Finding Vertex Cover: Acceleration via CUDA

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

Jinbin Ju, Electrical Engineering, Texas A&M University

Derek Rodriguez, Computer Engineering, Texas A&M University

Introduction

The Vertex Cover Problem

- Classical problem NP-Complete Problem (one of the twenty-one Karp's NP-Complete Problems)

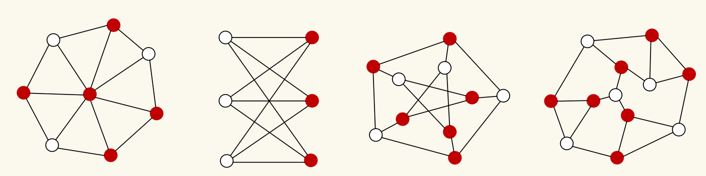


Figure 1— Examples of Vertex Covers

Fixed-Parameter Tractable Algorithm (FPT)

- Parameter k (positive integer)
- Determine whether a vertex cover of at most k vertices exists or not

Our Approach

- Distribute computation between CPU and GPU (graph decomposition)
- Synchronize computation between CPU and GPU
- Synchronize threads in a block
- Apply reduction rules for vertices with more than k edges, and degree 1 vertices

Results

- Tested on graphs produced from biological data
- Current implementation achieves more than 5x speedup

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. Within the set of all vertex covers, there exists a minimum vertex cover such the cardinality of this cover is less than or equal to the cardinality of all other vertex covers of G. Resolving the minimum vertex cover of a graph is one of Karp's 21 NP-complete problems (1972). 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 another perspective, the parameterized vertex cover problem is to find a vertex cover of at most k vertices. The fastest documented 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. Algorithms have been implemented on clusters to find vertex cover of at most k vertices for those graphs.

GPGPU Implementation

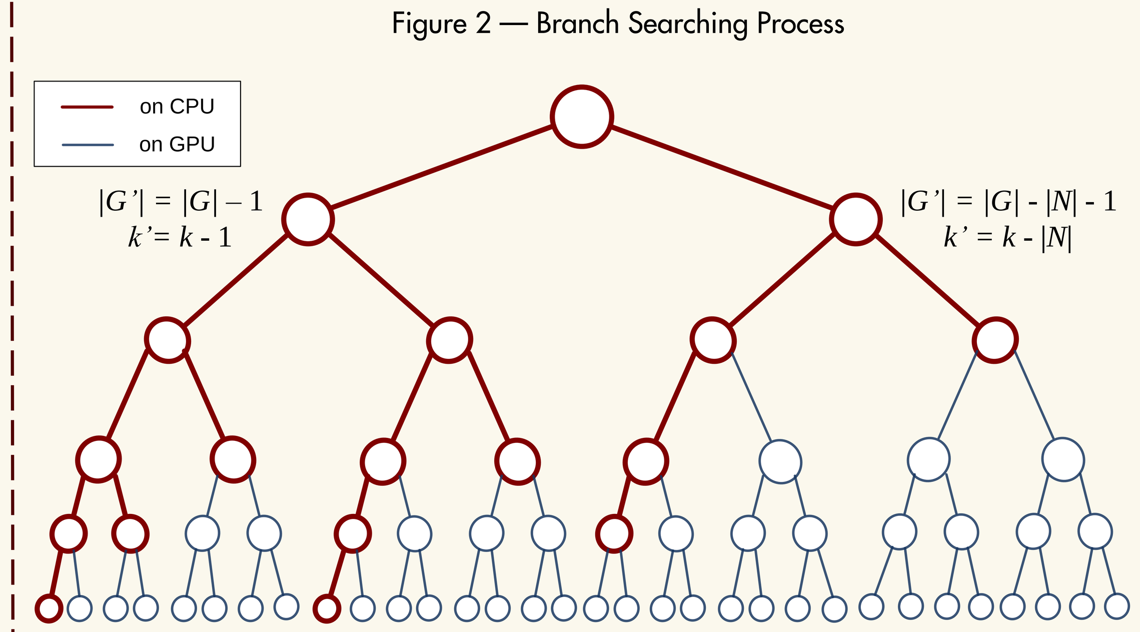
GPGPUs are successful in improving performance of programs and algorithms. However, graph algorithms are not easy to be implemented on GPGPUs with significant performance speedup. We are investigating the challenges and opportunities for implementing those algorithms on GPGPUs for the parameterized vertex 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

GPU Program

- Configurable number of blocks
  - Around 60
- Configurable number of threads per block
  - 32
- Degree Arrays per block are in shared memory

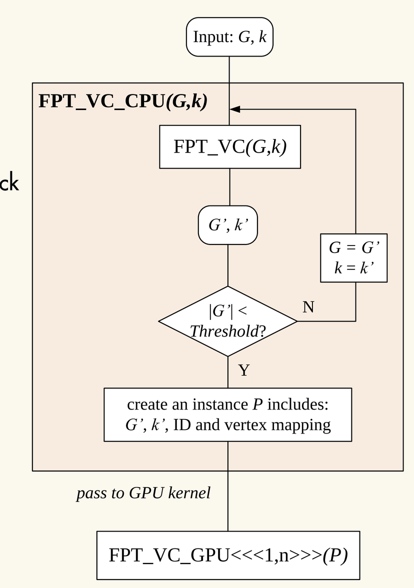


Figure 3a — GPU Program Flow Chart

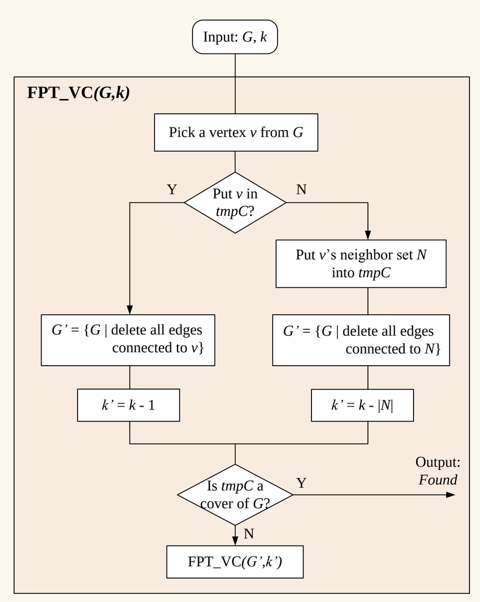


Figure 3b — CPU Program Flow Chart

CPU 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
  - 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 more than k neighbors
  - With degree 1
  - With degree 2 if max degree is 2

Synchronization of Computation

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

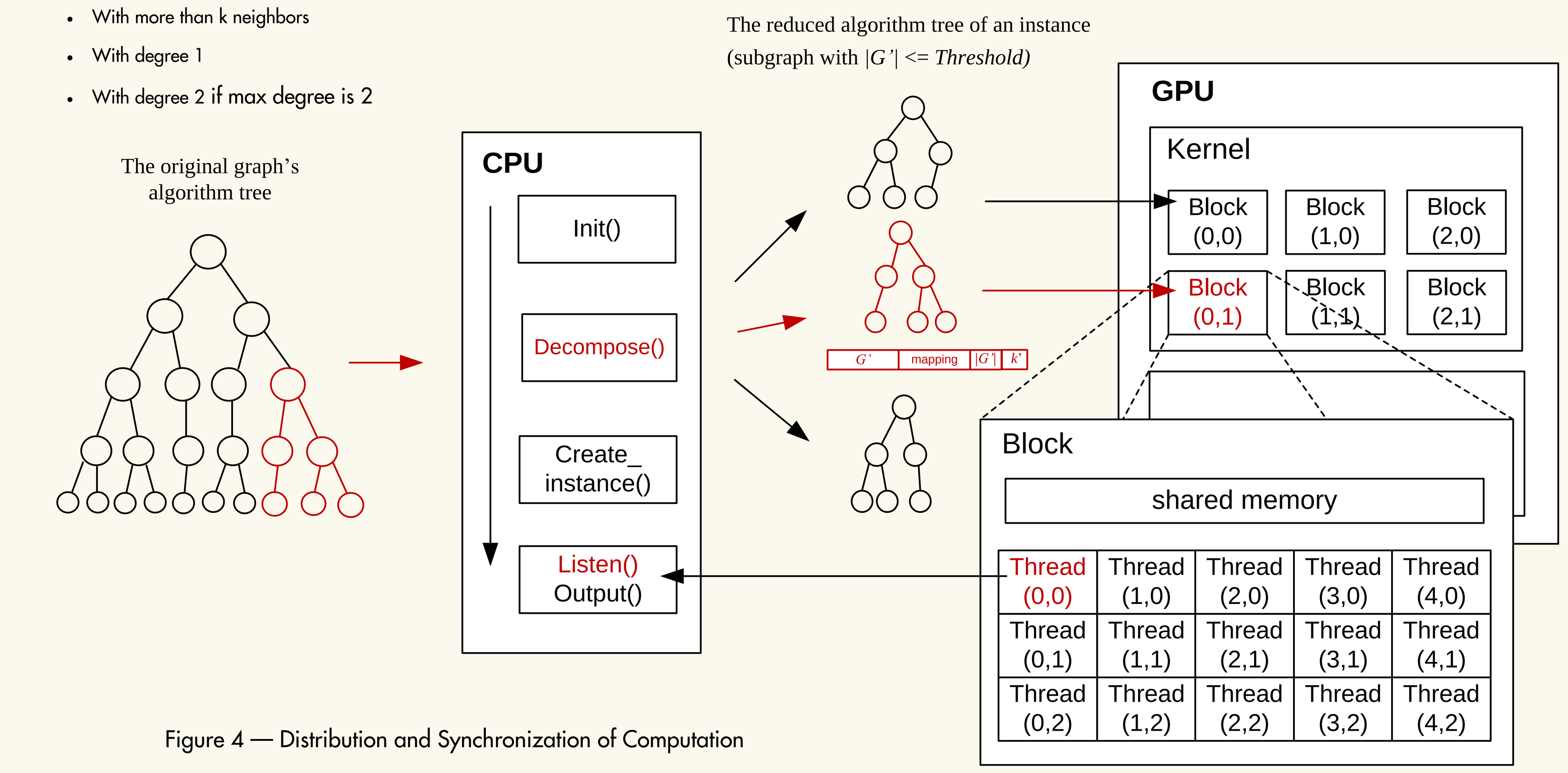


Figure 4 — Distribution and Synchronization of Computation

Results & Conclusions

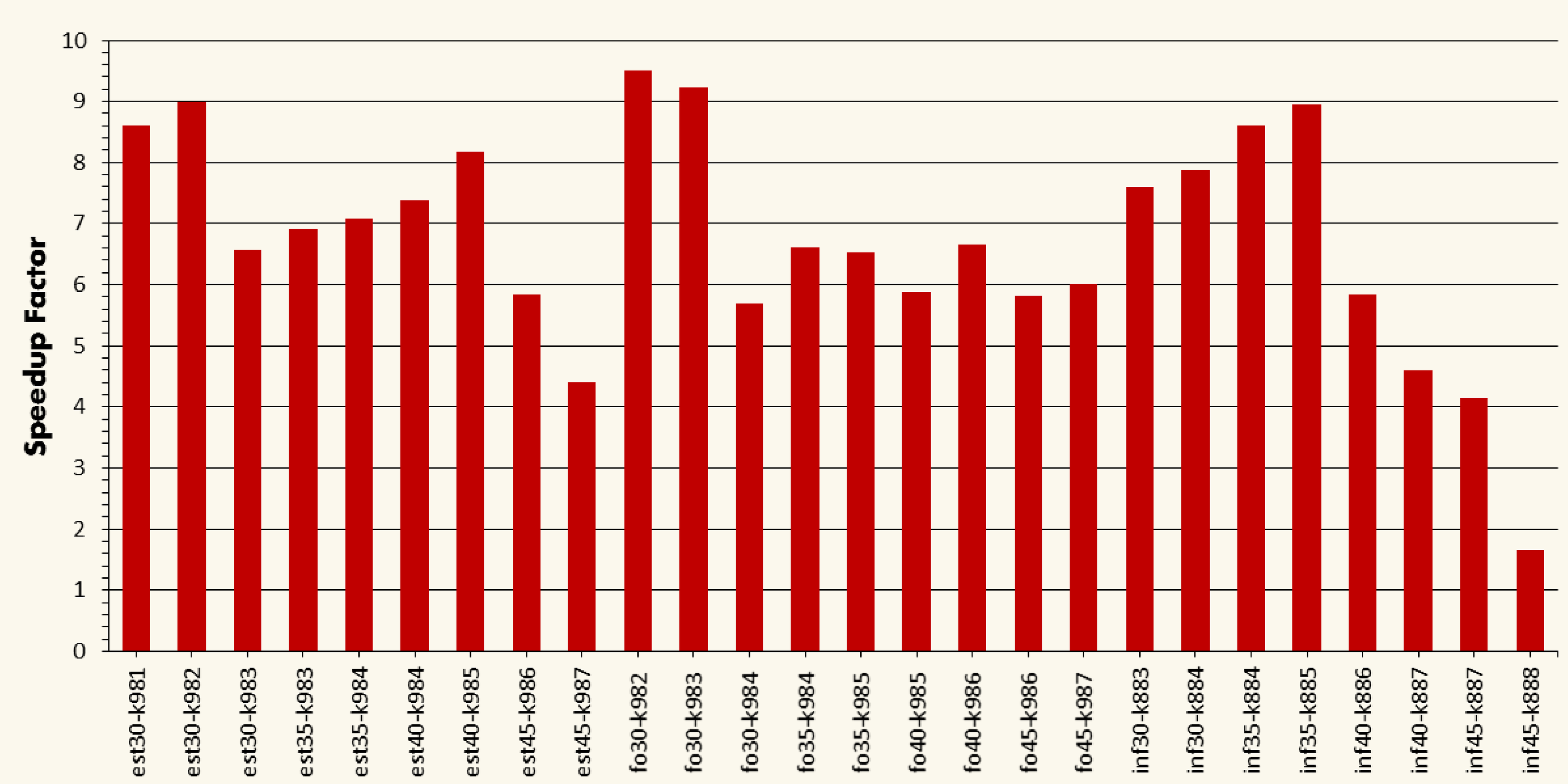


Figure 5 — Speedup of CPU+GPU Program Over Serial Program

Graph-k	Serial (s)	CPU+GPU (s)
est30-k981	11085	1286.8476
est30-k982	3336.282	370.7691
est30-k983	6.3398	0.96515
est35-k983	2990.1624	432.88845
est35-k984	312.4582	44.14175
est40-k984	808.8308	109.69125
est40-k985	108.4648	13.27725
est45-k986	281.0682	48.1827
est45-k987	6.412	1.4582
fo30-k982	29694.2858	3122.20845
fo30-k983	1693.952	183.39125
fo30-k984	6.412	1.1273
fo35-k984	6733.2666	1016.94735

Graph-k	Serial (s)	CPU+GPU (s)
fo35-k985	264.1242	40.5193
fo40-k985	2097.998	357.0809
fo40-k986	149.4656	22.48525
fo45-k986	544.4594	93.7265
fo45-k987	42.2078	7.02375
inf30-k883	1501.6508	197.52165
inf30-k884	351.2456	44.5824
inf35-k884	406.5822	47.23765
inf35-k885	386.9278	43.1862
inf40-k886	148.9256	25.4885
inf40-k887	15.463	3.35825
inf45-k887	71.5358	17.22985
inf45-k888	0.9304	0.5575

Table 1— Program Running Times

Summary

- For 11 out of 26 input graphs, the speed up factor is greater than 7
- For 22 out of 26 input graphs, our program has speed up factor of more than 5

Future Research

Profiling and Optimization

- Collect time required for polling states in GPU
- Collect time required for memory copy between CPU and GPU

Multiple GPU

- Our current experiments showed around 30% slowdown if two GPUs are used

MPI + GPU

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

Redesign of Algorithm

- Important to use threads in a block more efficiently

Dynamic Configuration

- Different input graphs demand different configurations for optimal performance

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 benchmarking 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.