HOMEWORK 1
EECE 5640: High Performance Computing

Shreyash Shrikant Jadhav
NUID: 001057608

Part 1

a.) The three benchmarks used were drystone, linpack and whetstone.
System used taking the measurement was COE server. Following is the configuration of server.
CPU Model: Intel Xeon CPU
Frequency: 2660 MHz
Number of cores: 144 (24 CPUs and 6 cores each CPU)
Memory size: 47 GB
Operating system version: CentOS Linux 7

Drystone:
Drystone readings were taken for the loop count of 500000000.

	No optimization	O1	O2	O3
Drystone	57	16	16	12
500000000	57	16	14	13
	55	17	13	12
	56	18	14	13
	61	19	14	12
	56	18	13	12
	60	18	14	13
	56	18	13	12
	57	18	14	13
	56	18	13	12

Figure 1: Measurements for drystone

Linpack:

	MLOPS	MLOPS	MLOPS	MLOPS
Linpack	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1857.407407	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247
	566.666667	1857.407407	2156.989247	2156.989247
	566.666667	1910.47619	2156.989247	2156.989247

Figure 2: Measurement for Linpack

Whetstone:

	O0		O1		O2			
	time	MIPS	time	MIPS	time	MIPS	time	MIPS
Whetstone	18	2777.8	12	4166.7	10	5000	9	5555.6
500000	18	2777.8	12	4166.7	10	5000	10	5000
	18	2777.8	12	4166.7	10	5000	10	5000
	18	2777.8	11	4545.5	10	5000	9	5555.6
	18	2777.8	12	4166.7	10	5000	10	5000
	18	2777.8	12	4166.7	10	5000	10	5000
	18	2777.8	12	4166.7	10	5000	9	5555.6
	18	2777.8	12	4166.7	10	5000	10	5000
	18	2777.8	12	4166.7	10	5000	10	5000
	18	2777.8	12	4166.7	10	5000	9	5555.6

Figure 3: Measurement for Whetstone

For drystone with no optimization we get 15 seconds as average time. While in the case of Linpack we are measuring the performance with the MFLOPS (Million floating point instructions per second). We are seeing how the change in the optimization will produce change in the MFLOPS instead of time for the performance. In case of whetstone we are seeing both the time and MIPS. As MIPS does not give proper performance of the system we can use time as a metric to compare the optimization.
It was observed that, there was some difference in the reading during the ten runs. The following could be the reasons for the change in the performance:

- During first run the code is loaded into the cache from the memory, therefore it would have less spatial locality to access the data and this would result in the more execution time. And then once the code is in the cache the processor can access the data with greater speed. This could be one of the reasons.
- While searching on the internet I came across a term called as cold cache which could mean there is no values stored on cache and therefore for the first run, cache will store either the addresses or data on it. So, in the later runs the processor will fetch the contents from cache which is now became warm cache which means that cache is loaded with contents of the program and therefore later execution were faster
- The other result that I noted was with other users on the system, running the drystone program there was no significant difference observed in the execution time. But there was a significant slower observed when running on a system with no other user present on the system as explained above.
- The reason behind this could be that other users are online and are performing some operations. As the cache is shared among all the users, whenever we run our program the previous contents of the cache need to flushed in order to allocate space for our program.
- This will increase the execution time of the program for the first run. So as the cache gets warm it will decrease the execution time.
- But there was another thing I observed that in middle the execution time would increase, this could be because as there are other users using the cache space and depending of cache replacement policies could be the reason.

b.) The following are the optimization used:
1. –O: - This is default optimization that gcc compiles in when no optimization is specified. In this there is no optimization in the compilation code and therefore has faster compilation time as compared to the other optimization.
2. –O1: - This is the basic level of optimization. In this, the compiler tries to reduce the execution time and the code size without introducing any of the optimization which may take more

compilation time. In this compilation time is lower as compared to another higher optimization. There are around 45 flags that O1 activates.
3. –O2: - It is more optimized version of O1. In this gcc performs all the optimization which would not involve any space-speed trade-off. Using this optimization, the compilation time and the performance of the program increases. In this there are around 35 flags that this optimization activates.
4. –O3: - This is the more optimized version of O2. Along with the O2 flags there is more 16 flags of its own to increase the optimization of the program.

In my experiment I have used all the above optimizations. From figure 1,2 and 3 we can see that there is not much difference in the performance for O2 and O3 as the program is mostly optimized. In drystone using function inline we can see that the performance is increased which is performed at O1 as drystone is using different functions to perform the operations. Along with function inlining there is also speculative execution flag which helps in optimizing the conditional jumps.
In Linpack there is also same process being followed as it has different functions so function inlining is performed.
While in the case of the whetstone there are not many functions used so the function inlining wont produce significant change in the performance. But the speculative execution in conditional jumps will produce that effect.
So depending upon how the benchmark is written it will change the effect of the flags that it will have on the benchmark.

c.) We can write the linpack benchmark using parallel threads as the most of the operations are matrix multiplication which we can perform but we have to use mutex to preserve the shared variable. For each row of the matrix we can create a thread and then join it after the computation. While in the case of drystone, I dont think we can do it as there more serial operations taking place which cannot be parallelized.
In Whetstone if we create a new thread for each Loop then we can be able to parallelize the benchmark.

Part 2

a.)

Time for running with no. Of threads			
1	2	4	8
116.327	65.7538	55.7047	31.6936
113.347	62.3987	47.3398	32.7639
113.246	62.1108	46.3268	33.9211
62.8773	62.9894	56.5985	35.2764
81.6515	61.3994	57.6483	37.8906
57.2803	60.4831	56.1729	25.8241
77.4265	61.129	59.781	20.8678
108.179	64.9066	50.6819	21.0438
61.4429	60.3392	37.118	32.1698
93.7997	61.2909	40.7285	29.8991

Figure 4: Running time of the Merge sort.

```
82 97885 97901 97905 97905 97918 97926 97929 97941 97946 97947 97950 97950 97993 98011 98021 98028 98042 98062 98063 98081 98089 98105 98109 98110 98115 9
8120 98128 98131 98144 98168 98180 98188 98199 98201 98202 98210 98219 98222 98223 98230 98233 98236 98245 98247 98253 98254 98261 98261 98265 98283 98292
 98299 98307 98317 98318 98327 98328 98334 98336 98343 98344 98369 98371 98378 98388 98397 98419 98441 98467 98476 98483 98491 98497 98505 98507 98518 985
32 98560 98567 98583 98604 98608 98632 98645 98646 98655 98664 98666 98676 98686 98698 98713 98734 98739 98753 98757 98762 98773 98774 98778 98782 98795 9
8797 98805 98806 98814 98835 98855 98867 98876 98877 98893 98897 98929 98952 98967 98994 98995 99000 99015 99018 99028 99033 99037 99044 99046 99048 99065
 99077 99077 99093 99093 99094 99112 99125 99135 99136 99143 99151 99157 99176 99187 99197 99206 99215 99224 99224 99227 99234 99274 99289 99321 99330 993
47 99351 99352 99378 99385 99397 99403 99410 99412 99413 99421 99429 99430 99469 99469 99478 99481 99487 99521 99525 99531 99535 99547 99570 99574 99579 9
9585 99586 99587 99605 99615 99616 99631 99638 99654 99674 99688 99692 99704 99714 99725 99726 99727 99740 99749 99757 99757 99758 99774 99778 99782 99821
 99838 99864 99877 99914 99926 99937 99940 99952 99956 99961 99964 99986 99993 99998
Time taken: 38.2003
```

Figure 5: Output of the merge sort

The code was taken from online [1]. There is a sort checking function which checks whether the elements are arranged in proper order or not. If they are not it prints an error.

As we are increasing the threads the time is decreasing. This is the expected behaviour from the program. The performance is increasing as we are increasing the number of threads. The timings are in microseconds.

b.) The main problem that I encounter was joining the thread and then merging into one. In my case, as I was using merge sort to run parallelly. I faced problem to merge the eight 8 to perform the sort.

c.) The program shows a strong scaling because as we are increasing the number of threads by keeping the sized fixed. But then when we are merging after joining the threads it shows sort of weak scaling. So it is kind of both strong as well as weak scaling.

Part 3

a.) CPU Model: Intel(R) Xeon(R) Gold 6132

b.) Cache Hierarchy

- **L1 Instruction Cache**:
  i. **Size:** 32768 kB
  ii. **Associativity**: 8
- **L1 Data Cache:**
  i. **Size:** 32768
  ii. **Associativity**: 8
- **L2 Cache:**
  i. **Size:** 1048576
  ii. **Associativity:** 16
- **L3 Cache:**
  i. **Size:** 20185088
  ii. **Associativity:** 11

c.) Bandwidth of Interconnect: 100Gbps

Latency of the Interconnect: 600ns

d.) Version of Linux: Centos Linux 7

Part 4

1. Summit – IBM Power System AC922
    - Summit was launched in 2018.
    - It has a hybrid architecture where each node contains 2 IBM Power 9 processor and 6 NVIDIA Volta V100.
    - These processors and GPUs are connected together with NVIDIA high speed link which is NVLink.
    - It has 4608 nodes with the node performance of 42TF.
    - Each node has 512 Gb of DDR4 RAM and another 96GB of HBM2(High bandwidth) which is addressable by all the CPUs and GPUs.
    - The nodes are connected in a non-blocking fat tree using dual-rail Mellanox EDR InfiBand interconnect.
    - It has a peak power consumption of 13 MW.

2. Sierra – IBM Power System AC922
    - Sierra uses same CPU, GPUs and inter connects same as that of Summit
    - It has smaller number of CPU and GPUs per node compared to Summit.
    - It produces a maximum performance of 94640 Tflop/s with total 1572480 cores.
    - It requires power of 7438kW

3. Sunway TaihuLight
    - This super computer was developed in China. There is no proper documentation avialable on the website.
    - It employs Sunway SW26010 260C as the processor and interconnect of Sunway.
    - It has maximum performance of 93014.6 Tflop/s with power consumption of 15371 kW.
    - It has total cores of 10649600 and memory of 1310720 GB.

4. Tianhe-2A
    - This supercomputer has 4,981,760 cores with total memory of 2,277,376 GB.
    - It uses Intel Xeon E5-2692v2 12C with TH Express-2 interconnect.
    - It can achieve maximum performance of 61,44.5 Tflop/s
    - Uses about 18,482 kW power.

5. Frontera
    - This supercomputer has 448,448 cores with total memory of 1,537,536 GB.
    - It uses Intel Xeon Platinum 8280 28C with Mellanox InfiniBand HDR interconnect.
    - It can achieve maximum performance of 23,516.4 Tflop/s

6. Piz Daint
    - This supercomputer has 387,872 cores with total memory of 365,056 GB.
    - It uses Intel Xeon E5-2690v3 12C with Aries interconnect .
    - It can achieve maximum performance of 21230 Tflop/s
    - Uses about 2384kW power.

7. Trinity
    - This supercomputer has 979,072 cores with undocumented memory.
    - It uses Intel Intel Xeon Phi 7250 68C with Aries interconnect .
    - It can achieve maximum performance of 20,158.7 .7 Tflop/s

8. Al Bridging Cloud Infrastructure(ABCI)
    - This supercomputer has 448,448 cores with total memory of 1,537,536 GB.
    - It uses Intel Xeon Platinum 8280 28C with Mellanox InfiniBand HDR interconnect.
    - It can achieve maximum performance of 23,516.4 Tflop/s

9. SuperMUC-NG
- This supercomputer has 305,856 cores with total memory of 718,848 GB.
- It uses Intel Xeon Platinum 8174 24C with Intel Omni-Path interconnect.
- It can achieve maximum performance of 19,476.6 Tflop/s

10. Lassen
- This supercomputer has 288288 cores with total memory of 253,440 GB.
- It uses IBM POWER9 22C with Dual-rail Mellanox EDR Infiniband interconnect.
- It can achieve maximum performance of 18,200 Tflop/s

Summary of trends

By studying the above systems we can see that as we increase the number of cores we can see increase in the performance in the system. This stays true for the first two supercomputers, but in the 3rd and 4th we can see that though there is increase in the number of cores the performance is not improving. This could be because the CPU used for the first two are similar while the rest are different. There is not documented GPU in the 3rd and 4th supercomputer. By studying these system Power PC 9 is supposed to be the superior CPU among its competition, while NVIDIA Volta is the best among the GPUs. The combination of PowerPC 9 and NVIDIA is supposed to outperform all the other super computers. Summit and Sierra are having same configuration except for the number of CPU per node or cores per node. As the trend would suggest that better performance is obtained from the RISC processor as Power PC 9 is based on RISC. Other than the first two most them as using Intel processor which is based on CISC. Through Sunway TaihuLight we can see that increasing the cores would not increase the performance. In future most of the processor you be moving to RISC architecture.

Supercomputer Architecture

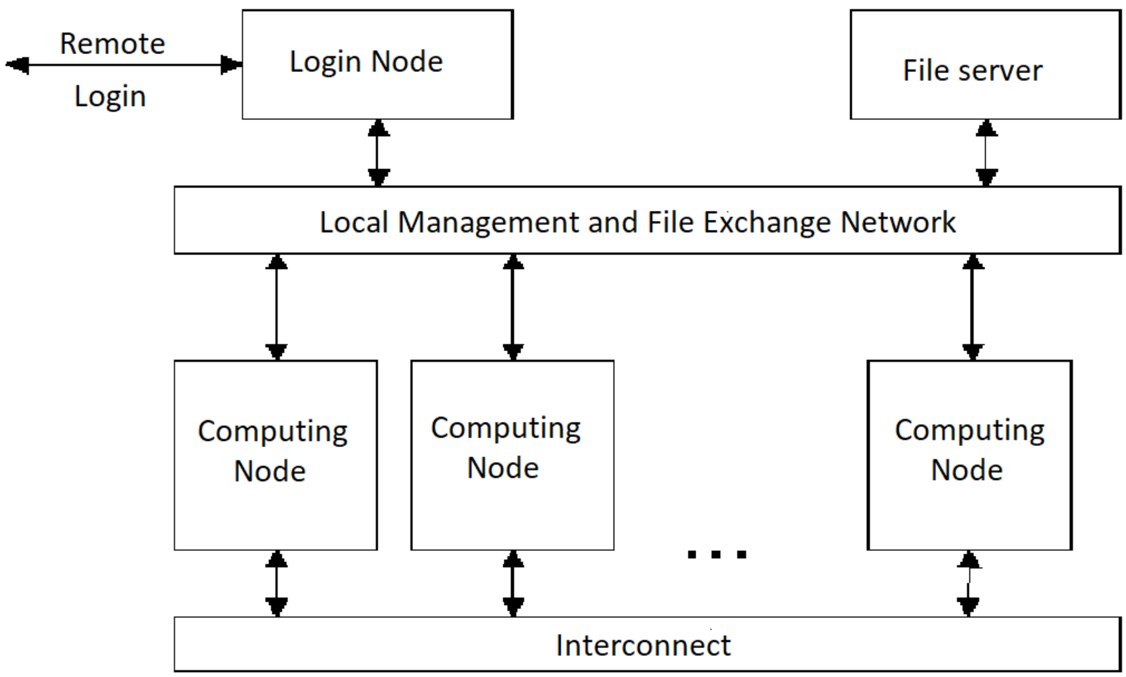

Figure 5: Proposed Architecture of supercomputer

The architecture of the supercomputer will be as shown above. The file server will also serve as a storage for the computing nodes like a hard disk which will be shared among all the nodes.

Each computing node will have its own CPUs and GPUs with its own memory. There would be a L4 cache which would be shared between CPUs and GPUs with the three cache in the CPU. There would a 250GB HBM2 for the GPU and 1TB DDR4 for the CPU for each node. I would employ a RISC based CPU with around 16 cores per CPU and each Node having around 4 CPU and 16 GPUs per node. In total there would be 5000 nodes with total of 320000 cores in total. There would be 25 PB of memory. An interconnect with the bandwidth of 1Tbps and latency of 500ns. Considering the present market products, the same specs that are used in Summit would be used. The operating system would RHEL which ever newer version would be available in the market.
The major highlight would be introduction of L4 cache and a powerful RISC processor.

References:

1. https://www.geeksforgeeks.org/merge-sort-using-multi-threading/
2. https://www.top500.org/lists/2019/11/
3. https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html