Mandelbrot Set with MPI: Analysis

W Truong

1. Summary

Mandelbrot set is a mathematical equation using complex numbers to generate fractals. Given a serial version of the Mandelbrot set and proposals for two statically parallel Mandelbrot sets (cyclic and block implementation), three parallelized Mandelbrot sets were implemented and executed: cyclic, block, and master-worker (aka master-slave, or MW).

This document is organized as follows: using the results for 1000x1000, 2000x2000, and 3000x3000 images for 40/80/120/160 processes from testing these implementations, tables are shown in the section 2. Section 2 shows graphs for speedup and efficiency that were made from the tables, where the graphs show the trends in speedup and efficiency as the number of processes increases for an image size. Section 4 compares the three implementations in terms of performance and scalability: specifically cyclic and block implementations with each other, cyclic and block with the master-worker model.

2. Test results

1000x1000, 2000x2000, and 3000x3000 were used to test the performance of the three models (cyclic, block, and MW) when compared to the serial Mandelbrot set using 40, 80, 120, and 160 processes on each image size. For testing with 160 processes, a 10000x10000 was tested solely for scalability when the number of processes is constant. All total times are rounded to 5 decimal places. Speedup and efficiency are rounded to 3 decimal places. The following formulas were used:

(1) Speedup = serial time/ parallel time
(2) Efficiency = speedup / number of processes
(3) Total time = communication time + computation time

Image Size	Serial Time (seconds)
1000x1000	0.47
2000x2000	1.94
3000x3000	4.48
10000x10000	51.15

Table 1*: Times for serial Mandelbrot set; this is used across all other tables for speedup.

Image Size	Total time (seconds)			Communication time (cyclic and block only)		Speedup			Efficiency		
	Cyclic	Block	MW	Cyclic	Block	Cyclic	Block	MW	Cyclic	Block	MW
1000x1000	0.02303	0.03191	0.01351	0.00938	0.03061	20.408	14.730	34.802	0.510	0.368	0.870
2000x2000	0.06395	0.12262	0.05286	0.01157	0.12256	30.337	15.822	36.702	0.758	0.396	0.918
3000x3000	0.13388	0.27511	0.12095	0.01965	0.27389	33.462	16.285	37.040	0.837	0.407	0.926

Table 2: Test results for various image sizes at 40 processes.

Image Size	Total time (seconds)			Communication time (cyclic and block only)		Speedup			Efficiency		
	Cyclic	Block	MW	Cyclic	Block	Cyclic	Block	MW	Cyclic	Block	MW
1000x1000	0.01912	0.02167	0.00807	0.01096	0.02145	24.587	21.685	58.212	0.307	0.271	0.728
2000x2000	0.06047	0.07495	0.03358	0.03052	0.07522	32.080	25.885	57.779	0.401	0.324	0.722
3000x3000	0.11445	0.17056	0.07317	0.05169	0.16958	39.145	26.266	61.228	0.489	0.328	0.765

Table 3: Test results for various image sizes at 80 processes.

Image Size	Total time (seconds)			Communication time (cyclic and block only)		Speedup			Efficiency		
	Cyclic	Block	MW	Cyclic	Block	Cyclic	Block	MW	Cyclic	Block	MW
1000x1000	0.01808	0.01568	0.00797	0.01250	0.01547	26.001	29.978	58.942	0.217	0.250	0.491
2000x2000	0.05129	0.05442	0.02196	0.02822	0.05370	37.823	35.650	88.359	0.315	0.297	0.736
3000x3000	0.08534	0.10217	0.05502	0.03952	0.10177	52.496	43.849	81.428	0.437	0.365	0.679

Table 4: Test results for various image sizes at 120 processes.

Image Size	Total time (seconds)			Communication time (cyclic and block only)		Speedup			Efficiency		
	Cyclic	Block	MW	Cyclic	Block	Cyclic	Block	MW	Cyclic	Block	MW
1000x1000	0.01324	0.01602	0.00763	0.00821	0.01582	35.490	29.338	61.607	0.222	0.183	0.385
2000x2000	0.04093	0.05713	0.01708	0.02395	0.05645	47.395	33.959	113.56	0.296	0.212	0.710
3000x3000	0.06433	0.09118	0.04206	0.02590	0.09007	69.643	49.136	106.522	0.435	0.307	0.666
10k x 10k	0.72093	1.22005	0.44256	0.30785	1.21414	70.950	41.925	115.579	0.443	0.262	0.722

Table 5*: Test results for various image sizes at 160 processes.

*: also tested with 10000x10000 for 160 processes for scalability.

3. Graphs

Using the tables from section 2, the graphs were obtained. The graphs were plotted based on an image size while scaling the number of processes (i.e. 1 graph on speedup and 1 graph on efficiency for every image size except the 10000x10000).

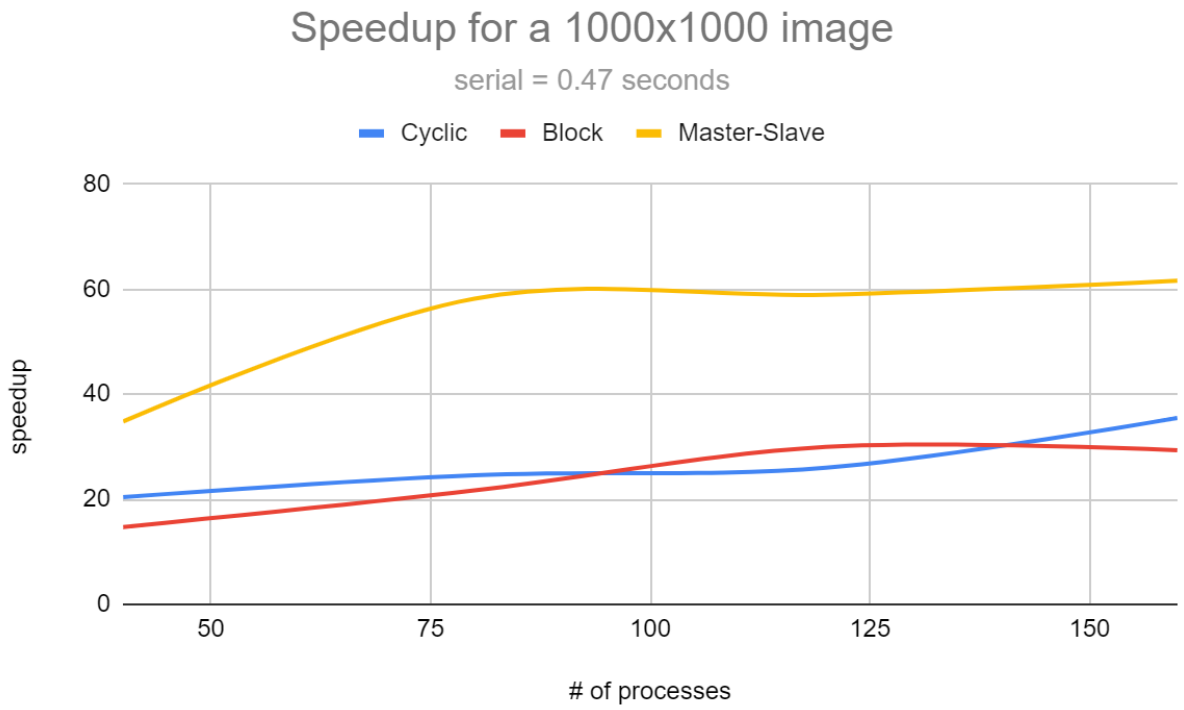

Figure 1: Speedup for a 1000x1000 image for increasing number of processes (from 40 to 160).

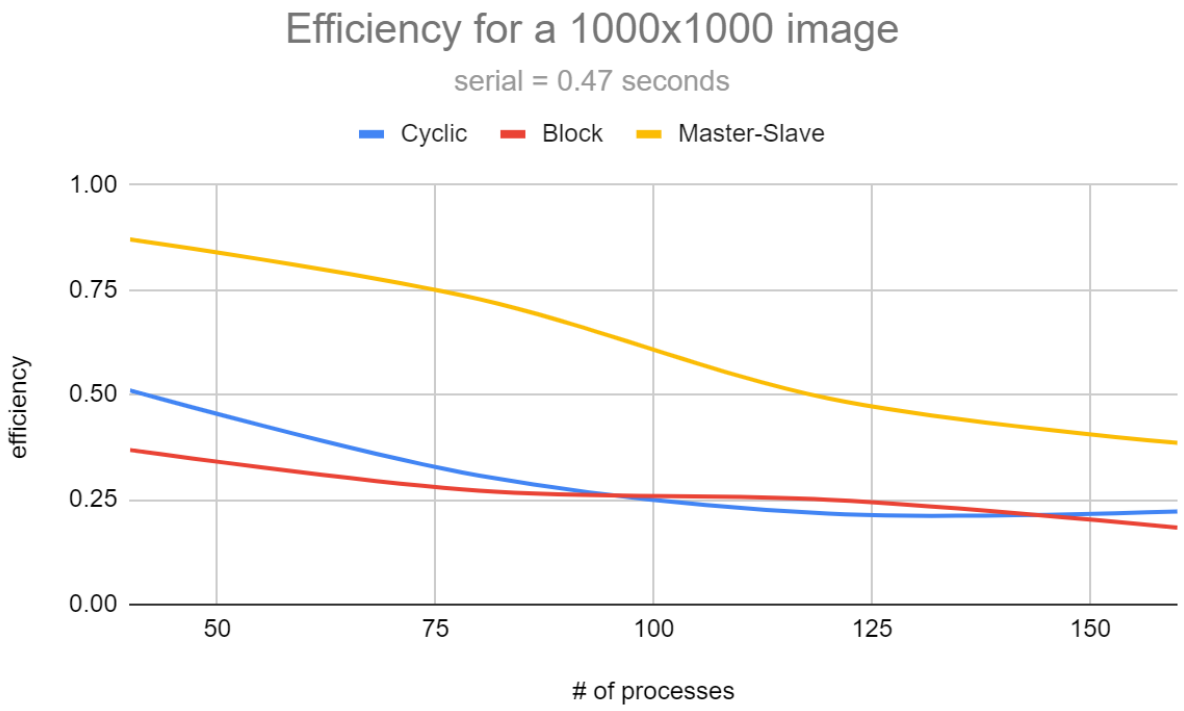

Figure 2: Efficiency for a 1000x1000 image for increasing number of processes (from 40 to 160).

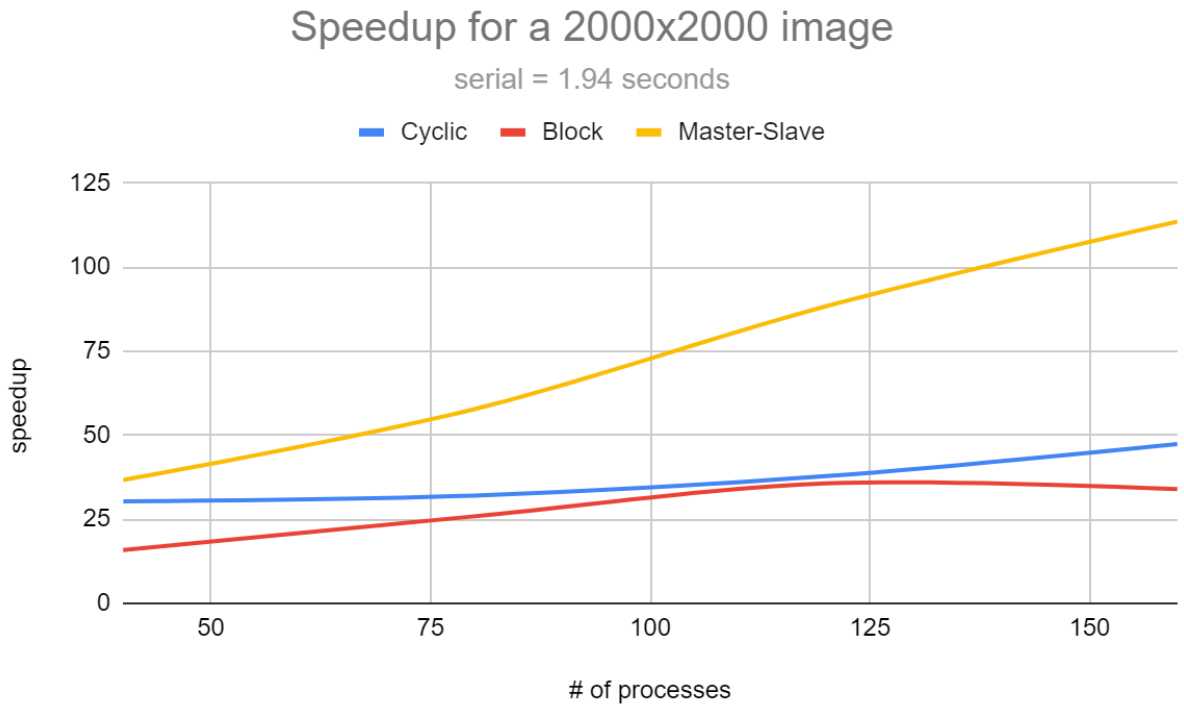

Figure 3: Speedup for a 2000x2000 image for increasing number of processes (from 40 to 160).

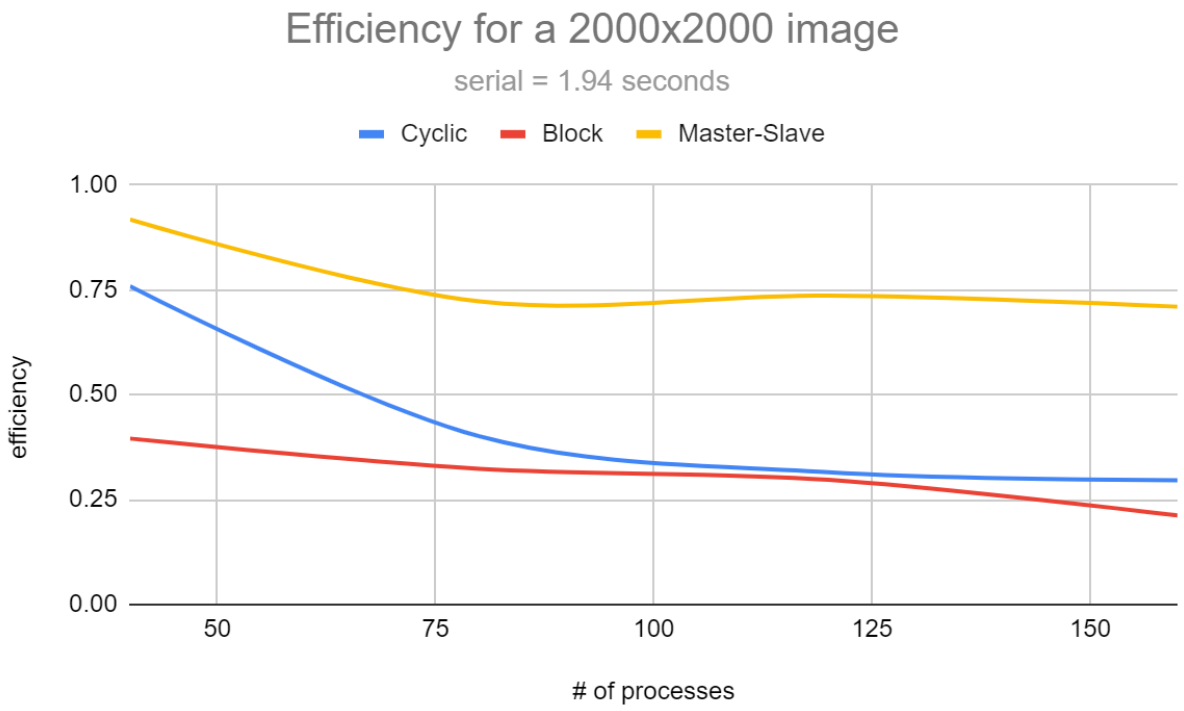

Figure 4: Efficiency for a 2000x2000 image for increasing number of processes (from 40 to 160).

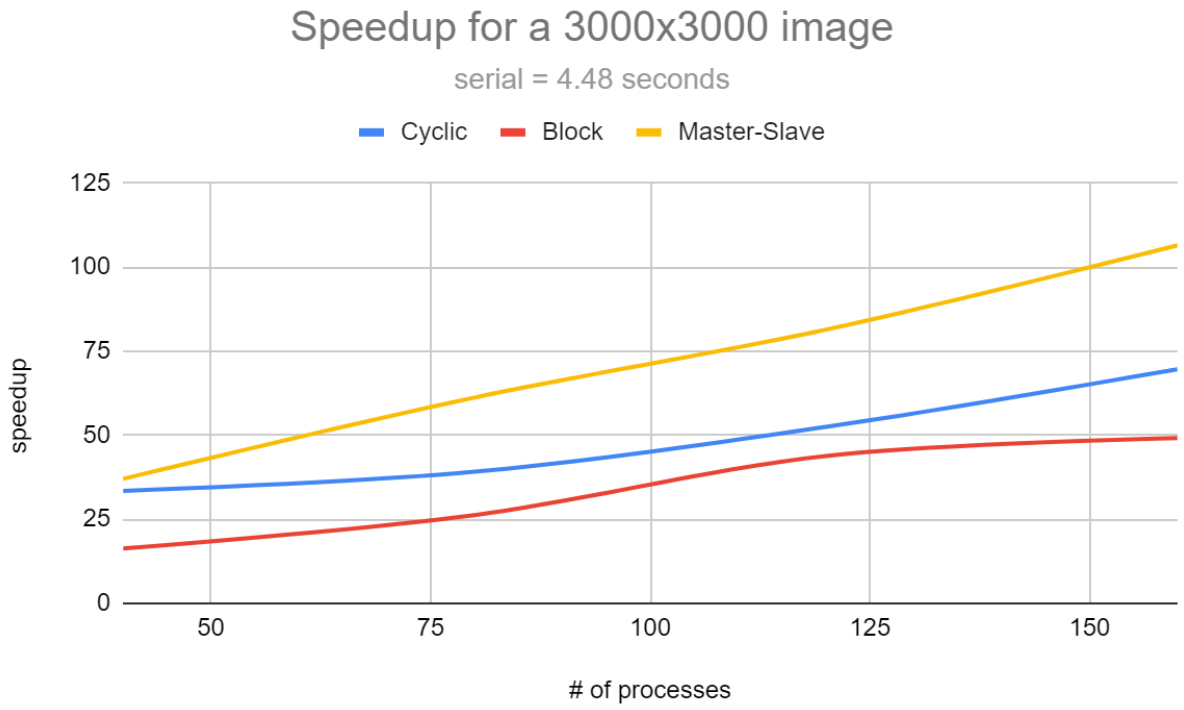

Figure 5: Speedup for a 3000x3000 image for increasing number of processes (from 40 to 160).

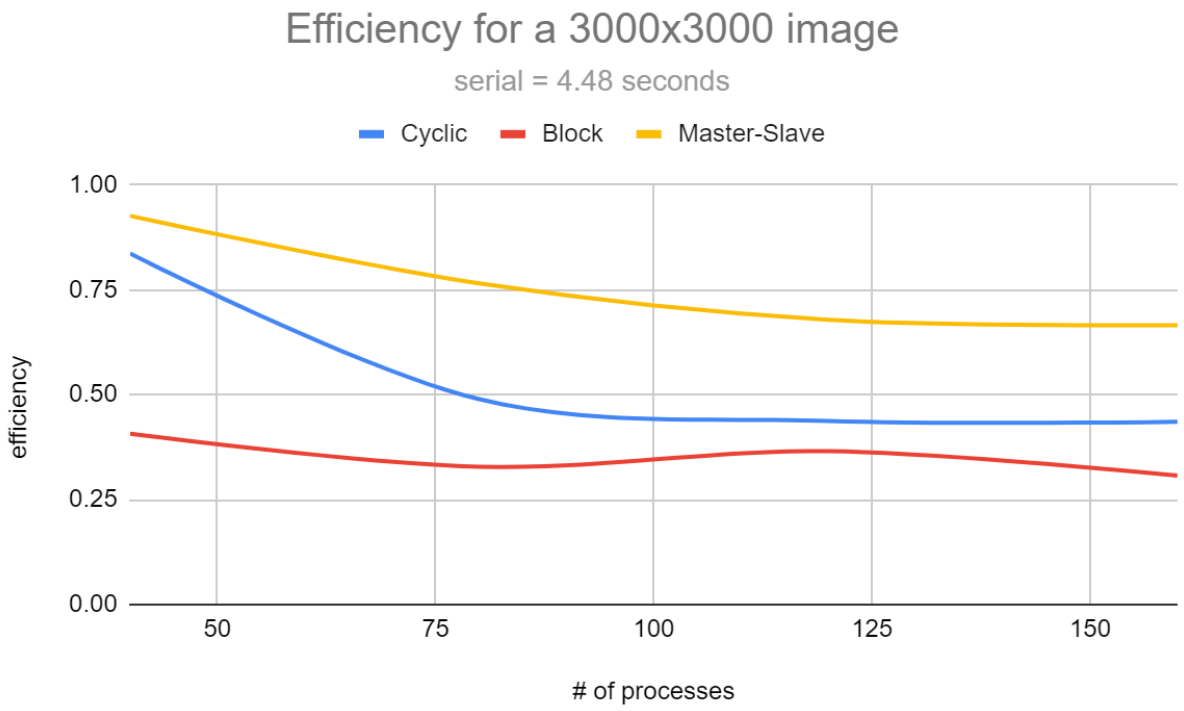

Figure 6: Speedup for a 3000x3000 image for increasing number of processes (from 40 to 160).

4. **Observations and Comparisons (answers are in blue; everything else is explanation)**

First, we talk about whose proposal (cyclic implementation by Sasha Cyclic and block implementation by Joe Block) is better. At a glance, it may *seem* that the block version is better because the data is contiguous. (There is no need to reorder the matrix after gathering and it pairs well with MPI_Gather, since MPI_Gather contiguously gathers from each process.) Although both versions use communication only once (MPI_Gather), the total times differ.

In every case, the cyclic implementation is faster overall in both total time and communication time; however, its computation time is noticeably slower than the block implementation. The cyclic version also has much better speedup and efficiency over the block version. The block version is better than the cyclic version only for computation time, but most of its time is taken by communication.

The gain in speedup in the cyclic version is also much higher than that in the block version when the number of processors is high; at a low number of processors (specifically from 40 processors to 80 processors), the block version has a higher gain in speedup. In terms of efficiency, the cyclic version overall beats the block version.

For these reasons, Sasha Cyclic should deserve a full-time job for having the better proposal over Joe Block.

We now compare the dynamic MW model over the static cyclic and block models; specifically, we compare using the speedup and efficiency. Although the MW model uses more communication instances than the cyclic and block models, the size of each message

in the MW model is only 1 row + 1 element (to store the row contents and row number) rather than several rows. In the static versions, the size of each message is calculated with this formula:

(4) static implementation: size of each message = floor(number of rows / number of processes)

If there are any leftover rows in the static versions, then the root process computes the rest. In general, as the number of processes increases, the speedup increases; however, the efficiency decreases.

The master-worker model has a significantly higher gain in speedup as more processes are added. The exception to this observation was for a small image size (1000x1000 image); its speedup hardly increased after increasing the number of processors from 80 to 120 and from 120 to 160. Despite that, it still has the highest efficiency compared to the cyclic and block versions.

When increasing the image size (to 10000x10000), the MW model has a slightly better speedup than at the smaller size (3000x3000), but the cyclic and block versions hardly gain any speedup. At 10000x10000 with 160 processors, the MW model has approximately the same total time as the serial version when the serial version is working with a 1000x1000 image (0.47 seconds for serial 1000x1000, 0.44 seconds for MW 10000x10000); the cyclic and block versions need a much smaller problem size to finish within 0.47 seconds (it takes 0.72 seconds and 1.22 seconds for the cyclic and block versions to finish working with a 10000x10000 image). For these reasons, the MW model has the highest scalability with large images.