Parallelization Approach

There are 3 classes, TerrainArea, SearchParallel and MonteCarloMinimizationParallel which is the main class.
SearchParallel have 2 constructors --> 1) public SearchParallel(int id)
2) public SearchParallel(int start,int end)
The shared objects/fields: SearchParallel [] array, int min_height, TerrainArea area and x y position of the min_height.
Local fields: posX, posY (1)SearchParallel

Objects created using (1)SearchParallel are stored in the shared array. The integer start and end in (2) SearchParallel represent the starting and the ending search index, indexing the shared array objects.
SearchParallel class extends RecursiveAction class from java.util.concurrent package.

The base condition for the void compute() method is that, the difference between the end index and start index should be less or equal to the SEQUENTIAL_CUTOFF.
Then, the array is searched for the (1)SearchParallel obj that is positioned(posX,posY) at at lower height than the current min_height(x,y). Then if found it updates the min_height.

If the base condition is failed, starting and ending index are halved into 2-->(start,mid) & (mid,end) where mid = (end+start)/2. Then 2 (2)SearchParallel objects are created, for which one them calls compute() directly and the other one calls the fork() method.

In my first approach for parallelization, SearchParallel only had one constructor that had SearchParallel [] as a parameter. The arr contained the SearchParallel obj that had an empty arr in their locality. The parallelization was producing correct results in terms of global minimum, but it was super slow due to the fact the call stack had too many copies of the array and the halving of the array was talking time(using for loops).

Correctness Evaluation

There are 3 test functions that were used to test for optimization. The Rosenbrock, EggHolder and McCormick function to test the correctness of the parallel algorithm. Each of these functions have its range where the global minimum is found.

McCormick Function

$$f(x,y) = \sin(x+y) + (x-y)^2 - 1.5x + 2.5y + 1$$

Global Minimum

$$f(-0.54719, -1.54719) = -1.9133$$

Boundary Test Values

$$-3 \leq y \leq 4 \quad -1.5 \leq x \leq 4—$$

Program Output
Arguments : row = col 5000; xmin = -1.5 xmax = 4 ymin= -3 ymax = 4
1.Search Density = 0.1
Parallel: Global Minimum at -19132 at x=-0.55 y=1.26
Serial: Global Minimum at -19128 at x=-0.5 y=-1.5

2. Search Density =0.3
Parallel: Global Min at -19132 at x= -0.55 y= 1.4
Serial: Global Minimum at -19128 at x=-0.5 y=-1.5

Eggholder Function

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\sin\sqrt{|x-(y+47)|}$$

Global Minimum

$$f(512, 404.2319) = -959.6407$$

Boundary Values

$$-512 \leq x, y \leq 512$$

Program Output
Arguments: row = col = 1000 xmin=-512 xmax=512 ymin=-512 ymax=512

1.Search Density=0.1
Parallel: Global minimum: -10522882 at x=510,77 y=499,71
Serial: Global minimum: -9393434 at x=-512,0 y=450,6

2. Search Density = 0.5
Parallel:Global minimum: -10522882 at x=511,59 y=498,69
Serial: Global minimum: -9393434 at x=-512,0 y=450,6

3. Search Density = 1.0
Parallel:Global minimum: -10522882 at x=511,28 y=498,69
Serial: Global minimum: -9393434 at x=-512,0 y=450,6

Rosenbrock Function

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

Global Minimum

$$f(1.0, 1.0) = 0$$

Boundary Values

$$-1.5 \leq x \leq 1.5$$
$$-1.5 \leq y \leq 1.5$$

Program Output
Arguments: row = col = 1000 xmin=-1.5 xmax=1.5 ymin=-1.5 ymax=1.5
1.Search Density=0.1
Parallel:Global minimum: 0 at x=0,34 y=1,01
Serial: Global minimum: 0 at x=1,0 y=1,0
2.Search Density=0.5
Parallel: Global minimum: 0 at x=1,31 y=1,00
Serial:Serial: Global minimum: 0 at x=1,0 y=1,0

3.Search Density = 1.0
Parallel: Parallel: Global minimum: 0 at x=1,00 y=1,00
Serial:Serial: Global minimum: 0 at x=1,0 y=1,0

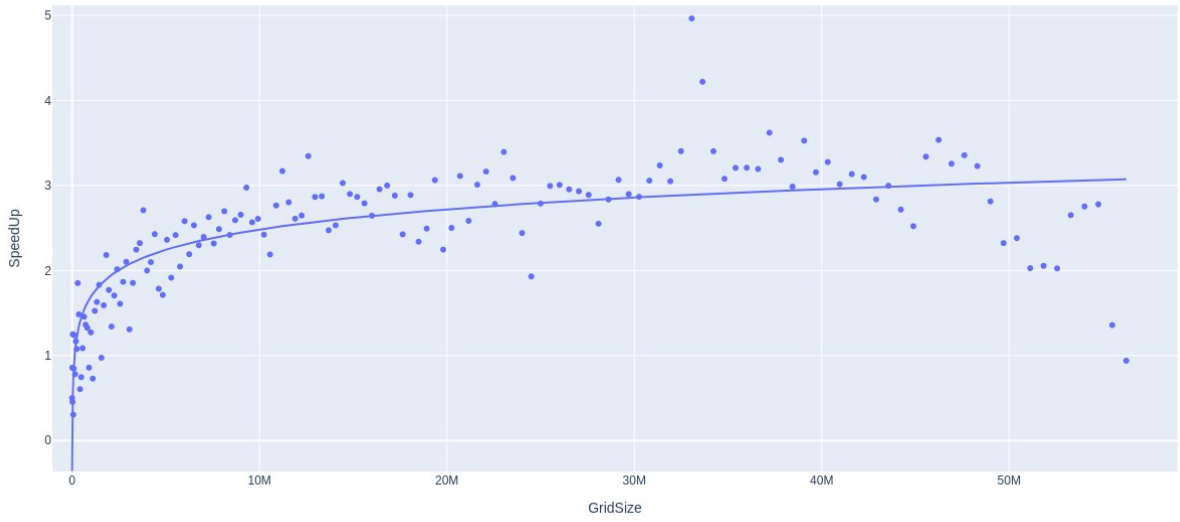
Efficiency Evaluation

Computer Architecture 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):                  4
  On-line CPU(s) list:   0-3
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i3-8145U CPU @ 2.10GHz
    CPU family:          6
    Model:               142
    Thread(s) per core:  2
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            12
    CPU max MHz:         3900.0000
    CPU min MHz:         400.0000
    BogoMIPS:            4599.93
```

SpeedUp Graphs

Title: Parallel algo's speedUp dependency on the grid size
SpeedUp = T1/Tp



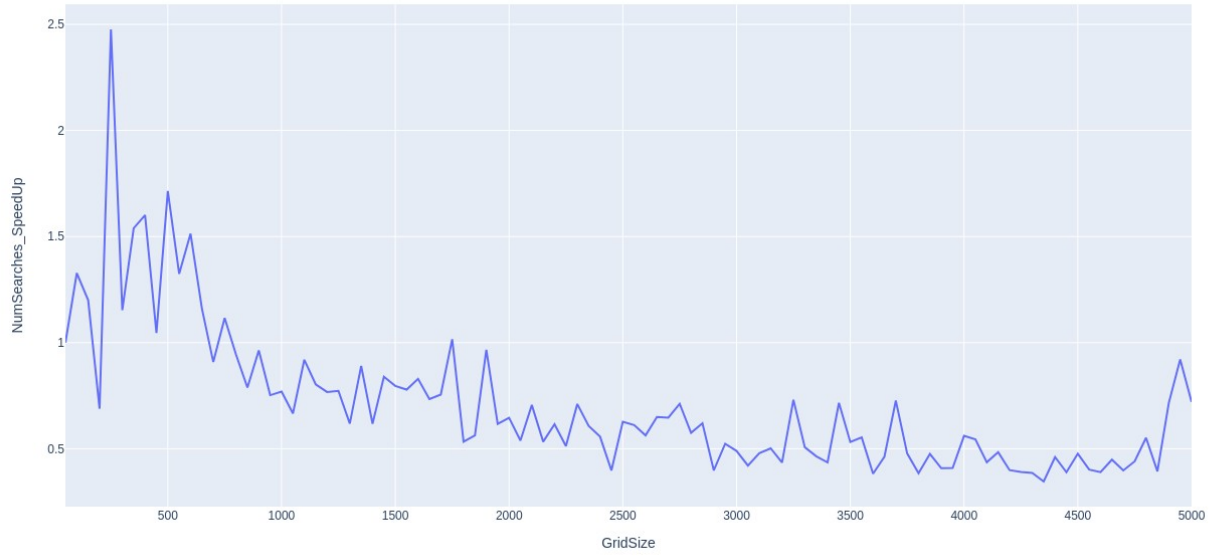
The graph above shows how the parallel algorithm speeds up as the grid size increases.
The graph shows Amdahl's law because as soon as the grid size passes 50M(row * col), the efficiency of the algorithm deteriorates. The Parallel algorithm reaches a mean maximum speedup of ~3.5

There’s an outlier with speedup of ~5. The outlier(s) are caused by the shift focus of the CPU. The CPU are doing other work while the programs are running, for example, having many desktop application opened while running the programs disturbs the focus of the CPU.

The ideal speedup is reached when the grid size ~35M, that is row = col= root of 35M. At this grid size, the ideal speedup should be so much high in value, >1000.

I have tested my methods several times, and I created a python file that takes the mean value of the given data. For each grid size, I ran the code n(arg specified) times, and took the mean of all the values output. Therefore, I can say my results are reliable.

Title:Variation of serial’ search speed with respect to parallel’s search speed



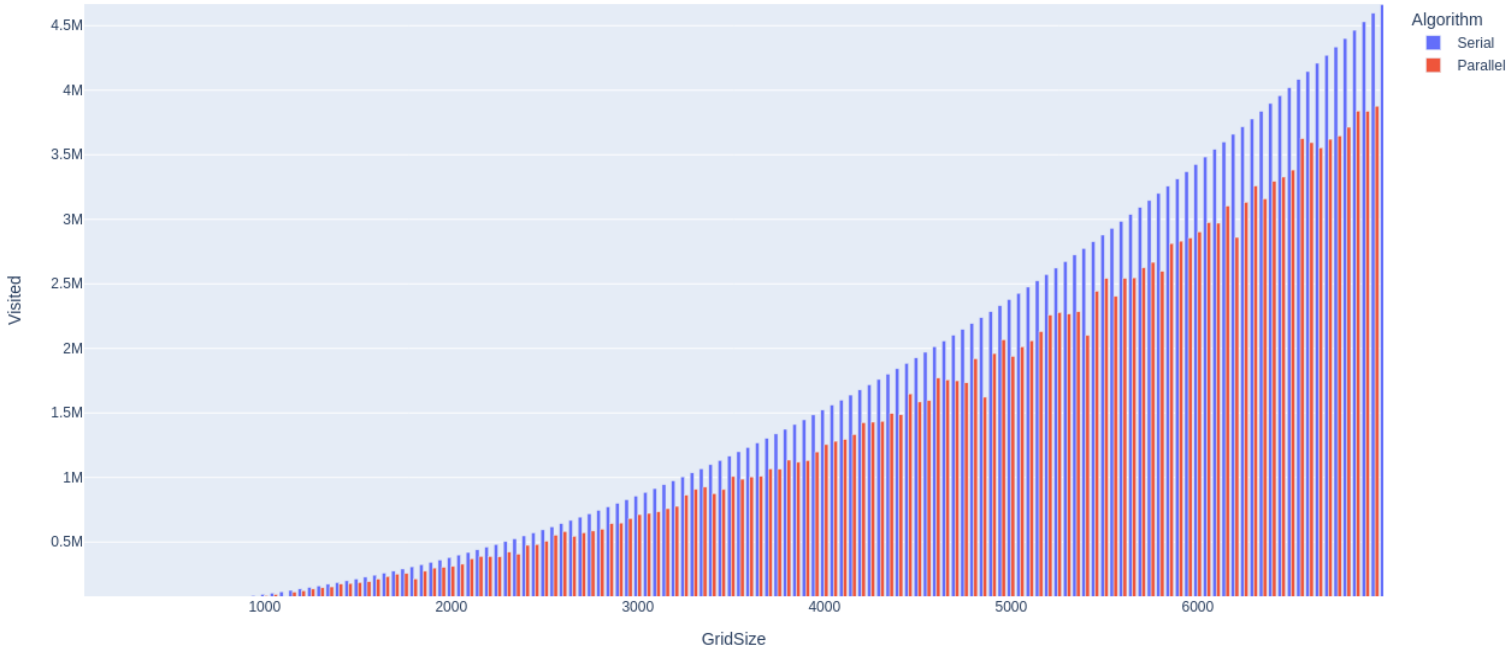
The search speedup varies logarithmically to respect to grid size.

Computer Architecture 2

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         40 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Xeon(R) CPU           E5620  @ 2.40GHz
    CPU family:          6
    Model:               44
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            2
    Frequency boost:     enabled
    CPU max MHz:         2401.0000
    CPU min MHz:         1600.0000
    BogoMIPS:            4788.37
```

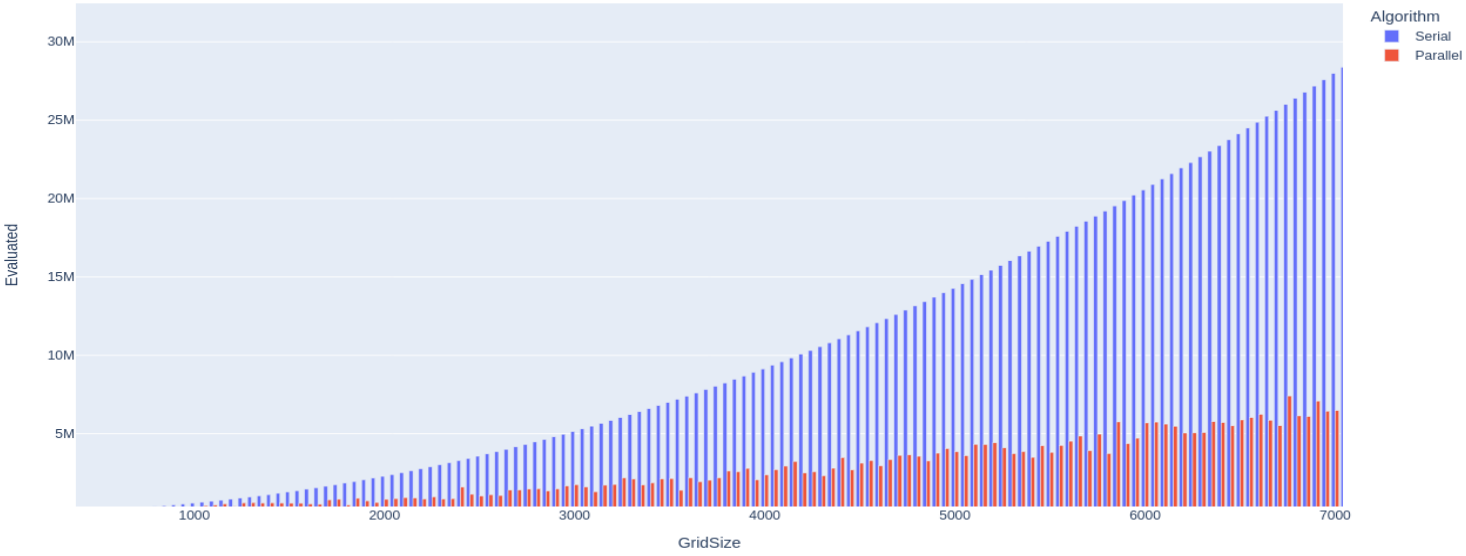
Graphs

Title: Number of visited grid for different grid sizes



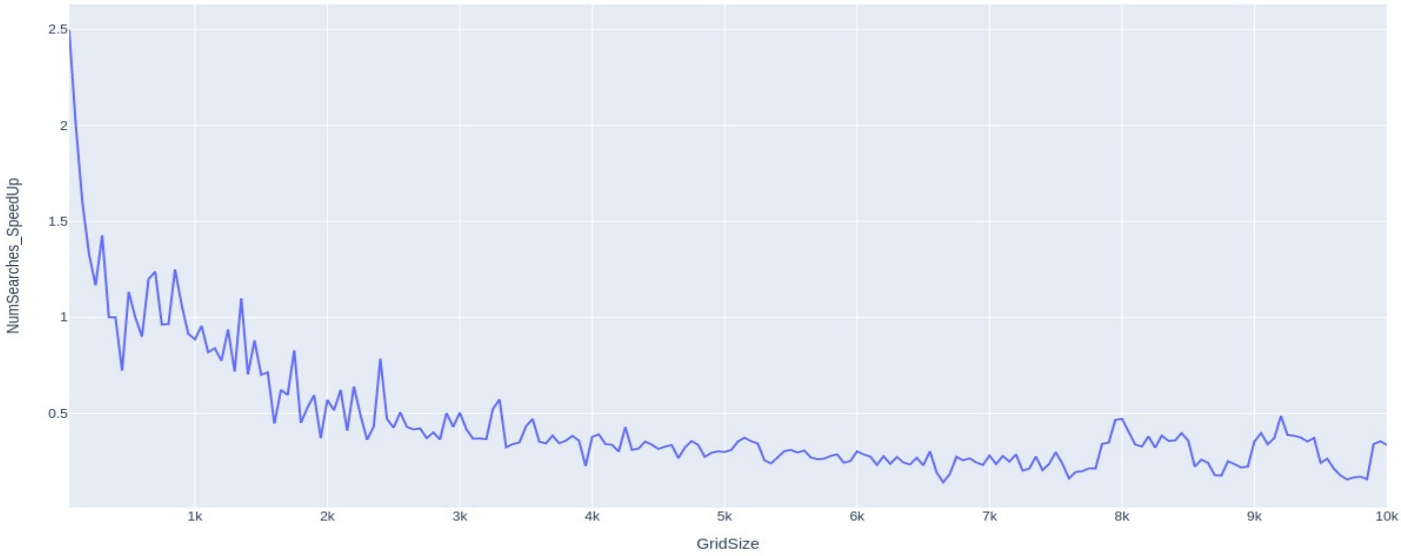
Unlike serial version, the parallel version does not visit each and every grid on the area.
One of the reason why the parallel version is more efficient than serial version.

Title: Number of evaluated grids for given grid sizes



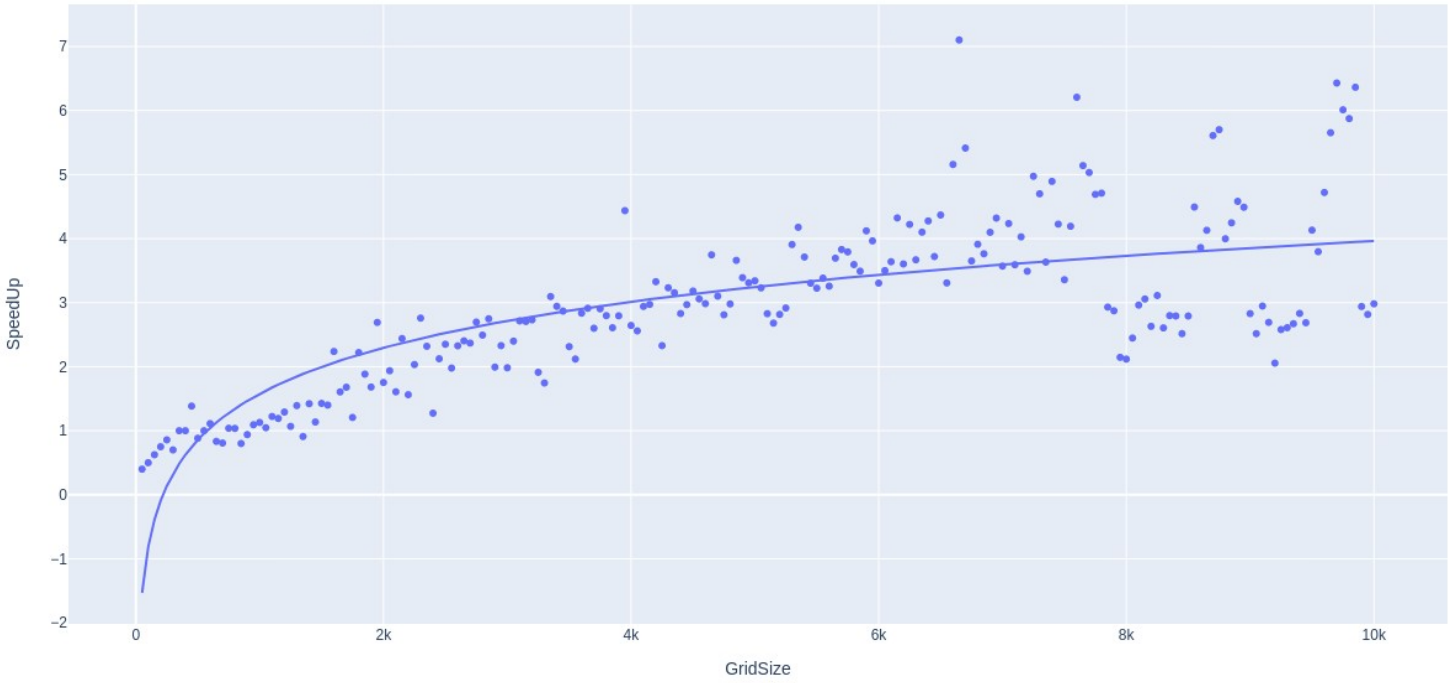
The parallel version does not evaluate every grid on the terrain area, making it more efficient.

Title: Variation of search speed of serial with respect to parallel search speed



As the grid size is getting larger, its gets clearer that the search speed(total searches/time taken) of the parallel version is about 4 times of the serial version.

Title: Variation of Serial' time with respect to of Parallel' time



As the grid size approaches larger values, there's a lot of noise on the speed up. This could have been caused by the fact that the CPU's focus is disturbed.

Conclusions

Monte Carlo methods are computational techniques that makes use of randomness to solve mathematical problems.
The problem of the assignment concerns with the optimization, making use of Monte Carlo methods. Without going through the entire set, randomly choose certain elements of the set(ordered),to find or approximate an element with certain qualities.

For the first few reads of the graph, if you were to extrapolate the graph, you'd see that the graph is linear(time speedUp), so that means that the algorithm is indeed scalable.

From the graphs, you can see that Amdahl's law is indeed obeyed. You cannot make the entire program to be parallel, because the other functions within the program that cannot be parallelized by nature, for example, the instantiating of the SearchParallel array can only be done in a sequential order.
That is to say, the maximum improvement perfomance of the system is limited by the part of the system that is strictly sequential.

As seen from bar graphs, one of the reasons that plays role in time efficiency of the parallel version is that, the program does not evaluate or visit every grid on the area, as visiting/evaluating every grid is a mad computational expense.

I therefore conclude that, it is worth parallelizing the Monte Carlo Minimization problem as the parallelization does not only lead to a more efficient time speed, but also with a better approximation, as can be seen from the correctness evaluation.