A High-Performance Toolkit for Fast Exact Algorithms[*]

Faisal N. Abu-Khzam, Michael A. Langston and Pushkar Shanbhag
Department of Computer Science, University of Tennessee, Knoxville, TN 37996–3450

Abstract

Ongoing work is described in which fast graph algorithms are combined with parallel, grid and reconfigurable technologies. This synergistic strategy can help solve problem instances too large or too difficult for standard techniques. Target problems need only be amenable to reduction and decomposition.

[*]This research is supported in part by the National Science Foundation under grants EIA–9972889 and CCR–0075792, by the Office of Naval Research under grant N00014–01–1–0608, by the Department of Energy under contract DE–AC05–00OR22725, and by the Tennessee Center for Information Technology Research under award E01–0178–081.

Overview

We describe work in progress that combines novel algorithmic methods with powerful platforms and supporting infrastructure. We employ these emergent tools and technologies to launch systematic attacks on problems of significance. Preliminary results show considerable promise, often reducing runtimes from days to seconds, and bringing us ever closer to solving problems previously viewed as hopelessly out of reach.

Exemplar

For brevity, we restrict our attention to the clique problem. Clique is probably one of the best known $\mathcal{NP}$-complete problems, with relevance in a variety of applications. In bioinformatics, for example, clique has utility in toolchains for phylogeny, microarray analysis and SELDI (surface enhanced laser desorption/ionization). Here researchers seek to discover a large clique of highly correlated protein sequences, DNA samples or biomarkers. We solve this problem by asking instead for a small vertex cover (G has a clique of size at least $n - k$ if and only if $\overline{G}$ has a vertex cover of size at most k.)

Fast Exact Algorithms

Our vertex cover algorithms exploit reduction and decomposition. During reduction, we condense an arbitrarily difficult instance into its combinatorial core. It has long been known that, if a cover is present, removing vertices whose degree exceeds k reduces G to a graph of size at most k^2 [2]. More complex techniques rely on linear programming relaxation [8, 13]. We have fine-tuned and implemented these and a number of more recent ideas [1], culminating in a suite of polynomial-time routines that yield cores of size $2k$ or less. When reduction is complete, the core is ready for decomposition. Decomposition is challenging, because the solution space that must be searched typically holds an exponential number of candidates. For this we use a tree to structure the search for a satisfying cover. Each internal node of the tree represents a choice. For example, one might make the choice at the root by selecting an arbitrary vertex, v. The left (right) subtree may then denote the set of all solutions in which v is to be in (not in) the cover. For problems like vertex cover that are fixed-parameter tractable, reduction and decomposition are often termed kernelization and branching, respectively. Reduction and decomposition work equally well for many problems that are not fixed-parameter tractable. See, for example, the work reported in [14] for the hitting set problem.

Resources

We complement the algorithmic engine described above with parallel machines, gridware and, when needed, hardware acceleration. Parallelization works well with decomposition. The spawning of processes is structured by the tree used to explore the core's search space. Once spawned, however, these tasks are left to run in a virtually unstructured manner. Neither barrier synchronization nor MPI-like tools are required. Suppose, for example, that 32 processors are available. Decomposition will use the first 5 ($<< k$) levels of its tree to split the input into 32 subgraphs, one for each processor. In turn, each processor will, in parallel, examine its subgraph using the search tree technique. Almost any architectural model will do. We have run initial experiments on several different platforms including NOWs, SMPs, and near-random confederations of motley machines. We have also tried assorted grid middleware, including NetSolve [6], Condor [11] and Globus [7]. Our best results have generally been obtained with minimal intervention, however, in the extreme case by launching naked secure shells (SSHs). For recalcitrant subproblems, we aim to gain additional acceleration through the use of reconfigurable hardware [9]. We have recently brought on line a cluster with 12 Unix CAD workstations and eight Pilchard boards developed by Philip Leong's research group [10]. We can access these directly, or through the use of the program description file mechanism of NetSolve. Each board fits into the DIMM slot of a Linux box, in this manner greatly reducing FPGA-CPU I/O latency. We are currently prototyping, synthesizing and testing VHDL versions of our codes.

Preliminary Results

We have extracted data from the National Center for Biotechnology Information (NCBI) [12]. This is because synthetic data sets (e.g., pseudo-random graphs, grids, etc) are well-known to reflect inaccurately on genuine biological data. One of the first things we observed is probably best characterized as super-super-linear speedup. Parallel algorithms on 32 processors were sometimes finishing in seconds, while sequential methods were taking days. We at first thought our sequential routines must be hung. Traces revealed, however, that they were humming along nicely. It is just that biologically meaningful graphs tend to have a lot of edges, and so their search spaces are huge. Of course we do not always achieve speedups of this magnitude. But we virtually always seem to achieve at least moderately super-linear speedup. It turns out that this general sort of behavior was first observed a few years ago [5], when it was discovered that solutions tend to be highly non-uniformly distributed. Thus decomposition does much more than merely help guide and parallelize the search. It very often happens that one or more processors finds a solution relatively close to the root of its respective subtree. Yet the obvious sequential algorithm plods along, exhaustively examining each and every subtree until it stumbles across a solution-laden region of the search space.

The following table shows a few representative timing results we obtained using graphs derived from the well-known globin domain. The number of protein sequences (and hence the graph size) is 972. In these runs we invoked one of our sequential reduction codes, followed by one of our parallel decomposition and search codes. We employed 32 processors, each running at 512 MHz. Over the spectrum of possible thresholds, our routines never required more than ten seconds of total elapsed time. In many cases, reduction was so effective that it eliminated the core completely, and with it the need for decomposition and search. In contrast, the fastest previously-published approach [3] used 27 processors, each running at 1.2 GHz, and required over an hour to solve instances with 730 sequences.

Threshold Value	Clique Size	Reduction Time	Decomposition and Search Time
6	655	<2 seconds	not needed
9	609	<5 seconds	<3 seconds
12	552	<3 seconds	<1 seconds
15	492	<4 seconds	<1 seconds

Moving Forward

We are now implementing a wide assortment of improvements to our codes, as well as the aforementioned hardware acceleration via FPGAs. We are ramping up rapidly, now solving problem instances whose vertices number as high as 3000 (for phylogeny) and 5000 (for microarrays). Also, our methods have been selected for inclusion in a new ClustalW [4] parallel algorithms portal. Given the interest in applications for these methods, we are excited to be on the threshold of solving problems of genuine practical merit, many of which were until recently thought by practitioners to be so large that they were unassailable.

Acknowledgments

We are grateful to Frank Dehne and Mike Fellows for their collaboration and encouragement, and to Bill Cook for providing us with the streamlined linear programming code we used in our reduction algorithms.

References

[1] F. N. Abu-Khzam, M. A. Langston, and P. Shanbhag. Vertex cover: A case study on the practical feasibility of exact algorithms for fixed-parameter tractable problems. Technical Report UT-CS-02-494, Department of Computer Science, University of Tennessee, 2002.

[2] J.F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993.

[3] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. Technical report, Department of Computer Science, Carleton University, Ottawa, Canada, 2002.

[4] ClustalW. See `http://helix.nih.gov/apps/bioinfo/clustalw.html`.

[5] F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Coarse grained parallel fixed-parameter tractable algorithms. Technical report, Department of Computer Science, Carleton University, Ottawa, Canada, 2000.

[6] J. Dongarra. NetSolve. See `http://icl.cs.utk.edu/netsolve`.

[7] I. Foster and C. Kesselman. The globus project. See `http://www.globus.org/`.

[8] S. Khuller. The vertex cover problem. *ACM SIGACT News*, 33:31–33, June 2002.

[9] J. M. Lehrter, F. N. Abu-Khzam, D. W. Bouldin, M. A. Langston, and G. D. Peterson. On special-purpose hardware clusters for high-performance computational grids. In *Proceedings, International Conference on Parallel and Distributed Computing and Systems*, pages 1–5, 2002.

[10] P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, and K. H. Lee. Pilchard – a reconfigurable computing platform with memory slot interface. In *IEEE Symposium on Field-Programmable Computing Machines*, 2001.

[11] M. Livny and M. Solomon. Condor, high throughput computing. See `http://www.cs.wisc.edu/condor/`.

[12] NCBI. See `http://www.ncbi.nlm.nih.gov/`.

[13] G.L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[14] K. Weihe. Covering trains by stations or the power of data reduction. In *Proceedings, International Conference on Algorithms and Experiments*, pages 1–8, 1998.