Introduction:

In this report we use parallel computing by leveraging the Fork-Join framework to enhance the speed of finding a stable of an abelian sandpile.

Parallelisation Approach:

The Fork/Join framework in Java is utilized in the AutomatonSimulationParallel class (which depends on the Grid class) to parallelize the grid update process for the Abelian Sandpile cellular automaton. By extending RecursiveTask the code breaks down the grid update task into smaller subtasks that can be executed in parallel. The grid is divided into segments of rows and these segments are processed in parallel. If the segment is larger than a predefined threshold -CUTTOFF_Dimension (which is 100) it is further divided into two smaller tasks. This recursive division continues until each task is small enough to be executed sequentially. The results of these tasks are then combined using the Fork/Join framework which manages the parallel execution of these subtasks. The main method is then ran in the RunParallel class which runs the abelian sandpile algorithm in parallel.

This divide-and-conquer approach allows the program to take full advantage of multi-core processors by executing different parts of the grid update in parallel. As a result, the program can achieve faster execution times for large grids.

Algorithm Validation:

To validate the parallelisation a image processing algorithm “compare_images.py” was implemented. The compare_images.py script ensured that the outputs of the parallel and serial versions matched precisely by running compare_images.py to perform bit-by-bit comparisons of these outputs. It was verified that the parallelised code produced results consistent with the serial execution. This confirmed that the parallelisation did not introduce discrepancies or errors, ensuring the correctness and reliability of the parallel implementation. The compare_images.py script is included in the code submission on the automarker.

For each pair of images, it can be run by the following command on ubuntu: “python3 compare_images.py path/to/first/image.png path/to/second/image.png”.

It will then output, "The images are identical." If the images are the same bit for bit or otherwise "The images are different." if the images are different bit by bit.

Tested Machine Architectures:

The algorithm speeds were tested on two computer machines which was my personal computer (Computer 1) and one of the UCT Computers (Computer 2). These are their architectures:

Computer 1:

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Address sizes: 39 bits physical, 48 bits virtual

Byte Order: Little Endian

CPU(s): 8

On-line CPU(s) list: 0-7

Vendor ID: GenuineIntel

Model name: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz

CPU family: 6

Model: 140

Thread(s) per core: 2

Core(s) per socket: 4

Socket(s): 1

Stepping: 1

CPU max MHz: 4200.0000

CPU min MHz: 400.0000

BogoMIPS: 4838.40

Computer 2:

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Address sizes: 39 bits physical, 48 bits virtual

Byte Order: Little Endian

CPU(s): 4

On-line CPU(s) list: 0-3

Vendor ID: GenuineIntel

Model name: Intel(R) Core(TM) i3-8100 CPU @ 3.60GHz

CPU family: 6

Model: 158

Thread(s) per core: 1

Core(s) per socket: 4

Socket(s): 1

Stepping: 11

CPU(s) scaling MHz: 22%

CPU max MHz: 3600.0000

CPU min MHz: 800.0000

BogoMIPS: 7200.00

Benchmarking and Performance Evaluation:

The algorithms were tested using the same dimensions and values of grids on each computer. These grids can also be found in the “input” folder of the submission.

7 tests were performed on each computer namely:

8 x 8 Grid – all values 4

16 x 16 Grid – all values 4

65 x 65 Grid – all values 4

150 x 150 Grid – all values 8

370 x 370 Grid – all values 8

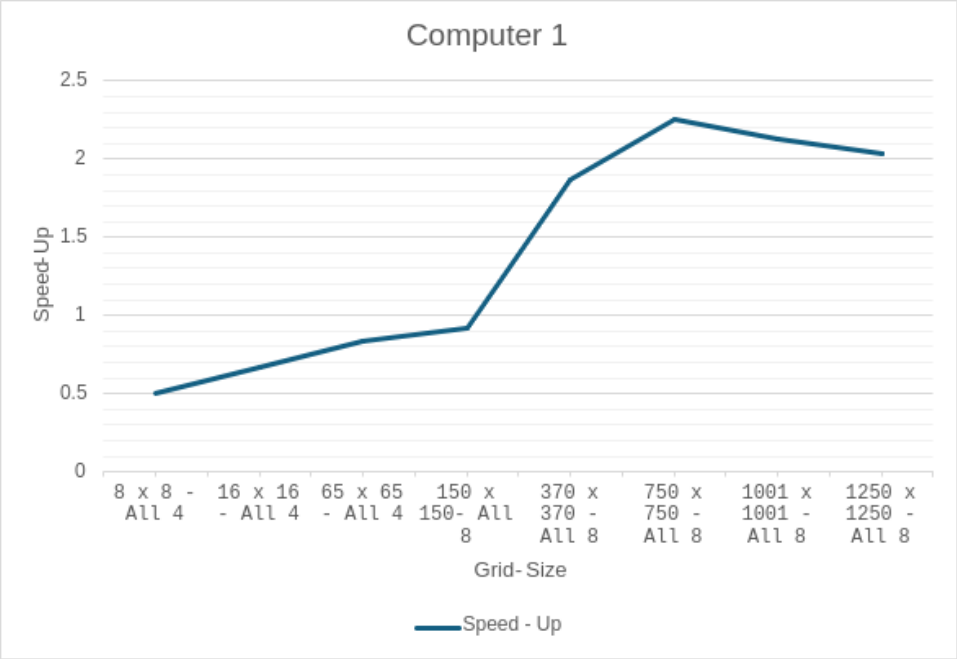
750 x 750 Grid – all values 8

1001 x 1001 Grid – all values 8

It was felt that testing these grids specifically provided a better representation of the "work" each program had to do progressively as they got bigger.

Each Grid was tested 5 times, and a median time was taken on algorithm to execute on each program.

This is the speed up graph for Computer 1:

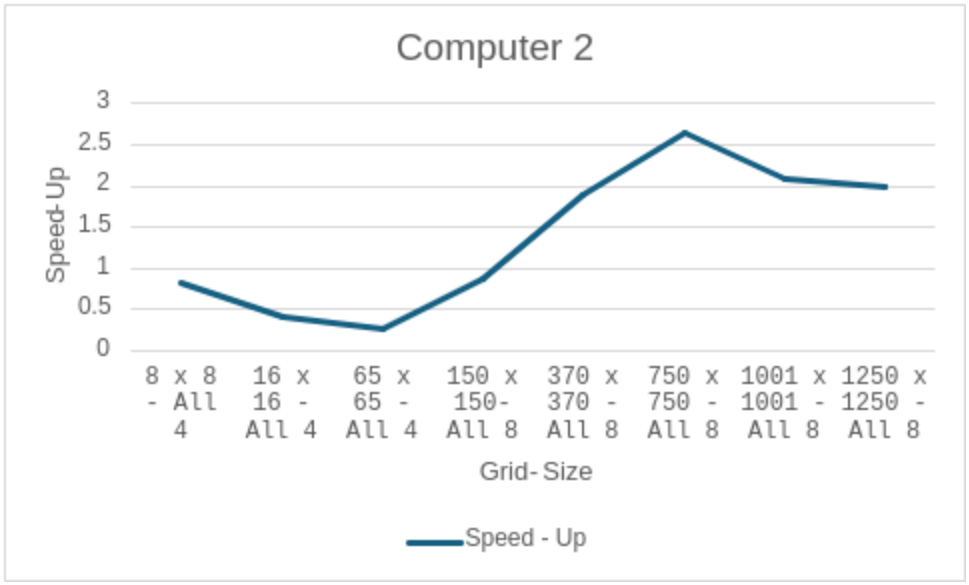


The best speedup for the parallelised code is achieved with larger grid sizes, where the benefits of parallel processing outweigh the overhead. For smaller grids (8x8 to 65x65), speedup was low ranging from 0.5 to 0.83. This is because computational load is not sufficient to overcome the parallelisation overhead for small grids. However for larger grids (150x150 and above) the speedup improves significantly peaking at 2.25 for a 750x750 grid as the workload justifies the cost of managing parallel tasks.

The maximum observed speedup of 2.25 is less than the theoretical 8 for 8 processors. This is due to practical factors such as task management overhead and memory limitations. While the speedup increases with larger grid sizes, it does not reach the theoretical speedup.

The dip in speedup for very large grid sizes (1001 x 1001 and 1250 x 1250) is mainly due to increased parallelization overhead, memory and cache limitations. As grid size grows, the overhead of managing parallel tasks becomes more processor heavy, and system memory and cache constraints can slow down data access and processing.

This is the speed up graph for Computer 2:



Here also the best speedup is observed with larger grid sizes, particularly at 750x750, where a speedup of 2.65 is achieved. This is because larger grids provide a sufficient workload to make parallelization worthwhile, allowing the available cores to be fully utilized. Smaller grids, such as 8x8 and 16x16 show lower speedups (0.83 and 0.42, respectively) due to the overhead of parallel task management and even a significant dip for the 65x65 grid, even though I suspect it's due to random error to dip so much, reflects inefficiencies in utilizing the cores for smaller data sizes.

The maximum speedup of 2.65 on the second computer less than the ideal speedup of 4, which is the theoretical maximum for a system with 4 cores. This gap is due to factors like parallelization overhead like what was observed on the first computer. While the speedup is significant, it indicates that not all cores are being utilized as efficiently as possible, and there is room for improvement.

Comparing the second computer to the first one (which has 8 cores), the first computer shows a generally higher potential for speedup due to its higher core count with a maximum speedup of 2.65 on a 750x750 grid, compared to the 2.25 speedup on the first computer. However, the second computer achieves a slightly higher speedup for the same grid size likely due to better cache utilization or less overhead in managing fewer cores but it remains to be seen if it is statistically significant since these values are not too far off each other. Both computers demonstrate the challenges of achieving ideal speedup which is expected given practical constraints of overheads.

Conclusion:

Considering the results, parallelisation provided some benefits but didn't fully achieve its potential. On the second computer with 4 cores, the maximum speedup was 2.65 which is significantly below the theoretical maximum of 4. Similarly, on the first computer with 8 cores the speedup was even less efficient as a point estimate but perhaps not much different statistically from the first computer. The overhead associated with parallel task management reduced the effectiveness of parallelization especially for smaller grid sizes where the speedup was minimal or even lower than 1 indicating a slowdown.

Given these factors while parallelisation did improve performance for larger grids, the benefits were only marginally worth the additional complexity when tested on more processors. This is an example of Amdahl’s Law of diminishing returns of adding more processors . The efficiency gains were not substantial enough to justify the overhead for some cases especially the smaller grids. Therefore, the overall value of parallelisation in this context is mixed and it may not have been entirely worth the effort particularly when considering the diminishing returns for larger grids and the suboptimal speedup achieved.