TU Wien
Faculty of Informatics
Research Group for Parallel Computing

**Basics of Parallel Computing**
2024S
Assignment 1

May 2, 2024

**2 Person Group 13**
1: Pia SCHWARZINGER, 12017370
2: Yahya JABARY, 11912007

## 2 The Tasks

### 2.2 Compute Speed-up and Parallel Efficiency for 2 Instance Sizes

Tables 1 and 2 present the parallel runtime measurements and the respective speed-up and efficiency values for the cases $c_s$ and $c_b$.

Table 1: Runtime and speed-up of parallel Julia set generator for $c_s$ case.

| size | p | mean runtime (s) | speed-up | par. eff. |
|---|---|---|---|---|
| 155 | 1 | 0.276419 | 1 | 1.42473 |
| 155 | 2 | 0.14614 | 1.89147 | 1.34742 |
| 155 | 4 | 0.0850049 | 3.2518 | 1.15824 |
| 155 | 8 | 0.0684728 | 4.03691 | 0.71894 |
| 155 | 16 | 0.0727858 | 3.7977 | 0.338169 |
| 155 | 24 | 0.0845242 | 3.27029 | 0.194137 |
| 155 | 32 | 0.0975262 | 2.8343 | 0.126191 |
| 1100 | 1 | 13.42 | 1 | 1.40137 |
| 1100 | 2 | 6.66189 | 2.01444 | 1.41149 |
| 1100 | 4 | 3.37751 | 3.97333 | 1.39203 |
| 1100 | 8 | 1.72664 | 7.7723 | 1.36149 |
| 1100 | 16 | 0.908652 | 14.7691 | 1.29356 |
| 1100 | 24 | 0.654567 | 20.5021 | 1.19713 |
| 1100 | 32 | 0.564249 | 23.7838 | 1.04156 |

Keep in mind that while the speed-up was calculated using $p = 1$ as the reference point, the parallel efficiency was calculated using an average of the sequential runtime, by running the following commands 3 times on the Hydra-cluster and averaging the results:

```
srun -p q_student -t 1 -N 1 -c 32 python3 julia.py --size 155 --nprocs 1
# 155;20;1;0.39382300106808543
srun -p q_student -t 1 -N 1 -c 32 python3 julia.py --size 1100 --nprocs 1
# 1100;20;1;18.806384983938187
```

Table 2: Runtime and speed-up of parallel Julia set generator for $c_b$ case.

| size | p | mean runtime (s) | speed-up | par. eff. |
|---|---|---|---|---|
| 155 | 1 | 0.394086 | 1 | 0.711286 |
| 155 | 2 | 0.210897 | 1.86862 | 0.66456 |
| 155 | 4 | 0.121818 | 3.23504 | 0.575259 |
| 155 | 8 | 0.0862224 | 4.57057 | 0.406373 |
| 155 | 16 | 0.0852527 | 4.62256 | 0.205497 |
| 155 | 24 | 0.0963721 | 4.08921 | 0.121191 |
| 155 | 32 | 0.108531 | 3.6311 | 0.0807109 |
| 1100 | 1 | 19.1049 | 1 | 0.68691 |
| 1100 | 2 | 9.67163 | 1.97536 | 0.678447 |
| 1100 | 4 | 4.90018 | 3.89883 | 0.669536 |
| 1100 | 8 | 2.4526 | 7.78967 | 0.66885 |
| 1100 | 16 | 1.28631 | 14.8525 | 0.637646 |
| 1100 | 24 | 0.900398 | 21.2183 | 0.607295 |
| 1100 | 32 | 0.746145 | 25.6049 | 0.549633 |

Same as before, the parallel efficiency was calculated using an average of the sequential runtime, by running the following commands 3 times on the Hydra-cluster and averaging the results:

```
srun -p q_student -t 1 -N 1 -c 32 python3 julia.py --size 155 --nprocs 1 --benchmark
# 155;20;1;0.2803074959665537
srun -p q_student -t 1 -N 1 -c 32 python3 julia.py --size 1100 --nprocs 1 --benchmark
# 1100;20;1;13.123375411145389
```

It's important to mention that we opted for logarithmic scaling on all y-axes in our graphs. This choice was made to enhance the visibility of function shapes/gradients over the marginal differences. However, it's crucial to be mindful that this results in each increment on the y-axis carrying significantly more weight.

Also: Going forward, we won't focus as much on how different seeds (S-Case vs. B-Case) change the Julia set fractal. This is because the load size has a much bigger impact on our results. The results we've seen using different seeds have been very similar overall.

### 2.2.1 Comparing: Absolute Speed-up vs. Number of Processes

Figure 1 shows the absolute runtime in comparison to number of cores for both the S-case and B-case for two different sizes.

When comparing the impact of process count on runtime across datasets of differing sizes, we observe a few key trends.

With the larger dataset, there's a significantly sharper decline in runtime as we increase the number of processes/processors (these terms can *approximately* be used interchangeably since setting `chunksize=1` in Python's `multiprocessing.Pool.map` means one task is mapped to each processor at a time - unless we have fewer cores than processes). Additionally, the runtime appears to flatten out as we increase the processor count. Conversely, with the smaller dataset, the decline in runtime is less pronounced, and starting from 8 processes, the overhead from fork syscalls actually outweighs the benefits of parallel computation, leading to an increase in runtime.

### 2.2.2 Comparing: Relative Speed-up vs. Number of Processes

Figure 2 demonstrates the relative speed up in comparison to number of cores for both the S-case and B-case for the same sizes as before.

In our earlier analysis, we delved into the marginal decrease in runtime with the addition of each process. Essentially, this mirrors the concept of speedup, particularly when our baseline is at $p = 1$. These findings reinforce what we've already discussed. However, it's important to note that the speedup gap between the two loads in the S-case is significantly larger. This likely stems from the increased number of iterations and computations needed per pixel to achieve convergence in the visualization.

### 2.2.3 Comparing: Parallel Efficiency vs. Number of Processes

The final figure 3 reveals a generally similar downward trend for both seed-cases in the significance per process with increasing number of processes (note that this is parallel efficiency, distinct from utilization). Additionally the decrease in parallel efficiency is a lot more pronounced in the smaller workload and comes very close to 0 as we reach $p = 32$ in both seed-cases. A crucial distinction lies in the initial baseline: in the B-case, the starting point for process efficiency is substantially lower (by a factor of 2) than the S-case. This significant difference in initial efficiency is not directly apparent from the graph itself and would require further examination of the data provided in the appendix. This could be another indicator for a higher compute-intensity in the S-case in addition to the gap previously observed in the speed-up.

### 2.2.4 Discussion

The findings highlight the importance of considering load size and parallel processing efficiency in computational analysis. Variation in seed choice has minimal impact compared to load size on results. Larger datasets show sharper decline in runtime with increasing processes, while smaller ones exhibit diminishing returns and eventual increase in runtime due to overhead. Parallel efficiency decreases with more processes, with the B-case starting at a lower efficiency than the S-case, indicating higher compute-intensity in the latter.

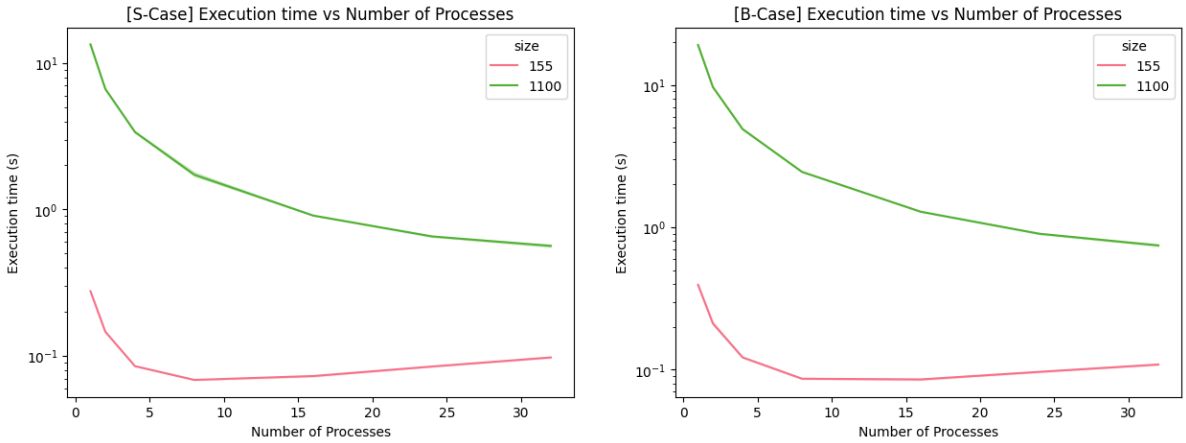


Figure 1: Absolute Runtime vs. Number of Processes

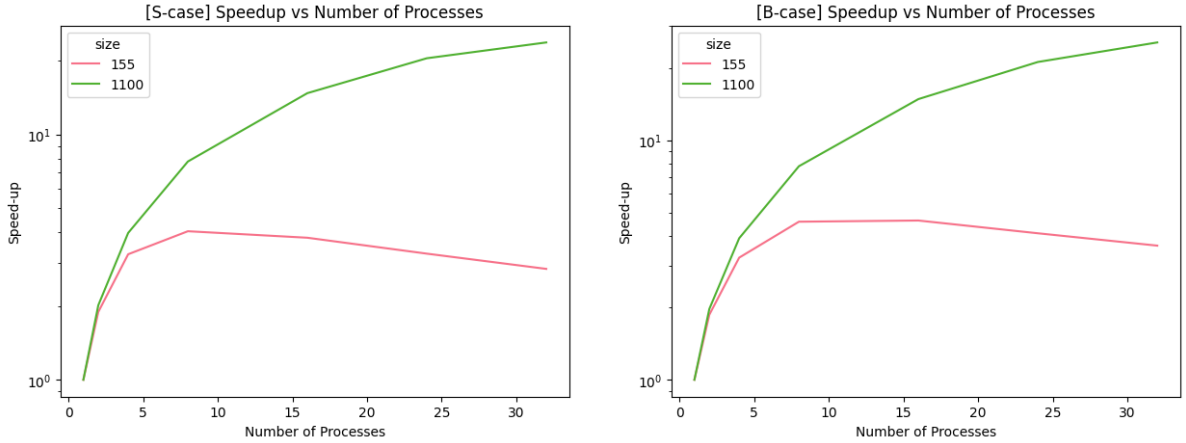


Figure 2: Relative Speed-up vs. Number of Processes

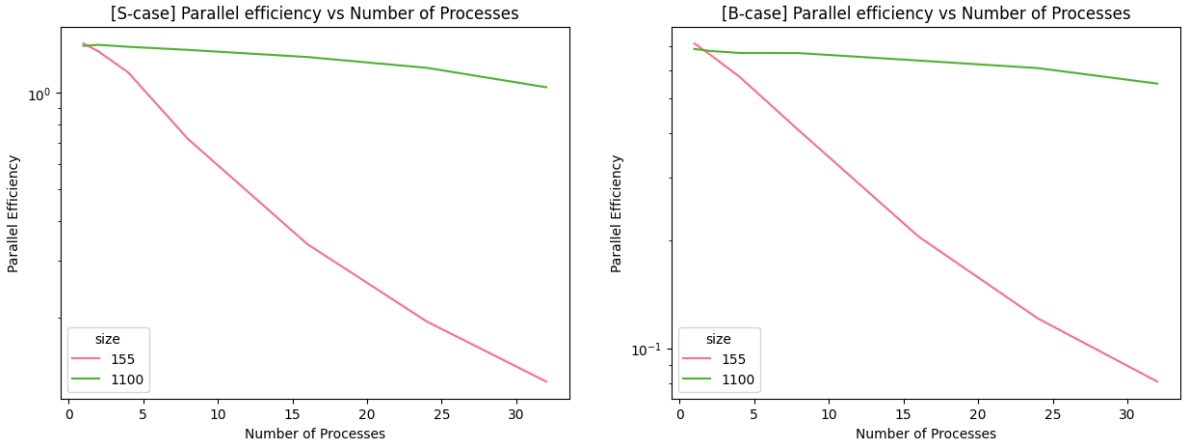


Figure 3: Parallel Efficiency vs. Number of Processes

### 2.3 Influence of Patch Size

Table 3 presents the average runtimes in seconds of varying patch sizes, while keeping the number of cores and problem size the same. Again, the means are taken from three runs on the cluster.

Table 3: Influence of patch size on mean runtime

| size | patch | nprocs | avg_time |
|---|---|---|---|
| 950 | 1 | 32 | 59.680444 |
| 950 | 5 | 32 | 2.439178 |
| 950 | 10 | 32 | 0.827185 |
| 950 | 20 | 32 | 0.589660 |
| 950 | 55 | 32 | 0.649828 |
| 950 | 150 | 32 | 1.791236 |
| 950 | 400 | 32 | 5.347212 |

Figure 4 now visually shows the influence of the patch size on the mean runtime.

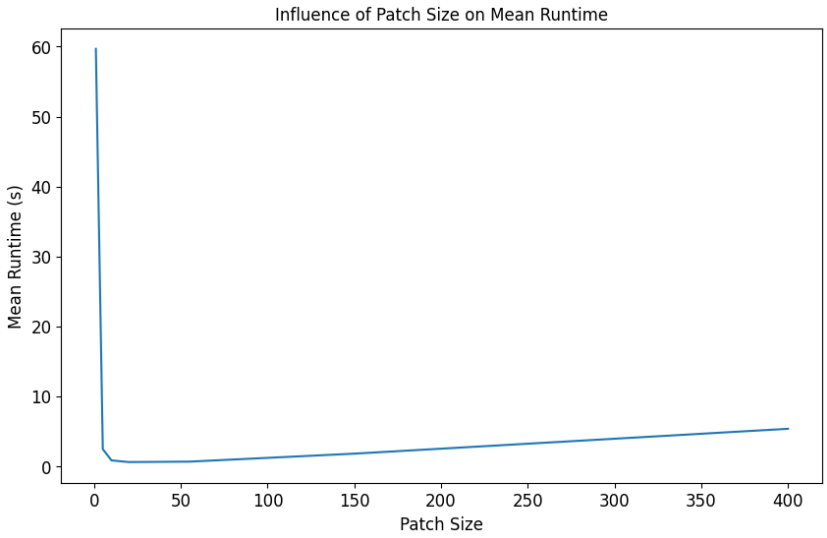


Figure 4: Mean Runtime vs. Patch Size - Influence of patch size

As we can tell having a patch size of 1 leads to an extremely high runtime but drastically decreases once multiple patches are involved. This is because computational resources can be used more effectively and parallel processing can be improved. However, starting from 150 patches upwards, the runtime increases which is probably due to the overhead that comes with managing a large number of parallel processes.

### 2.4 Finding the Best Patch Size

In order to find the best possible patch size, table 4 and figure 5 present the results of our experiments.

Table 4: Best patch size

| size | patch | nprocs | avg_time |
|---|---|---|---|
| 800 | 1 | 16 | 50.020686 |
| 800 | 2 | 16 | 12.428263 |
| 800 | 3 | 16 | 5.235198 |
| 800 | 4 | 16 | 2.781848 |
| 800 | 5 | 16 | 1.778088 |
| 800 | 6 | 16 | 1.262205 |
| 800 | 7 | 16 | 1.025805 |
| 800 | 8 | 16 | 0.871605 |
| 800 | 9 | 16 | 0.841269 |
| 800 | 10 | 16 | 0.788733 |
| 800 | 11 | 16 | 0.777117 |
| 800 | 12 | 16 | 0.730836 |
| 800 | 13 | 16 | 0.721792 |
| 800 | 14 | 16 | 0.732972 |
| 800 | 15 | 16 | 0.712528 |
| 800 | 16 | 16 | 0.736369 |
| 800 | 17 | 16 | 0.714250 |
| 800 | 18 | 16 | 0.709131 |
| 800 | 19 | 16 | 0.711833 |
| 800 | 20 | 16 | 0.704302 |
| 800 | 21 | 16 | 0.712072 |
| 800 | 22 | 16 | 0.715015 |
| 800 | 23 | 16 | 0.718865 |
| 800 | 24 | 16 | 0.710345 |
| 800 | 25 | 16 | 0.706855 |
| 800 | 26 | 16 | 0.709854 |
| 800 | 27 | 16 | 0.715558 |
| 800 | 28 | 16 | 0.708274 |
| 800 | 29 | 16 | 0.702031 |
| 800 | 30 | 16 | 0.705310 |

Based on figure 5, one can tell that the runtime seems to converge starting from a patch size of 10. The minimum and therefore best patch size according to the numbers provided in table 4, is 29 which leads to an average runtime of 0.702 seconds.

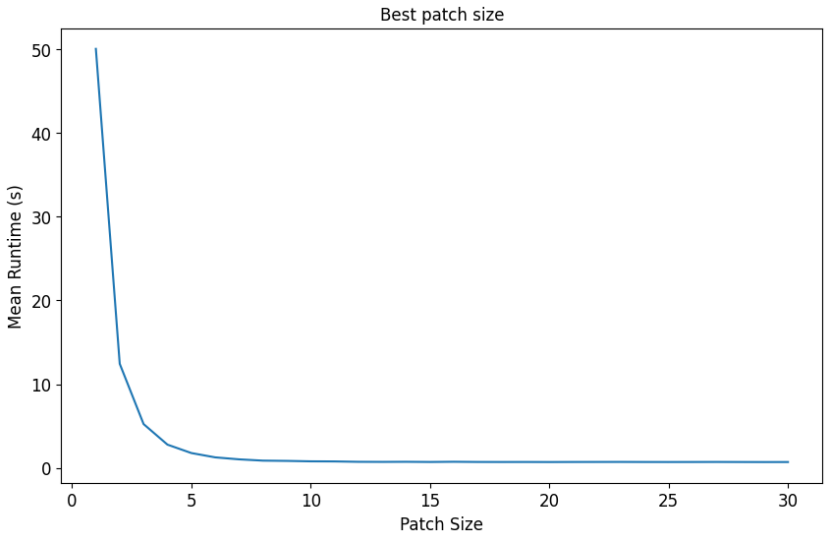


Figure 5: Mean Runtime vs. Patch Size - Finding best patch size

## 3 Speed-up Analysis

### 3.1 Absolute speed up

One can calculate the absolute speed up by computing $S_a^{A_i}(n,p) = \frac{T^*_{seq}(n)}{T_{par}(n,p)}$.

Algorithm sequential:

$$T^*_{\text{seq}}(n) = O(n \log n)$$
$$T^*_{\text{seq}}(1000) = O(1000 \log 1000) \approx 6907.76$$

Algorithm A1:

$$T^{A_1}_{\text{par}}(n,p) = O\left(\frac{n \log n}{p} + \log n\right)$$
$$T^{A_1}_{\text{par}}(1000,4) \approx 1733.85$$
$$T^{A_1}_{\text{par}}(1000,16) \approx 438.64$$
$$T^{A_1}_{\text{par}}(1000,64) \approx 114.84$$

$$S_a^{A_1}(1000,4) = \frac{6907.76}{1733.85} \approx 3.98$$
$$S_a^{A_1}(1000,16) = \frac{6907.76}{438.64} \approx 15.74$$
$$S_a^{A_1}(1000,64) = \frac{6907.76}{114.84} \approx 60.15$$

Algorithm A2:

$$T_{\text{par}}^{A_2}(n,p) = O\left(\frac{n\log n}{p} + n\right)$$
$$T_{\text{par}}^{A_2}(1000,4) \approx 2726.94$$
$$T_{\text{par}}^{A_2}(1000,16) \approx 1431.73$$
$$T_{\text{par}}^{A_2}(1000,64) \approx 1107.93$$

$$S_a^{A_2}(1000,4) = \frac{6907.76}{2726.94} \approx 2.53$$
$$S_a^{A_2}(1000,16) = \frac{6907.76}{1431.73} \approx 4.82$$
$$S_a^{A_2}(1000,64) = \frac{6907.76}{1107.93} \approx 6.23$$

### 3.2 Parallel Efficiency

One can calculate the parallel efficiency by computing $E^{Ai}(n,p) = \frac{S_a^{A_i}(n,p)}{p}$.

Algorithm A1:

$$E^{A_1}(1000,4) = \frac{S_a^{A_1}(1000,4)}{4} \approx 1.0$$
$$E^{A_1}(1000,16) = \frac{S_a^{A_1}(1000,16)}{16} \approx 0.98$$
$$E^{A_1}(1000,64) = \frac{S_a^{A_1}(1000,64)}{64} \approx 0.94$$

Algorithm A2:

$$E^{A_2}(1000,4) = \frac{S_a^{A_2}(1000,4)}{4} \approx 0.63$$
$$E^{A_2}(1000,16) = \frac{S_a^{A_2}(1000,16)}{16} \approx 0.30$$
$$E^{A_2}(1000,64) = \frac{S_a^{A_2}(1000,64)}{64} \approx 0.1$$

### 3.3 Potential Speed-up

According to Amdahl's Law the potential speed up is bounded by the sequential fraction of the code.

Algorithm A1:
The sequential fraction of A1 is $s_1 = \frac{\log n}{\frac{n \log n}{p} + \log n} = \frac{p}{n+p}$.

$$S^{A_1}(n,p) = \frac{1}{s_1 + \frac{1-s_1}{p}} \leq \frac{1}{s_1} = \frac{1}{\frac{p}{n+p} + \frac{1-\frac{p}{n+p}}{p}} \leq \frac{1}{\frac{p}{n+p}}$$

Algorithm A2:
The sequential fraction of A2 is $s_2 = \frac{\log n}{\frac{n \log n}{p} + n} = \frac{\log n * p}{n * (\log n + p)}$.

$$S^{A_2}(n,p) = \frac{1}{s_2 + \frac{1-s_2}{p}} \leq \frac{1}{s_2} = \frac{1}{\frac{\log n * p}{n*(\log n+p)} + \frac{1-\frac{\log n * p}{n*(\log n+p)}}{p}} \leq \frac{1}{\frac{\log n * p}{n*(\log n+p)}}$$

However, in the solution above "p" is still involved. Therefore, we decided to use a different approach and take a look at the ratio of the sequential part of the algorithms (in terms of O notation) and take the ratio to the best performing sequential algorithm.

Algorithm A1:
The sequential part of A1 is $s_1 = \log n$.

$$S^{A_1}(n) = \frac{n \log n}{\log n} = n$$

Algorithm A2:
The sequential fraction of A2 is $s_2 = n$.

$$S^{A_2}(n) = \frac{n \log n}{n} = \log n$$

## 4 Weak Scaling Analysis

### 4.1 Growth of input size

$$\begin{aligned} O(\frac{n^4}{p}) &= O(w) \\ \frac{n^4}{p} &= w \\ n^4 &= pw \\ n &= (pw)^{\frac{1}{4}} \end{aligned}$$

Since the average work per processor is w and the parallel algorithm runs in $O(\frac{n^4}{p})$, n needs to increase as the fourth root of the product of w and p to keep the average work at O(n).

### 4.2 Input size for different p

Given that n=100 when p=1, the various input sizes for different numbers of processors can be computed as follows. This time we decided to perform the calculations in R closely following the code presented in the lecture, as can be seen in listing 1.

Listing 1: Calculation n for various values of p

```
n <- 100
w <- n^4
df <- data.frame(p=c(2,4,8,16,128))
df$n <- ceiling((df$p*w)^(1/4))
print(df)
```

Table 4.2 shows the output of the code and presents the necessary input size for various numbers of cores.

| $p$ | $n$ |
|---|---|
| 2 | 119 |
| 4 | 142 |
| 8 | 169 |
| 16 | 200 |
| 128 | 337 |

Table 5: Required input sizes for different numbers of processors

## Appendix

### System Information

Listing 2: Output of 'lscpu' command

```
bopc23s9@hydra-head:~$ lscpu

Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          16
On-line CPU(s) list:             0-15
Thread(s) per core:              1
Core(s) per socket:              16
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           85
Model name:                      Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
Stepping:                        4
CPU MHz:                         1000.151
CPU max MHz:                     3700.0000
CPU min MHz:                     1000.0000
BogoMIPS:                        4200.00
L1d cache:                       512 KiB
L1i cache:                       512 KiB
L2 cache:                        16 MiB
L3 cache:                        22 MiB
NUMA node0 CPU(s):               0-15
```

### Raw Data

| size | patch | nprocs | time |
|---|---|---|---|
| 155 | 26 | 1 | 0.27615245105698705 |
| 155 | 26 | 1 | 0.27596693905070424 |
| 155 | 26 | 1 | 0.2771365772932768 |
| 1100 | 26 | 1 | 13.435972596984357 |
| 1100 | 26 | 1 | 13.452274681068957 |
| 1100 | 26 | 1 | 13.371691829990596 |
| 155 | 26 | 2 | 0.14638189878314734 |
| 155 | 26 | 2 | 0.14576987363398075 |
| 155 | 26 | 2 | 0.1462678317911923 |
| 1100 | 26 | 2 | 6.670089309103787 |
| 1100 | 26 | 2 | 6.613465172238648 |
| 1100 | 26 | 2 | 6.7021133550442755 |
| 155 | 26 | 4 | 0.08490922581404448 |
| 155 | 26 | 4 | 0.08491231314837933 |
| 155 | 26 | 4 | 0.08519301796332002 |
| 1100 | 26 | 4 | 3.3561069620773196 |
| 1100 | 26 | 4 | 3.3553116391412914 |
| 1100 | 26 | 4 | 3.421122542116791 |
| 155 | 26 | 8 | 0.06856991816312075 |
| 155 | 26 | 8 | 0.06856326712295413 |
| 155 | 26 | 8 | 0.06828531296923757 |
| 1100 | 26 | 8 | 1.6898975907824934 |
| 1100 | 26 | 8 | 1.7937131081707776 |
| 1100 | 26 | 8 | 1.6963165369816124 |
| 155 | 26 | 16 | 0.07205457100644708 |
| 155 | 26 | 16 | 0.0730455950833857 |
| 155 | 26 | 16 | 0.07325727492570877 |
| 1100 | 26 | 16 | 0.9077705508098006 |
| 1100 | 26 | 16 | 0.9078279393725097 |
| 1100 | 26 | 16 | 0.9103577299974859 |
| 155 | 26 | 24 | 0.0845845378935337 |
| 155 | 26 | 24 | 0.08471266506239772 |
| 155 | 26 | 24 | 0.08427527407184243 |
| 1100 | 26 | 24 | 0.656599803827703 |
| 1100 | 26 | 24 | 0.6573387430980802 |
| 1100 | 26 | 24 | 0.6497617312707007 |
| 155 | 26 | 32 | 0.0973281990736723 |
| 155 | 26 | 32 | 0.09837603475898504 |
| 155 | 26 | 32 | 0.09687442006543279 |
| 1100 | 26 | 32 | 0.5734831769950688 |
| 1100 | 26 | 32 | 0.5541784320957959 |
| 1100 | 26 | 32 | 0.565086467191577 |

Table 6: Output 2.2.1

| size | patch | nprocs | time |
|---|---|---|---|
| 155 | 26 | 1 | 0.392238802742213 |
| 155 | 26 | 1 | 0.39877972193062305 |
| 155 | 26 | 1 | 0.3912382051348686 |
| 1100 | 26 | 1 | 19.34426457900554 |
| 1100 | 26 | 1 | 18.90565126761794 |
| 1100 | 26 | 1 | 19.064889647066593 |
| 155 | 26 | 2 | 0.20706806099042296 |
| 155 | 26 | 2 | 0.20968409720808268 |
| 155 | 26 | 2 | 0.2159389308653772 |
| 1100 | 26 | 2 | 9.65854894882068 |
| 1100 | 26 | 2 | 9.758664200082421 |
| 1100 | 26 | 2 | 9.597689433023334 |
| 155 | 26 | 4 | 0.12078673578798771 |
| 155 | 26 | 4 | 0.12187985517084599 |
| 155 | 26 | 4 | 0.12278737686574459 |
| 1100 | 26 | 4 | 4.8538289330899715 |
| 1100 | 26 | 4 | 4.949722504708916 |
| 1100 | 26 | 4 | 4.896979348734021 |
| 155 | 26 | 8 | 0.08609647722914815 |
| 155 | 26 | 8 | 0.08687825128436089 |
| 155 | 26 | 8 | 0.0856924457475543 |
| 1100 | 26 | 8 | 2.4588615149259567 |
| 1100 | 26 | 8 | 2.4403636367060244 |
| 1100 | 26 | 8 | 2.4585742950439453 |
| 155 | 26 | 16 | 0.08432547515258193 |
| 155 | 26 | 16 | 0.08534767106175423 |
| 155 | 26 | 16 | 0.08608504896983504 |
| 1100 | 26 | 16 | 1.2857805700041354 |
| 1100 | 26 | 16 | 1.28942183079198 |
| 1100 | 26 | 16 | 1.2837325409054756 |
| 155 | 26 | 24 | 0.09584450582042336 |
| 155 | 26 | 24 | 0.09703108435496688 |
| 155 | 26 | 24 | 0.09624077333137393 |
| 1100 | 26 | 24 | 0.9055881663225591 |
| 1100 | 26 | 24 | 0.8943913546390831 |
| 1100 | 26 | 24 | 0.9012130876071751 |
| 155 | 26 | 32 | 0.10889460984617472 |
| 155 | 26 | 32 | 0.10903614293783903 |
| 155 | 26 | 32 | 0.1076613599434495 |
| 1100 | 26 | 32 | 0.7539521278813481 |
| 1100 | 26 | 32 | 0.7489344119094312 |
| 1100 | 26 | 32 | 0.7355474359355867 |

Table 7: Output 2.2.2

| size | patch | nprocs | time |
|---|---|---|---|
| 950 | 1 | 32 | 60.38315498735756 |
| 950 | 1 | 32 | 58.993379454128444 |
| 950 | 1 | 32 | 59.664796941913664 |
| 950 | 5 | 32 | 2.4295669128187 |
| 950 | 5 | 32 | 2.457375079859048 |
| 950 | 5 | 32 | 2.430592040065676 |
| 950 | 10 | 32 | 0.8235339601524174 |
| 950 | 10 | 32 | 0.8456530389375985 |
| 950 | 10 | 32 | 0.8123667249456048 |
| 950 | 20 | 32 | 0.592178599908948 |
| 950 | 20 | 32 | 0.5885163000784814 |
| 950 | 20 | 32 | 0.5882865060120821 |
| 950 | 55 | 32 | 0.640842576045543 |
| 950 | 55 | 32 | 0.6622877516783774 |
| 950 | 55 | 32 | 0.6463527246378362 |
| 950 | 150 | 32 | 1.804339012131095 |
| 950 | 150 | 32 | 1.7855969751253724 |
| 950 | 150 | 32 | 1.7837719358503819 |
| 950 | 400 | 32 | 5.342996523249894 |
| 950 | 400 | 32 | 5.354430069215596 |
| 950 | 400 | 32 | 5.3442102540284395 |

Table 8: Output 2.3

| size | patch | nprocs | time |
|---|---|---|---|
| 800 | 1 | 16 | 5.177.733.509.801.320 |
| 800 | 1 | 16 | 48.679.752.216.208.700 |
| 800 | 1 | 16 | 49.604.969.315.230.800 |
| 800 | 2 | 16 | 12.208.631.441.928.400 |
| 800 | 2 | 16 | 12.400.218.938.011.600 |
| 800 | 2 | 16 | 1.267.593.849.496.910 |
| 800 | 3 | 16 | 5.231.156.553.607.430 |
| 800 | 3 | 16 | 5.173.088.782.932.600 |
| 800 | 3 | 16 | 530.134.785.734.117 |
| 800 | 4 | 16 | 2.795.454.104.896.630 |
| 800 | 4 | 16 | 28.061.751.988.716.400 |
| 800 | 4 | 16 | 2.743.914.455.641.060 |
| 800 | 5 | 16 | 17.844.559.531.658.800 |
| 800 | 5 | 16 | 1.795.137.454.289.940 |
| 800 | 5 | 16 | 1.754.671.474.918.720 |
| 800 | 6 | 16 | 12.681.594.076.566.300 |
| 800 | 6 | 16 | 12.242.525.001.056.400 |
| 800 | 6 | 16 | 12.942.032.269.202.100 |
| 800 | 7 | 16 | 10.078.809.931.874.200 |
| 800 | 7 | 16 | 10.316.235.939.972.100 |
| 800 | 7 | 16 | 10.379.106.737.673.200 |
| 800 | 8 | 16 | 0.8636039649136364 |
| 800 | 8 | 16 | 0.8836448309011757 |
| 800 | 8 | 16 | 0.8675666828639805 |
| 800 | 9 | 16 | 0.8217232823371887 |
| 800 | 9 | 16 | 0.8141451748088002 |
| 800 | 9 | 16 | 0.8879383755847812 |
| 800 | 10 | 16 | 0.8032627403736115 |
| 800 | 10 | 16 | 0.790965499356389 |
| 800 | 10 | 16 | 0.7719701924361289 |
| 800 | 11 | 16 | 0.775052934885025 |
| 800 | 11 | 16 | 0.7863136758096516 |
| 800 | 11 | 16 | 0.7699855691753328 |
| 800 | 12 | 16 | 0.7268759598955512 |
| 800 | 12 | 16 | 0.7416912410408258 |
| 800 | 12 | 16 | 0.7239395705983043 |
| 800 | 13 | 16 | 0.7274400270543993 |
| 800 | 13 | 16 | 0.7136987750418484 |
| 800 | 13 | 16 | 0.7242374992929399 |
| 800 | 14 | 16 | 0.7333642770536244 |
| 800 | 14 | 16 | 0.7298130737617612 |
| 800 | 14 | 16 | 0.7357385512441397 |
| 800 | 15 | 16 | 0.7113026860170066 |
| 800 | 15 | 16 | 0.7115092552267015 |

Table 9: Output 2.4 Part 1

| size | patch | nprocs | time |
|---|---|---|---|
| 800 | 15 | 16 | 0.7147729960270226 |
| 800 | 16 | 16 | 0.7343269921839237 |
| 800 | 16 | 16 | 0.7277633068151772 |
| 800 | 16 | 16 | 0.7470175549387932 |
| 800 | 17 | 16 | 0.7087195003405213 |
| 800 | 17 | 16 | 0.7273335340432823 |
| 800 | 17 | 16 | 0.7066979217343032 |
| 800 | 18 | 16 | 0.7008385430090129 |
| 800 | 18 | 16 | 0.7211737250909209 |
| 800 | 18 | 16 | 0.7053795196115971 |
| 800 | 19 | 16 | 0.704965204000473 |
| 800 | 19 | 16 | 0.7185817346908152 |
| 800 | 19 | 16 | 0.7119517708197236 |
| 800 | 20 | 16 | 0.7045390028506517 |
| 800 | 20 | 16 | 0.7090244409628212 |
| 800 | 20 | 16 | 0.6993431178852916 |
| 800 | 21 | 16 | 0.7067825132980943 |
| 800 | 21 | 16 | 0.7152607310563326 |
| 800 | 21 | 16 | 0.7141714952886105 |
| 800 | 22 | 16 | 0.7289159940555692 |
| 800 | 22 | 16 | 0.7070880490355194 |
| 800 | 22 | 16 | 0.7090400322340429 |
| 800 | 23 | 16 | 0.7122855372726917 |
| 800 | 23 | 16 | 0.7262964779511094 |
| 800 | 23 | 16 | 0.7180132349021733 |
| 800 | 24 | 16 | 0.7030764939263463 |
| 800 | 24 | 16 | 0.7233439306728542 |
| 800 | 24 | 16 | 0.7046146122738719 |
| 800 | 25 | 16 | 0.7067729285918176 |
| 800 | 25 | 16 | 0.7068284871056676 |
| 800 | 25 | 16 | 0.7069635195657611 |
| 800 | 26 | 16 | 0.7092408840544522 |
| 800 | 26 | 16 | 0.7142819189466536 |
| 800 | 26 | 16 | 0.7060380070470273 |
| 800 | 27 | 16 | 0.7240377422422171 |
| 800 | 27 | 16 | 0.703698709141463 |
| 800 | 27 | 16 | 0.7189373909495771 |
| 800 | 28 | 16 | 0.7107441178523004 |
| 800 | 28 | 16 | 0.7102009649388492 |
| 800 | 28 | 16 | 0.7038759361021221 |
| 800 | 29 | 16 | 0.7044810829684138 |
| 800 | 29 | 16 | 0.7004124890081584 |
| 800 | 29 | 16 | 0.7011993718333542 |
| 800 | 30 | 16 | 0.7012511510401964 |
| 800 | 30 | 16 | 0.7043010322377086 |
| 800 | 30 | 16 | 0.7103770151734352 |



Table 10: Output 2.4 Part 2