SYgraph: Accelerating Heterogeneous Graph Analytics Applications

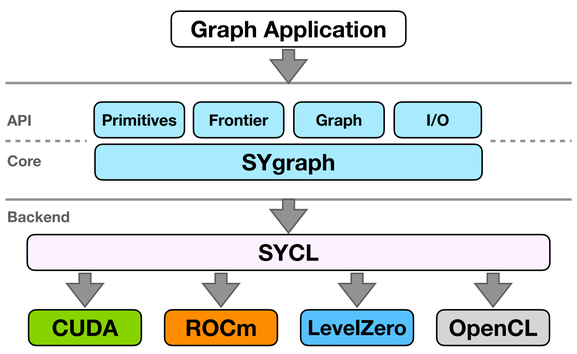
Antonio De Caro, Biagio Cosenza
Department of Computer Science, University of Salerno, Italy
{antdecaro, bcosenza}@unisa.it

ABSTRACT

Graph analytics play a crucial role in a wide range of fields, including social network analysis, bioinformatics, and scientific computing, due to their ability to model and explore complex relationships. However, optimizing graph algorithms is inherently difficult due to their memory-bound constraints, often resulting in poor performance on modern massively parallel hardware. In addition, most state-of-the-art implementations are designed for NVIDIA GPUs, limiting their applicability on supercomputers equipped with AMD and Intel GPUs, for example. To address these challenges, we propose SYgraph, a portable heterogeneous graph analytics framework written in SYCL. SYgraph provides an efficient two-layer bitmap data layout optimized for GPU memory, eliminates the need for pre- or post-processing steps, and abstracts the complexity of working with diverse target platforms. Experimental results demonstrate that SYgraph delivers competitive performance on NVIDIA GPUs while being able to target any SYCL-supported device, such as AMD and Intel GPUs.

SYGRAPH OVERVIEW

SYgraph offers a user-friendly API for defining and executing graph algorithms, making *graph analytics accessible on GPU without requiring deep hardware expertise*. The API abstracts the concept of Frontier [1]—the set of active elements in a graph computation—providing four core operators to efficiently manage and manipulate these data structures.



- the **Application Layer** provides a straightforward API that enables graph analytics *without requiring deep hardware knowledge*;
- the **SYgraph Core** *manages load-balancing* on essential primitives such as *advance*, *compute*, and *filter* to operate on active graph frontiers;
- and the **SYCL Layer** that enables *portability*.

SYGRAPH API

The SYgraph API simplifies graph algorithm development with three core methods for managing frontiers:

- **Advance** discovers new vertices based on a lambda condition;
- **Filter** removes vertices that meet a specified condition;
- **Compute** executes operations on active vertices defined by a lambda.

These methods abstract complexity, allowing efficient implementation of graph algorithms. Below an implementation of BFS and SSSP algorithms using SYgraph.

BFS

```
Graph& G = readGraph();
size_t distances = sycl::malloc_device(...)
auto in_frontier = makeFrontier<frontier_view_t::vertex>(G);
auto out_frontier = makeFrontier<frontier_view_t::vertex>(G);

size_t iter = 0;

while (!in_frontier.empty()) {
  operators::advance::frontier(G, in_frontier, out_frontier,
  [=](vertex_t u, vertex_t v, edge_t edge, weight_t weight) {
      bool unvisited = (iter + 1) < distances[v];
      return unvisited;
  }).wait();

  operators::compute(G, out_frontier,
  [=](vertex_t v) {
      distances[v] = iter + 1;
  }).wait();

  frontier::swap(in_frontier, out_frontier);
  out_frontier.clear();
  iter++;
}
```

SSSP

```
/* Init Data Here */
size_t iter = 0;
while (!in_frontier.empty()) {

 operators::advance::frontier(G, in_frontier, out_frontier,
  [=](vertex_t u, vertex_t v, edge_t edge, weight_t weight) {
    weight_t u_dst = sync::load(&distances[u]);
    weight_t v_dst = u_dst + weight;
    weight_t recover_dst = sync::load(&distances[v]);
    recover_dst = sync::min_cas(&(distances[v]),
                                &(v_dst));
    return (v_dst < recover_dst);
  }).wait();

  operators::filter::external(G, out_frontier, in_frontier,
  [=](vertex_t v) {
    if (visited[v] == iter) return false;
    visited[v] = iter;
    return true;
  }).wait();
  out_frontier.clear();
  iter++;
}
```

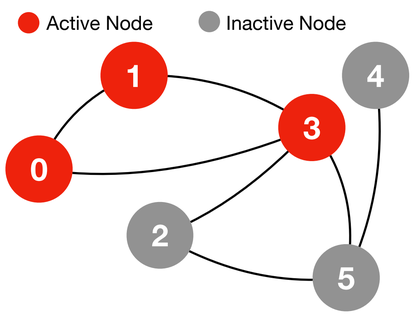
In purple are highlighted the SYgraph's core functionalities, while the highlighted lines are offloaded on the GPU and executed by each work-item.

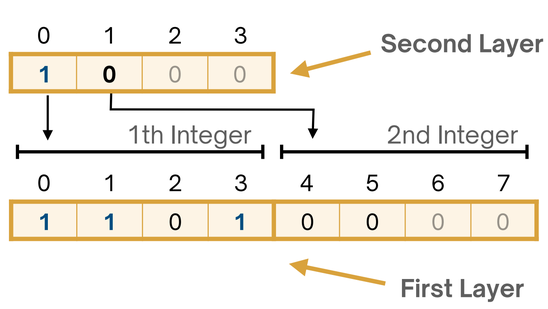
TWO-LAYER BITMAP FRONTIER

SYgraph uses a two-layer bitmap layout to efficiently represent active graph vertices and reduce memory overhead:

- **First Layer** marks active vertices;
- **Second Layer** identifies integers with active bits.

This structure eliminates the need for post-processing after advance operations to remove redundant vertices while enhancing memory compression.



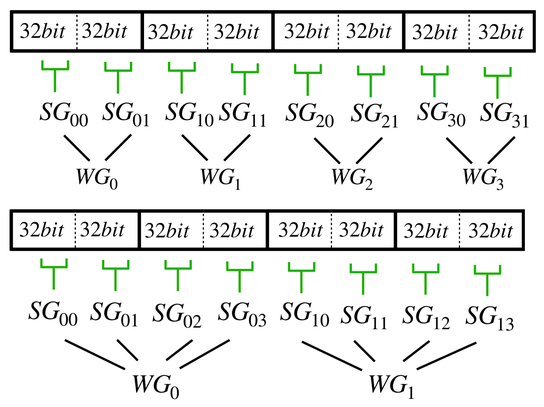


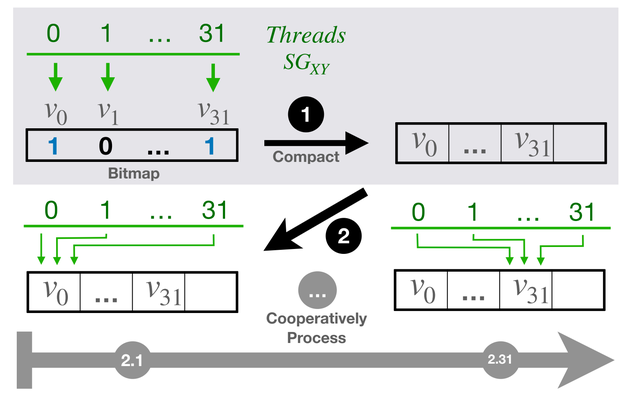
For representation purposes we consider 4-bit integers

IMPLEMENTATION

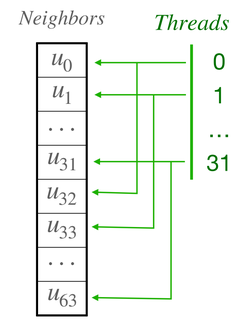
SYgraph maps graph algorithms to GPUs using SYCL queues, with graphs stored in CSR format in global memory. Three core primitives—advance, filter, and compute—are implemented with *parallel for* loops:

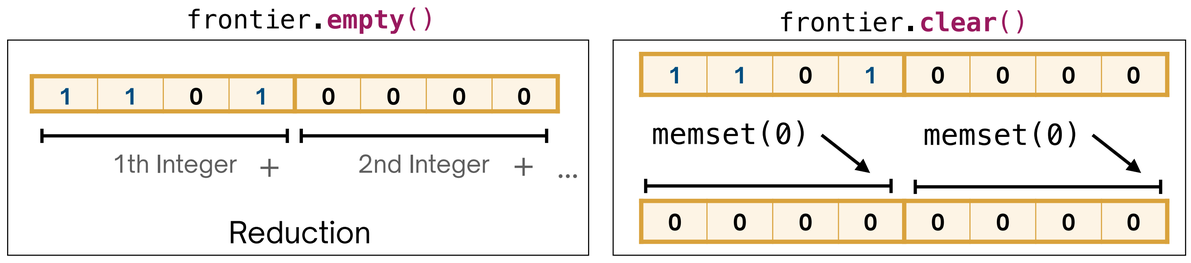
- **Filter** and **Compute**: Assign active vertices to individual work-items.
- **Advance**: Uses a load-balancing strategy to prevent imbalance from varying vertex degrees.





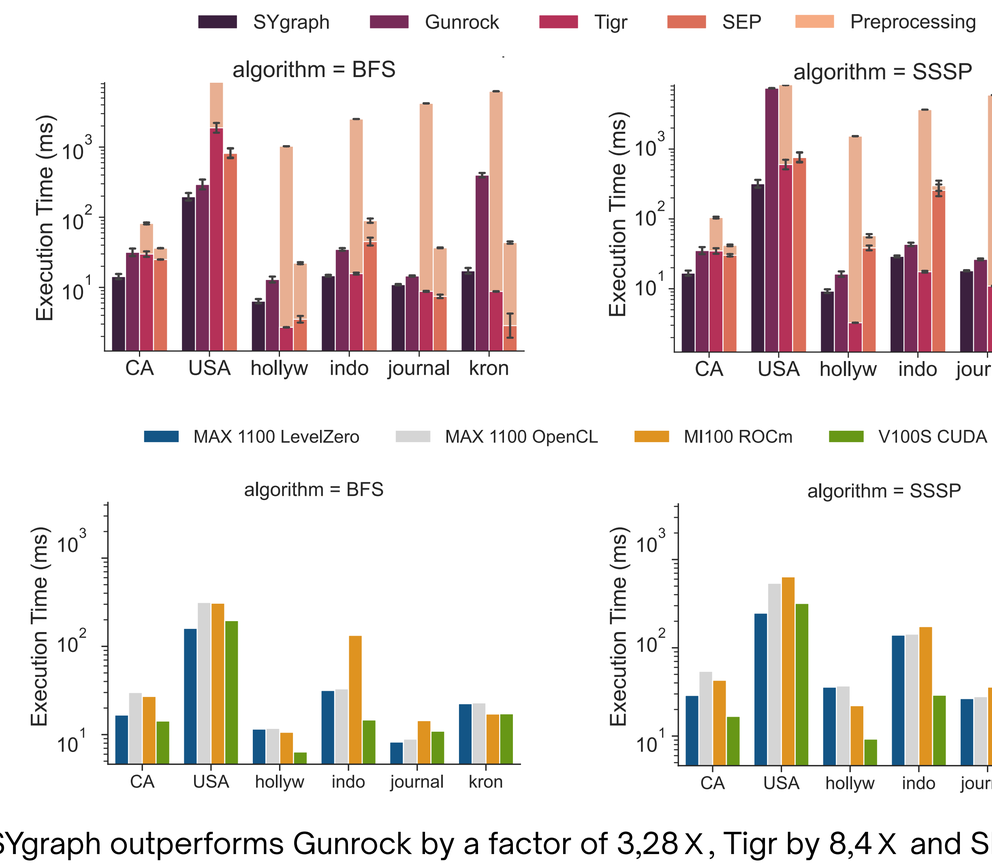
In the **advance** operation, workgroups are dynamically assigned bitmap integers based on the compute unit's capacity. Active vertices are compacted into local memory using a *scan operation*.
Subgroups then collaboratively process slices of the compacted vertices, handling one vertex's neighborhood at a time in a *coalesced manner* for optimal parallelism.





RESULTS

We tested SYgraph on Breadth First Search (BFS) and Single Source Shortest Path (SSSP) against Gunrock [1], Tigr [2], and SEP-Graph [3] on a NVIDIA V100S. Then we evaluated SYgraph on multiple target architectures. All the tests were conducted by using *oneAPI 2024.2.0*.



SYgraph outperforms Gunrock by a factor of 3,28 X, Tigr by 8,4 X and SEP-Graph by 2,53 X.

1. Wang, Yangzihao, et al. "Gunrock: A high-performance graph processing library on the GPU." Proceedings of the 21st ACM SIGPLAN symposium on principles and practice of parallel programming. 2016.
2. Nodehi Sabet, Amir Hossein, Junqiao Qiu, and Zhijia Zhao. "Tigr: Transforming irregular graphs for gpu-friendly graph processing." ACM SIGPLAN Notices 53.2 (2018): 622-636.
3. Wang, Hao, et al. "SEP-graph: finding shortest execution paths for graph processing under a hybrid framework on GPU." Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming. 2019.