GPU-Accelerated VLSI Routing Using Group Steiner Trees

Basileal Imana, Venkata Suhas Maringanti and Peter Yoon
Department of Computer Science
Trinity College Hartford CT
06106
{basileal.imana, venkatasuhas.maringanti, peter.yoon}@trincoll.edu

ABSTRACT
The problem of interconnecting nets with multi-port terminals in VLSI circuits is a direct generalization of the Group Steiner Problem (GSP). The GSP is a combinatorial optimization problem which arises in the routing phase of VLSI circuit design. This problem has been intractable, making it impractical to be used in real-world VLSI applications. This poster presents our initial work on designing and implementing a parallel approximation algorithm for the GSP based off an existing heuristic on a distributed architecture. Our implementation uses a CUDA-aware MPI-based approach to compute the approximate minimum-cost Group Steiner tree for several industry-standard VLSI graphs. Our implementation achieves up to 302x speedup compared to the best known serial work for the same graph. We present the speedup results for graphs up to 3k vertices. We also investigate some performance bottleneck issues by analyzing and interpreting the program performance data.

1. INTRODUCTION
A number of optimization problems with different application areas can be modeled by the GSP: given an undirected weighted graph $G = (V, E)$ and a family $N = \{N_1, \ldots, N_k\}$ of k disjoint groups of nodes $N_i \subseteq V$, find a minimum-cost tree which contains at least one node from each group N_i. One of such problems is the global routing phase in VLSI design.

This poster presents a GPU-based algorithm for GSP and its implementation based on Depth Bounded Steiner Tree Approximation heuristic [1].

2. A GPU-ACCELERATED APPROACH

2.1 Metric Closure on GPU
Given a graph G, replace G by its metric closure, i.e., compute all-pair shortest paths (APSP) for the given graph and replace every edge cost with the corresponding minimum u-v path cost. For APSP, we use Blocked Floyd-Warshall algorithm similar to [2] on a GPU. G is then replaced with the metric closure.

2.2 Group Steiner Heuristic
Using G from the previous step, we construct a minimum-cost Group Steiner tree by launching multiple processes using CUDA-aware MPI. A *d-star* is defined to be a rooted tree of depth at most d [1]. With every vertex in G as a potential root r, we construct *2-star* trees as illustrated in Figure 1.

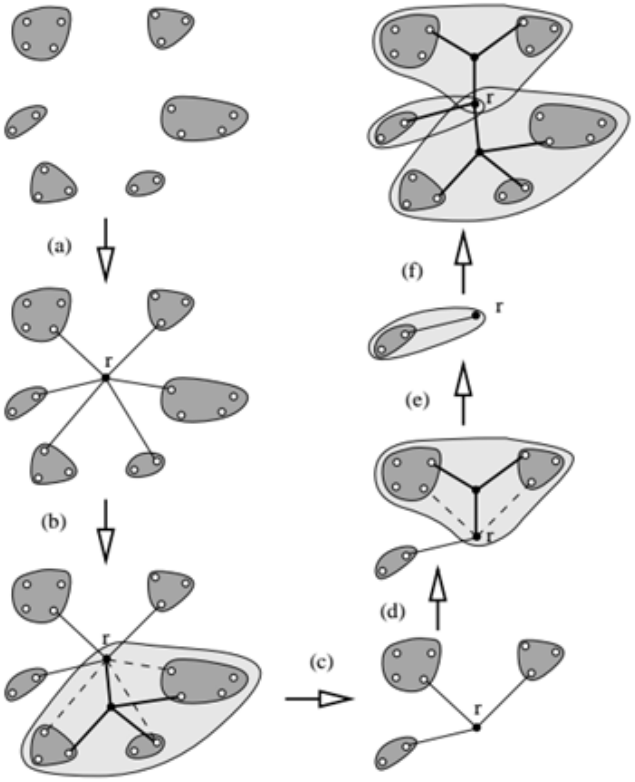


Figure 1. Steps (a) through (f) of the GSP Heuristic [1]

As shown in Figure 1, (a) construct a rooted *1-star*, i.e. a tree of depth 1 where all leaves are ports, one from each group. A root, an intermediate node and a set of groups together form a *partial-star*. Steps (c) though (e) compute such partial-stars until all groups are spanned. Step (f) combine all partial-stars computed in the previous step to form a *2-star* tree for the given root r.

Out of all such rooted 2-star trees obtained, the one with the minimum cost is the final solution.

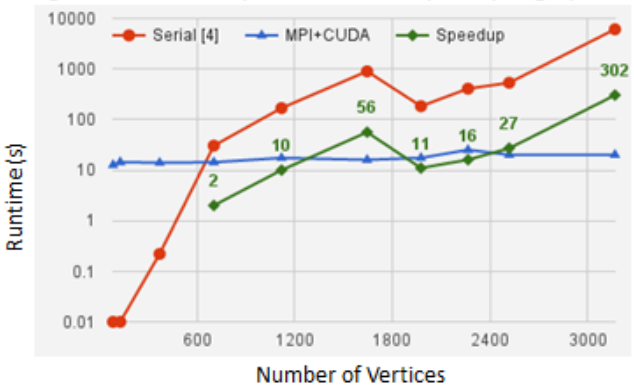
2.3 Work Distribution
The master process performs the procedure in section 2.1 and then broadcasts the modified graph to all slave processes. Each vertex (a potential root) is then mapped to a slave, whose task is to perform the procedure in section 2.2 and communicate the result back to the master. After receiving all results, the master then performs reduction to compute the overall solution.

3. PERFORMANCE EVALUATION
All performance experiments were run on Blue Waters supercomputer which uses a Cray XE6/XK7 system. We tested both our serial and parallel implementations using Wire Routing

Problem (WRP) instances from [3]. We compared our approximate solutions for problem instances with optimal solutions from [4]. Our analysis shows the cost error is less than 1% for all the graphs we tested on.

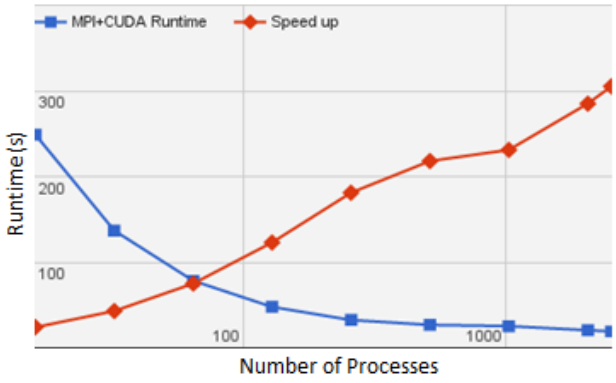
Figure 2 shows that our implementation outperforms the best known serial work [4]. Our algorithm achieves a speed-up for bigger graph sizes (>600 vertices), reaching up to 302x for a graph with 3168 vertices.

Figure 2. Serial time, parallel time and speedup vs graph size



A common task in HPC is measuring the *scalability* (also referred to as the *scaling efficiency*) of an application. Figure 3 indicates that our implementation is highly scalable with respect to the number of processes.

Figure 3. MPI+CUDA run time and speed-up vs processes



Due to the highly dynamic nature of the problem, our implementation exhibits some load imbalance and irregular memory access issues. To that end, we plan to optimize the work distribution scheme and memory consumption of our implementation.

4. REFERENCES

[1] Helvig C.S., Robins, G. and Zelikovsky, A. New Approximation Algorithms for Routing with Multi-Port Terminals. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *19*(10), 1118-1128.

[2] Lund, B. D., and Smith, J. W. A multi-stage CUDA Kernel for Floyd-Warshall. CoRR abs/1001.4108 (2010).

[3] Koch, T., Martin, A. and Voß, S. SteinLib: an updated library on Steiner tree problems in graphs. in Cheng, X. and Du, D.Z. eds. *Steiner Trees in Industry*, Springer US, Berlin, 2001, 285–326.

[4] Polzin, T., Vahdati, S. The Steiner tree challenge: An updated study, *11th DIMACS Implementation Challenge*. Retrieved July 06, 2015, from Princeton University: http://dimacs11.cs.princeton.edu/papers/PolzinVahdatiDIMACS.pdf