Nikolas Dean Brooks
Operating Systems Principles
Spring 2017

Project III

**Overview**

The purpose of the experiments in this project were to explore the impact that parallel computing (as carried out with varying number of threads and simultaneous processes) has on a relatively labor-intensive task.

*mandel* is a program that generates a Mandelbrot image, *display.bmp*, potentially using multiple threads to do so. *mandelmovie* is a program that runs *mandel* several times, potentially through multiple forked processes. For each process in *mandelmovie*, the parameters that *mandel* takes in are modified slightly to generate several images that will differ slightly from one another.

The experiments following the *Overview* were performed locally using Bash on Ubuntu on Windows (Ubuntu on Windows 10). My machine is a Lenovo ThinkPad E550. It uses a 64-bit architecture, Intel i7-5500U CPU @ 2.4 GHz, and 8GB of RAM.

*Running mandel*:

```
>> ls
bitmap.c bitmap.h Makefile mandel.c mandelmovie.c
>> make
~makes things~
>> ./mandel
~generates mandel.bmp using default values~
>> ./mandel -x -1.249 -y -.03438 -s .000001 -m 4000 -W 750 -H 750
~generates pretty mandel.bmp using the provided non-default values~
>> ./mandel -x -1.249 -y -.03438 -s .000001 -m 4000 -W 750 -H 750 -n 4
~quickly generates similar mandel.bmp using 4 threads~
>> display mandel.bmp
~displays mandel.bmp in popup window using ImageMagick and X-forwarding~
```

*Running mandelmovie*:

```
>> make
~makes things if things haven't already been made~
>> ./mandelmovie
~runs mandel 50 times to create 50 slightly different images. It does this one process at a time as default~
>> ./mandelmovie 3
~using mandel, makes 50 images 3 processes at a time, rather than the default one process at any time~
```

*Creating and playing mandel.mpg*:

To generate the video on the student machine:

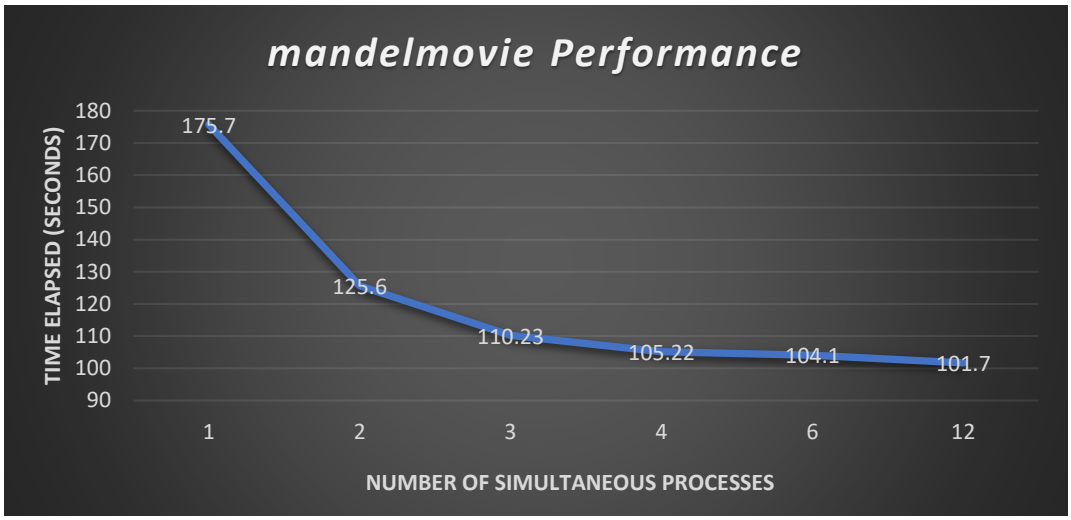
```
>> ffmpeg -I mandel%d.bmp mandel.mpg
~generates .mpg of mandel.bmp files~
>> ffplay mandel.mpg
~plays mandel.mpg (really choppily) on the student machine using X-forwarding~
```

To generate the video on my local machine:

```
>> animate "mandel%d.bmp[0-50]"
```

~displays .gif of *mandel.bmp* files strung together~
>> animate -delay 5 “mandel%d.bmp[0-50]”
~displays a less alarming .gif of the strung together *mandel.bmp* files~

## Mandelmovie Performance



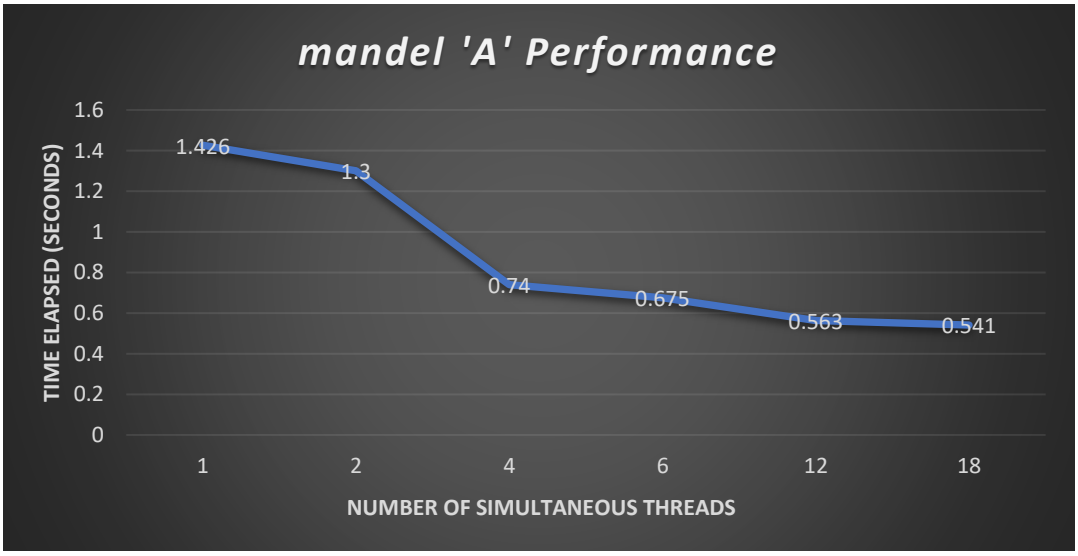

## Mandelmovie Analysis

We see that for smaller number of simultaneous processes allowed, N, there are great gains to be made in time as N increases. As N increases, diminishing returns are yielded.

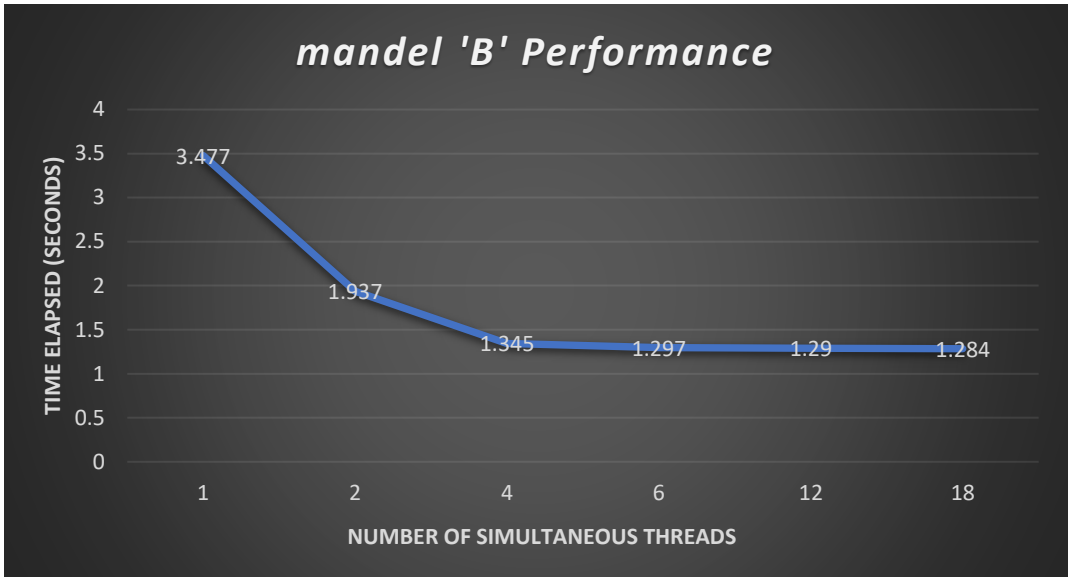
This is because a proportionately larger amount of time and effort is spent switching from process to process rather than completing the labor for each process. At a smaller N, this switching has a minimal impact (there’s less switching to be done when the processes are limited), and at a higher N, it significantly reduces the speed increase we saw at the higher N.

## Mandel Performance

**A:**



**B:**




**Mandel Analysis**

As M (the number of simultaneous threads) increases, there are diminishing returns in how much time is saved by using more threads. With a larger task like task *B*, the diminishing returns are extremely evident. The optimal number of threads (at least for my local machine) seems to be 4.

The curves for A and B have a different shape because of how the work is distributed across threads for those two tasks. In task A, certain threads will tend to do more work than others because of the part of the zoomed out picture that falls under their “job description”, resulting in a less smooth graph of time elapsed.