Ziting Liu 002981429 EECE5640 HW1

Problem 1 a :

The CPU model is Intel Xeon E5620 working at 2.40 GHz with the capability to overclock to 2.66 GHz. 16 CPUs are available to me. Memory size is 16033 mb. System version: 3.10.0-693.17.1.el7.x86_64.

The first benchmark program was taken from www.netlib.org/benchmark/linpackc.new. Which is a double precision benchmarking program that calculates FLOPS. With an input array size of 200 (max input size), the program did 35.06 sec, 35.80 sec, 35.83, 35.87 sec, 35.88. With an average runtime of 35.68 sec.

The second benchmark program was the same as the first one, except I modified the program to be a single precision floating point calculation. It did 37.49 sec, 37.40 sec, 37.36 sec, 37.37 sec, 37.36 sec. With an average runtime of 37.40 sec.

The third benchmark program was taken from

https://people.sc.fsu.edu/~jburkardt/c_src/memory_test/memory_test.html. This is a program that tests the memory performance of the machine. It declares and uses a very large array to test what are the memory limitations of the machine. It took 22.57 sec, 20.19 sec, 20.20 sec, 20.18 sec, 20.42 sec. With an average of 20.31 sec over five runs.

None of the execution time above is considerably faster than the other ones.

1b. compiler optimizations

-msse: This flag enables the SSE instruction sets for x86 and x86-64 architectures. Primarily in floating point intensive computing tasks. This flag was able to speed up the first benchmark test from 35.8 sec to 27.5 sec, and the second benchmark from 37.4 sec to 27.3 sec. Which is a great improvement. However, this flag slowed the third benchmark down from 20.3 sec to 24 sec.

-O3: This flag is the highest level of optimization possible. It has little effect on the single precision FLOPS, but it could speed up the double precision FLOPS from 35.68 sec to 23.77 sec. This flag was able to speed up the memory intensive test as well, from 20.3 sec to 17 sec.

1c. Speedup Benchmark

For the first and second benchmark tests, I discovered that the program is executing functions “matgen” and “dgefa” many times sequentially. Using pthreads and letting the function run parallel would be one way to obtain additional speedup.

However, for the third benchmark program, the memory intensive one. All the code seems to be running sequentially. I could not figure out how to speed up the program by parallelism.

Problem 2 a:

The sorting algorithm I went for is the merge sort. Merge sort code taken from https://www.geeksforgeeks.org/merge-sort/. Because this is an algorithm that does multiple parts concurrently, it makes sense to multi-thread this program. Time below was the average running time over 5 runs. https://www.geeksforgeeks.org/merge-sort-using-multi-threading/ showed me the general idea of how to turn a merge sort into a 4 threaded program. I followed

the idea and made modifications to my code, and turned it into a variable-number-threaded program.

As a **1-thread program**, it will sort 5000 random integers in 778 microseconds.

As a **2-thread program**, it will sort 5000 random integers in 622.8 microseconds.

As a **4-thread program**, it will sort 5000 random integers in 504.4 microseconds.

As an **8-thread program**, it will sort 5000 random integers in 526.2 microseconds.

From the running time I recorded above, we can see that more threads will speed up the sorting process, until it goes above 4 threads. I presume that the reason for this is that: creating threads costs time and it's just not worth it for lightweight sorting tasks. Sorting 5000 integers is not a heavy task, therefore 4 threads is the balance point for number of threads and performance.

2. b

To be honest, I did not have too many difficulties in turning it into a multi-threaded program, because I had experience with threads and mutex in undergrad. The only difficulty I faced was to figure out how to merge all the sorted sublists returned by all the threads at once, then I had a segmentation fault. Other than that, I believe everything went smoothly.

2. c

Strong scaling: A speed up of 48% was gained going from 1 to 4 threads. According to Amdahl's law of strong scaling: $1.48 = 1 / (s + p / 4)$. Therefore, we can work out that 57% of the sorting algorithm is serial, and only 43% of the work is parallel.

Weak scaling: According to Gustafson's law of weak scaling: $1.48 = s + p*4$. We can conclude that 84% of the work is in serial and only 16% of the work is parallel.

Problem 3

a. CPU model: Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz

b. L1d cache: 32K

L1i cache: 32K

L2 cache: 1024K

L3 cache: 14080K

c. Ethernet connection: ether a4:1f:72:19:70:e9 txqueuelen 1000 (Ethernet)

Download speed 213 Mbps

Upload speed 175.33 Mbps

Problem 4.

For the top 10 supercomputers in the top 500 list, it's not hard to see that more core, higher frequency, and higher power is always going to make supercomputers faster. Computers in the list all feature at least one, if not all three, of the features mentioned above. Although we have been told in class that adding more cores is not the easy way out, but it does seem like the case in the list. No.1 in the list has triple the cores than No.2, as well as triple the power. The result is that **Fugaku** has triple the computing power than **Summit.**

The other trend I noticed is that supercomputers in the US were able to do the same job with way less number of cores and power. I believe this is due to the gap in chip manufacturing ability between the US and the rest of the world. This is something that needs time to catch up. Before that, I guess other supercomputers has to add more cores into the system.

If I get to work on a supercomputer in the future, I imagine it will be like this.

First, the CPU frequency is going to be at least 3.0GHz, otherwise the system is going to be needing way too many cores. Therefore the power consumption is not going to look good. If the computer was going to be featured in this least 10 years from now, I believe that it has to do 800,000 for Rmax and 1,000,000 in Rpeak. Considering how fast these computers are growing.

For the interconnect, I discovered that the top 3 computers are all using something called a ‘dual-rail interconnect’. I believe that this is going to be essential in the future for supercomputers, or even something like a tri-rail.

When it comes to memory, I still believe the more memory the better. As we can see from this ranking, higher in ranking basically equals more memory available. I believe in the future, this number is going to be at least 8,000,000 GB.

I can’t predict what the power consumption is going to be. Because as the feature size of semiconductors is getting smaller and smaller, the power consumption can grow exponentially. Another factor is that countries around the world are all trying to be as “green” as possible. So I don’t know if such a power consuming beast is going to be allowed in the future. Maybe the Green500 list is what we should be looking at when I’m working on supercomputers.

Porblem 5:

In the Green 500 list, power efficiency is the most important factor. The main difference is that the peak performance does not matter anymore in this list. It can be very slow, but as long as it draws little power, it is considered as a good supercomputer. None of the computers in this top 10 list exceeds 3.0GHz. And the number of cores are relatively low compared to the regular top500 list. The computer that impressed me is the Perlmutter, it’s 6th place in Green 500 while being 5th in the regular top 500. I believe this is the kind of computer that we need to focus on.