Heuristic Acceleration of Routing in Transportation Simulations using GPUs

(Poster Abstract)

Sunil Thulasidasan
Computer,Computational and Statistical Sciences Division
Los Alamos National Laboratory
sunil@lanl.gov

ABSTRACT

We present a brief overview of how path computations for real-world road network graphs can be accelerated using GPUs, in the context of a large scale transportation simulator. This is motivated by the following observations: (i) Routing is the most computationally intensive part of a large scale transportation simulator, and thus, optimizations to routing that exploit any inherent structure and parallelism in the problem significantly improve application performance; (ii) near-optimal paths are often acceptable in lieu of the most optimal path, which motivates the use of heuristics in speeding up path computations; and finally, (iii) paths from source to destination in real-world road network graphs are spatially constrained, implying that to solve the path problem, one needs to consider only a small subset of the routing graph. This has important practical implications for a GPU, which, while offering a great degree of parallelism, is also significantly constrained by memory limitations. GPUs have already been shown to significantly outperform CPUs in path computation problems for small graphs. The previous observations allow us to deploy GPUs for much larger graphs that arise in transportation simulations.

Keywords

Transportation simulation, GPU, path computation

1. INTRODUCTION

The emergence of compute-intensive stream processors like the Graphics Processing Unit (GPU) represents a disruptive paradigm shift in the evolution of computer architecture. Though originally developed for numerically intensive media applications, advances in performances and programmability have resulted in these processors being successfully deployed for numerous computationally demanding scientific tasks [6]. The impressive arithmetic power and massive parallelism of today's graphics processors is an outcome of the highly parallel, numerically intensive nature of computer graphics. Not surprisingly, the general purpose computing tasks that have been able to leverage the performance benefits of these processors exhibit many computational similarities to graphics applications, namely 1) high data parallelism, 2) high compute intensity, and 3) data locality which results in structured (or coalesced)memory accesses. However, many problems do not readily exhibit the above properties; in particular, problems dealing with graph-based data structures, found in a variety of scientific domains, have been traditionally ill-suited to the GPU. However, recent work [3] and our own observations described in this paper provide some important insights as to how GPUs can be exploited in certain cases.

In this paper we present a brief overview of how GPUs can be utilized for path computations in real-world road graphs, in the context of FastTrans [8], a large scale transportation simulator that was developed at the Los Alamos National Laboratory. FastTrans is a scalable, parallel microsimulator for transportation networks that can simulate and route tens of millions of vehicular trips on real-world road networks. Using parallel discrete-event simulation techniques [2] and distributed-memory algorithms, we are able to model transportation networks of large geographic regions – consisting of over a million road network elements and over 20 million vehicles – and scale these simulations to execute on large, high performance clusters up to 20 times faster than real time.

Large-scale simulations are an important tool in the emerging field of infrastructure modeling, where simulating the behavior of millions of entities and their interactions with various interdependent infrastructure networks (like transportation, communication, electric power) demand significant computational resources. For FastTrans, the most computationally demanding module is routing. The routing graph for a large metropolitan region can contain tens of millions of vertices and edges, and a naive routing algorithm can severely cripple the performance of the simulator. In the following sections, we present an overview of the routing module in FastTrans including heuristic optimizations to increase performance, and a discussion of how GPUs can help further accelerate path computations in real-world road networks.

2. ROUTE COMPUTATIONS FOR TRANSPORTATION SIMULATIONS

The routing module is a crucial part of a transportation simulator – the algorithm and parameters used to route ve-

SIMUTOOLS 2011, March 21-25, Barcelona, Spain
Copyright © 2011 ICST 978-1-936968-00-8
DOI 10.4108/icst.simutools.2011.245589

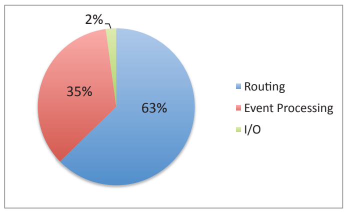

Figure 1: Execution profile of FastTrans in a serial simulation on a 3 GHz Mac-Pro workstation.

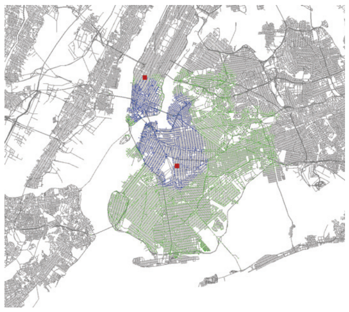

Figure 2: Nodes expanded in A^* (shown in blue) vs. Dijkstra (shown in green) in a real road network.

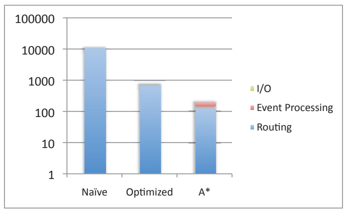

Figure 3: Logarithmic chart showing various overheads for Dijkstra, optimized Dijkstra, and A^* in a serial simulation run.

hicles are critical to the validity of the simulation model. In addition, it also accounts for more than half of the total execution time (Figure 1). We use *dynamic routing* in FastTrans, where routes are calculated on-demand; calculating routes dynamically allows us to achieve fast responses to congestion with the additional benefit of reduction in the input data size. However, route computations on large graphs can easily become a bottleneck, leading to severe performance degradation. We use several optimizations to make online route computations feasible in FastTrans, where we require the shortest path between a pair of nodes. While in the worst-case, the asymptotic time complexity of path-computation for a source-destination pair is same as computing a shortest-path tree, we observe that in practice, there are often significant differences in running times, especially when the source and destination are geographically close. We also exploit the fact that the problem domain (road-network routing) allows us to use paths that are not necessarily optimal; this motivates investigation of very fast heuristic algorithms that obtain only near-optimal paths.

2.1 Heuristic Routing: The A* Routing Algorithm

FastTrans initially used different variations of the standard Dijkstra's algorithm [1]. In the naive version, shortest-path trees, rooted at the source, are constructed for each routing query. In the optimized version, the search loop is terminated upon finding the shortest path to the destination. We then implemented A^* search [4], a variant of Dijkstra that employs a heuristic cost-function to bias the direction of the search towards the destination. Further optimizations that were used in all these implementations include smart label reset (where only nodes explored in a previous routing computation are re-initialized) and use of efficient data structures.

A^*, first proposed in AI literature [4, 7], exploits the near-Euclidean property of road-network graphs to expand the shortest path tree in the direction of the destination. By contrast, in Dijkstra, the search tree is expanded in a circular manner centered at the source node. This results in A^* arriving at the destination node much quicker than Dijkstra.

To bias the search towards the destination, we assign a cost to each intermediate vertex as follows: given source s, and destination t, for each intermediate vertex v, cost $C(v)$ is defined as:

$$C(v) = l(s, v) + D(v, t).$$

where $l(s, v)$ is the shortest path length from s to v, and $D(v, t)$ is the estimated cost from v to t computed as a function of the Euclidean distance between v and t. Since we are interested in the shortest path in terms of time rather than distance, $l(s, v)$ is the time-cost of the path from s to v, and we define $D(v, t)$ as:

$$D(v, t) = \frac{E(v, t)}{V_{max}}$$

where $E(v, t)$ is the Euclidean distance from v to t and V_{max} is the maximum allowable speed in the network. Note that the resulting paths using A^* are not provably optimal; however, since the road graph is almost Euclidean, our experiments showed that the paths computed by A^* are on average only 0.02% longer than the paths computed by Dijkstra. Figure 2 illustrates the performance of A^* versus an optimized version of Dijkstra on a real-world road network (from the northeast region of the United States). Notice that A^* explores a smaller set of nodes than even the optimized Dijkstra. Figure 3 shows, on a logarithmic scale, the speed-up as well as the routing overhead for the three implementations (Dijkstra, optimized Dijkstra, and A^*) from a code-profiling exercise of a serial simulation run. The serial run was executed on a 3 GHz Mac-Pro work-station for 20,000 itineraries that were randomly sampled from the study-set.

2.2 Heuristic Routing: A Case for GPU Acceleration

In the previous section we have seen how a heuristic based A^* search provides significant improvement in search speed by exploring only a fraction of the nodes explored by Dijkstra (see Figures 2 and also 4). These examples also illustrate a remarkable property of the road network graph, namely that shortest paths (or near-optimal paths) are often *spatially constrained* and have an inherent geometric structure to them. Indeed, very often, such paths do not deviate significantly from a line connecting the source and destination points and in such cases, it is possible to restrict the path-search to a very limited subset of the entire graph. This subset can be defined by a simple geometric container, such as a rectangular bounding box (Figure 4).

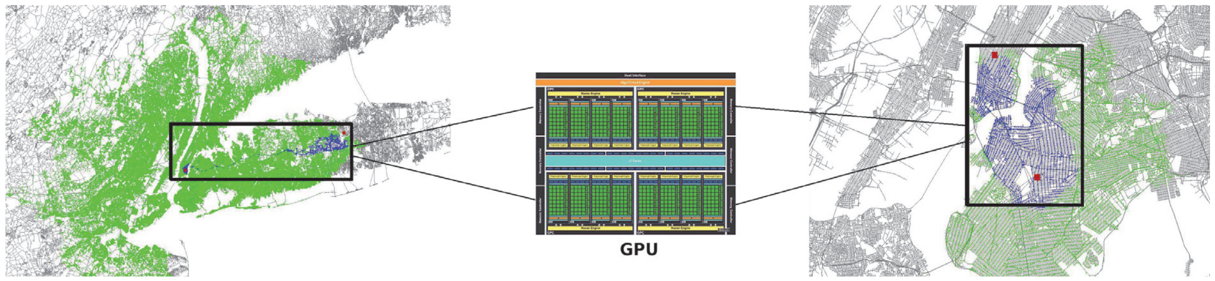

Figure 4: Only a small subset of the nodes in the network needs to be mapped to GPU memory for path computations

This property has important implications when considering implementing path computation problems on a GPU, which offers a great degree of parallelism, but only when (i) the problem data set fits reasonably within the memory constraints imposed by the GPU (the on-board GPU memory is significantly smaller than off-board RAM) (ii) the algorithm is implemented on structured, regular data that result in coalesced memory access patterns and finally (iii) the computation to communication ratio is high.

Efficient graph representations and graph exploration problems exhibit none of the previous three properties, and thus have been traditionally ill-suited to the GPU. However, some recent advances (including GPU programmability [5]) have shown how the GPU can offer significant performance benefits to these types of problems. In [3], Harish and Narayanan showed how graphs can be compactly represented in GPU memory for solving single-source shortest-path problems. Further, results in [3] also indicate that All-Pair Shortest-Path (APSP) problems – where shortest paths are computed between every source and destination pair – implemented via parallel single-source shortest path (SSSP) algorithms run significantly faster on the GPU compared to the classic Floyd-Warshall APSP algorithm [1]. While the authors in [3] consider GPU memory size to be a limiting factor in dealing with large graphs, from our observations on path characteristics that arise in real-world graphs (where we only need to deal with small sub-regions at a time) we see that in many cases, memory considerations need not be a limiting factor.

In simulating real-world scenarios, in numerous instances, sub-regions that can be mapped to GPU memory are frequently encountered. Parallel SSSP algorithms employing A^* heuristics are then deployed within the sub-region to speed up computations in such cases. It is important to note here that route computations on the GPU are meant to *accelerate* the execution of the application (the transportation simulator in this case) by offloading a significant amount of the computational load to the GPU; the GPU is not meant to be able to handle any and all routing computations. Obviously, there are instances when this approach might not be appropriate; for instance, when the source and destination are separated by a topological barrier (a lake or a mountain for instance) the resulting bounding-box region might be too constrictive and a path might not be found. Such cases are best handled on the CPU. Further, when the source and destination are far apart, the graph within the bounding box will be too large to fit into GPU memory. For this case, we are exploring multilevel coarsening strategies for graphs (where only the highways and major roads in the road network are taken into account) and applying some of the techniques described above.

3. ACKNOWLEDGMENTS

The author wishes to thank Shiva Kasiviswanathan for help with the routing experiments and the Laboratory Directed Research and Development (LDRD) program at the Los Alamos National Laboratory for funding this work.

4. REFERENCES

[1] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[2] R. M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, 1990.

[3] P. Harish and P. J. Narayanan. Accelerating large graph algorithms on the gpu using cuda. In *Proceedings of the 14th international conference on High performance computing*, HiPC'07, pages 197–208, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[5] nVidia. CUDA. `http://en.wikipedia.org/wiki/cuda`.

[6] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[7] R. Sedgewick and J. Vitter. Shortest paths in Euclidean graphs. *Algorithmica*, 1(1):31–48, 1986.

[8] S. Thulasidasan and S. Eidenbenz. Accelerating traffic microsimulations: A parallel discrete-event queue-based approach for speed and scale. In *Proceedings of the Winter Simulation Conference*, 2009.