How well do CPU, GPU and Hybrid Graph Processing Frameworks Perform?*

Tanuj Kr Aasawat, Tahsin Reza, Matei Ripeanu
University of British Columbia
{taasawat, treza, matei}@ece.ubc.ca

Abstract—The importance of high-performance graph processing to solve big data problems targeting high-impact applications is greater than ever before. Recent graph processing frameworks target different hardware platforms (e.g., shared memory systems, accelerators such as GPUs, and distributed systems) and differ with respect to the programming model they adopt (e.g., based on linear algebra formulations of graph algorithms or enabling direct access to the graph structure). To better understand the impact of these choices, this paper, presents a comparative study of five state-of-the-art graph processing frameworks: two CPU-only frameworks - GraphMat and Galois, two GPU-based frameworks - Nvgraph and Gunrock; and Totem, a hybrid (CPU+GPU) framework. We use three popular graph algorithms (PageRank, Single Source Shortest Path, and Breadth-First Search), and massive scale graphs with up to billions of edges. Our evaluation focuses on three performance metrics: (i) execution time, (ii) scalability and (iii) energy consumption.

Keywords—Graph Processing, CPU, GPU, Hybrid Systems, Performance Evaluation, PageRank, SSSP, BFS.

I. INTRODUCTION

Graph algorithms serve a wide range of big-data problems, such as social network analysis, transportation network analysis, bioinformatics, financial and business analytics to name a few [2], [3], [4]. Over the years, several graph processing frameworks have been developed, targeting different architectures. State-of-the-art examples are, Galois [5] and Ligra [6] targeting large, shared memory systems; Nvgraph [7], Gunrock [8], and CuSha [9] targeting GPUs; as well as Giraph [10] and GraphLab [11] targeting multi-node systems. GraphLab and GraphMat [12] can harness parallelism both on a single-node as well as in a distributed setting. Totem [13] is a hybrid system, enabling graph processing on the CPU and GPU.

Motivation. Programming models for high-performance graph processing can be grouped in to two classes, namely, Vertex-or Edge-centric Programming Models and the Linear Algebra based approaches [12]. Although vertex/edge-centric models have gained popularity due to their expressiveness they often do not enable composability of well defined building blocks, thus often lead to higher development costs [12]. Linear algebra-based solutions, on the other hand, use Sparse Matrix Vector multiplication (SPMV) as a basic operation to model graph traversal [14], [15].

In order to achieve maximum performance and efficiency, it is important to strike the right balance between the architecture choice and the computation model. Existing graph processing frameworks primarily target an architecture and embrace one of the above computation models (i.e., vertex/edge-centric models or SPMV). However, it is not transparent which architecture and computation model maximizes performance and efficiency of the system as a whole. Furthermore, it is a topic of debate, what is the ideal graph programming model, vertex/edge-centric models or linear algebra. We are interested in understanding the state of the art offerings of each of these computational models with respect to performance on large datasets.

Contributions. For five state-of-the-art graph processing frameworks, we study the impact of the computational model and the hardware platform supported (CPU, GPU and hybrid). For our study, we consider two state of the art CPU-only frameworks Galois [5] and GraphMat [12]. For GPU-based frameworks, we choose Nvgraph [7] and Gunrock [8]. Our study includes Totem [13], a hybrid (CPU+GPU) graph processing framework. Among these frameworks, GraphMat and Nvgraph support a linear algebra based programming model, while the rest follow the popular vertex programming approach. We use three popular graph algorithms: PageRank, Single Source Shortest Paths (SSSP) and Breath-First Search (BFS). We consider both real-world and synthetic RMAT [16] generated graphs with up to billions of edges. We use three performance metrics: (i) execution time, (ii) scalability, and (iii) energy consumption.

Earlier work [1], [17], [18], [19] has compared different graph processing systems and/or proposed generic frameworks [20], [21] and benchmarks [22], [23] to enable such comparisons. One aspect that differentiates this work from most of the related work in this area is extending the comparison to include energy efficiency.

The rest of the paper is organized as follows: Section II provides a brief overview of the graph processing frameworks we evaluate. The evaluation methodology is discussed in Section III and experiment setup is detailed in Section IV. Section V presents the experiments and results, followed by concluding discussions in Section VI.

II. GRAPH PROCESSING FRAMEWORKS

We briefly describe the graph frameworks we use for our comparative study. Table I summarizes their target architecture, the computation model they follow and graph storage format.

Nvgraph. Nvgraph [7] is a graph processing library developed by Nvda. It leverages sparse linear algebra to express graph problems. It models graph computation as operations on semi-ring [15]. Nvgraph v9.0 release includes implementation (closed source) of a number of algorithms including PageRank, SSSP and BFS. Nvgraph supports graph processing on a single
GPU only. Depending on the graph algorithm, graph is stored either as Compressed Sparse Column (CSC) or as Compressed Sparse Row (CSR) format.

**Gunrock.** Gunrock [8] is a multi-GPU graph processing framework. It follows the Bulk Synchronous Processing (BSP) model. It leverages both the vertex-centric and edge-centric processing modes and alternates between vertex and edge centric modes depending on the algorithm execution state. Gunrock uses the CSR format for vertex-centric and the Coordinate format (COO) for edge-centric operations. Gunrock leverages the Cooperative Thread Array (CTA) technique by Merrill et al. [24] for balancing workload among GPU threads. In CTA, the workload of a single vertex, which is directly related to its degree, can be mapped to multiple threads (even larger than a warp) to achieve better load balancing.

**Galois.** Galois [5] is a shared-memory, graph processing framework targeting multi-core CPUs. Galois follows an asynchronous processing mode and enables lock-free computation. It incorporates a novel system-topology-aware task scheduler that supports priority scheduling. Galois exposes a high-level API which allows algorithm implementation following different graph abstractions, e.g., Gather-Apply-Scatter [25] and Push-versus-Pull [26]. Galois uses CSR format to store the graph in memory.

**GraphMat.** GraphMat [12] is a CPU-only, MPI-based distributed graph processing framework developed by Intel. On a single-node, GraphMat exploits thread level parallelism. GraphMat follows the BSP computation model and maps vertex-centric abstraction to sparse matrix operations. Similar to Nvgraph, it expresses graph computation as the semi-ring model based Sparse Matrix Vector multiplication [14]. GraphMat represents the sparse matrix in Doubly Compressed Sparse Column (DCSC) [27] format to efficiently store very large sparse matrices. Furthermore, it incorporates optimizations for load balancing and effective cache utilization.

**Totem.** Totem [13], [28], [29] is an in-memory graph processing framework for single-node heterogeneous systems consisting of the CPU and GPU. It supports parallel CPU-only, multi-GPU only, as well as a hybrid mode: processing on both the CPU and GPUs simultaneously. Totem follows the BSP programming model and supports vertex-centric processing. Totem incorporates several optimizations to achieve high performance. For example, partitioning the graph by vertex degree and placing higher degree vertices on CPU (to exploit the large CPU cache) and lower degree on GPU (to exploit massive parallelism), and sorting the edges by degree. Totem uses the CSR format for in-memory graph storage. In Totem, the GPU kernels uses the Virtual Warp [30] technique (which shares the same basic idea of distributing the workload of a vertex among multiple threads) to achieve load balance in the presence of heavily imbalanced graphs.

### III. Evaluation Methodology

We describe the benchmarks and the performance metrics.

**A. Benchmark Algorithms**

We consider PageRank, Single-Source Shortest Path (SSSP) and Breadth-First Search (BFS) as these algorithms have been widely studied in the context of high-performance graph processing systems, and have been used in past studies [5], [8], [12]. BFS and SSSP are also used as benchmarks for the Graph500 competition [31]. All three algorithms have implementations available in each of the frameworks we evaluate; thus, we believe these implementations, optimized by the authors of these frameworks, are best to drive their respective framework’s performance.

**PageRank.** PageRank is the well-known algorithm used by search engines for ranking web pages. In PageRank, a vertex computes its rank based on the rank of its neighbors. The algorithm continues until convergence of the vertex rank or a predefined number of iterations have been completed. PageRank has a high compute-to-memory ratio. It could be implemented as a pull-based or push-based algorithm [26]. In the pull-based approach, each vertex ‘pulls’ the rank of its neighbors, over the incoming edges, to compute its new rank. In the push-based approach, each vertex ‘pushes’ its rank to its neighbors, over the outgoing edges. Note that the push-based approach is less efficient, since, its parallel implementation requires atomic operations [5]. Totem and Galois implement pull-based approach. Totem’s algorithm kernel executes for predefined number of iterations [13]. Nvgraph, Gunrock and Galois implement the iterative-convergent method where user can provide maximum number of iterations [7] [8] [5]. The algorithm terminates when, either the maximum number of iterations has been completed or the algorithm has converged. For GraphMat, we updated the code so that it runs for a fixed number of iterations. Similar to Nvgraph, GraphMat implements PageRank as an SPMV kernel. Algorithm 1 [29] presents a typical pull-based PageRank algorithm in vertex programming model. A linear algebra based construct of PageRank is presented in Algorithm 2 [32].

<table>
<thead>
<tr>
<th>Framework</th>
<th>Platform</th>
<th>Computation Model</th>
<th>Memory Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nvgraph</td>
<td>GPU</td>
<td>SPMV</td>
<td>CSR (PageRank, SSSP)</td>
</tr>
<tr>
<td>Gunrock</td>
<td>GPU</td>
<td>Vertex Programming</td>
<td>CSR and COO</td>
</tr>
<tr>
<td>Galois</td>
<td>CPU</td>
<td>Vertex Programming</td>
<td>CSR</td>
</tr>
<tr>
<td>GraphMat</td>
<td>CPU</td>
<td>SPMV</td>
<td>DCSC</td>
</tr>
<tr>
<td>Totem</td>
<td>CPU-GPU</td>
<td>Vertex Programming</td>
<td>CSR</td>
</tr>
</tbody>
</table>

**Algorithm 1** Pseudo-code of PageRank in vertex programming model.

```
1: function PAGE(RANK(Partition p)
2:    delta = (1 - dFactor)/verCount
3:    for v in p.vertices do
4:        sum = 0
5:        for nbr in p.incomingNbrs do
6:            sum = sum + nbr.rank
7:        end for
8:    v.rank = delta + dFactor * sum
9: end for
```

**SSSP.** Single-Source Shortest Path (SSSP) is a graph traversal algorithm and finds shortest path between a given source
vertex and all other vertices (within the same connected component) in the graph. It has wide applications including IP routing, transportation networks, and social network analysis. In SSSP, each edge is associated with a predefined weight which, typically, is a measure of ‘cost’ to make a transition from one vertex to one of its neighbors. Totem uses an adaptation of the Bellman-Ford algorithm [33], as shown in Algorithm 3 [29], to implement SSSP and is relatable to [34]. The implementation applies an optimization, allows asynchronous ‘relax’ operations, which reduces the number of iteration (i.e., BSP supersteps) by allowing a vertex to be set to ‘active’ and perform ‘relax’ operations in the same iteration [13]. Both Nvgraph and GraphMat implements SSSP as an SPMV kernel. An example of linear algebra based SSSP algorithm is presented in Algorithm 4 [32]. Galois includes an implementation of the Delta-Stepping SSSP algorithm [35], while Gunrock implements Dijkstra’s algorithm [36].

**Algorithm 2** A linear algebra based implementation of PageRank algorithm.

```plaintext
1: function PAGE-RANK(Matrix A, Vector rankVector)
2:     delta = (1 - dFactor)/verCount
3:     rankVector = rankVector / verCount
4:     for iter = 1 to maxIter do
5:         rankVector = dFactor * A * rankVector + delta
6:     end for
7: end function
```

BFS. Breadth First Search (BFS) is a graph traversal algorithm which can determine the connected component starting from a given source vertex as well as the level of each vertex in the resulting BFS tree. BFS can be seen as a special case of SSSP where the weight of each edge is one. Totem, Galois and Gunrock implement the direction-optimized BFS algorithm [37]. Nvgraph and GraphMat implements BFS as an SPMV kernel. Algorithm 5 [29] presents a typical level-synchronous BFS algorithm expressed in vertex programming model, while Algorithm 6 [38] presents the linear algebra construct of the same.

**Algorithm 3** Pseudo-code of the level-synchronous BFS in vertex programming model.

```plaintext
1: function SSSP(Partition p)
2:     finished = true
3:     for v in p.vertices do
4:         if p.active[v] == false then
5:             continue
6:         end if
7:         p.active[v] = false
8:         for nbr in v.nbrs do
9:             if !p.visited.atomicSet(nbr) then
10:                new = p.dist[v] + v.weights[nbr]
11:                old = p.dist[nbr]
12:                if new < old then
13:                   if old == atomicMin(p.dist[nbr], new)
14:                      p.active[nbr] = true
15:                      finished = false
16:                   end if
17:               end if
18:            end for
19:        end for
20:     end function
```

**Algorithm 4** A linear algebra based implementation of the SSSP algorithm.

```plaintext
1: function SSSP(Matrix A, Vector distance, int source)
2:     distance = ∞
3:     distance[source] = 0
4:     dist_prev = distance
5:     while !finished do
6:         distance = A.min_+ dist_prev
7:         if diff(distance, dist_prev) == 0 then
8:             finished = true
9:         end if
10:        dist_prev = distance
11:     end while
12: end function
```

**Algorithm 5** Pseudo-code of the level-synchronous BFS in vertex programming model.

```plaintext
1: function BFS(Partition p, int level)
2:     finished = true
3:     for v in p.vertices do
4:         if v.level != level then
5:             continue
6:         end if
7:         for nbr in v.nbrs do
8:             if !p.visited.isSet(nbr) then
9:                 if p.visited.atomicSet(nbr) then
10:                   n.level = level + 1
11:                   finished = false
12:               end if
13:           end if
14:       end for
15:    end for
16:    return finished
```

**Algorithm 6** A linear algebra based implementation of the BFS algorithm.

```plaintext
1: function BFS(Matrix A, Vector distance, int source)
2:     distance = 0
3:     distance[source] = 1
4:     frontier = distance
5:     for level = 1 to verCount do
6:         frontier = A.x frontier & ~distance
7:         next = find(frontier)
8:         if isEmpty(next) then
9:             break
10:        end if
11:        distance[next] = level + 1
12:    end for
13:    distance = distance - 1
14: end function
```

**B. Evaluation Metrics**

We evaluate the frameworks with respect to three performance metrics: execution time, scalability and energy consumption.
**Execution Time.** We measure running time of an algorithm in seconds. Consistent with the standard practice in the domain, ‘execution time’ does not include time spent in pre- or post-processing steps such as graph loading, graph partitioning and result aggregation. For GPU-based frameworks, host-to-GPU data transfer times are not considered.

**Scalability.** We study how the frameworks respond with increasing number of processing units. For the GPU-based frameworks, we measure speedup achieved by two GPUs over a single GPU. For the CPU-based systems, we explore how each framework scales from one CPU-socket to two CPU-sockets. Also, we compare the best performing hybrid configuration of Totem with Totem on four CPU-sockets.

**Energy Consumption.** We measure energy consumption. Power (watts) is measured at the wall outlet using the WattsUP meter [39] which collects samples at one second intervals. We collect power for the entire duration of running a benchmark on a framework and then discard the readings for the pre- and post-processing. Therefore, Energy consumption = (average power reading during algorithm execution) × (algorithm execution time).

Idle system power of System 1(Table II) is ∼280W. The system includes two GPU. The idle power of each GPU is ∼20W (measured using the ‘nvidia-smi’ tool). When reporting energy consumption for the CPU-based frameworks, GPU idle-power is ignored (as if the system did not house any GPU). However, reported energy consumption for the GPU-based frameworks includes power consumption by the host system as well.

**IV. Experiment Setup**

The Testbed. Table II lists the characteristics of the machines we have used. For the GPU-based frameworks, we use Nvidia Tesla K40c GPU. For comparing the CPU-based frameworks, we use a dual-socket Intel Xeon E5-2695 v3 (Haswell) system. For GPU scalability experiments, we use up to two GPUs. Totem hybrid modes utilizes System 1. Additionally, we run Totem CPU-only on System 2, which has four Intel Xeon E5-4870 v2 (Ivy Bridge) CPU-sockets, to compare it with Totem hybrid.

Datasets. We use diverse real-world and synthetic datasets, as listed in Table III. soc-liveJournal [40], com-orkut [41], Road-USA [42] and Twitter [2] are real-world graphs. Synthetic Recursive MATrix (RMAT) [16] graphs are generated following Graph500 standards [43] with the following parameters: (A,B,C) = (0.57, 0.19, 0.19), and edge factor of 16. Not all graphs are used to compare every framework and algorithm. Road-USA is used for SSSP as it has very large diameter. For all the dataset we remove duplicate edges as well as self-loops. Unless otherwise mentioned, symmetric graphs are used for the experiments. Since, GraphMat do not support 64-bit integers, we only could use directed versions of the RMAT27 and Twitter graphs.

**V. Experiment Results**

**A. Execution Time**

PageRank. Table IV shows the average execution time of one iteration of PageRank on different frameworks for the graphs. (The normalized execution rate, in Traversed Edges per Second - TEPs, that is, the number of edges in the graph divided by the execution time, is presented in Figure 1.) For the GPU-only frameworks, (Nvgraph, Gunrock and Totem-1G), Totem-1G gives strongest competition to Nvgraph in most of the cases and shows 0.68×-2.71× better performance than Nvgraph. Gunrock is the slowest among the GPU-only frameworks, for both real-world and synthetic graphs, and is 1.63×-1.99× slower than Nvgraph and 1.35×-6.02× slower than Totem-1G. Some data points for larger graphs are missing as these

**TABLE II: Testbed Characteristics.**

<table>
<thead>
<tr>
<th>GPU</th>
<th>PCIe</th>
<th>L3 Cache</th>
<th>#CPU Cores</th>
<th>L1 Cache</th>
<th>L2 Cache</th>
<th>Max. Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>2× Nvidia Tesla K40c</td>
<td>13.0 × 8x16</td>
<td>70 MB</td>
<td>28</td>
<td>512 GB DDR3</td>
<td>1536 GB DDR3</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE III: Datasets used for evaluation.**

<table>
<thead>
<tr>
<th>Graph</th>
<th>#Vertices</th>
<th>#Edges</th>
<th>Max. Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orkut</td>
<td>3M</td>
<td>254M</td>
<td>33,313</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>4.8M</td>
<td>68M</td>
<td>28.8M</td>
</tr>
<tr>
<td>Road-USA</td>
<td>28.8M</td>
<td>439.994</td>
<td></td>
</tr>
<tr>
<td>RMAT22</td>
<td>4M</td>
<td>128M</td>
<td>168,729</td>
</tr>
<tr>
<td>RMAT23</td>
<td>8M</td>
<td>256M</td>
<td>272,808</td>
</tr>
<tr>
<td>RMAT24</td>
<td>16M</td>
<td>512M</td>
<td>439,994</td>
</tr>
<tr>
<td>RMAT27</td>
<td>128M</td>
<td>3B</td>
<td>1,910,241</td>
</tr>
<tr>
<td>RMAT30</td>
<td>1B</td>
<td>16B</td>
<td>7,257,172</td>
</tr>
<tr>
<td>Twitter</td>
<td>52M</td>
<td>79B</td>
<td>3,691,240</td>
</tr>
</tbody>
</table>
TABLE IV: Execution time (in seconds) for PageRank, SSSP and BFS algorithms on the workloads in Table III for all the frameworks (lower is better). Different operating modes of Totem represented as Totem-xSyG, where ‘x’ is the number of CPU-sockets used and ‘y’ is the number of GPUs used.

1Directed versions of the RMAT27 and Twitter graphs are used, since GraphMat do not support 64-bit integers.

Among the CPU-based frameworks, Totem-2S is 1.48×-5.59× faster than GraphMat, and 5.1×-32.4× faster than Galois.

On larger graphs, that is RMAT27 and Twitter, the performance advantage of the hybrid version of Totem is clearer: Totem-2S2G performs 7.51× and 50.44× faster than Galois for the RMAT27 and Twitter graphs, respectively. For GraphMat, both the graphs are directed; therefore, it traverses half the number of edges compared to Totem and Galois. Totem-2S2G is 3.91× and 2.29×, faster than GraphMat for the RMAT27 and Twitter graphs, respectively.

SSSP. Figure 2 presents the execution rate (TEPS) for SSSP. Table IV shows the average execution time on different frameworks for the graphs. Among the GPU-only frameworks, Totem-1G outperforms Gunrock by 1.04×-1.97× for all the graphs except for Road-USA (which has a very large diameter), in which case Gunrock is 4.7× faster. Nvgraph is the slowest for both the real-world and synthetic graphs. Nvgraph is 3.9×-7.4× slower than Totem-1G and 2.97×-34.93× slower than Gunrock.

For the CPU-based frameworks, Totem-2S performs the best and is 1.36×-6.62× faster than Galois, and 2.98×-7.98× faster than GraphMat.

In the hybrid mode, Totem-2S2G performs 3.24× and 2.66× better than Galois for the RMAT27 and the Twitter graphs.

---

#### Fig. 1: PageRank Billion Traversed Edges Per Second (TEPS) for all the frameworks and different operating modes for Totem represented as Totem-xSyG, where ‘x’ is the number of CPU-sockets used and ‘y’ is the number of GPUs used. Workloads, on x-axis, are presented in detail in Table III. Note, the y-axis is for TEPS (higher the better). Some graphs do not fit in GPU memory so the corresponding datapoints are missing.

#### Fig. 2: SSSP Billion Traversed Edges Per Second (TEPS) for all the frameworks and different operating modes for Totem represented as Totem-xSyG, where ‘x’ is the number of CPU-sockets used and ‘y’ is the number of GPUs used. Workloads, on x-axis, are presented in detail in Table III. Note, the y-axis is for TEPS (higher the better). Some graphs do not fit in GPU memory so the corresponding datapoints are missing.
Fig. 3: BFS Billion Traversed Edges Per Second (TEPS) for all the frameworks and different operating modes for Totem represented as Totem-xSyG, where ‘x’ is the number of CPU-sockets used and ‘y’ is the number of GPUs used. Workloads, on x-axis, are presented in detail in Table III. Note, the y-axis is for TEPS (higher the better). Some graphs do not fit in GPU memory or failed to run, so the corresponding datapoints are missing.

graph, respectively and 8.1× and 9.2× faster than GraphMat on the two large graphs.

**BFS.** Figure 3 shows execution rate (TEPS) for BFS. Table IV shows the average execution time on different frameworks for the graphs. Among GPU-only frameworks, Gunrock outperforms Nvgraph and Totem-1G by 0.84×-3.98× and 1.88×-5.65× respectively.

For the, CPU-based frameworks, Totem-2S performs the best and is 1.13×-4.39× faster than Galois, and 10.22×-44.93× faster than GraphMat.

In the hybrid mode, Totem-2S2G performs 8.26× and 4.23× faster than Galois for the RMAT27 and the Twitter graph, respectively. In comparison to GraphMat, Totem-2S2G is 84.48× and 41.85× faster than the RMAT27 and the Twitter graph, respectively.

**Summary of Key Observations**

- In almost all cases, for small graphs that fit in GPU memory, Totem-2S is faster than all the GPU-based frameworks. This is primarily because the graphs that can fit on GPU memory also benefit by the huge cache on CPUs.

- In all cases, even for smaller graphs where the overheads of coordinating a CPU and a GPU may become apparent, the hybrid solution offers a performance advantage.

- For linear algebra based frameworks, CSR format is well suited for SSSP and BFS, where computation is over out-edges. While, CSC format is apt for PageRank, where computation is over in-edges. This is primarily because of better prefetching and coalesced memory access.

- The SPMV model appears best suited for GPU architecture, and the vertex programming computation appears best suited for CPU.

**TABLE V: Memory consumption, in MB, for RMAT22 (edge list size: 512 MB) for the three graph algorithms, by all the graph processing frameworks and different operating modes for Totem represented as Totem-xSyG, where x is the number of CPU-sockets used and y is the number of GPUs used.

1 GraphMat consumes 9,354 MB during the pre-processing step.

![Graph scalability](image)

Fig. 4: GPU scalability: Speedup offered by adding a second GPU. Totem vs Gunrock for the three algorithms using the small Orkut graph. On the y-axis is 2G (two GPUs) speedup over 1G (one GPU).

These frameworks use different memory layouts for in-memory graph storage, as well as leverage supplementary data structures to achieve high performance. Table V lists total memory consumption for the three algorithms when using the RMAT22 graph. GraphMat, surprisingly, consumes 9,345 MB during the pre-processing step, the highest among all the frameworks. On average, Nvgraph requires the least amount of memory. Totem stores both the original graph edge-list as well as the respective CPU and GPU partitions in memory, therefore, ends up requiring memory about two times the size of the graph.

**B. Scalability**

We evaluate strong-scaling, i.e., how the frameworks response with increasing number of processing units.

**GPU.** For the GPU-based frameworks, in Figure 4, we see that by adding an additional GPU, Totem achieves speedup of 1.59×, 1.13× and 1.78× for PageRank, SSSP and BFS, respectively. Gunrock, on the other hand, shows better scalability for BFS only (1.98×). For PageRank and SSSP the gain is abysmal 1.22× and 1.08×, respectively.
CPU. Figure 5 presents the scalability of the CPU-based frameworks and Totem. We first explore how each framework scales from one CPU-socket to two CPU-sockets. For PageRank, GraphMat scales perfectly by $2.19 \times$, followed by Totem $(1.45 \times)$. Galois, however, show negative scaling; it runs $1.37 \times$ faster on one socket compared to two sockets. Totem-2S scales better for SSSP $(1.48 \times)$ than GraphMat $(1.37 \times)$ and Galois $(1.12 \times)$. For BFS, Galois scales, by $2.5 \times$, much better than GraphMat $(1.43 \times)$ and Totem $(1.39 \times)$.

Hybrid. Next, we study how Totem scales with addition of CPUs and GPUs. We see that for PageRank and BFS, it performs better with addition of CPUs and GPUs. Replacing a CPU-socket by a GPU makes Totem-1S1G perform $1.11 \times$ and $1.44 \times$ better than Totem-2S for PageRank and BFS, respectively. But for SSSP, replacing one socket by one GPU or adding an extra GPU makes it slower. It only performs better for Totem-2S2G, which is $1.3 \times$ faster than Totem-2S.

Finally, we note some superlinear speedups (e.g., GraphMat with PageRank, and Galois with BFS) which we attribute to efficient cache utilization.

C. Energy Consumption

Figure 6 presents energy consumed by the GPU frameworks and Totem for the Orkut workload. We observed that Totem-1G consumes the least amount of energy for SSSP and is $4.1 \times$ and $1.42 \times$ more energy efficient than Nygraph and Gunrock, respectively. For PageRank, Nygraph holds slight advantage over Totem-1G, which is $1.09x$ more energy efficient than Totem-1G. For BFS, Gunrock outperforms Totem-1G and it consumes $5.26 \times$ less energy.

Furthermore, we observe that both Totem-2S as well as hybrid mode, are ‘greener’ than the GPU-based frameworks even on this small graph (except for BFS). In Figure 7, we compare the energy efficiency of Totem with the CPU-only frameworks on the larger Twitter graph. We see that Totem outperforms both GraphMat and Galois for all three algorithms. For all three algorithms, all three modes of Totem consumes similar amount of energy (because even though hybrid mode consumes more power, time-to-solution decreases accordingly). Totem-2S is $2.14 \times 33.58 \times$ and $1.98 \times 24.47 \times$, more energy efficient than Galois and GraphMat, respectively. From both Figures 6 and 7, we notice that BFS is the most energy efficient while SSSP is the least energy efficient algorithm, primarily because it causes more memory accesses compared to BFS. In SSSP, an edge is traversed multiple times, whereas in an optimized BFS implementation, an edge is visited only once. Although, in one iteration, PageRank traverses the same number of edges as BFS, they are vastly different in terms of creating work items (i.e., the way vertices are scheduled to visit their neighbors) and memory bandwidth utilization.

Additionally, we compare Totem-2S2G with Totem CPU-only on four CPU-sockets (i.e., Totem-4S) in Figure 8. Totem-4S consumes $2.4 \times$, $2.8 \times$, and $5.3 \times$ more energy for PageRank, SSSP and BFS than Totem-2S2G. However, Totem-2S2G is $1.48 \times$, $1.66 \times$, and $3.22 \times$ faster than Totem-4S for the respective algorithms. This confirms all-around advantages of the hybrid solution.
Fig. 8: Totem-4S vs Totem-2S2G: Comparison of execution time and energy consumption for RMAT30 (directed) graph. On the x-axis are the algorithms. On the Left y-axis is the execution time (second) (lower is better). Right y-axis is for energy consumption (watt-second) (lower is better).

VI. SUMMARY AND DISCUSSION

We make a number of interesting observations. First, our results suggest that SPMW programming model offers performance advantages on the GPU, while vertex programming is a better match for the CPU. Second, among the GPU frameworks, there is not a clear winner, though in most of the cases Totem-1G performs better than other GPU-only frameworks, except for BFS. Third, among the CPU-only frameworks, Totem-2S performs better than both Galois and GraphMat, for all the three algorithms. Overall, we find that Totem hybrid not only outperforms every other framework, but is also greenest of all.

Given a CPU-based system can process larger graphs (than GPU-only processing) and in many cases outperform the GPU-only solutions, graph processing on GPUs makes most sense in a hybrid setting, rather than in isolation on the GPU. Future hybrid graph processing solutions should leverage this finding and should exploit both computation approaches to harvest maximum performance of the heterogeneous processing units.

REFERENCES


