Task 3:

I converted the 100 size and all the code worked as it should. I then increased the size to 1000 and I kept getting a segmentation fault (core dump). I reduced the size to 800 and it worked, I couldn't find the reason for the core dump but it starts at 836. 835 runs fine but 836 fails. I did my number with 800 size matrices.

The sequential and the 1 thread was about the same on most accounts. I think the overhead of the multithread setup cost a few milliseconds. I don't know if the Pthreads was setup correctly but it seemed to be when ran. I may have only reduced it by a few milliseconds for timing but openMP was significantly different and was noticeably faster the more threads that were added. I think the segmentation issue may have caused some of the timing issues but I don't know and can't tell.

The numbers shown below for OpenMP are the static scheduling. The dynamic scheduling had a .4ms reduction typically but did vary sometimes to over. It seemed like the processor heatsink spun up every time dynamic was used. Weird physical change to the laptop I'm using, don't know if they are linked for the architecture is heating up more when managing the threads more or if was a timing, it seemed to be linked but could just be happenstance.

		Matrix	Multiplication	(for matrix	size 800)	
	Sequential	1 thread	2 threads	5 threads	7 threads	9 threads
Time	6.741ms					
Pthread		7.640ms	7.224ms	7.202ms	7.360ms	7.181ms
OpenMP		6.441ms	4.068ms	3.190ms	3.220ms	3.167
	Trapezoidal	Rule	Integration	(for interval	Size	1,000,000)
	Sequential	OpenMP 1 thread	OpenMP 2 threads	OpenMP 5 threads	OpenMP 7 threads	OpenMP 9 threads
Time	7.866 sec	8.183 sec	5.202 sec	4.161 sec	4.130 sec	4.116 sec

Task 4:

The jump in time from sequential to 1 thread was odd, I repeated the process several times and got the same result. Sequential vs 1 thread with openMP always shows sequential to be faster. I presume it's the overhead created by openMP compiling and not being used. The reduction in time for additional threads seemed to drop off like the speed up is a sqrt(x) graph and not linear. That would be a good example of the speed up graph from earlier in the year. Hardware and software have limits and I approached an echo of those limits with my algorithm. I did not that 5,7, and 9 threads seemed to be a few milliseconds off and sometimes had the same runtime. I wonder if that is because the performance is limited at that number of threads by my processor emulation additional threads than 5 or if the processor is maxed calculating PI.