# Blessing Hlongwane Report
# 5 pages

**Parallelization Strategy**

**Detailed Explanation:**

I looked at how I can separate the tasks using the forkJoin framework which uses a divide and conquer strategy in this stance I divided my task in rows (the left and right row), updating the cells in parallel then combining the rows

The main task is defined in the ParallelGrid class, which extends RecursiveTask<Boolean>. This class represents a task (left or right) of the grid that will be processed in parallel.

The results from both tasks are combined using logical OR to determine if any cell changed during the update. **Which was one of my issues** since I needed to be updating my separated grids (left and right) in sync without any race conditions making sure that the threads are in synchronization. To do that I needed to keep track of the changes that are happening in respect of my original grid so I had a copy grade that would record if there any changes made to the left and right grid , The main method runs a loop, If any cells change (changed is true), the update method is called to copy the updated values back to the primary grid. The loop continues until no cells change, indicating a stable state of the cells in the grid.

**Issues**

I needed to be updating my separated grids(left and right) in sync without any race conditions making sure that the threads are in synchronization. To do that I needed to keep track of the changes that are happening in respect of my original grid

**Considerations**

Should I separate the tasks in fork Join Framework in quadrants (up, left, down, right) or coloumns(up and down)

**Determining a threshold**

The THRESHOLD constant defines the maximum size of a grid segment that will be processed without further splitting, in this instance 80, I had to consider r the grid sizes I am using.
After experimenting 80 was a good cut off, which performed better than others

**Problems / difficulties**

- Finding an appropriate Threshold
- Without the loop method to run the fork/join framework, my grid would only be updated once.
- Benchmarking (Takes time)

**Validation**

- Looking at the outcome of the pictures, compare them to the sequential and parallel solution which produced the same picture , also checking the number of steps
- Checking the given grid sizes pictures with the sequential and parallel solution, also adding different grid sizes (253, 317, 535, 690 and 762).

**Benchmark**

**Steps**

Select an Algorithm between the sequential and parallel solution

Choosing the Architectures between your computer(mine has 2 cores) and server computer(has 4 cores)

Defining the Range of Data Sizes and creating their csv's (8, 65, 253, 317, 535, 690, 762, 1001 different grid sizes)

Execution and recording time(running the solution on the chosen architecture and recording the time it takes using  CurrentTimeMillis in Java).

Data Collection: Collect the results for both architectures with different solutions ran and different defined grid sizes

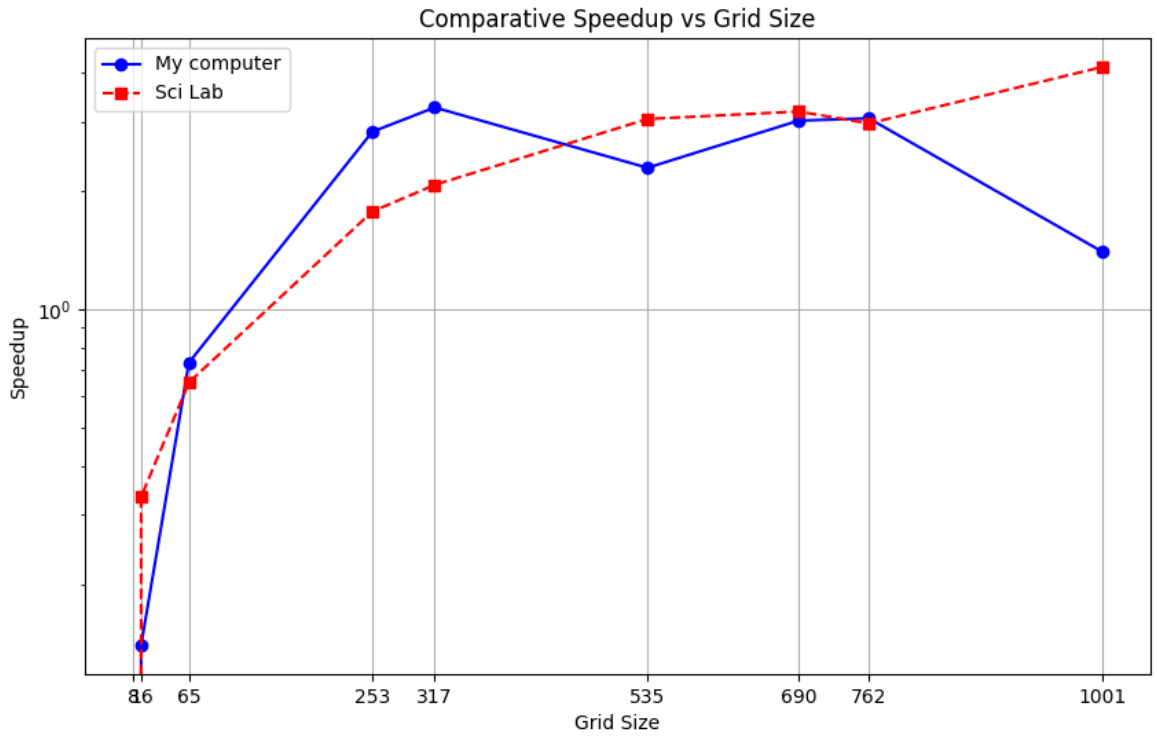
Calculating Speedup then graph the results

Concluding on which computer has the best speed up

**Machine Architecture.**

Server Computer - (4 cores, 8 logical)

My computer – (2 cores, 4 logical processors)



## 2.1 Range of Grid Sizes for Optimal Performance

- **My computer:** The best performance is observed for grid sizes around 253 and 317, with a significant drop in performance at size 690.
- **SciLab:** The SciLab machine performs well across a broader range of grid sizes, with maximum speedup occurring at size 1001.

## 2.2 Maximum Speedup

**My computer:** The maximum speedup observed is around 3.26 at grid size 317, which is reasonable given the dual-core nature of my machine. It's slightly below the ideal speedup of 2x(Physical cores), likely due to thread management overhead and less-than-ideal parallel workload distribution. My laptop has 4 logical cores (with hyper-threading). The observed speedup is **3.26x**, which is close to the theoretical maximum of 4x (Logical Processors). This indicates that my program is effectively utilizing most of the available processing power, though it does not reach the theoretical maximum.

- **Server Computer:** The maximum speedup is 4.13 at grid size 1001, which is close to the ideal speedup of 4x (Physical cores). The server Lab system has 8 logical cores with hyper threading. The peak observed speedup is about half of the theoretical maximum of 8x(Logical processors). This suggests that while the parallel algorithm performs well, there may be factors limiting full utilization of all cores.

## 2.3 Reliability of Measurements

- **Reliability:** The measurements appear reliable based on consistent patterns observed in the speedup data, particularly on the Server machine.
- Reliability of measurements
  I each grid size 10 times and recorded the averages which makes my experiment reliable because I'm using the average for grid size and the certain method I used.

### 2.4 Trends/ Anomalies

- The trend follows an exponential graph that reaches are a maximum point for both my computer and the server computer**.** The significant drop in speedup at grid size 1001 on my computer
- Anomalies occur due to issues such as overheads and data races (When I'm reading and writing data entries)
- Anomalies are also caused by Grid Size and Workload Distribution (The workload is not evenly distributed across all cores)

### 3. Conclusion

- Larger grid sizes (e.g., 762, 692, and 1001) on Server Computer show the best speedup due to better workload distribution across multiple cores.
- The best speedup on my computer is when grid size is 317, this might be due to the threshold and the number of cores my pc has being able to evenly distribute the work and making use of the resources my computer has.
- **Worthwhile Parallelization:** Parallelization is worthwhile, especially on machines with more cores and larger grid sizes, where the computational workload can be effectively divided and processed concurrently. The speedup observed on Server Computer approaches the theoretical maximum, demonstrating the benefits of parallelization for large-scale problems.