LAB 1: THE IMPACT OF MEMORY SYSTEM DESIGN TRADEOFFS ON MACHINE PERFORMANCE

GOAL

- Becoming familiar with our computing facilities.
- Understanding the cluster architecture and getting familiar with the cluster computing environment.
- Studying the impact of memory system design tradeoffs on machine performance.

WebClass - Lab Instructions Web Page has been assembled that describes many of the facets of the PDS Lab where you will do most of you work. It will be referenced throughout the rest of this document. The main page of WebClass is http://pds.ucdenver.edu/webclass/index.html

SUBMISSION

- Submit your lab answers as Word documents (.doc, .docx, .xls) attachments to Canvas. You can answer the lab questions directly in this file or in a separate file. Make sure the submitted file's name is "Lab1_firstname_lastname.docx".
- Section 1, 2 must be done individually. Section 3 can be done in teams.

If you have any questions regarding labs, please send email to TAs for assistance. The labs in this class require you to be familiar with basic Linux commands. Quick tutorials to some useful command here:

- https://www.linuxtrainingacademy.com/linux-commands-cheat-sheet/
- https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners
- https://www.tutorialspoint.com/linux_admin/basic_centos_linux_commands.htm
- https://centoshelp.org/resources/commands/linux-system-commands/
- https://www.tecmint.com/linux-commands-cheat-sheet/

Section 1 (1 pt). Getting Started with Multi-core clusters

1. Login Account

Before/On the starting date of Lab 1, check canvas announcement for your account information (username and password) to login to Heracles.

Once obtaining account information, you can login to the servers using Secure Socket Shell (ssh). On Windows, you can use PuTTY (download at https://www.putty.org/). On Mac, you can use "ssh" command. Please familiarize yourself with ssh and make sure you can login to each server successfully before proceeding to next sections.

2. Multi-Core Cluster Architecture

Study Heracles at http://pds.ucdenver.edu/webclass/Heracles_Architecture.html. You can also monitor Heracles by accessing https://132.194.186.215/mcms/ in order to observe some metrics (CPU, Network, memory, etc.). Note that you need to be on our campus's network and if the link shows a not secure message, you can go to Detail, then "go to the webpage".

In your word doc, answer the following questions.

Question 1 (0.5 pt). Write a short paragraph in your own words about your understanding of this machine.

1 master node, 16 nodes without a GPU and 1 node with GPU, making for a total of 18 nodes. All nodes connect to the master node. The master is used to compile the code (gcc -o), allows the user to login and submits the program for execution. Never run code on master, it is doing a lot of work. If we need to run code then we ssh to run code on the other nodes. If we need to run any code on GPU, then we need to run it on node 18, otherwise just regular C++ code then we can just run it on nodes 2-17. There are 4 levels of cache. It's just a computer spread out very wide.

Question 2 (0.5 pt). Give few examples of applications that can benefit from this type of machine. What challenges come with this type of machine?

Initially, the node-like structures reminded me of the cryptocurrency mining network with all of these verifying nodes so my first thought was thinking that it could be used for that and dove into the raw computing power on the one node if it can actually handle it. After looking at the NVIDIA Tesla P100 specifications, it has the same "relative performance" as my personal GPU, the AMD Radeon 5700 XT. I was also comparing benchmarks between this GPU and NVIDIA's new 3080 TI. If we were to compare one of the four P100's against one 3080 TI, it has about half the technical specs as the 3080 TI. So the challenge if this were to be applied to mining then it wouldn't be as powerful as current technology while trying to be cost efficient with all of the current nodes. I can imagine any sort of gaming, homeworks and any other application that you can use on a regular desktop(running a VM, cloud computing, multithreaded applications) would also benefit from this type of machine. But there are 17 other nodes so there is a collective use for the entire Heracles machine. Heracles is mostly optimized to observe the workings of a multiprocessor machine, through each cycle and from each level. Going from instruction fetches at the microprocessor core of each node at the level of understanding Register Transfer Language(RTL). Diving into the CPU, the Xeon Intel E5-2650v4, it is benchmarked to handle a lot of multithreading, which

A challenge with this machine would be scaling the hardware to accommodate extra throughput. Another challenge would be to be able to write the code to send out to specific nodes. You can write code that will compile and execute for 16 nodes and make sure that those programs do not require a GPU. For the one that does require a GPU, the code must know to send to the node with the GPU on it. You also need to make sure that the program that you write gets sent to A node, not the master node.

Section 2 (4.5 pts). Case Study 2: Putting It All Together: Highly Parallel Memory Systems

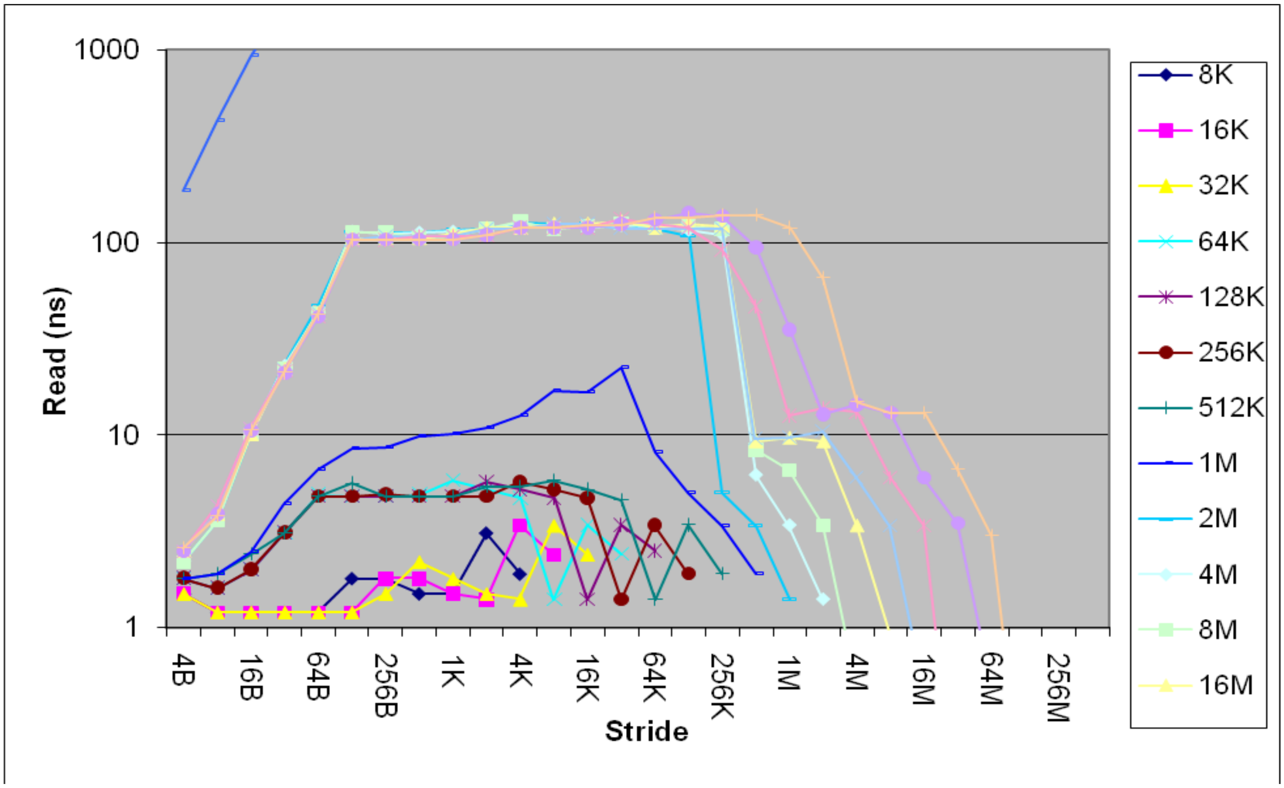
The questions in this Section must be answered individually.

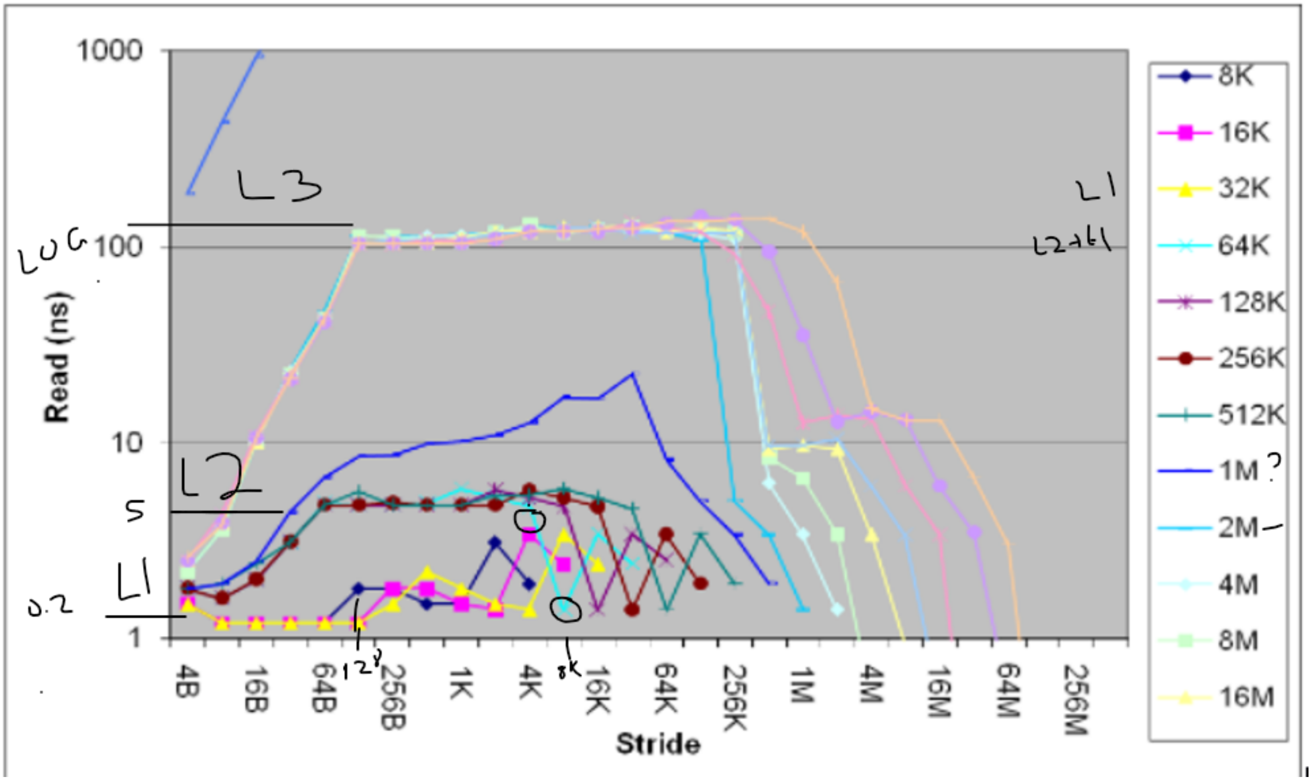
Read the "**Case Study 2: Putting It All Together: Highly Parallel Memory Systems**" including its descriptions and the program (Figure 2.29) in the text book 6th edition (pages 150-152) and answer the following questions:

Question 1 (0.5 pt). Explain **briefly** your understanding of this Case Study 2.

This is about evaluating the behavior of a memory system. This is achieved by invoking different levels in the memory hierarchy. The program tracks physical addresses and virtual addresses and loops multiple times to measure CPU time. There is also a main memory and hard disk

Question 2. (4 pts) Use the sample program results in **Figure 2.33 in the textbook** to answer following questions





a) (1 pt) How many levels of cache are there? Why?

- typically there are 3 levels of cache
  - We can see here at the bottom, closest to the 1ns read time, we have the first level cache, which is the fastest read time. It is estimated to be 32Kb

        - The middle is L2 cache, where the next round of reading occurs, it can either be 1M or 256K
        - The third is L3 cache where it is the slowest and is about 4M

b) (1 pt) What are the overall size and block size of the first-level cache? Why?

    - Cache size is achieved by increasing N until cache misses start to occur
    - 8K is when cache misses start to occur, but since it starts to miss at the first array size and it being so small, does L1 cache even exist? The largest size of first level cache is 32 K where all of the arrays can fit in comfortably.
    - line size = block size = 128 because that is where the misses start to occur at 128 strides
    - 32 instructions/ block
    - rate at which misses occur every b/s iteration. Rate increases with s and achieves maximum
    - 32K/128 bytes = 250 blocks would be in the cache

c) (1 pt) What is the miss penalty of the first-level cache? Why?

    - miss penalty for L1 cache is 4 ns. It is based on the difference in size between L2 and L1 and determines the most significant change. The next array size up from L1 is L2 and the most significant change is at 5ns. Subtract from the 1ns of L1 cache, it is 4 ns

d) (1 pt) What is the associativity of the first-level cache? Why?

    - associativity is given by N/S for a >= 2
    - if cache L2 is 8 way set associative then cache L1 is 8 way set associative
    - Once the array gets to a certain size, it can be accessed in L1
    - For N >D (L1 cache size)
    - 64/8 = 8 way set

To answer the above questions, you may need to read a paper by Saavedra-Barrera[1992]. This paper is entitled, "***CPU performance evaluation and execution time prediction using narrow spectrum benchmarking***". Obtain this paper here [https://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/CSD-92-684.pdf]. You can also download the paper via the Auraria Library portal; it is also freely available on Google Scholar. Within **Chapter 5 of this paper, read pages 169 to 187**.

183

Section 3. (4.5 pts) CPU performance evaluation on Heracles.

- The goal of this section is to apply what you have learned in case study 2 to evaluate the memory system of Heracles.

- To obtain reliable results in this section, we suggest you run on the nodes that no one is using it. You can check the CPU utilization of Heracles by accessing its website. If there is someone else using a node, the CPU utilization will be high. This indicates you should run on other nodes. We also suggest that you run the program with each configuration for at least 3 times and average your results.
- In any cases, you should know that Heracles are not a “perfect” system to run this experiment. It is not guaranteed that cache at the time you run experiments is clean (or empty). Thus, to answer the questions in this Section, you should try to reason your answers based on your understandings. No single correct answer is accepted in this section.
-

Question 1. Compile and run the given source code: **mhdHeracles.cpp** for Heracles. Fill the below table and generate a plot (Hint: this plot is similar to the plot in Figure 2.29). See Section 4 for instructions how to compile and run a program on Heracles.

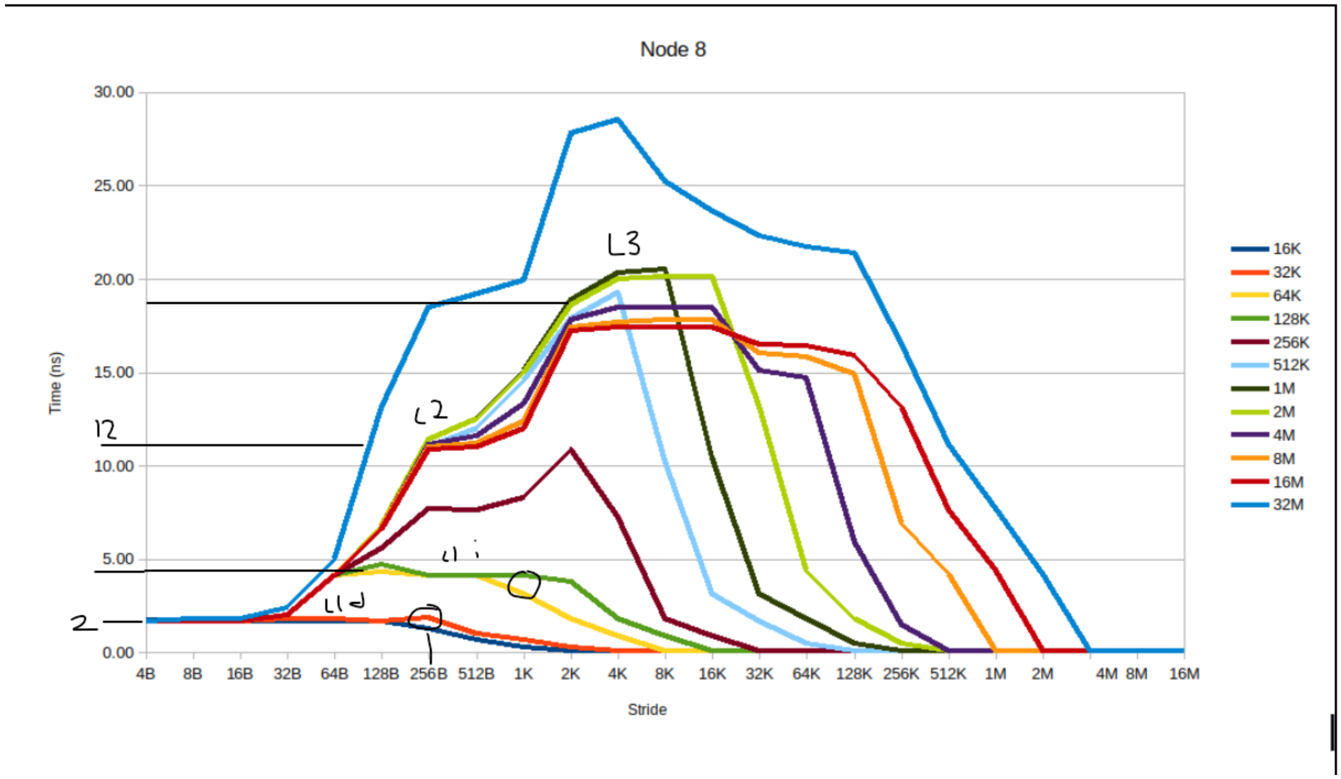
Heracles Results (0.5 pt):

Node 8

	4B	**8B**	**16B**	**32B**	**64B**	**128B**	**256B**	**512B**	**1K**	**2K**	**4K**	**8K**	**16K**	**32K**	**64K**	**128K**	**256K**	**512K**	**1M**	**2M**	**4M**	**8M**	**16M**	**32M**
16K	1.72956	1.7	1.7	1.7	1.7	1.7	1.3	0.7	0.3	0.1	0.1	0.1												
32K	1.7	1.7	1.7	1.8	1.8	1.7	1.9	1.0	0.7	0.3	0.1	0.1	0.1											
64K	1.7	1.7	1.7	2.0	4.1	4.3	4.1	4.1	3.1	1.8	0.9	0.1	0.1	0.1										
128K	1.7	1.7	1.7	2.0	4.1	4.7	4.1	4.1	4.1	3.8	1.8	0.9	0.1	0.1	0.1									
256K	1.7	1.7	1.7	2.0	4.1	5.6	7.7	7.6	8.3	10.9	7.2	1.8	0.9	0.1	0.1	0.1								
512K	1.7	1.7	1.7	2.0	4.1	6.7	11.0	12.0	14.6	17.9	19.3	10.2	3.1	1.7	0.5	0.1	0.1							
1M	1.7	1.7	1.7	2.0	4.1	6.8	11.4	12.5	15.1	18.9	20.3	20.5	10.4	3.1	1.8	0.5	0.1	0.1						
2M	1.7	1.7	1.7	2.0	4.1	6.8	11.4	12.5	15.0	18.6	20.0	20.1	20.1	13.2	4.4	1.8	0.5	0.1	0.1					
4M	1.7	1.7	1.7	2.0	4.1	6.7	11.1	11.6	13.3	17.8	18.5	18.5	18.5	15.1	14.7	5.9	1.5	0.1	0.1	0.1				
8M	1.7	1.7	1.7	2.0	4.1	6.7	11.0	11.2	12.4	17.4	17.7	17.8	17.8	16.0	15.8	14.9	6.9	4.2	0.1	0.1	0.1			

16 M	1.7	1.7	1.7	2.0	4.1	6.6	10.9	11.0	12.0	17.2	17.4	17.4	17.4	16.5	16.4	15.9	13.1	7.6	4.4	0.1	0.1	0.1		
32 M	1.7	1.8	1.8	2.4	5.0	13.1	18.5	19.2	19.9	27.8	28.5	25.2	23.6	22.3	21.7	21.4	16.5	11.1	7.7	4.2	0.1	0.1	0.1	
64 M	1.7	1.8	1.9	2.9	6.3	18.7	34.9	36.5	38.2	65.9	68.0	68.9	76.1	77.1	77.8	74.7	24.7	16.7	12.1	7.7	4.3	0.1	0.1	0.1

Plot Heracles graph here



Use the results from Heracles to answer the following questions:

a) (1 pt) How many levels of cache are there? Why?

- There are 3 levels of cache, L1 is split, but still on the same L1 cache. It is listed on the Heracles architecture website that is sited and listed as reference on top of this document
  Cache Hierarchy
  L1d cache: 32K
  L1i cache: 32K

L2 cache: 256K
L3 cache: 30MB

b) (1 pt) What are the overall size and block size of the first-level cache? Why?
   - Size of the cache is achieved by increasing the value of N until cache misses start to occur
   - First level cache is split into two, so we won't consider a miss where the cache's split.
   - L1d starts to have a cache miss at 32K with a block size of 256
   - L1i starts to have a cache miss at 64k with a block size of 1K

c) (1 pt) What is the miss penalty of the first-level cache? Why?
   - 10 ns since this is a linear scale now, we just subtract where L2 starts and where the root of L1 starts. L2 starts at 12 ns and L1 starts at 2 so the miss penalty is 10 ns
   - if we were to include 256 KB into L2 cache, then the miss time would be 4ns

d) (1 pt) What is the associativity of the first-level cache? Why?
   - Associativity is given by N/s where N = array size and s = stride
   - So if we apply the same process as section 3, L2 would merge down to L1 cache 512/64K = 8 way set associativity

http://pds.ucdenver.edu/webclass/Heracles_Architecture.html

SECTION 4. COMPILING AND RUNNING THE PROGRAMS

Copy source code files to Heracles

Step 1: Logon to the cluster (see Lab 1) and follow the steps:

Step 2: Make a folder named csc5593 in your home directory using the **mkdir** command.

```
mkdir /path/to/directory
```

For example:

```
mkdir /home/john/csc5593
```

Step 3: Copy the source code files to csc5593. If you are using MAC or Linux, you can copy these files using scp or sftp command. If you are using Windows, you can use Bitvise, WinSCP, or SSH Secure Shell to login and copy files from your PC to the cluster.

- Change the working directory to csc5593 using **cd** command

```
cd /path/to/directory
```

For example:

```
cd /home/john/csc5593
```

Step 4: Compile source code files

To compile the C++ Programs on Heracles, read:

http://pds.ucdenver.edu/webclass/Compiling%20C_C++%20and%20Fortran%20programs.html

Step 5: Run your program on Heracles.

The first step when running your program on a compute node is to find an open node to run on.

Access this link to monitor Heracles Cluster: https://132.194.186.215/mcms/

Try to select a node that shows the lowest usage.

Before you run your program on Heracles, read this entire page: http://pds.ucdenver.edu/webclass/Heracles-RunningPrograms.html

To run your program via batching system (SLURM), visit:
http://pds.ucdenver.edu/webclass/Heracles-RunningPrograms%20Slurm.html