Parallel Programming for HPC - Project

Davide Rossi

University of Trieste

July 8, 2024

Table of contents

- Exercise 1: Distributed matrix-matrix multiplication
- Exercise 2: Jacobi's algorithm with Send-Recv communication
- Exercise 3: Jacobi's algorithm with One-Sided communication

Distributed matrix-matrix multiplication

Code versions

- Basic version with the naive algorithm (triple loop)

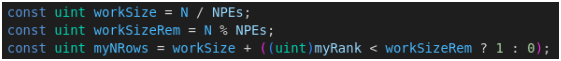
Code versions

- Basic version with the naive algorithm (triple loop)
- Improved CPU version using BLAS library

Code versions

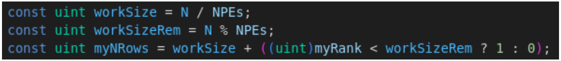
- Basic version with the naive algorithm (triple loop)
- Improved CPU version using BLAS library
- GPU version using CUDA and CUBLAS library

Domain distribution

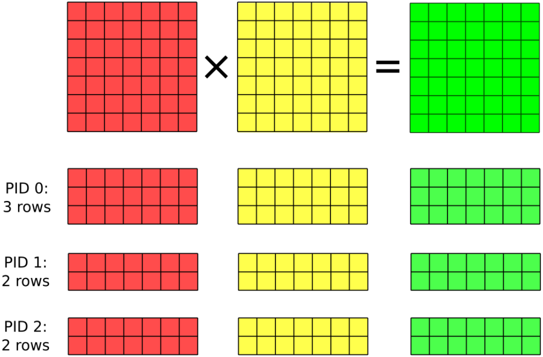


```
const uint workSize = N / NPEs;
const uint workSizeRem = N % NPEs;
const uint myNRows = workSize + ((uint)myRank < workSizeRem ? 1 : 0);
```

Domain distribution

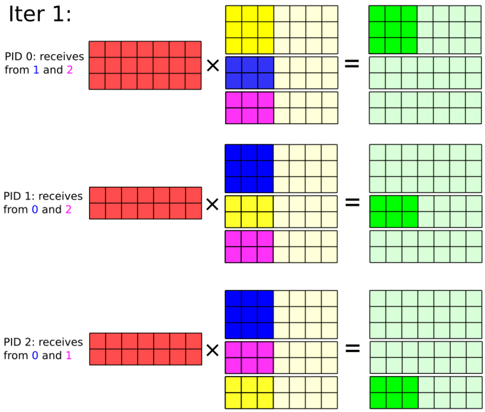


```
const uint workSize = N / NPEs;
const uint workSizeRem = N % NPEs;
const uint myNRows = workSize + ((uint)myRank < workSizeRem ? 1 : 0);
```

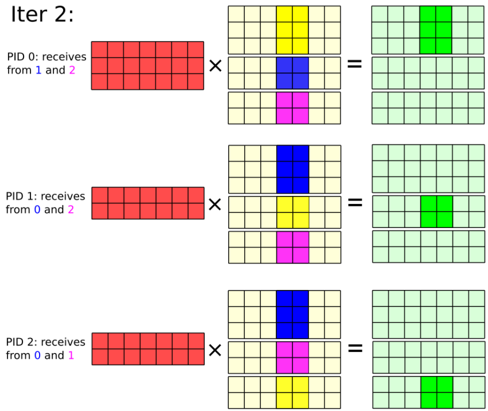


Example: $N = 7, \; NPEs = 3$

Example: first iteration


Iter 1:
PID 0: receives from 1 and 2
×
=
PID 1: receives from 0 and 2
×
=
PID 2: receives from 0 and 1
×
=

Example: second iteration



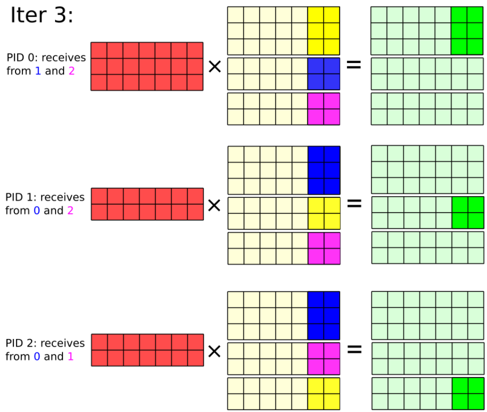
Iter 2:

PID 0: receives from 1 and 2

PID 1: receives from 0 and 2

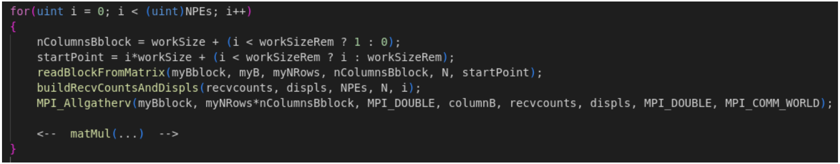
PID 2: receives from 0 and 1

Example: third iteration



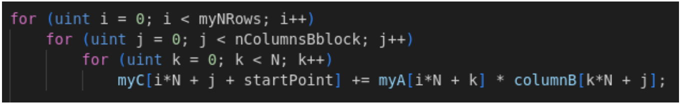
Iter 3:
PID 0: receives from 1 and 2
×
=
PID 1: receives from 0 and 2
×
=
PID 2: receives from 0 and 1
×
=

Main code



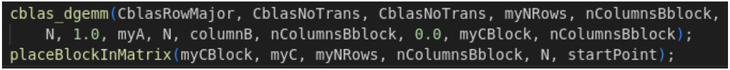
```
for(uint i = 0; i < (uint)NPEs; i++)
{
    nColumnsBblock = workSize + (i < workSizeRem ? 1 : 0);
    startPoint = i*workSize + (i < workSizeRem ? i : workSizeRem);
    readBlockFromMatrix(myBblock, myB, myNRows, nColumnsBblock, N, startPoint);
    buildRecvCountsAndDispls(recvcounts, displs, NPEs, N, i);
    MPI_Allgatherv(myBblock, myNRows*nColumnsBblock, MPI_DOUBLE, columnB, recvcounts, displs, MPI_DOUBLE, MPI_COMM_WORLD);

    <--  matMul(...)  -->
}
```

CPU baseline: naive algorithm



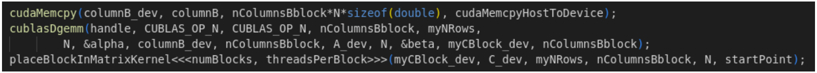
```
for (uint i = 0; i < myNRows; i++)
    for (uint j = 0; j < nColumnsBblock; j++)
        for (uint k = 0; k < N; k++)
            myC[i*N + j + startPoint] += myA[i*N + k] * columnB[k*N + j];
```

CPU improvement: BLAS



```
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, myNRows, nColumnsBblock,
    N, 1.0, myA, N, columnB, nColumnsBblock, 0.0, myCBlock, nColumnsBblock);
placeBlockInMatrix(myCBlock, myC, myNRows, nColumnsBblock, N, startPoint);
```

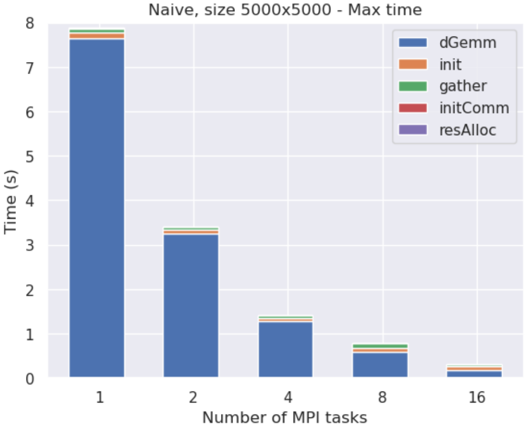
GPU: CUBLAS



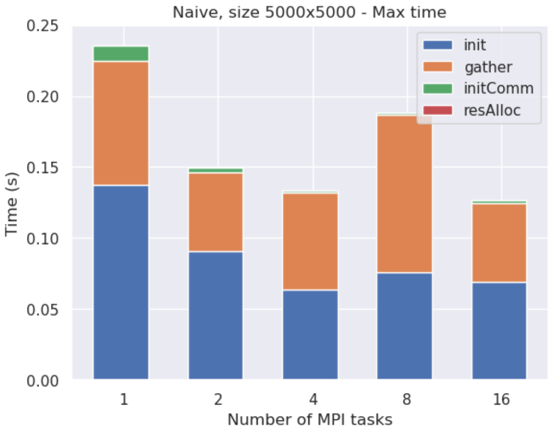
```
cudaMemcpy(columnB_dev, columnB, nColumnsBblock*N*sizeof(double), cudaMemcpyHostToDevice);
cublasDgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, nColumnsBblock, myNRows,
            N, &alpha, columnB_dev, nColumnsBblock, A_dev, N, &beta, myCBlock_dev, nColumnsBblock);
placeBlockInMatrixKernel<<<numBlocks, threadsPerBlock>>>(myCBlock_dev, C_dev, myNRows, nColumnsBblock, N, startPoint);
```

Results

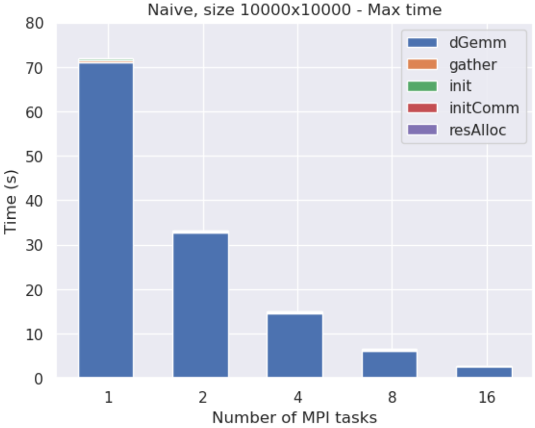
Naive algorithm: size 5000



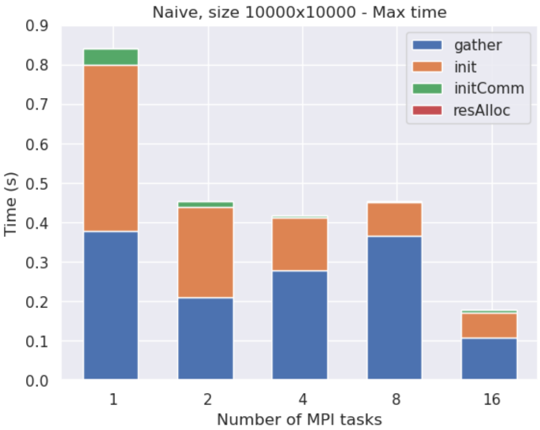
Naive algorithm: size 5000



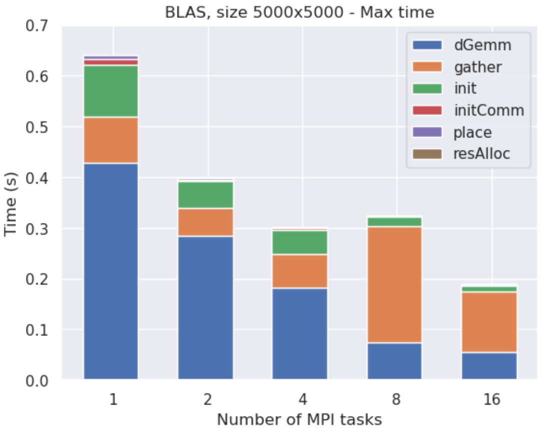
Naive algorithm: size 10000



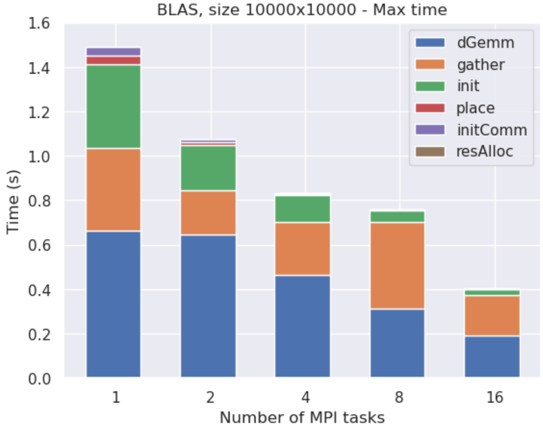
Naive algorithm: size 10000



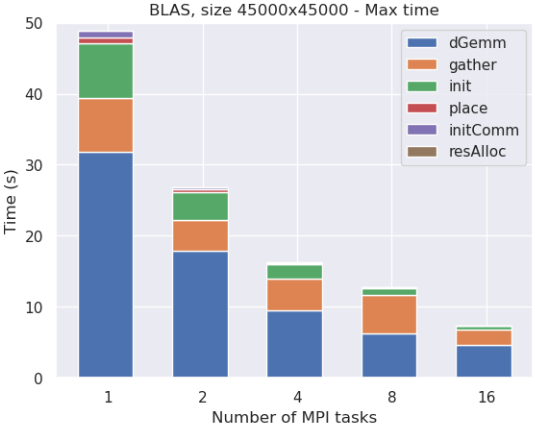
BLAS: size 5000



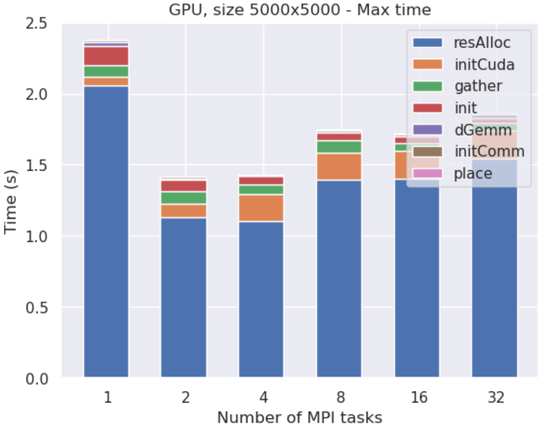
BLAS: size 10000



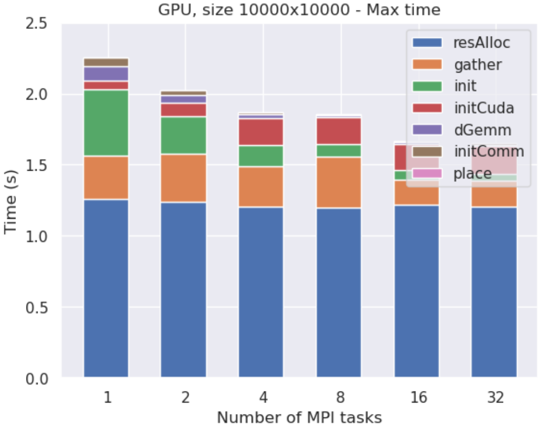
BLAS: size 45000



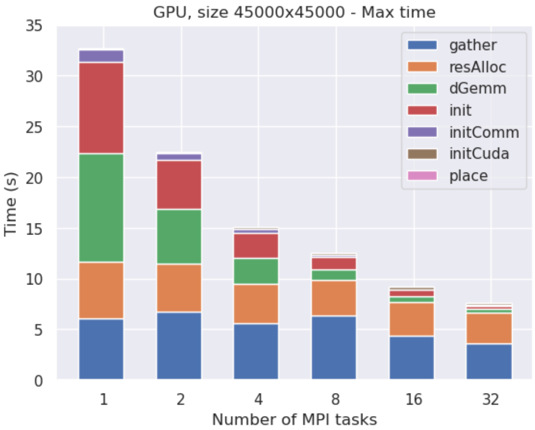
GPU: size 5000



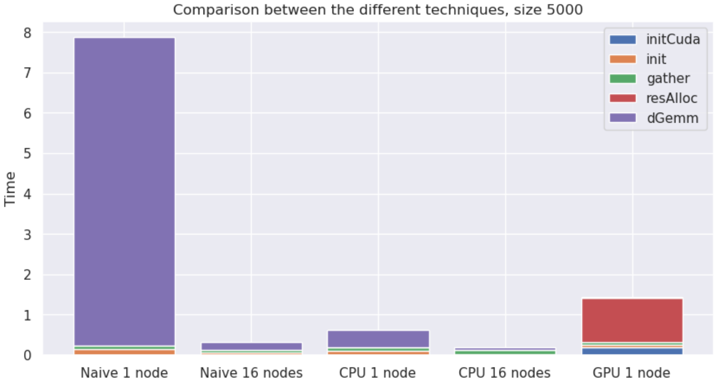
GPU: size 10000



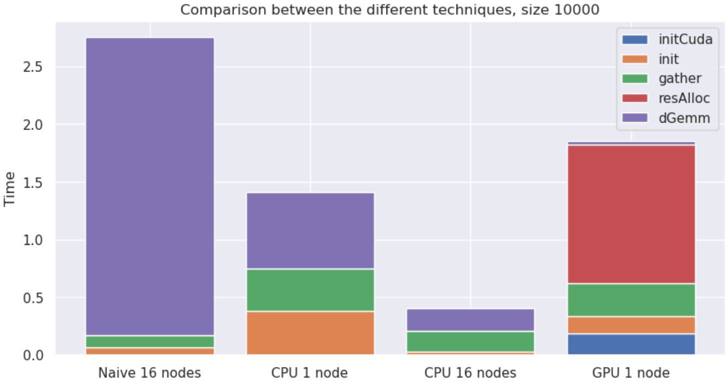
GPU: size 45000



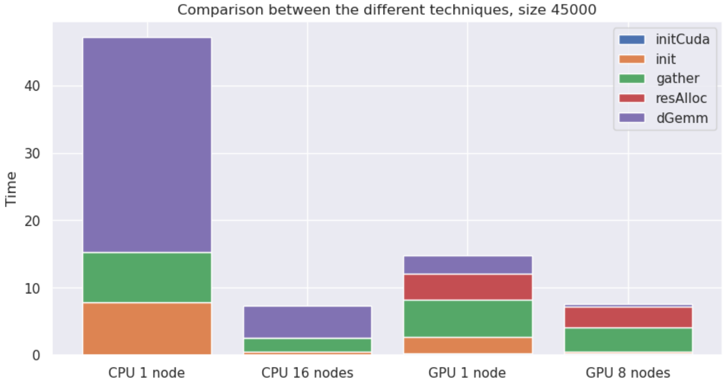
Comparison: size 5000



Comparison: size 10000



Comparison: size 45000



Jacobi's algorithm with Send-Recv communication

Laplace's equation

$$\nabla^2 V = 0$$

Laplace's equation

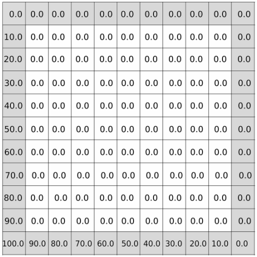
$$\nabla^2 V = 0$$

In $\mathbb{R}^2$:

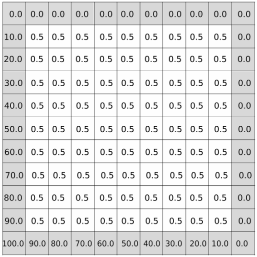
$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0$$

The algorithm

1. Initialize two matrices as:



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100.0	90.0	80.0	70.0	60.0	50.0	40.0	30.0	20.0	10.0	0.0



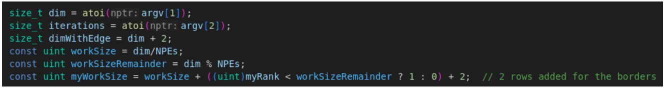
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
20.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
30.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
40.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
50.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
60.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
70.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
80.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
90.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0
100.0	90.0	80.0	70.0	60.0	50.0	40.0	30.0	20.0	10.0	0.0

2. Perform the update as:

$$V_{i,j}^{k+1} = \frac{1}{4}\left(V_{i-1,j}^{k} + V_{i+1,j}^{k} + V_{i,j-1}^{k} + V_{i,j+1}^{k}\right)$$

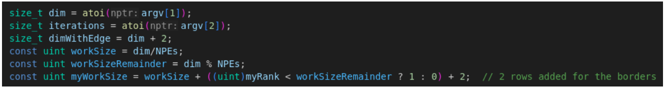
3. swap the pointers of the two matrices and repeat 2 and 3

Domain distribution

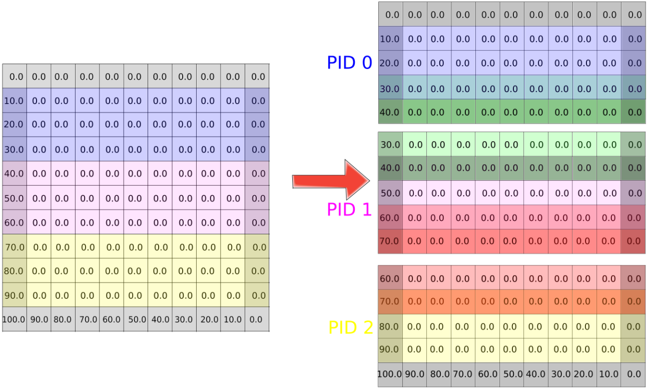


```
size_t dim = atoi(nptr: argv[1]);
size_t iterations = atoi(nptr: argv[2]);
size_t dimWithEdge = dim + 2;
const uint workSize = dim/NPEs;
const uint workSizeRemainder = dim % NPEs;
const uint myWorkSize = workSize + ((uint)myRank < workSizeRemainder ? 1 : 0) + 2;  // 2 rows added for the borders
```

Domain distribution



```
size_t dim = atoi(nptr: argv[1]);
size_t iterations = atoi(nptr: argv[2]);
size_t dimWithEdge = dim + 2;
const uint workSize = dim/NPEs;
const uint workSizeRemainder = dim % NPEs;
const uint myWorkSize = workSize + ((uint)myRank < workSizeRemainder ? 1 : 0) + 2;  // 2 rows added for the borders
```



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100.0	90.0	80.0	70.0	60.0	50.0	40.0	30.0	20.0	10.0	0.0

PID 0

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

PID 1

30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

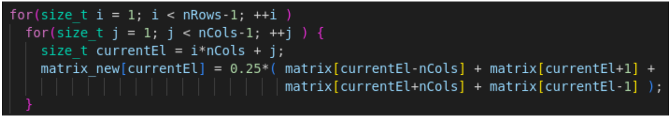
PID 2

60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100.0	90.0	80.0	70.0	60.0	50.0	40.0	30.0	20.0	10.0	0.0

Example: $dim = 9$, $NPEs = 3$

Code

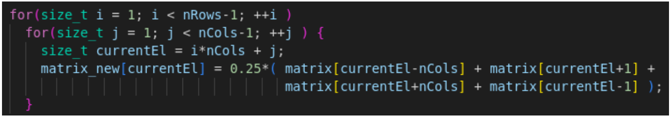
- Update:



```
for(size_t i = 1; i < nRows-1; ++i )
  for(size_t j = 1; j < nCols-1; ++j ) {
    size_t currentEl = i*nCols + j;
    matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                   matrix[currentEl+nCols] + matrix[currentEl-1] );
  }
```

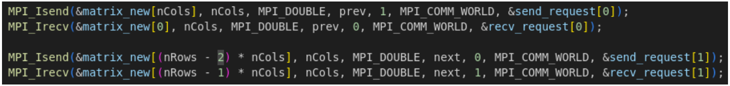
Code

- Update:



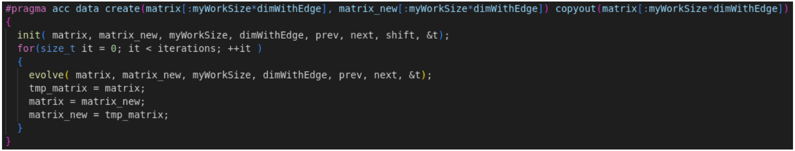
```
for(size_t i = 1; i < nRows-1; ++i )
  for(size_t j = 1; j < nCols-1; ++j ) {
    size_t currentEl = i*nCols + j;
    matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                   matrix[currentEl+nCols] + matrix[currentEl-1] );
  }
```

- Communication:



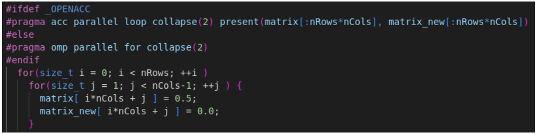
```
MPI_Isend(&matrix_new[nCols], nCols, MPI_DOUBLE, prev, 1, MPI_COMM_WORLD, &send_request[0]);
MPI_Irecv(&matrix_new[0], nCols, MPI_DOUBLE, prev, 0, MPI_COMM_WORLD, &recv_request[0]);

MPI_Isend(&matrix_new[(nRows - 2) * nCols], nCols, MPI_DOUBLE, next, 0, MPI_COMM_WORLD, &send_request[1]);
MPI_Irecv(&matrix_new[(nRows - 1) * nCols], nCols, MPI_DOUBLE, next, 1, MPI_COMM_WORLD, &recv_request[1]);
```

Move to GPU: OpenACC



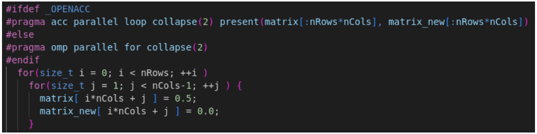
```
#pragma acc data create(matrix[:myWorkSize*dimWithEdge], matrix_new[:myWorkSize*dimWithEdge]) copyout(matrix[:myWorkSize*dimWithEdge])
{
  init( matrix, matrix_new, myWorkSize, dimWithEdge, prev, next, shift, &t);
  for(size_t it = 0; it < iterations; ++it )
  {
    evolve( matrix, matrix_new, myWorkSize, dimWithEdge, prev, next, &t);
    tmp_matrix = matrix;
    matrix = matrix_new;
    matrix_new = tmp_matrix;
  }
}
```

Move to GPU: OpenACC

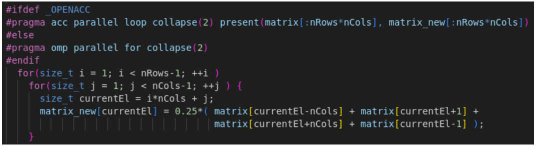


```
#ifdef _OPENACC
#pragma acc parallel loop collapse(2) present(matrix[:nRows*nCols], matrix_new[:nRows*nCols])
#else
#pragma omp parallel for collapse(2)
#endif
  for(size_t i = 0; i < nRows; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      matrix[ i*nCols + j ] = 0.5;
      matrix_new[ i*nCols + j ] = 0.0;
    }
```

Move to GPU: OpenACC

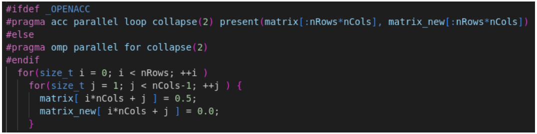


```
#ifdef _OPENACC
#pragma acc parallel loop collapse(2) present(matrix[:nRows*nCols], matrix_new[:nRows*nCols])
#else
#pragma omp parallel for collapse(2)
#endif
  for(size_t i = 0; i < nRows; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      matrix[ i*nCols + j ] = 0.5;
      matrix_new[ i*nCols + j ] = 0.0;
    }
```

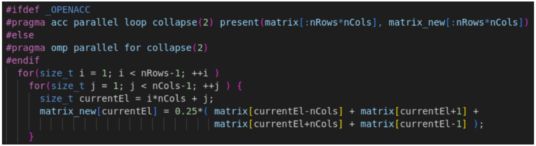


```
#ifdef _OPENACC
#pragma acc parallel loop collapse(2) present(matrix[:nRows*nCols], matrix_new[:nRows*nCols])
#else
#pragma omp parallel for collapse(2)
#endif
  for(size_t i = 1; i < nRows-1; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      size_t currentEl = i*nCols + j;
      matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                     matrix[currentEl+nCols] + matrix[currentEl-1] );
    }
```

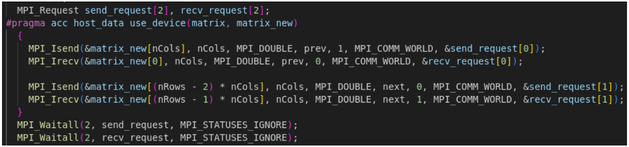
Move to GPU: OpenACC



```
#ifdef _OPENACC
#pragma acc parallel loop collapse(2) present(matrix[:nRows*nCols], matrix_new[:nRows*nCols])
#else
#pragma omp parallel for collapse(2)
#endif
  for(size_t i = 0; i < nRows; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      matrix[ i*nCols + j ] = 0.5;
      matrix_new[ i*nCols + j ] = 0.0;
    }
```



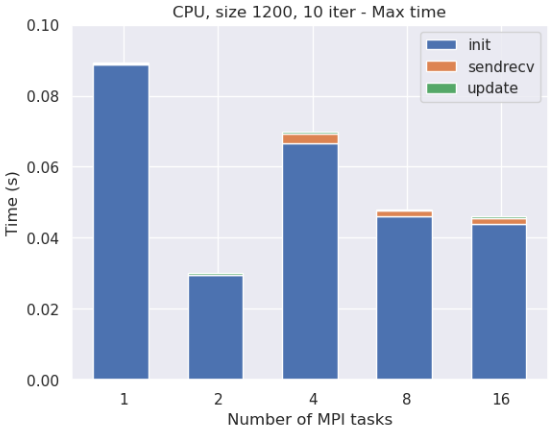
```
#ifdef _OPENACC
#pragma acc parallel loop collapse(2) present(matrix[:nRows*nCols], matrix_new[:nRows*nCols])
#else
#pragma omp parallel for collapse(2)
#endif
  for(size_t i = 1; i < nRows-1; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      size_t currentEl = i*nCols + j;
      matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                     matrix[currentEl+nCols] + matrix[currentEl-1] );
    }
```



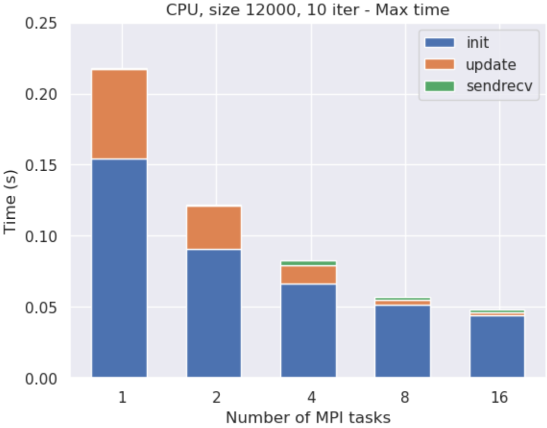
```
  MPI_Request send_request[2], recv_request[2];
#pragma acc host_data use_device(matrix, matrix_new)
  {
    MPI_Isend(&matrix_new[nCols], nCols, MPI_DOUBLE, prev, 1, MPI_COMM_WORLD, &send_request[0]);
    MPI_Irecv(&matrix_new[0], nCols, MPI_DOUBLE, prev, 0, MPI_COMM_WORLD, &recv_request[0]);

    MPI_Isend(&matrix_new[(nRows - 2) * nCols], nCols, MPI_DOUBLE, next, 0, MPI_COMM_WORLD, &send_request[1]);
    MPI_Irecv(&matrix_new[(nRows - 1) * nCols], nCols, MPI_DOUBLE, next, 1, MPI_COMM_WORLD, &recv_request[1]);
  }
  MPI_Waitall(2, send_request, MPI_STATUSES_IGNORE);
  MPI_Waitall(2, recv_request, MPI_STATUSES_IGNORE);
```

Results

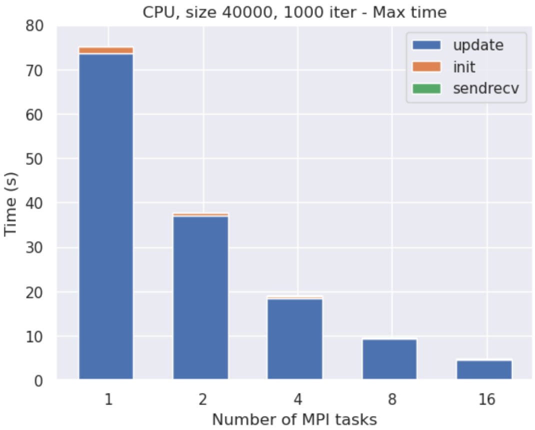
CPU, size 1200, 10 iterations



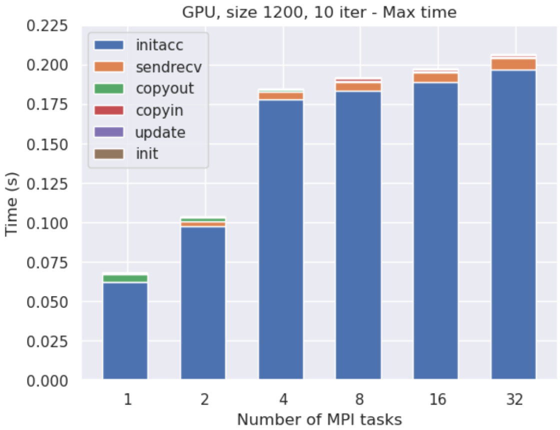
CPU, size 12000, 10 iterations



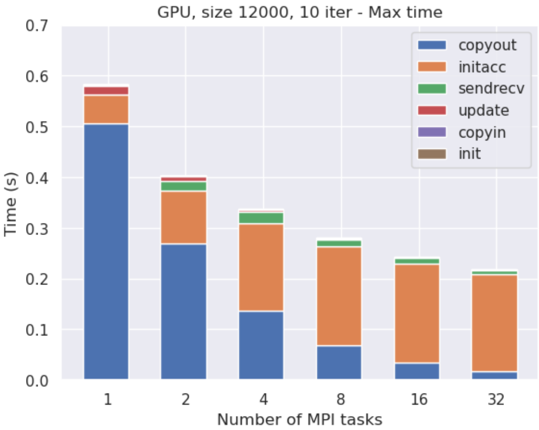
CPU, size 40000, 1000 iterations



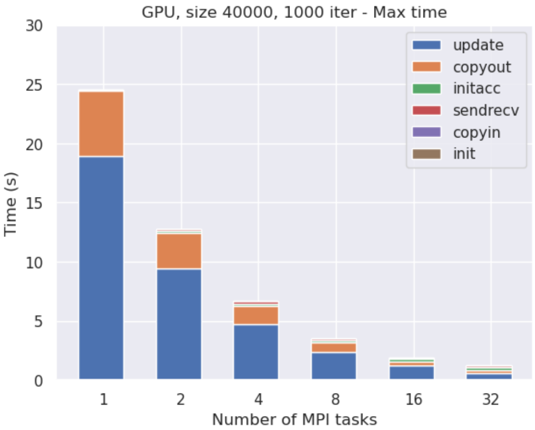
GPU, size 1200, 10 iterations



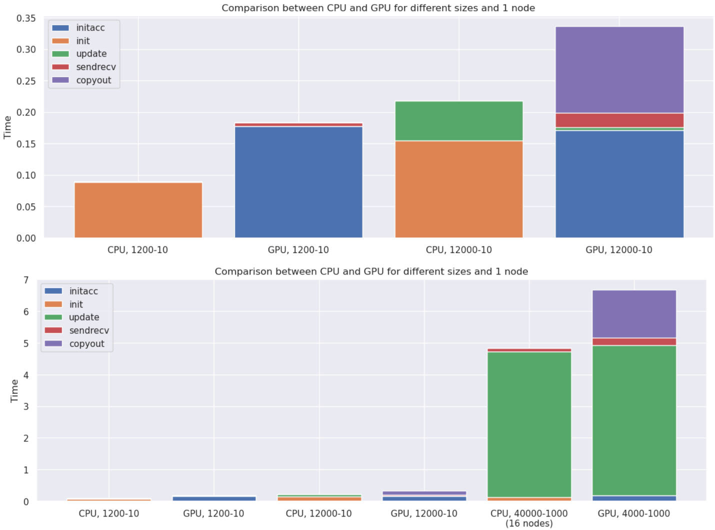
GPU, size 12000, 10 iterations



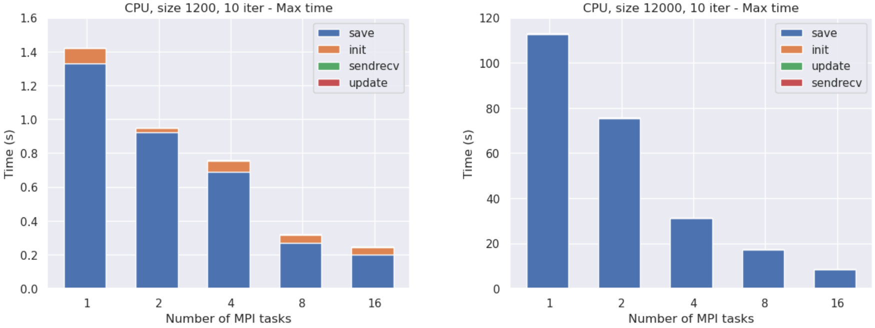
GPU, size 40000, 1000 iterations



Comparison

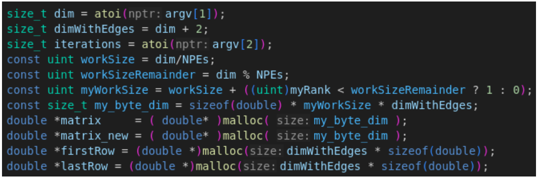

Comparison between CPU and GPU for different sizes and 1 node
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Time
initacc
init
update
sendrecv
copyout
CPU, 1200-10
GPU, 1200-10
CPU, 12000-10
GPU, 12000-10
Comparison between CPU and GPU for different sizes and 1 node
7
6
5
4
3
2
1
0
Time
initacc
init
update
sendrecv
copyout
CPU, 1200-10
GPU, 1200-10
CPU, 12000-10
GPU, 12000-10
CPU, 40000-1000
(16 nodes)
GPU, 4000-1000

Save time


CPU, size 1200, 10 iter - Max time
Time (s)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1
2
4
8
16
Number of MPI tasks
save
init
sendrecv
update
CPU, size 12000, 10 iter - Max time
Time (s)
0
20
40
60
80
100
120
1
2
4
8
16
Number of MPI tasks
save
init
update
sendrecv

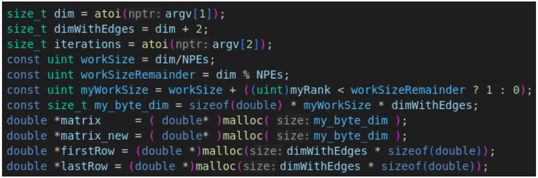
Jacobi's algorithm with One-Sided communication

Domain distribution

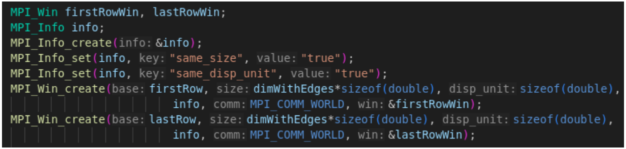


```
size_t dim = atoi(nptr: argv[1]);
size_t dimWithEdges = dim + 2;
size_t iterations = atoi(nptr: argv[2]);
const uint workSize = dim/NPEs;
const uint workSizeRemainder = dim % NPEs;
const uint myWorkSize = workSize + ((uint)myRank < workSizeRemainder ? 1 : 0);
const size_t my_byte_dim = sizeof(double) * myWorkSize * dimWithEdges;
double *matrix     = ( double* )malloc( size: my_byte_dim );
double *matrix_new = ( double* )malloc( size: my_byte_dim );
double *firstRow = (double *)malloc(size: dimWithEdges * sizeof(double));
double *lastRow = (double *)malloc(size: dimWithEdges * sizeof(double));
```

Domain distribution

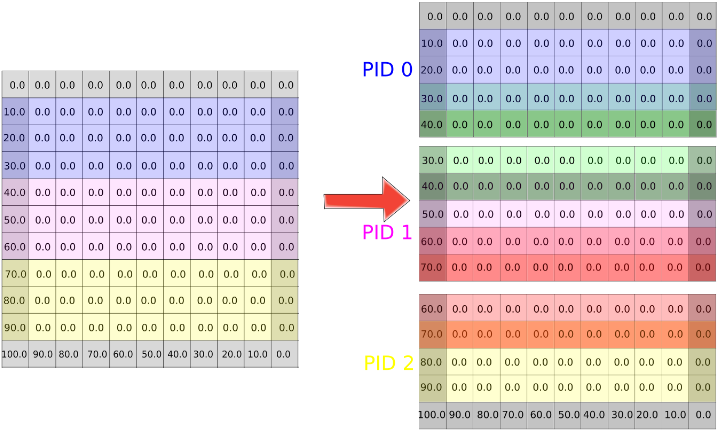


```
size_t dim = atoi(nptr: argv[1]);
size_t dimWithEdges = dim + 2;
size_t iterations = atoi(nptr: argv[2]);
const uint workSize = dim/NPEs;
const uint workSizeRemainder = dim % NPEs;
const uint myWorkSize = workSize + ((uint)myRank < workSizeRemainder ? 1 : 0);
const size_t my_byte_dim = sizeof(double) * myWorkSize * dimWithEdges;
double *matrix     = ( double* )malloc( size: my_byte_dim );
double *matrix_new = ( double* )malloc( size: my_byte_dim );
double *firstRow = (double *)malloc(size: dimWithEdges * sizeof(double));
double *lastRow = (double *)malloc(size: dimWithEdges * sizeof(double));
```



```
MPI_Win firstRowWin, lastRowWin;
MPI_Info info;
MPI_Info_create(info: &info);
MPI_Info_set(info, key: "same_size", value: "true");
MPI_Info_set(info, key: "same_disp_unit", value: "true");
MPI_Win_create(base: firstRow, size: dimWithEdges*sizeof(double), disp_unit: sizeof(double),
                        info, comm: MPI_COMM_WORLD, win: &firstRowWin);
MPI_Win_create(base: lastRow, size: dimWithEdges*sizeof(double), disp_unit: sizeof(double),
                        info, comm: MPI_COMM_WORLD, win: &lastRowWin);
```

Domain distribution



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100.0	90.0	80.0	70.0	60.0	50.0	40.0	30.0	20.0	10.0	0.0

PID 0

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

PID 1

30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

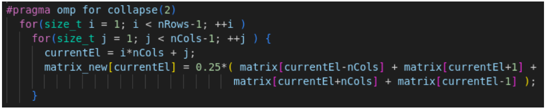
PID 2

60.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100.0	90.0	80.0	70.0	60.0	50.0	40.0	30.0	20.0	10.0	0.0

Example: $dim = 9$, $NPEs = 3$

Code

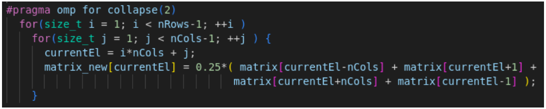
- Update:



```
#pragma omp for collapse(2)
  for(size_t i = 1; i < nRows-1; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      currentEl = i*nCols + j;
      matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                    matrix[currentEl+nCols] + matrix[currentEl-1] );
    }
```

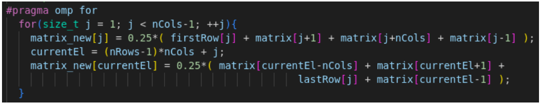
Code

- Update:



```
#pragma omp for collapse(2)
  for(size_t i = 1; i < nRows-1; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      currentEl = i*nCols + j;
      matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                     matrix[currentEl+nCols] + matrix[currentEl-1] );
    }
```

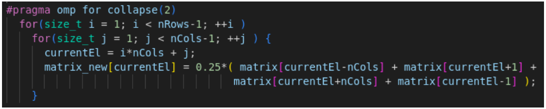
- Update bounds:



```
#pragma omp for
  for(size_t j = 1; j < nCols-1; ++j){
    matrix_new[j] = 0.25*( firstRow[j] + matrix[j+1] + matrix[j+nCols] + matrix[j-1] );
    currentEl = (nRows-1)*nCols + j;
    matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                            lastRow[j] + matrix[currentEl-1] );
  }
```

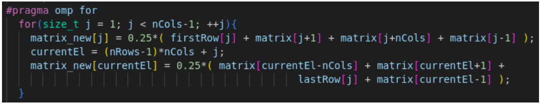
Code

- Update:



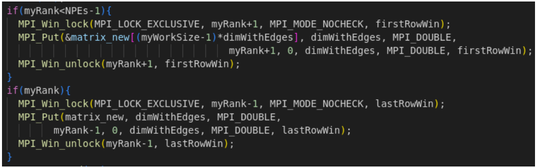
```
#pragma omp for collapse(2)
  for(size_t i = 1; i < nRows-1; ++i )
    for(size_t j = 1; j < nCols-1; ++j ) {
      currentEl = i*nCols + j;
      matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                    matrix[currentEl+nCols] + matrix[currentEl-1] );
    }
```

- Update bounds:



```
#pragma omp for
  for(size_t j = 1; j < nCols-1; ++j){
    matrix_new[j] = 0.25*( firstRow[j] + matrix[j+1] + matrix[j+nCols] + matrix[j-1] );
    currentEl = (nRows-1)*nCols + j;
    matrix_new[currentEl] = 0.25*( matrix[currentEl-nCols] + matrix[currentEl+1] +
                                              lastRow[j] + matrix[currentEl-1] );
  }
```

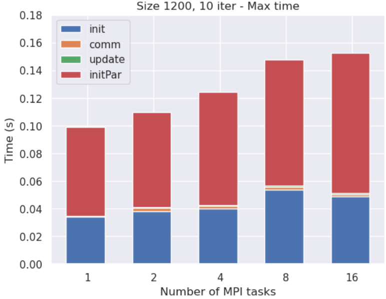
- Communication:



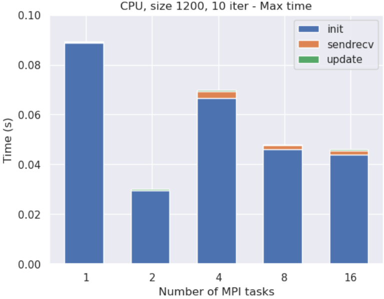
```
if(myRank<NPEs-1){
  MPI_Win_lock(MPI_LOCK_EXCLUSIVE, myRank+1, MPI_MODE_NOCHECK, firstRowWin);
  MPI_Put(&matrix_new[(myWorkSize-1)*dimWithEdges], dimWithEdges, MPI_DOUBLE,
                                  myRank+1, 0, dimWithEdges, MPI_DOUBLE, firstRowWin);
  MPI_Win_unlock(myRank+1, firstRowWin);
}
if(myRank){
  MPI_Win_lock(MPI_LOCK_EXCLUSIVE, myRank-1, MPI_MODE_NOCHECK, lastRowWin);
  MPI_Put(matrix_new, dimWithEdges, MPI_DOUBLE,
        myRank-1, 0, dimWithEdges, MPI_DOUBLE, lastRowWin);
  MPI_Win_unlock(myRank-1, lastRowWin);
}
```

Results

CPU, size 1200, 10 iterations

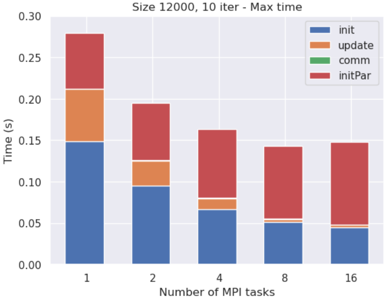


One-Sided comm

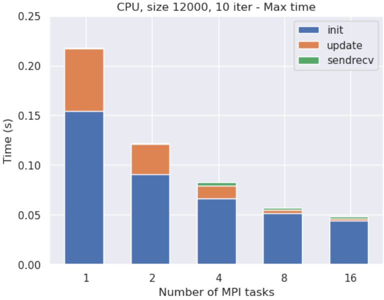


Send-Recv comm

CPU, size 12000, 10 iterations

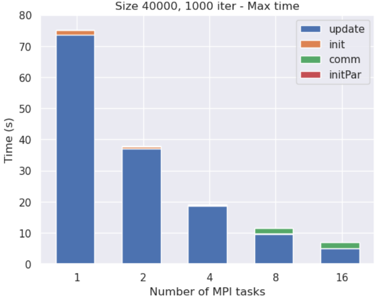


One-Sided comm

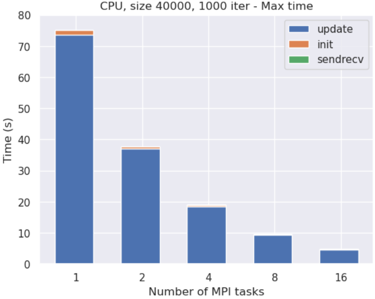


Send-Recv comm

CPU, size 40000, 1000 iterations

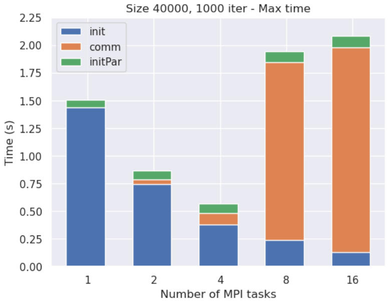


One-Sided comm

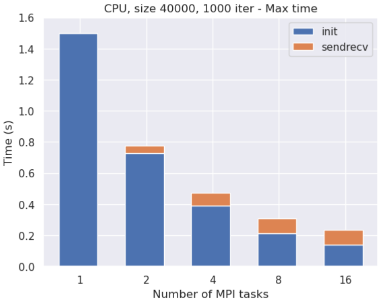


Send-Recv comm

CPU, size 40000, 1000 iterations

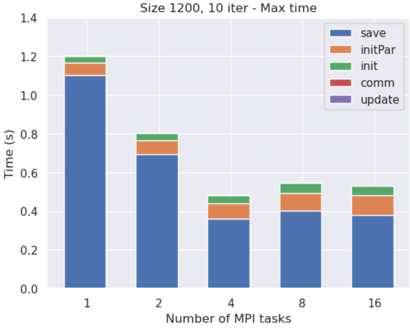


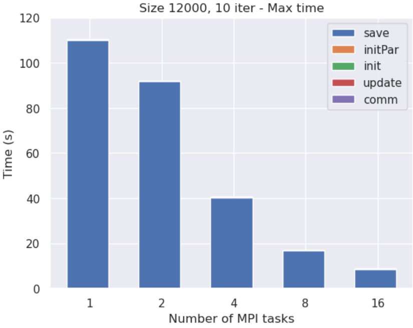
One-Sided comm



Send-Recv comm

Save time


Size 1200, 10 iter - Max time
Time (s)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1
2
4
8
16
Number of MPI tasks
save
initPar
init
comm
update


Size 12000, 10 iter - Max time
Time (s)
120
100
80
60
40
20
0
1
2
4
8
16
Number of MPI tasks
save
initPar
init
update
comm

End