Finding Vertex Cover: Acceleration via CUDA

Yang Liu, High Performance Research Computing, Texas A&M University

Jinbin Ju, Department of Electrical & Computer Engineering, Texas A&M University

Derek Rodriguez, Department of Electrical & Computer Engineering, Texas A&M University

Introduction

The Vertex Cover Problem

- Classical NP-complete problem (one of the twenty-one Karp's NP-complete problems)

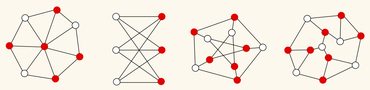


Figure 1— Examples of Vertex Covers

Fixed-Parameter Tractable Algorithm (FPT)

- Parameter k (positive integer) and input size n
- Determine whether a vertex cover of at most k vertices exists or not in time f(k)p(n) where f(k) is independent of n and p(n) is polynomial of n

Our Approach

- Distribute and synchronize computation between CPU and GPU (graph decomposition)
- Synchronize threads in a block
- Apply reduction rules to vertices with degrees greater than k and vertices with degree one

Results

- Tested on graphs created from biological data
- Current implementation is up to 11 times faster than serial program

Purpose & Application

Vertex Cover

Given a graph G, a vertex cover of G is a vertex subset C such that every edge of G is incident to a vertex in C. Given a graph G and a positive integer k, determining whether a vertex cover of at most k vertices is one of Karp's 21 NP-complete problems (1972). Within the set of all vertex covers, there exists a minimum vertex cover such that the cardinality of this cover is less than or equal to the cardinality of all other vertex covers of G. Currently, the best exact algorithm to find a minimum vertex cover is of complexity $O(1.2018^n)$, which is highly impractical for large datasets.

Parameterized Vertex Cover

From the perspective of parameterized complexity, the parameterized vertex cover problem is to find a vertex cover of at most k vertices if such a vertex cover exists or to return 'No' otherwise. The current fastest algorithm for the parameterized problem is of complexity $O(1.2738^k+n)$.

Application

An application of the parameterized problem is to solve the phylogeny problem. Data from NCBI is downloaded and preprocessed to generate graphs. Those graphs have up to 1000 vertices. The range of k is from 883 to 987. Algorithms have been implemented on clusters to find vertex cover of at most k vertices for those graphs.

GPGPU Implementation

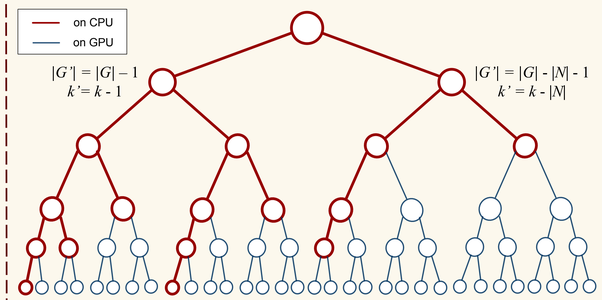
GPGPUs are successful in improving performance of programs and algorithms. However, graph algorithms are not easily implemented on GPGPUs with significant performance speedup. We are investigating the challenges and opportunities for implementing those algorithms on GPGPUs for the parameterized vertex cover problem. Such investigations will result in new perspectives and methods on algorithm engineering of complex algorithms.

Hardware

	CPU	GPU
	Intel E5-2670 v2	Nvidia Telsa K20m
Number of Cores	10	2496
Peak Performance	400 GFLOPS	3.52 TFLOPS
Memory	64 GB	5 GB

Techniques

Figure 2 — Branch Searching Process



Branching Process

- Pick a vertex v (max degree)
- Two branches
  - Put v in vertex cover (left branch)
  - Put v's neighbors in vertex cover (right branch)
- Branch recursively until
  - a vertex cover of at most k vertices is found
  - or no such vertex covers exist
- Imbalanced Search Tree

CPU Program

- Configurable number of blocks
  - Around 90
- Configurable number of threads per block
  - 32
- Degree Arrays per block are in shared memory

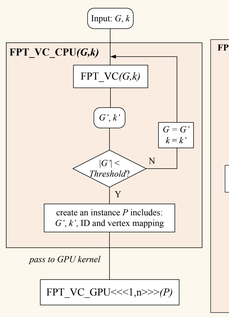


Figure 3a — CPU Program Flow Chart

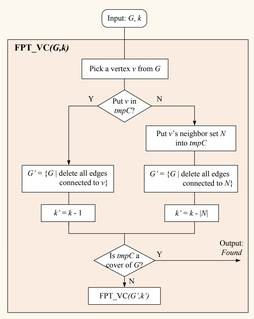


Figure 3b — GPU Program Flow Chart

GPU Program

- No dynamic change of input graph
- Use degree arrays and original input graph to infer subgraph information
- Adopt reduction rules

Distribution of Computation

- Important for performance
- Controlled by thresholds
  - Non-negative integer t (now is 80).
  - Subgraphs with no more than t vertices are sent to GPU for processing
  - Adjustable by input to control the distribution of computation
  - For optimal performance, t is different for different input graphs

Reduction Rules

- No branching for vertices with:
  - degree greater than k
  - degree one
  - degree two if max degree is two

Synchronization of Computation

- Copy subgraphs to GPU
  - CUDA memory asynchronous copy on separate stream
  - Concurrent kernel execution on separate stream
- Poll GPU states
  - Pinned mapped memory
- Synchronization among threads in a block
  - Shared memory

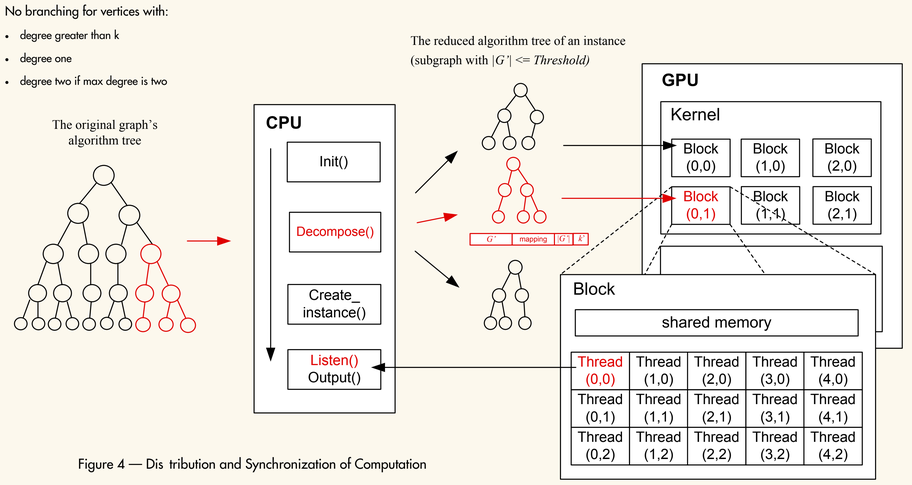


Figure 4 — Dis tribution and Synchronization of Computation

Results & Conclusions

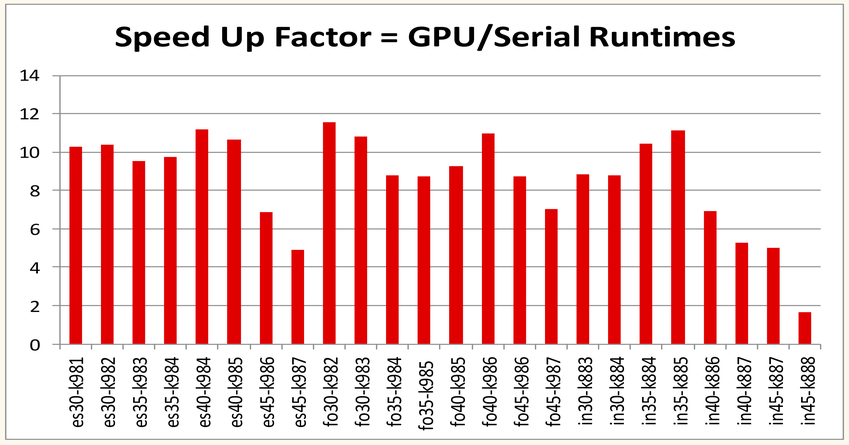


Figure 5 — Speedup of CPU+GPU Program Over Serial Program

Table 1 — Program Running Times

Graph-k	Serial (s)	CPU+GPU (s)
es30-k981	11639.52	1132.826
es30-k982	3489.943	335.3726
es35-k983	3132.381	328.5616
es35-k984	325.9834	33.4564
es40-k984	847.3066	75.6706
es40-k985	112.8362	10.6178
es45-k986	295.173	42.8384
es45-k987	6.5156	1.3328
fo30-k982	31225.86	2704.92
fo30-k983	1777.257	164.0328
fo35-k984	7088.113	806.7852
fo35-k985	276.9754	31.6658
fo40-k985	2202.807	238.1776
fo40-k986	156.3932	14.2208
fo45-k986	573.7118	65.595
fo45-k987	44.3914	6.3102
in30-k883	1569.43	177.0106
in30-k884	365.643	41.7094
in35-k884	425.896	40.842
in35-k885	403.413	36.3172
in40-k886	156.062	22.4966
in40-k887	16.109	3.057
in45-k887	74.8566	14.8912
in45-k888	0.9754	0.5964

Configurations

- Vertex threshold: 90
- Number of blocks on GPU: 80

Summary

- For 9 out of 26 input graphs, the speed up factor is greater than 10
- For 22 out of 26 input graphs, our program has speed up factor of more than 6
- Only one graph has speedup factor less than 1.

Future Research

Multiple GPU

- Our current experiments have not shown speedup when two GPUs are used.

MPI + GPU

- For difficult graphs, multiple CPUs + GPUs are necessary
- Load balancing is very important

Redesign of Algorithm

- Important to use threads in a block more efficiently

Dynamic Configuration

- Our profiling results show that for some graphs, the bottleneck is on CPU, while for some other graphs, the bottleneck is on GPU
- Different input graphs demand different configurations for optimal performance. That is, for different graphs, the vertex threshold and number of blocks likely have different optimal values to achieve maximized speedup.

Acknowledgements & References

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No 1442734. This work was supported in part by the computing resources and technical support of High Performance Research Computing at Texas A&M University.

We would like to extend thanks to Michael Langston and Gary Rogers for providing us with the graph data used throughout the bench-marking process. Additionally, we would like to thank Robert Crovella for his helpful discussion on CUDA.

References

[1] R. Kabbara. A Parallel Search Tree Algorithm for Vertex Covers on Graphical Processing Units. Master Thesis, Lebanese American University, 2013

[2] D. Weerapurag, J. Eblen, G. Rogers, and M. Langston. Parallel Vertex Cover: A Case Study in Dynamic Load Balancing. Pp 25-32. Proceedings of the Ninth Australasian Symposium on Parallel and Distributed Computing (AusPDC), 2011, Perth, Australia.