Evaluation Report

# The Context

This assignment had two parts: 1) compute a portion of Mandelbrot set with a single program (mandel.c) using multiple threads to compute a single image; and 2) compute multiple versions (50) of the Mandelbrot set (mandelseries.c), from a single starting point of 2 scale units down to the scale unit I chose as my main image, using multiple single-threaded mandel.c processes.

# The Experiments

The experiments consisted of:

* running the mandelseries program, with our chosen mandel configuration, using 1, 2, 3, 4, 5, and 10 simultaneous processes, 1 time overall; and,
* running the below two mandel commands using 1, 2, 3, 4, 5, 10, and 50 threads, five times each:
  + mandel -x -.5 -y -.5 -s 1 -m 2000 -n #
  + mandel -x 0.2869325 -y 0.0142905 -s .000001 -W 1024 -H 1024 -m 1000 -n #

The purpose of these experiments is to hypothesize the optimal number of CPU-bound threads and processes that can run at any given time on the UTA Omega server.

For the mandelseries runs, the mandel configuration used was:  
**Starting:** mandel -s 2 -y -1.03265 -m 7000 -x -.163013 -W 600 -H 600  
**Final:** mandel -s .000025 -y -1.03265 -m 7000 -x -.163013 -W 600 -H 600

# The Results – mandelseries #

Increasing the number of simultaneous mandel processes proportionately decreased the time it took to complete the set of 50 Mandelbrot images, as shown below:

|  |  |
| --- | --- |
| # processes | execution time (usec) |
| 1 | 128281106 |
| 2 | 66313512 |
| 3 | 45697991 |
| 4 | 35686193 |
| 5 | 35592314 |
| 10 | 34877131 |

See the graph on the next page.

Once the 4-processes mark was reached, any additional simultaneous processes provided little benefit. This indicates Omega only has 4 processing cores available to me for CPU-bound work. This doesn’t necessarily indicate the number of total cores is 4, but only that 4 are usable by me.

There were no negative side-effects in this scenario. This was a bit surprising as I half-expected the execution time to increase ever so slightly due to context-switching between processes when using, in this experiment, 10 simultaneous processes. Since I did not observe this, I deduce the context-switching is fast and efficient enough to not be detrimental to this particular process in this particular scenario. Another possibility is that my processes were being throttled.

# The Results – mandel A

For the first mandel experiment:  
mandel -x -.5 -y -.5 -s 1 -m 2000 -n #  
I had these results:

|  |  |
| --- | --- |
| # threads | execution time (usec) |
| 1 | 1453392 |
| 2 | 1314132 |
| 3 | 942662 |
| 4 | 661122 |
| 5 | 626655 |
| 10 | 427797 |
| 50 | 385586 |

The execution time decreased every time more threads were added. The decrease seemed to be linear.

These results lend more credence to the possibility that my mandelseries’ mandel processes *were* being throttled, but my threads in a single process weren’t subject to the same throttling. If I had reached the max number CPU cores available to me at 4, then I would have expected the mandel execution time graph to closely match the curve of the mandelseries graph, but it doesn’t.

It’s also very possible the amount of work being done by each thread became so minimal that adding new threads didn’t provide a big-enough decrease in the amount of work. This conclusion becomes more likely once the second mandel experiment, discussed below, comes into play.

For this mandel experiment, I would say the optimal number of threads is 10, since jumping from 1-10 provided a much more marked decrease in execution time, as opposed to going from 10 to 50, which had very little impact.

# The Results – mandel B

For the second mandel experiment:  
mandel -x 0.2869325 -y 0.0142905 -s .000001 -W 1024 -H 1024 -m 1000 -n #  
I had these results:

|  |  |
| --- | --- |
| # threads | execution time (usec) |
| 1 | 3555131 |
| 2 | 1878129 |
| 3 | 1396045 |
| 4 | 1039173 |
| 5 | 1048288 |
| 10 | 978115 |
| 50 | 930931 |

This graph more closely matches the mandelseries graph – once the number of threads reached 4, there was very little time benefit, or none at all, to adding more threads. The decrease in time was also proportional, up until the 4-thread point.

This makes me think I either maxed-out the number of available processor cores, or was throttled.

This particular mandel computation was also much more computationally-intensive than the first one. Spreading the work across more threads for something that does not require much work seems to have more benefits than spreading a higher amount of work across the same number of threads.

The optimal number of threads in this scenario appears to be 4, since adding more threads was either slightly detrimental, probably due to context-switching, or provided very little benefit.