# miniVite + Metall: A Case Study of Accelerating Graph Analytics Using Persistent Memory Allocator

Keita Iwabuchi (LLNL), Sayan Ghosh (PNNL), Roger Pearce (LLNL), Mahantesh Halappanavar (PNNL), Maya Gokhale (LLNL)

## 1) Background

### Large-scale graph processing

- Remarkable achievements in accelerating graph analytics algorithms
- Often ingest the same data — construct/generate graph data with a proper data structure — before running an analytics
  - Preparing data is often slower than the analytics itself

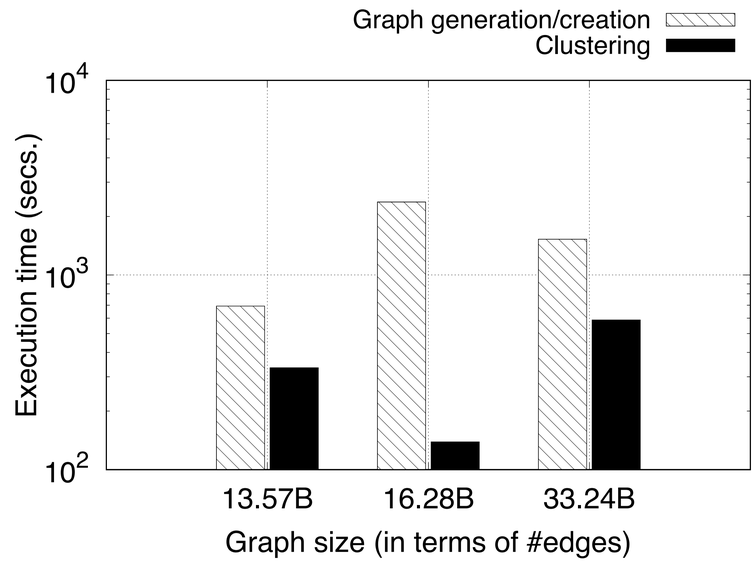
### Persistent memory (PM)

- Substantial performance improvements and cost reductions
- Stored data can be re-analyzed and updated beyond the lifetime of a single execution

Can we accelerate real graph workloads by storing graph data into persistent memory?

## 2) miniVite[Ghosh'18]

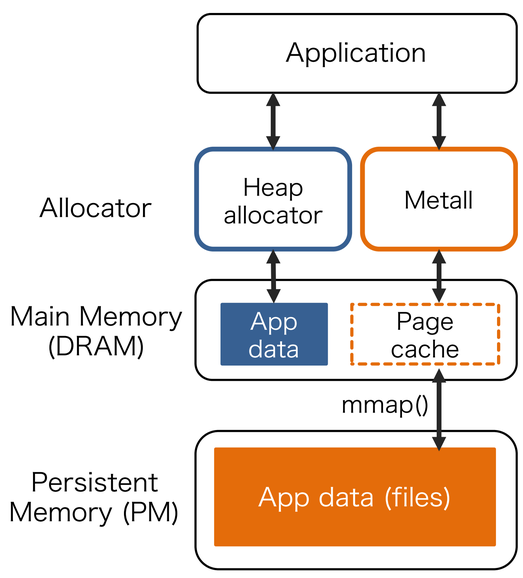
- Implements the very first phase of the Louvain method (graph clustering/community detection) without rebuilding the graph
- Part of the U.S DOE ECP proxy application suite
- Graph generation part can be up to 20X slower than graph clustering




Performance of graph generation/creation as compared to clustering in miniVite 8192 processes at ALCF Theta.

## 3) Metall[Iwabuchi'19]

- C++ Allocator for Persistent Memory (PM)
- Enables applications to allocate heap-based objects into persistent memory
- Applications can access persistent memory region as if it were DRAM
- Provides the API developed by Boost.Interprocess to allocate custom C++ objects, including STL containers, into PM




- No serialization
- Kernel performs demand paging (page in/out on demand)

## 4) miniVite + Metall

- Added modes to store/load a graph to/from persistent memory
- Changed the original graph class to an allocator-aware one

```
template<class vertex_id_type, class allocator>
class graph {
  std::vector<vertex_id_type, allocator_type> vec;
  // Two more vectors here

  // Constructor takes an allocator object
  graph(allocator alloc) : vec(alloc) {}

  // No change in the other part
};
```

- Allocate a graph object using Metall

```
void create() {
 metall::manager mgr(metall::create_only, "/ssd/data");
 graph_t* g = mgr.construct<graph_t>("graph")
                                (mgr.get_allocator<int>());
 // Construct a graph. No code change here.
}
```

- Load a graph object using Metall

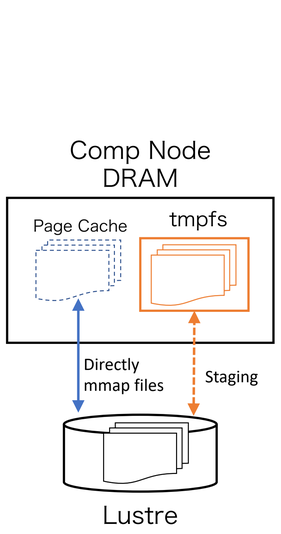
```
void Load() {
 metall::manager mgr(metall::open_only, "/ssd/data");
 graph_t* g = mgr.find<graph_t>("graph").first;
 // Run analytics (clustering). No change here.
}
```

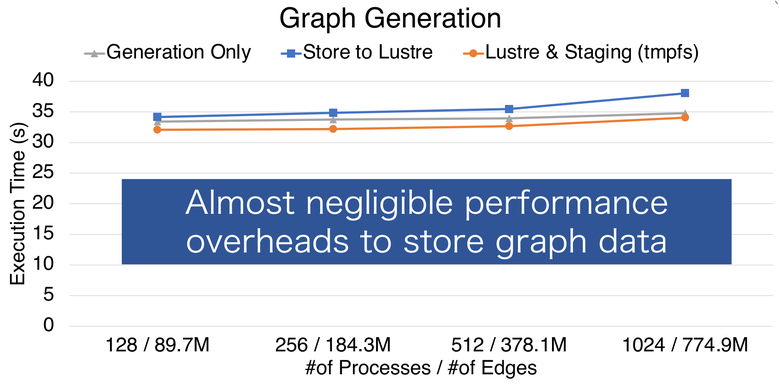
## 5) Evaluation

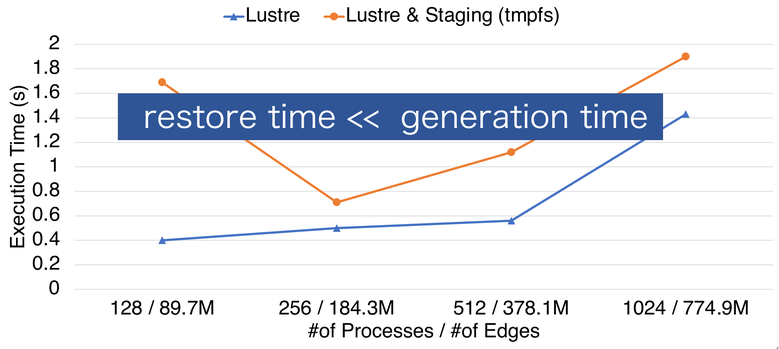
- Evaluate the graph generation/storing time and restoring time on two systems
- Staging: use tmpfs/local SSD as a scratchpad — copy files at the beginning and the end of each job

### NERSC Cori

(clustering step took 0.9 - 1.4 s)

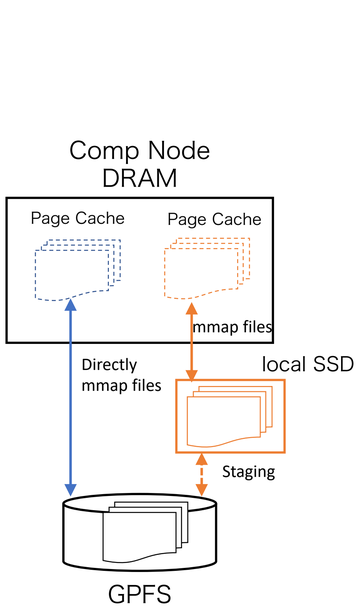


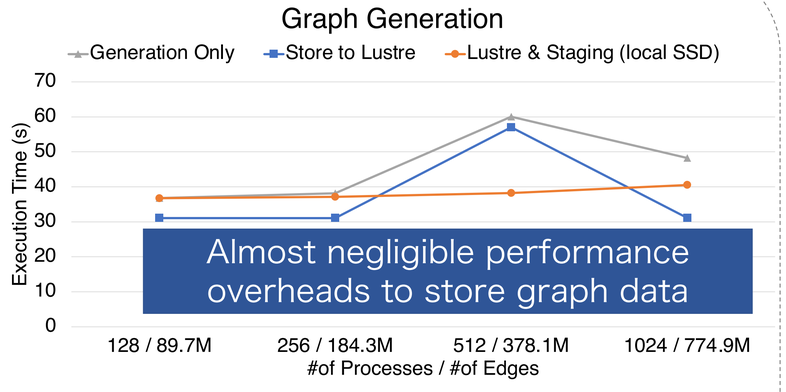



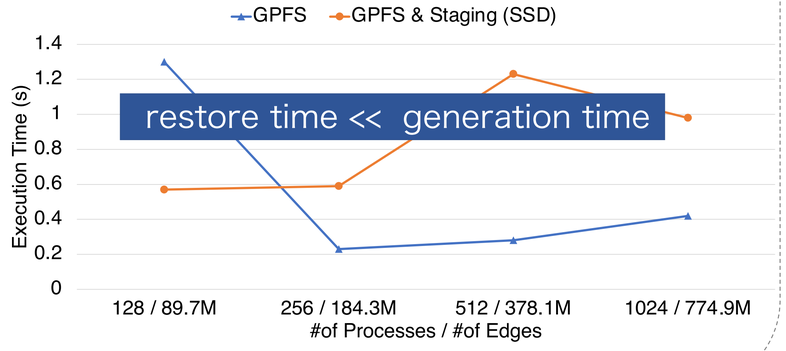




### OLCF Summit

(clustering step took 1.2 - 1.8 s)






Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-POST-813828).