Understanding the SIMD Efficiency of Graph Traversal on GPU*

Yichao Cheng, Hong An, Zhitao Chen, Feng Li, Zhaohui Wang, Xia Jiang, and Yi Peng

University of Science and Technology of China, Hefei, China
yichao@mail.ustc.edu.cn, han@ustc.edu.cn, {czthack, fli186, wzhustc, jiangxia, peng1990}@mail.ustc.edu.cn

Abstract. Graph is a widely used data structure and graph algorithms, such as breadth-first search (BFS), are regarded as key components in a great number of applications. Recent studies have attempted to accelerate graph algorithms on highly parallel graphics processing unit (GPU). Although many graph algorithms based on large graphs exhibit abundant parallelism, their performance on GPU still faces formidable challenges, one of which is to map the irregular computation onto GPU's vectorized execution model.

In this paper, we investigate the link between graph topology and performance of BFS on GPU. We introduce a novel model to analyze the components of SIMD underutilization. We show that SIMD lanes are wasted either due to the workload imbalance between tasks, or to the heterogeneity of each task. We also develop corresponding metrics to quantify the SIMD efficiency for BFS on GPU. Finally, we demonstrate the applicability of the metrics by using them to profile the performance for different mapping strategies.

Keywords: BFS, GPU, irregular computation, graph topology, SIMD underutilization, SIMD efficiency

1 Introduction

Many practical applications (e.g., electronic design automation, geographical information, social networking and intelligent transportation systems) need to explore large-scale data sets which are represented by graphs. Recent research [13, 18, 19] has shown that there is abundant runtime parallelism in graph applications.

Graphics processing unit (GPU) has recently become a popular parallel platform for general computing due to its massive computational power and relatively low costs. Although many GPU-implemented applications have been

* We thank Xiaoqiang Li, Haibo Zhang and Tao Wang for their constructive feedback. This work is supported financially by the National Hi-tech Research and Development Program of China under contract 2012AA010902, the National Basic Research Program of China under contract 2011CB302501.

claimed to achieve more than 100x speedup, GPUs appear poorly suited for sparse graph and other irregular computation [8]. One of the reasons is that modern GPU relies on high SIMD lanes occupancy to boost performance. A full efficiency will be achieved only if all the SIMD lanes agree on their execution paths. However real-world graph instance has an irregular out-degree distribution so that the computation tasks in graph applications are intrinsically imbalanced.

Prior work [15] has attempted to assign each task with a virtual warp of threads to alleviate workload imbalance. Their results suggest that the performance is not proportional to the virtual warp size because the warp-centric method suffers from underutilization in SIMD operations, especially when the average vertex degree is smaller than the warp size. Therefore, the optimal virtual warp size is dependent on input graph and must be tuned by hand. Although their method successfully accelerates irregular graph algorithms, many questions remain unanswered. For example, what is the relationship between graph topology and SIMD efficiency? How can we quantify the SIMD efficiency for graph algorithms on GPU? What are the trade-offs among different mapping strategies? Given an input graph, how can we maximize the parallelism and SIMD efficiency?

In this work, we intend to understand the relation between graph topology, SIMD utilization and performance through parallelizing one fundamental graph algorithm on GPU: breadth-first search (BFS). BFS is a common primitive for building complex graph algorithms. In summary, this paper makes the following contributions:

- We conduct an architecture-independent characterization of graph irregularity and potential concurrency within BFS.
- We develop a model to analyze the components of SIMD underutilization. This analytical model can help us understand the relationship between graph topology and SIMD utilization. We discover that the SIMD underutilization either comes from the imbalance of vertex degree distribution, or from the heterogeneity of each vertex degree.
- We quantify the SIMD efficiency for different workload allocation strategies with three metrics derived from our model. We perform experiments to reveal the impact of SIMD efficiency on the performance of BFS on GPU. We show how to use these metrics to analyze the performance for different mapping strategies.

2 Background

2.1 GPU Architecture and CUDA Programming Model

NVIDIA GPUs consist of a scalable array of Streaming Multiprocessors (SM). Each SM contains a set of cores, called Stream Processors (SP), or CUDA cores. Multiple SPs are grouped to share a single instruction unit, and execute instructions in an SIMD (Single Instruction, Multiple Data) manner. Logically, a *warp*

is a group of (e.g. 32) threads executing one common instruction at a time. Although modern GPU architectures allow threads in a *warp* to execute different instruction paths, this can severely degrade performance because the execution needs to be serialized. Threads from a warp falling into different instruction paths are said to *diverge*; Data-dependent branch is the common trigger of *divergence*. Moreover, different warps can execute actively on the same group of cores in an SMT (Simultaneous Multithreading) fashion, as a way to improve throughput despite high latencies.

CUDA is a programming model for leveraging the NVIDIA GPU for general-purpose computation. A CUDA program consists of different phases that are running either on host CPU or GPU. The GPU code is called *kernel*. A kernel is organized as a hierarchy of threads: threads are grouped into *blocks*, and blocks are grouped into a *grid*. In particular, each thread is mapped onto a CUDA core and each block is mapped onto a SM. Threads within a block can cooperate with each other through a scratchpad memory and barriers.

This abstraction hides the details of thread execution. It requires programmers to specify the hierarchical structure of threads before launching the kernel, then the underlying hardware is in charge of scheduling the threads. This scheme can improve the programming productivity but does not naturally fit the graph algorithms for two reasons. First, the intrinsic irregularity of real-world graph instance yields significant workload imbalance. So an implementation unaware of the execution model can cause thread divergence and underutilize hardware. Second, the parallelism pattern within graph algorithms depends upon the structure of input graph, which is hard to predict.

2.2 Breadth-first Search

Breadth-first search (BFS) is a fundamental graph algorithm. Given a graph $G(V, E)$, and a source vertex r, the BFS algorithm systematically traverses the edges of G and produces a BFS tree (Fig. 1) with the root r and all reachable vertices. Each node in the BFS tree is assigned with a *level* value (or minimum hops) that equals to its height in the tree. The algorithm guarantees all vertices at level k are visited before any vertices at level $k + 1$ are discovered. As shown in Fig. 1, the vertices of the same *level* compose a *vertex-frontier* and all their out-going edges logically form an *edge-frontier*. In essence, the process of BFS can be viewed as iterations over two vertex-frontiers: in each iteration, vertices in the *current* frontier are being visited, and all their unvisited neighbors are added to the *next* frontier. The process repeats until the *current* frontier is empty.

Algorithm 1 describes the kernel used in a traditional GPU implementation. As can be seen, each vertex (indicated by u) in the current frontier (C) is assigned to a single thread (Line 2). The for-loop in Line 3-8 iterates over u's neighbors and attempts to visit them. The visited neighbors are put into the next frontier (N). Noticeably, the number of iterations each thread needs to perform depends on the size of neighborhood. As will be characterized in Section 3.3, the real-world graph instances display tremendous heterogeneity. Therefore one thread

Algorithm 1 *BFS_kernel*

```
1: tid = getThreadId
2: u = C_tid
3: for v ∈ u' s neighbors do
4:     if v has not been visited then
5:         Visit v
6:         N = N ∪ {v}
7:     end if
8: end for
```

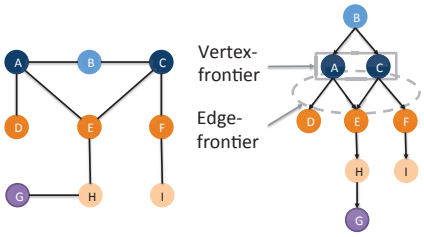


Fig. 1. BFS tree of a sample graph.

having more neighbors to examine can stall the other threads within its warp, thus degrading performance.

2.3 Coarse- and Fine-grained Mapping

Generally, there are two simple workload assignment schemes: (a) *coarse-grained mapping* assigns each thread with a single element in the vertex-frontier. Then the thread that has more neighbors to process can stall other threads within its warp; (b) *Fine-grained mapping* enlists a warp of threads to process one element in the frontier. In this case, SIMD lanes are wasted when the size of edge list of one node is smaller than the warp size.

Intuitively, the fine-grained mapping is preferable to the coarse-grained mapping for three reasons. First, the coarse-grained mapping is much more vulnerable to the irregularity of workload. Second, the fine-grained mapping exploits more concurrency, thus having a greater chance of saturating the hardware resources. Finally, the memory access pattern in the fine-grained mapping is more coalesced since the edge list of a node is accessed by a warp of threads. However the fine-gained mapping can yield higher underutilization when the average out-degree is low.

3 Dataset Characterization

3.1 Overview of Datasets

In this paper, we consider fifteen graphs from different application domains: the Florida road network is from the 9th DIMACS Implementation Challenge [2]; The Amazon co-purchase network and the LiveJournal social network are from the Stanford Large Data Collection [3]; We generate the 2D mesh datasets ourselves; The *rmat* and the *random* graph are constructed using SNAP graph library [4]; The remaining datasets are from the 10th DIMACS Implementation Challenge [1].

Table 1 characterizes these datasets in terms of scale. As can be seen, the graphs are in various sizes: the largest *random1* graph (with about 2.0 M nodes

Table 1. Graph datasets

Name	Description	Nodes	Edges	$\bar{d}$
mesh	6-point 2D mesh	1.0 M	6.3 M	6.0
FLA-road	Florida road map	1.1 M	2.7 M	2.5
thermal	3D nonlinear thermal problem	1.2 M	7.4 M	6.0
ecology	Animal/gene flow	1.0 M	4.0 M	4.0
audikw	Automotive finite element analysis	0.9 M	76.7 M	81.3
coPapers	Citation networks	0.4 M	32.0 M	73.9
LiveJournal	Social network	4.8 M	69.0 M	14.2
kkt-power	Optimal power flow	2.1 M	13.0 M	6.3
Amazon	Co-purchased products network	0.4 M	3.2 M	8.0
rmat1	Small world graph (RMAT)	5.0 M	60.0 M	12.0
rmat2	Small world graph (RMAT)	2.0 M	100.0 M	50.0
random1	Uniformly random graph	2.0 M	128.0 M	64.0
random2	Uniformly random graph	2.0 M	100.0 M	50.0
random3	Uniformly random graph	5.0 M	60.0 M	12.0
kron2	Kronecker (Graph500)	1.0 M	89.2 M	85.1

and 128.0 M edges) and the smallest *FLA-road* graph (with about 1.1 M nodes and 2.7 M edges) differ by almost two orders of magnitude. In terms of average degree ($\bar{d}$) , some graphs (e.g. *FLA-road*, *thermal*, *mesh*, *ecology*, and *kkt-power*) are sparse while others (*rmat1,2* and *random1-3*) are relatively dense. The *coPapers* and *audikw* datasets have a small node set (about 0.4 M and 0.9 M, respectively) but a considerable edge number (about 73.9 M and 81.3 M, respectively).

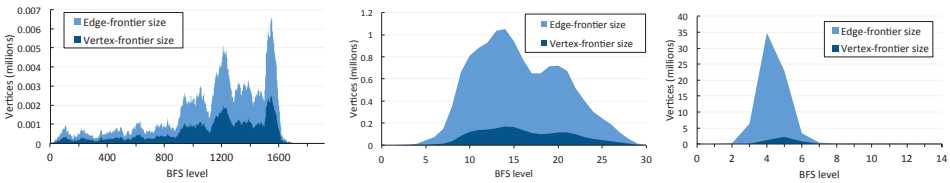
3.2 Profiling Available Concurrency during BFS

There are two levels of concurrency lying in each iteration of BFS: (a) one is among the vertices that can be processed simultaneously and (b) the other is among the outgoing edges of each vertex. So the potential concurrency during BFS can be represented by the *vertex-frontier size* and the *edge-frontier size*.

Fig. 2 plots such two-level concurrency exposed in each BFS level. As can be seen, parallelism grows slowly in the *FLA-road*. This is because most nodes in the graph have 2 or 3 neighbors. As a result, the number of vertices that can be put into the frontier is limited during each BFS iteration. Three other sparse graphs (*mesh*, *ecology* and *thermal*) also have this property. We also observed that these graphs have a much longer diameter (greater than 1000). On the contrary, the diameter of *LiveJournal* is small and most of its concurrency is lying in a few levels. This property, so-called *small world phenonmenon* [26], has also been observed in *kron20*, *rmat1,2* and *coPapers*.

3.3 Characterization of Irregularity

Although BFS exhibits abundant runtime concurrency, it is non-trivial to map the computation tasks to GPU's massive parallel resources because the work-



(a) *FLA-road* (b) *kkt-power* (c) *LiveJournal*

Fig. 2. Plots of vertex- and edge-frontier size in each BFS level on three sample graphs.

load is usually imbalanced. Fig. 3 depicts the out-degree distribution across the datasets in boxplots[1].

As can be seen, the leftmost four graphs (*ecology*, *mesh*, *thermal* and *FLA-road*) are observed within a tight range of out-degree: the difference between the highest and the lowest out-degree is no more than 8; and most vertices differ by only 1 out-degree. The *Amazon* graph is relatively regular, more than a half of the vertices have 10 outgoing edges, but the remaining nodes have an out-degree between 0 to 9. The small-world networks (*rmat1,2*) are pretty concentrated: most vertices are within a small range (10∼14 and 47∼53, respectively). On the other hand, the *random1-3* networks are far more irregular: vertices with different out-degree are distributed evenly between a wide range. The rightmost four graphs (*coPapers*, *kkt-power*, *LiveJournal* and *kron20*) are highly skewed. As a result, outliers exist in these graphs (up to 131,503 degree in *kron20*). Among all the datasets, the *kron20* network is the most irregular graph, which has an out-degree span of nearly 0.1 M.

4 Topology and Utilization

In this section, we introduce a model which can help us analyze the impact of graph topology on SIMD utilization. As shown in Fig. 4, we divide a warp of threads into groups and assign each group with one vertex in the frontier. Given a frontier of N vertices, we enlist N groups of threads to process them independently. Each thread group performs its work in a narrower logical SIMD window iteratively. For example, thread group 1 requires two iterations to fulfill its job and thread group 2 requires four. The logical SIMD window of group 1 can not be released due to the physical SIMD convention. The warp size is 32. The group size is denoted as S ($S \in \{1, 2, \cdots, 32\}$). The coarse-grained and the fine-grained mapping can be viewed as two special cases where $S = 1$ and $S = 32$, respectively. The work amount (out-degree) assigned to each group is denoted as d_k ($0 \leq k \leq N - 1$).

[1] In each boxplot, the two horizontal lines (topmost and bottommost) represent the greatest and the least value (out-degree). The top and the bottom of the box denote the first (25%) and the third *quartile* (75%), respectively. Finally, the diagonal cross in the box stands for the *median* (50%)

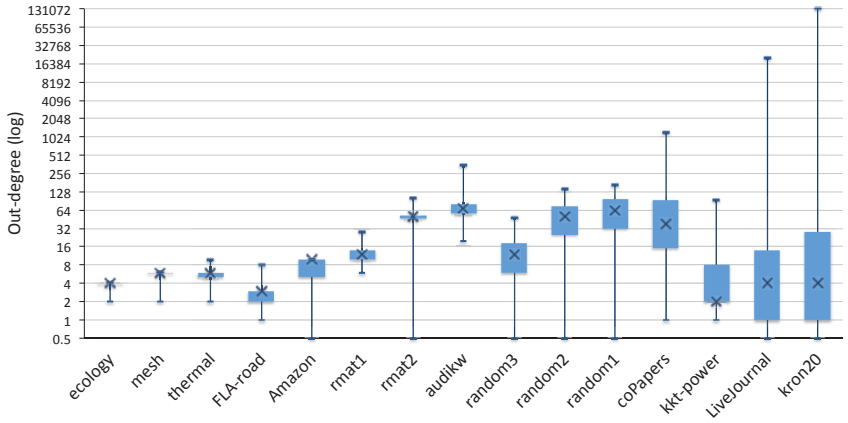


Fig. 3. Boxplot of out-degree distribution across the datasets.

We divide the SIMD underutilization of each group into two parts: (a) **Intra-group underutilization** U_A^k: represented by the "bubbles" (shown here as the circles with red diagonal lines) within the logical SIMD window of group k; (b) **Inter-group underutilization** U_R^k: defined as the workload gap (represented by the dashed circles) between group k and the group having the heaviest workload in the same warp.

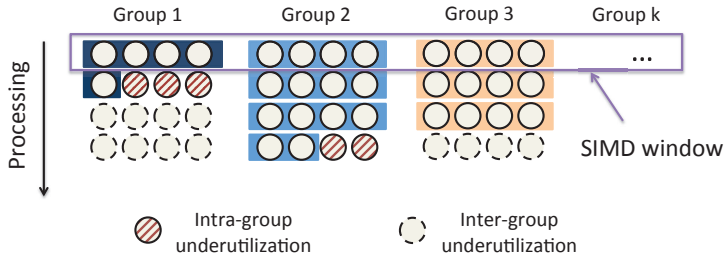


Fig. 4. Intra- and inter-group underutilization within a warp.

U_A^k can be given by: $U_A^k = S \cdot \lceil \frac{d_k}{S} \rceil - d_k$. Replace $\lceil \frac{d_k}{S} \rceil$ with $\frac{d_k}{S} + \phi_k$, where $\phi_k \in [0, 1)$. So the overall intra-group underutilization can be represented by:

$$U_A = \sum_{k=0}^{N-1} S \cdot \phi_k \tag{1}$$

Then we have $U_A \in [0, N \cdot S)$. The lower bound of U_A is achieved when $S = 1$. Let D_k denote the dominating workload within the warp where group k is located. Then U_R^k is given by:

$$U_R^k = S \cdot \left(\left\lceil \frac{D_k}{S} \right\rceil - \left\lceil \frac{d_k}{S} \right\rceil \right) \tag{2}$$

Replace $\left\lceil \frac{D_k}{S} \right\rceil$ with $\frac{D_k}{S} + \theta_k$, where $\theta_k \in [0, 1)$. Similarly, let $\left\lceil \frac{d_k}{S} \right\rceil = \frac{d_k}{S} + \sigma_k$, where $\sigma_k \in [0, 1)$:

$$\begin{aligned} U_R^k &= S \cdot \left[\left(\frac{D_k}{S} + \theta_k \right) - \left(\frac{d_k}{S} + \sigma_k \right) \right] \\ &= D_k - d_k + S \cdot (\theta_k - \sigma_k) \\ &= D_k - d_k + S \cdot \pi_k \quad (-1 < \pi_k < 1) \end{aligned}$$

By comparing the above equation with Eq. 2, we notice that the role of $S \cdot \pi_k$ is to round $D_k - d_k$ to the closest multiple of S. This term can be neglected if S is much smaller than $D_k - d_k$. In this case, the overall inter-group underutilization U_R can be represented by:

$$U_R = \sum_{i=0}^{N-1} (D_k - d_k) \tag{3}$$

According to Eq. 1 and Eq. 3, we can draw the following conclusions:

- U_R is primarily induced by the heterogeneity of workloads. The term $D_k - d_k$ in Eq. 3 reflects the degree of out-degree dispersion between vertices. When a graph is regular, $D_k - d_k$ is almost zero, then U_R is negligible. On the other hand, U_R is significant when traversing an irregular graph.
- U_R is sensitive to the group size. More specifically, D_k may rise as we decrease S. This is because the more groups located in a warp, the more likely that the warp will encounter a "outlier". When $S = 32$, there is only one group in the warp, so U_R is 0.
- U_A is determined by the intrinsic irregularity of vertex degree (indicated by ϕ_k). For example, if a vertex has a degree of 23, then $U_A^k = 9, 9, 1, 1, 1, 0$, when $S = 32, 16, 8, 4, 2, 1$, respectively. In practice, U_A can be effectively limited if we shrink the group size. When $S = 1$, U_A is eliminated entirely.

Given a vertex frontier, the underutilization of SIMD lanes can convert between U_R and U_A when different group sizes are chosen. When S is small, most of the SIMD lanes are wasted in the form of inter-group underutilization; conversely, when S is large, the intra-group underutilization is the dominant factor. We observe that the inter-group underutilization is greatly affected by the topology of graph.

5 Results and Discussion

In this section, we derive metrics from our model for profiling SIMD efficiency. We show that the performance of BFS on GPU is influenced by available concurrency, spatial locality, and SIMD utilization. Finally, we discuss other use scenarios where the model can be applied to.

5.1 Methodology

We adopt a two-phase algorithm similar to previous works [12, 21]. Specifically, the algorithm expands edges from a queue, collects expanded vertices into a bitmask, and compacts the bitmask to a queue before the next iteration starts. In expansion-phase, we assign each element in the queue with a variable-sized (1, 2, ..., 32) group of threads. The corresponding mapping strategies are simply referred as *sm-{1,2,...,32}*.

We store the graphs mentioned in Section 3.1 in Compressed Sparse row (CSR) format, which is adopted by most previous works [10, 15, 16, 20, 21, 24] for its efficiency. Our experiments are conducted using a 1.15GHz NIVIDIA Telsa C2050 GPU (Fermi Architecture) with 14 32-core SMs and 3GB of device memory. The host machine has an Intel Xeon E5506 2.13 GHz processor with 12 GB RAM. All of our programs are compiled with nvcc 5.5 and the '–O3 –arch=sm_20' flag.

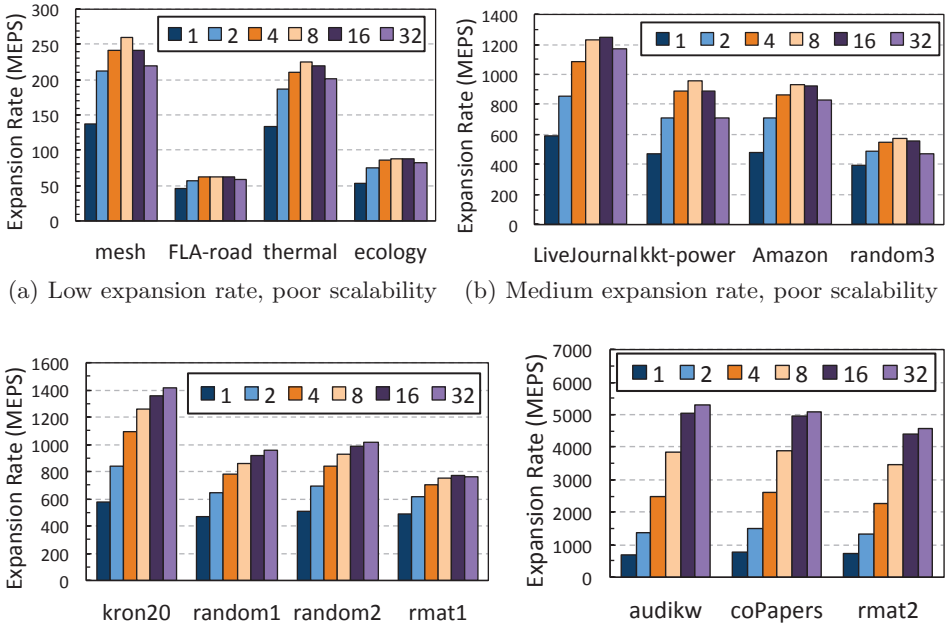
In this work, we concentrate on the workload imbalance problem in expansion-phase. To benchmark the performance, we calculate the *expansion rate* in Millions of Edges Per Second (MEPS) by taking the ratio of the expanded edges to the execution time for expansion-phase. For each plotted data point, we perform BFS 10 times from 10 pseudo randomly selected vertices and average the runtime.

5.2 Comparing Mapping Strategies

We first compare the performance for different mapping strategies across all the datasets (Fig. 5). All the datasets fall into four categories according to their behavior:

- *Low expansion rate, poor scalability.* Four datasets in this category (*mesh*, *FLA-road*, *thermal*, and *ecology*) appear poorly suited for GPU implementation. Recall that concurrency grows slowly in those graphs (Section 3.2). Therefore, the GPU implementation can not spawn sufficient amount of work to saturate the hardware in most BFS iterations. Employing a fine-grained mapping strategy can introduce more concurrency. But it also leads to higher SIMD underutilization, since the average out-degree in those datasets is low (<8). Hence, *sm-32* and *sm-16* are even slower than *sm-8*.
- *Medium expansion rate, poor scalability.* Datasets in this category (*LiveJounral*, *kkt-power*, *Amazon*, and *random3*) have considerable expansion rate due to their rich concurrency. However their performance still does not scale with the group size. Namely, *sm-32* can not outperform all its counterparts. For example, the optimal mapping granularity for *random3* and *LiveJournal* is 8 and 16, respectively. This observation is coherent with [15].
- *Medium expansion rate, good scalability.* Four graphs in this category (*kron20*, *random1,2*, and *rmat1*), on the contrary, have better scalability. That is, the expansion rate increases with group size (except for *rmat1*). Recall that most vertices in *rmat1* have an out-degree range between 10 to14 (Fig. 3). Therefore, the optimal group size for *rmat1* is 16.

– *High expansion rate, good scalability.* Three datasets in this category (*audikw*, *coPapers*, and *rmat2*) not only have high expansion rate, but also scale well with group size. We observed abundant spatial locality in these datasets. Finally, the expansion rate on these datasets achieves up to ∼5000 MEPS in the case of *sm-32*.



(a) Low expansion rate, poor scalability (b) Medium expansion rate, poor scalability

(c) Medium expansion rate, good scalability (d) High expansion rate, good scalability

Fig. 5. Expansion rate in MEPS for different mapping granularity.

5.3 Evaluting the SIMD Efficiency

We derive three metrics: `UR`, `UA` and `ME` from the model in Section 4 for quantifying the SIMD efficiency. `UR` and `UA` stands for the inter- and intra-group underutiliztion rate, respectivley; `ME` (*mapping efficiency*) is a metric that reflects the utilization rate for a certain workload assignment strategy. `ME` is complementary to `UA` and `ME`. We calculate these metrics online by simulating the SIMD windows operating on the vertex-frontier during BFS.

Fig. 6 captures the utilization trend with increasing group size across datasets. As can be seen, `UA` (light blue bar) always rises with the group size and `UR` (orange bar) falls with it. This is consistent with our conclusion in Section 4.

For some graphs (e.g. *kron20*, *audikw*, and *copapers*), increasing group size can effectively alleviate the inter-group underutilization with introducing minor intra-group underutilization, thus improving the mapping efficiency. This explains why these datasets have good scalability. While the SIMD utilization for some datasets (*random1,2* and *rmat2*) presents a descending trend, the mapping efficiency can be maintained at a high level (80%) when increasing group size. So the performance for these datasets scales with group size too. The mapping efficiency of *rmat1* declines by nearly half from *sm-16* to *sm-32*. This explains why the expansion rate of *sm-32* is slower than *sm-16* (Fig. 5(c)).

But for some datasets (e.g. *LiveJournal*, *kkt-power*, *random3*), the benefits of reducing inter-group undertutilization are soon outweighed by the fast-growing intra-group underutilization. For other graphs (e.g. *mesh*, *FLA-road, thermal, ecology*, and *Amazon*), increasing group size can do little help to the inter-group underutilization. On the contrary, it will only lead to severe intra-group underutilization. All these datasets correspond to the aforementioned categories that have poor scalability. The optimal group size for them is either 8 or 16.

5.4 Discussion

Our analysis shows that the performance of BFS on GPU depends on many factors. The available concurrency is the first-order impact. It is only when the input graph offers sufficient concurrency, the capabilities of GPU (massive threads, lightweight context switching, etc) can be exploited. However, when the hardware resources are saturated, the utilization of SIMD lanes becomes critical. As can be seen, different datasets favor different workload allocation granularities due to the diversity of graph structures.

Here we use the metrics to profile the SIMD efficiency. There are other scenarios where these metrics can be used: (a) *Online feedback for decision makers.* One can devise an adaptive BFS implementation that takes the SIMD efficiency into account. Notably, simulating all SIMD operations on all the vertices in every BFS level can bring huge overhead, which can be reduced by sampling (that is, by analyzing partial vertices in partial BFS levels); (b) *Feedback for auto-tuning compilers.* The optimization space of the compiler can be pruned by using these metrics to estimate the performance; (c) *Static Analysis.* One can estimate the SIMD efficiency on a target architecture by statically examining the nodes in the graph.

6 Related Work

BFS was well studied in past decades. The serial algorithm uses a queue data structure to maintain the vertex-frontier, and performs optimal $O(|V| + |E|)$ work amount [9] since each vertex/edge in the graph is visited/traversed only once. There are already massive researches on designing an efficient parallel BFS algorithm for supercomputers [6, 28] and multi-socket systems [5, 7, 20, 25, 27].

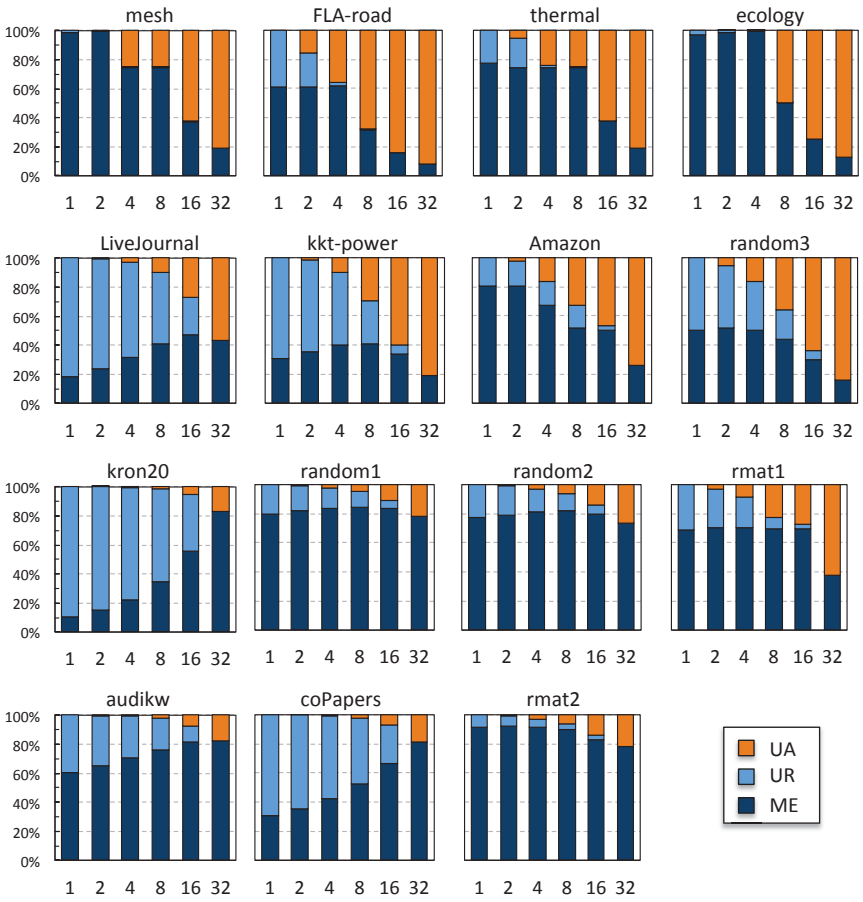


Fig. 6. SIMD utilization for different mapping strategy across datasets.

Modern GPU offers an opportunity to accelerate highly parallel graph algorithms (e.g., breadth-first search [10, 15, 16, 22, 23], connected components [14], shortest path [17]). Harish and Narayanan [11] presented a handful of CUDA implementations for graph algorithms, which is considered a pioneer research in this area. Their BFS implementation inspects all the vertices in each BFS iteration using coarse-grained mapping.

Hong et al. [15] proposed a generalized programming scheme to alleviate the load imbalance problem in graph algorithms. Their method allows tradeoffs between workload imbalance and ALU underutilization by varing a single parameter (i.e., virtual warp size). Their results show that the optimal virtual warp size is dependent on the input graph and must be tuned by hand.

Merrill et al. [24] intended to achieve high SIMD utilization by leveraging prefix sums. In their implementation, threads coordinate their unique computation task by performing prefix sums to the edge lists. Therefore, no SIMD lanes

are underutilized in the expansion phase. However the SIMD underutilization will occur during calculating the prefix sums. So they devise a hybrid strategy to mitigate the overhead.

Due to the heterogeneity of real-world graph datasets, some work considered an adaptive solution. Hong et al. [16] proposed a hybrid method which dynamically selects between CPU and GPU implementations for each BFS iteration. Li et al. [21] explored the implementation space for graph algorithms on GPU in different dimensions (e.g., mapping granularity, vertex-frontier representation). Their runtime uses either a thread-wise or a block-wise mapping in the expansion phase.

7 Conclusion

It is non-trivial to map the the irregular computation tasks in graph application onto GPU's hardware. Traditional simple workload allocation schemes can easily underutilize the hardware. A better solution is to assign each task with a group of threads. However, the optimal mapping granularity is usually dependent on the input graph and must be manually tuned.

In this paper, we attempt to understand the link between graph topology and SIMD utilization with one fundamental graph traversal algorithm (BFS). We present a model for analyzing the components of SIMD underutilization. Based on the model, we discover that the SIMD lanes are underutilized either due to the imbalance of vertex degree distribution, or to the heterogeneity of each vertex degree. We also devise three metrics for quantifying the SIMD efficiency. By using these metrics, the performance for different mapping strategies on a variety of datasets can be explained. We expect that the method presented in this paper will provide a foundation for developing techniques of static analysis and runtime optimization.

References

1. 10th dimacs implementation challenge. `http://www.cc.gatech.edu/dimacs10/index.shtml`, accessed: 2013-12-15
2. 9th dimacs implementation challenge. `http://www.dis.uniroma1.it/~challenge9/download.shtml`, accessed: 2013-12-15
3. Stanford large network dataset collection. `http://snap.stanford.edu/data/index.html`, accessed: 2013-12-15
4. Stanford network analysis platform. `https://snap.stanford.edu/snap/index.html`, accessed: 2013-12-15
5. Agarwal, V., Petrini, F., Pasetto, D., Bader, D.A.: Scalable graph exploration on multicore processors. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–11. IEEE Computer Society (2010)
6. Bader, D.A., Madduri, K.: Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2. In: Parallel Processing, 2006. ICPP 2006. International Conference on. pp. 523–530. IEEE (2006)

7. Beamer, S., Asanovic, K., Patterson, D.: Direction-optimizing breadth-first search. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for. pp. 1–10. IEEE (2012)
8. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. pp. 44–54. IEEE (2009)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., et al.: Introduction to algorithms, vol. 2. MIT press Cambridge (2001)
10. Deng, Y., Wang, B.D., Mu, S.: Taming irregular EDA applications on GPUs. In: Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on. pp. 539–546. IEEE (2009)
11. Harish, P., Narayanan, P.J.: Accelerating large graph algorithms on the GPU using CUDA. In: High performance computing–HiPC 2007, pp. 197–208. Springer (2007)
12. Harish, P., Vineet, V., Narayanan, P.J.: Large graph algorithms for massively multithreaded architectures. Centre for Visual Information Technology, I. Institute of Information Technology, Hyderabad, India, Tech. Rep. IIIT/TR/2009/74 (2009)
13. Hassaan, M.A., Burtscher, M., Pingali, K.: Ordered vs. unordered: a comparison of parallelism and work-efficiency in irregular algorithms. In: Proceedings of the 16th ACM symposium on Principles and practice of parallel programming. pp. 3–12. ACM (2011)
14. Hawick, K.A., Leist, A., Playne, D.P.: Parallel graph component labelling with gpus and cuda. Parallel Computing 36(12), 655–678 (2010)
15. Hong, S., Kim, S.K., Oguntebi, T., Olukotun, K.: Accelerating CUDA graph algorithms at maximum warp. In: Proceedings of the 16th ACM symposium on Principles and practice of parallel programming. pp. 267–276. ACM (2011)
16. Hong, S., Oguntebi, T., Olukotun, K.: Efficient parallel graph exploration on multi-core CPU and GPU. In: Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on. pp. 78–88. IEEE (2011)
17. Katz, G.J., Kider Jr, J.T.: All-pairs shortest-paths for large graphs on the GPU. In: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware. pp. 47–55. Eurographics Association (2008)
18. Kulkarni, M., Burtscher, M., Inkulu, R., Pingali, K., Casçaval, C.: How much parallelism is there in irregular applications? In: ACM Sigplan Notices. vol. 44, pp. 3–14. ACM (2009)
19. Kulkarni, M., Pingali, K., Walter, B., Ramanarayanan, G., Bala, K., Chew, L.P.: Optimistic parallelism requires abstractions. In: ACM SIGPLAN Notices. vol. 42, pp. 211–222. ACM (2007)
20. Leiserson, C.E., Schardl, T.B.: A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In: Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures. pp. 303–314. ACM (2010)
21. Li, D., Becchi, M.: Deploying Graph Algorithms on GPUs: An Adaptive Solution. 2013 IEEE 27th International Symposium on Parallel and Distributed Processing pp. 1013–1024 (May 2013), `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6569881`
22. Luo, L., Wong, M., Hwu, W.m.: An effective GPU implementation of breadth-first search. In: Proceedings of the 47th Design Automation Conference. pp. 52–55. ACM (2010)
23. Merrill, D., Garland, M., Grimshaw, A.: High performance and scalable gpu graph traversal. Univ. of Virginia, Tech. Rep. UVA CS-2011-05 (2011)

24. Merrill, D., Garland, M., Grimshaw, A.: Scalable GPU graph traversal. In: ACM SIGPLAN Notices. vol. 47, pp. 117–128. ACM (2012)
25. Scarpazza, D.P., Villa, O., Petrini, F.: Efficient breadth-first search on the cell/be processor. Parallel and Distributed Systems, IEEE Transactions on 19(10), 1381–1395 (2008)
26. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-worldnetworks. nature 393(6684), 440–442 (1998)
27. Xia, Y., Prasanna, V.K.: Topologically adaptive parallel breadth-first search on multicore processors. In: Proceedings of the 21st IASTED International Conference. vol. 668, p. 91 (2009)
28. Yoo, A., Chow, E., Henderson, K., McLendon, W., Hendrickson, B., Catalyurek, U.: A scalable distributed parallel breadth-first search algorithm on BlueGene/L. In: Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference. p. 25. IEEE (2005)