Supplementary Material for GMDO

Peng FANG, Fang WANG, Zhan SHI, Dan FENG, Qianxu YI, Xianghao XU, Yongxuan ZHANG
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology, Wuhan, 430074, China

A. Random memory access in graph processing

To exhibit random access in graph processing based on CSR format, we show an example in Fig. 1 (a). The graph has 6 vertices and 7 edges. The CSR storage format for the graph is presented in Fig. 1 (b). The $Edge\ Array$ stores the vertex ID and the $Vertex\ Array$ stores the starting offset of the neighbor for the vertex in the $Edge\ Array$. Meanwhile, there is a $Property\ Array$ to store the status value during the application execution. From Fig. 1 (b), we can find that if a graph application accesses the vertex v_3, although the $N(v_3) = \{v_0, v_2, v_4\}$ have exhibited sequentially in the $Edge\ Array$, the status value of the vertices are scattered in the $Property\ Array$ due to the vertices ID are discontinuous. As such, there will incur frequent cache data replacement, especially for the natural sparse graphs.

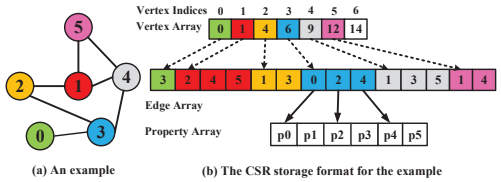

Fig. 1. The CSR storage format.

B. Classification of graph applications

We divided the graph applications into two types (traversal and iterative) based on the dynamic running characteristics. Traversal applications are sequentially accessing the entire graph from an initial vertex, focusing on traversing with a few computations. In contrast, iterative applications execute iteratively until all vertices reaching convergence, the vertex has to be updated synchronously, and the number of iterations heavily relies on the graph structure and the convergence conditions. Therefore, the memory data organization for traversal graph applications should reflect the relationship of accessing pattern between vertices which will facilitate to maintain good locality during traversing, and for iterative applications, it needs to consider the convergence characteristics and the compact organization of neighbors for active vertex, such that it can accelerate vertex convergence when performing an update on each active vertex.

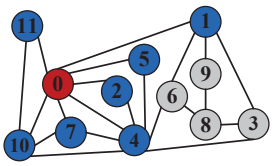

Fig. 2. Example of the GMDO.

C. Details of GMDO Strategy

To illustrate the effect of GMDO for the efficiency of memory data access, a Graph $G = (V, E)$ is shown in Fig. 2 as a running example, and Fig. 3 exhibits the data organization of the original, BFS, GMDO-VR, and GMDO-VI, respectively.

For the traversal applications, if the graph application accesses the vertex v_0 based on BFS as shown in Fig. 3(b), v_1 will be accessed after v_0 is processed, however, there are no neighbors of v_1 in the cache at this moment due to the vertex relevance degree $R(v_0, v_1) = 0$, it has to retrieve the data from the main memory, which will bring data exchange frequently and cause the computing unit to wait for a long time. In this case, if we consider the impact of R in the memory access pattern, the above situation will be quite different. In Fig. 3(c), we reorganize the data layout by GMDO-VR, v_{10} will be accessed next to v_0 due to $R(v_0, v_{10})$ is the highest than other neighbors of v_0. It indicates that the application can load the neighbors of v_{10} immediately from the on-chip cache as much as possible, which will facilitate to improve the cache hit ratio. Note that although v_0 and v_4 have more common neighbors than v_{10}, the $R(v_0, v_{10}) > R(v_0, v_4)$ means the neighbors of v_{10} are much more than v_4 in the on-chip cache for the next processing.

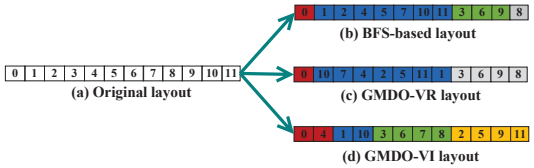

Fig. 3. (a), (b), (c), and (d) show the memory data organization by original, BFS, GMDO-VR, and GMDO-VI, respectively.

For the iterative applications, the number of iterations heavily relies on the graph structure and the convergence conditions. As the example of PageRank mentioned in

the main text, most of the low-degree vertices have been transformed into an inactive state in the first few iterations, while the rest of vertices need much more iterations to achieve convergence due to these vertices have a higher influence during processing. Besides, most real-world graphs are reported to be scale-free because their degree distributions follow a power-law, as least asymptotically. Based on these considerations, we reorganize the vertices layout by GMDO-VI as shown in Fig. 3(d). The high-degree vertices $\{v_0, v_4\}$ are preferentially accessed compared to the low-degree vertices $\{v_2,\ v_5,\ v_9,\ v_{11}\}$, it will guarantee the influential vertices and their neighbors have more opportunities to stay in the on-chip cache as the number of iterations increasing, while the BFS-based layout in Fig. 3(b) ignores the influence of high-degree vertices in the iterative applications, although it is friendly to the traversal applications.

D. Evaluation Methodology and Additional Experiments

Environment setup We evaluate the locality benefits of memory data organization strategy experimentally on a 2.0 GHz Intel Xeon E5-2620 commodity machine equipped with 32 GB main memory and 1 TB disk. The per-core L1 and L2 cache sizes for this CPU are 32KB and 256KB respectively, and the L3 cache size is 15MB.

Graph datasets We summarize the datasets used in our evaluation as shown in Table I. The selected graph datasets are mainly from SNAP, here, ca-GrQc (CA) is a collaboration network, web-Stanford (WS) and web-Google (WG) are web graphs, cit-Patents (CP) is a citation network, Google+ (G+) and Twitter (TWT) are online social networks. We also generate the synthetic graph RMAT21 (RM21) by the R-MAT algorithm to evaluate the performance of all techniques.

TABLE I
THE REAL-WORLD AND SYNTHETIC GRAPHS USED IN EXPERIMENTS.

Graph	Type	|V(G)|	|E(G)|
ca-GrQc	undirected	5242	14496
web-Stanford	directed	0.28M	2.30M
web-Google	directed	0.87M	5.10M
cit-Patents	directed	3.77M	16.52M
RMAT21	undirected	2.10M	33.55M
Google+	directed	28.94M	462.99M
Twitter	directed	61.57M	1.47B

Evaluation benchmarks As previously mentioned, two types of graph applications are tested in our evaluation: (1) traversal applications including the connected component (CC), single source shortest path (SSSP), and minimum spanning tree (MST); (2) iterative applications including the pagerank (PR) and label propagation algorithm (LPA). We also compare GMDO with the following four representative strategies: RCM, CD, Rabbit, and GOrder. RCM is a traversal-based graph vertex remapping using the reverse Cuthill-McKee algorithm. CD is a community detection-based approach adopts the label propagation algorithm to identify the community and then execute the vertex remapping for each community. Rabbit is a recently proposed graph reordering approach that maps the hierarchically dense communities in graphs to different levels of the cache hierarchy. GOrder is a fantastic graph reordering algorithm that relabels vertices to maximize the overlap between neighbors of vertices with consecutive IDs.

Evaluation for cache hit ratio We select the Linux kernel cache statistics $perf$ tool to evaluate the effect of memory data organization strategy on cache hit ratio for PageRank based on state-of-the-art graph systems: GraphChi and GridGraph. Table II shows the CPU cache statistic of running PageRank on seven real and synthetic graphs based on the original data organization and GMDO, respectively. Here, *L1-ref* is the number of L1 cache references (L1-dcache-loads), which is the total cache access number, because all cache accesses must be checked by the L1 cache firstly. *L1-mr* is the L1-cache miss ratio (L1-misses-rate). *LLC-ref* is the number of Last level cache (here is L3) references and *LLC-r* is the ratio of the cache reference checked in *LLC*, such that $LLC - r = \frac{LLC-ref}{L1-ref}$. A small *LLC-r* means that most cache references are hit by the first two levels fast cache, which can be effectively used to measure the performance of the cache hits. In Table II, GMDO achieves the smallest reduction of cache miss numbers in all levels of cache for the two graph systems, and significantly improves the cache hit ratio reach more than 90%, indicating a high CPU cache utilization in graph processing.

TABLE II
THE EFFECT OF STRATEGIES ON CACHE HIT RATIO

(a) PageRank application based on GraphChi

Graph	Layout	L1-ref	L1-mr	LLC-ref	LLC-r
CA	Original	602M	2.31%	4M	0.72%
	After-layout	512M	1.52%	2M	0.45%
WS	Original	2887M	13.20%	299M	10.39%
	After-layout	2807M	6.50%	109M	3.89%
WG	Original	5.8B	16.02%	776M	13.40%
	After-layout	5.6B	7.26%	292M	5.19%
CP	Original	75.4B	18.12%	8.3B	11.08%
	After-layout	70.3B	8.57%	4.1B	5.86%
RM21	Original	80.9B	24.03%	9.8B	12.09%
	After-layout	79.5B	9.25%	7.2B	9.01%
G+	Original	376.8B	22.59%	53.7B	14.25%
	After-layout	371.5B	7.75%	15.7B	4.22%
TWT	Original	979.7B	26.37%	158.8B	16.21%
	After-layout	921.6B	8.98%	62.8B	6.82%

(b) PageRank application based on GridGraph

Graph	Layout	L1-ref	L1-mr	LLC-ref	LLC-r
CA	Original	1118M	3.24%	4M	0.35%
	After-layout	1063M	2.45%	3M	0.28%
WS	Original	8.7B	4.30%	222M	2.55%
	After-layout	8.5B	1.87%	54M	0.64%
WG	Original	18.5B	4.85%	621M	3.35%
	After-layout	18.2B	1.76%	152M	0.83%
CP	Original	64.8B	4.38%	2.0B	3.06%
	After-layout	60.1B	3.35%	1.4B	2.33%
RM21	Original	106.7B	8.91%	6.8B	6.37%
	After-layout	106.5B	4.71%	3.5B	3.36%
G+	Original	758.5B	4.96%	26.0B	3.43%
	After-layout	720.6B	2.09%	11.7B	1.62%
TWT	Original	1782.5B	7.23%	81.7B	4.59%
	After-layout	1692.7B	4.02%	52.3B	3.09%