	36.6/17.3	21.3/1.3	65.5/11.3	54.7/1.4

proc.	8		16	
n	t	s	t	s
200	3.0/3.3	1.9/1.1	5.7/3.2	4.8/1.7
2000	150.7/7.7	140.5/1.8	387.5/6.7	374.4/3.2

[2] For each vertex an average degree is chosen and so many edges are generated. Since this has a tendency to lead to networks where the minimum cut lies either at the source or the sink, other ways of generating test graphs must be chosen for the detailed experiments.

Efficient Parallel Algorithms for Image Processing

Joseph JáJá
Dept. of Electrical Engineering
University of Maryland
joseph@src.umd.edu

We present efficient algorithms for low and intermediate level image processing on the scan line array processor, a SIMD machine designed for real-time video and image processing. For low level processing, we present algorithms that run in real-time. By real-time, we mean that, if the required processing is based on neighborhoods of size $m \times m$, then the output lines are generated at a rate of $O(m)$ operations per line and a latency of $O(m)$ scan lines. For intermediate level processing, we present optimal or close to optimal algorithms for translation, histogram computation, scaling, rotation, connected components, and convex hulls of multiple figures. The algorithms for connected components and convex hulls are significantly simpler and easier to implement than those already reported in the literature for linear arrays.

Game Tree Search on Massively Parallel Systems

Rainer Feldmann
University of Paderborn
Germany
chess@uni-paderborn.de

Tree search algorithms play an important role in many applications in the field of artificial intelligence. For instance, theorem provers, expert systems, robot control systems and game playing programs contain tree searching algorithms as their basic part. Tree searching is used whenever decisions must be made that are based on complex knowledge, which cannot be implemented directly on a machine. Came playing programs provide an excellent test bed for search algorithms.

For the problem to search game trees a very efficient sequential algorithm, the so called *alphabeta* - algorithm (or the Scout variant) is known. The *alphabeta* - algorithm traverses the game tree in a depth first manner. It allows to cut off large parts of the right subtrees of the game tree by using information already computed in the left subtrees. Thus, the *alphabeta* - algorithm was believed to be inherently sequential. Therefore, the efficiency of most of the parallel game tree search algorithms implemented so far is very poor, even if only a few processors are used. Mainly, there are three types of losses occurring in a parallel game tree search:

1. *Search overhead:*
   A parallel algorithm often may not have available the information which allows to cut off a subtree. Thus, the processors are searching parts of the game tree which cannot influence the final result and which the sequential *alphabeta* - algorithm would not have searched.

2. *Processor work load:*
   A decomposition of the game tree which guarantees a good average work load is difficult, because the sizes of the subtrees are unpredictable in advance. Therefore, any static mapping of subtrees onto processors will result in a poor processor work load.

3. *Communication overhead:*
   Processors have to communicate for two reasons: first to balance the work load and second to broadcast the information about the game tree computed by other processors.

In this paper we present a parallel game tree search algorithm which shows high efficiency even on massively parallel systems. The main concept is a completely dynamic game tree decomposition.

We implemented the "Young Brothers Wait Concept" which keeps the search overhead small but, additionally, does not decrease the processor work load too much. We present an efficient use of a distributed hash table in a distributed memory system, i.e. we simulate global memory in a distributed system. This distributed hash table is used to store the information computed by the processors and make them available to all of them. To achieve a good processor work load we implemented a combination of local, global, and medium range search for work, which allows a good load balancing for small communication costs only.

Experimental results are presented for up to 1024 processors showing a speedup of more than 340. With this, our parallel game tree search algorithm is the most efficient one known so far. It is the basis of our distributed chess program *ZUGZWANG*, which became Vize World Champion at the World Computer Chess Championships, 1992, in Madrid, Spain.

Joint work with Peter Mysliwietz and Burkhard Monien.

Efficient Parallelization of a Branch & Bound Algorithm for the Symmetric Traveling Salesman Problem

S. Tschöke
Department of Mathematics and Computer Science
University of Paderborn
Germany
sts@uni-paderborn.de

The traveling salesman problem is one of the most studied NP-complete problems of combinatorial optimization. Our interest was to see whether parallelism can be fully exploited in solving TSPs.

We present a general method to parallelize branch & bound algorithms on transputer networks and an application of this method to solve symmetric traveling salesman problems by a best first branch & bound strategy. The parallelization of the branch & bound algorithm is fully distributed. Every processor performs the same algorithm but on a different part of the solution tree. The algorithm running on each processor consists of three parts, a local heap management, a communication process and the sequential branch & bound part to generate and evaluate subproblems.

To distribute subproblems among the processors and to keep all local heaps on a nearly equal level we use a nearest-neighbor load-balancing strategy, i.e. subproblems and load informations are only sent to direct neighbors in the processor network. No global communication is needed [2]. With this strategy we minimize idle times and search overhead (only necessary parts of the solution tree are explored).

On the one hand we wanted to use an efficient sequential algorithm to solve the traveling salesman problem, but on the other hand we also wanted to provide a general method of parallelization applicable for other problems of combinatorial optimization. To achieve this we do not parallelize the sequential computation of the subproblems. The current best approach to solve TSPs is the branch & cut algorithm by Padberg and Rinaldi [4]. In order to parallelize this efficiently one would have to parallelize the sequential computation of the subproblems because of the huge effort spent on a single subproblem in comparison to the number of subproblems. Thus we chose the improved 1-tree relaxation branch & bound algorithm by Volgenant and Jonker [6] based on the lagrangean approach introduced by Held and Karp [1].

On TSP instances with many subproblems we achieve an almost linear speedup. At present we can efficiently solve euclidean symmetric traveling salesman problems up to a size of 318 cities on a asynchronous multiprocessor network of up to 1024 nodes (T800 transputers). The problem instances are taken from the standard TSP library [5].

We can conclude that the problem of efficient parallelization is solved for branch & bound algorithms with an appropriate ratio between the effort which is spent on a single subproblem and the number of subproblems computed.

The results were presented on the European Workshop on Parallel Computing EWPC 1992 in Barcelona and on the EURO XXII/TIMS XXXI Joint International Conference Operational Research/ Management Science 1992 in Helsinki.

References

[1] M. Held and R.M. Karp, *The traveling salesman problem and minimum spanning trees: part II*, Mathematical Programming 1 (1971), pp. 6-25

[2] R. Lüling, B. Monien, *Load Balancing for distributed Branch and Bound Algorithms*, Proc. of 6th Int. Parallel Processing Symposium 1992, pp. 543-549

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

[4] M. Padberg, G. Rinaldi, *A Branch and Cut Algorithm for the Resolution of Large Scale Symmetric Traveling salesman problems*, SIAM Review 33 (1991), S. 60-100

[5] G. Reinelt, *TSPLIB - A Traveling Salesman Problem Library*, ORSA Journal on Computing 3 (1991), pp. 376-384

[6] T. Volgenant, R. Jonker, *A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation*, European J. Operational Res. 9 (1982), pp. 83-89

Joint work with R. Lüling, B. Monien, and M. Räcke.

Solving the 0-1 Knapsack Problem on a Distributed Memory Multicomputer

Erik Tärnvik
Institute of Information Processing
University of Umeå
S-901 87 Umeå
Sweden
erikt@cs.umu.se

This talk summarizes our observations from using a parallel branch and bound algorithm to solve the 0-1 knapsack problem on a distributed memory multicomputer.

A branch and bound algorithm is essentially a tree search were a bounding function is used to exclude parts of the tree which can be shown not to include an optimal element. In searching the tree, we have used a depth-first search strategy. Searching algorithms of this type are hard to parallelize, since we cannot know beforehand which parts of the tree will be excluded. This means that we can not use a static distribution of the workload. In implementing the algorithm, we have used the Dynamo tool in order to use dynamic load balancing to achieve good parallel efficiency.

The algorithm has been implemented on the iPSC/2 hypercube multicomputer using the Dynamo dynamic load balancing tool. Computational results for different classes of problem instances are presented. The computational results indicate that the complexity of the problem instances determine the amount of parallelism available. For instances of high complexity, the parallel algorithm shows a linear speedup behavior. We conclude the talk by showing how a parallel reduction algorithm can be employed in order to obtain higher parallel performance for instances of low complexity.

Improved Parallel Algorithms via Approximating Probability Distributions

Aravind Srinivasan
Institute for Advanced Study & DIMACS
aravind@math.ias.edu

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; this leads to improved parallel algorithms for certain problems which can be solved deterministically using small-bias spaces, such as set-balancing, finding large cuts in graphs, finding heavy codewords in linear codes, and other problems. 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; this is a further step toward a powerful tool for derandomizing sequential and parallel algorithms.

Joint work with Suresh Chari and Pankaj Rohatgi (Cornell University).

Parallel Simulated Annealing using Selection and Migration–an Approach Inspired by Genetic Algorithms

Per S. Laursen
Department of Computer Science, University of Copenhagen
Universitetsparken 1, 2100 Copenhagen Ø, Denmark
svalle@diku.dk

Introduction. Parallelization of the generic Simulated Annealing algorithm is a difficult task, due to the inherent sequential nature of the algorithm. A popular alternative approach has therefore been to perform a number of independent annealings - typically one on each available processor - augmented by some variant of periodic "coordination" of the annealings. The generic coordination strategy is to periodically gather all current solutions, and pick the best current solution as the new starting point for all of the annealings, which then continue with this best current solution as their new current solution. Parallelization strategies of this kind are usually denoted *division strategies* in the literature.

Alternative Parallelization of Simulated Annealing. The above strategy for selecting a new starting point for the continued annealings can be perceived as a very primitive type of Genetic Algorithm, using an extreme elitist-strategy. The risk of premature convergence due to the hard selection also seems latent.

We have therefore conjectured that a more refined strategy for selection of new starting points may lead to improved performance, in addition to rendering global control of the search unnecessary, the latter being advantageous if the algorithm is to be ported to large distributed systems. We have seeked inspiration for such strategies from certain types of Parallel Genetic Algorithms, more specifically those versions based on the so-called *island model*. In this model, populations of solutions "live" in isolated territories (which could e.g. be individual processors), and can interact with other populations through migration and selection. That is, a solution may at some point migrate to a neighbouring population, and there compete on equal footing with the domestic solutions for a place in the next "generation" of solutions.

This strategy can also be used as a framework for a division-strategy based Parallel Simulated Annealing algorithm: Each processor maintains a number of individuals (solutions), and tries to improve the quality of these by performing Simulated Annealing iterations on each individual. After some fixed number of iterations, a number of individuals migrate to a neighbouring target processor, and subsequently participate in a selection on this target processor. The new generation of individuals is then again submitted to a number of Simulated Annealing iterations, followed by migration to new target processors, etc..

Experiments and Main Results. We have implemented three versions of parallel Simulated Annealing based on division strategies; a version using no coordination at all, a version using "traditional" global coordination, and a version using local coordination based on principles from the island model. All versions were implemented on a 16-processor network consisting of T800 Transputers.

First, the performance of the three versions was compared, using an annealing scheme with a fixed number of iterations, and a fixed coordination intensity. The three versions were applied to QAPs of size 30,40,60,80 and 100. With respect to solution quality, the version using local coordination outperformed the version using no coordination at all, in all five cases. For the smaller test problems, the version using traditional global coordination was outperformed by *both* of the other versions, while it was able to tie the solution quality of the version using local coordination, when applied to. the largest test problems. With respect to running time, it was observed that the local coordination strategy induced only very marginal increases in the running time - as compared to the version using no coordination - while global coordination induced more severe increases.

Second, we varied the coordination intensity, while still retaining the total number of iterations. Here we only applied the three versions to QAPs of size 30 and 100. For the small QAPs, we observed that global coordination was consistently outperformed by local coordination, no matter the choice of coordination intensity. For the large QAPs, the picture was less clear. Still, local coordination succeeded in obtaining the best overall solution quality. Running time was observed to increase roughly linear with coordination intensity, implying substantially prolonged running times for the global coordination version, when using intense coordination. This may however be partly attributed to a non-optimal implementation of the global coordination strategy as such. We do on the other hand not think that a better tuned implementation would alter the conclusions radically.

Third, we de- and increased the total number of interactions. This can be done in two ways; either by varying the number of Simulated Annealing iterations performed between each migration phase - while retaining the number of migration phases - or vice versa. Again, QAPs of size 30 and 100 were used. For the small problems, we observed that global coordination was consistently outperformed by the two other versions, no matter the variation strategy or number of iterations. For the large problems, global coordination on the other hand outperformed the other versions when the total number of iterations was small, while being outperformed again for long annealings. When comparing local coordination to no coordination, it was observed that the no-coordination version needed roughly twice the number of iterations to match the solution quality produced when using local coordination. This could be taken as a semi- quantitative measure of the effectiveness of this strategy.

Conclusions. The results indicate that the proposed local coordination strategy generally outperforms the two other versions, and that the proposed local coordination strategy itself does not inflict any significant increase in running time, while rendering global control of the search unnecessary. Application of the algorithm to other problem types will take place in the near future, to verify whether these observations have more general validity.

This research is financed by grants from the Danish Natural Science Research Council.

From the CRCW-PRAM to the Hypercube via the CREW-PRAM and the EREW-PRAM or In the Defense of the PRAM

Zvi Galil
Columbia University & Tel-Aviv University
galil@cs.columbia.edu

General simulations of powerful versions of PRAM by weaker versions or by an hypercube may incur a loss in time and in optimality. Going through the example of string matching, we identify conditions where such losses are avoidable. Starting with a CRCW-PRAM algorithm, a modified version of a known algorithm, we first eliminate concurrent write, obtaining a CREW-PRAM algorithm, then eliminate concurrent read, obtaining an EREW-PRAM algorithm, and finally implement the latter on the hypercube.

We obtain optimally fast algorithms for all these models that are better than the ones obtained via the general simulations for both the pattern preprocessing and the text search. In all cases but one (text search on the CREW-PRAM), such algorithms were not known before.

Joint work with Artur Czumaj, Leszek Gasieniec and Wojciech Plandowski.

Speaker Index

Co-Author Index