High performance computing vs. Cloud computing: Using a parallel Matrix Multiplication algorithm

Christopher Eichstedt Nicholas Jordy
Sergiu M. Dascalu Frederick C. Harris Jr.
Department of Computer Science and Engineering
College of Engineering, University of Nevada, Reno
Reno, Nevada
chris.eichstedt@nevada.unr.edu, orion996@gmail.com,
dascalus@cse.unr.edu, fred.harris@cse.unr.edu

***Abstract*—Using a parallel Matrix Multiplication algorithm we developed, deployed, tested and compared the results of high performance computing and cloud computing platforms. The test criteria for each was performance and cost effectiveness. To parallelize our serial algorithm, we utilized Berkley's Unified Parallel C++ library and converted our local data into a global values that are a part of a Partitioned Global Address Space. The HPC machine Bridges was already equipped with the necessary libraries and utilities for our project, but we were required to build a virtual machine on Azure from the ground up. This meant that we needed to first deploy a build model, install all necessary dependencies and then scale performance until it reached comparable levels to Bridges. This posed a challenge as resources for both platforms were limited, due to the ongoing pandemic of COVID-19. When we examined our results, we found that smaller jobs worked more efficiently when using Azure, but for more computationally intensive jobs, we found that Bridges was better suited as it had more power per node than anything we were able to implement using the restricted builds available on Azure. If the project were to continue forward, we would like access to more resources and a budget, as to test the limits of both platforms when pushed further.**

***Index Terms*—HPC, Cloud computing, Azure, Bridges, PSC, UPC++, VM, PGAS, PaaS, vCPUs**

I. Introduction

This purpose of this paper is to outline, test, and record findings regarding the performance and cost efficiencies of running a parallel algorithm on both, a high performance computing (HPC) machine and a cloud computing service. Through independent studies at the University of Nevada, Reno, we began studying Microsoft's cloud computing service, Azure. During our research, we began to understand the flexibility of different levels of computational power that it provides: from everyday machines to HPC machines. Also, we gained access to an HPC machine and began running parallel experiments using multiple threads and nodes. The machine we gained access to was Bridges, provided by the Pittsburgh Super Computing Center (PSC).

During our research, we found there to be a wide array of information and previous discoveries. The first, being scalability and performance, is the major contrast of both platforms. HPC utilizes performance, whereas cloud computing utilizes scalability. Randy Bias of Cloudscaling.com defines both as, "... performance measures the capability of a single part of a large system while scalability measures the ability of a large system to grow to meet growing demand. Scalable systems may have individual parts that are relatively low performing" [1]. Another comparison was the cost differences of each platform. This paper's experiments will utilize a small amount of resources, and so these costs won't apply in full to the project. However, If the problem being computed were to be scaled up, the costs change drastically. It then becomes an issue of whether or not the extra compute power is financially worth it. Chris Downing of HPCwire.com says "Cloud prices are targeted at enterprise customers, where hardware utilisation below 20 percent is common. Active HPC sites tend to be in the 70-90 percent utilisation range, making on-demand cloud server pricing decidedly unattractive" [2].

It is the goal of this paper to clarify which platform is more viable via cost or performance, when utilizing them for computationally intense jobs. For the purpose of these experiments, we have developed a parallel matrix multiplication algorithm, using Berkley's Unified Parallel C++ library (UPC++), that will be deployed on both the Bridges supercomputer and a custom built virtual machine (VM) on Azure.

The rest of this paper is structured as follows: Section II covers the background of the research done, as well as the literature necessary to understand the project. Section III covers the details regarding the approach to the problem and what implementations were done to achieve that approach. Section IV will cover the results gained from testing on both platforms: Azure and Bridges. Section V will conclude the final thoughts determined by the results. Section VI will outline what would be done, if the project were to continue.

II. Background

This section is meant to give clarity regarding the various aspects and their backgrounds used for the project. These topics are what you should know as to better understand the workings of the experiments and what they involved.

A. Azure

Azure allows us to customize the performance of virtual machines with different types of processors and memory as

well as allowing the user to utilize scaling operations. There are two different types of scaling operations used on Azure: Scaling Up and Scaling Out [3]. The term Scaling Up refers to increasing the power of a single VM, this can be done by changing the amount and type of virtual processors, also the size and type of memory of a single VM. This allows a user to tailor a VM to the specifications that they need for a given job. Azure allows a user to scale up in real time, permitting for flexibility at run time. The other operation, Scaling Out, refers to creating multiple VMs that run jobs in parallel with each other, increasing the performance through the combined efforts of each VM. This allows a user to create VMs with the same specifications while distributing the workload amongst them. For the purposes of our experiment, we will detail the chosen specifications for this project, in Section III.

B. Bridges

Our access to Bridges allowed us to use the RM partition, which contained 752 nodes. However, due to our agreement with the University, we were only allotted the use of a maximum of 64. Although further restrictions were given due to current events, they will be discussed in Section II-D. Each node on the Bridges RM partition utilizes two Intel Xeon Haswell 2.3 GHz, 14 core processors, 128 GB of DDR4-2133 RAM, and 2 4TB Hard Disk Drives [4]. Utilizing multiple nodes using parallelism, allows for complex algorithms to be executed at a rapid pace. The Bridges HPC machine more than met our needs regarding computational intensity for the project. Nodes on the RM partition have three main methods of communicating with each other: OpenMP, MPI, and UPC++.

C. UPC++

As defined by the official repository for Berkely's UPC++ library, "UPC++ is a C++ library that supports Partitioned Global Address Space (PGAS) programming, and is designed to interoperate smoothly and efficiently with MPI, OpenMP, CUDA and AMTs. It leverages GASNet-EX to deliver low-overhead, fine-grained communication, including Remote Memory Access (RMA) and Remote Procedure Call (RPC)" [5]. For the purposes of this project, we decided to use UPC++ based on the ease of use, control over the parallelism, and the inclusion of the aforementioned other methods.

D. Limitations

During the time the project was developed, an outbreak of a new strain of Coronavirus, called COVID-19, impacted the world with closures of major gatherings of more than ten people. This includes all non-essential businesses, and educational spaces such as the University of Nevada, Reno. This also affected our access to both platforms significantly. Bridges began to use upwards of 75 percent of its resources for COVID-19 and pandemic modeling, cutting the amount that other users like us were granted [6]. This caused mass delays when trying to test our implementation on Bridges. As for Azure, Microsoft diminished their workforce dedicated to the Azure server facilities causing usages spikes at the ones kept running. Upwards of 755 percent usage increases were reported during the pandemic [7]. Also, Azure made it so that virtual machine creation was limited to one per non-priority user [8]. This made it impossible to test scaling out and also limited us on what we could do with scaling up. This will be further detailed in Section III. Another limitation for scaling up comes from the budget we had while implementing the project and the cost to make a comparable VM. This will be further discussed in Section IV.

III. Approach and Implementation

This section of the paper gives an overview of how we implemented similar design across two different platforms. Because Azure needed to be built from the ground up, using what resources we could manage given the current limitations, we decided to stride to rebuild the Bridges HPC machine using a scaled up VM. This unfortunately did not come to fruition as we had hoped. That said, below are the specifications and implementations used for the project, as well as notes regarding issues that arose.

A. Matrix Multiplication

The benchmark algorithm that we used to test both platforms was a matrix multiplication. This algorithm was simple to implement, and gets more complex as it scales making it perfect for this project. Figure 1 is the naive implementation for Matrix Multiplication. It has a compute intensity of n to the power of three, making it scale up greatly as we increase the size of n.

```
//Matrix Multiply for a n by n matrix
For i=0 to n
        For j=0 to n
                For k=0 to n
                        cij = cij + (aik *bkj)
```

Fig. 1. The Pseudo Code for a naive matrix multiplication algorithm

B. UPC++

To parallelize this algorithm, we used UPC++. As mentioned before, UPC++ uses a partitioned global address space (PGAS). This allowed us to have each individual node access the same data and perform computations simultaneously on that data. This cuts down on redundancy that would normally occur through message passing, or other primitive forms of distributed memory communication. A node can edit data in the PGAS, and then another will be able to interact with the same data. To do this, UPC++ makes use of global pointers, which can be accessed by all nodes.

In the naive algorithm, A, B, and C are represented as C++ arrays. UPC++ can create global arrays, that function similarly to the prior, allowing us to access the matrices globally on each processor. For example, accessing global arrays works in the same way as accessing a local array. However, you cannot use bracket logic, e.g. A[3]. Instead, you have to use pointer arithmetic, e.g. A+3. Also, when you write to, and read from a global array, you have to use the UPC++ command rput() and

rget() to do so. Figure 2 illustrates the conversion from reading and writing from a C++ array to a UPC++ global array.

```
A[3] = 21 -> rput(21, global_A+3);

printf(%d, B[4]) -> printf(%d, (rget(global_B+4)));
```

Fig. 2. The conversions from local arrays to global arrays

C. Parallel Matrix Multiplication

After converting to global data arrays, we now had to divide the work evenly amongst all nodes. To do this, we had each node calculate its section of the array based upon its rank. Each rank had a local value for how many elements of the matrix C, it was responsible for calculating. It was determined by the equation:

$$elementsPerRank = (matrixSize^2)/numRanks$$

Using this equation we can determine each nodes' section of the array, finding the start iterator determined by the equation:

$$startItr = elementsPerRank * currentRank$$

and the end iterator determined by the equation:

$$endItr = elementsPerRank * (currentRank + 1)$$

However, if elementsPerRank was greater than the size of the array, then it would default to being equal to the end of the array. After these iterators were created, the i and j loops of the parallel algorithm needed to be changed so that each node would only work on the section it was supposed to. Figure 3 shows the new pseudo-code for the parallelized algorithm.

```
//Parallel Matrix Multiply for a n by n matrix
For i=start_itr to end_itr
        For j=start_itr to end_itr
                For k=0 to n
                        cij = cij + (aik *bkj)
```

Fig. 3. The Pseudo Code for a parallel matrix multiplication algorithm

All that has to be done after the algorithm is call an instance of "upcxx::barrier()," which will tell the program to wait for all nodes to be done with their work up to that point. Then because each node accesses the same version of the matrix C, no extra communication needs to be done. Every node now has the result of A * B.

D. Scaling up vs. Scaling out

1) Scaling Up: During the project we scaled up a VM to be able to be used as multi-node computer. Azure allows its users to add more power to each VM with relative ease. You can change almost all of the hardware on the fly allowing you to make a VM that is more powerful than one node on Bridges easily. We could not Scale Up Bridges because we did not have access to the hardware.

2) Scaling Out: During the project we were able to request different numbers of nodes on Bridges easily. We would just ask for an amount of nodes, and wait in a queue to receive them. Once received, we could run the job we wanted, and then release them. As mentioned in Section II-D, we were not able to scale out Azure due to the COVID-19 pandemic. There is a method to scale out with multiple VMs called VNet. VNet simply connects multiple VMs together using peer-to-peer connections [9]. We were not able to test this because we could only create one VM, as we were not priority customers. Instead we attempted to create a series of nodes between the the vCPUs, but this was still more akin to Scaling Up than Scaling Out.

E. Azure Implementation

It was the goal of the project to accurately duplicate the specifications we found when using the Bridges HPC machine, using a custom built VM on the Azure platform. Azure is a fully scalable service that allows for a variety of hardware choices. This is thanks to their Platform as a Service (PaaS) model [10]. It means that if the VM you are running is low on memory or processing power, you can choose to scale up to a stronger build that may work better for your project. With this in mind, we sought to replicate the impressive specifications found on Bridges, using what was readily available for Azure. Unfortunately, as described in Section II-D, we were not able to request the necessary resources to build such a VM and so did our best to find the most capable model, according to cost and availability. Table I includes all information regarding the builds used for this project.

TABLE I
AZURE VMS USED

VM Name	vCPUs	RAM	Temporary Storage
E8v3	8	64GB	200GB
B8MS	8	32GB	64GB

After assembling a VM that met our budget as well as what was available, we launched an Ubuntu 18.04 server and began installing the necessary libraries. These included: Github for code centralization, the g++ compiler, UPC++ Version 1.0, as well as a few minor libraries that were necessary for dependency. Once the environment was up and running, we created scripts to run the programs using a maximum UPC++ and GASNet segment, according to the available memory. Due to the limiting amount of virtual central processing units (vCPUs), we scaled our test node size down to a maximum of eight nodes. The results can be found in Section IV.

F. Bridges Implementation

The Bridges supercomputer is maintained and owned by the Pittsburgh Supercomputing Center, which means there was no changes made to the software or hardware. Each node on Bridges uses an Ubuntu 18.04 kernel, which is similar to our Azure VM build. All of the necessary libraries and packages for UPC++ and various compilers were already installed for

use. This standardization allowed for a streamlined process when using Bridges, as all of the testing environment nuance was handled by PSC themselves. Table II includes all information regarding the Bridges HPC machine, specifically the RM partition that we tested on.

TABLE II
BRIDGES RM PARTION SPECIFICATIONS [4]

Number Of Nodes	752
CPUs	2 Intel Haswell CPUs 14 cores/CPU; 2.3GHz
RAM	128GB, DDR4-2133
Node-Local Storage	2 HDDs, 4TB each

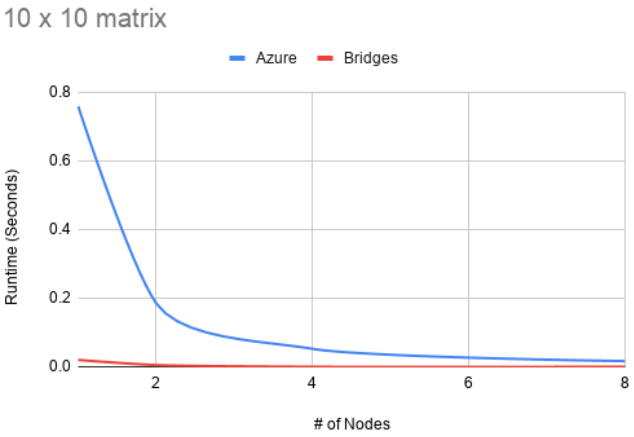

Fig. 4. Run times for a 10x10 matrix

IV. RESULTS

This section contains results from running experiments using the implementations, described in Section III. The testing criteria for both platforms involved scaling up and and scaling out, which was mentioned in Section II-A. Due to the current limitations for Azure, discussed in Section II-D, the maximum node count was set to eight, and the table matrix sizes to: 10, 25, 50 and 100. This was done to ensure all tests could be completed in a timely manner.

A. Matrix Size: 10 x 10

The first series of tests involved a 10 x 10 matrix. The results found in Table III detail the performance when running parallel on both platforms, while Figure 4 illustrates the run time difference between them.

TABLE III
AZURE VS. BRIDGES, 10 X 10 MATRIX

Name	Nodes	Time (seconds)
Azure	1	0.759447
Bridges	1	0.02
Azure	2	0.1848125
Bridges	2	0.005
Azure	4	0.052302
Bridges	4	0.0
Azure	8	0.016624
Bridges	8	0.0

These tests were run a number of times and averaged for best performance. This being the smallest sized matrix, performance was the fastest of all tests. Neither platform had any time above one second and gradually decreased to near zero results when increasing the amount of parallel nodes. However, it is a noticeable gap between Azure and Bridges throughout the tests.

B. Matrix Size: 25 x 25

The second series of tests involved a 25 x 25 matrix. The results found in Table IV detail the performance when running parallel on both platforms, while Figure 5 illustrates the run time difference between them.

TABLE IV
AZURE VS. BRIDGES, 25 X 25 MATRIX

Name	Nodes	Time (seconds)
Azure	1	2.920167
Bridges	1	1.82
Azure	2	0.755789
Bridges	2	0.46
Azure	4	0.190546
Bridges	4	0.11
Azure	8	0.095644
Bridges	8	0.03

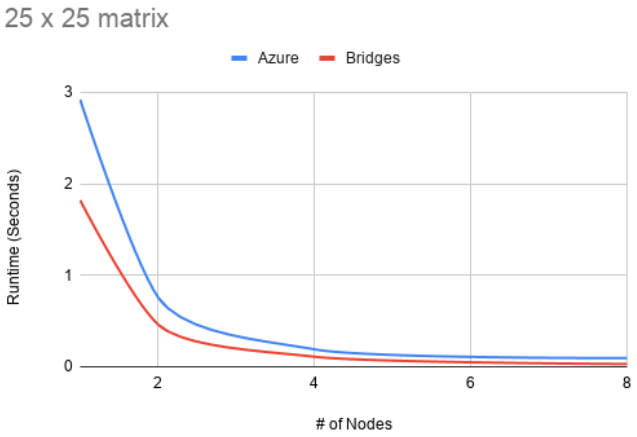

Fig. 5. Run times for a 25x25 matrix

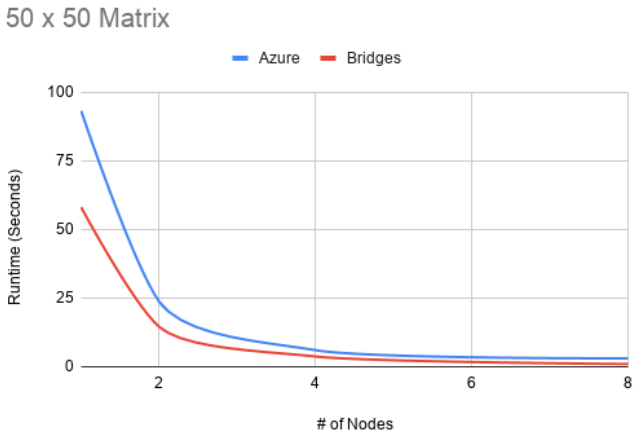

Fig. 6. Run times for a 50x50 matrix

These tests were run a number of times and averaged for best performance. At first, Azure is running considerably slower on one node, but begins to run at a near comparable rate to Bridges once the size is increased. We begin to see a trend where the result differences become smaller as the rate of nodes increases.

These tests were run a number of times and averaged for best performance. Similar to the results found in Section IV-C, we see a continued increase in performance by Azure that nearly mirrors Bridges, with the outlying result continuing to be when simulating on a single node.

C. Matrix Size: 50 x 50

The third series of tests involved a 50 x 50 matrix. The results found in Table V detail the performance when running parallel on both platforms, while Figure 6 illustrates the run time difference between them.

D. Matrix Size: 100 x 100

The fourth and final series of tests involved a 100 x 100 matrix. The results found in Table VI detail the performance when running parallel on both platforms, while Figure 7 illustrates the run time difference between them.

TABLE V
AZURE VS. BRIDGES, 50 X 50 MATRIX

Name	Nodes	Time (seconds)
Azure	1	93.232552
Bridges	1	58.110001
Azure	2	23.688656
Bridges	2	14.52
Azure	4	6.048251
Bridges	4	3.75
Azure	8	3.075061
Bridges	8	1

TABLE VI
AZURE VS. BRIDGES, 50 X 50 MATRIX

Name	Nodes	Time (seconds)
Azure	1	Not available
Bridges	1	1851.439941
Azure	2	Not available
Bridges	2	462.920013
Azure	4	Not available
Bridges	4	117.110001
Azure	8	Not available
Bridges	8	32.169998

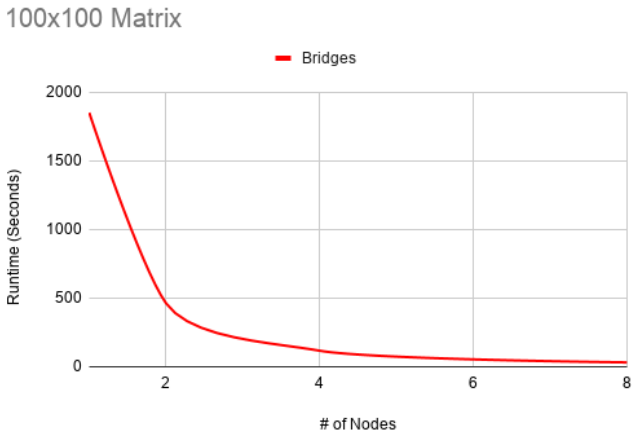

Fig. 7. Run times for a 100x100 matrix

These tests were run a number of times and averaged for best performance. This is when we found disparity between Azure and Bridges to be at its greatest. As described in Section II-D, we were limited in the build model we could employ when developing and launching our own VM. Although the intention was to emulate what resources we had on Bridges, compromises had to be made and became most evident the greater the size of the matrix. Originally, we intended to run computations on matrices larger than 100 x 100. This had to be changed due what was available using our VM. After a large number of attempts, we were never able to run a successful 100 x 100 matrix on the Azure VM as it would continuously kill the process due to running out of memory. As detailed in Section III-D2, we were not able to scale out our VM as we had been determined, a low priority customer during the current events.

E. Cost Efficiency

As mentioned in Section II-D we were not able to use the VM type that we wanted on our Azure server. This was also due to our 200 USD budget that we had to work with for the project. Table VII shows a list of more desirable VMs that are more comparable to one of Bridges' RM Nodes.

TABLE VII
AZURE VMS COMPARABLE TO BRIDGES' RM NODES [11]

VM	vCPUs	RAM(GB)	Price per hour(USD)
H8m	8	112	1.33
H16	16	112	1.97
G3	8	112	2.469
E16v3	16	128	1.177
E16-4sv3	16	128	1.177
E16-8sv3	16	128	1.177
F72sv2	72	144	3.873

1) Azure Pricing: The E-series VMs use an Intel XEON® E5-2673 v4 (Broadwell) processor, which is similar to Bridges. Each E-series VM also has 128 GB of RAM which is comparable to Bridges. The 4sv3 and 8sv3 allow for the use of multiple vCPUS per process making them similar the the two processor nodes that Bridges has. The F-series VMs use a Intel XEON® Platinum 8168 (SkyLake) processor, which is much better than Bridges. Each F-series VM also has a good amount of RAM, with the F72sv2 having 144GB. The F72sv2 is more comparable to a small HPC by itself as it has 72 vCPUs. The G-series VMs use a Intel® XEON® processor E5 v3 which is about the same as Bridges. These VMs have memory optimization built in and are made for databasing. The H-series VMs have the same processor that Bridges does, an Intel® XEON® E5-2667 v3 Haswell 2.3 GHz as well as utilize DDR4 RAM just like Bridges.

For cost efficiency, we chose the H-series VM as it is extremely similar to the hardware on Bridges, as well as made to be used with HPC style algorithms. Specifically, we used the H16 VM. If we were to run 752 H16 Nodes to make our cloud supercomputer, we would accrue an amount of 1,066,636.80 USD per month that it was active. However, this is not the only cost we would accrue. Storage on Azure VMs is temporary. If we wanted to do large scale data structure operations on our VM we would also need a storage account. The Storage account most comparable to Bridges would be a managed disk account. To get the same amount of data storage as one Bridges node, we would need a 8TB disk account, which is 946.08 USD per month [12]. However, we would need 752 of these accounts, which would run us 711,452.16 USD. This combined with the total monthly cost for the VMs gives us a grand total of 1,778,088.96 USD per month to run our cloud supercomputer.

2) Bridges Pricing: The exact pricing of Bridges is proprietary, so the following is guesses based on market values of the hardware. The Haswell processors run about 1,380 USD per processor, which is 2,760 USD per node. The RAM is about 384.99 USD per node and the storage is 300 USD per node. This gives us a total of about 3,444.99 USD per node. So, for 752 nodes, its costs 2,590,632.48 USD.

This cost is a one time fee. The only monthly costs for Bridges are things like: power, ventilation, location, etc. However, these things are also proprietary, and so we could not calculate them.

V. CONCLUSIONS

When running a parallel matrix multiplication algorithm on both Azure and Bridges, it is evident that one has greater performance than the other. However, although not identical we could make a comparable system with relative ease using Azure's services. The total cost for making such a machine would be extremely more expensive than an HPC machine. This conclusion is not complete because of the proprietary information we would need to get the real price of Bridges. That said, based on our research regarding the cost, we have reason to believe that it would be the same as purchasing the hardware necessary to build an HPC similar to Bridges as it would be per month to run the system on Azure.

The VM that we made cost 20 USD to run it for a week, but was significantly weaker than a single RM Node on

Bridges. However, with this set up, we were able to get scaling efficiencies that gave similar results. This is due in part to running a smaller algorithm that required less resources. Because of the size of the job, we could easily spend a lot less using Azure for smaller scale than the price of running and maintaining Bridges. That said, as the problem size increases and it gets more computationally intensive, the amount of resources needed becomes more and more expensive when using Azure, making it cost effective for smaller projects, but not for projects that require a larger amount of computations such as data modeling.

It is after much research, implementation, experimentation and data gathering that we grasp a new knowledge on the power, cost and scalability of high performance computing and cloud computing platforms. In the experiment done, the Azure system that we utilized was not as strong as Bridges, but delivered similar results. It was only after we reached limitations that we started to see the difference. However, we now have reason to believe that we could create Azure systems to run certain jobs that would be more cost effective. Though, in the long run it would be too expensive to run more computationally intense simulations comparable to the Bridges HPC machine. We now have a better understanding of how parallelism works and the performance we can expect from either high performance computing or cloud computing.

VI. Future Work

If this project were to continue forward, we would petition for access to more resources on Azure as well as set up some form of credit to maintain no out of pocket cost. We believe that Azure is capable of handling the processes we can run on Bridges, and with the ability to pay as you use resources, can be more cost effective in smaller bursts. Also, we would like to test the limits of both platforms running at peak performance, although with current events limiting this goal, to attempt so at another time would be optimal.

References

[1] R. Bias, "Grid, cloud, hpc ... what's the diff?" Available at http://cloudscaling.com/blog/cloud-computing/grid-cloud-hpc-whats-the-diff/(Last Accessed: 2020/4/19).

[2] C. Downing, "How the cloud is falling short for hpc," Available at https://www.hpcwire.com/2018/03/15/how-the-cloud-is-falling-short-for-research-computing/ (Last Accessed: 2020/4/19).

[3] Microsoft, "Scaling up and scaling out in windows azure web sites," Available at https://azure.microsoft.com/en-us/blog/scaling-up-and-scaling-out-in-windows-azure-web-sites/(Last Accessed: 2020/5/11).

[4] PSC, "System configuration," Available at https://www.psc.edu/bridges/user-guide/system-configuration(Last Accessed: 2020/5/10).

[5] BerkeleyLab, "Upc++ version 1.0," Available at https://bitbucket.org/berkeleylab/upcxx/wiki/Home(Last Accessed: 2020/5/11).

[6] PSC, "Covid-19 research," Available at https://www.psc.edu/bridges-for-covid-19-research(Last Accessed: 2020/5/11).

[7] Microsoft, "Update 2 on microsoft cloud services continuity," Available at https://azure.microsoft.com/en-us/blog/update-2-on-microsoft-cloud-services-continuity/(Last Accessed: 2020/5/10).

[8] M. J. Foley, "Microsoft: Cloud services demand up; prioritization rules in place due to covid-19," Available at https://www.zdnet.com/article/microsoft-cloud-services-demand-up-775-percent-prioritization-rules-in-place-due-to-covid-19/(Last Accessed: 2020/5/10).

[9] Microsoft, "Tutorial: Connect virtual networks with virtual network peering using the azure portal," Available at https://docs.microsoft.com/en-us/azure/virtual-network/tutorial-connect-virtual-networks-portal(Last Accessed: 2020/5/10).

[10] ——, "What is paas?" Available at https://azure.microsoft.com/en-us/overview/what-is-paas/(Last Accessed: 2020/5/11).

[11] ——, "Windows virtual machines pricing," Available at https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/(Last Accessed: 2020/5/10).

[12] ——, "Azure storage overview pricing," Available at https://azure.microsoft.com/en-us/pricing/details/storage/(Last Accessed: 2020/5/10).