ECE574: Cluster Computing
Spring 2019 Midterm Exam

Name: Spencer Comstock

This page intentionally blank.

Q1: 20/20
Q2: 20/20
Q3: 19/20
Q4: 20/20
Q5: 20/20

94/100
+5
99.5

20/20

1. Performance Measurement

You run an OpenMP Sobel convolution on the `space_station_hires.jpg` picture on a Raspberry Pi 3B+ device and get the following results:

Single Thread
```
real 12.00s
user 11.80s
sys   0.23s
```

Two Threads
```
real  6.00s
user 12.00s
sys   0.20s
```

Four Threads
```
real  4.00s
user 12.90s
sys   0.27s
```

Eight Threads
```
real  5.30s
user 14.70s
sys   1.40s
```

(a) What is the speedup and parallel efficiency going from one to two threads?

$S_p = \frac{12.00}{6.00} = 2$

$S_p = 2$

$E_p = \frac{S_p}{p} = \frac{2}{2} = 1$

$E_p = 1$

(b) What is the speedup and parallel efficiency going from one to four threads?

$S_p = \frac{12.00}{4.00} = 3.$

$S_p = 3$

$E_p = \frac{S_p}{p} = \frac{3}{4}$

$E_p = 0.75$

(c) Why is the ~~user time sometimes higher~~ than the real time?

The user time is sometimes higher because it takes the time that it took on each processor and totals them all up. So, as you increase the number of threads, the user time is most likely going to be higher than the real.

(d) Do we have enough data to know if the program exhibits weak scaling?

No, this is because the data size is not changing. The space_station_hires.jpg is constant. For it to be weak scaling both the processes and data size must change.

(e) Do we have enough data to know if the program exhibits strong scaling?

Yes, this is because the data size of the space_station_hires jpg stays the same. To be strong scaling, the data size must stay the same and the processes must change, which it does.

(f) Why might performance go down when moving from four to eight threads?

The performance might go down because of the processor bounds. At some point, the more you increase the processes it will eventually get worse.

20/20

2. Parallel Hardware

(a) Which of the following statements is true of a shared-memory system? Circle all that apply.

i. One shared memory address space for all processors.

ii. All processors share one operating system image.

iii. The fastest way for threads to communicate is by passing messages over a network.

(b) You have a chunk of C code that looks like this:

```
int A[4096][4096],B[4096][4096],C[4096][4096],x,y;

int main(int argc, char **argv) {

        for(y=0;y<4096;y++) {
                for(x=0;x<4096;x++) {
                        A[x][y]=B[x][y]+C[x][y];
                }
        }
}
```

You change the above code so the loops look like this:

```
        for(x=0;x<4096;x++) {
                for(y=0;y<4096;y++) {
                        A[x][y]=B[x][y]+C[x][y];
                }
        }
```

and the code ends up running eight times as fast.

What is the most likely cause of this speedup, and why does it happen?

This speedup is caused by loop changing. Before, it was probably accessing memory out of order. After, the memory was most likely accessed more in order so there was less cache misses. This probably caused it to speed up.

19/20

3. Pthread Programming

```
#define NUM_THREADS 16384

int bytes_initialized=0;
int memory[NUM_THREADS];

void *init_memory(void *data) {

    int t=(int)data;

    memory[t]=t;

    bytes_initialized++;
}

int main(int argc, char **argv) {

    pthread_t threads[NUM_THREADS];
    int t;

    for(t=0; t<NUM_THREADS; t++) {
        pthread_create(&threads[t], NULL,
                init_memory, (void *)t);
    }

    for(t=0; t<NUM_THREADS; t++) pthread_join(threads[t],NULL);

    printf("Bytes initialized: %d\n",bytes_initialized);
}
```

(a) In the above code, can you say what value will be printed at the end for the `bytes_initialized` value? Why? No, this is because the processes could try to increment the bytes_initialized number at the same time and it not might increment properly. It could possibly only increment once.

(b) How could you change the code so that `bytes_initialized` has the expected value at the end of the program? You could add locks into the code so that each thread locks, adds 1 to the number, and then unlocks, while the others wait for the lock to be unlocked.

(c) If you run the above program as-is on a machine with 8 cores, is it likely to run faster or slower than the equivalent serial code? Why?

It is most likely to run slower. This is because with 8 cores, there is more work it must do to initialize everything. Also, init memory isn't that long of a process, so it probably won't help much to have

-1 Also not 16K threads

20/20

4. OpenMP Programming

(a) Given the below OpenMP code, what is the maximum speedup you would expect to get out of the code (assuming you had an arbitrarily large number of cores available)? Why?

I would assume to get a maximum speedup of 2. This is because there are only two # pragma omp section. This causes it to have one run on the first convolve and the other run on the second. No matter how many cores you have, only two sections need to be run so 2 is the speedup.

```
#pragma omp parallel sections
{

#pragma omp section
{
        generic_convolve((void *)&convolve_data[0]);
}
#pragma omp section
{
        generic_convolve((void *)&convolve_data[1]);
}}
```

(b) In the below code, the various iterations of the `do_some_work()` function all take approximately the same about of time. Do you think changing the scheduling from static to dynamic would improve the runtime? Explain your answer.

No, this is because all do_some_work() functions take the same amount of time. The point of using dynamic is so that if one thread finishes what it is doing it doesn't have to wait and can work on another. But since the do_some_function takes the same amount of time every time, the threads should all finish at the same time and won't be waiting.

```
#pragma omp parallel private(i)

#pragma omp for schedule(static)
    for (i=0; i < 1024; i++) {
        do_some_work(i);
    }
```

20/20

5. Distributed Systems / MPI

(a) You have a cluster made out of 24 Raspberry Pi2 systems, connected by 100MB Ethernet. Each Raspberry Pi2 node has 4 cores and 1GB of memory.

i. You have a workload that only scales to 4 threads and fits in 800MB of RAM. You plan to run it on the Pi cluster mentioned above. Would it make more sense to code this in OpenMP or MPI? Why? I would suggest OpenMP since you only need 1 Raspberry Pi2 to run it on. You don't really need the other ones, so OpenMP is your best bet.

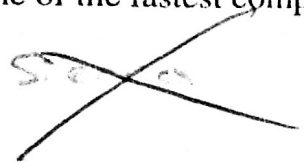
ii. You have a workload that scales linearly up to 128 cores and requires 16GB of RAM. You plan to run it on the Pi cluster mentioned above. Would it make more sense to code this in OpenMP or MPI? Why? MPI, this is because you want to use multiple pi's and you want to use MPI to use the other pi's. This is because the workload should be split up across the system. Message passing is your best option here.

(b) In the below code from HW#6 , we use MPI_Bcast() to send image data from rank#0 to all other ranks. Did we need to do this step on previous homeworks? Why was this only necessary with MPI? No, this is because each process already had a copy of image. Here, each machine needs to get a copy of the image, so the image is broadcasted from the master and received by the rest.

```
result = MPI_Bcast(image.pixels,                    /* buffer */
                   image.x*image.y*image.depth,     /* count */
                   MPI_CHAR,                        /* type */
                   0,                               /* root source */
                   MPI_COMM_WORLD);
```

6. **Bonus Question**

(a) What was the name of the fastest computer on the November 2018 Top500 list?



(b) What country was it located in?

The United States of America

~~LLNL~~ ORNL

UIS