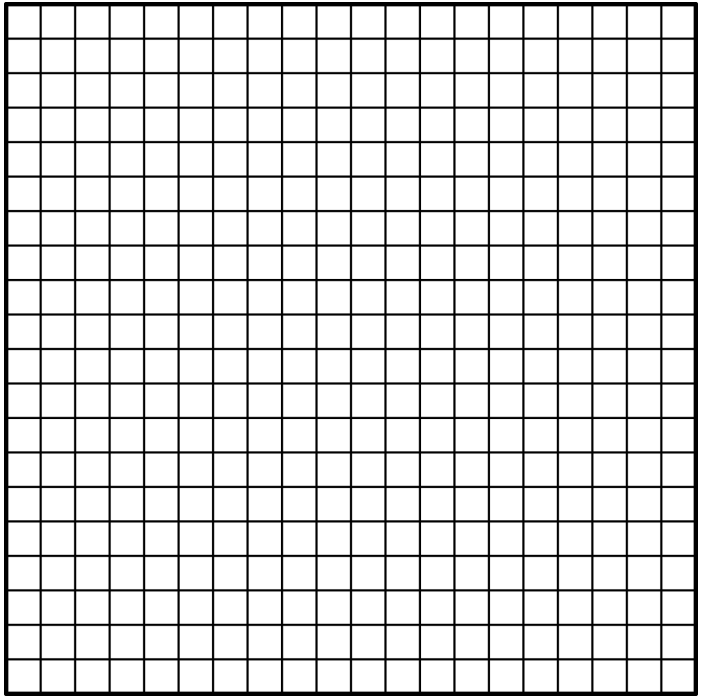
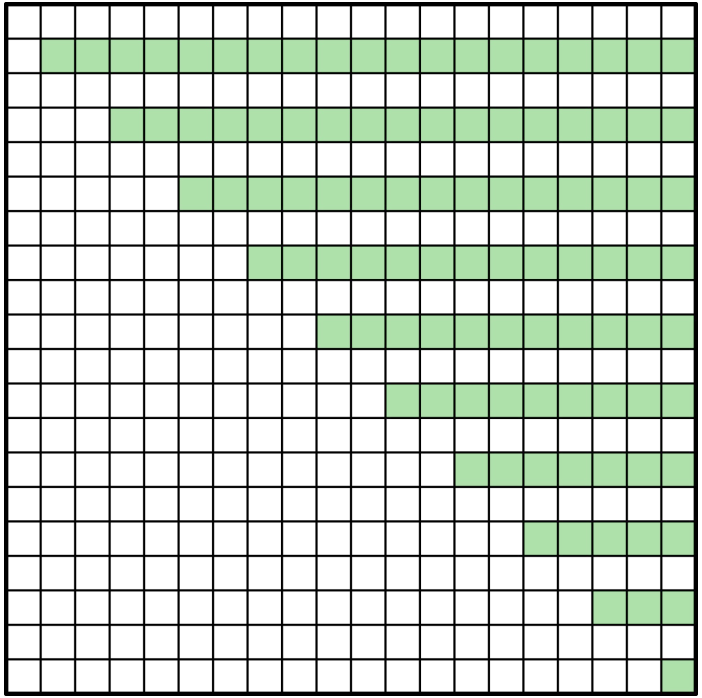
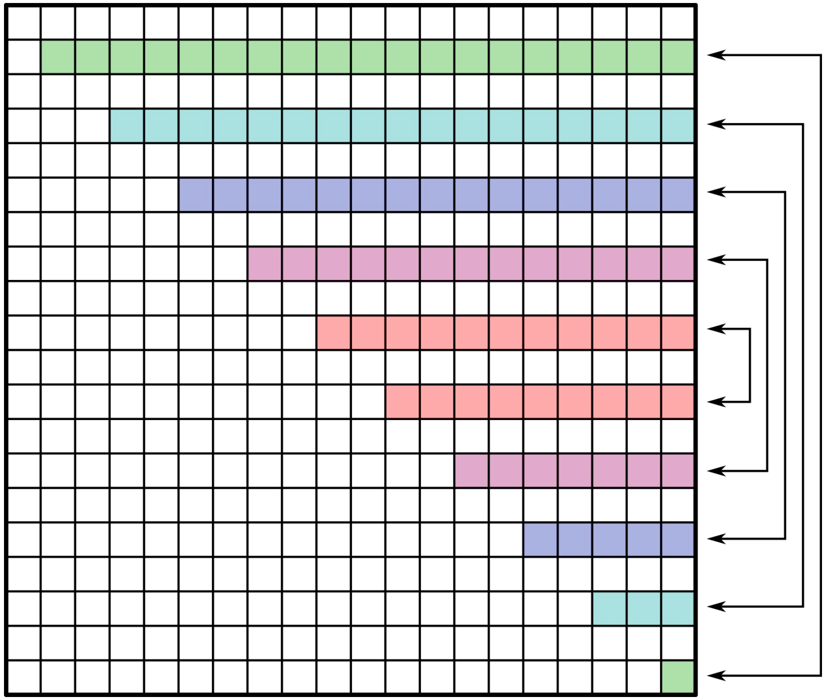
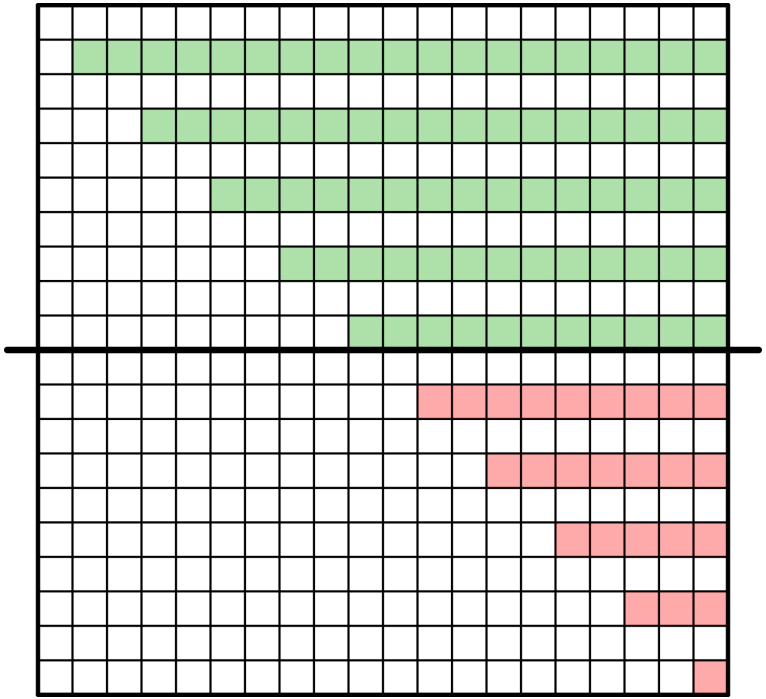
Parallelized matrix-vector multiplication of an upper triangular matrix with only odd indexed rows filled, generated in CSR form

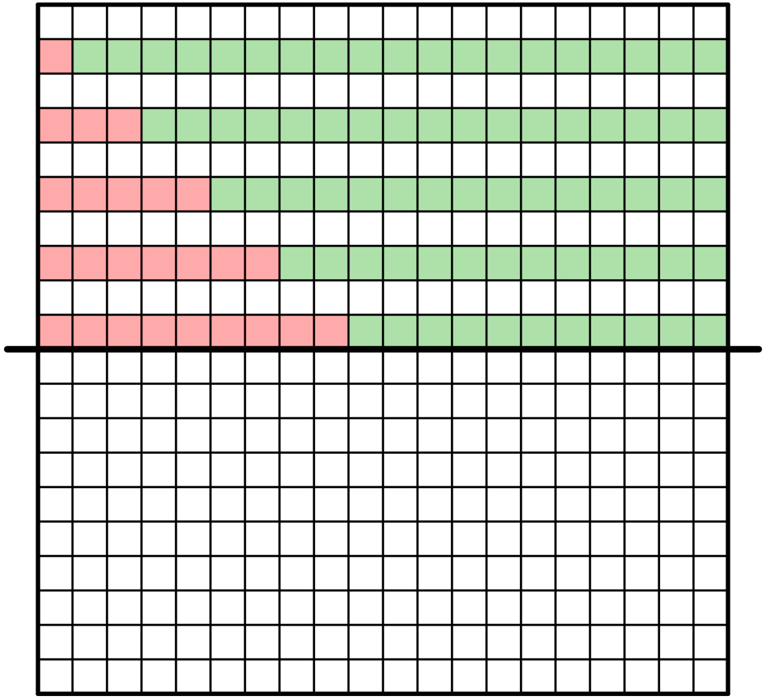
Muhammed Enis Şen

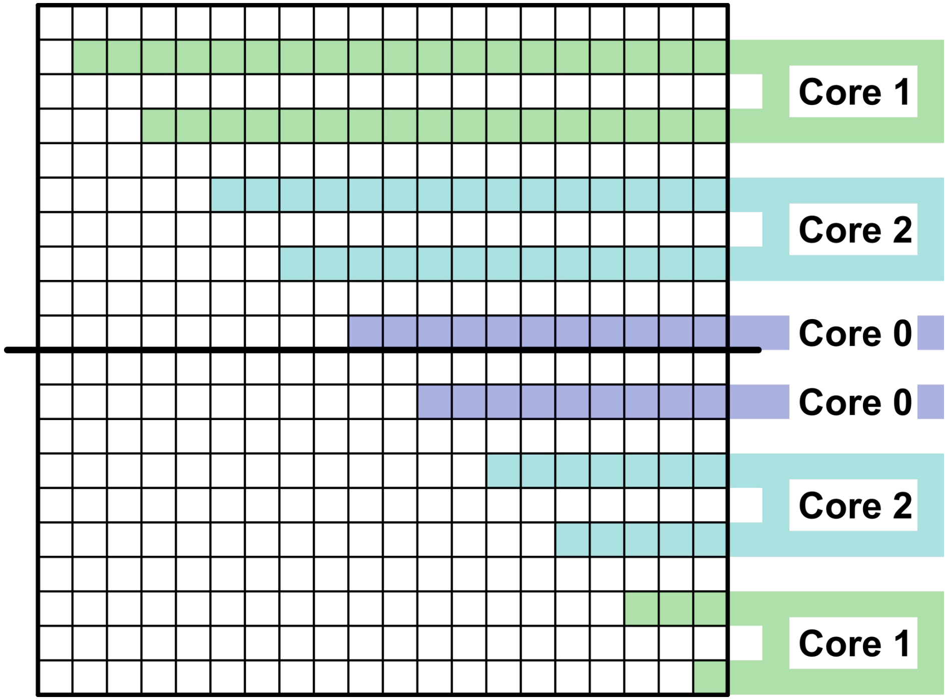












Core 1
Core 2
Core 0
Core 0
Core 2
Core 1

Taking matrix size and calculating the row numbers

```
int mat_n[argc-1];
for(i=1;i<argc;i++)
    mat_n[i-1] = atoi(argv[i]);

for(m=0;m<argc-1;m++){

    mat_size = mat_n[m];
    nnz_count = mat_size * mat_size / 4;

    // Calculate how many rows each core will get
    worker_rows = (mat_size / 4) / (core_count - 1);
    master_rows = (mat_size / 4) - worker_rows * (core_count - 1);
    // A unique variable for each core that takes different values if master or worker
    core_rows = (rank != 0) ? worker_rows : master_rows;
```

Preparing the arrays in master core

```
if (rank == 0){
    // Fill the vector to be multiplied in master core
    fillVecRand(vec, mat_size);

    // Allocate memory and fill values array
    memoryAllocationDouble(&values, nnz_count);
    fillVecRand(values, nnz_count);

    // Allocate memory and fill row_start and col_idx arrays
    memoryAllocationInt(&row_start, mat_size+1);
    memoryAllocationInt(&col_idx, nnz_count);
    fillCSR(row_start, col_idx, nnz_count, mat_size);

    //matVecMult(values, row_start, col_idx, vec, res_vec, mat_size);
    //printArrays(res_vec, row_start, col_idx, nnz_count, mat_size);
}
// Broadcast vec to every core
MPI_Bcast(vec, mat_size, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Master core sending non-zero elements with their col_idx

```
elements_to_be_sent=0; total_sent=0; remaining_to_be_received=0;
for(i=1;i<core_count;i++){

    elements_to_be_sent = mat_size * worker_rows - worker_rows * worker_rows * ( 2*i-1);
    remaining_to_be_received += mat_size * worker_rows - elements_to_be_sent;
    MPI_Send(&elements_to_be_sent, 1, MPI_INT, i, 0, MPI_COMM_WORLD);

    MPI_Send(&values[total_sent], elements_to_be_sent, MPI_DOUBLE, i,  1, MPI_COMM_WORLD);
    MPI_Send(&values[nnz_count-remaining_to_be_received], mat_size * worker_rows - elements_to_be_sent,
            MPI_DOUBLE, i,  2, MPI_COMM_WORLD);

    MPI_Send(&col_idx[total_sent], elements_to_be_sent, MPI_INT, i,  3, MPI_COMM_WORLD);
    MPI_Send(&col_idx[nnz_count-remaining_to_be_received], mat_size * worker_rows - elements_to_be_sent,
            MPI_INT, i,  4, MPI_COMM_WORLD);

    total_sent += elements_to_be_sent;
}
```

Point-to-Point Communication

Worker cores receiving and performing the multiplication

```
}else{
    // Worker cores receive their respective parts of arrays values and col_idx
    MPI_Recv(&elements_to_be_sent, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);

    memoryAllocationDouble(&values_cores, core_rows*mat_size);
    MPI_Recv(&values_cores[0], elements_to_be_sent, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&values_cores[elements_to_be_sent], worker_rows * mat_size - elements_to_be_sent, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD, &status);

    memoryAllocationInt(&col_idx_cores, core_rows*mat_size);
    MPI_Recv(&col_idx_cores[0], elements_to_be_sent, MPI_INT, 0, 3, MPI_COMM_WORLD, &status);
    MPI_Recv(&col_idx_cores[elements_to_be_sent], worker_rows * mat_size - elements_to_be_sent, MPI_INT, 0, 4, MPI_COMM_WORLD, &status);

    // res_vec_cores is calculated
    matVecMult_Calc(values_cores, col_idx_cores, vec, res_vec_cores, mat_size, worker_rows, rank, elements_to_be_sent);

    // Calculated results get sent to master core
    MPI_Send(&res_vec_cores[0], core_rows, MPI_DOUBLE, 0, 5, MPI_COMM_WORLD);
    MPI_Send(&res_vec_cores[core_rows], worker_rows, MPI_DOUBLE, 0, 6, MPI_COMM_WORLD);
}
```

Point-to-Point Communication

Master core receiving and arranging the result vector

```
// Master calculates its own part if any rows are left after the split
elements_to_be_sent = (3*nnz_count/4-total_sent);
if(master_rows != 0){
    matVecMult_Calc(&values[total_sent], &col_idx[total_sent], vec,
                    &res_vec[mat_size/4-master_rows], mat_size, master_rows,
                    rank, elements_to_be_sent);

}

// Master collects res_vec_cores arrays from cores and fills res_vec
for(i=1;i<core_count;i++){
    MPI_Recv(&res_vec[worker_rows*(i-1)], worker_rows, MPI_DOUBLE, i, 5, MPI_COMM_WORLD, &status);
    MPI_Recv(&res_vec[mat_size/2-worker_rows*i], worker_rows, MPI_DOUBLE, i, 6,
             MPI_COMM_WORLD, &status);
}
```

Point-to-Point Communication

Master core sending each core their respective parts

```
// Master core scatters the values and col_idx arrays using the arrays
// sent_cnt1, send_cnt2, send_disp1, send_disp2
MPI_Scatterv(values, send_cnt1, send_disp1, MPI_DOUBLE,
             values_cores, send_cnt1[rank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Scatterv(values, send_cnt2, send_disp2, MPI_DOUBLE,
             &values_cores[send_cnt1[rank]], send_cnt2[rank], MPI_DOUBLE, 0, MPI_COMM_WORLD);

MPI_Scatterv(col_idx, send_cnt1, send_disp1, MPI_INT,
             col_idx_cores, send_cnt1[rank], MPI_INT, 0, MPI_COMM_WORLD);
MPI_Scatterv(col_idx, send_cnt2, send_disp2, MPI_INT,
             &col_idx_cores[send_cnt1[rank]], send_cnt2[rank], MPI_INT, 0, MPI_COMM_WORLD);

// Master calculates its own part if any rows are left after the scatter step
if (core_rows != 0){
    matVecMult_Calc(values_cores, col_idx_cores, vec, res_vec_cores, mat_size, core_rows, rank,
                    elements_to_be_sent);
}
```

Collective Communication

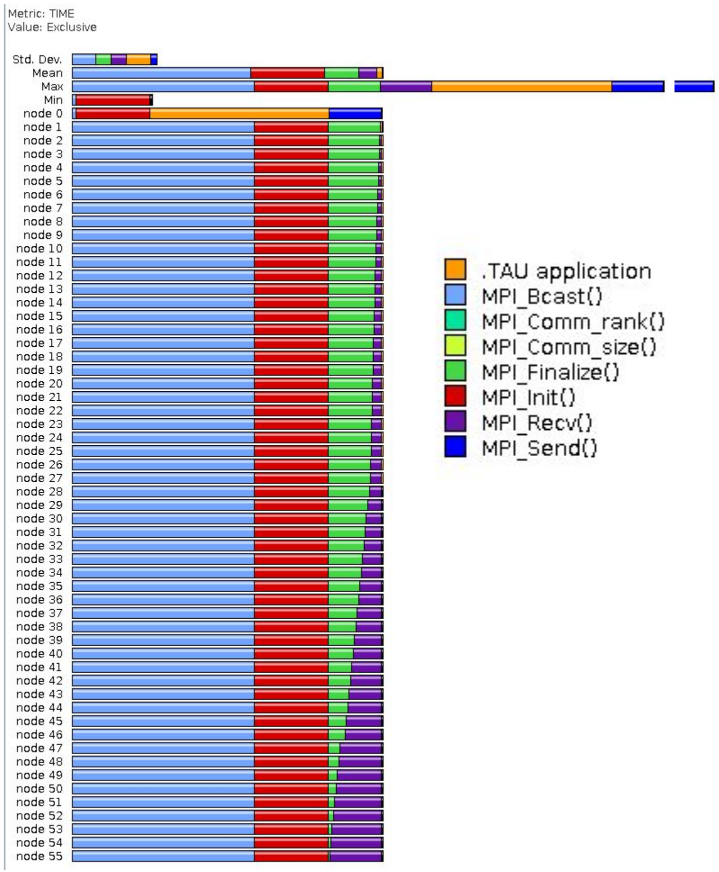
Master core gathering the calculated results into res_vec

```
// Master core gathers the values_cores and col_idx_cores arrays using the arrays
// sent_cnt1, send_disp1, send_disp2
MPI_Gatherv(res_vec_cores, core_rows, MPI_DOUBLE,
            res_vec, send_cnt1, send_disp1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Gatherv(res_vec_cores, core_rows, MPI_DOUBLE,
            res_vec, send_cnt1, send_disp2, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

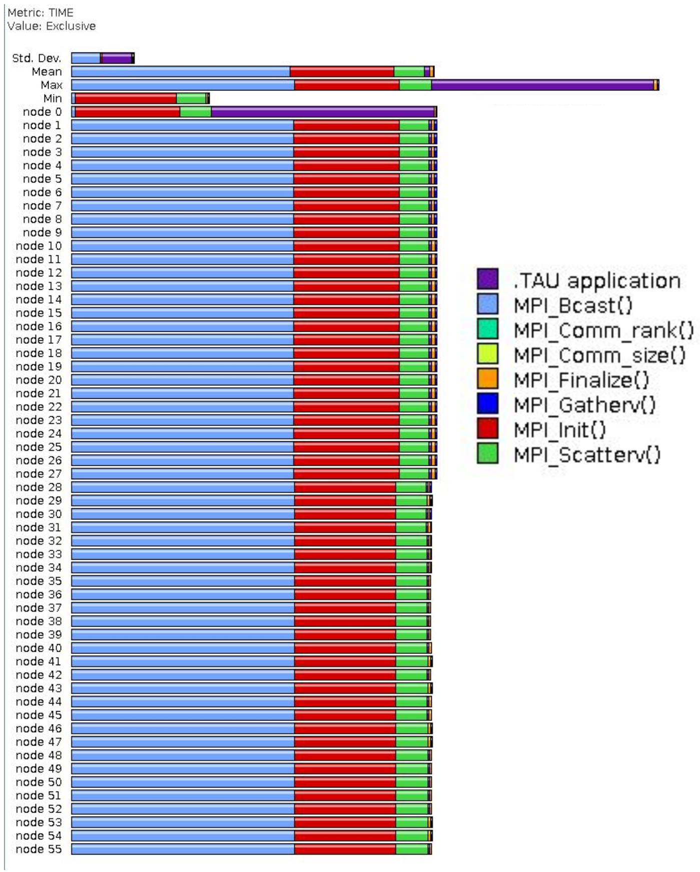
Collective Communication

SPMV multiplication modified to work with my algorithm

```
void matVecMult_Calc(double *values, int *col_idx, double *vec, double *res_vec,
                     int mat_size, int row_number, int rank, int upper_half_elements){

    int i, j, comp=0;
    double sum;

    for(i=1 ; i<=row_number ; i++){
        sum = 0;
        for(j=0 ; j<for_limit(mat_size, rank, row_number, i, 1) ; j++)
            sum += values[comp+j] * vec[col_idx[comp+j]];
        res_vec[i-1] = sum;
        comp += j;
    }

    for(i=1 ; i<=row_number ; i++){
        sum = 0;
        for(j=0 ; j<for_limit(mat_size, rank, row_number, i, 0) ; j++){
            sum += values[comp+j] * vec[col_idx[comp+j]];
        }
        res_vec[i-1+row_number] = sum;
        comp += j;
    }
}
```



Metric: TIME
Value: Exclusive
Std. Dev.
Mean
Max
Min
node 0
node 1
node 2
node 3
node 4
node 5
node 6
node 7
node 8
node 9
node 10
node 11
node 12
node 13
node 14
node 15
node 16
node 17
node 18
node 19
node 20
node 21
node 22
node 23
node 24
node 25
node 26
node 27
node 28
node 29
node 30
node 31
node 32
node 33
node 34
node 35
node 36
node 37
node 38
node 39
node 40
node 41
node 42
node 43
node 44
node 45
node 46
node 47
node 48
node 49
node 50
node 51
node 52
node 53
node 54
node 55
.TAU application
MPI_Bcast()
MPI_Comm_rank()
MPI_Comm_size()
MPI_Finalize()
MPI_Init()
MPI_Recv()
MPI_Send()



Metric: TIME
Value: Exclusive
Std. Dev.
Mean
Max
Min
node 0
node 1
node 2
node 3
node 4
node 5
node 6
node 7
node 8
node 9
node 10
node 11
node 12
node 13
node 14
node 15
node 16
node 17
node 18
node 19
node 20
node 21
node 22
node 23
node 24
node 25
node 26
node 27
node 28
node 29
node 30
node 31
node 32
node 33
node 34
node 35
node 36
node 37
node 38
node 39
node 40
node 41
node 42
node 43
node 44
node 45
node 46
node 47
node 48
node 49
node 50
node 51
node 52
node 53
node 54
node 55
.TAU application
MPI_Bcast()
MPI_Comm_rank()
MPI_Comm_size()
MPI_Finalize()
MPI_Gatherv()
MPI_Init()
MPI_Scatterv()

Matrix sizes