On the Cloud We Can't Wait: Asynchronous Actors Perform Even Better on the Cloud

Anonymous Authors

Abstract. This study investigates the performance of asynchronous actor programming and synchronous Partitioned Global Address Space (PGAS) versions of graph kernels on a Cloud platform and a High-Performance Computing (HPC) platform. Using the Bale suite of graph microkernels, we compare the execution times of kernels implemented with OpenSHMEM (synchronous PGAS) and HClib-actor (asynchronous) on both Azure Cloud with Ethernet and an HPC cluster with InfiniBand. Our results reveal significant performance differences between these platforms. While the asynchronous version outperforms the synchronous version in both settings, the performance gap is dramatically wider on the Cloud platform, with the asynchronous one showing up to 1,000x improvement over the synchronous one in some cases. We also observe highly variable execution times in the Cloud, likely due to shared resources and unpredictable data center traffic. These findings highlight the critical importance of choosing appropriate programming models for different computational platforms, especially as Cloud platforms are becoming more affordable and easier to access than traditional HPC clusters. Our work provides valuable insights for both researchers and practitioners in optimizing parallel programming strategies across diverse computational settings.

Keywords: The Actor Model · PGAS · Cloud Computing · High-Performance Graph Analytics.

1 Introduction

1.1 Background

Large-scale clusters have grown rapidly in recent times - a phenomenon understood to be the consequence of the end of Moore's law and Dennard scaling, which has led to limits on the potential performance gain from a single chip. This has led to the rise of scalable programming models that can utilize the capabilities of such clusters effectively. In particular, there has been a growing interest in employing the Partitioned Global Address Space (PGAS) model [11], which gives the programmer an illusion of shared memory programming for such large-scale platforms.

Coordination between nodes in PGAS is typically done with Bulk-Synchronous-Parallel (BSP) communication, but recent work [10, 9] shows the promise of enabling actor-based fine-grained asynchronous messaging with message aggregation in large-scale graph applications. Specifically, [10, 9] shows that the abstraction of asynchronous actor/selector programming allows graph algorithms to be

expressed very efficiently while allowing for intuitive programming. HClib-actor is a PGAS runtime that leverages the concept of selectors — independent computational entities, or actors, that communicate via message passing—to achieve high levels of concurrency without the pitfalls of traditional locking mechanisms. Selectors are derived from [6] and are a modification of the more conventional actor model (originally proposed in the 1970s [5]), where each actor possesses multiple mailboxes.

Cloud computing and high-performance computing (HPC) clusters represent two prevalent environments where these programming strategies are deployed. Cloud computing offers scalable, on-demand resources via platforms like Azure, AWS, GCP, and numerous others, enabling flexible resource allocation and cost efficiency. Historically, Cloud infrastructures were predominantly utilized for internet services, which relied on inexpensive off-the-shelf hardware, with their computing capabilities not matching those of dedicated HPC clusters. This disparity was not only due to the differences in processing power but also because of the faster communication networks used in clusters, like Infiniband, compared to the slower Ethernet-based connections in Cloud setups. However, recent advancements have significantly bridged this gap. Improvements in Cloud technologies, such as the availability of specialized interconnects like Mellanox Infiniband in the public Cloud, have enhanced communication efficiencies, making Cloud infrastructures more comparable to traditional clusters. Despite these advancements, the Cloud environment still poses unique challenges, such as issues related to virtualization and the non-deterministic nature of locality, which can affect performance. Conversely to the heterogeneity of the Cloud, HPC clusters are characterized by dedicated, largely homogeneous hardware and optimized network topologies designed for maximum performance and efficiency in executing large-scale computations.

1.2 Motivation

Despite the extensive use of asynchronous actor programming and blocking strategies, there remains a gap in understanding how these paradigms perform in different computational environments, particularly Cloud versus HPC clusters. Industry practitioners and researchers often assume that the performance characteristics observed in one environment will translate to another, yet this is not always the case. The unique architectural and operational differences between Cloud platforms and HPC clusters can significantly impact the performance of parallel programming models.

Our preliminary experiments have shown that the performance difference between asynchronous message passing and blocking strategies is more pronounced in the Cloud compared to traditional HPC clusters. This observation prompts a deeper investigation into the factors contributing to this discrepancy. Understanding these factors is crucial for making informed decisions about deploying concurrent systems in various environments and optimizing performance and resource utilization.

1.3 Objectives

This work studies the performance of asynchronous actor programming and synchronous PGAS strategies in Cloud environments versus HPC clusters. More specifically, we:

- Measure and compare the performance of graph kernels written in HClib-actor (an asynchronous actor programming runtime) and an equivalent non-asynchronous version on both Azure Cloud and HPC clusters.
- We identify and analyze the factors contributing to the observed performance differences in these environments and provide insights and recommendations for industry practitioners on deploying these programming strategies effectively in different computational settings.

By addressing these objectives, this paper seeks to bridge the knowledge gap between theory and practice in the realm of HPC-on-the-Cloud, offering valuable guidance to academia and industry. The findings of this study are expected to inform future research and development in optimizing the deployment of asynchronous and blocking strategies in diverse computational environments.

2 Related Work

2.1 Partitioned Global Address Space

PGAS is a parallel programming model that provides a global memory address space partitioned among the processors. Languages such as Unified Parallel C (UPC)[2], Chapel [1], and X10 [4] have implemented PGAS concepts to simplify the memory access semantics, thereby enhancing the productivity and performance of parallel application development.

The OpenSHMEM programming model [3], one implementation of PGAS, provides efficient one-sided communication primitives that allow a process to directly access the memory of another process without the involvement of the target process's CPU. This feature is particularly beneficial for achieving low-latency communication in high-performance computing (HPC) environments.

2.2 Asynchronous Actor Programming

The asynchronous Actor programming model has been the subject of active research and development for several decades. Initially introduced in the late 1970s, it is designed to provide a natural abstraction for concurrent computation. In this model, "actors" are the fundamental units of computation that encapsulate state and behavior, communicate via asynchronous message passing, and make decisions based on the messages they receive. This decoupling of computation and communication helps achieve high concurrency and scalability levels.

Recent advancements in Actor-based programming frameworks, such as Akka, Erlang, and Orleans, have demonstrated the effectiveness of this model in building robust, scalable, and fault-tolerant distributed systems. These frameworks

leverage the actor model to manage complex concurrency issues, making them popular choices for developing distributed applications in industry and research settings. The actor model's adaptability to various domains, including real-time data processing, telecommunications, and Cloud computing, underscores its relevance and potential.

2.3 Cloud Environments

With the growing adoption of Cloud computing for large-scale computational tasks, there has been increasing interest in evaluating the performance of parallel programming models in Cloud environments. Cloud platforms such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud offer scalable and cost-effective solutions for running parallel applications. Still, they also introduce new challenges, such as network virtualization, variable latency, and resource contention.

Little research explores how these Cloud-specific factors impact the performance of asynchronous actors and PGAS models. The authors believe that Cloud environments' inherent elasticity and resource abstraction can lead to performance differences from those observed in traditional HPC settings. For example, the virtualization layer in Cloud platforms can introduce additional latency, which may affect the efficiency of synchronous communication patterns more than asynchronous ones.

2.4 Gaps or Opportunites

While there is extensive literature on the performance of asynchronous actor and PGAS programming models in HPC environments, relatively few studies focus on their performance in Cloud computing contexts.

This study aims to fill this gap by providing a detailed comparative analysis of asynchronous actor-based programming (HClib-actor) and PGAS programming (OpenSHMEM) on the Azure Cloud platform. By focusing on graph microkernels, which are representative of a wide range of real-world applications, this research offers actionable insights for industry practitioners looking to optimize Cloud-based applications and academics interested in advancing the state of parallel computing research.

3 Methodology

3.1 Experimental Setup

Our experiments involve running the same set of kernels on a Cloud platform and an HPC cluster. We use the PACE [8] cluster for the HPC platform and Azure D-series virtual machines for the Cloud platform. Since we are focused on the latency overheads introduced by the network, we utilize only one core per machine on both platforms. PACE nodes use Intel Xeon Gold 6226 CPUs, and

Cloud nodes utilize Intel Xeon Platinum 8473C CPUs. PACE is equipped with InfiniBand networking; all nodes are in the same data center. The networking between Cloud nodes is far more heterogeneous and more liable to changing data center traffic. There is also most only a region-level locality guarantee. Finally, to keep findings general to most Cloud networks, we chose not to enable Azure-only networking features like single-root input/output virtualization (SR-IOV) or proximity placement. The latter does not offer locality guarantees beyond best effort, and in practice, the performance benefit observed was small enough that we chose to turn it off to make our results more generalizable. In the same spirit of targeting a "generic cloud" machine and network, we decided not to use Azure's "HPC-optimized" offering, which includes a setup very similar to an HPC Cluster, such as having Infiniband networking between nodes. The D-series VMs we picked for experiments use Ethernet.

3.2 Benchmark and Metric

In this study, we employ the Bale suite of graph microkernels to evaluate the performance of asynchronous actor-based programming and blocking PGAS models. The Bale suite includes seven kernels designed to capture a wide range of irregular access patterns common in many applications:

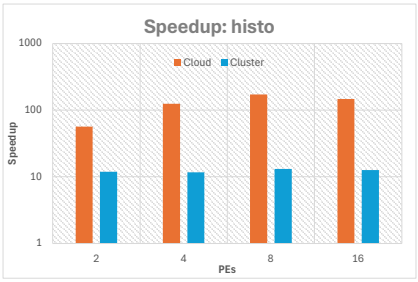
1. Histogram: Builds histogram from a set of randomly generated data, some remote and some local.
2. Index Gather: Gathers elements from a (remote) array based on a list of indices.
3. Permute: Permutes the elements of an array.
4. Randperm: Generates and stores a random permutation of array elements.
5. Transpose: Transposes a large, sparse matrix.
6. Triangle: Count the number of triangles (three-degree cycles) in a sparse graph
7. Toposort: Topological sort on a sparse DAG.

These benchmarks collectively cover a variety of computational tasks over sparse matrices. For each kernel, we study three PGAS versions - one that is implemented in OpenSHMEM (and uses synchronous communication) and two asynchronous versions - one implemented using the Conveyors message aggregation library [7]. The other uses the HClib-actor runtime [10, 9] built on top of the Conveyors library. The measured metric is execution time in seconds.

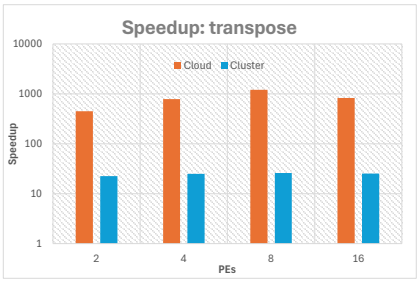
4 Experimental Observations

This section details our observations from running the Bale suite on both platforms. Aggregates of execution time for each kernel across three programming models and both platforms are provided in Table 2. Charts depicting this data are provided in Figure 2 and Figure 3 - since the difference in execution times

Fig. 1: Comparing relative Speedup of Selector over OpenSHMEM variant of two graph kernels on an HPC Cluster and the Cloud. Note that the vertical axis is logarithmic.



(a) Speedup on histogram kernel



(b) Speedup on matrix transpose kernel

between kernel versions is extreme on the Cloud, we have used a logarithmic axis (base-10) for the columns on the right, except for the very first figure where we use a linear scale to show the standard error better. Table 1 provides speedup data across the Cloud and Cluster platforms. Alternatively, it answers the question: **How many times faster is the selector version compared to the synchronous OpenSHMEM version?** Speedup comparisons across these two platforms are depicted in charts for two kernels in Figure 1.

From the observations, the easy-to-predict insight is that the HPC Cluster is faster than the Cloud due to the Cluster having a dedicated interconnect. However, this study also revealed several interesting performance facts, like the variation in execution times on the Cloud and the incredible improvements wrested by switching to asynchronous communication on such slower networks.

Broad advantages of networking in sparse graph operations As can be seen in table 2, in our experiments, all kernels perform much better on the HPC cluster, which is to be expected by virtue of the latter having an Infiniband 100HDR interconnect.

Highly Variable Cloud Execution times We observe that the workloads on the Cloud have extreme variation, especially when executing the slower kernels. For instance, the OpenSHMEM histogram kernel had a 68-second difference between the slowest and fastest observation on the Cloud. However, on the Cluster, we observe that the variation between the slowest and fastest histogram kernel was in the order of milliseconds. We theorize this is due to the non-deterministic nature of data-center traffic since we use shared resources. We control for this variation by repeating each experiment 5 times with significant delay between runs. Each value in the table 2 represents the average execution time of 5 runs while the error bars in the left columns of Figures 2 and 3 and in the first row of the right column in Figure 3 represent the variation in times.

Orders-of-magnitude performance boost with asynchronous actors The results reveal a stark benefit of using asynchronous approaches like Conveyors and selectors on the Cloud; on the HPC cluster, the selector version is at most one order of magnitude (10x-30x) slower, but this difference becomes two to three orders of magnitude (1000x) when we move to the Cloud. We highlight the speedup in Figure 1 and Table 1.

Table 1: Speedup gained from asynchronous actors over Bale graph kernels. Speedup on the HPC cluster is displayed in the shaded (blue) columns, and Speedup on the Cloud is displayed in unshaded (white) columns. Speedup is computed as the quotient resulting from the selector version running time divided by the OpenSHMEM version running time from Table 2.

Kernel	**2**	**2**	**4**	**4**	**8**	**8**	**16**	**16**
histogram	11.9	56.5	11.6	124.2	13.1	171.7	12.6	146.2
index-gather	14	196.7	20.2	387.3	23.1	442.1	23.9	379.1
permute	13.8	249.8	20.1	582.5	22.9	704.6	23	690.9
randperm	9.8	155.8	11.3	335.7	11.7	397.9	11.5	387.6
topological sort	39	555.6	46.8	905.9	50.5	921.6	48.2	380.9
matrix transpose	22.7	451.2	25	792.7	26.1	1206.9	25.5	833.9
triangle counting	15.7	356	22.6	725.3	26	914.5	27.1	876.5

5 Conclusion

Our study reveals significant performance disparities between asynchronous actor programming and synchronous PGAS strategies in Cloud versus HPC cluster environments. Asynchronous approaches show dramatically amplified benefits in Cloud settings, with performance gaps widening from one order of magnitude in HPC clusters to two or three orders in the Cloud. We also observe high execution time variability in Cloud environments, underscoring the challenges of shared resources and fluctuating data center traffic. These findings emphasize the critical role of network infrastructure in sparse graph operations and suggest that asynchronous programming models should be preferred for Cloud-based HPC applications, especially for workloads with irregular access patterns.

5.1 Opportunities and Future Work

Our work does not perform cost modeling or estimation but focuses purely on execution time. Future work could augment this study with the dollar values of running these kernels on each test harness. This study also limits itself to the most generally available type of virtual machine on the Cloud, and future work could study how specific configurations (usually restricted to a particular vendor) could offer performance improvements. Future work can also focus on developing

Table 2: Bale suite running times on an HPC cluster and a Cloud environment. All running times are in seconds. Cluster running times are in shaded (blue) columns, and Cloud running times are in unshaded (white) columns. All values are computed as averages of five executions.

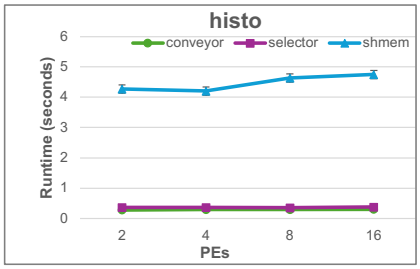
Average running time per kernel		**Number of PEs**							
Kernel	**Time**	**2**	**2**	**4**	**4**	**8**	**8**	**16**	**16**
histogram	conveyor	0.2832	2.8876	0.2926	1.7036	0.2934	2.023	0.3032	6.3998
	selector	0.3608	2.7106	0.3644	1.9702	0.3548	1.7934	0.3798	5.0764
	shmem	4.2676	153.053	4.202	244.629	4.6294	262.0524	4.7478	608.085
index-gather	conveyor	0.6378	9.0226	0.6548	6.9162	0.662	6.0698	0.6724	6.7602
	selector	0.8116	8.633	0.815	6.7688	0.82	6.2924	0.85	7.5202
	shmem	11.3134	1697.404	16.3852	2620.942	18.9024	2781.248	20.2506	2850.5
permute	conveyor	0.285	2.4858	0.2922	2.5552	0.2892	1.9004	0.3132	2.3514
	selector	0.3304	2.6586	0.3312	1.8768	0.3376	1.695	0.3606	2.029
	shmem	4.5272	664.108	6.6486	1093.146	7.6992	1194.291	8.2578	1401.715
randperm	conveyor	0.2338	1.3418	0.2396	1.0888	0.2416	1.1378	0.2558	1.4236
	selector	0.265	1.6654	0.2682	1.2648	0.273	1.3332	0.2872	1.51
	shmem	2.594	259.461	3.007	424.518	3.1876	530.468	3.2786	585.274
topological sort	conveyor	0.1088	1.2472	0.1112	1.189	0.1128	1.344	0.1162	2.3386
	selector	0.1464	1.2074	0.1492	1.1246	0.1518	1.2526	0.1676	3.172
	shmem	5.7026	670.719	6.979	1018.718	7.6578	1154.388	8.0726	1207.932
matrix transpose	conveyor	0.2568	1.4614	0.2584	1.0742	0.2606	1.1566	0.2758	1.6644
	selector	0.2824	1.422	0.2874	1.2322	0.2918	0.934	0.3072	1.578
	shmem	6.3982	641.528	7.1758	976.745	7.588	1127.212	7.8224	1315.747
triangle counting	conveyor	0.5196	3.8704	0.5206	3.3188	0.5176	2.8854	0.5398	2.9268
	selector	0.6004	4.1872	0.604	3.2088	0.6088	2.59075	0.6266	3.0792
	shmem	9.3682	1490.288	13.6376	2327.121	15.7818	2369.132	16.955	2698.743

adaptive runtime systems to handle the unpredictable nature of shared Cloud resources and further optimize asynchronous programming models for Cloud environments.

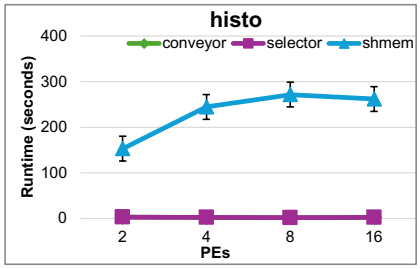
References

1. Callahan, D., Chamberlain, B.L., Zima, H.P.: The cascade high productivity language. Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, 2004. Proceedings. pp. 52–60 (2004), https://api.semanticscholar.org/CorpusID:5217126
2. Carlson, W.W., Draper, J.M., Culler, D.E., Yelick, K.A., Brooks, E.D., Warren, K.H.: Introduction to upc and language specification (2000), https://api.semanticscholar.org/CorpusID:59868665
3. Chapman, B., Curtis, T., Pophale, S., Poole, S., Kuehn, J., Koelbel, C., Smith, L.: Introducing openshmem: Shmem for the pgas community. In: Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model. PGAS '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/2020373.2020375, https://doi.org/10.1145/2020373.2020375
4. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and

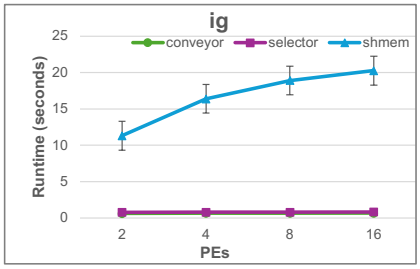
Fig. 2: Bale performance on the two test harnesses - HPC cluster (left column) and Cloud (right column). Note the vertical axes (i.e., execution times) use a linear scale in the figures in the left column and a logarithmic scale in the figures (barring the first row) in the right column, respectively.



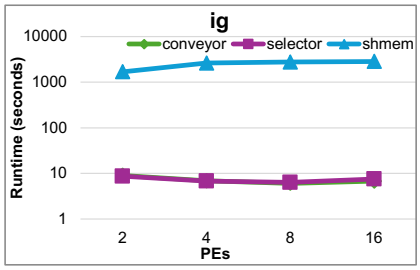
(a) Histogram on HPC Cluster



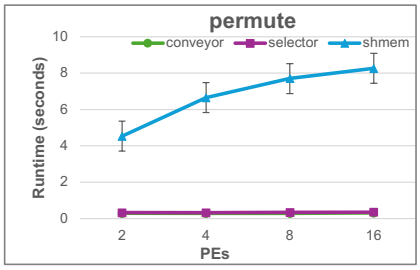
(b) Histogram on Cloud



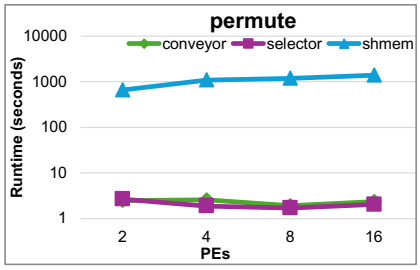
(c) Index-gather on HPC Cluster



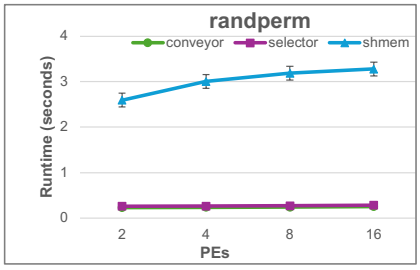
(d) Index-gather on Cloud



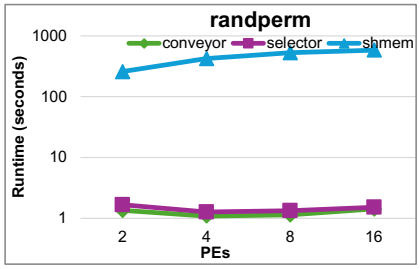
(e) Matrix permute on HPC Cluster



(f) Matrix permute on Cloud

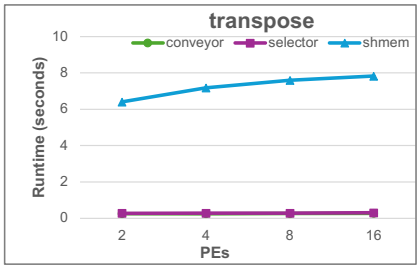


(g) Random Permutation on HPC Cluster

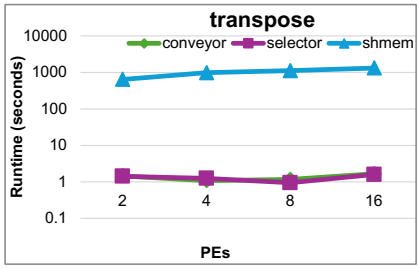


(h) Random Permutation on Cloud

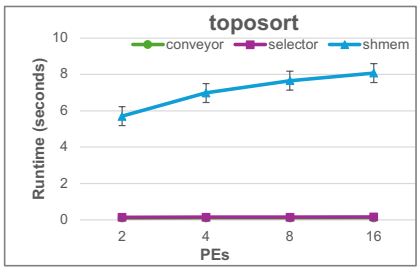
Fig. 3: (Fig 2 continued ....) Note the vertical axes (i.e., execution times) use a linear scale in the figures in the left column and a logarithmic scale in the figures in the right column, respectively.



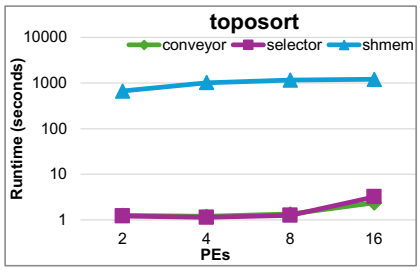
(a) Matrix Transpose on HPC Cluster



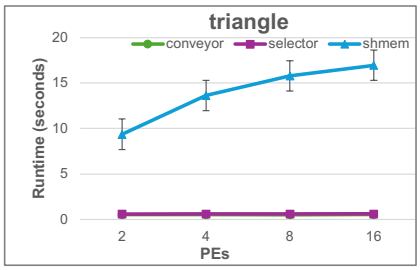
(b) Matrix Transpose on Cloud



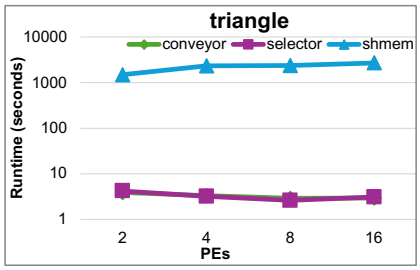
(c) Topological Sort on HPC Cluster



(d) Topological Sort on Cloud



(e) Triangle Counting on HPC Cluster



(f) Triangle Counting on Cloud

Applications. p. 519–538. OOPSLA '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1094811.1094852, https://doi.org/10.1145/1094811.1094852
5. Hewitt, C.E., Bishop, P.B., Steiger, R.: A universal modular actor formalism for artificial intelligence. In: International Joint Conference on Artificial Intelligence (1973), https://api.semanticscholar.org/CorpusID:18601146
6. Imam, S.M., Sarkar, V.: Selectors: Actors with multiple guarded mailboxes. In: Proceedings of the 4th International Workshop on Programming Based on Actors Agents & Decentralized Control. p. 1–14. AGERE! '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2687357.2687360, https://doi.org/10.1145/2687357.2687360
7. Maley, F.M., DeVinney, J.G.: Conveyors for streaming many-to-many communication. In: 2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3). pp. 1–8. IEEE (2019)
8. PACE: Partnership for an Advanced Computing Environment (PACE) (2017), http://www.pace.gatech.edu
9. Paul, S.R., Hayashi, A., Chen, K., Elmougy, Y., Sarkar, V.: A fine-grained asynchronous bulk synchronous parallelism model for pgas applications. Journal of Computational Science **69**, 102014 (2023). https://doi.org/https://doi.org/10.1016/j.jocs.2023.102014, https://www.sciencedirect.com/science/article/pii/S1877750323000741
10. Paul, S.R., Hayashi, A., Chen, K., Sarkar, V.: A productive and scalable actor-based programming system for PGAS applications **13350**, 233–247 (2022). https://doi.org/10.1007/978-3-031-08751-6_17
11. Yelick, K., Bonachea, D., Chen, W.Y., Colella, P., Datta, K., Duell, J., Graham, S.L., Hargrove, P., Hilfinger, P., Husbands, P., Iancu, C., Kamil, A., Nishtala, R., Su, J., Welcome, M., Wen, T.: Productivity and performance using partitioned global address space languages. In: Proceedings of the 2007 International Workshop on Parallel Symbolic Computation. p. 24–32. PASCO '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1278177.1278183, https://doi.org/10.1145/1278177.1278183