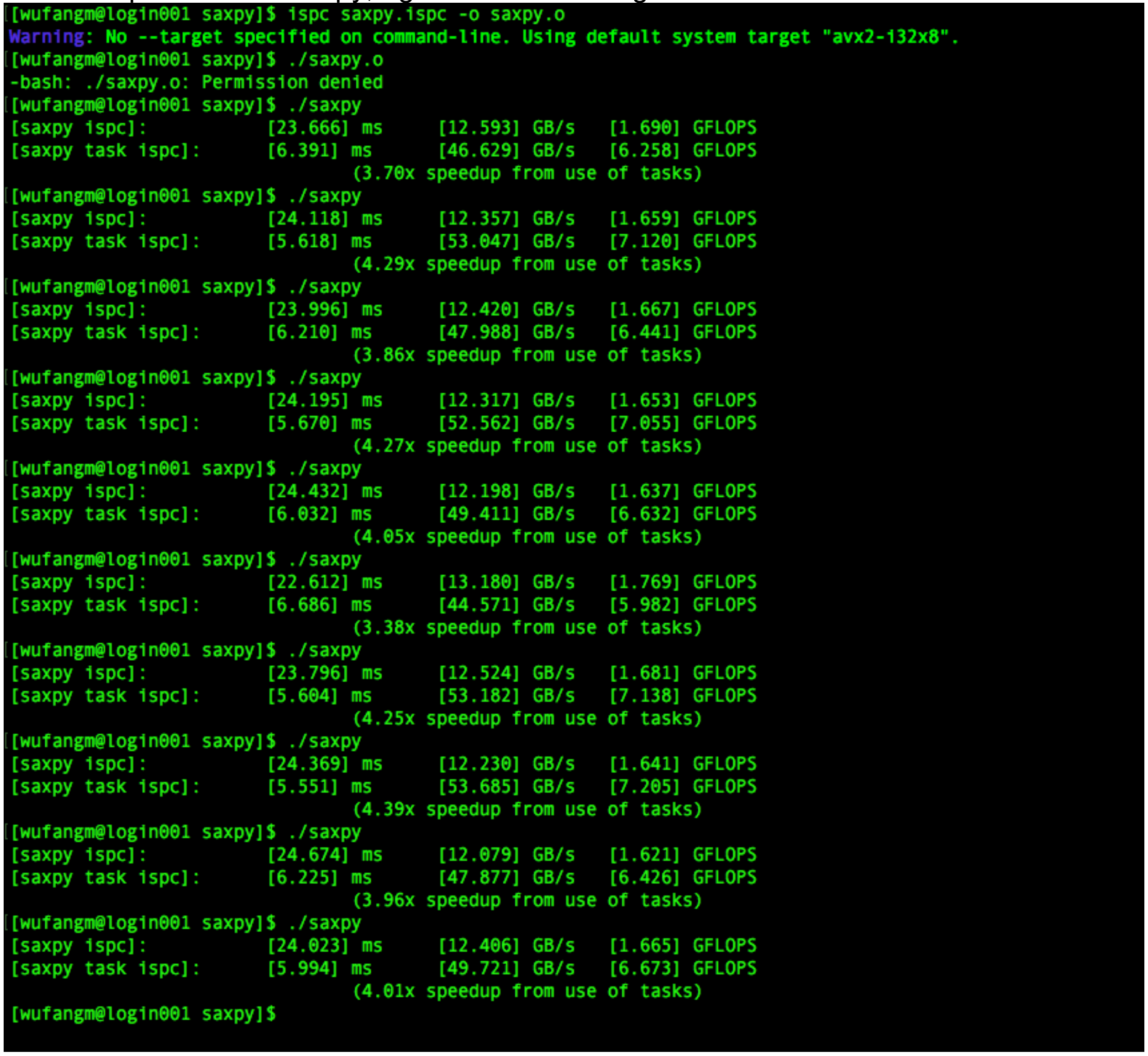
Part 1: BLAS saxpy

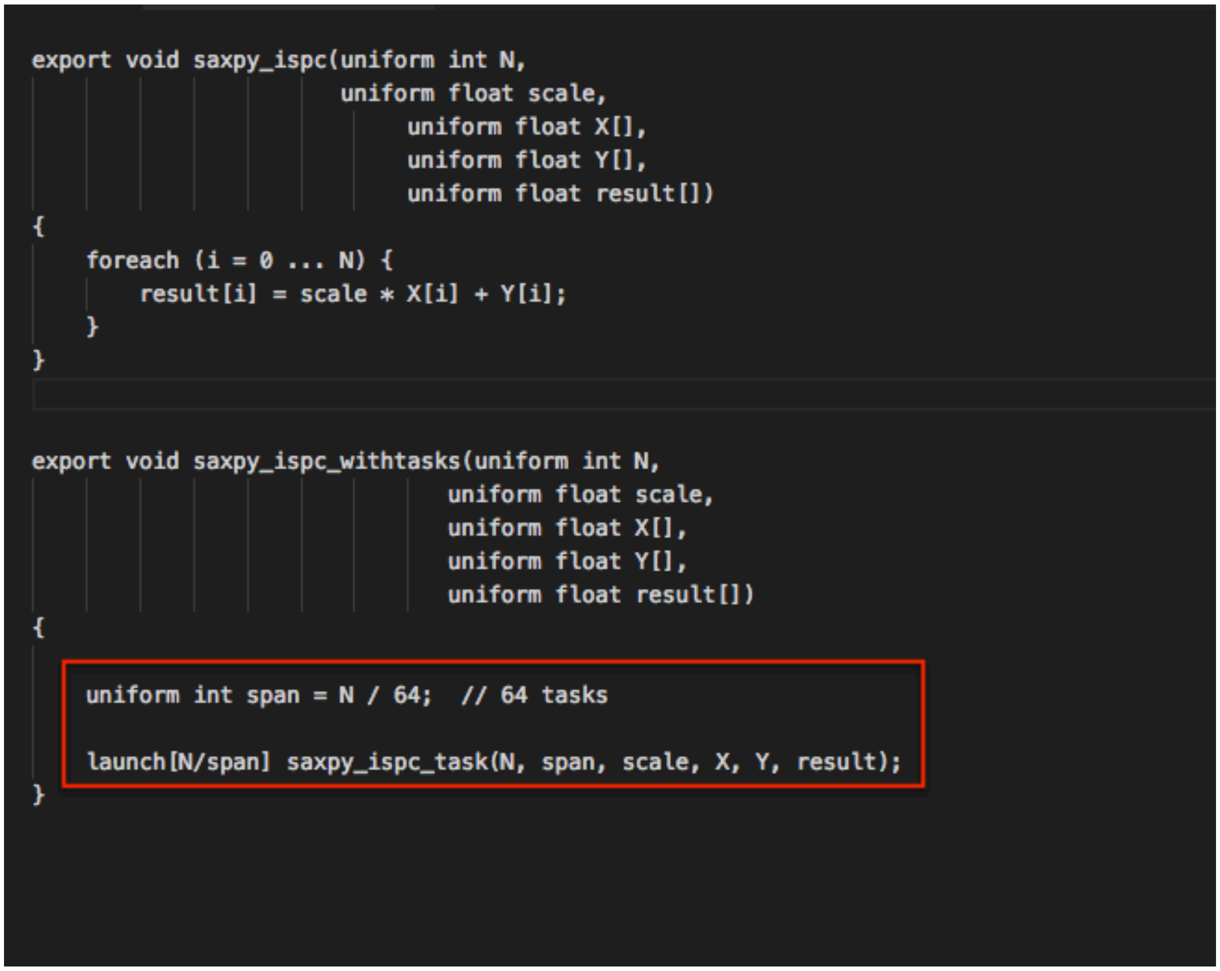
After compile and run saxpy, I got results showing below:



```
[wufangm@login001 saxpy]$ ispc saxpy.ispc -o saxpy.o
Warning: No --target specified on command-line. Using default system target "avx2-i32x8".
[wufangm@login001 saxpy]$ ./saxpy.o
-bash: ./saxpy.o: Permission denied
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [23.666] ms     [12.593] GB/s   [1.690] GFLOPS
[saxpy task ispc]:      [6.391] ms      [46.629] GB/s   [6.258] GFLOPS
                                (3.70x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [24.118] ms     [12.357] GB/s   [1.659] GFLOPS
[saxpy task ispc]:      [5.618] ms      [53.047] GB/s   [7.120] GFLOPS
                                (4.29x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [23.996] ms     [12.420] GB/s   [1.667] GFLOPS
[saxpy task ispc]:      [6.210] ms      [47.988] GB/s   [6.441] GFLOPS
                                (3.86x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [24.195] ms     [12.317] GB/s   [1.653] GFLOPS
[saxpy task ispc]:      [5.670] ms      [52.562] GB/s   [7.055] GFLOPS
                                (4.27x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [24.432] ms     [12.198] GB/s   [1.637] GFLOPS
[saxpy task ispc]:      [6.032] ms      [49.411] GB/s   [6.632] GFLOPS
                                (4.05x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [22.612] ms     [13.180] GB/s   [1.769] GFLOPS
[saxpy task ispc]:      [6.686] ms      [44.571] GB/s   [5.982] GFLOPS
                                (3.38x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [23.796] ms     [12.524] GB/s   [1.681] GFLOPS
[saxpy task ispc]:      [5.604] ms      [53.182] GB/s   [7.138] GFLOPS
                                (4.25x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [24.369] ms     [12.230] GB/s   [1.641] GFLOPS
[saxpy task ispc]:      [5.551] ms      [53.685] GB/s   [7.205] GFLOPS
                                (4.39x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [24.674] ms     [12.079] GB/s   [1.621] GFLOPS
[saxpy task ispc]:      [6.225] ms      [47.877] GB/s   [6.426] GFLOPS
                                (3.96x speedup from use of tasks)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy ispc]:           [24.023] ms     [12.406] GB/s   [1.665] GFLOPS
[saxpy task ispc]:      [5.994] ms      [49.721] GB/s   [6.673] GFLOPS
                                (4.01x speedup from use of tasks)
[wufangm@login001 saxpy]$
```

Figure 1: Observing the result between the performance of ISPC without tasks and ISPC with tasks.

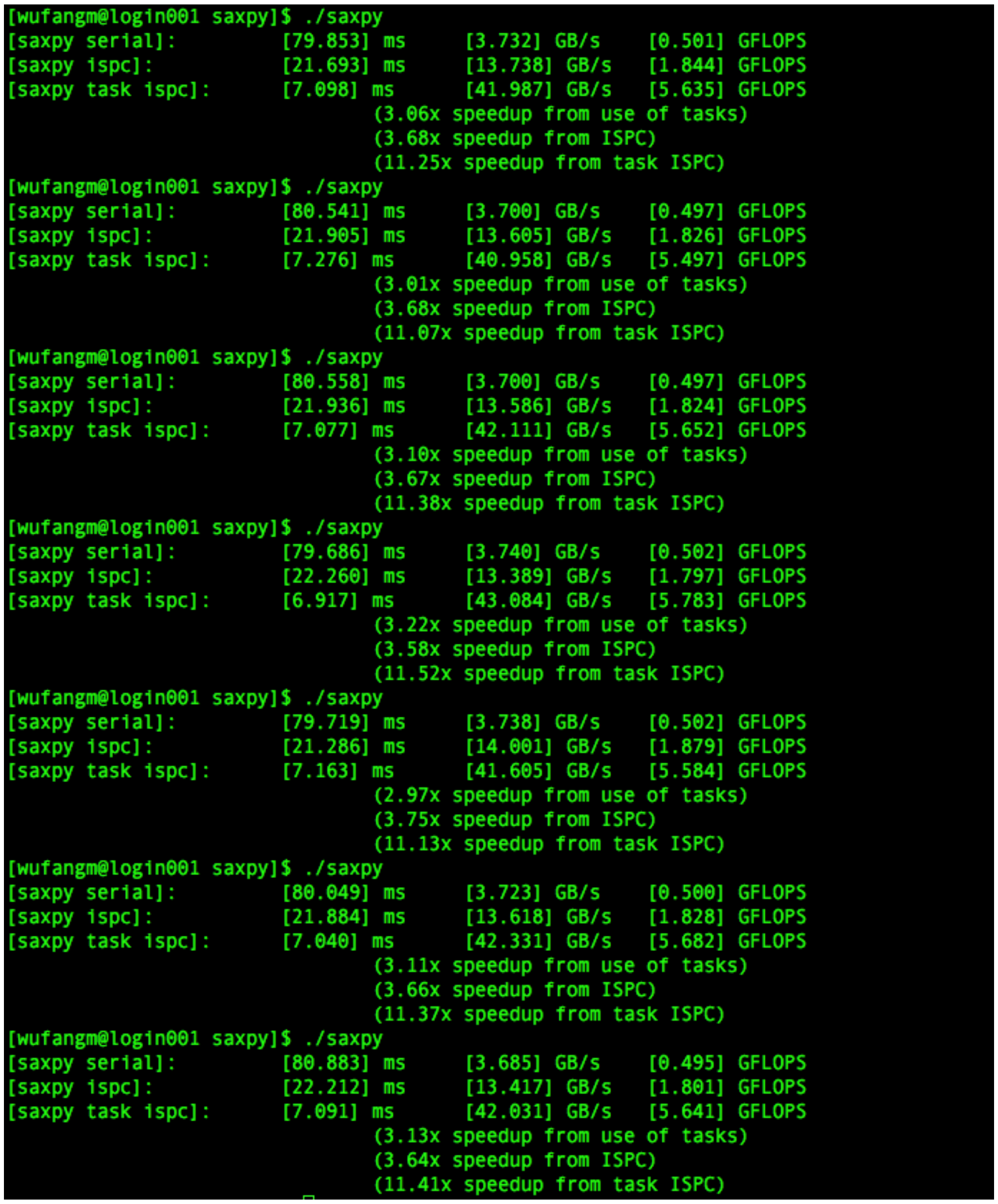
In Figure 1, I can observe that ISPC with tasks has better performance compared with ISPC without tasks(ISPC with tasks is almost 4 times of ISPC without tasks). The reason, personally speaking, is that despite ISPC without tasks uses the SIMD framework provided by ISPC, It does not consider the idea of parallelism in the algorithm, which still iterates the tasks that will be processed. Conversely, ISPC with tasks utilizes the idea of parallelism, as the Figure 2 shows, in the task processing, where ISPC with tasks divides the task into 64 parts and computes them with multiple cores.



```
export void saxpy_ispc(uniform int N,
                       uniform float scale,
                            uniform float X[],
                            uniform float Y[],
                            uniform float result[])
{
    foreach (i = 0 ... N) {
        result[i] = scale * X[i] + Y[i];
    }
}


export void saxpy_ispc_withtasks(uniform int N,
                               uniform float scale,
                               uniform float X[],
                               uniform float Y[],
                               uniform float result[])
{

    uniform int span = N / 64;  // 64 tasks

    launch[N/span] saxpy_ispc_task(N, span, scale, X, Y, result);
}
```

Figure 2: *saxpy_ispc_withtasks* uses the idea of parallelism to process the takes instead of dealing with teaks in an iterative way in the function of *saxpy_ispc*.

For the second question, my answer is yes. Compared with the linear computing, the ISPC framework does greatly improve the performance of this program. In Figure 3, I observed that ISPC with tasks is almost 12 times faster than serial computing, which is a huge promotion. Even for the performance of ISPC without tasks, its performance is nearly 3 times faster than serial function.

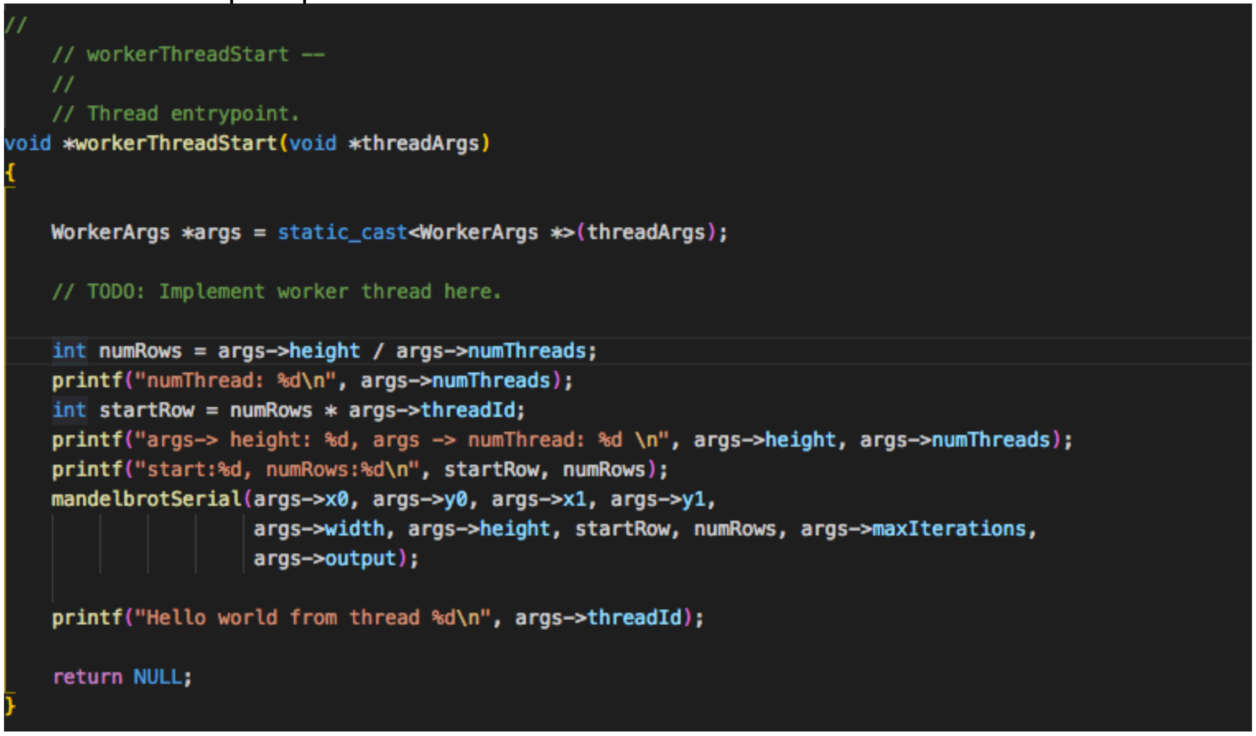


```
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [79.853] ms     [3.732] GB/s    [0.501] GFLOPS
[saxpy ispc]:           [21.693] ms     [13.738] GB/s   [1.844] GFLOPS
[saxpy task ispc]:      [7.098] ms      [41.987] GB/s   [5.635] GFLOPS
                                (3.06x speedup from use of tasks)
                                (3.68x speedup from ISPC)
                                (11.25x speedup from task ISPC)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [80.541] ms     [3.700] GB/s    [0.497] GFLOPS
[saxpy ispc]:           [21.905] ms     [13.605] GB/s   [1.826] GFLOPS
[saxpy task ispc]:      [7.276] ms      [40.958] GB/s   [5.497] GFLOPS
                                (3.01x speedup from use of tasks)
                                (3.68x speedup from ISPC)
                                (11.07x speedup from task ISPC)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [80.558] ms     [3.700] GB/s    [0.497] GFLOPS
[saxpy ispc]:           [21.936] ms     [13.586] GB/s   [1.824] GFLOPS
[saxpy task ispc]:      [7.077] ms      [42.111] GB/s   [5.652] GFLOPS
                                (3.10x speedup from use of tasks)
                                (3.67x speedup from ISPC)
                                (11.38x speedup from task ISPC)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [79.686] ms     [3.740] GB/s    [0.502] GFLOPS
[saxpy ispc]:           [22.260] ms     [13.389] GB/s   [1.797] GFLOPS
[saxpy task ispc]:      [6.917] ms      [43.084] GB/s   [5.783] GFLOPS
                                (3.22x speedup from use of tasks)
                                (3.58x speedup from ISPC)
                                (11.52x speedup from task ISPC)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [79.719] ms     [3.738] GB/s    [0.502] GFLOPS
[saxpy ispc]:           [21.286] ms     [14.001] GB/s   [1.879] GFLOPS
[saxpy task ispc]:      [7.163] ms      [41.605] GB/s   [5.584] GFLOPS
                                (2.97x speedup from use of tasks)
                                (3.75x speedup from ISPC)
                                (11.13x speedup from task ISPC)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [80.049] ms     [3.723] GB/s    [0.500] GFLOPS
[saxpy ispc]:           [21.884] ms     [13.618] GB/s   [1.828] GFLOPS
[saxpy task ispc]:      [7.040] ms      [42.331] GB/s   [5.682] GFLOPS
                                (3.11x speedup from use of tasks)
                                (3.66x speedup from ISPC)
                                (11.37x speedup from task ISPC)
[wufangm@login001 saxpy]$ ./saxpy
[saxpy serial]:         [80.883] ms     [3.685] GB/s    [0.495] GFLOPS
[saxpy ispc]:           [22.212] ms     [13.417] GB/s   [1.801] GFLOPS
[saxpy task ispc]:      [7.091] ms      [42.031] GB/s   [5.641] GFLOPS
                                (3.13x speedup from use of tasks)
                                (3.64x speedup from ISPC)
                                (11.41x speedup from task ISPC)
```

Figure 3: Performance comparison after adding linear calculation

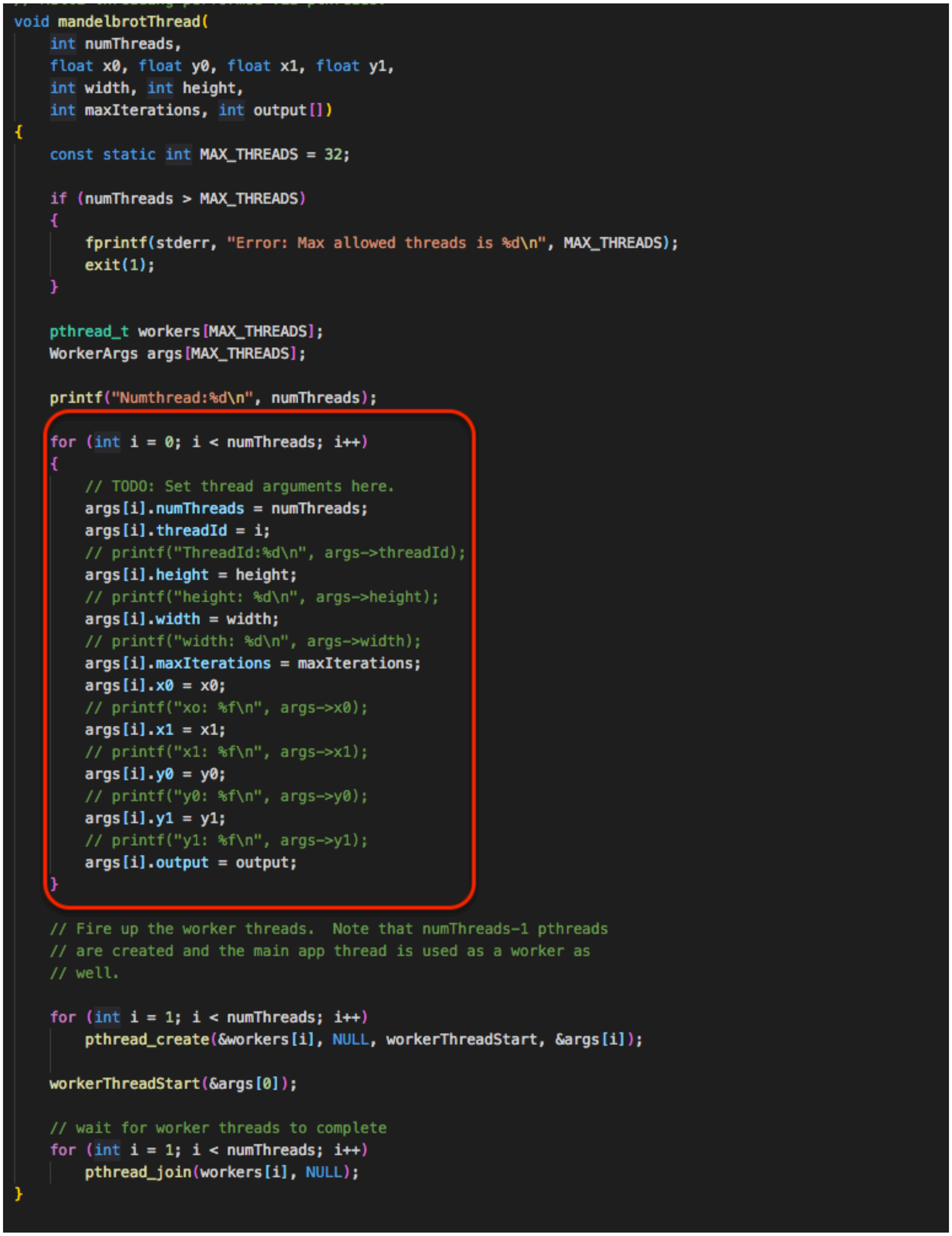
Part 2: Parallel Fractal Generation Using Pthreads

I have added code in the 2 part of *mandelbrotThread()*(shown in Figure 4 and Figure 5). In the Figure 4, I divide the image evenly by row according to the number of threads. For example, If the number of threads is 2, then the image is divided into two parts. Similarly, if the number of threads is 16, the image is divided into 16 parts. The calculation of each part of image is handled by on thread. In the Figure 5, I pass every parameters of thread to each thread. In my program, I use startRow to record the current line that thread should compute and the numRows records the number of rows for each thread per operation.



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

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

    // TODO: Implement worker thread here.

    int numRows = args->height / args->numThreads;
    printf("numThread: %d\n", args->numThreads);
    int startRow = numRows * args->threadId;
    printf("args-> height: %d, args -> numThread: %d \n", args->height, args->numThreads);
    printf("start:%d, numRows:%d\n", startRow, numRows);
    mandelbrotSerial(args->x0, args->y0, args->x1, args->y1,
                     args->width, args->height, startRow, numRows, args->maxIterations,
                     args->output);

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

    return NULL;
}
```

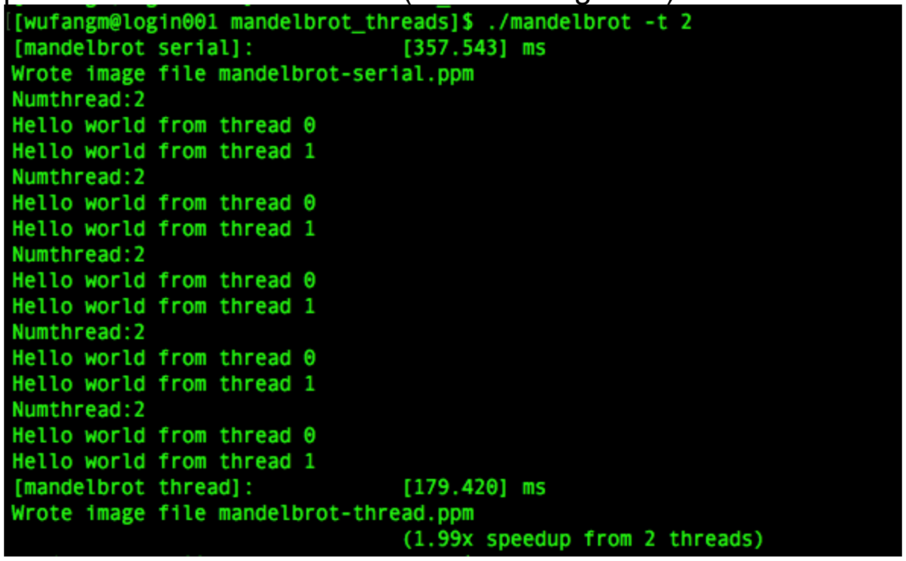
Figure 4



```c
void mandelbrotThread(
    int numThreads,
    float x0, float y0, float x1, float y1,
    int width, int height,
    int maxIterations, int output[])
{
    const static int MAX_THREADS = 32;

    if (numThreads > MAX_THREADS)
    {
        fprintf(stderr, "Error: Max allowed threads is %d\n", MAX_THREADS);
        exit(1);
    }

    pthread_t workers[MAX_THREADS];
    WorkerArgs args[MAX_THREADS];

    printf("Numthread:%d\n", numThreads);

    for (int i = 0; i < numThreads; i++)
    {
        // TODO: Set thread arguments here.
        args[i].numThreads = numThreads;
        args[i].threadId = i;
        // printf("ThreadId:%d\n", args->threadId);
        args[i].height = height;
        // printf("height: %d\n", args->height);
        args[i].width = width;
        // printf("width: %d\n", args->width);
        args[i].maxIterations = maxIterations;
        args[i].x0 = x0;
        // printf("xo: %f\n", args->x0);
        args[i].x1 = x1;
        // printf("x1: %f\n", args->x1);
        args[i].y0 = y0;
        // printf("y0: %f\n", args->y0);
        args[i].y1 = y1;
        // printf("y1: %f\n", args->y1);
        args[i].output = output;
    }

    // Fire up the worker threads.  Note that numThreads-1 pthreads
    // are created and the main app thread is used as a worker as
    // well.

    for (int i = 1; i < numThreads; i++)
        pthread_create(&workers[i], NULL, workerThreadStart, &args[i]);

    workerThreadStart(&args[0]);

    // wait for worker threads to complete
    for (int i = 1; i < numThreads; i++)
        pthread_join(workers[i], NULL);
}
```

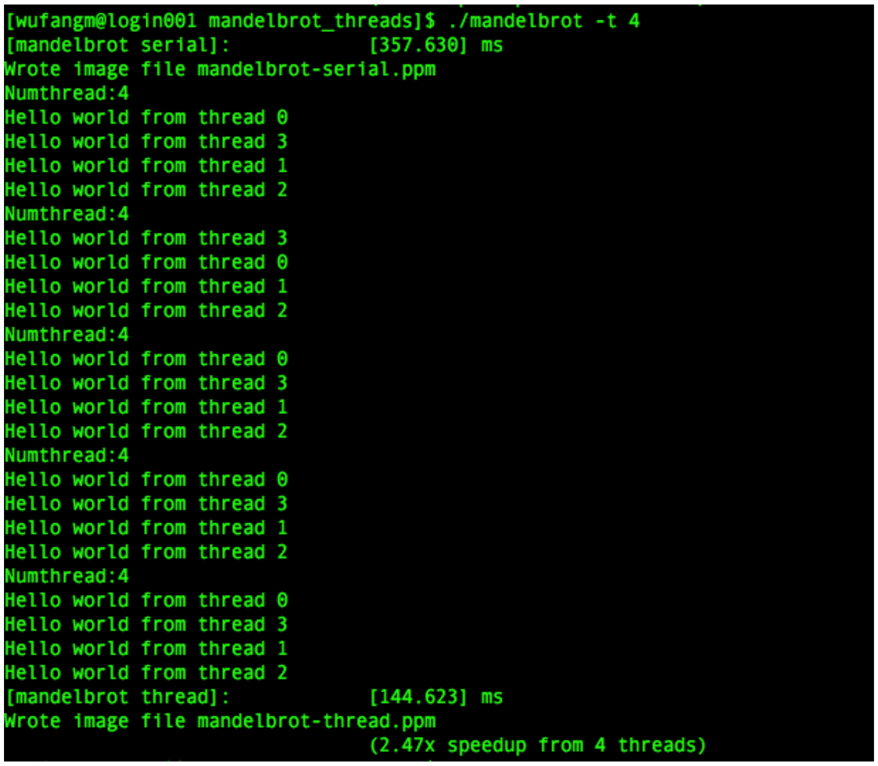
Figure 5

When running this program for View 1, I found that the more threads participate in the calculation, the faster the program running. When two threads participate in the calculation, the speed is nearly twice time(1.99X) compared with mandelbrotSerial. Similarly, when 4,8,16 participate in the calculation, the speed is 2.47X, 3.97X and &7.21X compared with mandelbrotSerial(shown in Figure 6).



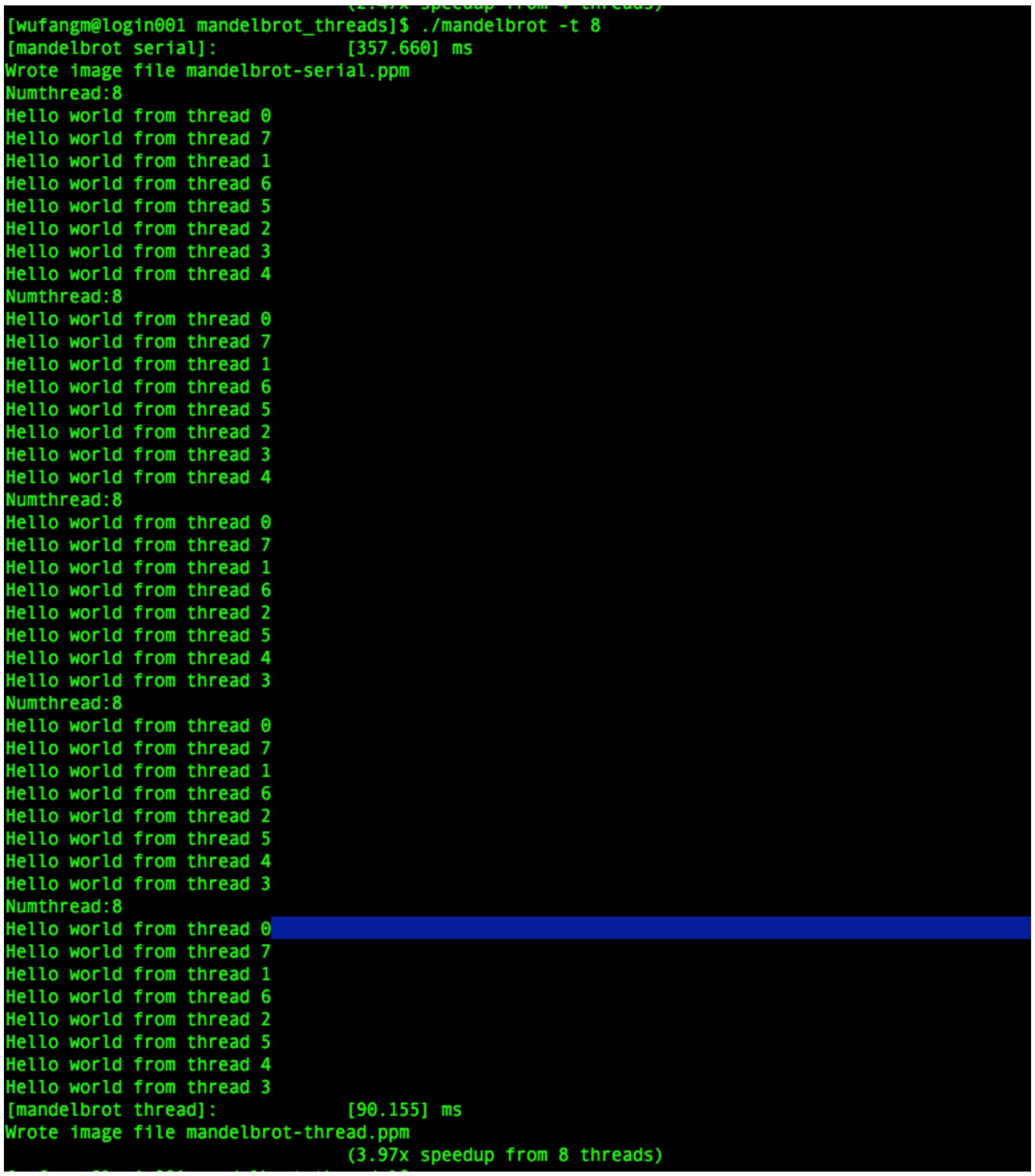
```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 2
[mandelbrot serial]:            [357.543] ms
Wrote image file mandelbrot-serial.ppm
Numthread:2
Hello world from thread 0
Hello world from thread 1
Numthread:2
Hello world from thread 0
Hello world from thread 1
Numthread:2
Hello world from thread 0
Hello world from thread 1
Numthread:2
Hello world from thread 0
Hello world from thread 1
Numthread:2
Hello world from thread 0
Hello world from thread 1
[mandelbrot thread]:            [179.420] ms
Wrote image file mandelbrot-thread.ppm
                                (1.99x speedup from 2 threads)
```

Figure 6(1): running program in 2 threads



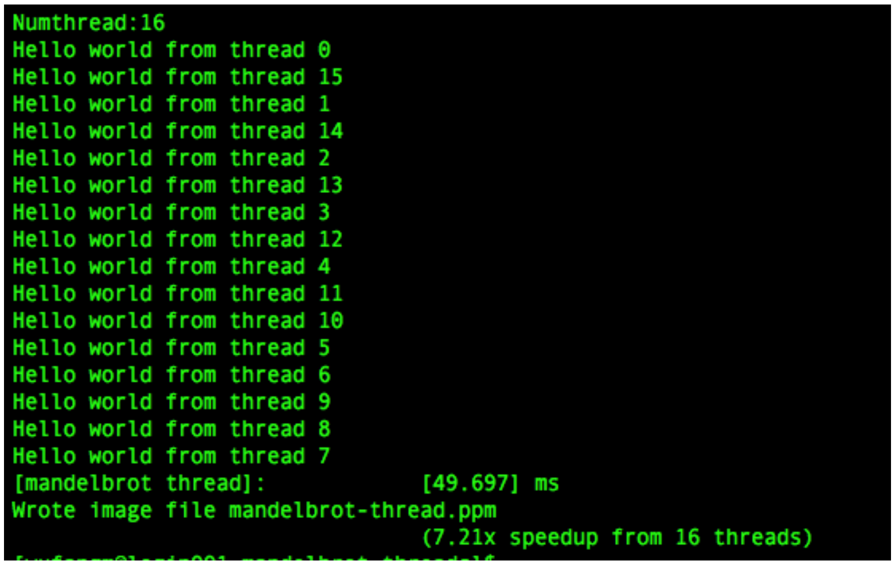
```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 4
[mandelbrot serial]:            [357.630] ms
Wrote image file mandelbrot-serial.ppm
Numthread:4
Hello world from thread 0
Hello world from thread 3
Hello world from thread 1
Hello world from thread 2
Numthread:4
Hello world from thread 3
Hello world from thread 0
Hello world from thread 1
Hello world from thread 2
Numthread:4
Hello world from thread 0
Hello world from thread 3
Hello world from thread 1
Hello world from thread 2
Numthread:4
Hello world from thread 0
Hello world from thread 3
Hello world from thread 1
Hello world from thread 2
Numthread:4
Hello world from thread 0
Hello world from thread 3
Hello world from thread 1
Hello world from thread 2
[mandelbrot thread]:            [144.623] ms
Wrote image file mandelbrot-thread.ppm
                                (2.47x speedup from 4 threads)
```

Figure 6(2): running program in 4 threads



```
                                   (2.47x speedup from 4 threads)
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 8
[mandelbrot serial]:               [357.660] ms
Wrote image file mandelbrot-serial.ppm
Numthread:8
Hello world from thread 0
Hello world from thread 7
Hello world from thread 1
Hello world from thread 6
Hello world from thread 5
Hello world from thread 2
Hello world from thread 3
Hello world from thread 4
Numthread:8
Hello world from thread 0
Hello world from thread 7
Hello world from thread 1
Hello world from thread 6
Hello world from thread 5
Hello world from thread 2
Hello world from thread 3
Hello world from thread 4
Numthread:8
Hello world from thread 0
Hello world from thread 7
Hello world from thread 1
Hello world from thread 6
Hello world from thread 2
Hello world from thread 5
Hello world from thread 4
Hello world from thread 3
Numthread:8
Hello world from thread 0
Hello world from thread 7
Hello world from thread 1
Hello world from thread 6
Hello world from thread 2
Hello world from thread 5
Hello world from thread 4
Hello world from thread 3
Numthread:8
Hello world from thread 0
Hello world from thread 7
Hello world from thread 1
Hello world from thread 6
Hello world from thread 2
Hello world from thread 5
Hello world from thread 4
Hello world from thread 3
[mandelbrot thread]:               [90.155] ms
Wrote image file mandelbrot-thread.ppm
                                   (3.97x speedup from 8 threads)
```

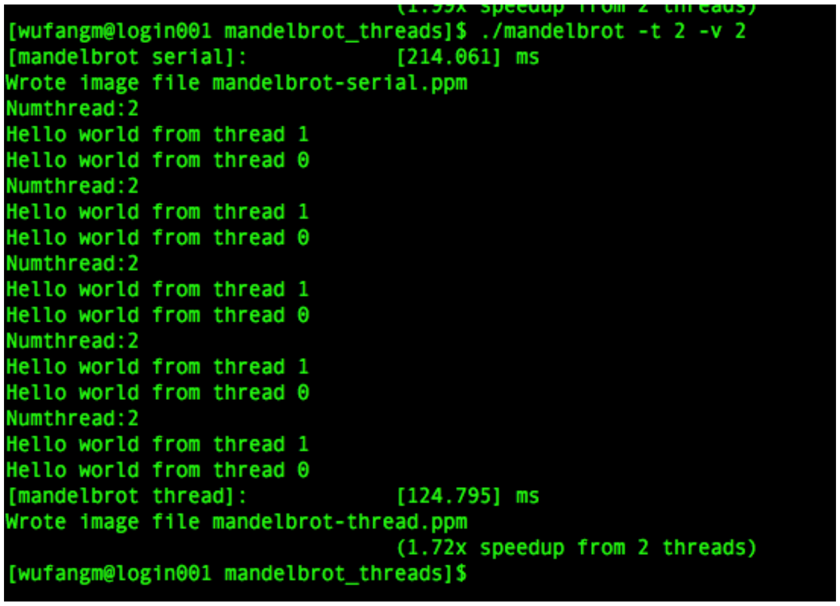
Figure 6(3): running program in 8 threads



```
Numthread:16
Hello world from thread 0
Hello world from thread 15
Hello world from thread 1
Hello world from thread 14
Hello world from thread 2
Hello world from thread 13
Hello world from thread 3
Hello world from thread 12
Hello world from thread 4
Hello world from thread 11
Hello world from thread 10
Hello world from thread 5
Hello world from thread 6
Hello world from thread 9
Hello world from thread 8
Hello world from thread 7
[mandelbrot thread]:            [49.697] ms
Wrote image file mandelbrot-thread.ppm
                                (7.21x speedup from 16 threads)
```

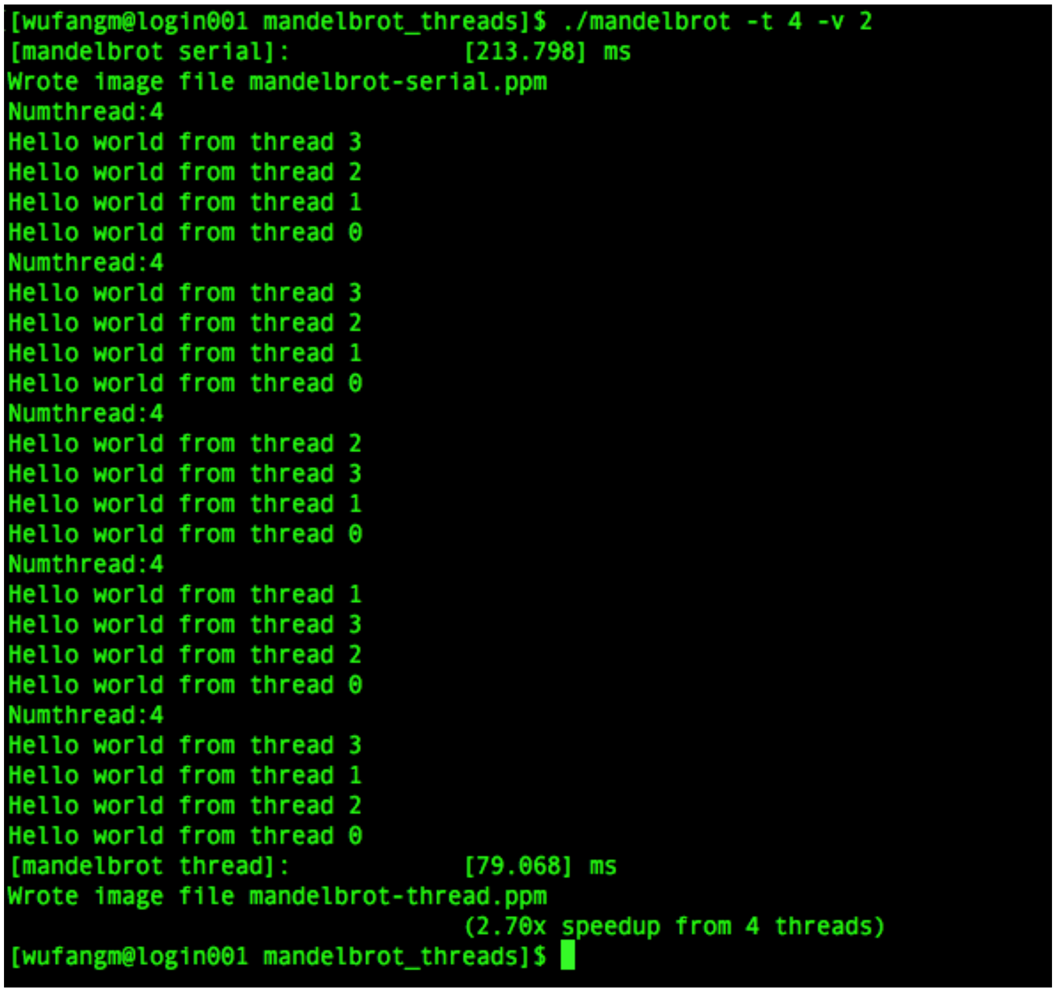
Figure 6(4): running program in 16 threads

When I calculate in View 2, I found that there is no big change in the computation speed (shown in Figure 7). The speed improvement for 2,4,6,18 threads is 1.72x, 2.70x, 4.09x and 7.27x



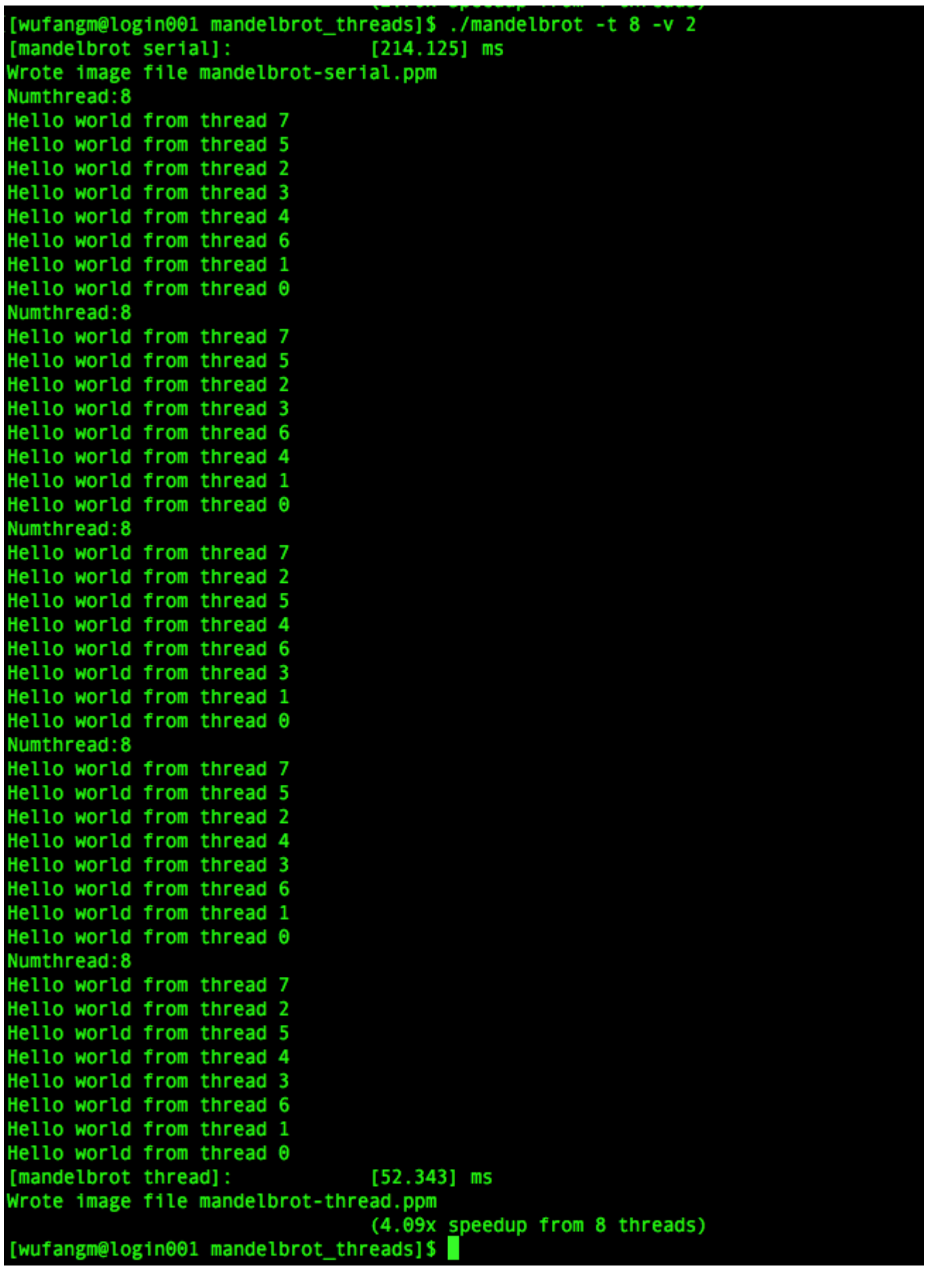
```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 2 -v 2
[mandelbrot serial]:            [214.061] ms
Wrote image file mandelbrot-serial.ppm
Numthread:2
Hello world from thread 1
Hello world from thread 0
Numthread:2
Hello world from thread 1
Hello world from thread 0
Numthread:2
Hello world from thread 1
Hello world from thread 0
Numthread:2
Hello world from thread 1
Hello world from thread 0
Numthread:2
Hello world from thread 1
Hello world from thread 0
[mandelbrot thread]:            [124.795] ms
Wrote image file mandelbrot-thread.ppm
                                (1.72x speedup from 2 threads)
[wufangm@login001 mandelbrot_threads]$
```

Figure 7(1): running program in 2 threads for View 2



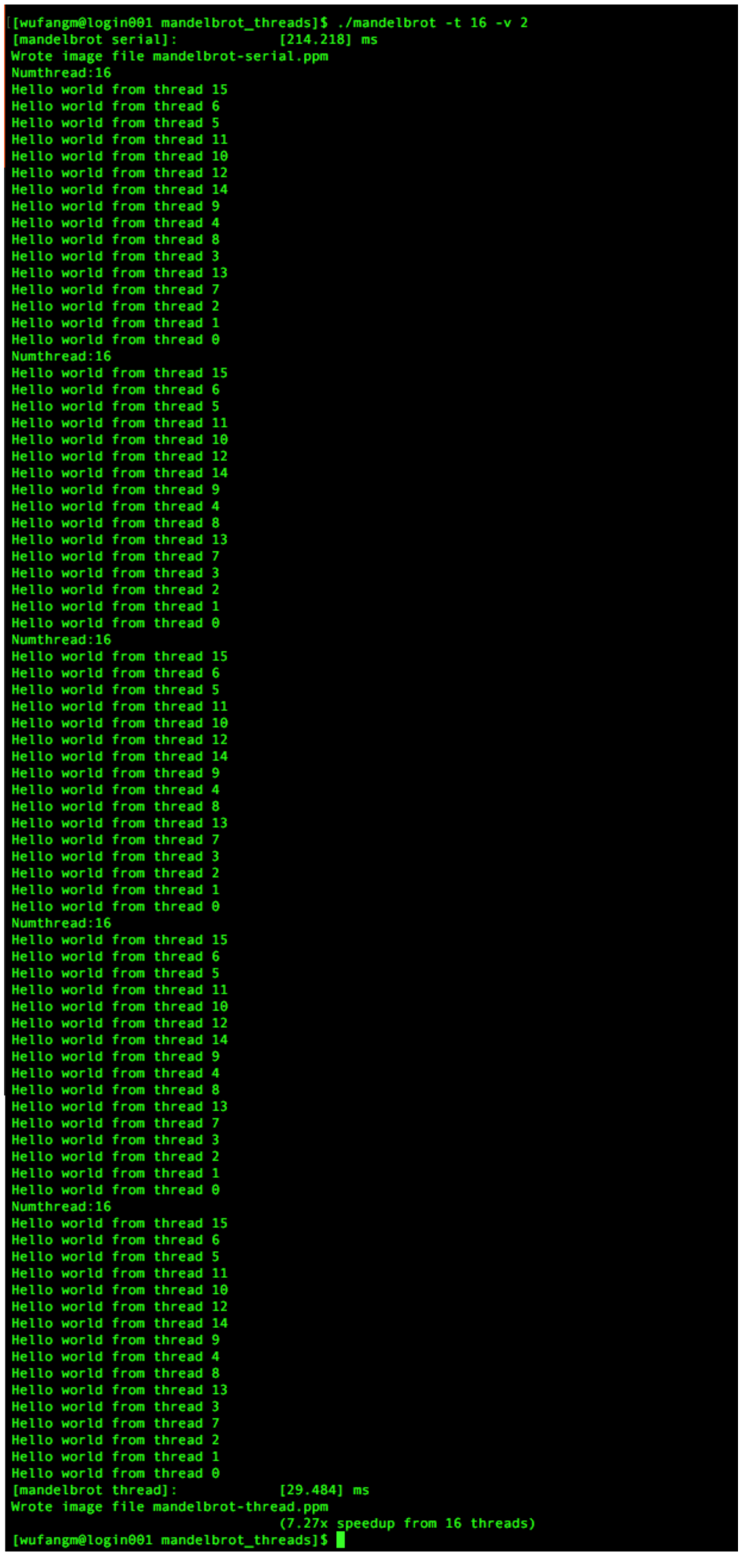
```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 4 -v 2
[mandelbrot serial]:            [213.798] ms
Wrote image file mandelbrot-serial.ppm
Numthread:4
Hello world from thread 3
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
Numthread:4
Hello world from thread 3
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
Numthread:4
Hello world from thread 2
Hello world from thread 3
Hello world from thread 1
Hello world from thread 0
Numthread:4
Hello world from thread 1
Hello world from thread 3
Hello world from thread 2
Hello world from thread 0
Numthread:4
Hello world from thread 3
Hello world from thread 1
Hello world from thread 2
Hello world from thread 0
[mandelbrot thread]:            [79.068] ms
Wrote image file mandelbrot-thread.ppm
                                (2.70x speedup from 4 threads)
[wufangm@login001 mandelbrot_threads]$ 
```

Figure 7(2): running program in 4 threads for View 2



```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 8 -v 2
[mandelbrot serial]:            [214.125] ms
Wrote image file mandelbrot-serial.ppm
Numthread:8
Hello world from thread 7
Hello world from thread 5
Hello world from thread 2
Hello world from thread 3
Hello world from thread 4
Hello world from thread 6
Hello world from thread 1
Hello world from thread 0
Numthread:8
Hello world from thread 7
Hello world from thread 5
Hello world from thread 2
Hello world from thread 3
Hello world from thread 6
Hello world from thread 4
Hello world from thread 1
Hello world from thread 0
Numthread:8
Hello world from thread 7
Hello world from thread 2
Hello world from thread 5
Hello world from thread 4
Hello world from thread 6
Hello world from thread 3
Hello world from thread 1
Hello world from thread 0
Numthread:8
Hello world from thread 7
Hello world from thread 5
Hello world from thread 2
Hello world from thread 4
Hello world from thread 3
Hello world from thread 6
Hello world from thread 1
Hello world from thread 0
Numthread:8
Hello world from thread 7
Hello world from thread 2
Hello world from thread 5
Hello world from thread 4
Hello world from thread 3
Hello world from thread 6
Hello world from thread 1
Hello world from thread 0
[mandelbrot thread]:            [52.343] ms
Wrote image file mandelbrot-thread.ppm
                                (4.09x speedup from 8 threads)
[wufangm@login001 mandelbrot_threads]$
```

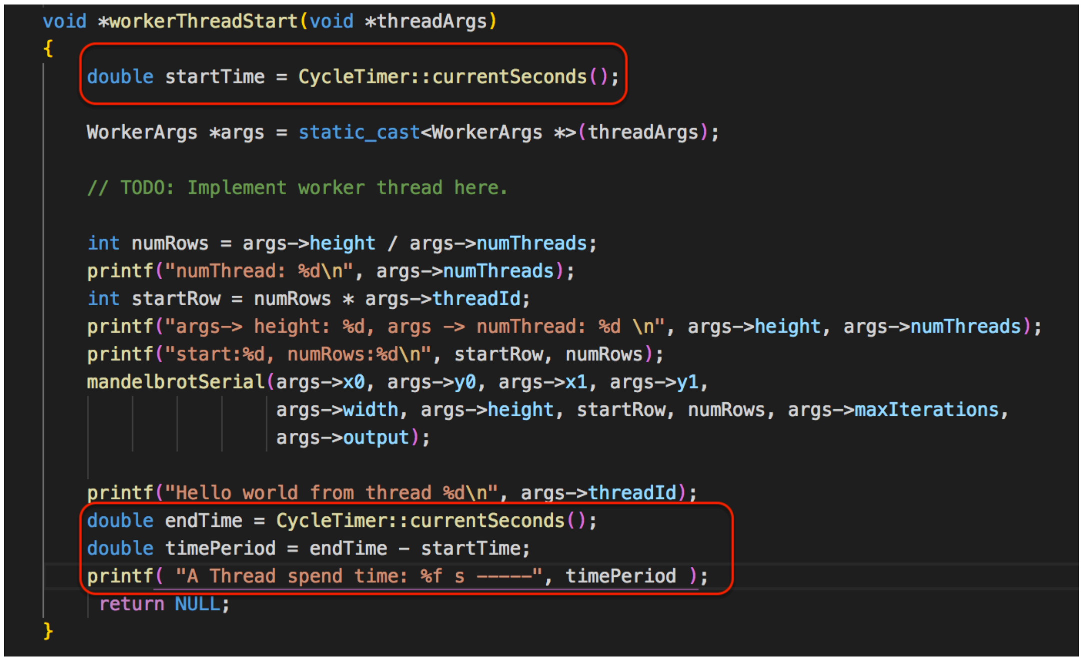
Figure 7(3): running program in 8 threads for View 2



```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 16 -v 2
[mandelbrot serial]:            [214.218] ms
Wrote image file mandelbrot-serial.ppm
Numthread:16
Hello world from thread 15
Hello world from thread 6
Hello world from thread 5
Hello world from thread 11
Hello world from thread 10
Hello world from thread 12
Hello world from thread 14
Hello world from thread 9
Hello world from thread 4
Hello world from thread 8
Hello world from thread 3
Hello world from thread 13
Hello world from thread 7
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
Numthread:16
Hello world from thread 15
Hello world from thread 6
Hello world from thread 5
Hello world from thread 11
Hello world from thread 10
Hello world from thread 12
Hello world from thread 14
Hello world from thread 9
Hello world from thread 4
Hello world from thread 8
Hello world from thread 13
Hello world from thread 7
Hello world from thread 3
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
Numthread:16
Hello world from thread 15
Hello world from thread 6
Hello world from thread 5
Hello world from thread 11
Hello world from thread 10
Hello world from thread 12
Hello world from thread 14
Hello world from thread 9
Hello world from thread 4
Hello world from thread 8
Hello world from thread 13
Hello world from thread 7
Hello world from thread 3
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
Numthread:16
Hello world from thread 15
Hello world from thread 6
Hello world from thread 5
Hello world from thread 11
Hello world from thread 10
Hello world from thread 12
Hello world from thread 14
Hello world from thread 9
Hello world from thread 4
Hello world from thread 8
Hello world from thread 13
Hello world from thread 7
Hello world from thread 3
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
Numthread:16
Hello world from thread 15
Hello world from thread 6
Hello world from thread 5
Hello world from thread 11
Hello world from thread 10
Hello world from thread 12
Hello world from thread 14
Hello world from thread 9
Hello world from thread 4
Hello world from thread 8
Hello world from thread 13
Hello world from thread 3
Hello world from thread 7
Hello world from thread 2
Hello world from thread 1
Hello world from thread 0
[mandelbrot thread]:            [29.484] ms
Wrote image file mandelbrot-thread.ppm
                                (7.27x speedup from 16 threads)
[wufangm@login001 mandelbrot_threads]$
```

Figure 7(4): running program in 16 threads for View 2

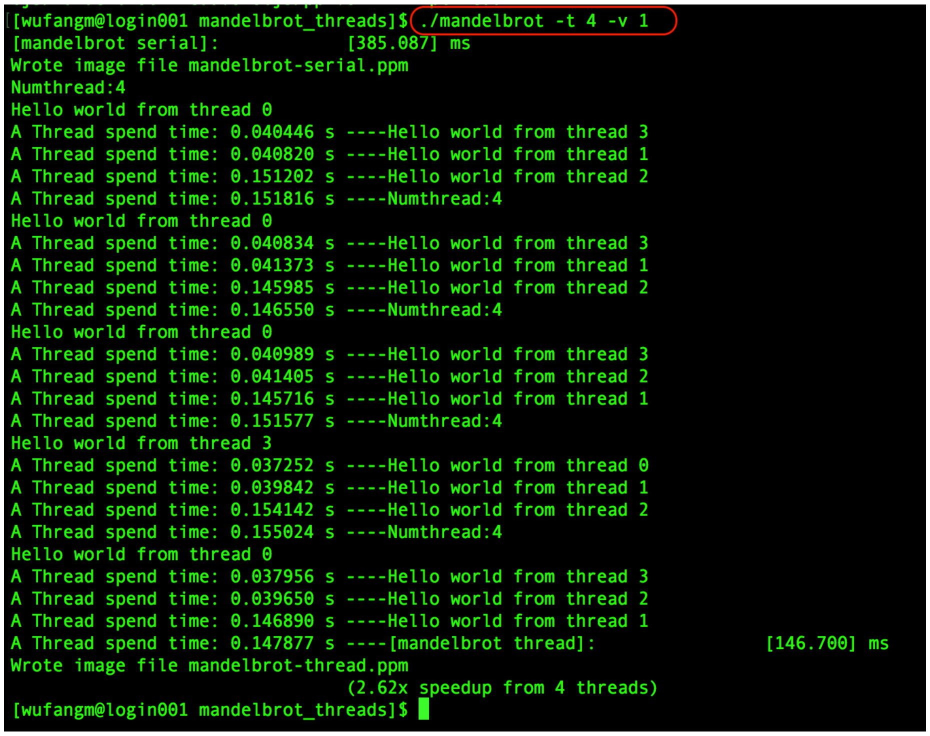
To ensure my result, I have added a clock in the function of *workerThreadStart()* to record time spend for each thread. I recorded time at the beginning and the end of calculation of this function.(the code shown in Figure 8)



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

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

    // TODO: Implement worker thread here.

    int numRows = args->height / args->numThreads;
    printf("numThread: %d\n", args->numThreads);
    int startRow = numRows * args->threadId;
    printf("args-> height: %d, args -> numThread: %d \n", args->height, args->numThreads);
    printf("start:%d, numRows:%d\n", startRow, numRows);
    mandelbrotSerial(args->x0, args->y0, args->x1, args->y1,
                     args->width, args->height, startRow, numRows, args->maxIterations,
                     args->output);

    printf("Hello world from thread %d\n", args->threadId);
    double endTime = CycleTimer::currentSeconds();
    double timePeriod = endTime - startTime;
    printf( "A Thread spend time: %f s -----", timePeriod );
     return NULL;
}
```

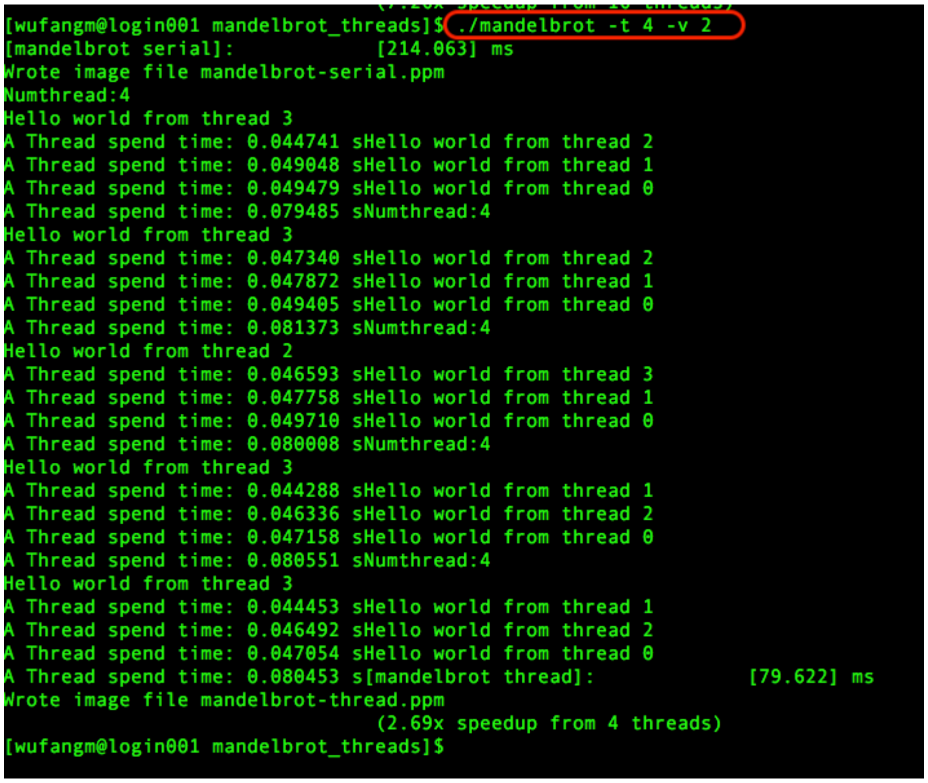
Figure 8

Then I used 4 threads to compute the View1 and View2 (shown in Figure 9). I found that there is a little difference for the time spend of calculating both View1 and View2, when I used multithreads to calculate them.



```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 4 -v 1
[mandelbrot serial]:            [385.087] ms
Wrote image file mandelbrot-serial.ppm
Numthread:4
Hello world from thread 0
A Thread spend time: 0.040446 s ----Hello world from thread 3
A Thread spend time: 0.040820 s ----Hello world from thread 1
A Thread spend time: 0.151202 s ----Hello world from thread 2
A Thread spend time: 0.151816 s ----Numthread:4
Hello world from thread 0
A Thread spend time: 0.040834 s ----Hello world from thread 3
A Thread spend time: 0.041373 s ----Hello world from thread 1
A Thread spend time: 0.145985 s ----Hello world from thread 2
A Thread spend time: 0.146550 s ----Numthread:4
Hello world from thread 0
A Thread spend time: 0.040989 s ----Hello world from thread 3
A Thread spend time: 0.041405 s ----Hello world from thread 2
A Thread spend time: 0.145716 s ----Hello world from thread 1
A Thread spend time: 0.151577 s ----Numthread:4
Hello world from thread 3
A Thread spend time: 0.037252 s ----Hello world from thread 0
A Thread spend time: 0.039842 s ----Hello world from thread 1
A Thread spend time: 0.154142 s ----Hello world from thread 2
A Thread spend time: 0.155024 s ----Numthread:4
Hello world from thread 0
A Thread spend time: 0.037956 s ----Hello world from thread 3
A Thread spend time: 0.039650 s ----Hello world from thread 2
A Thread spend time: 0.146890 s ----Hello world from thread 1
A Thread spend time: 0.147877 s ----[mandelbrot thread]:            [146.700] ms
Wrote image file mandelbrot-thread.ppm
                                (2.62x speedup from 4 threads)
[wufangm@login001 mandelbrot_threads]$
```

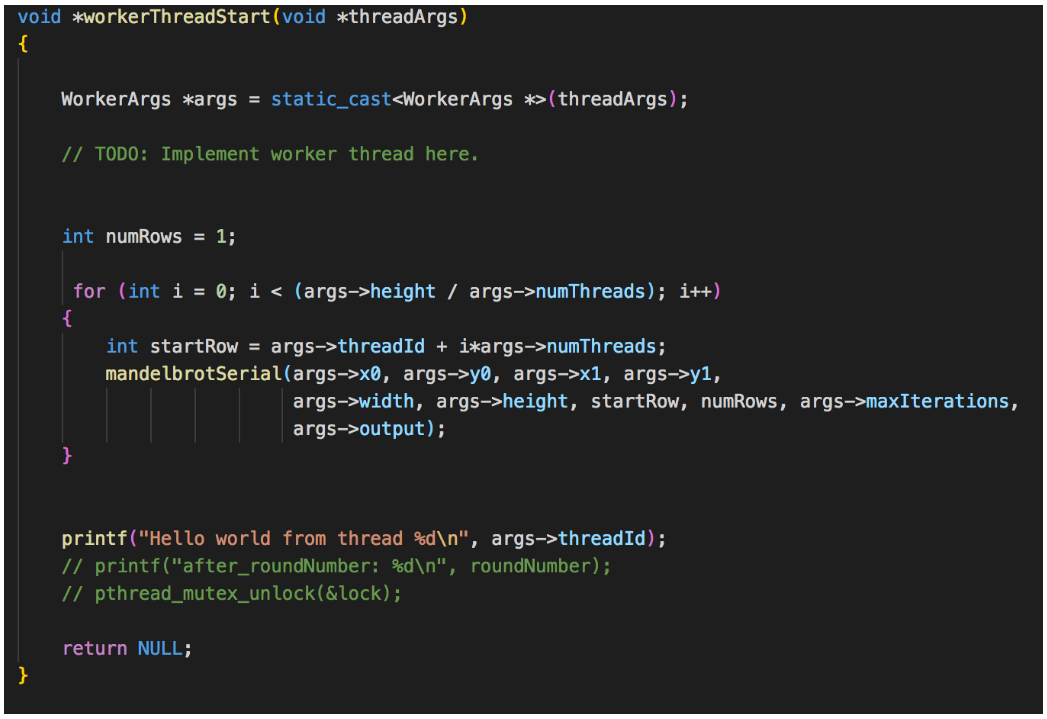
Figure 9(1): recording the calculation time for View1



```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 4 -v 2
[mandelbrot serial]:            [214.063] ms
Wrote image file mandelbrot-serial.ppm
Numthread:4
Hello world from thread 3
A Thread spend time: 0.044741 sHello world from thread 2
A Thread spend time: 0.049048 sHello world from thread 1
A Thread spend time: 0.049479 sHello world from thread 0
A Thread spend time: 0.079485 sNumthread:4
Hello world from thread 3
A Thread spend time: 0.047340 sHello world from thread 2
A Thread spend time: 0.047872 sHello world from thread 1
A Thread spend time: 0.049405 sHello world from thread 0
A Thread spend time: 0.081373 sNumthread:4
Hello world from thread 2
A Thread spend time: 0.046593 sHello world from thread 3
A Thread spend time: 0.047758 sHello world from thread 1
A Thread spend time: 0.049710 sHello world from thread 0
A Thread spend time: 0.080008 sNumthread:4
Hello world from thread 3
A Thread spend time: 0.044288 sHello world from thread 1
A Thread spend time: 0.046336 sHello world from thread 2
A Thread spend time: 0.047158 sHello world from thread 0
A Thread spend time: 0.080551 sNumthread:4
Hello world from thread 3
A Thread spend time: 0.044453 sHello world from thread 1
A Thread spend time: 0.046492 sHello world from thread 2
A Thread spend time: 0.047054 sHello world from thread 0
A Thread spend time: 0.080453 s[mandelbrot thread]:            [79.622] ms
Wrote image file mandelbrot-thread.ppm
                                (2.69x speedup from 4 threads)
[wufangm@login001 mandelbrot_threads]$
```

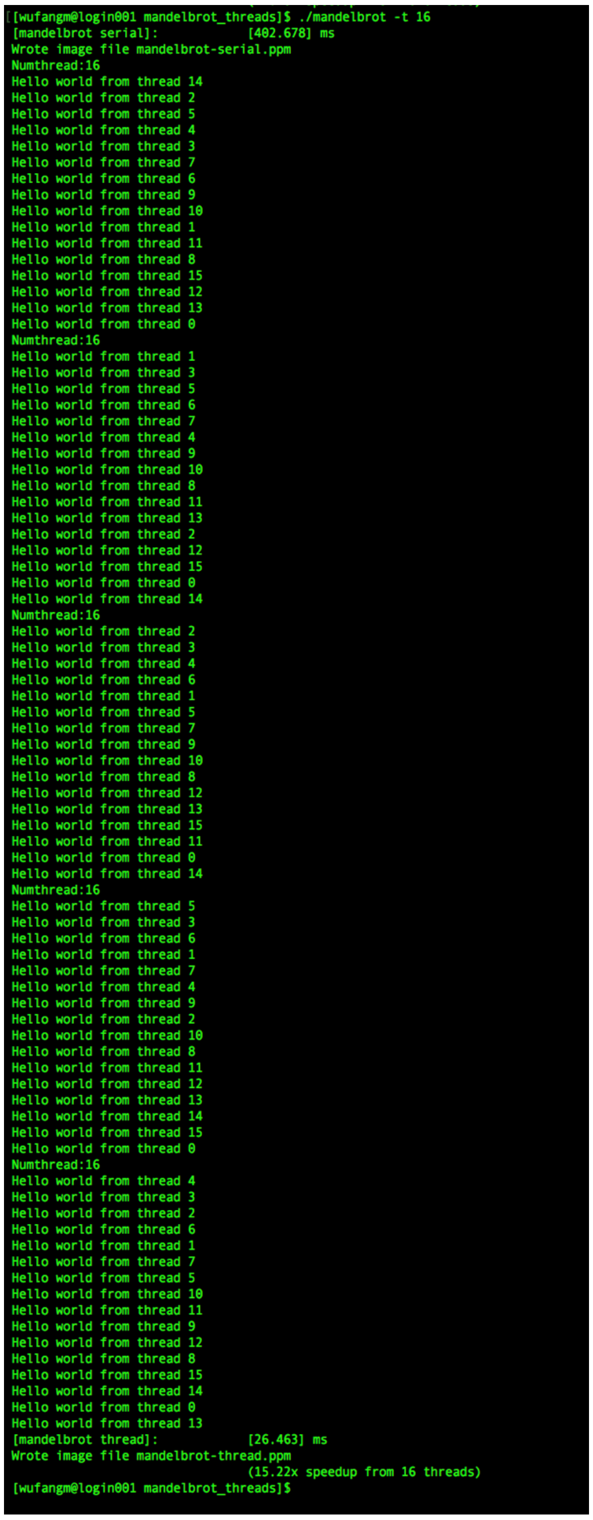
Figure 9(2): recording the calculation time for View2

For the part 4 in this problem, I have changed the strategy for computing the image. Instead of evenly separating the image to the number of threads, I choose to let each thread just calculate a single rows in each time, where the whole thread system will iterate many times during the process of calculating the image(the code shown in Figure 10). When I run this program by using 16 threads in this strategy, it could improve nearly 15 times compared with the sequential way(shown in Figure 11).



```
void *workerThreadStart(void *threadArgs)
{

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

    // TODO: Implement worker thread here.


    int numRows = 1;

     for (int i = 0; i < (args->height / args->numThreads); i++)
    {
        int startRow = args->threadId + i*args->numThreads;
        mandelbrotSerial(args->x0, args->y0, args->x1, args->y1,
                         args->width, args->height, startRow, numRows, args->maxIterations,
                         args->output);
    }


    printf("Hello world from thread %d\n", args->threadId);
    // printf("after_roundNumber: %d\n", roundNumber);
    // pthread_mutex_unlock(&lock);

    return NULL;
}
```

Figure 10

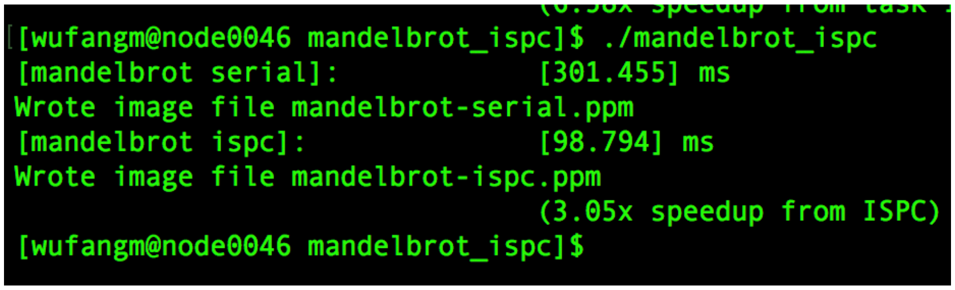


```
[wufangm@login001 mandelbrot_threads]$ ./mandelbrot -t 16
[mandelbrot serial]:            [402.678] ms
Wrote image file mandelbrot-serial.ppm
Numthread:16
Hello world from thread 14
Hello world from thread 2
Hello world from thread 5
Hello world from thread 4
Hello world from thread 3
Hello world from thread 7
Hello world from thread 6
Hello world from thread 9
Hello world from thread 10
Hello world from thread 1
Hello world from thread 11
Hello world from thread 8
Hello world from thread 15
Hello world from thread 12
Hello world from thread 13
Hello world from thread 0
Numthread:16
Hello world from thread 1
Hello world from thread 3
Hello world from thread 5
Hello world from thread 6
Hello world from thread 7
Hello world from thread 4
Hello world from thread 9
Hello world from thread 10
Hello world from thread 8
Hello world from thread 11
Hello world from thread 13
Hello world from thread 2
Hello world from thread 12
Hello world from thread 15
Hello world from thread 0
Hello world from thread 14
Numthread:16
Hello world from thread 2
Hello world from thread 3
Hello world from thread 4
Hello world from thread 6
Hello world from thread 1
Hello world from thread 5
Hello world from thread 7
Hello world from thread 9
Hello world from thread 10
Hello world from thread 8
Hello world from thread 12
Hello world from thread 13
Hello world from thread 15
Hello world from thread 11
Hello world from thread 0
Hello world from thread 14
Numthread:16
Hello world from thread 5
Hello world from thread 3
Hello world from thread 6
Hello world from thread 1
Hello world from thread 7
Hello world from thread 4
Hello world from thread 9
Hello world from thread 2
Hello world from thread 10
Hello world from thread 8
Hello world from thread 11
Hello world from thread 12
Hello world from thread 13
Hello world from thread 14
Hello world from thread 15
Hello world from thread 0
Numthread:16
Hello world from thread 4
Hello world from thread 3
Hello world from thread 2
Hello world from thread 6
Hello world from thread 1
Hello world from thread 7
Hello world from thread 5
Hello world from thread 10
Hello world from thread 11
Hello world from thread 9
Hello world from thread 12
Hello world from thread 8
Hello world from thread 15
Hello world from thread 14
Hello world from thread 0
Hello world from thread 13
[mandelbrot thread]:            [26.463] ms
Wrote image file mandelbrot-thread.ppm
                                (15.22x speedup from 16 threads)
[wufangm@login001 mandelbrot_threads]$
```

Figure 11

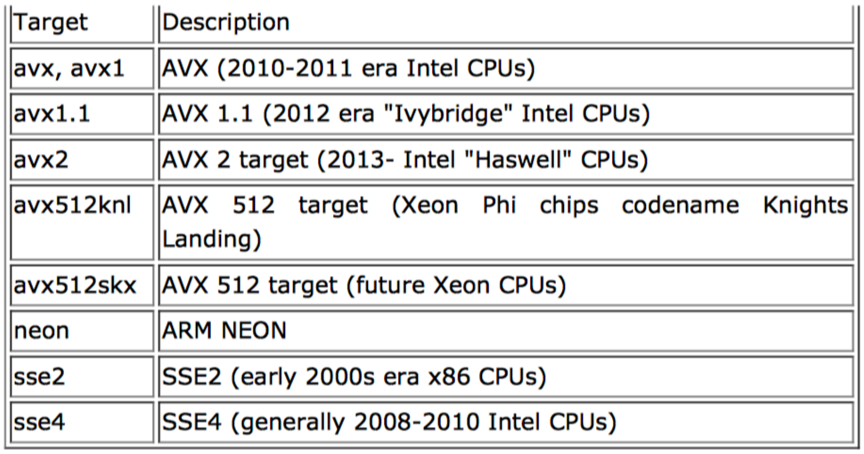
Part 3: parallelize with ISPC:

1. In this part, I use the 16 CPU cores in e5-2665 to run the mandelbrot.ispc. After run the program, we will get information below(Figure 12). In the result, we can find that ISPC with task can improve 3.05X compared with sequential computation. In addition, E5-2665 uses both support AVS and SEE configuration. In the Table 1, we can see the target and description for each configuration. In my opinion, the ideal speed up I expect would be 64X, which is a huge different with actual result.



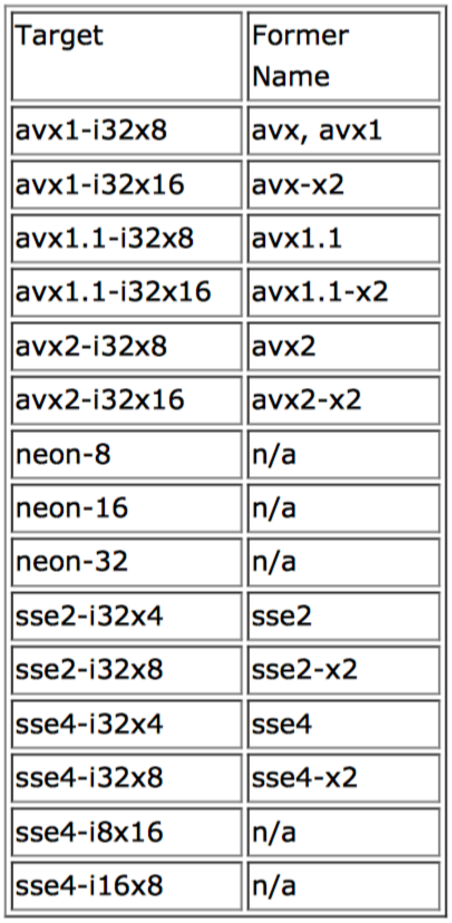
```
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc
[mandelbrot serial]:            [301.455] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [98.794] ms
Wrote image file mandelbrot-ispc.ppm
                                (3.05x speedup from ISPC)
[wufangm@node0046 mandelbrot_ispc]$
```

Figure 12



Target	Description
avx, avx1	AVX (2010-2011 era Intel CPUs)
avx1.1	AVX 1.1 (2012 era "Ivybridge" Intel CPUs)
avx2	AVX 2 target (2013- Intel "Haswell" CPUs)
avx512knl	AVX 512 target (Xeon Phi chips codename Knights Landing)
avx512skx	AVX 512 target (future Xeon CPUs)
neon	ARM NEON
sse2	SSE2 (early 2000s era x86 CPUs)
sse4	SSE4 (generally 2008-2010 Intel CPUs)

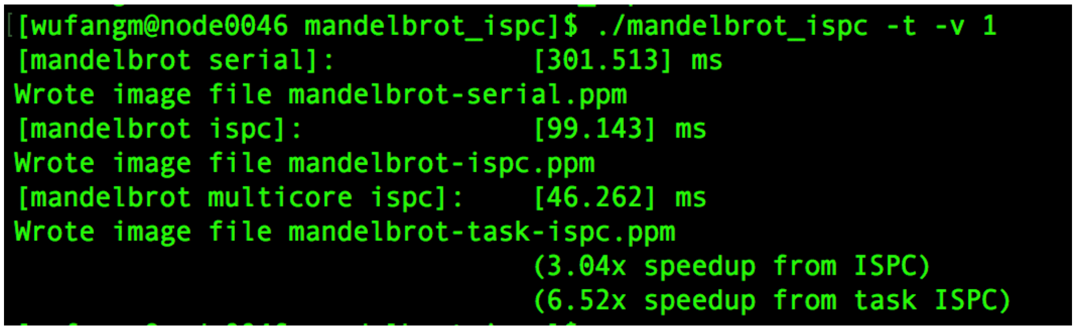
Table1 (1)



Target	Former Name
avx1-i32x8	avx, avx1
avx1-i32x16	avx-x2
avx1.1-i32x8	avx1.1
avx1.1-i32x16	avx1.1-x2
avx2-i32x8	avx2
avx2-i32x16	avx2-x2
neon-8	n/a
neon-16	n/a
neon-32	n/a
sse2-i32x4	sse2
sse2-i32x8	sse2-x2
sse4-i32x4	sse4
sse4-i32x8	sse4-x2
sse4-i8x16	n/a
sse4-i16x8	n/a

Table1 (2)

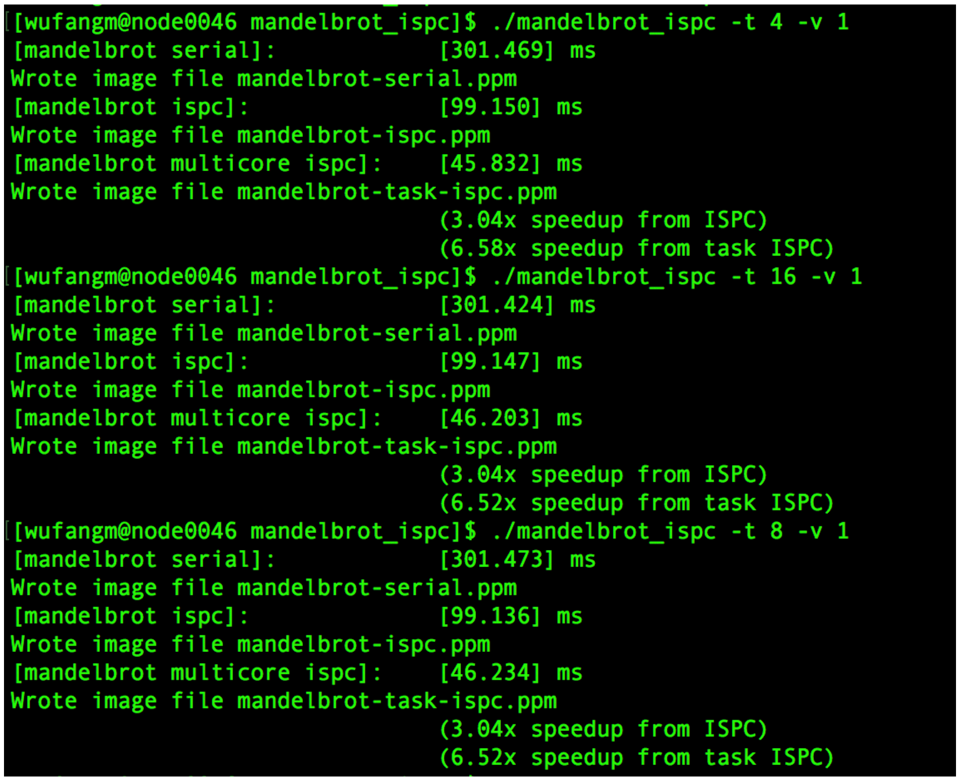
2. After I run mandelbrot.ispc with task, we can see the information below(Figure 13). The ISPC with task is twice time of the ISCP without task when compute the View1.



```
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc -t -v 1
[mandelbrot serial]:            [301.513] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [99.143] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [46.262] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (3.04x speedup from ISPC)
                                (6.52x speedup from task ISPC)
```

Figure 13

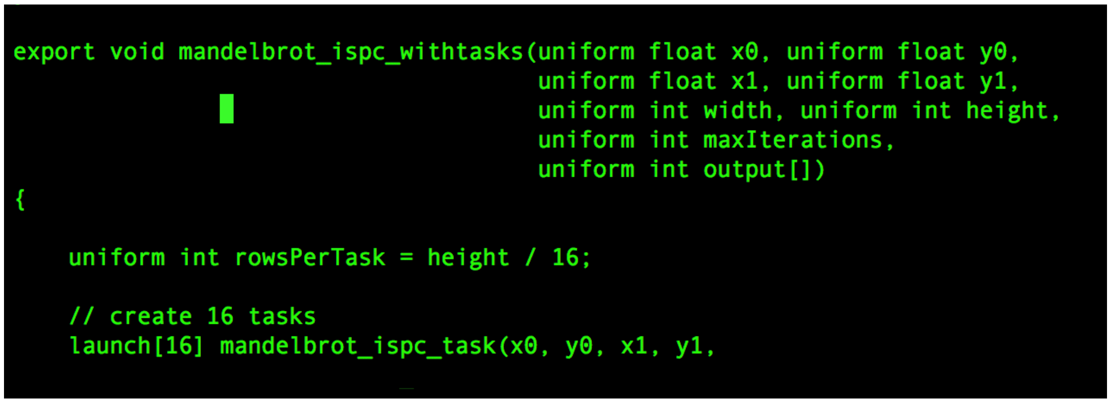
In the program, if I just change the task number in command line, the operation speed of ISPC with task and ISPC without task would just change a little compare with only adding "--task"(shown in Figure 14).



```
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc -t 4 -v 1
[mandelbrot serial]:            [301.469] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [99.150] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [45.832] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (3.04x speedup from ISPC)
                                (6.58x speedup from task ISPC)
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc -t 16 -v 1
[mandelbrot serial]:            [301.424] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [99.147] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [46.203] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (3.04x speedup from ISPC)
                                (6.52x speedup from task ISPC)
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc -t 8 -v 1
[mandelbrot serial]:            [301.473] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [99.136] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [46.234] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (3.04x speedup from ISPC)
                                (6.52x speedup from task ISPC)
```

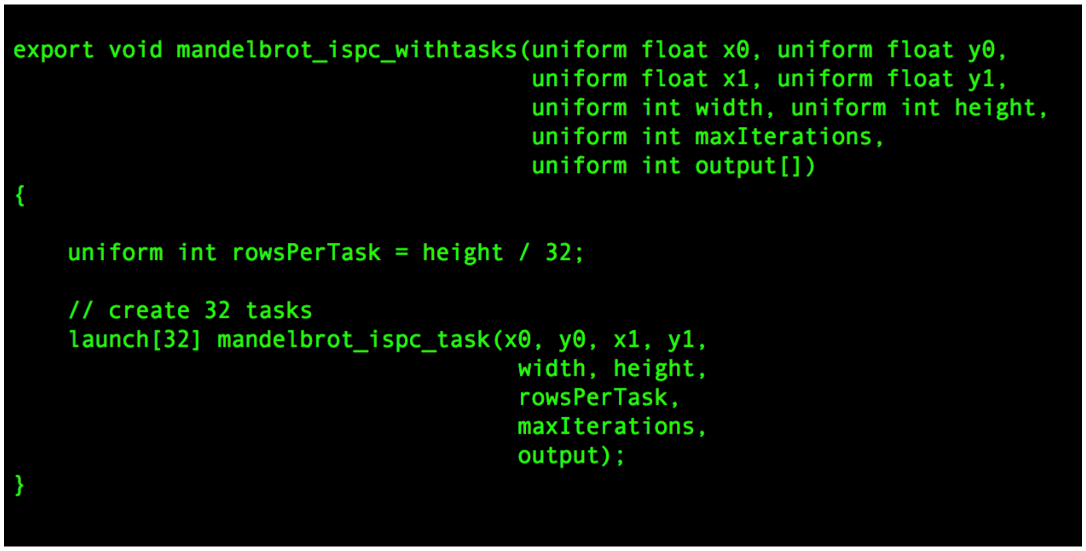
Figure 14

For the second question, I have changed the task number to 16 and 32(shown in Figure 15). After I observed the performance, I find that the more task ISPC have, the faster speed will performance. In 16 tasks, ISPC with 16 tasks is almost 4X faster than ISPC with 2 tasks(shown in Figure 15(1)). In 32 tasks, ISPC with 16 tasks is 5X faster than ISPC with 2 tasks(shown in Figure 15(2)).



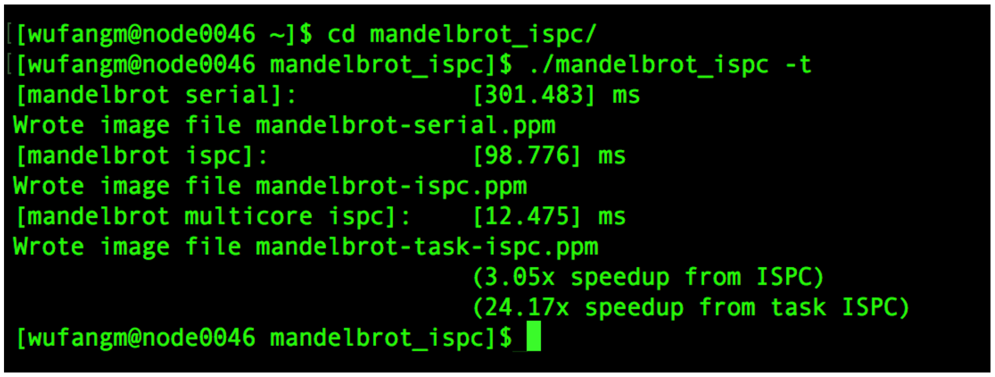
```
export void mandelbrot_ispc_withtasks(uniform float x0, uniform float y0,
                                      uniform float x1, uniform float y1,
                                      uniform int width, uniform int height,
                                      uniform int maxIterations,
      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

    uniform int rowsPerTask = height / 16;

    // create 16 tasks
    launch[16] mandelbrot_ispc_task(x0, y0, x1, y1,
```

Figure 15(1): changing the number of task to 16



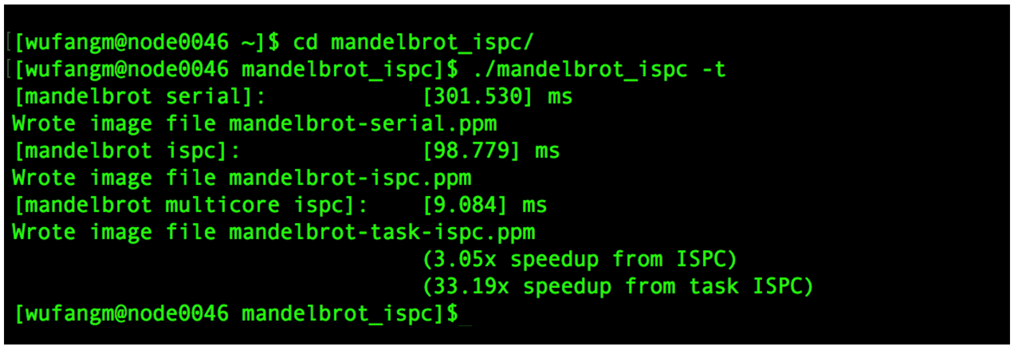
```
export void mandelbrot_ispc_withtasks(uniform float x0, uniform float y0,
                                      uniform float x1, uniform float y1,
                                      uniform int width, uniform int height,
                                      uniform int maxIterations,
      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

    uniform int rowsPerTask = height / 32;

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

Figure 15(2): changing the number of task to 32



```
[wufangm@node0046 ~]$ cd mandelbrot_ispc/
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc -t
[mandelbrot serial]:            [301.483] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [98.776] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [12.475] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (3.05x speedup from ISPC)
                                (24.17x speedup from task ISPC)
[wufangm@node0046 mandelbrot_ispc]$
```

Figure 16(1): The performance when changing the number of tasks to 16



```
[wufangm@node0046 ~]$ cd mandelbrot_ispc/
[wufangm@node0046 mandelbrot_ispc]$ ./mandelbrot_ispc -t
[mandelbrot serial]:            [301.530] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [98.779] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [9.084] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (3.05x speedup from ISPC)
                                (33.19x speedup from task ISPC)
[wufangm@node0046 mandelbrot_ispc]$
```

Figure 16(2): The performance when changing the number of tasks to 32