

# Point to Point Communication and Collective MPI Performance Test

Yitao Shen  
Rensselaer Polytechnic Institute  
Troy, NY, 12180  
larst@affiliation.org

Jinqiang Jiang  
Rensselaer Polytechnic Institute  
Troy, NY, 12180  
larst@affiliation.org

Chau-Lin Huang  
Rensselaer Polytechnic Institute  
Troy, NY, 12180  
huangc11@rpi.edu

Lorson Blair  
Rensselaer Polytechnic Institute  
Troy, NY, 12180  
larst@affiliation.org

Christopher D. Carothers  
Rensselaer Polytechnic Institute  
Troy, NY, 12180  
chrisc@cs.rpi.edu

## ABSTRACT

Communication among multiple nodes plays a critical role in the performance of High Performance Computing. MPI [2] offers a great number of libraries to maximize and test the communication performance in the parallel computing networks. The message passing has been observed to spend additional time in transferring information from one node to another, and several parallel models have been devised to properly describe the phenomenon, which in terms serve as criteria for future benchmarking. In this work, we build a collective MPI performance test to evaluate the performance among different models. To accomplish this, we have a parallel version of Game of Life program optimized with MPI communication scheme and CUDA for GPU parallelization. As far as the authors are concerned, this work could be the first intend to verify the networking performance of a GPU-aided program in terms of different parallel models. In terms of hardware, the benchmarking takes place on AiMOS [4], an eight petaflop supercomputer using a heterogeneous system architecture built with IBM POWER9 CPUs and NVIDIA GPUs. On the software side, the program relies on IBM Spectrum MPI and NVidia CUDA math library [17].

## KEYWORDS

Parallel Computing, High Performance Computing, CUDA, MPI

### ACM Reference Format:

Yitao Shen, Jinqiang Jiang, Chau-Lin Huang, Lorson Blair, and Christopher D. Carothers. 2020. Point to Point Communication and Collective MPI Performance Test. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*.

Unpublished working draft. Not for distribution.

Permission is made digital or hard copies for personal or professional use, not for redistribution, profit, or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Woodstock '18, June 03–05, 2018, Woodstock, NY

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/10.1145/1122445.1122456>

2020-05-05 21:49. Page 1 of 1–7.

ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/1122445.1122456>

## 1 INTRODUCTION

In order to make modern parallel computing software efficient and powerful, we need to understand what are the underlying caveats that may slow the process down. There are various models proposed with the endeavour to effectively describe the parallel system and yet simple enough to capture the essential factors that affect the performance of the system in substantial degree. Each of the models is no better than the others but only is deemed most useful under suitable contexts. Through the performance analysis with the aid of these mathematical models, we can further find the bottlenecks that could be mitigated to optimize the communication performance of the parallel system.

In this article, we will discuss the main parallel performance models, and through the benchmarking of the parallel version of Game of Life program, we can further utilize these models to evaluate the point-to-point and collective communication performances in the fashion of weak or strong scaling.

As per the summary, in section 2 we will discuss previous studies related to the topic of our discussion in this article. Section 3 we will give a brief introduction about Conway's Game of Life algorithm and how we modified it into its MPI-enabled parallel version. In section 4 and 5 we will present the hardware and frameworks we utilize to perform the benchmarking. More importantly, in section 6 we will discuss the main performance models to describe parallel systems. The course of our discussion will culminate in the introduction of the performance metrics in section 7 and the experiment results in section 8. Last but not least, we will finalize the article with a brief summary and propose prospective future works in section 9.

## 2 RELATED WORKS

There are several works discussing the benchmarking of MPI performance on parallel systems using different libraries. Pješivac-Grbović et al. [16] give a thorough overview of the parallel models, and compare the performances on inter-cluster MPI collective operations on two systems. The authors in the mentioned article also demonstrate that the gap

between the message sendings depends on the number of unique destination nodes.

There are also works whose authors discuss the communication performance on the parallel version of Conway's Game of Life. L. Ma et al. [13] demonstrate the performance boosting by implementing the algorithm into its parallel counterpart. The article's author implemented the parallel program with OpenMP library, which takes advantage of the multithreading in the CPU rather than seeking time efficiency improvement from multi-node perspective.

In our work we referenced an open-source framework for implementing network benchmarks presented by T. Hoefer et al. [9] This framework separates communication patterns from communication modules which allows independently added benchmark types and network protocols. The authors also presented a LogGPS pattern [8] which supports measurement of LogP and LogGP models parameters such as latency, overhead and gap per bytes over MPI.

### 3 GAME OF LIFE AS AN ALGORITHM

The Game of Life as invented by the British mathematician John Horton Conway in 1970 [6] is a cellular automaton. The algorithm is a zero-player game and as the game evolves throughout undetermined number of iterations, the outcome is determined by the given initial configuration. The game is taken place on a two-dimensional orthogonal grid of square cells. The cell status is atomic, that is it can only be found as alive or dead. Each cell's status is determined by eight adjacent neighboring cells. At each step in time, any cell with fewer than two live neighbors dies due to under-population. On the contrary, if the cell lives with two or three live neighbors survives to the next step. If there are more than three immediately adjacent cells, the cell perishes due to over-population. Lastly, the cell resuscitates with exactly three live neighbors and can be seen as the result of cellular reproduction.

The operations involved in the Game of Life include instantiation of the world configuration, the update of the cells in the world, and swapping the newly updated world with the previous one. The operations not only is possible to implement serially, but also with parallel speed-up mechanism to take advantage of the efficient memory manipulation offered by NVIDIA CUDA math library, or through the de-facto networking of various nodes through MPI libraries when the dimension of the world increases toward dimension of high orders of magnitude. For the message passing optimization, we have to take care of the "ghost" rows at the MPI rank boundaries apart from allocating suitable memory chunk for the rank itself. To achieve such requirement, special care needs to be taken at sending and receiving messages at the boundaries where the current rank's top ghost row meets with the previous rank's last row, and where the bottom ghost row meets the first row of the next rank. In addition, the CUDA kernel also needs to undergo modifications to account for the lack of top and bottom edge world wrapping. Therefore, at the initialization of the world configuration, the placement

of elements at the edge of the world should be placed in determined ranks. Once the instantiation of the initial world configuration takes place, the MPI-and-CUDA-enabled Game of Life program opens way for further benchmarking.

### 4 GPU AND CUDA TOOLKIT

According to [17], AiMOS uses NVIDIA Tesla V100 GPUs in conjunction with the compute nodes. The NVIDIA Tesla V100 accelerator contains the Volta GV100 GPU. Volta is the codename for NVIDIA's GPU microarchitecture release on December 7, 2017. Volta was NVIDIA's first chip to feature Tensor Cores, designed specially to yield higher deep learning performance than regular CUDA cores. [15]. The architecture is implemented with TSMC's 12 nm FinFET process.

Tesla V100 delivers 7.8 TFLOPS of double precision floating point (FP64) performance, 15.7 TFLOPS of single precision (FP32) performance, and 125 Tensor TFLOPS based on GPU Boost clock.

In AiMOS cluster, there are 16 nodes each containing four NVIDIA Tesla V100 GPUs with 16 GiB of memory each. In addition, there are 252 nodes each possessing six of the same accelerators with 32 GiB of memory each [5].

The CUDA Toolkit version used in AiMOS is CUDA 9.1 and 10.0 [5].

### 5 MPI

AiMOS adopts MPI Spectrum as its message passing interface library. The MPI Spectrum is developed by IBM as a high-performance, production quality implementation of MPI to accelerate end-to-end communication. MPI Spectrum is based on the open-source MPI library, but is integrated with improved RDMA networking and supports NVIDIA GPUs based on IBM PAMI backend. Another feature is that MPI Spectrum enhances collective library running blocking and non-blocking algorithms [14].

### 6 MODELS TO DESCRIBE THE PARALLEL SYSTEMS

To verify the communication performance of the GPU-accelerated MPI-aided Game of Life program, different configurations of nodes are set. The collective MPI performance is expressed in terms of the four common networking performance models: Hockney [7], LogP [3], LogGP [1], and PLogP [10]. The parallel models can be seen as a sequence of proposals toward establishing the proper description for both point-to-point and collective communication time consumption under any parallel computing system.

The Hockney model is considered the simplest parallel model of communication. The model assumes that the time taken to send a message is

$$T = \alpha + \beta m$$

where  $\alpha$  is the size of the message, and  $\beta$  is the inverse of the bandwidth, while  $m$  represents the message size. The model is suitable to describe point-to-point communication systems.

The LogP model intends to offer a simple yet more detail view to facilitate the finding of bottlenecks in possible communication latency. The model is described with four parameters: the latency  $L$ , overhead  $o$ , gap between the sending of messages  $g$ , and the number of processors or nodes involved in the communication  $P$ . The model assumes that only small amount of messages is transferred simultaneously. The time needed to transfer messages between nodes takes

$$T = L + 2o$$

where  $L$  is the latency, and  $o$  as the overhead.

Since LogP does not monitor transmission of long messages, LogGP further extends such aspect in its description. A new parameter Gap per byte ( $G$ ) is taken into account, which is defined as the time per byte for a long message [1]. Unlike the LogP model which restricts to constant small size messages, LogGP allows sending larger messages. Typically, time taken to transfer a message is:

$$T = L + 2o + (m - 1)G$$

where  $G$  is the gap per byte and  $m$  is the size of the message.

In the work of T. Kielmann et al. [10], parametrized LogP is introduced as a slight extension of LogP and LogGP models that is capable of monitoring the minimal gap between two messages without saturating the network for each message size. In addition to the parameters contained in LogP, additional parameters are included: the sender and receiver overheads, message transfer time and data copying time to and from the network interfaces are included in the latency. Moreover, the gap parameter is defined as the minimum time interval between consecutive message transmission or reception. The overall time spent in the message transferring can be expressed as:

$$T = L + g(m)$$

where  $g(m)$  is the gap per message. The worth pointing out that the sender  $o_s(m)$  and receiver  $o_r(m)$  overheads depend on the message size.

## 7 PERFORMAMNCE METRICS

## 8 PERFORMANCE RESULTS

## 9 CONCLUSION AND FUTURE WORKS

In this work we perform benchmarking on four different parallel models to evaluate the performance of the CUDA-aided Game of Life program. The GPU-accelerated program is integrated with MPI's communication mechanism to effectively manage the memory access as well as transferring among different nodes.

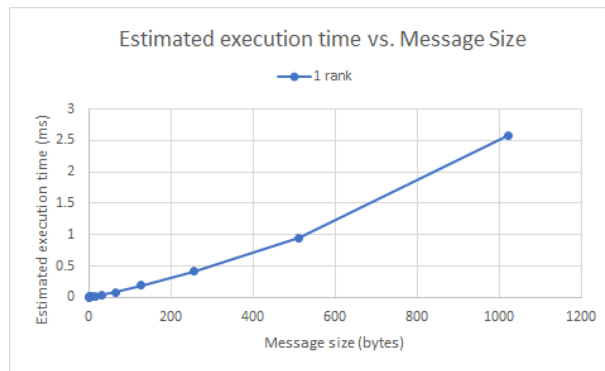
For future works, there are two possible extensions on the program to study how different mechanism may further improve or deteriorate the performance of the current version of Game of Life program. The first possible direction is to incorporate multithreading to the current version to observe how sharing resources on the same node may affect the message communication, and another possible study involves the

execution of the same GPU-accelerated program on different topologies such as the Fat Tree [12], Dragonfly [11] or Slimfly [18] networks to investigate the difference in performance described with the mentioned parallel models.

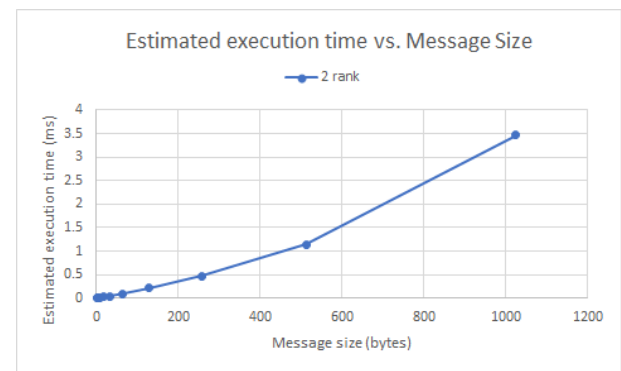
## 10 ACKNOWLEDGMENTS

## REFERENCES

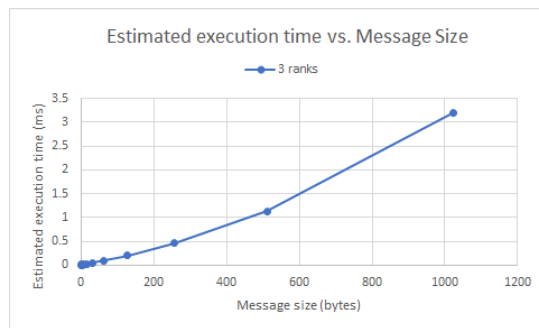
- [1] Albert Alexandrov, Mihai F Ionescu, Klaus E Schauser, and Chris Scheiman. 1995. LogGP: incorporating long messages into the LogP model one step closer towards a realistic model for parallel computation. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*. 95–105.
- [2] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. 1994. The MPI message passing interface standard. In *Programming environments for massively parallel distributed systems*. Springer, 213–218.
- [3] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. 1993. LogP: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*. 1–12.
- [4] Center for Computational Innovations. [n.d.]. Center for Computational Innovations. <https://cci.rpi.edu/aimos>
- [5] Center for Computational Innovations. [n.d.]. DCS Supercomputer. [https://secure.cci.rpi.edu/wiki/index.php?title=DCS\\_Supercomputer#Using\\_GPUs](https://secure.cci.rpi.edu/wiki/index.php?title=DCS_Supercomputer#Using_GPUs)
- [6] Martin Gardner. 1970. MATHEMATICAL GAMES: The fantastic combinations of John Conway's new solitaire game" life,". *Scientific American* 223 (1970), 120–123.
- [7] Roger W Hockney. 1994. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel computing* 20, 3 (1994), 389–398.
- [8] Torsten Hoeftler, Andre Lichei, and Wolfgang Rehm. 2007. Low-overhead LogGP parameter assessment for modern interconnection networks. In *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 1–8.
- [9] Torsten Hoeftler, Torsten Mehlan, Andrew Lumsdaine, and Wolfgang Rehm. 2007. Netgauge: A Network Performance Measurement Framework. *High Performance Computing and Communications Lecture Notes in Computer Science, HPCCO7, presented in Houston, USA 4782* (2007), 659–671.
- [10] Thilo Kielmann, Henri E Bal, and Kees Verstoep. 2000. Fast measurement of LogP parameters for message passing platforms. In *International Parallel and Distributed Processing Symposium*. Springer, 1176–1183.
- [11] John Kim, William Dally, Steve Scott, and Dennis Abts. 2009. Cost-efficient dragonfly topology for large-scale systems. *IEEE micro* 29, 1 (2009), 33–40.
- [12] Charles E Leiserson. 1985. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers* 100, 10 (1985), 892–901.
- [13] Longfei Ma, Xue Chen, and Zhouxiang Meng. 2012. A performance Analysis of the Game of Life based on parallel algorithm. *arXiv preprint arXiv:1209.4408* (2012).
- [14] NVIDIA. [n.d.]. IBM Spectrum MPI. <https://developer.nvidia.com/ibm-spectrum-mpi>
- [15] NVIDIA. [n.d.]. NVIDIA Volta - The Tensor Core GPU Architecture designed to Bring AI to Every Industry. <https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/>
- [16] Jelena Pješivac-Grbović, Thara Angskun, George Bosilca, Graham E Fagg, Edgar Gabriel, and Jack J Dongarra. 2007. Performance analysis of MPI collective operations. *Cluster Computing* 10, 2 (2007), 127–143.
- [17] top500.org. [n.d.]. AiMOS - IBM Power System AC922, IBM POWER9 20C 3.45GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Volta GV100. <https://www.top500.org/system/179781>
- [18] Noah Wolfe, Christopher D Carothers, Misbah Mubarak, Robert Ross, and Philip Carns. 2016. Modeling a million-node slim fly network using parallel discrete-event simulation. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 189–199.



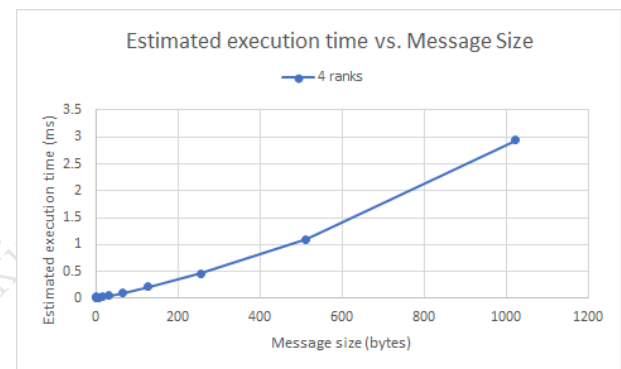
(a) Rank 1



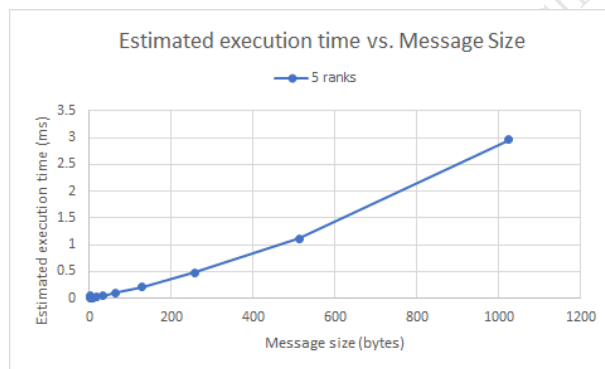
(b) Rank 2



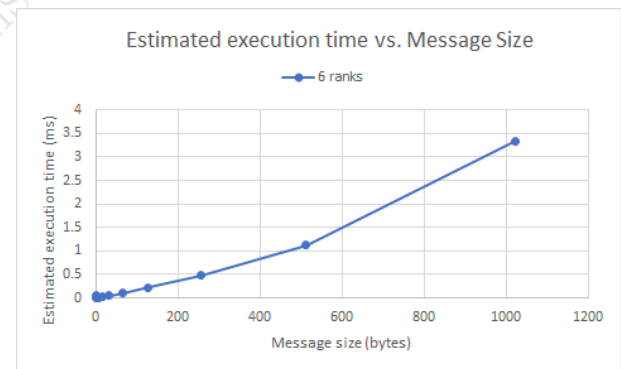
(c) Rank 3



(d) Rank 4



(e) Rank 5



(f) Rank 6

Figure 1: Estimated execution time vs message size along different ranks.

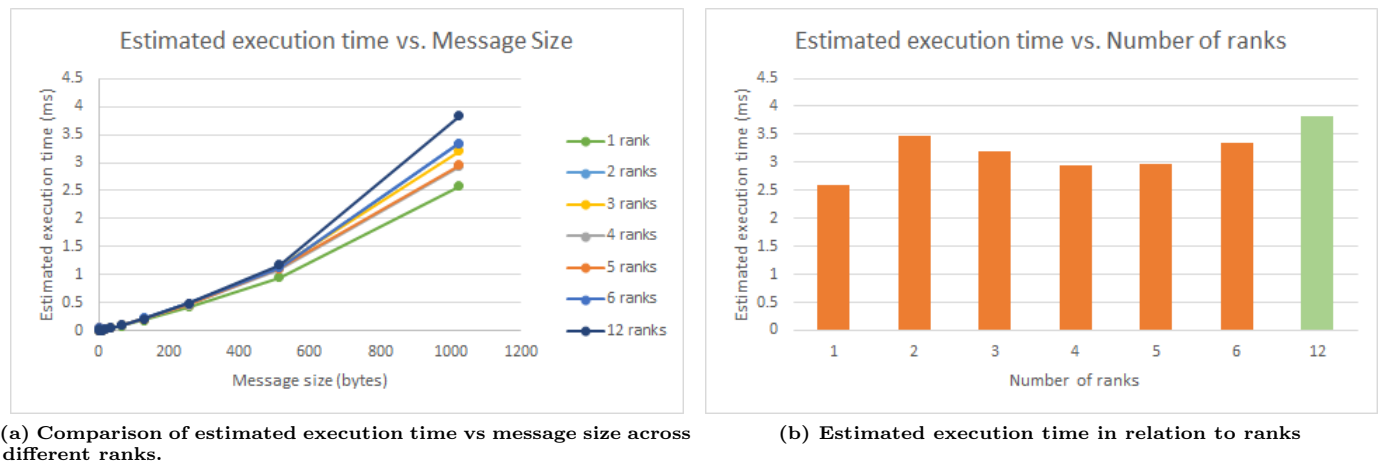


Figure 2: The relationship between message sizes and number of ranks to estimated execution time.

581	639
582	640
583	641
584	642
585	643
586	644
587	645
588	646
589	647
590	648
591	649
592	650
593	651
594	652
595	653
596	654
597	655
598	656
599	657
600	658
601	659
602	660
603	661
604	662
605	663
606	664
607	665
608	666
609	667
610	668
611	669
612	670
613	671
614	672
615	673
616	674
617	675
618	676
619	677
620	678
621	679
622	680
623	681
624	682
625	683
626	684
627	685
628	686
629	687
630	688
631	689
632	690
633	691
634	692
635	693
636	694
637	695
638	696

697	755
698	756
699	757
700	758
701	759
702	760
703	761
704	762
705	763
706	764
707	765
708	766
709	767
710	768
711	769
712	770
713	771
714	772
715	773
716	774
717	775
718	776
719	777
720	778
721	779
722	780
723	781
724	782
725	783
726	784
727	785
728	786
729	787
730	788
731	789
732	790
733	791
734	792
735	793
736	794
737	795
738	796
739	797
740	798
741	799
742	800
743	801
744	802
745	803
746	804
747	805
748	806
749	807
750	808
751	809
752	810
753	811
754	812