The optimization strategy regarding Single Core optimization will be the following:

- Memory Alignment (64byte for AVX512) for better vectorization.
- Optimization of DGEMM() with intrinsics for faster calculations.
- Matrix Padding will be implemented for all Matrices which are used in DGEMM() to get matrices of a height/width divisible by 8 (8 doubles * 64 bit each = 512 bit for vectorization). Since there are different scenarios for DGEMM with different parameters, there might be introduced different variants of DGEMM() functions for specific use-cases. So that an optimal version of DGEMM for the specific padded matrices will be used.
- Roughly 90% of the values in GlobalMatrices.cpp are zeros, so these matrices are pretty sparse. These matrices are only used directly in DGEMM() or in computeFlux() where they are also used in DGEMM(). They always end up being used as the parameter A in DGEMM. The goal here is to find the pattern in the global Matrices, to save up to 90% of these calculations in the best case. This is possible because in DGEMM() A is only used for summed up multiplications. And when the value is zero, the calculation can be skipped. Also Matrix Padding might be applied to GlobalMatrices, since they are used in DGEMM().
- Divisions are usually only performed on Scalars and not in heavy used Kernels or in most-inner loops. But some Divisions are performed, even if their result never changes, since only constants or variables of global are used. These calculations will be replaced with precalculated constants calculated at the beginning of the execution. For example -globals.hx / (globals.hx * globals.hy) (Simulator.cpp Line 53) will be precalculated.
- Structure of Arrays instead of Array of Structures is already implemented with the help of Grid<> Template. This will not be touched.

MPI

1. Provide asynchronous communication by non-blocking communication. This done by using MPI_ISend and MPI_IRecv. To see if the communication has finished MPI_Wait() is called.

MPI_ISend : identify an area in memory to serve as a send buffer. This process continue immediately without waiting for the message to be copied out from application buffer.

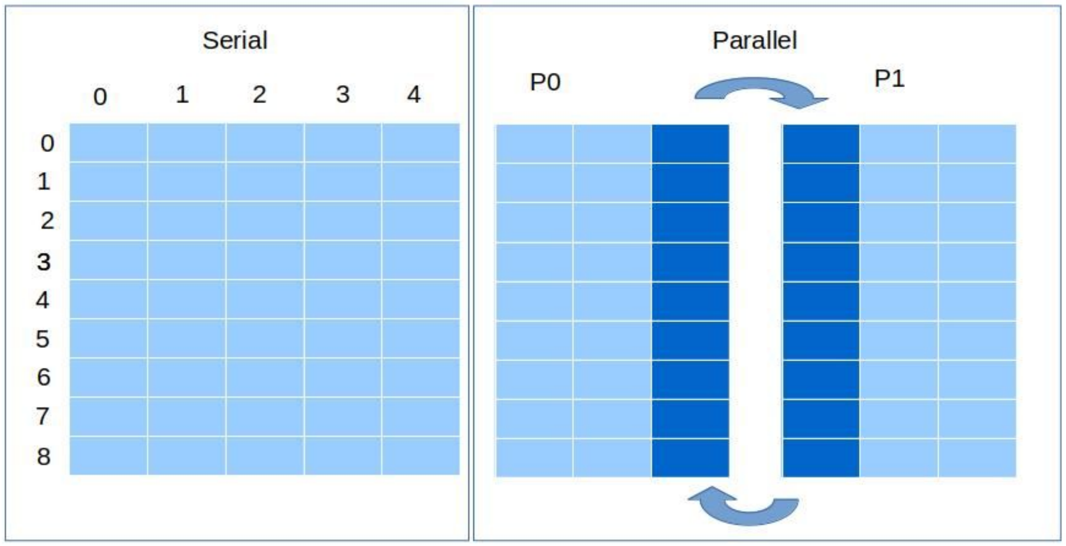
MPI_IRecv : identify an area to server as receive buffer. This process continue without waiting the message to be received & copied into application buffer.

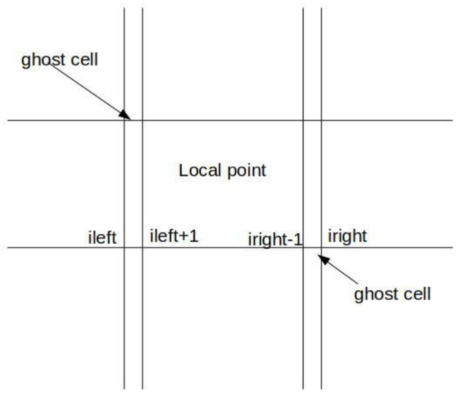
```
MPI_Request request[4];
MPI_Status status[4];

/*receive from left neighbor*/
MPI_Irecv(&xlocal(ileft,0), n, MPI_DOUBLE, left,
tag_right, MPI_COMM_WORLD, &request[0]);

/* receive from right neighbour*/
```

```
MPI_Irecv(&xlocal(iright,0), n , MPI_DOUBLE, right,
tag_left, MPI_COMM_WORLD, &request[1]);

/* send new value to right neighbour async */
MPI_Isend(&xnew(iright-1,0), n, MPI_DOUBLE, right,
tag_right, MPI_COMM_WORLD, &request[2]);

/* send new value to left neighbour async */
MPI_Isend(&xnew(ileft+1,0), n, MPI_DOUBLE, left,
tag_left, MPI_COMM_WORLD, &request[3]);
```

At the end of application, we have to ensure that all process has finished the execution therefore MPI_Waitall is called.

2. Using ghost cell:
Parallelizing problem of structured grid of points that are updated repeatedly based on the values of a fixed set of neighboring points in the same grid sometimes are processed by different processors. This cause the neighboring chunks that are needed for the calculation are located in different processes. By implementing Ghost cell communication overhead can be reduced.
For example we have a serial grid of data:





3. All reduce for the errors
MPI_Allreduce(&error_local, &error, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
By calling this make process zero write out the value of the error.

OMP

- Omp parallelization will cover GEMM and Simulator. For loops are parallelized with associated pragmas.
- The main function will also be investigated for possible parallelization speedups.
- No data dependency is observed in mentioned program sections.
- Tasks will be used in case of an encounter with recursive calls anywhere in the code. Optimal granularity for nesting will be found out empirically and enforced with final clause.