

Combinatorial Parallel and Scientific Computing*

Ali Pinar[†] and Bruce Hendrickson[‡]

1 Introduction

Combinatorial algorithms have long played a pivotal enabling role in many applications of parallel computing. Graph algorithms in particular arise in load balancing, scheduling, mapping and many other aspects of the parallelization of irregular applications. These are still active research areas, mostly due to evolving computational techniques and rapidly changing computational platforms. But the relationship between parallel computing and discrete algorithms is much richer than the mere use of graph algorithms to support the parallelization of traditional scientific computations. Important, emerging areas of science are fundamentally discrete, and they are increasingly reliant on the power of parallel computing. Examples include computational biology, scientific data mining, and network analysis. These applications are changing the relationship between discrete algorithms and parallel computing. In addition to their traditional role as enablers of high performance, combinatorial algorithms are now customers for parallel computing. New parallelization techniques for combinatorial algorithms need to be developed to support these nontraditional scientific approaches.

This chapter will describe some of the many areas of intersection between

*Pinar is also supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract DE-AC03-76SF00098. Hendrickson was funded by the Applied Mathematics Research program, U.S. Department of Energy, Office of Science, and works at Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Department of Energy under contract DE-AC-94AL85000.

[†]High Performance Computing Research Department, Lawrence Berkeley National Laboratory, Berkeley, CA, email: apinar at lbl dot gov.

[‡]Discrete Algorithms and Math Department, Sandia National Laboratories, Albuquerque, NM, email: bah at sandia dot gov.



discrete algorithms and parallel scientific computing. Due to space limitations, this chapter is not a comprehensive survey, but rather an introduction to a diverse set of techniques and applications with a particular emphasis on work presented at the Eleventh SIAM Conference on Parallel Processing for Scientific Computing. Some topics highly relevant to this chapter (e.g. load balancing) are addressed elsewhere in this book, and so we will not discuss them here.

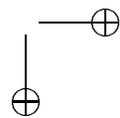
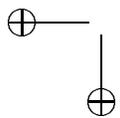
2 Sparse Matrix Computations

Solving systems of sparse linear and nonlinear equations lies at the heart of many scientific computing applications including accelerator modeling, astrophysics, nanoscience, and combustion. Sparse solvers invariably require exploiting the sparsity structure to achieve any of several goals: preserving sparsity during complete/incomplete factorizations, optimizing memory performance, improving the effectiveness of preconditioners, and efficient Hessian and Jacobian construction, among others. The exploitation of sparse structure involves graph algorithms, and is probably the best known example of the role of discrete math in scientific computing.

2.1 Sparse Direct Solvers

Direct methods for solving sparse linear equations are widely used especially for solving ill-conditioned systems such as those arising in fusion studies or interior point methods for optimization. They are also used when high accuracy solutions are needed as with the inversion operator for the shift-and-invert algorithms for eigencomputations, solving coarse grid problems as part of a multigrid solver, and solving subdomains in domain decomposition methods. The sizes of the problems arising in these applications necessitate parallelization, not only for performance, but also for memory limitations. Most direct solvers require one processor to hold the whole matrix for preprocessing steps such as reordering to preserve sparsity during factorization, column/row permutations to avoid or decrease pivoting during numerical factorization, and symbolic factorization, and this requirement to have one processor store the whole matrix is an important bottleneck to scalability. Recent studies have addressed parallelization of these less time consuming parts of sparse direct solvers.

Having large entries on the diagonal at the time of elimination is important for numerical accuracy during LU factorization. The dynamic approach for this problem is to move a large entry to the diagonal at each step during factorization by row and column permutations. However, dynamic pivoting hinders performance significantly. Alternative is the static approach where large entries are permuted to the diagonal a priori. Although somewhat less robust numerically, this static pivoting approach achieves much higher performance. The problem of permuting large entries to the diagonal to reduce or totally avoid pivoting during factorization, can be fruitfully recast as the identification of a heavy, maximum-cardinality matching in the weighted bipartite graph of the matrix. An example is illustrated in Fig. 1. In the bipartite graph of a matrix, each row and each column of the matrix is represented by a vertex. An edge connects a row vertex to a column



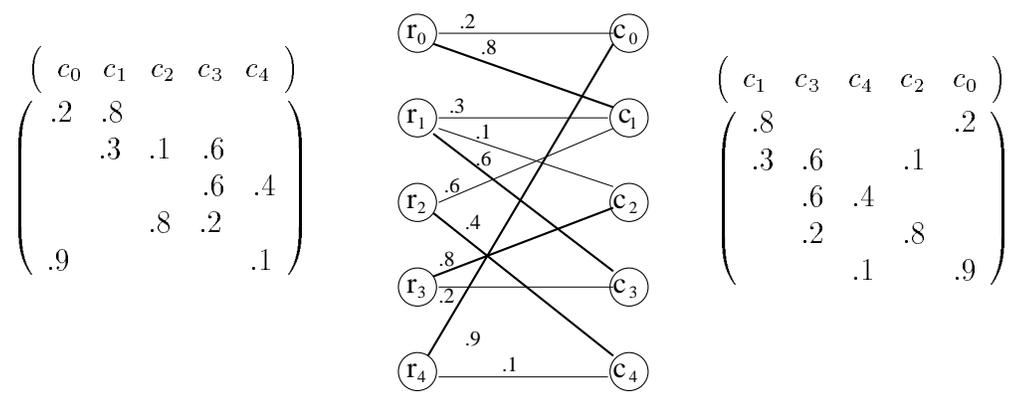


Figure 1. *Permuting large entries to the diagonal. Dark edges in the graph correspond to edges in the matching in the bipartite graph of the matrix on left. Matrix on right is the permuted matrix with respected to the matching where columns are reordered as (mate of the 1st row, mate of the 2nd row, ...).*

vertex if the corresponding matrix entry at this row and column is nonzero, and the weight of the edge is set equal to the absolute value of the matrix entry. A complete matching between rows and columns identifies a reordering of columns or rows of the matrix, in which all the diagonal values are nonzero. Heavier weighted edges in the matching translate to larger values on the diagonal after permutation. Notice that a maximum weighted matching maximizes the sum of absolute values of diagonal entries. By assigning the logarithms of absolute values of entries to edges one can maximize the product of diagonal entries with maximum matching.

While bipartite matching is a well-studied problem in graph theory, designing parallel algorithms that perform well in practice remains as a challenge. Most sequential algorithms for bipartite matching rely on augmenting paths, which is hard to parallelize. Bertsekas' auction algorithm is symbolically similar to Jacobi and Gauss-Seidel algorithms for solving linear systems, and thus more amenable to parallelization. As the name implies Bertsekas' algorithm resembles an auction, where the prices of the columns are gradually increased by buyers (rows) that are not matched. Each row bids on the cheapest column, and the process ends, when all rows are matched to a column. Riedy and Demmel [18] studied the parallel implementation of Bertsekas' auction algorithm. They observed, as in all parallel search algorithms, speedup anomalies with superlinear speedups and slowdowns. Overall, they showed that the auction algorithm serves very well as a distributed memory solver for weighted bipartite matching.

Another important and challenging problem in sparse direct solvers is the development of parallel algorithms for sparsity preserving orderings for Cholesky/LU factorization. The two most widely used serial strategies for sparsity preserving orderings are instantiations of two of the most common algorithmic paradigms in computer science. Minimum degree and its many variants are greedy algorithms, while nested dissection is an example of a divide-and-conquer approach. Nested

dissection is commonly used for parallel orderings since its divide-and-conquer nature has natural parallelism, and subsequent triangular solution operations on the factored matrix grant better efficiency on parallel systems. Nevertheless, parallelizing minimum degree variants remain as an intriguing question, although previous attempts have not been very encouraging [6].

Another component of direct solvers that requires a distributed algorithm is the symbolic factorization phase [7] for sparse Cholesky/LU factorization. Symbolic factorization is performed to determine the sparsity structure of the factored matrix. With the sparsity structure known in advance, the numerical operations can be performed much more quickly. Symbolic factorization takes much less time than numerical factorization, and is often performed sequentially in one processor. A distributed memory algorithm however, is critical due to memory limitations. Grigori et al. have studied this problem and reported promising initial results [7].

A more in depth discussion on Sparse Direct methods can be found in Chapter ?? of this book.

2.2 Decompositions with Colorings

Independent sets and coloring algorithms are also commonly used in sparse matrix computations. A set of vertices is independent if no edge connects any pair of vertices in the set. A coloring is a union of disjoint independent sets that cover all the vertices. The utility of an independent set arises from the observation that none of the vertices in the set depend upon each other, and so operations can be performed on all of them simultaneously. This insight has been exploited in the parallelization of adaptive mesh codes, in parallel preconditioning and in other settings. Algebraic multigrid algorithms use independent sets for coarse grid construction. Partitioning problems that arise in the efficient computation of sparse Jacobian and Hessian matrices can be modeled using variants of the graph coloring problem. The particular coloring problem depends on whether the matrix to be computed is symmetric or nonsymmetric, whether a one-dimensional partition or a two-dimensional partition is to be used, whether a direct or a substitution based evaluation scheme is to be employed, and whether all nonzero matrix entries or only a subset need to be computed. Gebremedhin [5] has developed a unified graph theoretic framework to study the resulting problems, and developed shared memory parallel coloring algorithms to address several of them.

2.3 Preconditioning

Iterative methods for solving linear systems also lead to graph problems, particularly for preconditioning. Incomplete factorization preconditioners make use of many of the same graph ideas employed by sparse direct solvers. Efficient data structures for representing and exploiting the sparsity structure, and reordering methods are all relevant here. Domain decomposition preconditioners rely on good partitions of a global domain into subproblems, and this is commonly addressed by (weighted) graph or hypergraph partitioning. Algebraic multigrid methods make use of graph matchings and independent sets in their construction of coarse grids or smoothers.

Support theory techniques for preconditioning often make use of spanning trees and graph embeddings.

3 Utilizing Computational Infrastructure

Utilization of the underlying computational infrastructure commonly requires combinatorial techniques. Even for applications where problems are modeled with techniques of continuous mathematics, effective utilization of the computational infrastructure requires decomposition of the problem into subproblems and mapping them onto processors, scheduling the tasks to satisfy precedence constraints, designing data structures for maximum uniprocessors performance, and communication algorithms to exchange information among processors. Solution to all these problems require combinatorial techniques.

3.1 Load Balancing

One area where discrete algorithms have made a major impact in parallel scientific computing is partitioning for load balance. The challenge of decomposing an unstructured computation among the processors of a parallel machine can be naturally expressed as a graph (or hypergraph) partitioning problem. New algorithms and effective software for partitioning have been key enablers for parallel unstructured grid computations. Some problems, e.g. particle simulations, are described most naturally in terms of geometry instead of the language of graphs. A variety of geometric partitioning algorithms have been devised for such problems. In addition, space-filling curves and octree methods have been developed to parallelize multipole methods. Research in partitioning algorithms and models continues to be an active area, mostly due to evolving computational platforms and algorithms. For instance with increasing gap between computation and communication speeds, distribution of the communication work has become an important problem. The next generation petaflops architectures are expected to have orders of magnitude more processors. An increased number of processors, along with the increasing gap between processor and network speeds, will expose some of the limitations of the existing approaches. Novel decomposition techniques and interprocessor communication algorithms will be required to cope with these problems. Recent advances in load balancing are discussed in depth in Chapter ?? of this book.

3.2 Memory Performance

The increasing gap between CPU and memory performances argues for the design of new algorithms, data structures, and data reorganization methods to improve locality at memory, cache, and register levels. Combinatorial techniques come to the fore in designing algorithms that exhibit high performance on the deep memory hierarchies on current architectures and on the deeper hierarchies expected on the next generation supercomputers. Cache oblivious algorithms [4], developed in the last few years, hold the promise of delivering high performance for irregular problems while being insensitive to sizes of the multiple caches. Another approach for better

cache utilization is cache aware algorithms [?], where the code is tuned to make the working set fit into the cache (e.g. blocking during dense matrix operations), or repeated operations are performed for the data already in the cache (e.g. extra iterations for stationary point methods), since the subsequent iterations come at a much lower cost when the data is already in the cache.

Performance of sparse matrix computations are often constrained by the memory performance due to the irregular memory access patterns and extra memory indirections needed to exploit sparsity. For sparse matrix-vector multiplication, it is possible to reorder the matrix to improve memory performance. Bandwidth or envelope reduction algorithms have been used to gather nonzeros of the matrix around the diagonal for a more regular access pattern, and thus fewer cache misses. A new more promising method is the blocking techniques that have been used for register reuse, and reducing memory load operations [19, 16, 10]. These techniques represent the sparse matrix as a union of dense submatrices. This requires either replacing some structural zeros with numerical zeros so that all dense submatrices are of uniform size [10], or splitting the matrix into several submatrices so that each submatrix covers blocks of different sizes [19, 16]. Experiments show that notable speedups can be achieved through these blocking techniques, reaching close to the peak processor performances.

3.3 Node Allocation

A recent trend for parallel architectures is computational clusters built of off-the-shelf components. Typically in such systems, communication is slower, but it is possible to build very large clusters, due to easy incrementability. With slow communication, along with large numbers of processors, choosing which set of processors to perform a parallel job becomes a critical task for overall performance both in terms of the response time of individual tasks and system throughput. The problem of choosing a subset of processors to perform a parallel job is studied as the node allocation problem, and the objective is to minimize network contention by assigning jobs to maximize processor locality. Bender et al. [12] empirically showed a correlation between the average number of hops that a message has to go through after node allocation and the runtime of tasks. They also proposed node allocation heuristics that increase throughput by 30% on average. Their algorithms linearly order the processors of the computational cluster by using space-filling curves. Nodes are then allocated for a task, to minimize the span of processors in this linear order. This algorithm requires only one pass over the linearized processor array. To break ties, best-fit or first-fit strategies were studied, and first-fit performed slightly better in the experiments. One direction for further work is to lift the linearized processor array assumption and generalize the node allocation techniques to higher dimensions where the connectivity of the parallel machine is more explicitly modeled.

4 Parallelizing Irregular Computations

Irregular computations are amongst the most challenging to parallelize. Irregularity can arise from complex geometries, multiscale spatial or temporal dependencies, or

a host of other causes. As mentioned above, graphs and hypergraphs are often used to describe complex data dependencies, and graph partitioning methods play a key role in parallelizing many such computations. However, there are many irregular applications that cannot be parallelized merely by partitioning, because the data dependencies are more complex than the graphs can model. Two examples are discussed below: multipole calculations and radiation transport.

4.1 Multipole Calculations

Perhaps a better definition of an irregular problem is one whose solution cannot be decomposed into a set of simple, standard, kernel operations. But with this definition, the space of irregular problems depends upon the set of accepted kernels. As parallel computing matures, the set of well-understood kernels steadily increases and problems that had once seemed irregular can now be solved in more straightforward manners. An excellent example of this trend can be found in the work of Hariharan and Aluru [8] on multipole methods for many-body problems.

Multipole methods are used to simulate gravitational or electromagnetic phenomena in which forces extend over long ranges. Thus, each object in a simulation can effect all others. This is naively an $O(n^2)$ calculation, but sophisticated algorithms can reduce the complexity to $O(n \log n)$ or even $O(n)$. These multipole algorithms represent collections of objects at multiple scales, combining the impact of a group of objects into a compact representation. This representation is sufficient to compute the effect of all these objects upon far-away objects.

Early attempts to parallelize multipole methods were complex, albeit effective. Space was partitioned geometrically and adaptively, load balancing was fairly ad hoc, communication was complex and there were no performance guarantees. By anyone's reckoning, this was a challenging, irregular computation. In more recent work, Hariharan and Aluru [8] have proposed a set of core data structures and communication primitives that enable much simpler parallelization. In this work, the complexity of early implementations is replaced by a series of calls to standard parallel kernels like prefix and MPI collective communication operations. By building an application out of well-understood steps, Hariharan and Aluru are able to analyze the parallel performance and provide runtime guarantees. With this perspective, multipole algorithms no longer need be seen as irregular parallel computations.

4.2 Radiation Transport on Unstructured Grids

Another example of an irregular computation is the simulation of radiation transport on unstructured grids. Radiation effects can be modeled by the discrete ordinates form of the Boltzmann transport equation. In this method, the object to be studied is modeled as a union of polyhedral finite elements, and the radiation equations are approximated by an angular discretization. The most widely used method to solve these equations is known as source iteration and relies on "sweeps" on each discretized angle. A sweep operation visits all elements in the order of the specified direction. Each face of the element is either "upwind" or "downwind"

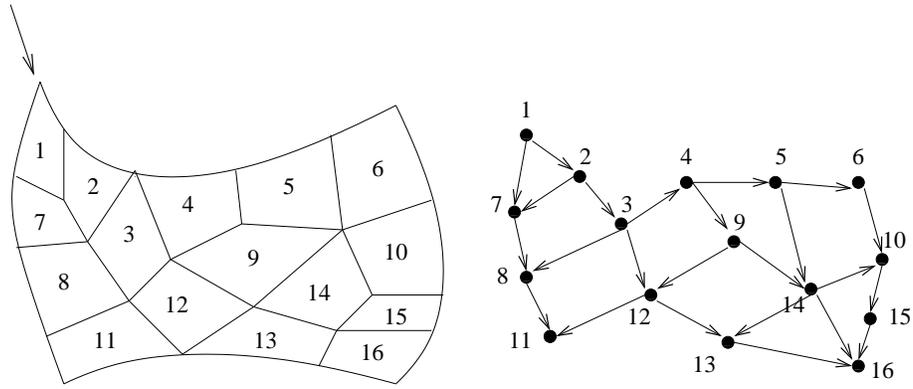


Figure 2. Directed graph for the sweep operation.

depending on the direction of the sweep. Computations at each node requires that we first know all the incoming flux, which corresponds to the upwind faces, and the output is the outgoing flux, that corresponds to flux through downwind faces.

As illustrated in Fig. 2, this process can be formally defined using a directed graph. Each edge is directed from the upwind vertex to the downwind one. The computations associated with an element can be performed if all the predecessors of the associated vertex have been completed. Thus, for each angle, the set of computations are sequenced as a topological sort of the directed graph. A problem arises, when the topological sort cannot be completed, i.e., the graph has a cycle. If cycles exist, the numerical calculations need to be modified, typically by using old information along one of the edges in each cycle, thereby removing the dependency. Decomposing the directed graph into strongly connected components will yield groups of vertices with circular dependencies. Thus scalable algorithms for identifying strongly connected components in parallel are essential. Most algorithms for finding strongly connected components rely on depth-first search of the graph, which is inherently sequential. Pinar et al. [15] described an $O(n \lg n)$ divide-and-conquer algorithm that relies on reachability searches. McLendon et al. [13] worked on an efficient parallel implementation of this algorithm and applied it to radiation transport problems.

The efficient parallelization of a sweep operation is crucial to radiation transport computations. A trivial solution is to assign a set of sweep directions to each processor, this however requires duplicating the mesh at each processor, which is infeasible for large problems. A scalable solution requires distributing the grid among processors and doing multiple sweeps concurrently. This raises the questions of how to distribute the mesh among processors and how to schedule operations on grid elements for performance.

Sweep scheduling is a special case of the precedence-constrained scheduling problem, which is known to be NP-Complete. For radiation transport, several heuristic methods have been developed and shown to be effective in practice [14, 17], but they lack theoretical guarantees. Recently, Kumar et al. [11] described the

first provably good algorithm for sweep scheduling. Their linear time algorithm gives a schedule of length at most $O(\log^2 n)$ times that of the optimal schedule. Their *random delay* algorithm assigns a random delay to each sweep direction. Each mesh element is then assigned to a processor uniformly at random. Each processor participates in the sweeps without violating the precedence constraints, and applying a random delay to each sweep. Kumar et al. show that this algorithm will give a schedule of length at most $O(\log^2 n)$ times the optimal schedule. Later, they propose an improved heuristic with the same asymptotic bound on the worst schedule length, but that performs better in practice. Experimental results on simulated runs on real meshes show that important improvements are achieved by using the proposed algorithms.

5 Computational Biology

In recent years, biology has experienced a dramatic transformation into a computational and even an information-theoretic discipline. Problems of massive size abound in newly acquired sequence information of genomes and proteomes. Multiple alignment of the sequences of hundreds of bacterial genomes is a computational problem that can be attempted only with a new suite of efficient alignment algorithms on parallel computers. Large-scale gene identification, annotation, and clustering expressed sequence tags (EST) are other large-scale computational problems in genomics. These applications are constructed from a variety of highly sophisticated string algorithms. Currently there are more than 5 million human EST's available in databases and this collection continues to grow. These massive data sets necessitate research into parallel and distributed data structures for organizing the data effectively.

Other aspects of biology are also being transformed by computer science. Phylogenetics, the reconstruction of historical relationships between species or individuals, is now intensely computational, involving string and graph algorithms. The analysis of micro-array experiments, in which many different cell types can simultaneously be subjected to a range of environments, involves cluster analysis and techniques from learning theory. Understanding the characteristics of protein interaction networks and protein-complex networks formed by all the proteins of an organism is another large computational problem. These networks have the *small-world* property: the average distance between two vertices in the network is small relative to the number of vertices. Semantic networks and models of the world-wide web are some other examples of such small world networks. Understanding the nature of these networks, many with billions of vertices and trillions of edges, is critical to extracting information from them or protecting them from attack. A more detailed discussion on computational problems in biology is provided in Chapter ?? of this book.

One fundamental problem in bioinformatics is sequence alignment, which involves identifying similarities among given sequences. Such alignments are used to figure out what is similar and what is different in the aligned sequences, which might help identify the genomic bases for some biological processes. One application of se-

quence alignment is finding DNA *signatures*. A signature is a group of subsequences in the DNA that is preserved in all strains in a set of pathogens, but unique when compared to all other organisms. Finding signatures requires multiple sequence alignments at the whole genome level. While dynamic programming is commonly used to optimally align small segments, the complexity of these algorithms is the product of the lengths of the sequences being aligned. The complexity, and the gap between its mathematical optimality and biological effectiveness make dynamic programming algorithms undesirable for whole genome level alignments. Hysom and Baldwin [9] worked on an alternative. They use suffix trees to find long subsequences that are common in all sequences. In a suffix tree, each suffix is represented by a path from the root to a leaf, and its construction takes only linear time and space. Once the suffix tree is constructed, long common subsequences can be easily found by looking at internal nodes of the tree. Among these long subsequences *anchors* are chosen for the basis of alignment, so that in the final alignment anchors are matched to each other, and the problem is decomposed to align subsequences between the anchors. Hysom and Baldwin use this decomposition to parallelize the alignment process.

6 Information Analysis

Advances in technology have enabled production of massive volumes of data through observations and simulations in many scientific applications such as biology, high-energy physics, climate modeling, and astrophysics. In computational high-energy physics, simulations are continuously run, and notable events are stored in detail. The number of events that need to be stored and analyzed is on the order of several millions per year. This number will go up dramatically in coming years as new accelerators are completed. In astrophysics, much of the observational data is now stored electronically, creating a *virtual telescope* whose data can be accessed and analyzed by researchers world wide. Genomic and proteomic technologies are now capable of generating terabytes of data in a single day's experimentation. A similar data explosion is impacting fields besides the conventional scientific computing applications and even the broader societies we live in, and this trend seems likely to continue.

The storage, retrieval, and analysis of these huge data sets is becoming an increasingly important problem, that cries out for sophisticated algorithms and high performance computing. Efficient retrieval of data requires a good indexing mechanism, however even the indexing structure itself often occupies a huge space due to the enormous size of the data, which makes the design of compact indexing structure a new research field [?]. Moreover the queries on these data sets are significantly different than those for traditional databases and so require new algorithms for query processing. For instance, Google's page ranking algorithm successfully identifies important web pages among those relevant to specified keywords. This algorithm is based on eigenvectors of the link graph of the web. Linear algebra methods are used elsewhere in information processing in latent semantic analysis techniques for information retrieval. In a similar cross-disciplinary vein, understanding the output

of large scale scientific simulations is increasingly demanding tools from learning theory and sophisticated visualization algorithms.

Graphs provide a nice language to represent the relationships arising in various fields such as the Web, gene regulatory networks, or people interaction networks. Many such networks have *power law* degree distributions. That is, the number of nodes with d neighbors is proportional to $1/d^\beta$ for some constant $\beta > 0$. This constant has been observed to be between 2 and 3 for a wide assortment of networks. One consequence is that these networks have small diameters, $O(\log \log n)$, where n is the number of nodes. A deeper understanding of the properties of complex networks, and algorithms that exploit these properties, will have a significant impact upon our ability to extract useful information from many different kinds of data.

The analysis of very large networks requires parallel computing. To parallelize the analysis, the network must first be divided among the processors. Chow et al. have studied this partitioning problem [2]. Partitioning a network into loosely-coupled components of similar sizes is important for parallel query processing, since loosely-coupled components enable localizing most of the computation to a processor with limited communication between processors. Although existing partitioning techniques are sufficient for many scientific computing problems, the data dependencies in complex networks are much less structured, and so new parallelization techniques are needed.

7 Solving Combinatorial Problems

The increasing use of combinatorial techniques in parallel scientific computing will require the development of sophisticated software tools and libraries. These libraries will need to be built around recurring abstractions and algorithmic kernels. One important abstraction for discrete problems is that of integer programming. A wide assortment of combinatorial optimization problems can be posed as integer programs. Another foundational abstraction is that of graph algorithms. For both of these general approaches, good parallel libraries and tools will need to be developed.

7.1 Integer Programming

Many of the combinatorial optimization problems that arise in scientific computing are NP-hard, and thus it is unreasonable to expect an optimal solution to be found quickly. While heuristics are a viable alternative for applications where fast solvers are needed and sub-optimal solutions are sufficient, for many other applications a provably optimal or near-optimal solution is needed. Examples of such needs arise in vehicle routing, resource deployment, sensor placement, protein structure prediction and comparison, robot design and vulnerability analysis. Large instances of such problems can only be solved with high-performance parallel computers.

Mixed-integer linear programming (MILP) involves optimization of a linear function subject to linear and integrality constraints, and is typically solved in practice by intelligent search based on branch-and-bound and branch-and-cut (constraint generation). Branch and Bound (B&B) recursively sub-divides the space of feasible solutions by assigning candidate values to integer variables, i.e., $x_i =$

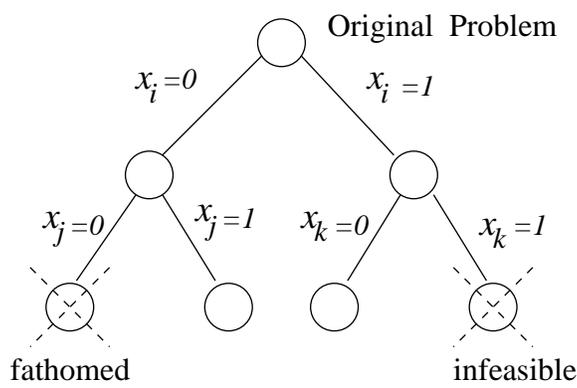


Figure 3. Branch-and-bound algorithm

0, 1, 2, ... Each branch represents the subdomain of all solutions where a variable has the assigned value, e.g., $x_i = 0$. These steps correspond to the “branching” component of a B&B algorithm. The other important component is bounding, which helps avoid exploring an exponential number of subdomains. For each subdomain a lower bound on the minimum (optimal) value of any feasible solution is computed, and if this lower bound is higher than the value of the best candidate solution, this subdomain is discarded. Otherwise, B&B recursively partitions this subdomain and continues the search in these smaller subdomains. Optimal solutions to subregions are candidates for the overall optimal. The search proceeds until all nodes have been solved or pruned, or until some specified threshold is met between the best solution found and the lower bounds on all unsolved subproblems.

Efficiency of a B&B algorithm relies on availability of a feasible solution that gives a tight upper bound on the optimal solution value, and a mechanism to find tight lower bounds on problem subdomains, to fathom subdomains early, without repeated decompositions. Since B&B can produce an exponential number of subproblems in the worst case, general and problem-specific lower and upper bound techniques are critical to keep the number of subproblems manageable in practice. Heuristics are commonly used for upper bounds. What makes MILPs attractive for modeling combinatorial models is that a lower bound on a MILP can be computed by dropping the integrality constraints and solving the easier linear-programming relaxation. Linear programming (LP) problems can be efficiently solved with today’s technology. However, tighter lower bounds necessitate closing the gap between LP polytope and the MILP polytope, that is narrowing the LP feasible space to cover only a little more than the integer feasible space. This can be achieved by dynamic constraint (a.k.a. cutting plane) generation, either for the whole problem or for the subdomains.

Branch-and-bound algorithms can effectively utilize large numbers of processors in a parallel processing environment. However, the ramp-up phase remains as a challenge. Eckstein et al. [3] designed and developed a Parallel Integer and Combinatorial Optimizer (PICO) for massively parallel computing platforms. They

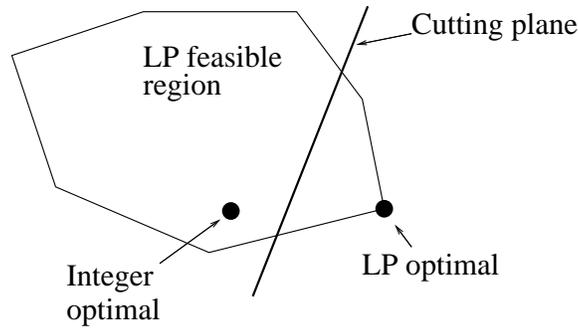


Figure 4. *Cutting planes close the gap between IP and LP feasible regions.*

observed that the presplitting technique that starts with branching to decompose the problem into one subdomain per processor often leads to poor performance, because it expands many problems that would be fathomed in a serial solution. Alternatively, they studied parallelizing the ramp-up phase, where many processors work in parallel on a single subdomain. This requires parallelization of preprocessing, LP solvers, cutting plane generation, and gradient computations to help with choosing which subdomain to decompose. A more detailed discussion on massively parallel integer programming solvers can be found in Chapter ?? of this book.

7.2 Libraries for Graph Algorithms

The importance of graph algorithms is growing due to the broad applicability of graph abstractions. This is particularly true in bioinformatics and scientific data mining. Scientific problems often generate enormous graphs that can only be analyzed by parallel computation. However, parallelization of graph algorithms is generally very hard and is an extremely challenging research field. Bader and colleagues have studied the parallelization of a number of fundamental graph operations, such as spanning trees and ear decompositions on SMPs for small numbers of processors. In Bader's spanning tree implementation [1], each processor starts growing trees from different vertices by repeatedly adding a vertex adjacent to a vertex in the current tree. Race conditions are handled implicitly by the SMP, and load balancing is achieved by work stealing between processors. Bader and Cong [1] also studied construction of a minimum spanning tree (MST), where the objective is to construct a spanning tree with minimum edge-weight sum. They used Boruvka's MST algorithm, which labels each edge with the smallest weight to join the MST, and at each iteration adds the edge with minimum cost to the tree. Bader and Cong experimented with different data structures for Boruvka's algorithm, and with a new algorithm where each processor runs Prim's algorithm until it is maximal, and then switched to Boruvka's algorithm. Their approach was the first to obtain speedup on parallel MST algorithms.

This and related work needs to be bundled into easy-to-use toolkits to facilitate the greater use of graph algorithms in parallel applications.

8 Conclusions

In this chapter, we have introduced a few of the areas in which combinatorial algorithms play a crucial role in scientific and parallel computing. Although some of these examples reflect decades of work, the role of discrete algorithms in scientific computing has often been overlooked. One reason for this is that the applications of combinatorial algorithms are scattered across the wide landscape of scientific computing, and so a broader sense of community has been hard to establish. This challenge is being addressed by the emergence of *combinatorial scientific computing* as a recognized subdiscipline.

It is worth noting that some of the most rapidly growing areas within scientific computing (e.g. computational biology, information analysis, etc.) are particularly rich in combinatorial problems. Thus, we expect combinatorial ideas to play an ever-growing role in high performance computing in the years to come.

Acknowledgements

We are grateful to Srinivas Aluru, David Bader, Chuck Baldwin, Michael Bender, Edmond Chow, Jim Demmel, Tina Eliassi-Rad, Assefaw Gebremedhin, Keith Henderson, David Hysom, Anil Kumar, Fredrik Manne, Alex Pothen, Madhav Marathe, and Jason Riedy for their contributions to the Eleventh SIAM Conference on Parallel Processing for Scientific Computing.

Bibliography

- [1] D. A. BADER AND G. CONG, *A fast, parallel spanning tree algorithm for symmetric multiprocessors (SMPs)*, in Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2004), Santa Fe, NM, Apr. 2004.
- [2] E. CHOW, T. ELIASSI-RAD, K. HENDERSON, B. HENDRICKSON, A. PINAR, AND A. POTHEN, *Graph partitioning for complex networks*. Presentation at SIAM Conf. on Parallel Processing and Scientific Computing, San Francisco, February 2004.
- [3] J. ECKSTEIN, W. HART, AND C. PHILLIPS, *Pico: An object-oriented framework for parallel branch-and-bound, inherently parallel algorithms in feasibility and optimization and their applications*, Elsevier Scientific Series on Studies in Computational Mathematics, (2001), pp. 219–265.
- [4] M. FRIGO, C. LEISERSON, H. PROKOP, AND S. RAMACHANDRAN, *Cache-oblivious algorithms*, in Proc. 40th IEEE Symp. on Foundations of Computer Science (FOCS 99), 1999, pp. 285–297.
- [5] A. H. GEBREMEDHIN, *Practical Parallel Algorithms for Graph Coloring Problems in Numerical Optimization*, PhD thesis, Department of Informatics, University of Bergen, 2003.
- [6] J. GILBERT. Personal communication, February 2004.
- [7] L. GRIGORI, X. S. LI, AND Y. WANG, *Performance evaluation of the recent developments in parallel superlu*. Presentation at SIAM Conf. on Parallel Processing and Scientific Computing, San Francisco, February 2004.
- [8] B. HARIHARAN AND S. ALURU, *Efficient parallel algorithms and software for compressed octrees with application to hierarchical methods*, Parallel Computing, to appear.
- [9] D. HYSOM AND C. BALDWIN, *Parallel algorithms and experimental results for multiple genome alignment of viruses and bacteria*. Presentation at SIAM Conf. on Parallel Processing and Scientific Computing, San Francisco, February 2004.
- [10] E.-J. IM, K. YELICK, AND R. VUDUC, *Optimization framework for sparse matrix kernels*, International Journal of High Performance Computing Applications, 18 (2004), pp. 135–158.

- [11] M. KOWARSCHIK, U. RUDE, C. WEISS, AND W. KARL, *Cache-aware multi-grid methods for solving poisson's equation in two dimensions*, Computing, 64 (2000), pp. 381–399.
- [12] V. A. KUMAR, M. MARATHE, S. PARTHASARATHY, A. SRINIVASAN, AND S. ZUST, *Provable parallel algorithms for radiation transport on unstructured meshes*, Tech. Rep. LA-UR-04-2811, Los Alamos National Laboratory, 2004.
- [13] V. LEUNG, E. ARKIN, M. A. BENDER, D. BUNDE, J. JOHNSTON, A. LAL, J. MITCHELL, C. PHILLIPS, AND S. SEIDEN, *Processor allocation on cplant: Achieving general processor locality using one-dimensional allocation strategies*, in Proceedings of the 4th IEEE International Conference on Cluster Computing (CLUSTER), 2002, pp. 296–304.
- [14] W. MCLENDON III, B. HENDRICKSON, S. PLIMPTON, AND L. RAUCHWERGER, *Finding strongly connected components in distributed graphs*, J. Parallel Distrib. Comput. Submitted for publication. Earlier version in Proc. 10th SIAM Conf. Parallel Processing for Sci. Comput.
- [15] S. D. PAUTZ, *An algorithm for parallel S_n sweeps on unstructured meshes*, Nucl. Sci. Eng., 140 (2002), pp. 111–136.
- [16] A. PINAR, L. K. FLEISCHER, AND B. HENDRICKSON, *A divide-and-conquer algorithm to find strongly connected components*, Tech. Rep. LBNL-51867, Lawrence Berkeley National Laboratory, 2004.
- [17] A. PINAR AND M. HEATH, *Improving performance of sparse matrix vector multiplication*, in Proc. IEEE/ACM Conf. on Supercomputing 1999, Portland, OR, 1999.
- [18] A. PINAR, T. TAO, AND H. FERHATOSMANOGLU, *Compressing bitmap indices by data reorganization*, in Proc. 21st International Conference on Data Engineering (ICDE 2005), 2005.
- [19] S. PLIMPTON, B. HENDRICKSON, S. BURNS, W. MCLENDON III, AND L. RAUCHWERGER, *Parallel algorithms for S_n transport on unstructured grids*, Nucl. Sci. Eng. To Appear. Earlier version in Proc. SC'00.
- [20] J. RIEDY AND J. DEMMEL, *Parallel weighted bipartite matching*. Presentation at SIAM Conf. on Parallel Processing and Scientific Computing, San Francisco, February 2004.
- [21] S. TOLEDO, *Improving the memory-system performance of sparse matrix vector multiplication*, IBM Journal of Research and Development, 41 (1997).