VM based Evaluation of the Scalable Parallel Minimum Spanning Tree Algorithm for PGAS Model

Authors: V. Bejanyan (MSc), H. Astsatryan (PhD, HDR)

Organization: Center for Scientific Computing,
Institute for Informatics and Automation Problems,
National Academy of Sciences of the Republic of Armenia

Conference: 9th International Conference "Distributed Computing and Grid Technologies in Science and Education", Dubna, Russia

Date: June 10, 2021

Contents

1. Problem statement - MST

MST is a building block for many applications, such as finding approximate solutions for complex mathematical problems. There are several well-known MST algorithms, such as:

- Prim - Repeatedly adds minimum cost edge to the MST with one end inside the tree and another outside.
- Kruskal - Builds the forests of the minimum spanning tree and joins them by minimum cost edges.
- Boruvka - Keeps track of connected components, initially, every component consists of a single vertex, then merges components by minimum cost edge.

1. Problem statement - Traditional approaches

Usually parallelized in terms of shared memory (e.g. OpenMP intra-node):

- Memory can act as a limiting factor for shared memory systems.

Distributed memory (MPI inter-node):

- Graphs usually implemented using adjacency matrices.
- Matrices are copied over different ranks as chunks.
- Not well suited for computations with irregular access patterns.

1. Problem statement - Emerging alternatives (PGAS)

No known implementation using UPCXX for Partitioned Global Address Space (PGAS).

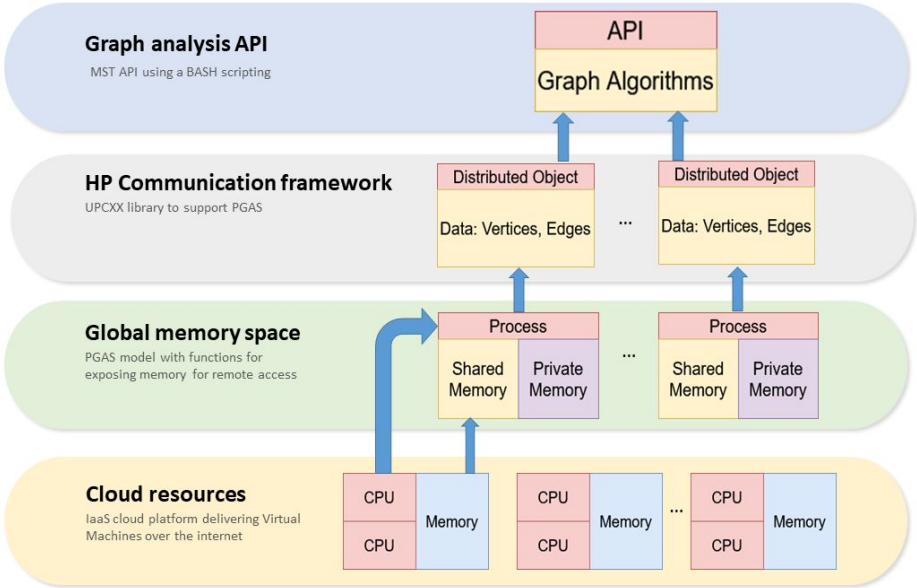
UPCXX allows us to:

- Base computation on asynchronous communication implemented in terms of futures.
  - Remote procedure calls (RPC) to execute code on different processing nodes.
  - RPC returns a feature helpful for computation and communication overlapping.
- Store data in public address space and share using distributed objects.
  - Distributed objects are passed as modifiable rank-aware arguments to the RPCs.
  - Built-in serialization for several STL containers.
- Barriers to global synchronization.

1. Problem statement: objective

- To study PGAS model using UPCXX:
  - Implement graph generation.
  - Implement MST solver.
  - Benchmarking and evaluations to identify bottlenecks and possibilities for future improvements.

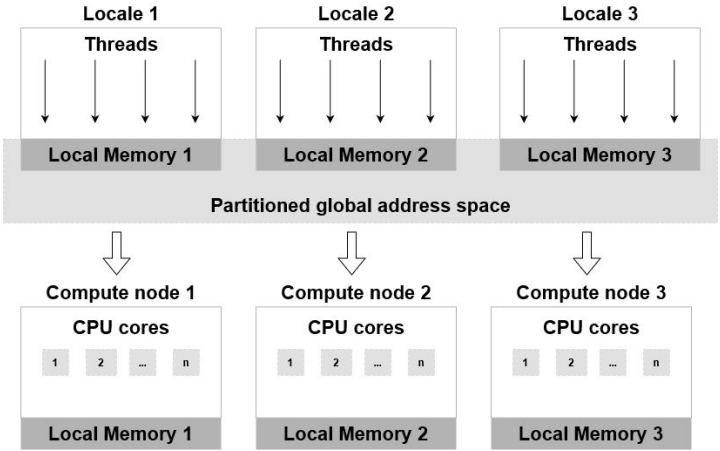
2. Our solution - Overview



2. Our solution - PGAS

A PGAS system consists of the following components:

- A set of processors, each with attached local storage.
- A mechanism by which at least a part of each processor's storage can be shared with others.
- Every shared memory location has an affinity - a processor on which the location is local and therefore the access is quick.



2. Our solution - UPCXX essentials

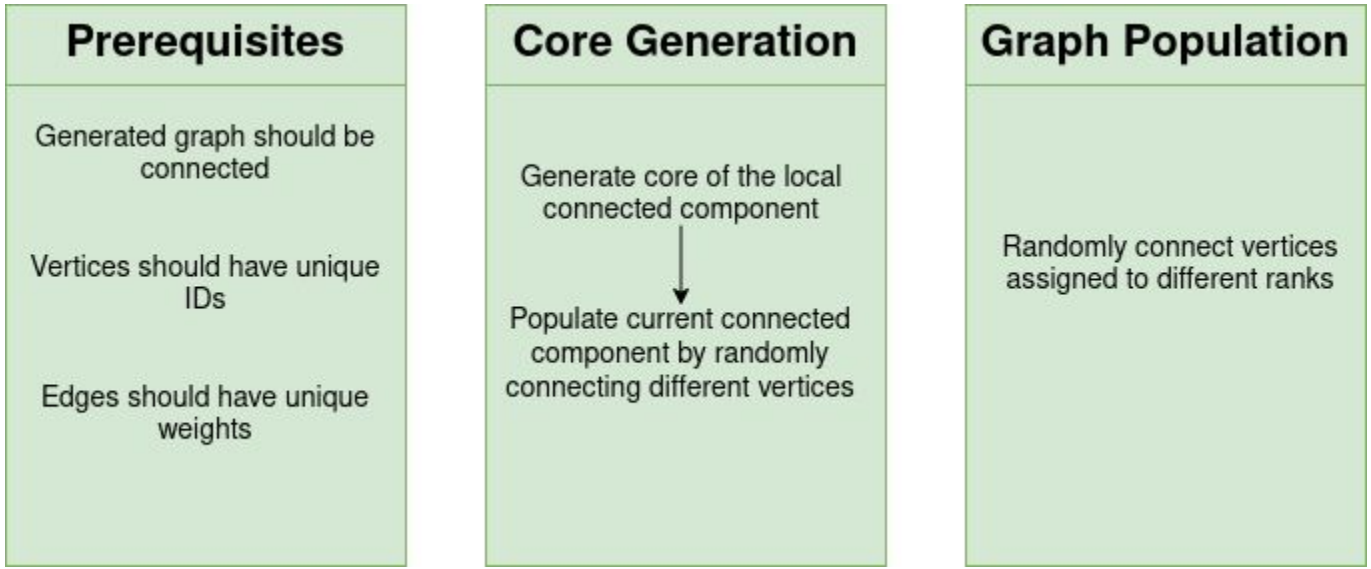
UPC++ is a C++ library to support the PGAS programming, and is designed to interoperate smoothly and efficiently with MPI, OpenMP, CUDA. It leverages GASNet-EX to deliver low-overhead, fine-grained communication, including Remote Memory Access (RMA) and Remote Procedure Call (RPC).

- RMA. UPC++ provides asynchronous one-sided communication (RMA, a.k.a. Put and Get) for the movement of the data among processes.
- RPC. UPC++ provides asynchronous RPC for running code (including C++ lambdas) on other processes.

2. Our solution - UPCXX essentials

- **Futures, promises and continuations**. Futures are central to handling asynchronous operation of RMA and RPC. UPC++ uses a continuation-based model to express task dependencies.
- **Global pointers and memory kinds**. UPC++ provides uniform interfaces for RMA transfers among host and device memories, including a reference implementation for CUDA GPUs.
- **Distributed objects**. UPC++ enables construction of a scalable distributed object from any C++ object type, with one instance on each rank of a team. RPC can be used to access remote instances.
- **Serialization**. UPC++ introduces several complementary mechanisms for efficiently passing large and/or complicated data arguments to RPCs.

2. Our solution - Graph generation algorithm



2. Our solution - Graph generation algorithm

- Lists of connected and unconnected vertices are used to ensure connectivity of the core. Every time the vertex is added to the core it's being removed from unconnected vertices list and added into connected.
- Distributed counter is used to ensure that there are no duplications in edge weights.
- Number of the extra edges is connected based on the connectivity percentage to populate the core.
- Ranks are iterated in a pairware manner to populate graph with inter node edges. Lambda is used as a mapper to translate random vertex ID into local ID.

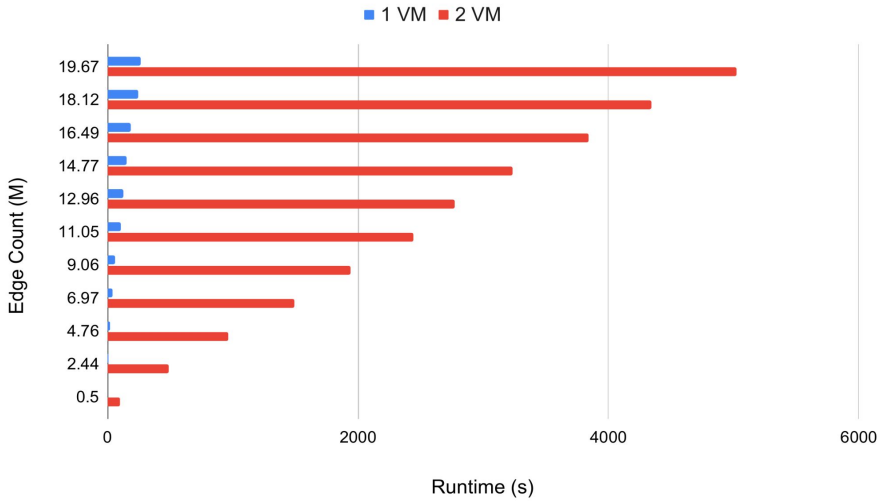
3. Evaluation results - Computational architecture

The nano, micro, small, medium, and large type instances of the Armenian IaaS cloud have been used

Instance size	Memory (GiB)	vCPU
am16.nano	16	1
am32.medium	32	1
am16.2xnano	16	2
am32.2xmedium	32	2
am16.micro	16	4
am32.xlarge	32	4
am16.2xmicro	16	8
am32.2xlarge	32	8
am16.small	16	16
am32.4xlarge	32	16

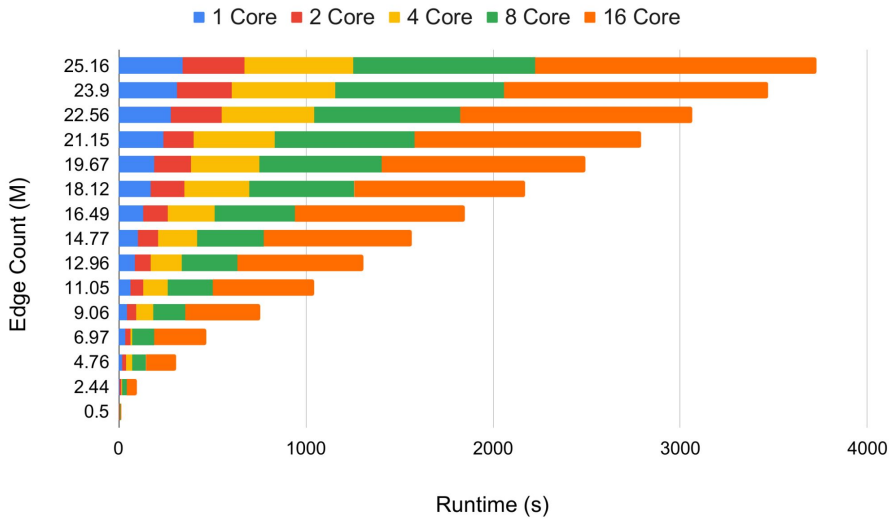
3. Evaluation results - Graph generation

Multi-node with distributed memory

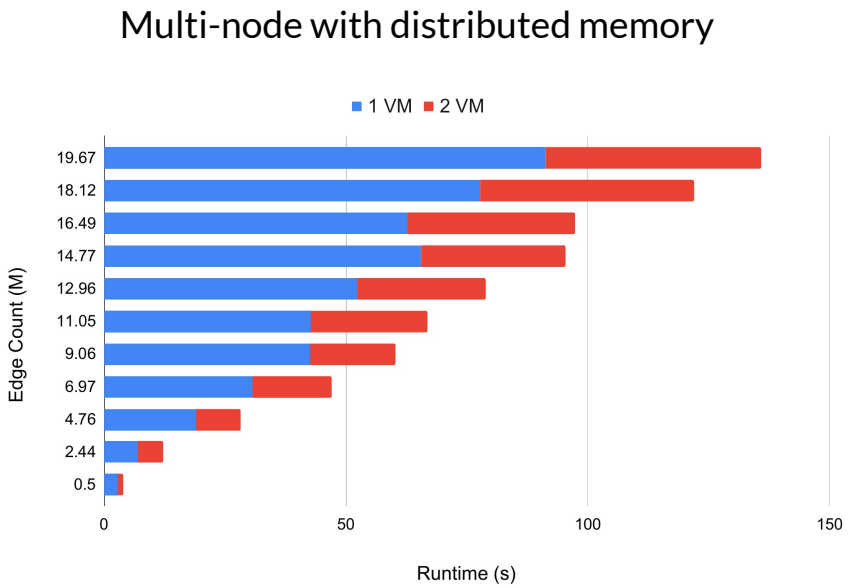


3. Evaluation results - Graph generation

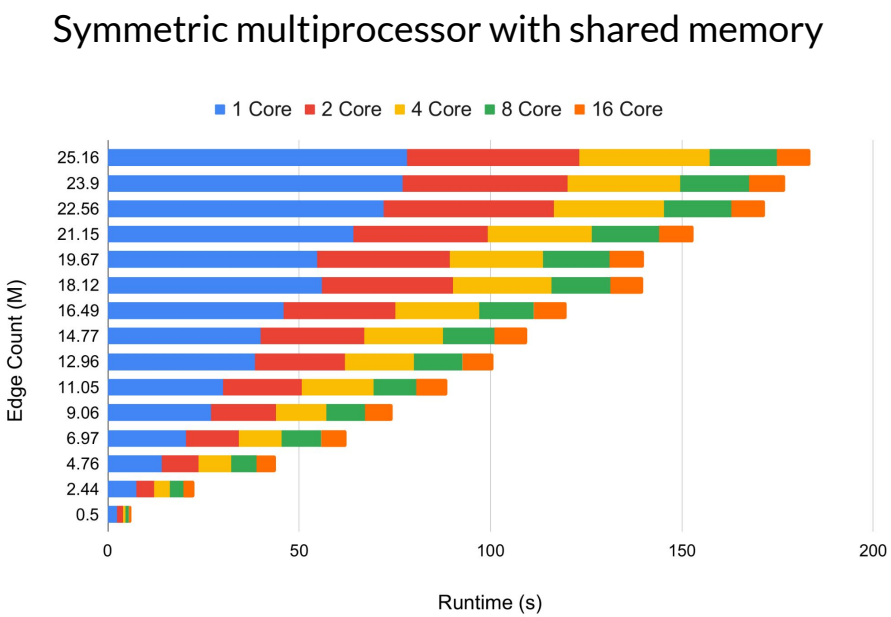
Symmetric multiprocessor with shared memory



3. Evaluation results - MST



3. Evaluation results - MST



4. Conclusion

- The Kruskal algorithm modification for PGAS model using UPCXX presented.
- Experimental results shows that:
  - Graph generation has pool scalability for both shared and distributed memory systems
  - MST has high scalability for both shared and distributed memory systems
- The implementation and all results are available at https://github.com/lnikon/pgas-graph

5. Future Work

- Planned to explore scalable and high performance algorithms for graph clustering, centrality, link analysis in the scope of open source PGAS graph algorithms library.
- Investigate Chapel Programming Language capabilities in the scope of GraphBLAS project.
- Provide easily deployable client-server application for HPC graph experimentation on cloud.

Thank you,

any questions?