**Part 1**

The overall graphs are U shaped. As array sizes increase, the omega converges to a point near 1.9 as opposed to 1.5.

**Part 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Size** | **SOR** | **SOR\_ji** | **SOR\_blocked (Block size -8)** |
| 3002 | 514 | 449 | 461 |
| 1002 | 937 | 1403 | 662 |
| 600 | 666 | 1247 | 976 |

Blocking does improve performance for the larger array sizes, however this improvement is not as considerable as for mmm. Furthermore, blocking is ineffective for the smaller array sizes. Unsure about why not effective as mmm.

**Part 3**

It can be seen that after about a length of 500, the threaded functions start to become more optimal than the regular function from the very first graph. When 30 iterations are used of delta 20, the breakeven point can be estimated more accurately to be about length of 380.

**Part 4**

It can be seen that the multithreaded version of SOR with 16 threads is more optimal than the version with 4 threads. This difference starts to become evident after array length of about 60. It can also be seen that the multithreaded version with 4 threads is less optimal than the Serial SOR version. However, the SOR version with 16 threads is overall more optimal than the serial version of the SOR code, except for a few anomaly points.

Again, it can be seen that the SOR version using 16 threads is more optimal than the version using 4 threads, even with a larger array length of 1000. When both the multithreaded versions are compared to the serialized version it can be seen that as the array length approaches 1000, the multithreaded versions are a lot more efficient.