CPSC 8200 Project 1 Report

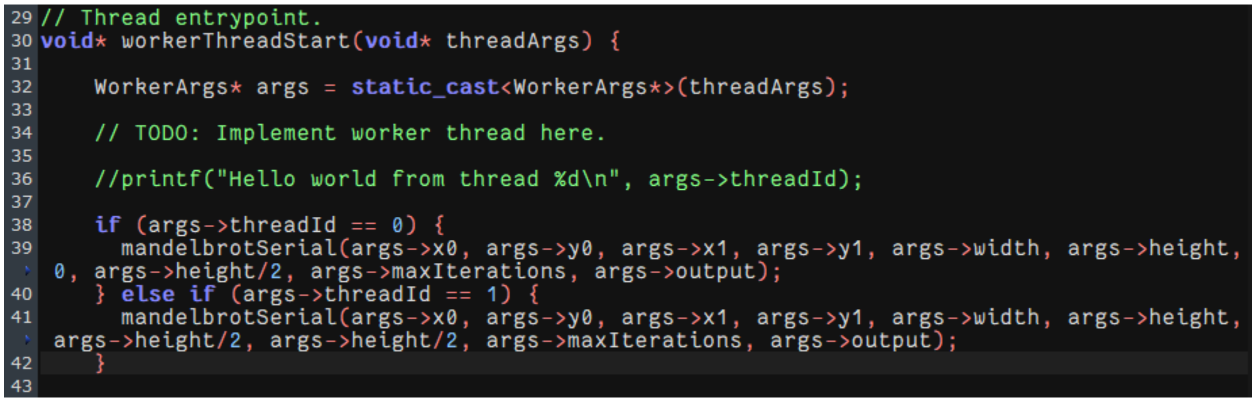
John Pascoe
jepasco@clemson.edu

1: Introduction

This writeup delves into the use of pthreads to parallelize the Mandelbrot set generation by analyzing the performance improvements achieved by distributing the workload across multiple threads. This writeup also investigates the utilization of BLAS saxpy operations enhanced with SIMD vectorization, to accelerate computation. Furthermore, it dives into ISPC, a specialized programming language for SIMD architectures, to evaluate its effectiveness in parallelizing Mandelbrot set generation. It also explores the impact of introducing task-based concurrency in ISPC to further enhance parallelism. Through a combination of benchmarking, profiling, and performance measurement, this writeup examines the speedup achieved by each parallelization technique and identifies the factors contributing to their effectiveness or limitations. The writeup presents a comprehensive analysis of the observed results, shedding light on the trade-offs and challenges associated with different approaches to parallel computing.

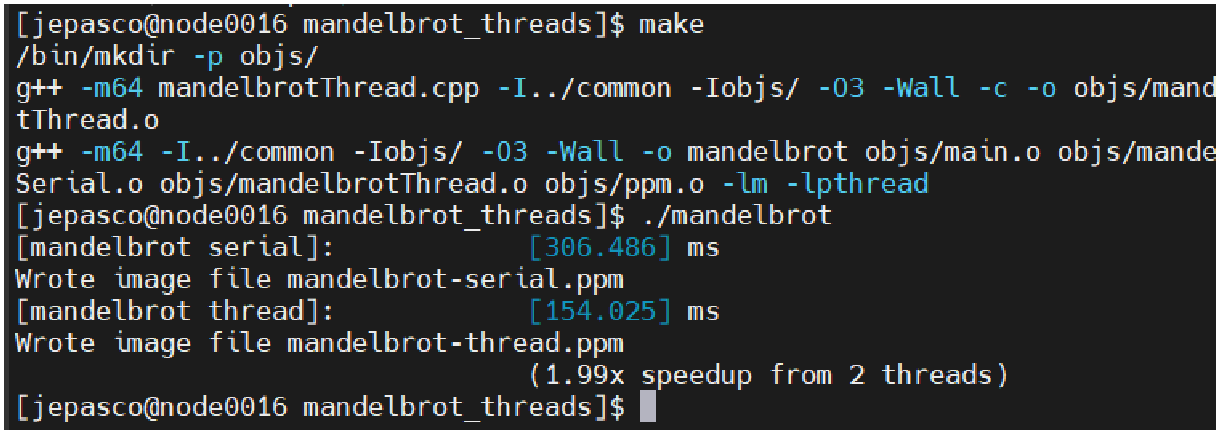
2: Results

Part 1: Parallel Fractal Generation

1. 

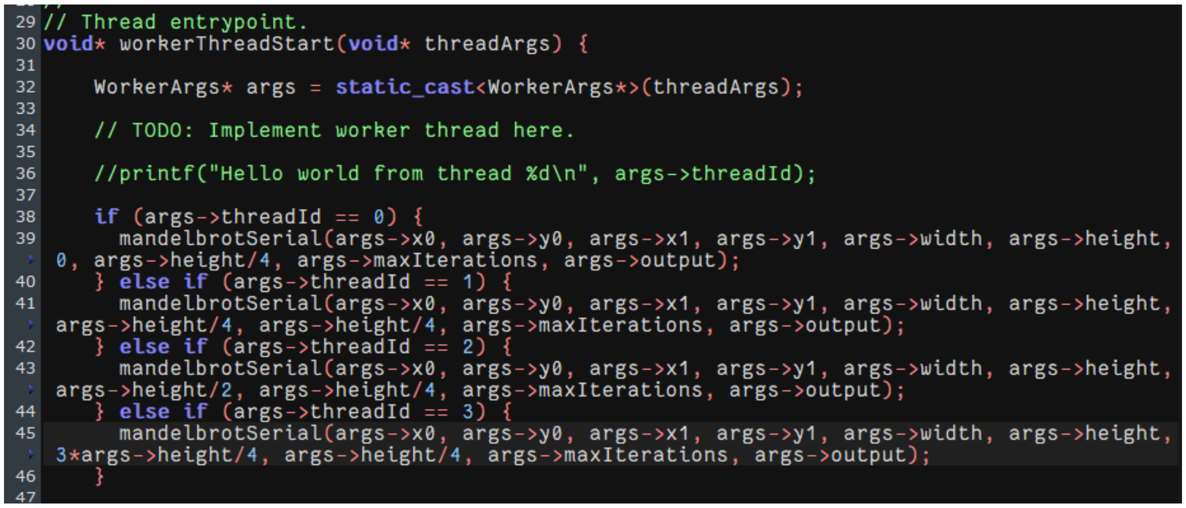
```
// Thread entrypoint.
void* workerThreadStart(void* threadArgs) {

    WorkerArgs* args = static_cast<WorkerArgs*>(threadArgs);

    // TODO: Implement worker thread here.

    //printf("Hello world from thread %d\n", args->threadId);

    if (args->threadId == 0) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height,
 0, args->height/2, args->maxIterations, args->output);
    } else if (args->threadId == 1) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height,
 args->height/2, args->height/2, args->maxIterations, args->output);
    }

```

   a. This is the code for using spatial decomposition to break down the mandelbrot generation with 2 threads.

2. 

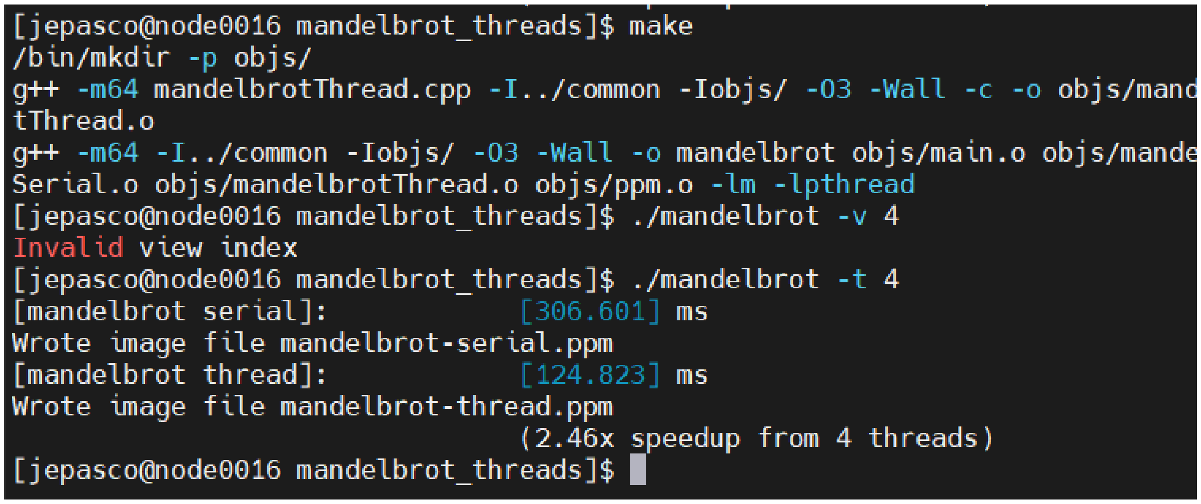
```
[jepasco@node0016 mandelbrot_threads]$ make
/bin/mkdir -p objs/
g++ -m64 mandelbrotThread.cpp -I../common -Iobjs/ -O3 -Wall -c -o objs/mand
tThread.o
g++ -m64 -I../common -Iobjs/ -O3 -Wall -o mandelbrot objs/main.o objs/mande
Serial.o objs/mandelbrotThread.o objs/ppm.o -lm -lpthread
[jepasco@node0016 mandelbrot_threads]$ ./mandelbrot
[mandelbrot serial]:            [306.486] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot thread]:            [154.025] ms
Wrote image file mandelbrot-thread.ppm
                                (1.99x speedup from 2 threads)
[jepasco@node0016 mandelbrot_threads]$
```

   a. This is the output for the speedup from utilizing 2 threads to break down the mandelbrot generation.

3. 

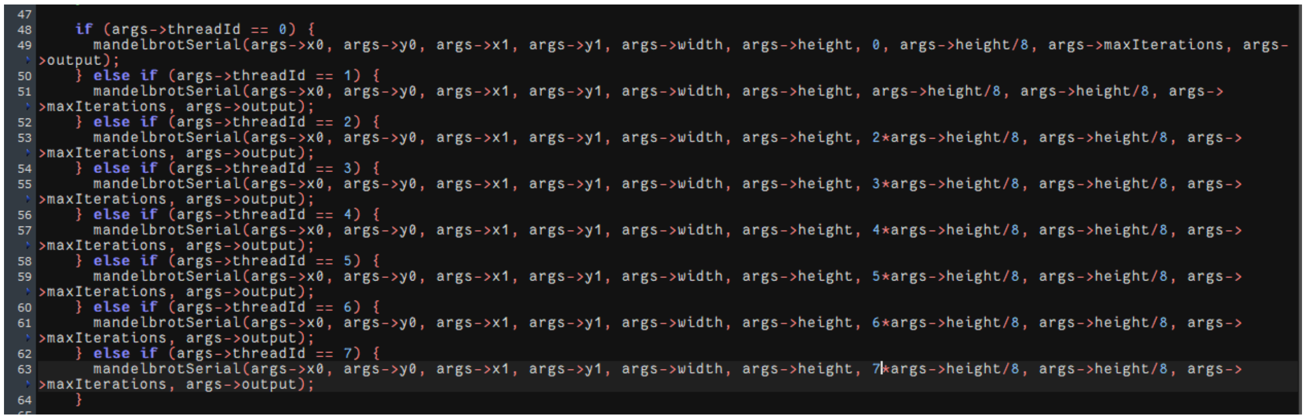
```
// Thread entrypoint.
void* workerThreadStart(void* threadArgs) {

    WorkerArgs* args = static_cast<WorkerArgs*>(threadArgs);

    // TODO: Implement worker thread here.

    //printf("Hello world from thread %d\n", args->threadId);

    if (args->threadId == 0) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height,
 0, args->height/4, args->maxIterations, args->output);
    } else if (args->threadId == 1) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height,
 args->height/4, args->height/4, args->maxIterations, args->output);
    } else if (args->threadId == 2) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height,
 args->height/2, args->height/4, args->maxIterations, args->output);
    } else if (args->threadId == 3) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height,
 3*args->height/4, args->height/4, args->maxIterations, args->output);
    }
```

   a. This is the code for using spatial decomposition to break down the mandelbrot generation with 4 threads.

4. 

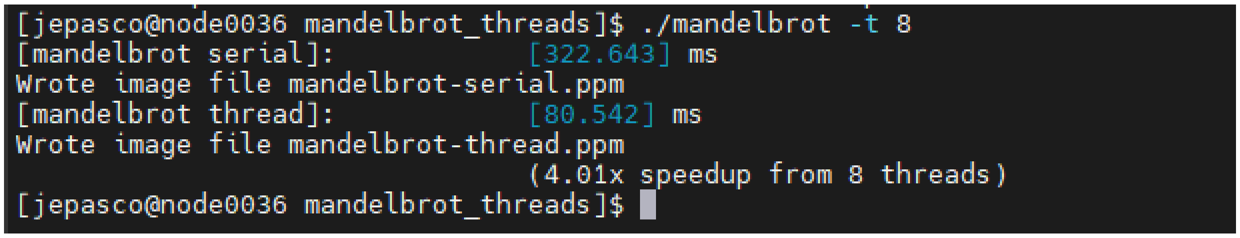
```
[jepasco@node0016 mandelbrot_threads]$ make
/bin/mkdir -p objs/
g++ -m64 mandelbrotThread.cpp -I../common -Iobjs/ -O3 -Wall -c -o objs/mand
tThread.o
g++ -m64 -I../common -Iobjs/ -O3 -Wall -o mandelbrot objs/main.o objs/mande
Serial.o objs/mandelbrotThread.o objs/ppm.o -lm -lpthread
[jepasco@node0016 mandelbrot_threads]$ ./mandelbrot -v 4
Invalid view index
[jepasco@node0016 mandelbrot_threads]$ ./mandelbrot -t 4
[mandelbrot serial]:            [306.601] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot thread]:            [124.823] ms
Wrote image file mandelbrot-thread.ppm
                                (2.46x speedup from 4 threads)
[jepasco@node0016 mandelbrot_threads]$
```

   a. This is the output for the speedup from utilizing 4 threads to break down the mandelbot generation.

5. 

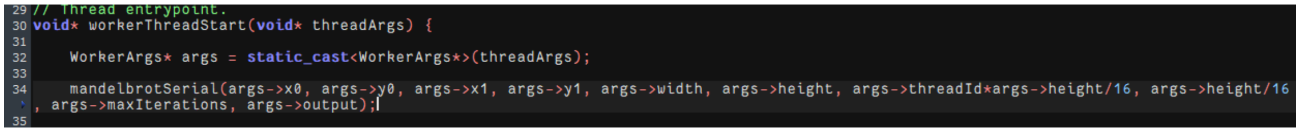
```
    if (args->threadId == 0) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 0, args->height/8, args->maxIterations, args-
>output);
    } else if (args->threadId == 1) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, args->height/8, args->height/8, args->
>maxIterations, args->output);
    } else if (args->threadId == 2) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 2*args->height/8, args->height/8, args->
>maxIterations, args->output);
    } else if (args->threadId == 3) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 3*args->height/8, args->height/8, args->
>maxIterations, args->output);
    } else if (args->threadId == 4) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 4*args->height/8, args->height/8, args->
>maxIterations, args->output);
    } else if (args->threadId == 5) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 5*args->height/8, args->height/8, args->
>maxIterations, args->output);
    } else if (args->threadId == 6) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 6*args->height/8, args->height/8, args->
>maxIterations, args->output);
    } else if (args->threadId == 7) {
      mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, 7*args->height/8, args->height/8, args->
>maxIterations, args->output);
    }
```

   a. This is the code for using spatial decomposition to break down the mandelbrot generation with 8 threads.

6. 

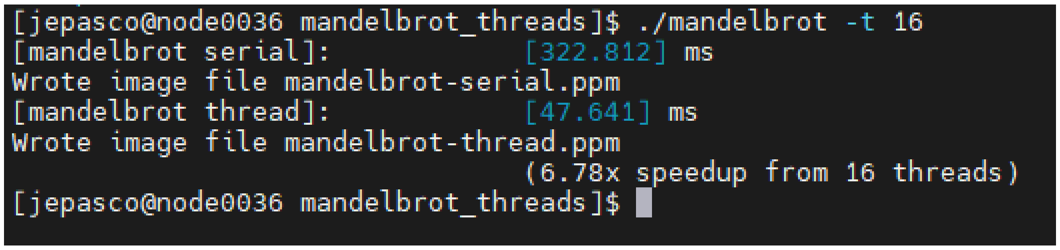
```
[jepasco@node0036 mandelbrot_threads]$ ./mandelbrot -t 8
[mandelbrot serial]:                [322.643] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot thread]:                [80.542] ms
Wrote image file mandelbrot-thread.ppm
                                    (4.01x speedup from 8 threads)
[jepasco@node0036 mandelbrot_threads]$
```

   a. This is the output for the speedup from utilizing 8 threads to break down the mandelbrot generation.

7. 

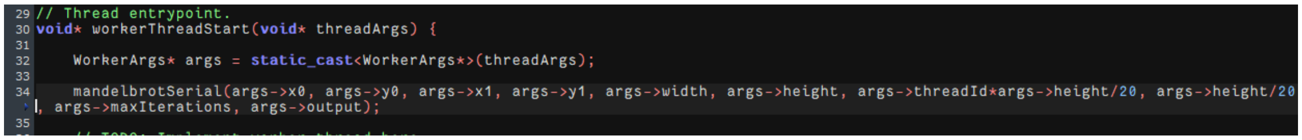
```
// Thread entrypoint.
void* workerThreadStart(void* threadArgs) {

    WorkerArgs* args = static_cast<WorkerArgs*>(threadArgs);

    mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, args->threadId*args->height/16, args->height/16
, args->maxIterations, args->output);
```

   a. This is the code for using spatial decomposition to break down the mandelbrot generation with 16 threads.

8. 

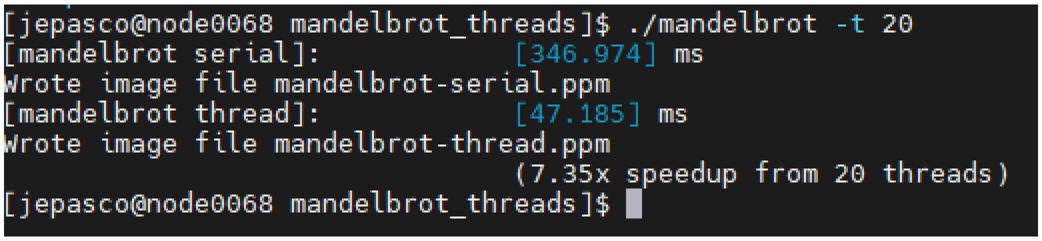
```
[jepasco@node0036 mandelbrot_threads]$ ./mandelbrot -t 16
[mandelbrot serial]:                [322.812] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot thread]:                [47.641] ms
Wrote image file mandelbrot-thread.ppm
                                    (6.78x speedup from 16 threads)
[jepasco@node0036 mandelbrot_threads]$
```

   a. This is the output for the speedup from utilizing 16 threads to break down the mandelbrot generation.

9. 

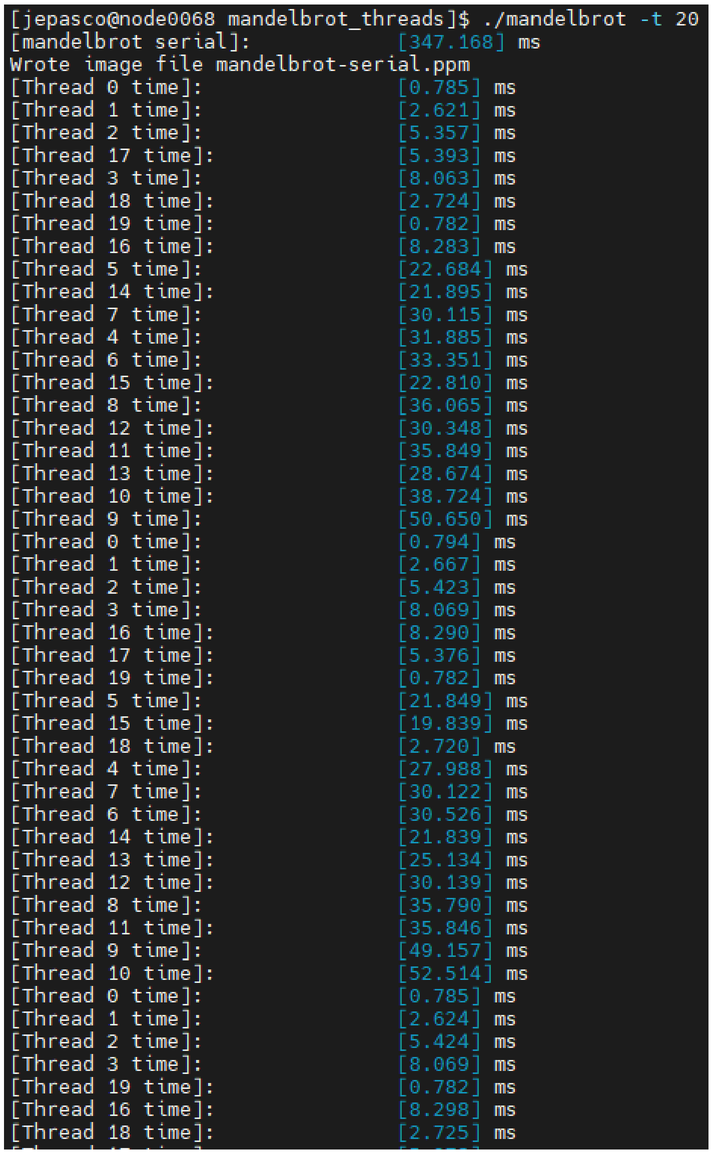
```
// Thread entrypoint.
void* workerThreadStart(void* threadArgs) {

    WorkerArgs* args = static_cast<WorkerArgs*>(threadArgs);

    mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, args->threadId*args->height/20, args->height/20
, args->maxIterations, args->output);
```

   a. This is the code for using spatial decomposition to break down the mandelbrot generation with 20 threads.

10. 

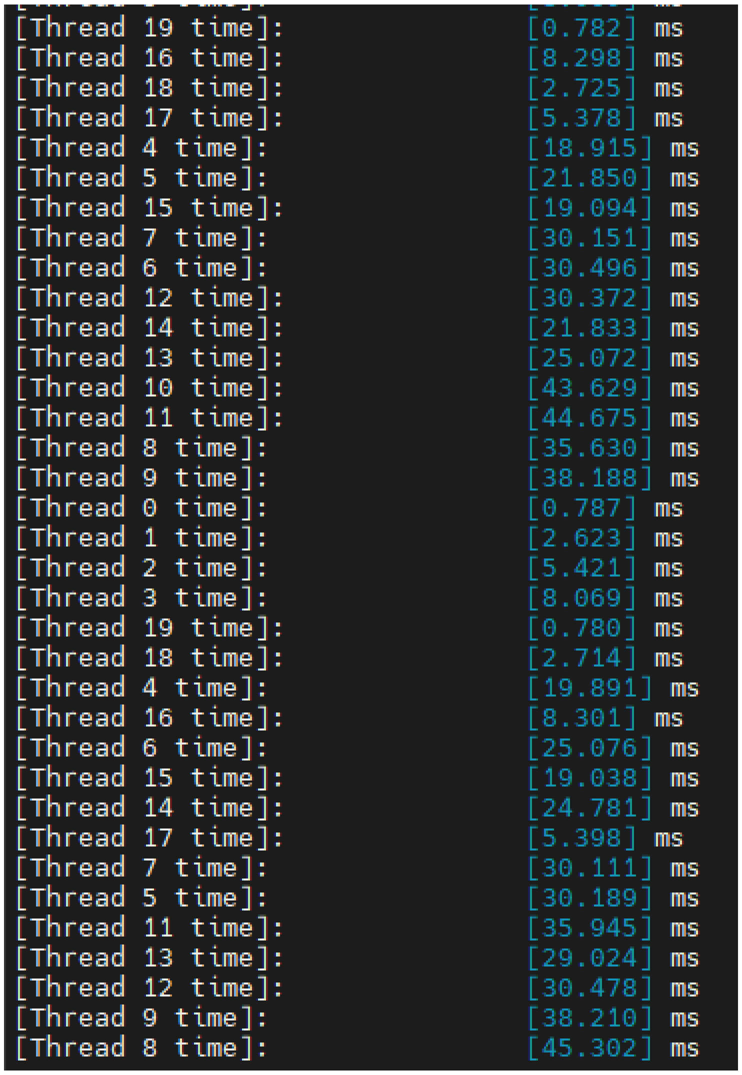
```
[jepasco@node0068 mandelbrot_threads]$ ./mandelbrot -t 20
[mandelbrot serial]:                [346.974] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot thread]:                [47.185] ms
Wrote image file mandelbrot-thread.ppm
                                    (7.35x speedup from 20 threads)
[jepasco@node0068 mandelbrot_threads]$
```

   a. This is the output for the speedup from utilizing 20 threads to break down the mandelbrot generation.

11. 

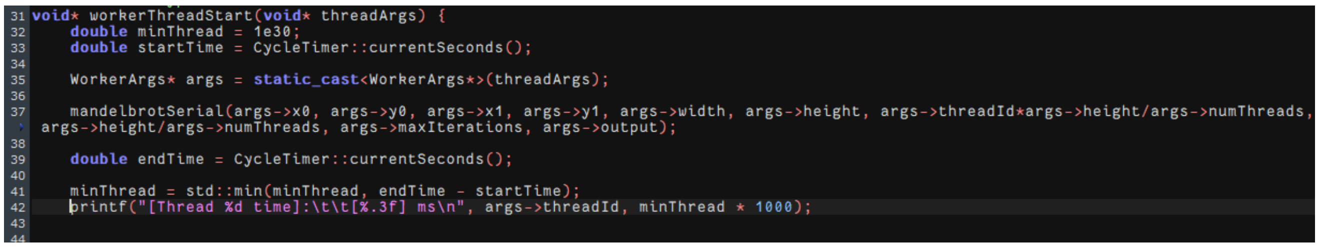
```
[jepasco@node0068 mandelbrot_threads]$ ./mandelbrot -t 20
[mandelbrot serial]:            [347.168] ms
Wrote image file mandelbrot-serial.ppm
[Thread 0 time]:                [0.785] ms
[Thread 1 time]:                [2.621] ms
[Thread 2 time]:                [5.357] ms
[Thread 17 time]:               [5.393] ms
[Thread 3 time]:                [8.063] ms
[Thread 18 time]:               [2.724] ms
[Thread 19 time]:               [0.782] ms
[Thread 16 time]:               [8.283] ms
[Thread 5 time]:                [22.684] ms
[Thread 14 time]:               [21.895] ms
[Thread 7 time]:                [30.115] ms
[Thread 4 time]:                [31.885] ms
[Thread 6 time]:                [33.351] ms
[Thread 15 time]:               [22.810] ms
[Thread 8 time]:                [36.065] ms
[Thread 12 time]:               [30.348] ms
[Thread 11 time]:               [35.849] ms
[Thread 13 time]:               [28.674] ms
[Thread 10 time]:               [38.724] ms
[Thread 9 time]:                [50.650] ms
[Thread 0 time]:                [0.794] ms
[Thread 1 time]:                [2.667] ms
[Thread 2 time]:                [5.423] ms
[Thread 3 time]:                [8.069] ms
[Thread 16 time]:               [8.290] ms
[Thread 17 time]:               [5.376] ms
[Thread 19 time]:               [0.782] ms
[Thread 5 time]:                [21.849] ms
[Thread 15 time]:               [19.839] ms
[Thread 18 time]:               [2.720] ms
[Thread 4 time]:                [27.988] ms
[Thread 7 time]:                [30.122] ms
[Thread 6 time]:                [30.526] ms
[Thread 14 time]:               [21.839] ms
[Thread 13 time]:               [25.134] ms
[Thread 12 time]:               [30.139] ms
[Thread 8 time]:                [35.790] ms
[Thread 11 time]:               [35.846] ms
[Thread 9 time]:                [49.157] ms
[Thread 10 time]:               [52.514] ms
[Thread 0 time]:                [0.785] ms
[Thread 1 time]:                [2.624] ms
[Thread 2 time]:                [5.424] ms
[Thread 3 time]:                [8.069] ms
[Thread 19 time]:               [0.782] ms
[Thread 16 time]:               [8.298] ms
[Thread 18 time]:               [2.725] ms
```

   a. This is (1) the output of compute times for each individual thread in a 20 thread process to generate the mandelbrot image (1).

12. 

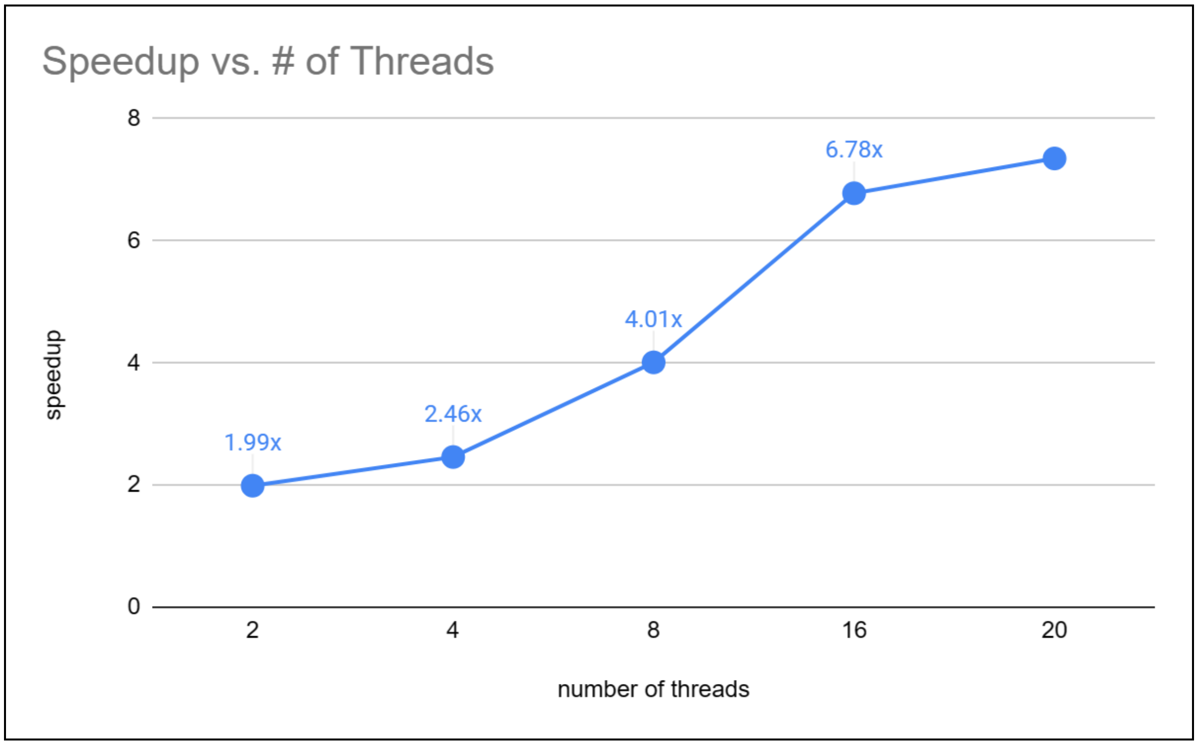
```
[Thread 19 time]:              [0.782] ms
[Thread 16 time]:              [8.298] ms
[Thread 18 time]:              [2.725] ms
[Thread 17 time]:              [5.378] ms
[Thread 4 time]:               [18.915] ms
[Thread 5 time]:               [21.850] ms
[Thread 15 time]:              [19.094] ms
[Thread 7 time]:               [30.151] ms
[Thread 6 time]:               [30.496] ms
[Thread 12 time]:              [30.372] ms
[Thread 14 time]:              [21.833] ms
[Thread 13 time]:              [25.072] ms
[Thread 10 time]:              [43.629] ms
[Thread 11 time]:              [44.675] ms
[Thread 8 time]:               [35.630] ms
[Thread 9 time]:               [38.188] ms
[Thread 0 time]:               [0.787] ms
[Thread 1 time]:               [2.623] ms
[Thread 2 time]:               [5.421] ms
[Thread 3 time]:               [8.069] ms
[Thread 19 time]:              [0.780] ms
[Thread 18 time]:              [2.714] ms
[Thread 4 time]:               [19.891] ms
[Thread 16 time]:              [8.301] ms
[Thread 6 time]:               [25.076] ms
[Thread 15 time]:              [19.038] ms
[Thread 14 time]:              [24.781] ms
[Thread 17 time]:              [5.398] ms
[Thread 7 time]:               [30.111] ms
[Thread 5 time]:               [30.189] ms
[Thread 11 time]:              [35.945] ms
[Thread 13 time]:              [29.024] ms
[Thread 12 time]:              [30.478] ms
[Thread 9 time]:               [38.210] ms
[Thread 8 time]:               [45.302] ms
```

   a. This is the output of compute times for each individual thread in a 20 thread process to generate the mandelbrot image (2).

13. 

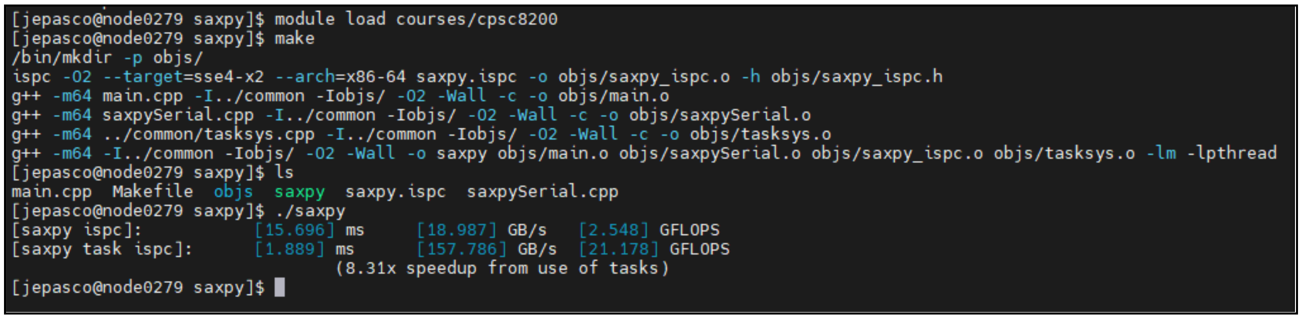
```
void* workerThreadStart(void* threadArgs) {
    double minThread = 1e30;
    double startTime = CycleTimer::currentSeconds();

    WorkerArgs* args = static_cast<WorkerArgs*>(threadArgs);

    mandelbrotSerial(args->x0, args->y0, args->x1, args->y1, args->width, args->height, args->threadId*args->height/args->numThreads,
 args->height/args->numThreads, args->maxIterations, args->output);

    double endTime = CycleTimer::currentSeconds();

    minThread = std::min(minThread, endTime - startTime);
    printf("[Thread %d time]:\t\t[%.3f] ms\n", args->threadId, minThread * 1000);
```

   a. This is the code utilized to time each individual thread.

14. 

   a. This is the graph displaying the speedup from using various numbers of threads to compute the mandelbrot image.

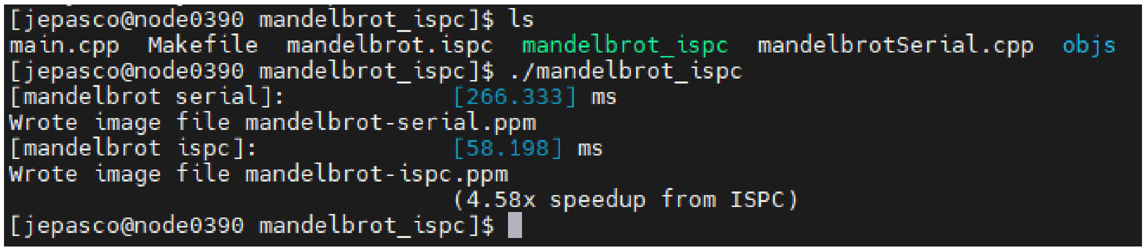
Part 2: BLAS Saxpy

1. 

```
[jepasco@node0279 saxpy]$ module load courses/cpsc8200
[jepasco@node0279 saxpy]$ make
/bin/mkdir -p objs/
ispc -O2 --target=sse4-x2 --arch=x86-64 saxpy.ispc -o objs/saxpy_ispc.o -h objs/saxpy_ispc.h
g++ -m64 main.cpp -I../common -Iobjs/ -O2 -Wall -c -o objs/main.o
g++ -m64 saxpySerial.cpp -I../common -Iobjs/ -O2 -Wall -c -o objs/saxpySerial.o
g++ -m64 ../common/tasksys.cpp -I../common -Iobjs/ -O2 -Wall -c -o objs/tasksys.o
g++ -m64 -I../common -Iobjs/ -O2 -Wall -o saxpy objs/main.o objs/saxpySerial.o objs/saxpy_ispc.o objs/tasksys.o -lm -lpthread
[jepasco@node0279 saxpy]$ ls
main.cpp  Makefile  objs  saxpy  saxpy.ispc  saxpySerial.cpp
[jepasco@node0279 saxpy]$ ./saxpy
[saxpy ispc]:           [15.696] ms     [18.987] GB/s   [2.548] GFLOPS
[saxpy task ispc]:      [1.889] ms      [157.786] GB/s  [21.178] GFLOPS
                                (8.31x speedup from use of tasks)
[jepasco@node0279 saxpy]$ 
```

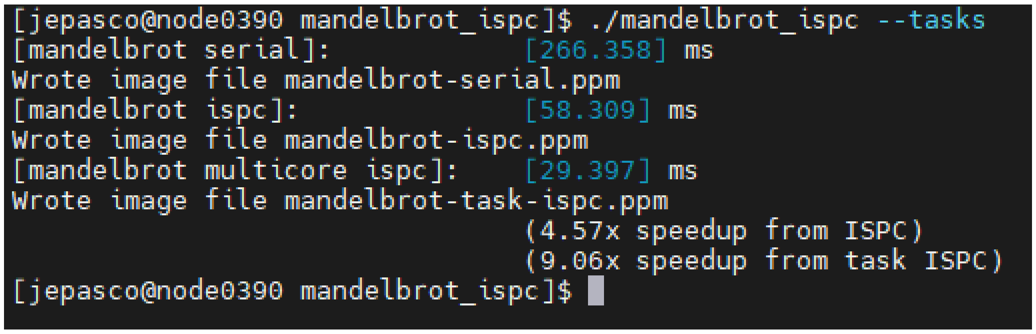
   a. This is an image of compiling the saxpy code and running it. The output indicates that there was an 8.31x speedup from the use of tasks over not using tasks.

Part 3: Parallelize with ISPC

1. 

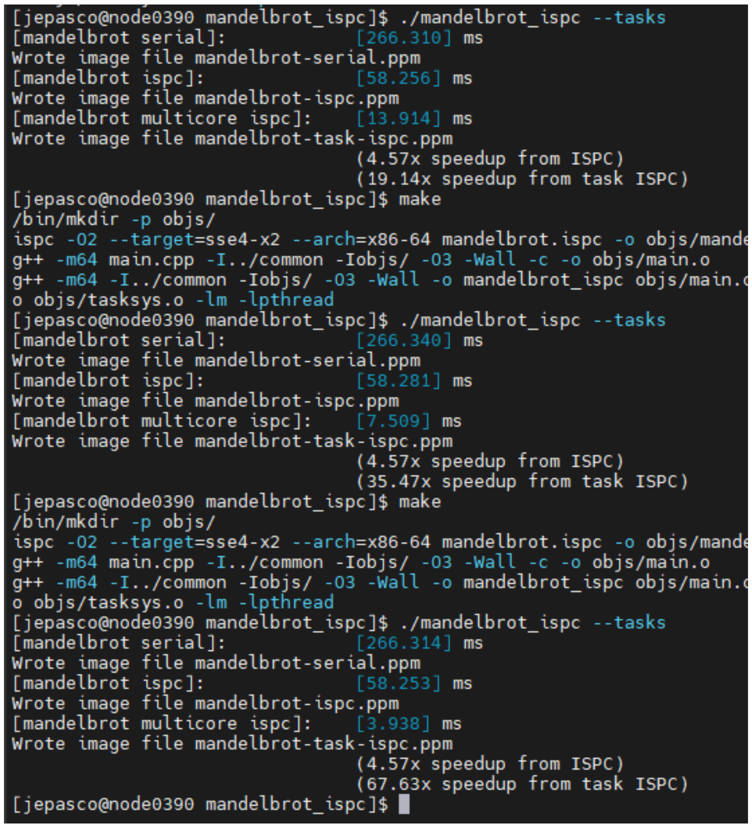
```
[jepasco@node0390 mandelbrot_ispc]$ ls
main.cpp  Makefile  mandelbrot.ispc  mandelbrot_ispc  mandelbrotSerial.cpp  objs
[jepasco@node0390 mandelbrot_ispc]$ ./mandelbrot_ispc
[mandelbrot serial]:            [266.333] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [58.198] ms
Wrote image file mandelbrot-ispc.ppm
                                (4.58x speedup from ISPC)
[jepasco@node0390 mandelbrot_ispc]$ 
```

   a. This is an image of compiling and running the mandelbrot with ISPC and not utilizing tasks. The overall speedup was around 4 times which is expected.

2. 

```
[jepasco@node0390 mandelbrot_ispc]$ ./mandelbrot_ispc --tasks
[mandelbrot serial]:            [266.358] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [58.309] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [29.397] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.57x speedup from ISPC)
                                (9.06x speedup from task ISPC)
[jepasco@node0390 mandelbrot_ispc]$ 
```

   a. This is an image of running the mandelbrot with ISPC and utilizing tasks.

3. 

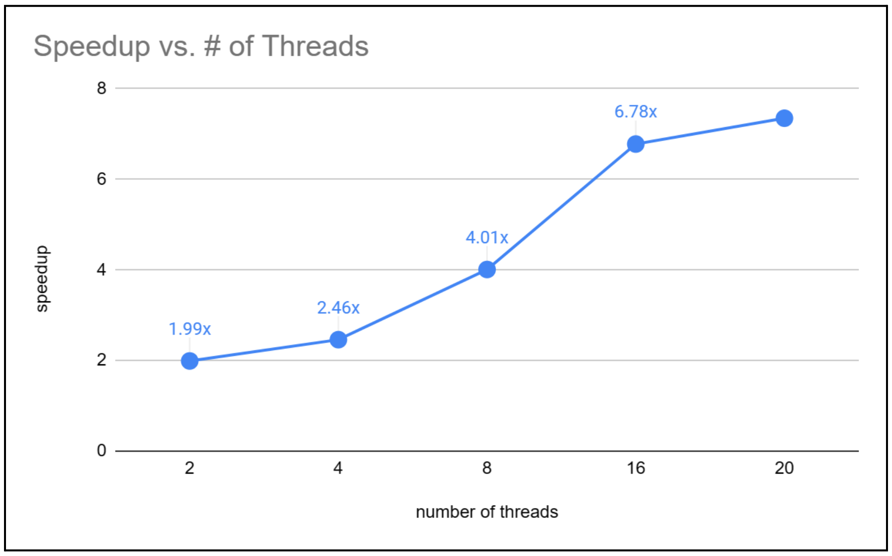
```
[jepasco@node0390 mandelbrot_ispc]$ ./mandelbrot_ispc --tasks
[mandelbrot serial]:            [266.310] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [58.256] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [13.914] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.57x speedup from ISPC)
                                (19.14x speedup from task ISPC)
[jepasco@node0390 mandelbrot_ispc]$ make
/bin/mkdir -p objs/
ispc -O2 --target=sse4-x2 --arch=x86-64 mandelbrot.ispc -o objs/mande
g++ -m64 main.cpp -I../common -Iobjs/ -O3 -Wall -c -o objs/main.o
g++ -m64 -I../common -Iobjs/ -O3 -Wall -o mandelbrot_ispc objs/main.c
o objs/tasksys.o -lm -lpthread
[jepasco@node0390 mandelbrot_ispc]$ ./mandelbrot_ispc --tasks
[mandelbrot serial]:            [266.340] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [58.281] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [7.509] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.57x speedup from ISPC)
                                (35.47x speedup from task ISPC)
[jepasco@node0390 mandelbrot_ispc]$ make
/bin/mkdir -p objs/
ispc -O2 --target=sse4-x2 --arch=x86-64 mandelbrot.ispc -o objs/mande
g++ -m64 main.cpp -I../common -Iobjs/ -O3 -Wall -c -o objs/main.o
g++ -m64 -I../common -Iobjs/ -O3 -Wall -o mandelbrot_ispc objs/main.c
o objs/tasksys.o -lm -lpthread
[jepasco@node0390 mandelbrot_ispc]$ ./mandelbrot_ispc --tasks
[mandelbrot serial]:            [266.314] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [58.253] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [3.938] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.57x speedup from ISPC)
                                (67.63x speedup from task ISPC)
[jepasco@node0390 mandelbrot_ispc]$ 
```

   a. This is an image of compiling and running the mandelbrot with ISPC and tasks with a varying number of tasks (4, 16, 32 tasks respectively).

3: Discussion

Questions

Part 1: Parallel Fractal Generation

1. As the number of threads increased, the speedup produced by threads did not stay linear. This can be seen in the graph below:
   a. 

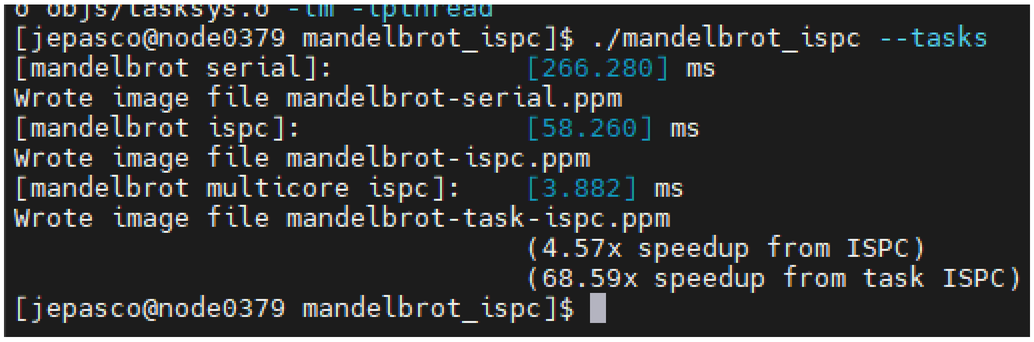
2. There is likely diminishing returns as the number of threads increases because, beyond a certain point, adding more threads doesn't lead to a proportional increase in performance due to factors like thread overhead and resource contention. As threads increase, so does the computation cost to determine what partition of work they complete. Whether speedup is linear or not depends on the scalability of the algorithm and the efficiency of the parallelization - which is an inherent limit of the Mandelbrot set calculation.
3. Utilizing timing code at the start and end of each working thread indicated that generally, as the number of threads increased, the individual thread execution times would decrease due to parallelism but not in a linear fashion. Other factors such as thread startup and shutdown overhead, cache contention, and resource sharing affect the time it takes for a thread to finish.
4. My single work decomposition policy was to breakdown the output array based on how many threads there were and the height of the image. By leveraging the amount of threads in the process, I could limit what each thread works on to the size of args->height/numThreads. I could also always determine where each thread should start by utilizing the calculation numThreads*args->height/numThreads. The maximum 16-thread speedup I was able to obtain was a 7.01x speedup, but they generally hovered around 6.78x. The speedup from both 4 to 8 threads and 8 to 16 threads are nearly the same with a speedup factor of 1.63 (4.01/2.46) and 1.69 (6.78/4.01) respectively.

Part 2: BLAX Saxpy

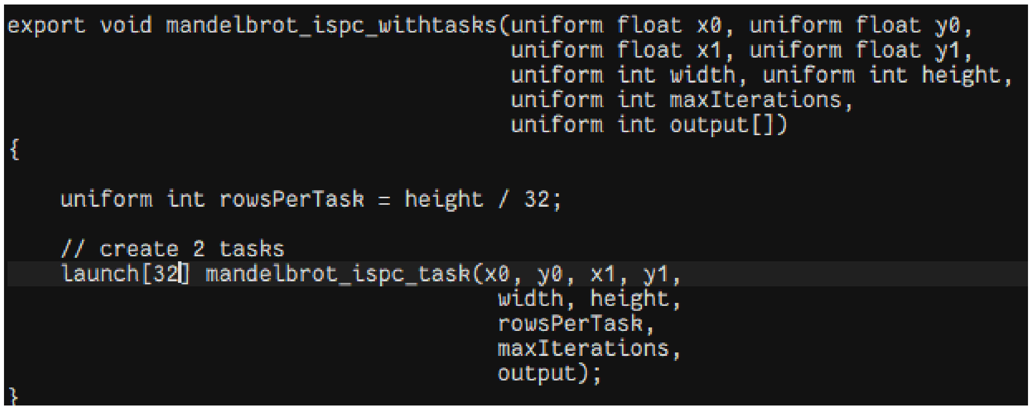
1. The speedup obtained through the use of tasks in ISPC for Saxpy was significant as it was 8.31x faster than without tasks. This indicates that there was effective parallelism in the implementation. There was also a notably large increase in both memory and GFLOP usage for the program with tasks. That is a common tradeoff when parallelizing code. I do not think the performance of the program can be substantially improved because Saxpy is not a highly parallelizable problem and near linear speedup is typically only achievable with problems that have a high degree of parallelism. Any further optimization through parallelizing other trivial parts of the program.

Part 3: Parallelize with ISPC

1. The maximum speedup expected for mandelbrot using 4-wide SSE vector instructions is generally up to 4x. Real-world performance may be different due to the nature of the computation, load imbalance, and data dependencies. Experimenting with different vectorization strategies and AVX instructions can help improve SIMD efficiency and increase the speedup. Different characteristics in terms of the distribution of points that belong to the mandelbrot set can impact SIMD efficiency as, if one view contains more complex points that require a higher number of iterations to determine their membership, SIMD efficiency may be lower, leading to a lower speedup. Especially if some SIMD lanes finish their work early which leads to underutilization of resources.
2. By utilizing the --tasks parameter, the overall speedup jumped from 4.57x with ISPC to 9.06x from tasked ISPC.
3. By changing the amount of tasks that produce the mandelbrot image, I was able to increase the speedup by well over 19 times. The tasks count that got me the closest to 19 times speedup was 4 tasks. In image 2 of Results, you can see that the speedup continued to increase even when 32 tasks were utilized. By carefully adjusting the number of tasks and measuring performance, it is possible to find the optimal task count that significantly improves performance. The optimal task count works best because it effectively utilizes the available CPU resources and ensures that the CPU cores remain busy with computations and minimize idle time. I utilized a node with 56 CPU cores so effectively splitting the number of tasks to that number of cores (or even parallelized to multiple threads) would stand to most likely be the most efficient task count to use. That way, all of the cores would effectively be in use and that is why it would work best to have that many tasks. The amount of tasks I found to work the best as exponents of 2 was 32 tasks as this is the closest number to the 56 cores without going over it. This amount of tasks lead to a speedup of over 68 times. Here is the command I used to obtain a node:
    a.
    ```
    qsub -I -l
    select=1:ncpus=56:ngpus=1:gpu_model=a100,walltime=2:00:00
    ```

b. 

```
o objs/tasksys.o -lm -lpthread
[jepasco@node0379 mandelbrot_ispc]$ ./mandelbrot_ispc --tasks
[mandelbrot serial]:            [266.280] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [58.260] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [3.882] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.57x speedup from ISPC)
                                (68.59x speedup from task ISPC)
[jepasco@node0379 mandelbrot_ispc]$
```

c. 

```
export void mandelbrot_ispc_withtasks(uniform float x0, uniform float y0,
                                      uniform float x1, uniform float y1,
                                      uniform int width, uniform int height,
                                      uniform int maxIterations,
                                      uniform int output[])
{

    uniform int rowsPerTask = height / 32;

    // create 2 tasks
    launch[32] mandelbrot_ispc_task(x0, y0, x1, y1,
                                     width, height,
                                     rowsPerTask,
                                     maxIterations,
                                     output);
}
```