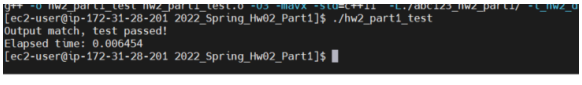
David Vondran, Diego Mercado
4/15/2022
CS450 Mini Project

In this project and experimentation, we decided to compare and contrast the differences between two important aspects of modern computing, these being cloud computing and high performance computing. Oxford dictionary defines cloud computing as "the practice of using a network of remote servers hosted on the internet to store, manage, and process data, rather than a local server or a personal computer". This means that computers can be remotely accessed and tasked with completing computational problems, allowing users of the service to temporarily make use of a more powerful or specialized system than what they may have available without needing to invest heavily into acquiring and supporting one of these systems themselves. High Performance Computing is defined as "the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business". This is utilized to solve very computationally expensive problems that would be difficult or impossible for a standard system to solve. For our purposes, we decided to make use of several tools that were available to us, these being Penn State's ROAR supercomputer cluster to work on our high performance computing, and Amazon's Amazon Web Services (AWS) to stand in for our cloud computing needs.

Amazon Web Services is a multi-facet cloud platform, which offers over 200 cloud-based services for its members. AWS can be utilized by users to host their projects and applications, as well as to use the pre-existing services to their advantage. For the sake of our project, we will be using AWS's ec2 instance, which is defined by Amazon as "a web service that provides secure, resizable compute capacity in the cloud". AWS's ec2 instance is free for use by anyone who has access to an AWS account, and can be used to make web-scalable computing easier for developers. For the sake of our project, we will be using this ec2 instance to run an application on the AWS cluster and compare the runtime and computing power of this to that of the PSU ROAR supercomputer. As the ec2 instance is significantly less powerful than that of the ROAR supercomputer, we do not expect for the results to be significantly similar, however this will help us gauge the relative power of the systems while comparing their costs and different applications.

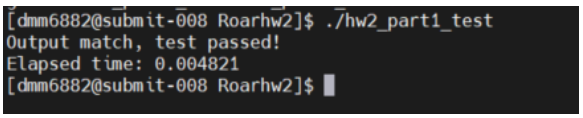
In regards to the PSU ROAR supercomputer, we will be testing our code on the ACI-i node of the cluster, which includes a 2.8 GHz Intel Xeon Processor, 20 CPU/server, 256 GB RAM, FDR Infiniband, and 40 Gbps Ethernet running on Red Hat Enterprise Linux 7. This is the node of the cluster that the homework assignments are computed on, and is a book comparison unit to compare the performance and use cases of the AWS machine to. On the other hand, the AWS machine contains an Intel Xeon 2.4 GHz Processor with 12 cores and 24 logical threads. The cache levels for this machine are L1: 32k, L2: 256k, L3: 30,720k. Overall, the total RAM available on this machine is 980Mb. The AWS Machine runs on Amazon Linux 2.

Using these two systems, we decided to compare the results of running different types of code on the different systems to compare the outputs in runtime. We decided to use some of the already completed homework assignments to benchmark the different systems, however some modifications were required to be made to the assignments as the AWS system had significantly limited amounts of RAM (less than 1 GB). These homeworks were homeworks 2, 3, and 4. We chose to run these code chunks Homework 2 was run one time on each system, as it is not an iteration dependent code chunk. We were able to find that this code executed in time .006454 seconds on the AWS cluster.



On the ROAR cluster, we were able to see that the same code ran in time .004821 seconds. This was significantly faster in terms of relativity on the

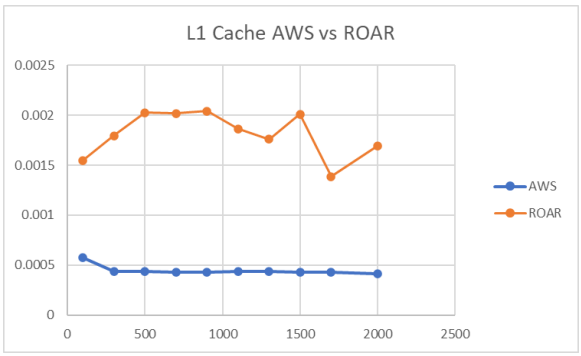
ROAR computer, which was expected by us due to the difference in specs.



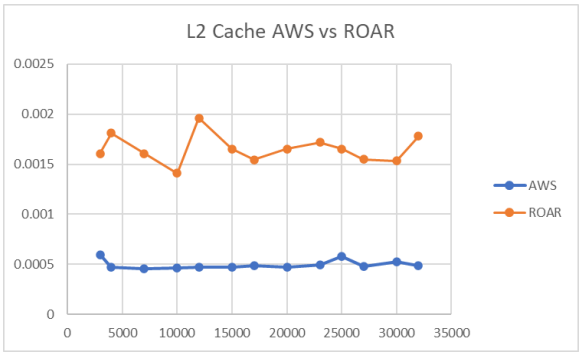
However, we can also see that both are very fast times with objectively not too much difference between the values in the overall sense. Thus, we will be testing the more involved problems of hw3 and hw4 to further discover the difference in computing power between the two systems.

First off to set up our AWS instance we mainly decided to pick all the options which were free tier eligible. To set up the machine we first need to pick from a variety of Operating Systems. Some of the OS AWS provides include: Amazon Linux 2, Ubuntu, WIndows, Red Hat and a few others. We decided to go with Amazon Linux 2 since it was what we were most familiar with. Our next decision was to pick an instance type, AWS provides a variety of specs which will suit any need. They can provide machines which are suited for computation optimization , memory optimization, accelerated computing and storage optimized. For the purpose of our project we decided to pick a general purpose machine since all the others would cost a fair amount. The specific machine we picked was the t2.micro which is a balance of compute, memory and network resources and includes a high frequency Intel Xeon processor. T2 instance types are generally used for a variety of general purpose workloads such as low latency interactive applications, small to medium databases, virtual desktops, development, building environments, code repositories and product prototypes. Since this is a virtual machine that we need to SSH, AWS provides the option of creating keys instead of using traditional passwords. Usually you would have a keypair that consists of a public key and a private key that are used to prove your identity when connecting to an Amazon Ec2 instance. The public key is stored on the instance while we store the private key. The keys are encrypted either with ED25519 or 2048-bit SSH-2 RSA keys. We decided to create one for our instance since it's generally considered good practice in cloud computing and keeps our system more secure. The last step to setup our machine is configuring the network settings. Each EC2 instance is part of a security group that can configure and edit inbound and outbound rules whether they are TCP or UDP there are many ways to configure the system. For our project we decided to limit the machine to only accept connections from my IP address. This provides an extra layer of security and since we are dealing with cloud computing security is one of the main pillars for achieving a well balanced system. Once we had our machine setup it was a matter of logging into the machine by specifying the private key, username and ip address. Finally we only needed to install a compiler that would run our code. We managed to install the g++ (GCC) 7.3.1 and with this we were able to compile and run our code. Since our machine does not have the great specs there were some changes that we needed on our homeworks when we were benchmarking both machines. For Homework 2 we had to modify our test.cpp and change the value of N. In this case we decided to divide by 10 since if we ran the original version in AWS the program would crash. This would happen because the values we were using could not be stored in the system's ram. For Homework 3 and 4 we had to modify the main.cpp and change the values of N to fill the caches for l L1, L2, L3 and Ram. For L1 we decided to go from 0 to 2000. For L2 we go from 3000 to 32000. For L3 we go from 35000 to 3800000. For RAM we go from 4000000 to 13000000. We looked at information about both processors in the AWS and ROAR systems. We determine what values of N would fit at each different cache level.

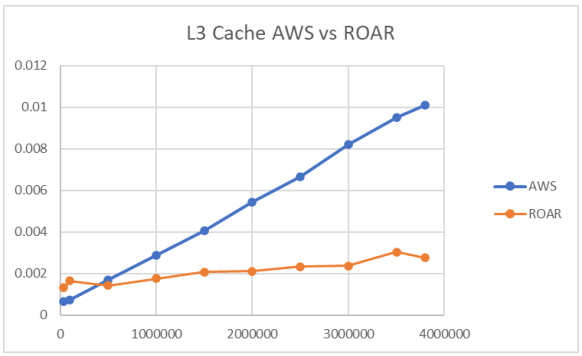
For homework 3, we gathered values of runtime for the different values of N we listed above. We ran the std method from homework 3 on both systems. Below is a graph showcasing the relation between the different values of N and the execution time with these different values being used.



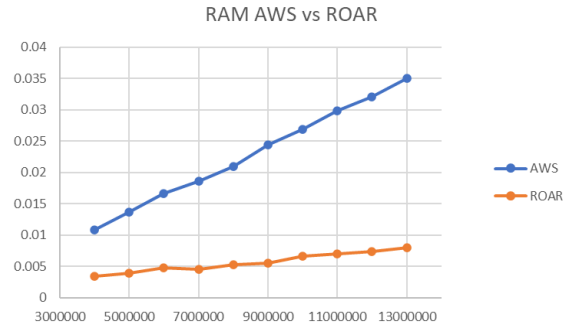
As we can see, the AWS values are significantly smaller than those of the ROAR, as well as significantly more consistent. While this initially came as a surprise to us, it made more sense when understanding that the AWS system was built with faster lower level cache, thus making memory access faster for small values. This is because the node we allocated for use on the AWS system has been designed with smaller computing problems in mind, and thus is faster when dealing with these smaller numbers. In regards to the level 2 cache, our experimentation resulted in the following graph.



The level 2 cache in this problem also shows a similar outcome to that of the level 1, with the AWS system providing faster times on the smaller values much more consistently. We can attribute this to the same reasons as those of the level 1 cache, with faster access time for smaller values, as the system was designed for more small scale problems than the ROAR's architecture. Moving on, we now have the values comparing the level 3 cache.

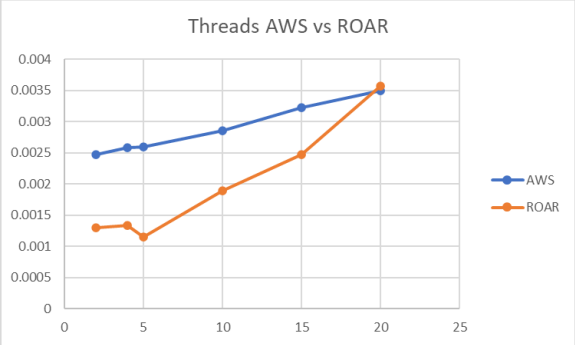


Something interesting to note about the ROAR system processor which we used is that it did not actually contain level 3 cache, but instead remained on level 2 for some of these values and moved on to RAM for some of them. However, we can still compare the execution times for the values given between the two systems. This range of values is where the ROAR begins to operate within the values it was designed for, and begins to quickly overtake the AWS system in execution time. We can see the AWS system quickly increasing linearly while the ROAR system increases very slowly or stays level. This showcases the better RAM access speed of the ROAR system and highlights the design decisions that went into it as opposed to those of the AWS system. Finally, we move onto the different values of the RAM for both systems.



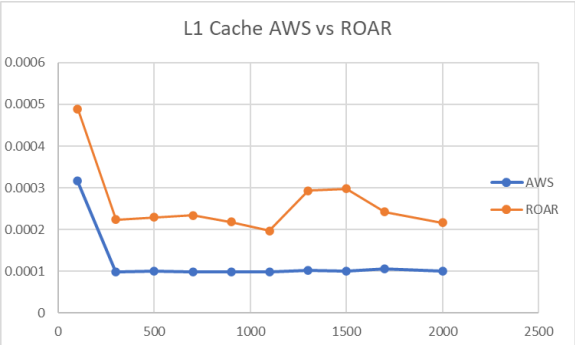
Here we can see that as the values of N increases further, the gap in execution time between the two systems steadily grows, and shows no signs of slowing down as the values further increase. This shows the superiority of the ROAR system over the AWS system for significantly large values, and further highlights the design philosophy that went into the construction of both systems. Finally, we decided to test the multithreading capabilities of both systems and compare them to one another by running the test with a variable number of threads.

For this test, we set N = 1000000. The results are shown below.

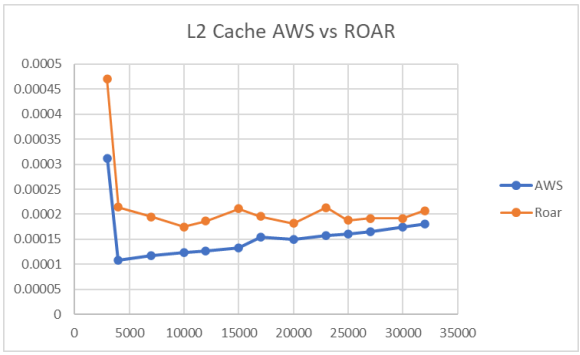


As shown, the ROAR system is generally significantly better at executing multithreading operations, which is explained by the larger number of cores that its processor contains and the server-based design that it was built around. The system is designed to handle significantly larger workloads that that of the lower-level AWS system we accessed, and thus it would make sense that Intel would design the processor used in thai system to be able to handle multithreading operations significantly better than the designers of the processor in the AWS system would need to. It is however interesting that for 20 threads, the ROAR system overtakes the AWS system slightly in execution time. We believe this to be a runtime outlier as it is not replicated in our rendition of the homework 4 results.

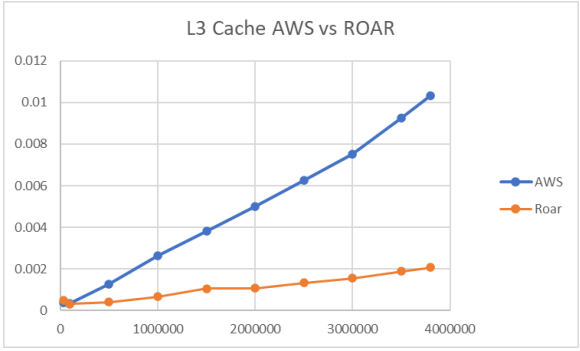
Now we will look at the same tests conducted on homework 4 version of the code. We used the same values for N in order to keep the results consistent and view the difference between them. Shown below, we have the level 1 cache runtime results running on the std method of homework 4.



We can see that the results for this run of the code are much more consistent and replicative of one another. This is likely contributed to the optimization of the OMP system, which is likely to be more optimized than our rendition of the std threads design. However, we still see that the AWS system executes faster and more consistently than the ROAR system, which highlights agan the different design philosophies and access speeds of the different cache levels. Moving on to the level 2 cache, we see the following results.

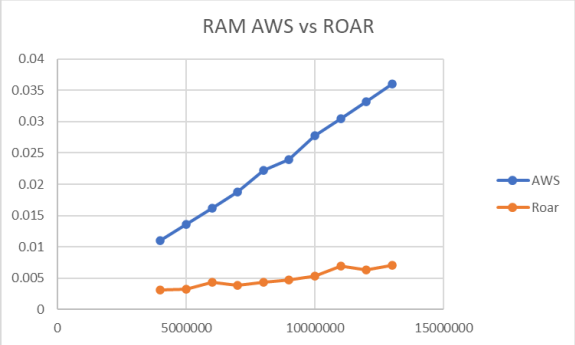


The graphs start off with a sharp drop-off at the beginning of the cache, which showcases the transition from one stage of the cache to the next. We then see the same behavior showcased in homework 3 again, in that AWS is consistently faster than ROAR, but as N increases, we see that the two grow closer together in terms of execution time. Towards the end of the level 2 cache, we see that the two have nearly identical execution times. Finally, transitioning into the level 3 cache (or the level 2 / RAM combo for the ROAR system), we find the following results.

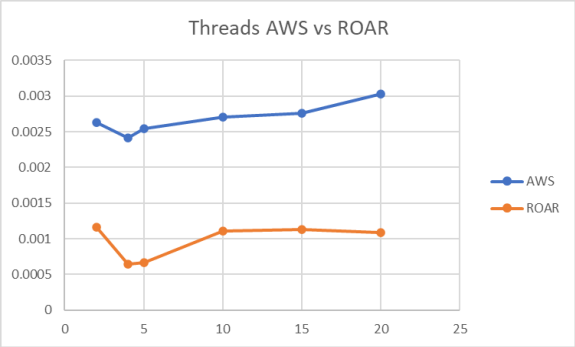


As documented previously, the execution time for the larger values of N is significantly lower for the ROAR system as opposed to the AWS system, and continues to grow in disparity as the values

increase. This once again highlights the difference in design philosophy between the two systems and their desired purpose. Finally, we have the RAM values of the two systems showcased in the graph below.



Here we can see the disparity continue to grow between the two systems, and as before, finalizes the logic behind the decisions that were made for each system. Lastly, we conducted the same experiment regarding variable number of threads between the two systems for homework 4 as well, with the results showcased below. The value of N for this experiment remains N = 1000000.



Here we can see again what was shown previously, in that the ROAR system is significantly stronger when it comes to execution with multithreading, this time without the strange outlier shown in the homework 3 rendition of the experiment. This shows once more the design decisions behind the ROAR computer and for what purposes it was designed.

As hinted largely above, the difference in the tow systems comes down largely to the workloads they were designed to handle and the problems that they are expected to solve. The AWS system is a lower level system designed to be used by an individual or small organization to handle relatively small computational tasks. This is highlighted by it's strong performance on smaller problems, where it's execution time was faster than that of the ROAR supercomputer cluster. However, as the size of the problem increases, the ROAR system is far and away the winner, showing faster execution times with increasing disparity between the results of the two execution times. The rate of growth between the two execution times is also shown to be significantly slower for the ROAR system opposed to the AWS system, indicating that this disparity will only continue to become more significant as size increases. It is important to note as well that due to the size of the memory of the AWS system (less than 1 GB RAM), all of our tests contained small values of N even on the large end, where the ROAR system is designed to be able to accommodate up to 256GB of information stored in it's RAM. Thus, our tests only covered the very beginning of the capabilities of the ROAR system on execution of large problems.

Ultimately, what our experimentation has shown is the difference in design philosophy between cloud computing and high performance computing. The two, while not necessarily existing separately from one another, are different things with different purposes in mind. Cloud computing offers access to different types of machines at any given time depending on the situation that is required. The user can simply request access to a machine that is able to handle more load more efficiently than whatever system they have available locally in order to compute their task. The power of these machines can range from moderate to extremely powerful, as they are then providing a benefit over whatever the user has available. However, purpose built supercomputers like Penn State's ROAR are designed as research and specialized-job computers with the idea of massive workloads and efficiency with large values in mind. These computers would be significantly less efficient to use as a general cloud-operated computer due to the massive power draw and infrastructure required, for what would assumably be a job that does not require this degree of resources. Overall, the two options are designed to serve different purposes, and while degrees of overlap may exist,

the systems have distinct purposes that serve different functions effectively.