PARALLEL ALGORITHMS FOR SOLVING LARGE ASSIGNMENT PROBLEMS

PI: Rakesh Nagi[1]
Co-PI: Ketan Date[1]

[1]University of Illinois at Urbana-Champaign

EXECUTIVE SUMMARY

The goal of our project is to develop fast and scalable algorithms for solving large instances of Linear Assignment Problem (LAP) and Quadratic Assignment Problem (QAP) using Graphics Processing Units (GPUs). LAP is polynomial-time solvable with cubic worst-case complexity, while the QAP is strongly non-deterministic polynomial-time hard (NP-Hard). To solve a linearized model of the QAP using branch-and-bound, lower bounds must be calculated using the Lagrangian dual technique, in which a large number of LAPs must be solved efficiently. Additionally, in a branch-and-bound scheme, a large number of nodes must be explored to find a provable optimal solution. To this end, we have used Blue Waters to develop: (1) A GPU-accelerated Hungarian algorithm for solving large LAPs in an efficient manner; (2) A GPU-accelerated Lagrangian dual ascent heuristic for obtaining lower bounds on the QAP. These algorithms will be used in a parallel branch-and-bound scheme to solve large QAPs to optimality.

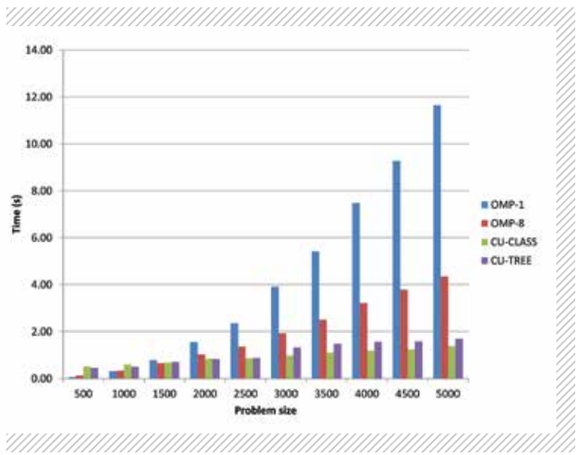


FIGURE 1: Comparison of execution times for CPU-based (OMP-1 and OMP-8) and GPU-based (CU-CLASS and CU-TREE) Hungarian algorithms.

INTRODUCTION

Assignment Problems are fundamental to the discovery in diverse branches of science and engineering. Some of their applications include information fusion, protein-protein interaction analysis, facilities design, vehicle routing, and resource scheduling. To gain meaningful insights, many applications demand quick solutions to large instances of Assignment Problems containing hundreds of thousands of vertices. This makes it incredibly challenging for the sequential algorithms designed for a single processor. Therefore, designing fast and scalable algorithms suitable for the state-of-the-art parallel programming architectures is essential. In this research, we intend to propose novel parallel algorithms for the Compute Unified Device Architecture (CUDA) enabled NVIDIA GPUs to solve the following two Assignment Problems.

Linear Assignment Problem (LAP): The objective of the LAP is to assign "n" resources to "n" tasks such that the total cost of the assignment is minimized. LAP can be solved in polynomial time using one of the many sequential/parallel algorithms that have been proposed in the literature. We chose to parallelize the famous Hungarian algorithm [1] on a GPU, whose theoretical complexity is O(n3).

Quadratic Assignment Problem (QAP): QAP was introduced by [2] as a mathematical model to locate indivisible economical activities (such as facilities) on a set of locations so as to minimize a quadratic cost function. One of the ways of solving the QAP is to convert it into a Mixed Integer Linear Program (MILP) by introducing additional variables and constraints and solve it using branch-and-bound with the help of a strong lower bounding technique. We chose to parallelize the Lagrangian dual ascent algorithm for Level-2 Refactorization-Linearization Technique (RLT2) proposed by [3], in which we need to solve O(n4) LAPs and adjust O(n6) Lagrange multipliers to obtain a strong lower bound on the QAP.

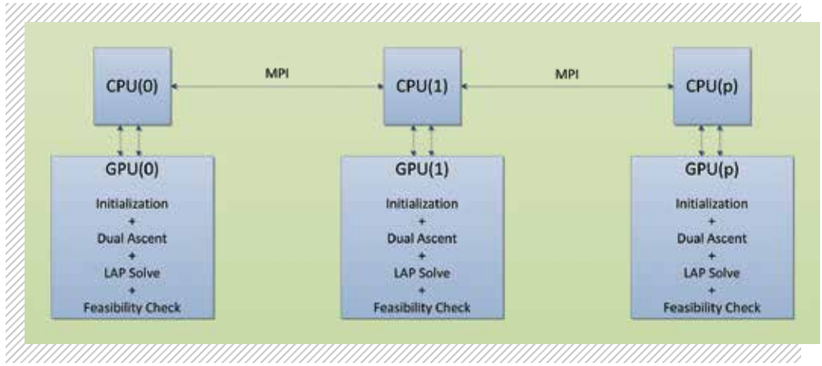


FIGURE 2: MPI+CUDA hybrid parallelization scheme for Lagrangian dual ascent.

METHODS & RESULTS

GPU-accelerated LAP solver: We have proposed two GPU-accelerated variants of the Hungarian algorithm. Our main contribution is an efficient GPU-based parallel algorithm for the augmenting path search, which is the most time intensive step. We show that our algorithm(s) can find multiple vertex-disjoint augmenting, which drastically reduces the execution time. Extensive numerical tests reveal that for problems with n > 1000, our algorithm(s) (CU-CLASS and CU-TREE) are substantially faster than the sequential and OpenMP implementations (OMP-1 and OMP-8) solved on a multi-core CPU (Fig. 1).

GPU-accelerated RLT2 solve: We designed a parallel Lagrangian dual ascent heuristic for solving RLT2 using hybrid MPI+CUDA architecture (Fig. 2), for which we used multiple GPUs. The O(n4) LAPs are split across these GPUs and solved using our GPU-accelerated Hungarian algorithm, while the O(n6) Lagrange multipliers are updated by multiple CUDA threads in parallel. This algorithm is scalable and provides good parallel speedup (Fig. 3) for problem instances Nug18, Nug20, Nug22, and Nug25 from the QAPLIB [4].

Parallel branch-and-bound solver for QAP: In this work, we used GPU-accelerated RLT2 solver in a branch-and-bound scheme to solve QAP instances to optimality. For a node in the search tree, we fix a facility to a location and solve the corresponding RLT2 sub-problem, whose objective value provides a lower bound on the QAP. If this value is greater than the incumbent solution then the node is fathomed, otherwise, it is branched further. Each node is processed using a bank of GPUs. By using multiple such banks, we can process multiple nodes in parallel. Table 1 shows the results for problem instances Nug18, Nug20, and Nug22 from the QAPLIB.

Problem	No. of banks	GPUs per bank	Nodes explored	Time (min)	Avg. Bank utilization
Nug18	10	9	514	16.75	0.916
Nug20	10	10	669	38.31	0.859
Nug22	12	12	969	56.63	0.931

TABLE 1: Branch-and-bound results for medium-sized QAPs.

WHY BLUE WATERS

In a typical branch-and-bound tree, we need to explore a large number of nodes in order to find an optimal solution. Also, as the problem size grows, the number of nodes that need to be explored grows exponentially. Therefore, we need a large number of processors which can explore the solution space in parallel. Additionally, the GPU-accelerated dual ascent procedure benefits from the large number of powerful GPU-enabled processors available at the Blue Waters facility. Coupling the parallel branch-and-bound with the fast GPU-based lower bounding techniques will enable us to solve large-sized problems from the QAPLIB, which still remain unsolved.

NEXT GENERATION WORK

Our ultimate objective through this research is to provide efficient solution methods for a class of Assignment Problems. These problems arise in diverse branches of science and engineering and

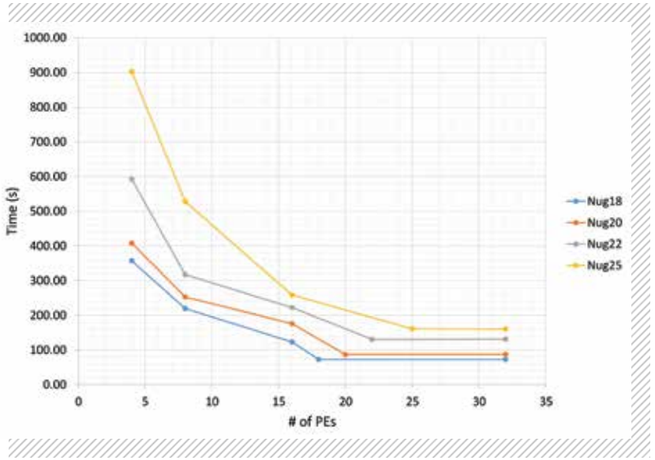


FIGURE 3: Scalability results for 200 iterations of parallel Lagrangian dual ascent on instances from QAPLIB.

they are part of many cutting-edge projects with high socio-economic impact. Using our methods, scientists and engineers will be able to solve Assignment Problems containing hundreds of thousands of vertices within a matter of minutes, leading to **transformative** discoveries. The parallel programming library developed during this research will provide a platform for solving large-scale Assignment Problems and comparing the performance of different algorithms, thus ensuring continued advancement of science and engineering.

PUBLICATIONS AND DATA SETS

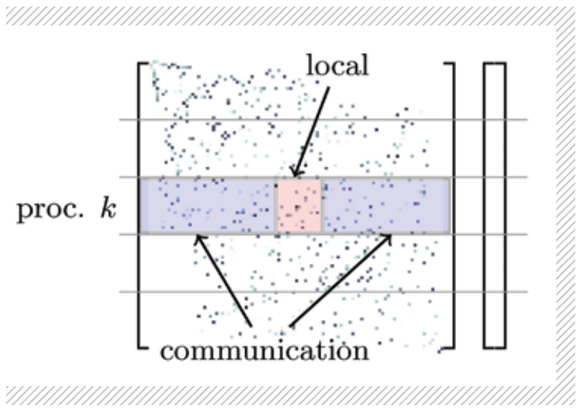
Date, K., R. Nagi, GPU-accelerated Hungarian algorithms for the Linear Assignment Problem. *Parallel Comput.*, (2016), http://dx.doi.org/10.1016/j.parco.2016.05.012

THE NEXT GENERATION OF LARGE-SCALE SPARSE MATRIX COMPUTATIONS

FIGURE 1: Communication in sparse matrix-vector multiplication

PI: Luke Olson[1]
Collaborators: Amanda Bienz[1], Bill Gropp[1], and Andrew Reisner[1]

[1]University of Illinois at Urbana-Champaign



EXECUTIVE SUMMARY

Sparse matrix computations are a common element in a range of high-performance computing applications and often dominate computation, particularly at scale. The goal of this project is to develop numerical schemes that limit parallel communication in sparse matrix methods, leading to efficient and scalable sparse operations without loss of accuracy.

Iterative methods for approximating the solution to sparse linear systems rely on sparse matrix-vector multiplication as a key computational kernel. Yet communication costs drive the complexity of these operations. The focus of this project is on algorithm design to alleviate communication costs at scale.

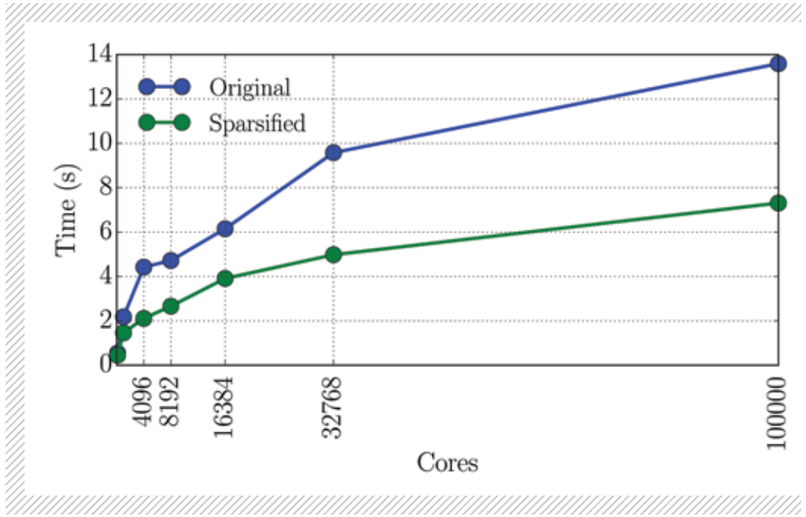


FIGURE 2: Effect of matrix sparsification on solver time.

This project has contributed two key developments: the ability to *sparsify* the matrix operations to limit communication and the method of redundant computation to localize computation, thus limiting communication distance and time.

INTRODUCTION

Sparse matrix problems of the form $Ax=b$ are common in many large-scale simulations. In this project, multigrid methods are considered as a method for the iterative approximation for this problem. Multigrid methods construct a series or hierarchy of smaller sparse matrices that are used to form an iterative refinement of the solution. This leads to fewer floating-point operations in the solution to $Ax=b$, but potentially very high communication costs and memory movement relative to the amount data in the computation.

A key kernel in this process is that of sparse matrix-vector multiplication $w \leftarrow A\ v$. Here, the sparsity pattern—or the location of non-zero entries in the matrix—governs the computational expense of the operation. A common bottleneck in the computation is the scenario where the sparse matrix resides on a high number of cores with a low number of matrix elements per core (Fig. 1).

Achieving efficiency in these computations is critical as many applications rely on these sparse operations throughout the simulation, leading to significant resource usage.

METHODS & RESULTS

Two methods were explored to address the communication demands in the problem. First, a method of sparsification whereby specific entries in the matrix are eliminated from the computation is explored. In multigrid, the matrices in the hierarchy are successively smaller, but relatively more dense, leading to wider communication patterns. A method is designed in [1] that avoids communicating work associated with *weak* or low-influence matrix entries in the problem. This leads to a notable speed-up of 2x, particularly at large core counts (Fig. 2). In addition, the *topology* of the system is also incorporated in the communication of sparse entries, preferring local on-node communication in the algorithm. This leads to a significant reduction in total message traffic, thereby reducing system contention and total simulation time (Fig. 3).

Ultimately, sparse matrices in the multigrid hierarchy are too small to reside on the full set of processing elements. Therefore a redistribution of data is needed to map matrix data to a subset of processors. This comes at a cost, which was explored in this project. By performing a redistribution of data based on a performance model, nearly scalable results can be achieved. As an example, for p processors,