GBTL + Metall – Adding Persistence to GraphBLAS

Kaushik Velusamy[1], Scott McMillan[2] ,Keita Iwabuchi[3], Roger Pearce[3]

University of Maryland at Baltimore County[1], Software Engineering Institute, Carnegie Mellon University[2]

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory[3]

Abstract

It is well known that software-hardware co-design is required for attaining high-performance implementations. System software libraries help us in achieving this goal. **Metall** persistent memory allocator is one such library.

Having a large capacity of persistent memory changes the way we solve problems and leads to algorithmic innovation.

In this work, we present **GraphBLAS** as a real application use case to demonstrate Metall persistent memory allocator benefits. We show an example of how storing and re-attaching graph containers using Metall, eliminates the need for graph reconstruction at a one-time cost of re-attaching to Metall datastore.

Metall

- Metall relies on a file-backed **mmap** mechanism to map a file in a file system into the virtual memory of an application and access as if it were regular memory.
- Such mmap mappings allows applications to address datasets larger than physical main-memory (out-of-core or external memory).
- Metall provides persistent memory snapshotting (versioning) capabilities.
- Metall provides the ability to create a persistent object for out-of-core graph algorithms with very large datasets and thereby allow real-time access to large datasets.

Example of using graph algorithms with GBTL containers and Metall

```
using metall_matrixType = grb::Matrix <T, metall::manager::allocator_type<char>>;

//================= Graph Construction in Metall Scope ========================
{
    metall::manager manager(metall::create_only, "/mnt/nvme/datastore");
    metall_matrixType *A = manager.construct< metall_matrixType>("gbtl_lil_matrix")
                                    (NUM_NODES, NUM_NODES, manager.get_allocator());

    myMatrix ->build(iA.begin(), jA.begin(), v.begin(), iA.size());

}

                //  Exist the program and reattach the data
//================= Triangle Counting in Metall Scope ========================
{
    metall::manager manager(metall::open_only, "/mnt/nvme/datastore");
    metall_matrixType *A = manager.find< metall_matrixType >("gbtl_lil_matrix").first;

    algorithms::triangle_count_masked_noT(*myMatrix);

}

                //  Exist the program and reattach the data
//================= single BFS in Metall Scope ================================
{
    metall::manager manager(metall::open_only, "/mnt/nvme/datastore");
    metall_matrixType *A = manager.find<metall_matrixType >("gbtl_lil_matrix").first;
    grb::Vector<T> parent_list(NUM_NODES);
    grb::Vector<T> root(NUM_NODES);
    root.setElement(0, 0);

    algorithms::bfs(*myMatrix, root, parent_list);

}
```

Results

- BFS algorithm time is much shorter than the graph construction time.
- Metall provides a mechanism to exit the program and reattach to the previously created data, avoiding construction time (Listing 2).
- This would be helpful to many graph analytics applications where the data structure reconstruction can be completely avoided.

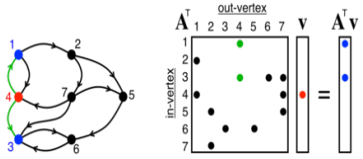
Graph Analytics

Some of the main problems in graph analytics are

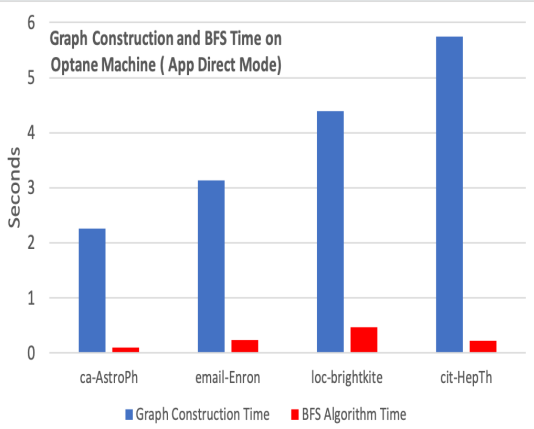
- need to **persist** the data beyond the scope of a single execution.
- Graph algorithms suffer from **irregular access patterns** that may limit their performance even on DRAM.
- Graph construction, indexing and regular updates are often more **expensive** than the analytics itself.

GraphBLAS

BFS with GraphBLAS [Img Ref.Wiki]



- Building blocks for computing on graph-structured data expressed in the language of linear algebra.
- The **GraphBLAS Template Library (GBTL)** is a C++ reference implementation of the GraphBLAS specification.



Conclusion

- Memory-mapped persistent pre-built data structures is helpful in enabling interactive real time data science applications
- Application developers can create custom complex persistent and consistent data structures.
- This ability to attach and detach from previously created datasets in a lightweight manner gives a powerful workflow software productivity benefit.

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-POST-819947) and DOD CMU SEI Contract FA8702-15-D-0002 [DM21-0180]