Improving Parallelism of Breadth First Search (BFS) Algorithm for Accelerated Performance on GPUs

Hao Wen*, Wei Zhang

Department of Computer Engineering and Computer Science

University of Louisville

* Graduated from ECE, VCU

UNIVERSITY OF LOUISVILLE

J.B. SPEED SCHOOL OF ENGINEERING

INTRODUCTION

- Graph processing operates on a large volume of highly connected data. Real-world applications of graph processing include:
  - Social network
  - Digital maps
  - Webpage hyperlinks
  - VeryLarge-Scale Integration (VLSI) layout of integrated circuit (IC), etc.

INTRODUCTION

- Breadth-First Search (BFS) serves as a basic primitive for many higher-level graph analysis applications, e.g.,the shortest path problem.

- BFS algorithm searches the graph layer by layer. Vertices in the same layer can be processed in parallel, which makes it suitable for GPU computing

INTRODUCTION

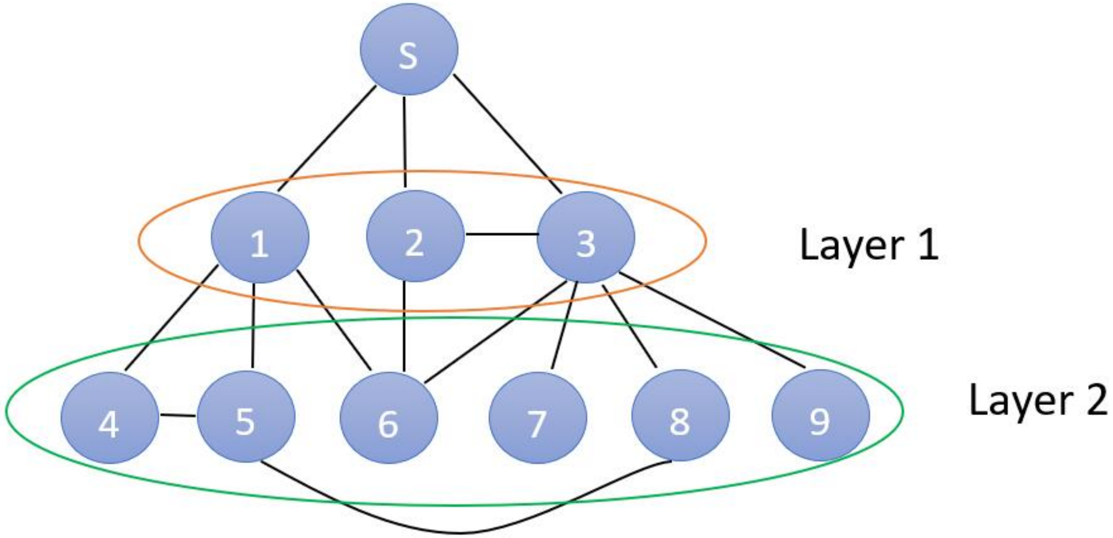
- Problem with BFS on GPU?
  - the irregularity of the graph makes the BFS difficult to be executed on GPUs efficiently.
  - GPU threads working on high-degree nodes take much longer time than the threads working on low degree nodes
  - many threads may be under-utilized due to the limited parallelism

INTRODUCTION

- Previous works try to solve the problem by modifying GPU execution models or taking advantage of CPU-GPU heterogeneous computing for fine-grained task management
- we propose to address this issue from its origin, i.e., virtually changing the graph itself to eliminate the irregularity (**Virtual BFS (VBFS)**).

Background

- BFS algorithm on GPU



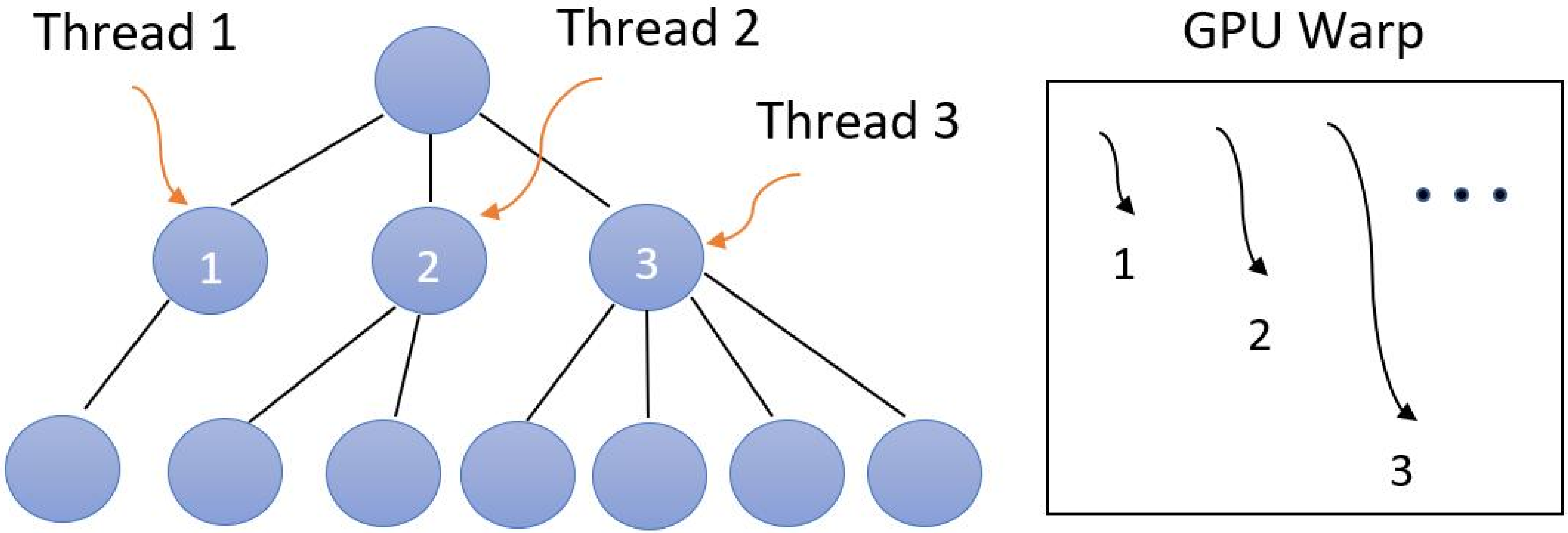
Algorithm 1 BFS algorithm on GPU

```
G.init(S);
Allocate GPU threads and launch GPU kernel;
Kernel1
Compute GPU thread ID tid;
if GraphMask[tid] then
    Clear GraphMask[tid];
    for All the neighbors of nodes[tid] do
        if This neighbor is not visited then
            Mark this neighbor visited;
            Calculate the distance of this neighbor;
            Set Corresponding UpdateGraphMask;
        end if
    end for
end if
End Kernel1

Kernel2
Compute GPU thread ID tid;
if UpdateGraphMask[tid] then
    GraphMask[tid]=UpdateGraphMask[tid];
    Clear UpdateGraphMask[tid];
end if
End Kernel2
```

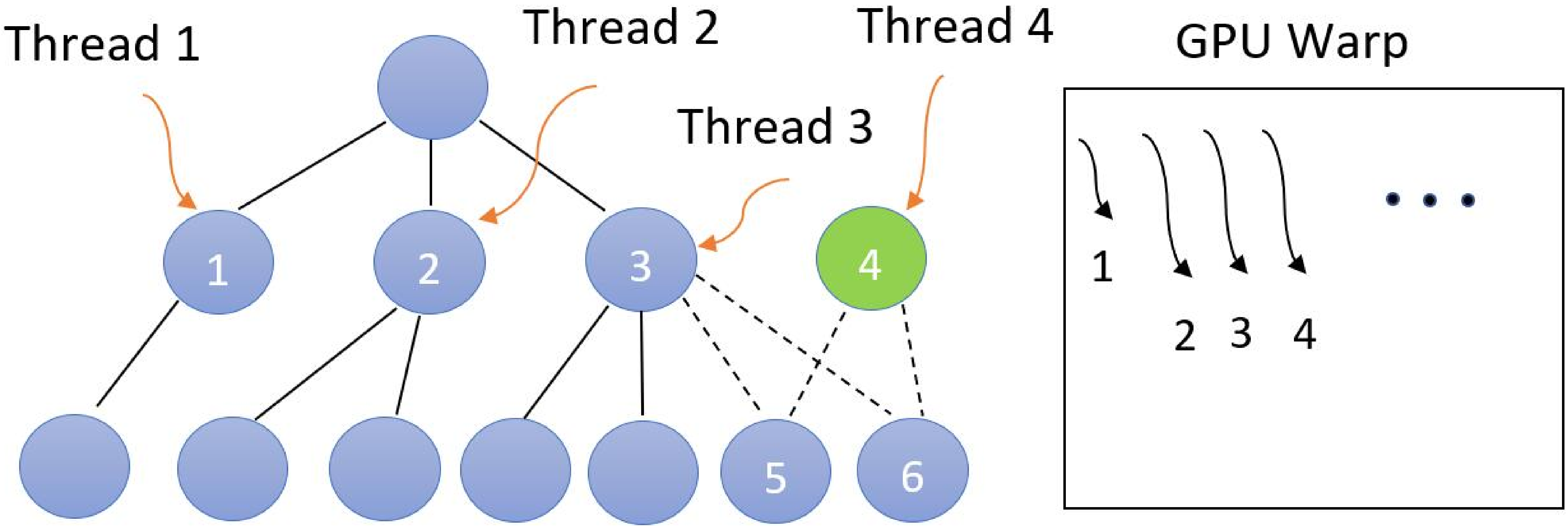
Motivation

- In the graph processing like BFS, the nodes are distributed to threads for execution. Graph irregularity leads to workload imbalance.



Motivation

- The idea of adding virtual vertex

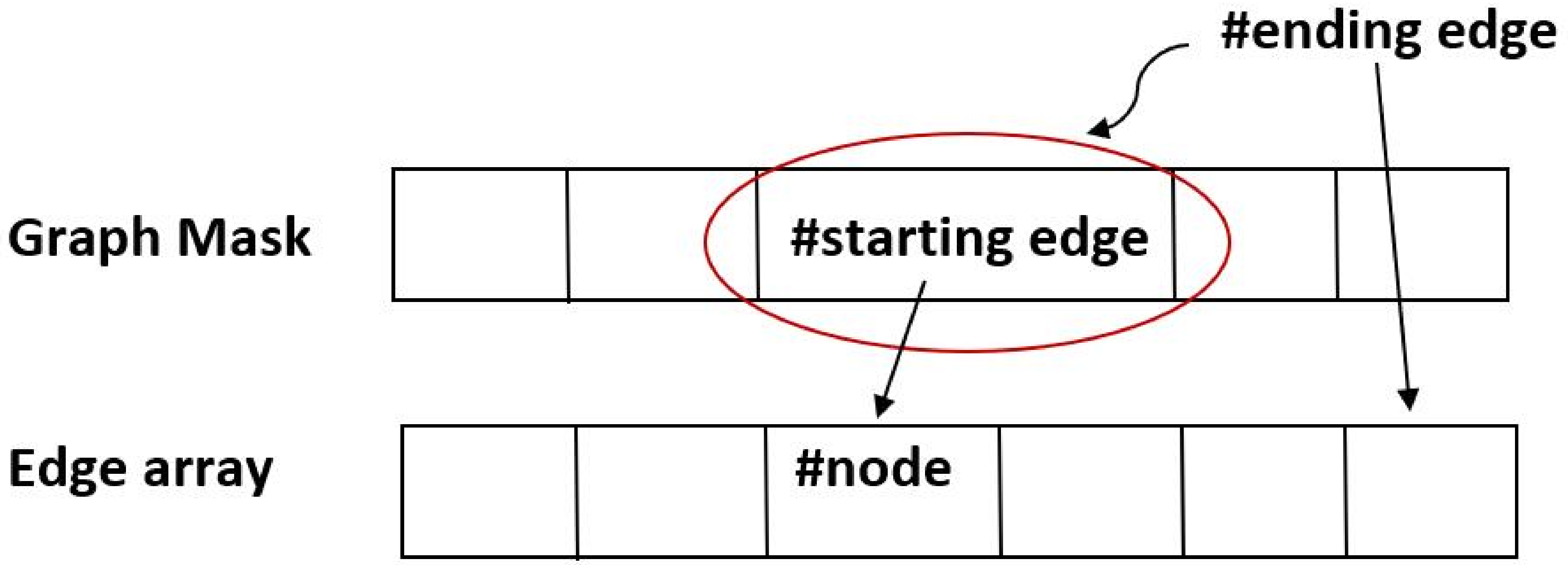


The rules of adding virtual vertices

- We define a group size of K edges. Virtual vertices are only added when the degree of outgoing edges of a node is greater than K
- If the degree of a node is exactly a multiple of K, it will be divided into groups of equal number of edges. Otherwise, there will be a group with residual edges less than K.

Representation of virtual vertices

- we do not need to have a new data structure to store these vertices



Correctness of the VBFS

- If a node has N (N>k) neighbors, we call the node the owner of the N neighbors. After group division, the only thing changed is that the N neighbors have more virtual owners besides the original owner.

Correctness of the VBFS

- If there is a path originally from vertex v1−>owner−>vertex v2, there must conceptually exist a path from vertex v1−>virtual owner−>vertex v2, so adding virtual vertices does not impact the connectivity.

Correctness of the VBFS

- The distance of v2 is also not affected since the virtual vertices can share the distance value of the original owner. Actually, adding virtual vertices is done layer by layer. The distance value propagation is synchronized by a global layer number.

GPU implementation of VBFS

Algorithm 2 $VBFS$ algorithm on GPU

```
G.init(S);
if the degree of source node > K then
    Divide the degree of source node into groups;
    Set corresponding GraphMask;
end if
Allocate GPU threads and launch GPU kernel;
Kernel1
Compute GPU thread ID tid;
if GraphMask[tid] then
    for All the neighbors represented by GraphMask[tid] do
        if This neighbor is not visited then
            Mark this neighbor visited;
            Calculate the distance of this neighbor;
            if the degree of this neighbor > K then
                Divide the degree of this neighbor into groups;
                Set corresponding UpdateGraphMask;
            end if
        end if
    end for
    Clear GraphMask[tid];
end if
End Kernel1

Kernel2
Compute GPU thread ID tid;
if UpdateGraphMask[tid] then
    GraphMask[tid]=UpdateGraphMask[tid];
    Clear UpdateGraphMask[tid];
end if
End Kernel2
```

Experimental Environment

- We use GPGPU-sim [3] to implement and evaluate our algorithm

GPGPU-Sim CONFIGURATION

Number of SMs	15
Size of L1 data cache per SM	48KB
L1 & L2 data cache block size	128B
L1 data cache associativity	4
Size of shared memory per SM	16KB
Size of L2 cache	768KB
L2 data cache associativity	8
Core clock frequency	700MHz

Experimental Environment

- We evaluate the VBFS on six graphs whose the number of nodes ranges from 128 to 4096. Each graph has two versions, a dense version and a sparse version.

GRAPHS INFORMATION

	Nodes	Edges(dense)	Edges(sparse)
Graph0	128	7750	2874
Graph1	256	29586	10538
Graph2	512	125120	26650
Graph3	1024	517990	54410
Graph4	2048	2028368	167440
Graph5	4096	5315733	371420

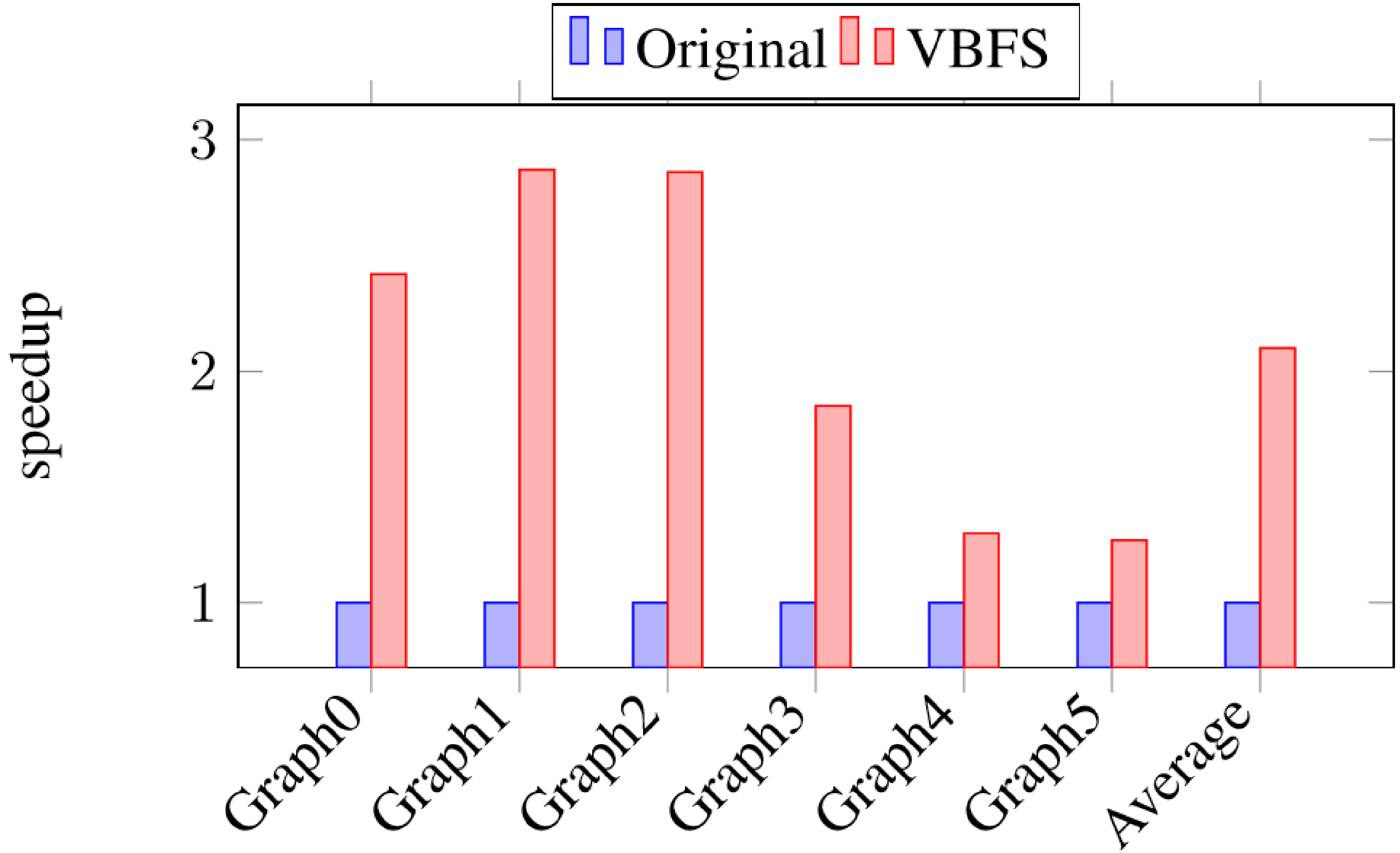
Experimental Environment

- The performance is normalized to the baseline GPU implementation(simulation cycles)
- Energy results are measured by GPUwattch.
- Energy Delay Product (EDP) is calculated as follows:

$$EDP = energy * delay = power * (delay)^2$$

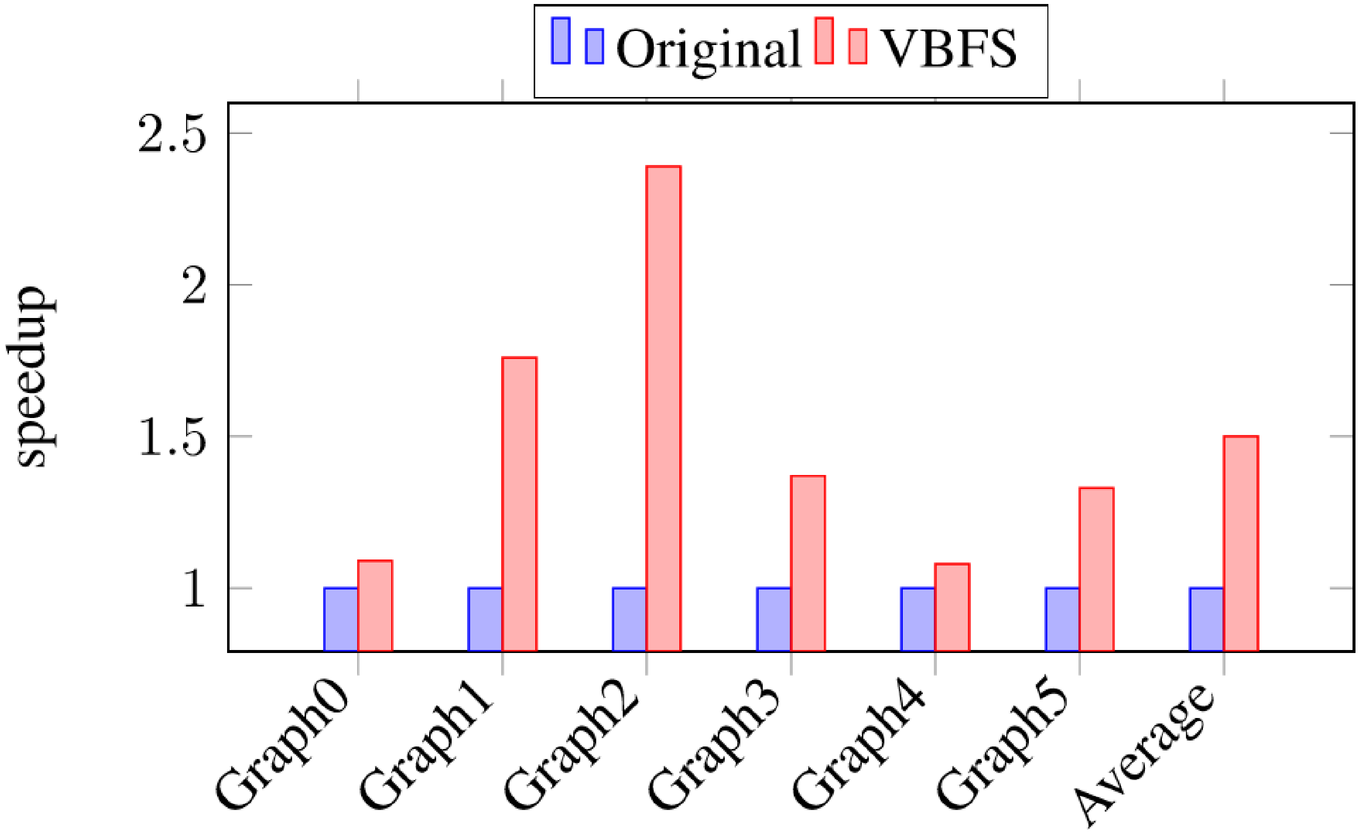
Results

- Performance comparison of the original BFS on GPU and VBFS on dense graphs



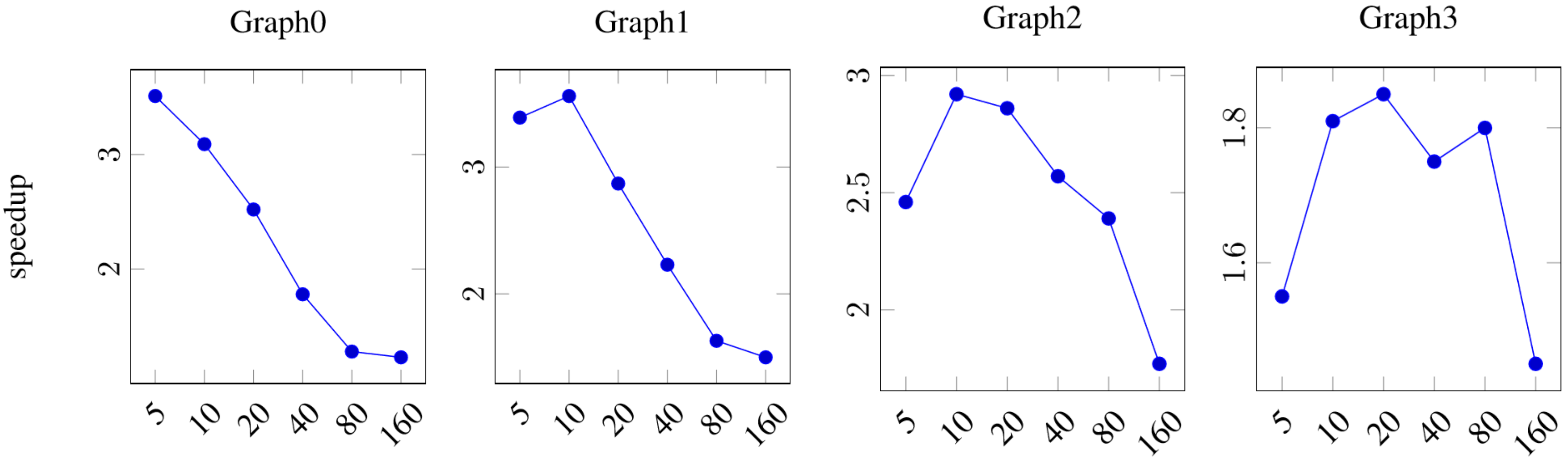
Results

- Performance comparison of the original BFS on GPU and VBFS on sparse graphs



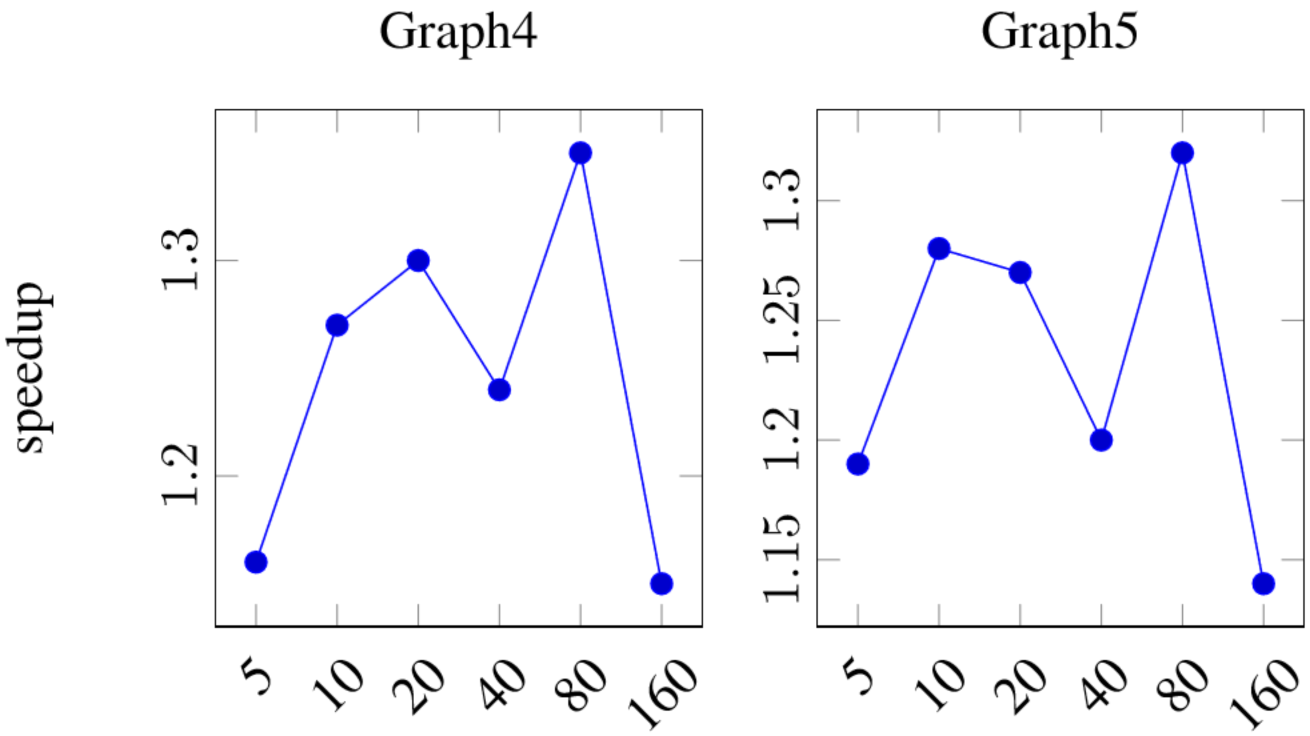
Results

- Impact of group size on the performance



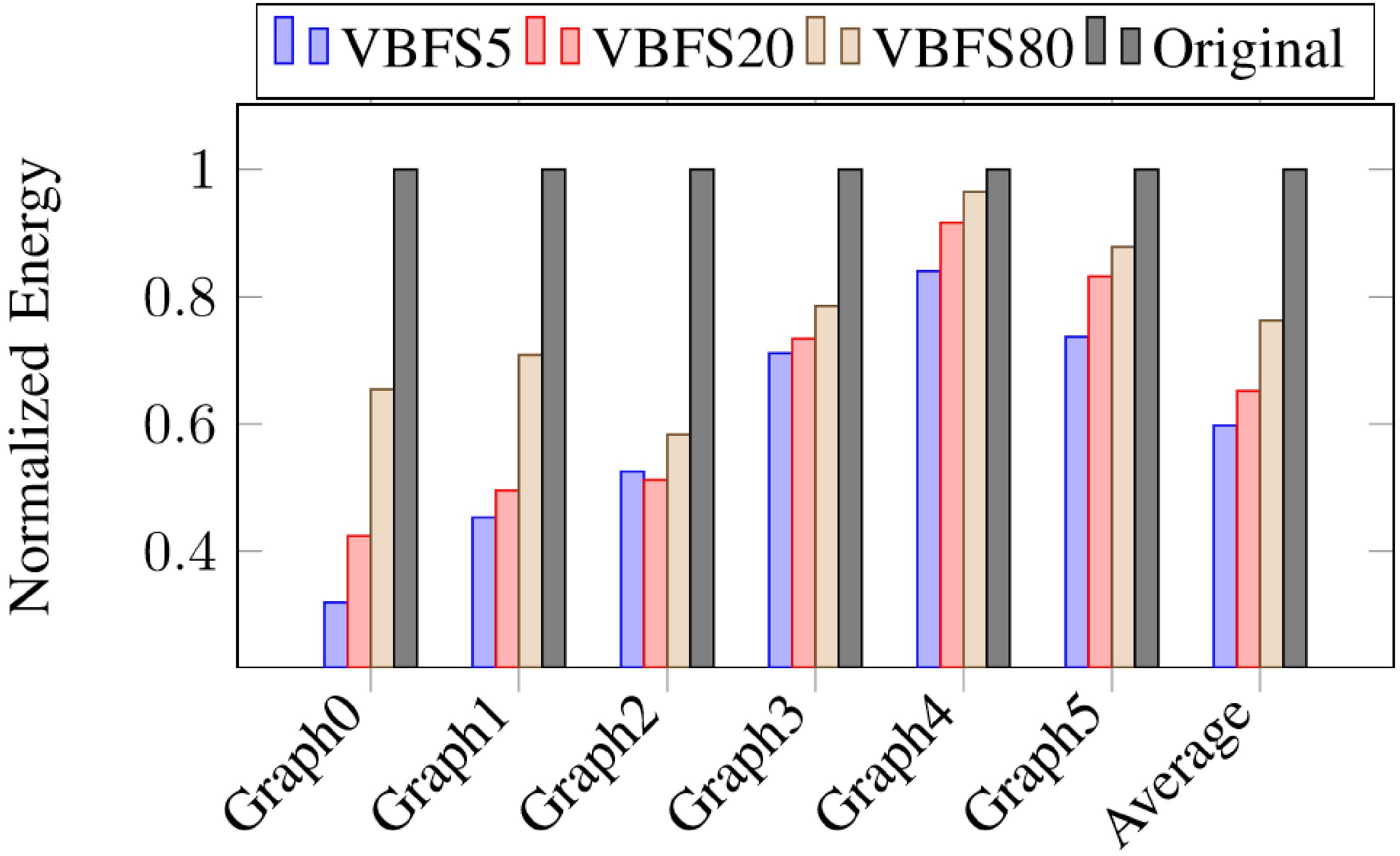
Results

- Impact of group size on the performance



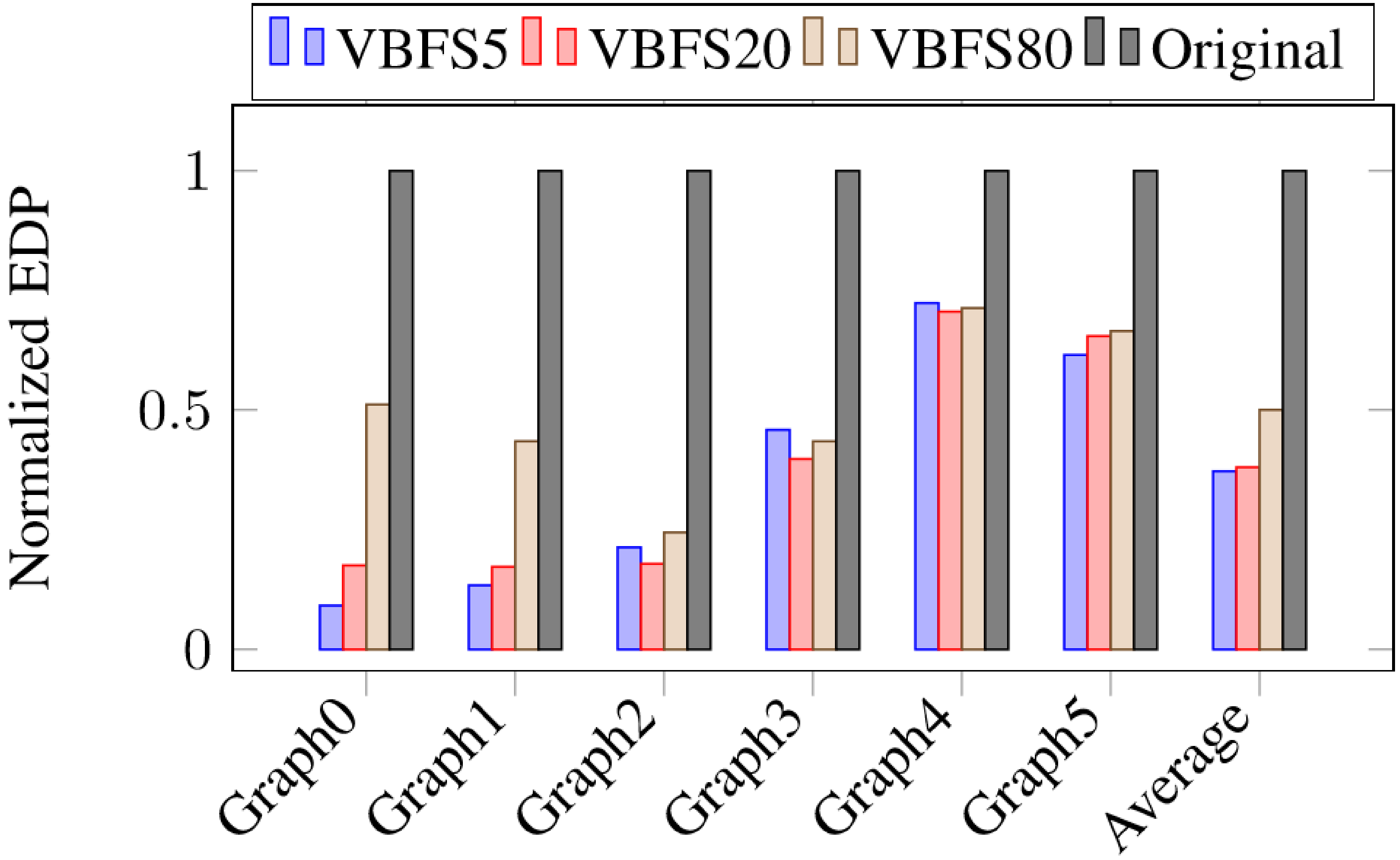
Results

- Energy



Results

- Normalized Energy Delay Product (EDP) of VBFS



Thank you!

UNIVERSITY OF LOUISVILLE

J.B. SPEED SCHOOL OF ENGINEERING