Assignment 3

Erkin Kirdan - Manuel Rothenberg - Gabrielle Poerwawinata

Parallel CG - Introduction

We want to solve the Laplacian, with the conjugate gradient method. (CG-method)

Important Functions:

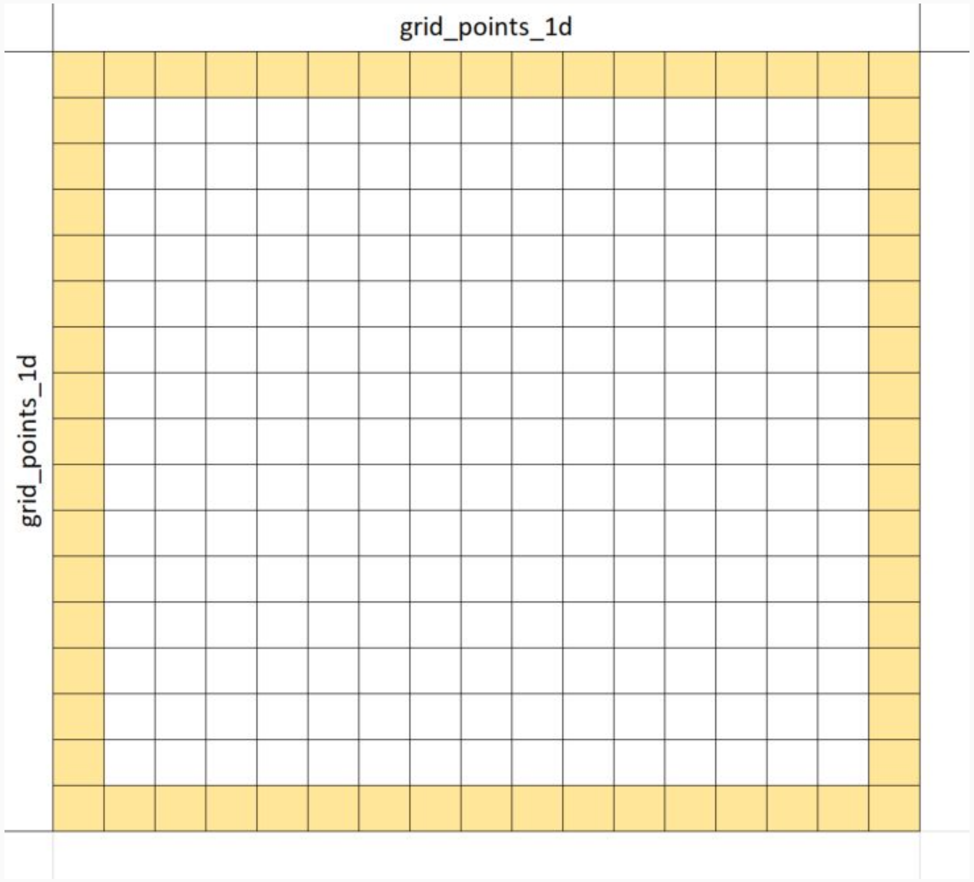
- g_product_operator(grid, result)
- g_scale_add(dest, src, scalar)
- g_scale(grid, scalar)
- g_dot_product(grid1, grid2)

Parallel CG - Introduction

Important Functions:

- operate on the whole grid, but on (mostly) independent data
- modify only the inner cells, border cells are (mostly) not used
- Exception is g_product_operator:
    - uses neighbor cells
    - uses border cells if they are neighbors

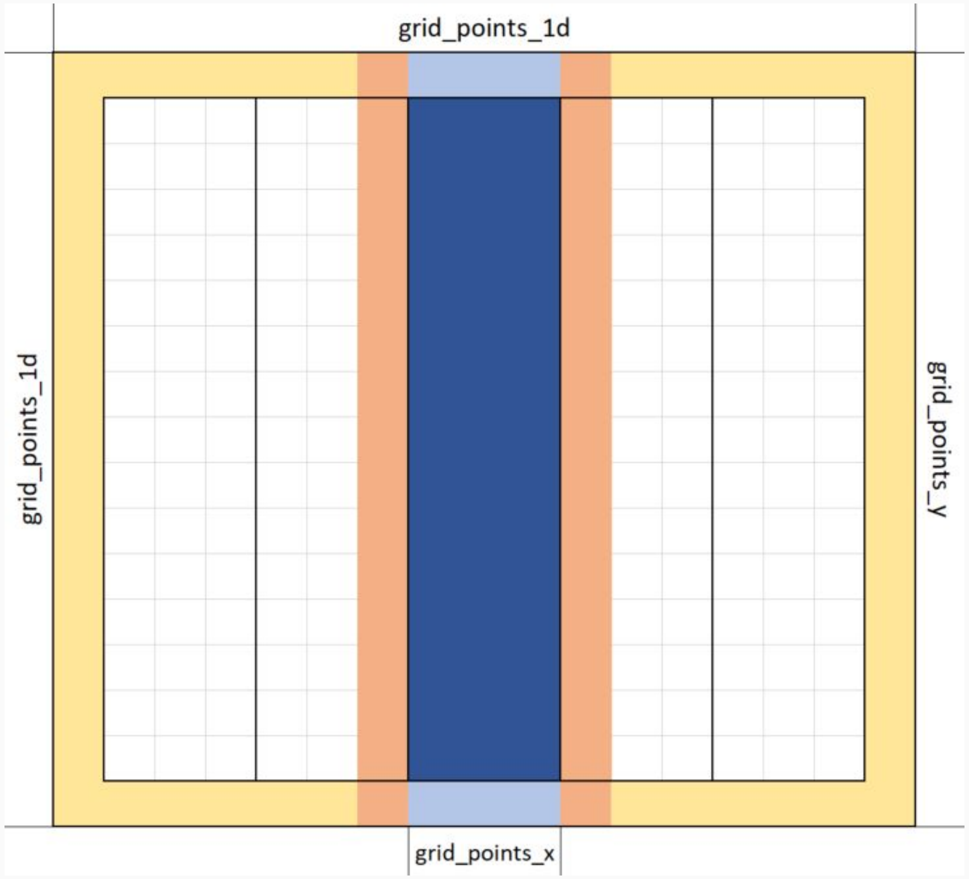
Parallel CG - Introduction



Example Grid:

- mesh_width = 0.0625
- grid_points_1d = 17

Parallel CG - Parallelization



- 1 Controller-tasks
- N-1 Calculation-tasks
- Distribute columns to each task

- MPI Communication to
    - Distribute Sub-Grids
    - Share Border-rows
    - Share Dot-Product Sub-results
    - Collect finale results

Parallel CG - MPI_TYPE_VECTOR

```
MPI_Datatype myVectorType1Col;
```

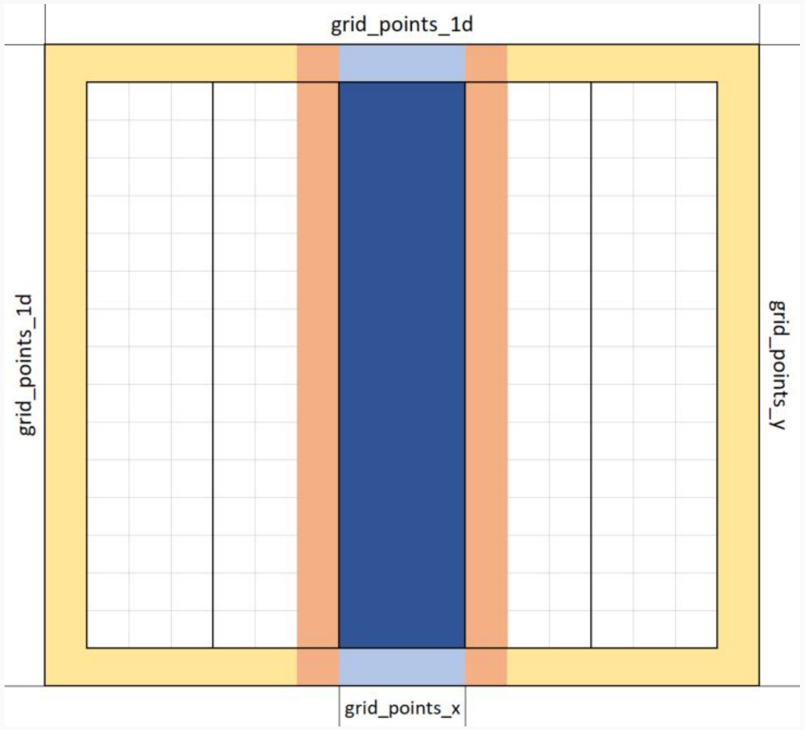
```
MPI_Type_vector(1, grid_points_1d, 0, MPI_DOUBLE, &myVectorType1Col);
MPI_Type_commit(&myVectorType1Col);
```

```
// Send/Receive Border Columns
if(rank > 1){
    MPI_Send(&grid[grid_points_y], 1, myVectorType1Col, rank-1, 0, MPI_COMM_WORLD);
    MPI_Recv(&grid[0], grid_points_y, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
if(rank < size-1){
    MPI_Send(&grid[grid_points_x*grid_points_y], 1, myVectorType1Col, rank+1, 0, MPI_COMM_WORLD);
    MPI_Recv(&grid[(1+grid_points_x)*grid_points_y], grid_points_y, MPI_DOUBLE, rank+1, 0,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

Parallel CG - Parallel Dot-Product

```
// calculate starting norm
// Do task, collect partial results and sum up
if(rank > 0){
    delta_new = g_dot_product(r, r);
    MPI_Send(&delta_new, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
}else{
    delta_new = 0;
    for(int i = 1; i < size; i++){
        double delta_buf = 0.0;
        MPI_Recv(&delta_buf, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        delta_new += delta_buf;
    }
}
// Send/Receive final result
if(rank > 0){
    MPI_Recv(&delta_new, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}else{
    for(int i = 1; i < size; i++){
        MPI_Send(&delta_new, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
    }
}
```

Parallel CG - Configuration Restriction



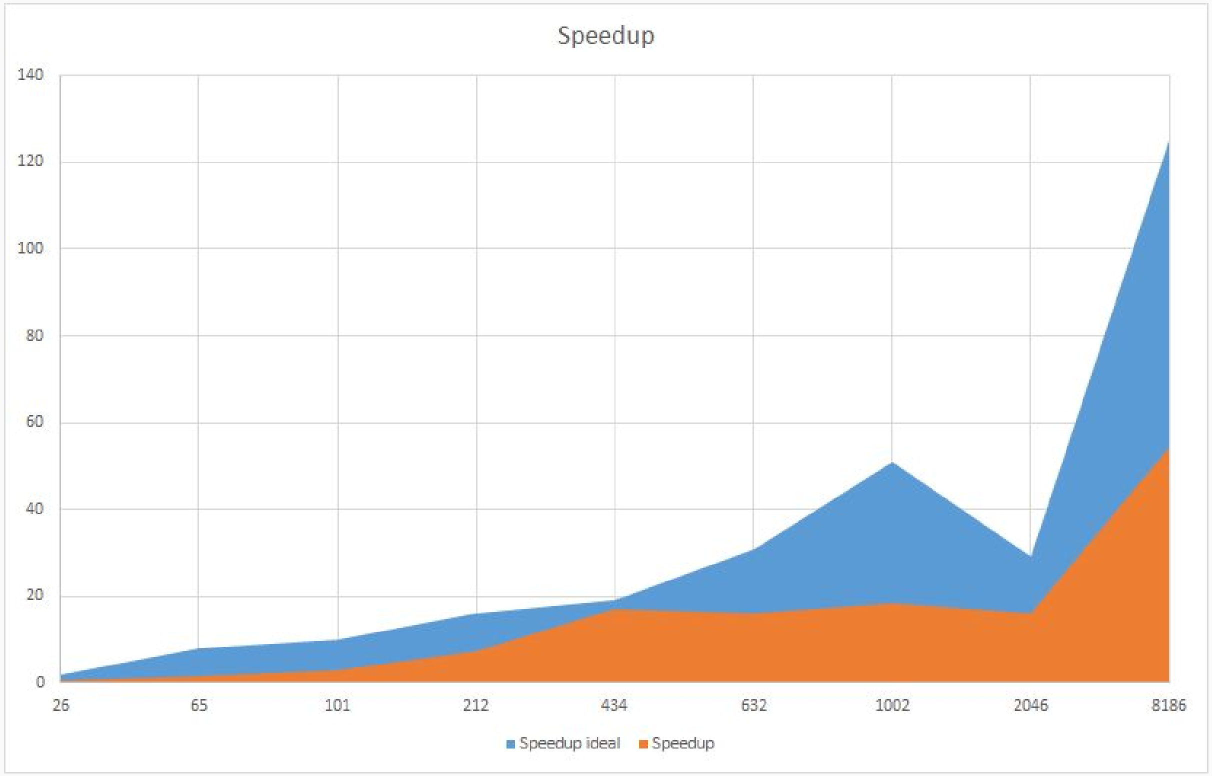
(grid_points_1d - 2) % (size - 1) == 0

Parallel CG - Performance Analysis

Configuration:

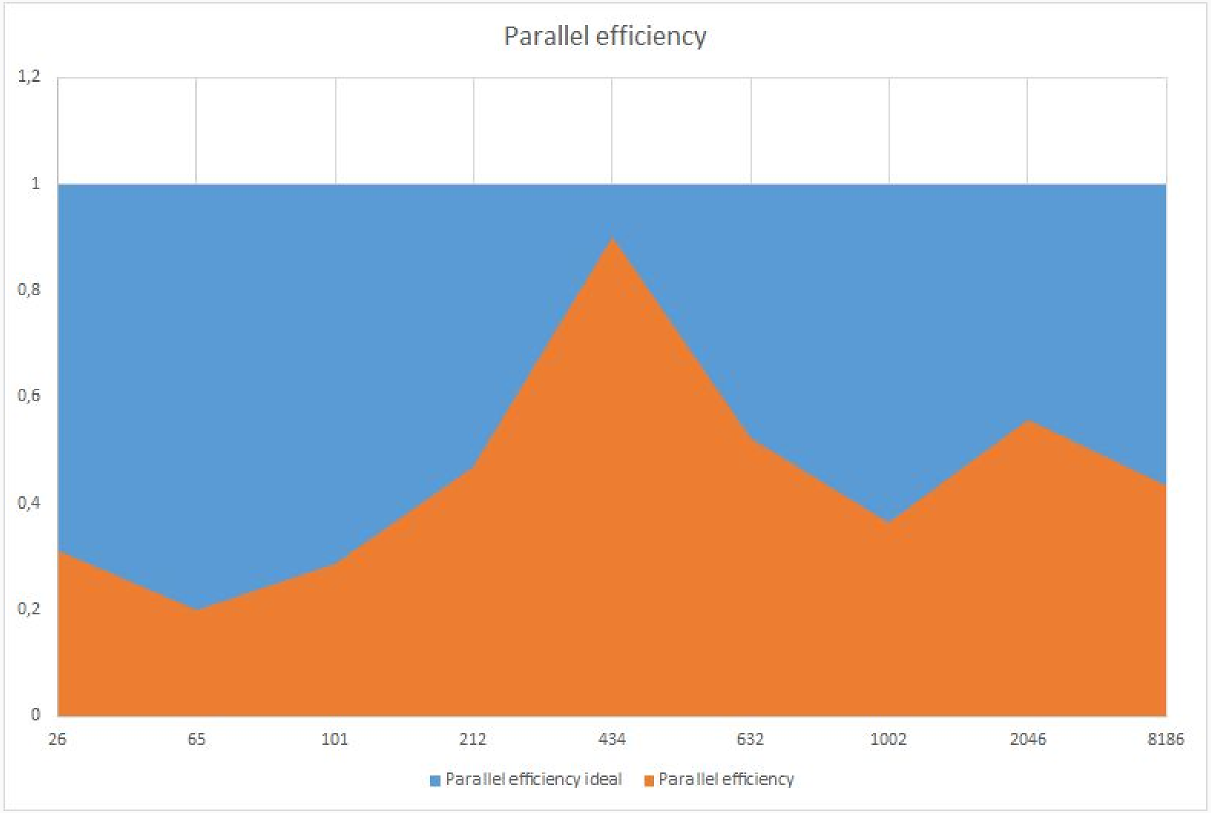
- grid_points_1d between 26 … 8186
- task range between 1 … 1024
- node count between 1 … 16

Parallel CG - SpeedUp



Peak SpeedUp = 54.28

Parallel CG - Parallel efficiency



Peak Parallel efficiency = 0.9

Parallel CG - Performance Analysis

Result:

- Peak SpeedUp increases with problem size
- Balance necessary between task-count and resulting communication
- More tasks -> smaller sub-problem -> more communication overhead