Parallel Local Elimination Algorithms for Sparse Discrete Optimization Problems

Daria Lemtyuzhnikova[1] and Oleg Shcherbina[2]

[1] Tavrian National University, Simferopol, Ukraine,
darabbt@gmail.com
[2] University of Vienna, Vienna 1090, Austria,
Oleg.Shcherbina@univie.ac.at,
WWW home page: http://www.mat.univie.ac.at/∼oleg

Abstract. The development and study of a parallel implementation of the graph-based local elimination algorithms on novel emergent parallel GPU-based architectures for solving sparse discrete optimization (DO) problems are discussed.

Keywords: parallel computing, discrete optimization, graphics processing unit, local elimination algorithm

1 Introduction

The use of discrete optimization (DO) models and algorithms makes it possible to solve many real-life problems in scheduling theory, optimization on networks, routing in communication networks, facility location in enterprize resource planing, and logistics. Applications of DO in the artificial intelligence field include theorem proving, SAT in propositional logic, robotics problems, inference calculation in Bayesian networks, scheduling, and others.

Even on contemporary serial computers solving combinatorial optimization problems is somewhat limited from a computational point of view. Parallel processing computers could greatly reduce the computation time for solving large-scale DO problems since there are a number of parallel operations that occur in solving DO problem. Due to the high computational requirements and inherent parallel nature of DO search techniques, there has been a great deal of interest in the development of parallel DO search methods since the dawn of parallel computing [5].

Decomposition DO algorithms are prime candidates for parallelization. As Bertsekas and Tsitsiklis [3] (p.225) mentioned, "When a parallel computing system is available, decomposition methods typically become even more attractive because the simple subproblems can be solved in parallel".

A graph-based unifying framework for the main structural DO decomposition algorithms is provided in [11] that allows to unify and clarify the notation and algorithms of various structural DO decomposition approaches, and to accommodate constraint satisfaction technology transfer.

Among decomposition approaches appropriate for solving sparse DO problems we mention *local elimination algorithms* (LEA) using the special block matrix structure of constraints and nonserial dynamic programming algorithms (NSDP) [2], which can exploit sparsity in the dependency graph of a DO problem and allow to compute a solution in stages such that each of them uses results from previous stages. (LEA) compute global information using local computations (i.e., computations of information about elements of neighborhoods of variables or constraints).

The main disadvantage of the LEAs, its great computational complexity for DO problems with high size of separators, can be reduced with parallel processing on clusters of workstations. The main problem is the large size of the separator, which separates the various blocks because a volume of enumeration is exponential on size of the separator. In order to reduce the amount of enumeration, postoptimality analysis can be used, since DO problems in the package, that corresponds to a block, differ in right-hand sides. However, our experiment showed a slight decrease in run time when using postoptimality analysis [12]. Another possibility to cope with the challenges posed by the large size separators is to develop approximate versions of the local algorithm, which does not completely enumerate all possible values of the variables included in the separator, but only partially. Thus, the only one way to implement and use exact local elimination algorithms for sparse DO problems with large separators is using of parallelization. Parallelization in DO can help to cope with this drawback using the possibility of parallel solving of DO problems in blocks.

The *elimination tree* [13] serves to characterize the parallelism in solving DO problems with LEA. In particular, the height of the elimination tree gives a rough measure of the parallel computation time. Thus, the elimination tree structure provides information on data dependency in DOP. It captures the essential ingredient for parallel elimination. An ordering of the elimination tree is a topological ordering where each node is numbered higher than all of its descendants.

There are various computational schemes for realizing the LEA, including the LEA of variables elimination, block-elimination algorithm, LEA based on tree decomposition. In LEA of variables elimination (nonserial dynamic programming), a variable is chosen, all the components of objective function and constraints that involve this variable are removed from the problem, and the marginal of the combination of these functions on the rest of the problem is added to the original problem. The problem obtained has one less variable and the same optimal objective function as the original problem [11]. By repeatedly eliminating all variables, we ultimately get the optimal objective function.

Instead of eliminating one variable after the other, one may eliminate several variables at once. This is called block elimination [2], [11]. To implement the block LEA, it is possible to use a tasks dependency graph which is indeed a tree, that must be processed from the leaves to the root.

Two sources of parallelization can be used in this process: the dominating one is based on partial problem assignments in separators, the second one comes from the branching of the tree-decomposition itself. Thus, there are two types of parallelism available in structural decomposition algorithm:

1) Parallelism of type 1 introduces parallelism when performing the operations on generated subproblems (in blocks). It consists of solving each DO subproblem in parallel for each block to accelerate the execution.

2) Parallelism of type 2 consists of processing the elimination tree in parallel by performing operations on several subproblems simultaneously. Independent branches of the elimination tree can be processed in parallel, and we refer to this as type 2 parallelism or tree parallelism. It is obvious that in general, tree parallelism can be exploited more efficiently in the lower part of the elimination tree than near the root node.

Nowadays, the novel emergent parallel computing architectures such as multicore processors, graphics processing units (GPUs), and grid environments provide new opportunities to apply parallel computing techniques to improve the local elimination algorithms search results and to lower the required computation times.

GPU programming has become increasingly popular in the scientific community during the last few years. In the early days, GPGPU programs used the normal graphics APIs for executing programs [9]. The Khronos Group has released the OpenCL[3] specification, which is a framework for writing programs that execute across platforms consisting of CPUs and GPUs. Other than OpenCL, there exist other frameworks that allow platform-independent programming for GPUs: 1) CUDA[4], 2) DirectCompute from Microsoft, 3) OpenGL Shading Language (GLSL). DirectCompute is specific to Microsoft Windows and therefore it is not portable between host Operating System (OS). GLSL does not have the scientific focus. OpenCL is an open standard that can be used to program CPUs, GPUs, and other devices from different vendors, while CUDA is specific to NVIDIA GPUs. OpenCL offers portability across GPU hardware, OS software, as well as multicore processors. Therefore OpenCL is our choice of implementing algorithms for structural decomposition of sparse discrete optimization problems.

To survey publications dedicated to parallel GPU-based DO algorithms we mention following works. A survey [10] on parallel ant colony optimization contains 106 references. Ant Colony Optimization is a population-based metaheuristic for solving optimization problems. Master-worker parallel ACO implementations have been quite popular mainly due to the fact that this model is conceptually simple and easy to implement. Luong et al. (2011) [8] reported about GPU-based parallel heuristics (such as Local Search Algorithms, Evolutionary Algorithms). Boyer et al. (2012) [4] proposed a parallel implementation via CUDA of the dense dynamic programming method for knapsack problems on a multi GPU system. Fujimoto, N. and Tsutsui, S. (2010) [6] develop highly-parallel TSP solver for a GPU computing platform. Satisfiability. Gulati and Khatri [7] Boolean Satisfiability on a Graphics Processor implemented a new variable or-

[3] OpenCL = Open Computing Language, see OpenCL: The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl

[4] CUDA = Compute Unified Device Architecture

dering approach in a complete procedure of solving SAT problem (MiniSAT), which runs on the CPU. Beckers et al. [1] considered parallel SAT-solving with OpenCL parallel programming environment and implemented a massively parallel SAT-solver on GPU.

We propose to use for parallel LEA implementation a hybrid master-worker system allowing the concurrent use of CPUs and GPUs. The GPUs, many core parallel machines with shared memory, act as the workers and perform solving DO subproblems for blocks. Synthesis of solution is performed by the CPU which has the role of the master.

References

1. Beckers, S., De Samblanx, G., De Smedt, F., Goedeme, T., Struyf, L., Vennekens, J.: (2011) Parallel SAT-solving with OpenCL, `http://hgpu.org/?p=5769`
2. Bertele, U., Brioschi, F.: Nonserial Dynamic Programming. Academic Press, New York (1972)
3. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice-Hall, Upper Saddle River (1989)
4. Boyer, V., El Baz, D., and Elkihel, M.: Solving knapsack problems on GPU. Comput. Oper. Res. 39 (1), 42-47 (2012)
5. Crainic, T., Le Cun, B., and Roucairol, C.: Parallel Branch-and-Bound Algorithms. Chapter 1 in: Parallel Combinatorial Optimization. Wiley Series on Parallel and Distributed Computing, pp. 1-28 (2006)
6. Fujimoto, N., and Tsutsui, S.: A highly-parallel TSP solver for a GPU computing platform. In: Proc. of the 7th int. conf. on Numerical methods and applications (NMA'10), Dimov, I. et al. (eds.). Springer-Verlag, Berlin, Heidelberg, pp. 264–271 (2010)
7. Gulati, K., and Khatri, S.P.: Boolean satisfiability on a graphics processor. ACM Great Lakes Symposium on VLSI 2010, pp. 123–126 (2010)
8. Luong, T-V., Melab, N., Talbi, E-G.: GPU-based multi-start local search algorithms. In: Proceedings of Learning and Intelligent Optimization (LION'2011), Rome, Italy, pp. 321–335 (2011)
9. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., and Purcell, T.J.: A survey of general-purpose computation on graphics hardware. In: Eurographics 2005, State of the Art Reports, August 2005, pp. 21-51 (2005)
10. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Applied Soft Computing 11, 5181–5197 (2011)
11. Shcherbina, O.: Graph-based local elimination algorithms in discrete optimization. In: Foundations of Computational Intelligence Volume 3. Global Optimization Series: Studies in Computational Intelligence, Vol. 203, pp. 235–266, Springer, Berlin/Heidelberg (2009)
12. Sviridenko, A., Shcherbina, O.: Block local elimination algorithms for solving sparse discrete optimization problems, `http://arxiv.org/abs/1112.6335`
13. Liu, J.W.H.: The role of elimination trees in sparse factorization. SIAM J. on Matrix Analysis and Applications 11, 134–172 (1990)