5

Distributed computing

Basic StarPU/MPI Example

Straightforward port of existing MPI code

```
1 for (loop = 0; loop < NLOOPS; loop++) {
2     if (!(loop == 0 && rank == 0))
3         MPI_Recv(&data, prev_rank, ...);
4
5     inc(&data);
6
7     if (!(loop == NLOOPS-1 && rank == size-1))
8         MPI_Send(&data, next_rank, ...);
9 }
```

Basic StarPU/MPI Example

Straightforward port of existing MPI code

- Asynchronous requests

```
1  for (loop = 0; loop < NLOOPS; loop++) {
2      if (!(loop == 0 && rank == 0))
3          starpu_mpi_irecv_submit(data_handle, prev_rank, ...);
4
5      starpu_mpi_task_insert(&inc_cl, STARPU_RW, data_handle, 0);
6
7      if (!(loop == NLOOPS-1 && rank == size-1))
8          starpu_mpi_isend_submit(data_handle, next_rank, ...);
9  }
10 starpu_task_wait_for_all();
```

Basic StarPU/MPI Example

Straightforward port of existing MPI code

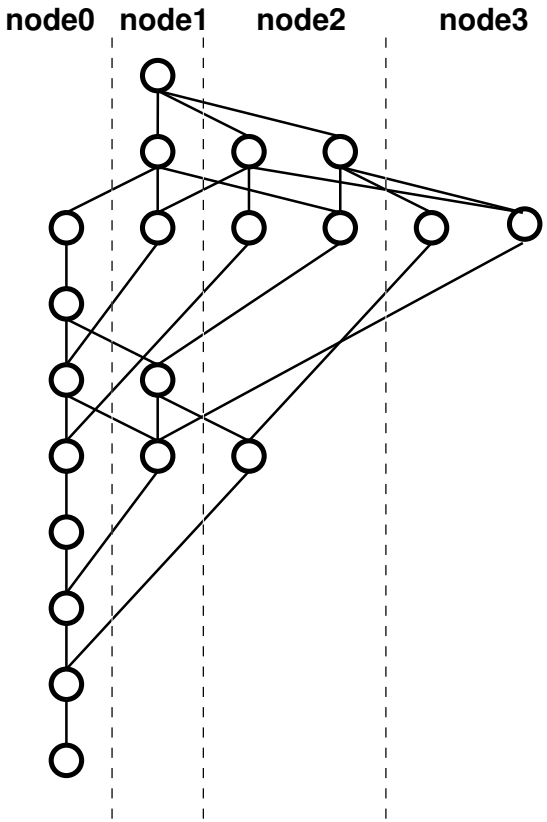
- Asynchronous requests
- Requests are DAG elements
  - Task may depend on **Irecv** completion
  - **Isend** may depend on task completion

```
1  for (loop = 0; loop < NLOOPS; loop++) {
2      if (!(loop == 0 && rank == 0))
3          starpu_mpi_irecv_submit(data_handle, prev_rank, ...);
4
5      starpu_mpi_task_insert(&inc_cl, STARPU_RW, data_handle, 0);
6
7      if (!(loop == NLOOPS-1 && rank == size-1))
8          starpu_mpi_isend_submit(data_handle, next_rank, ...);
9  }
10 starpu_task_wait_for_all();
```

Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow



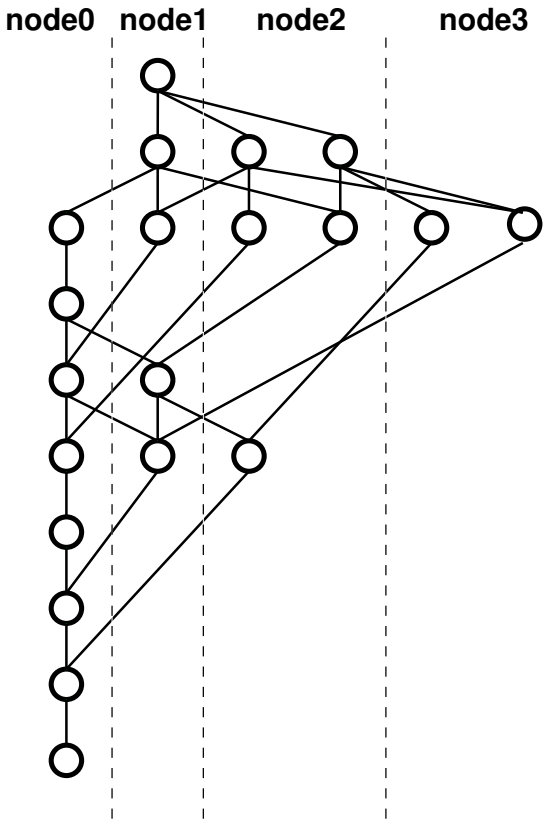
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Data↔Node Mapping

- Provided by the application
- Can be altered dynamically



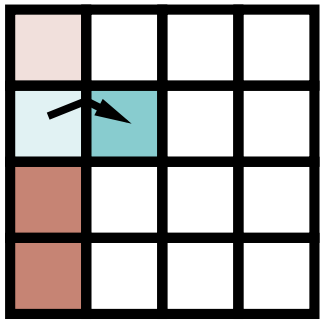
Distributed Support

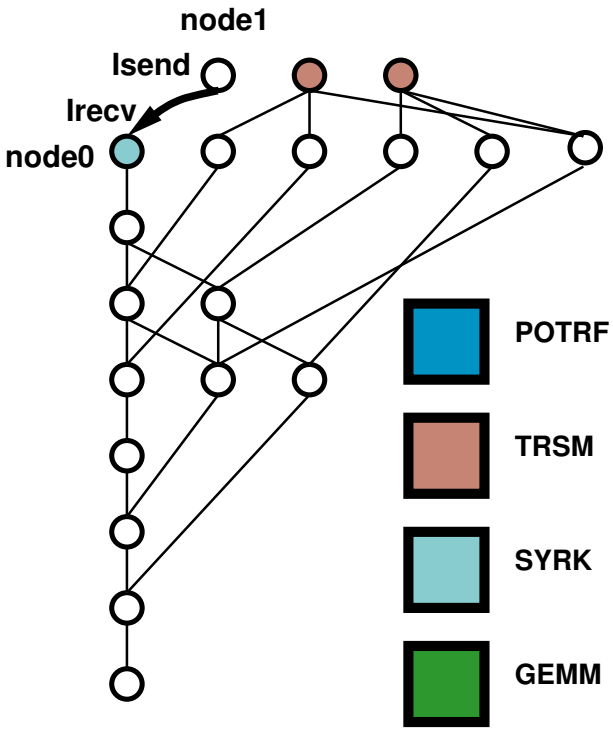
Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Inter-node dependence management

- Inferred from the task graph edges
- Automatic **Isend** and **Irecv** calls





Distributed Support

Sequential Task Flow Paradigm on Clusters

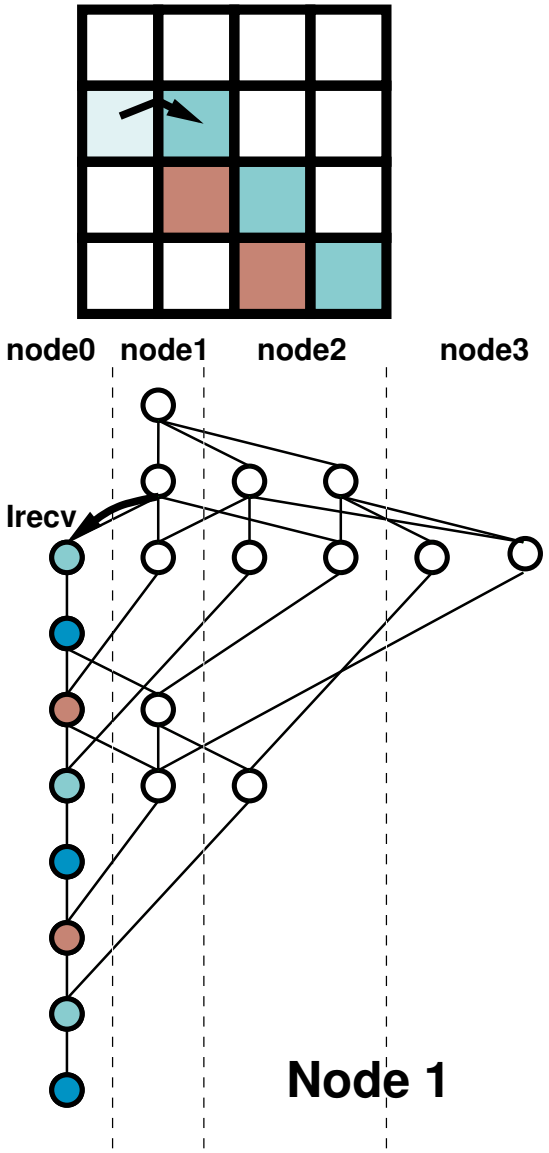
Each node unrolls the sequential task flow

Task↔Node Mapping

- Inferred from data location:
  - *Tasks move to data they modify*
- No global scheduling
- No synchronizations

Optimization

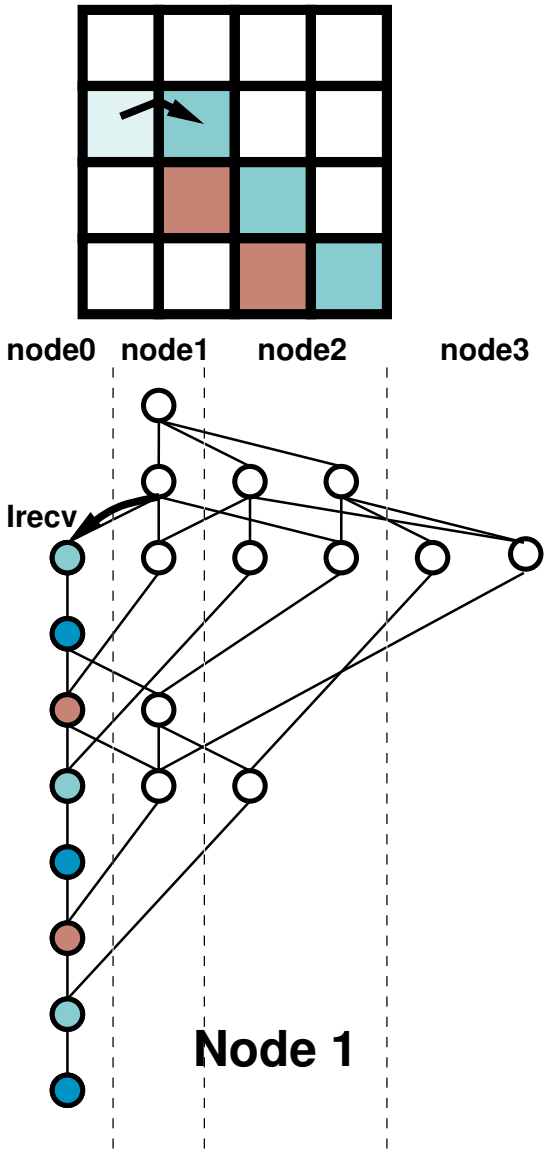
- Local DAG pruning

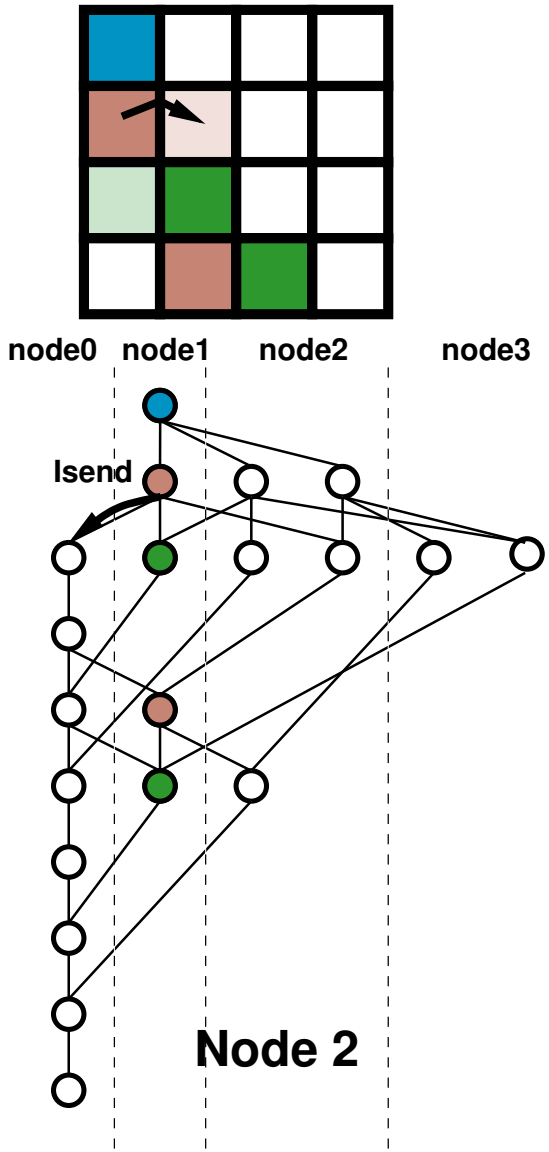


Communication

Nodes infer required transfers

- Task dependencies

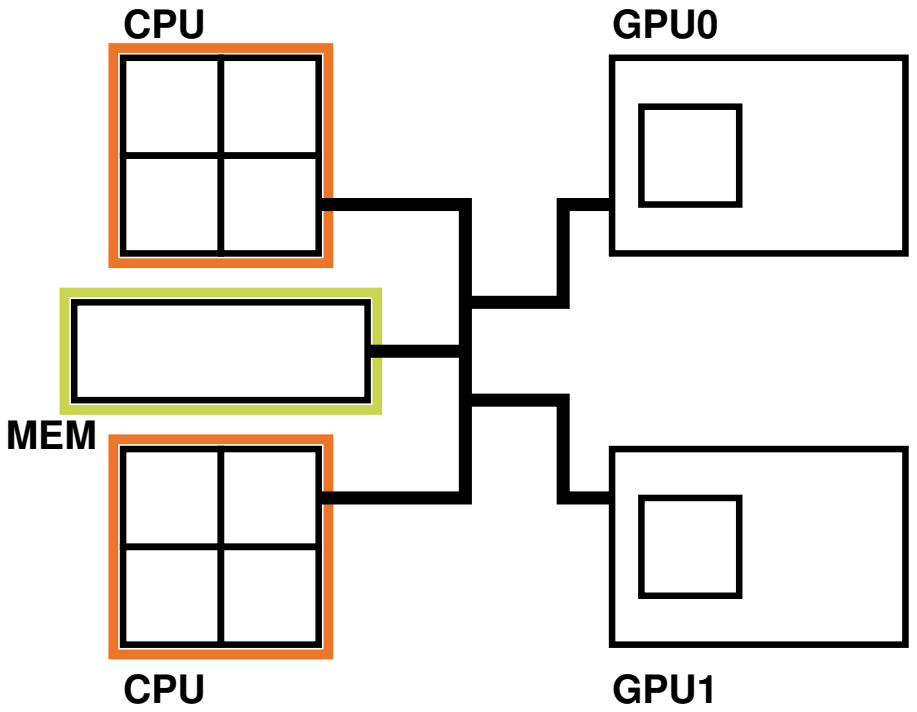




Communication

Nodes infer required transfers

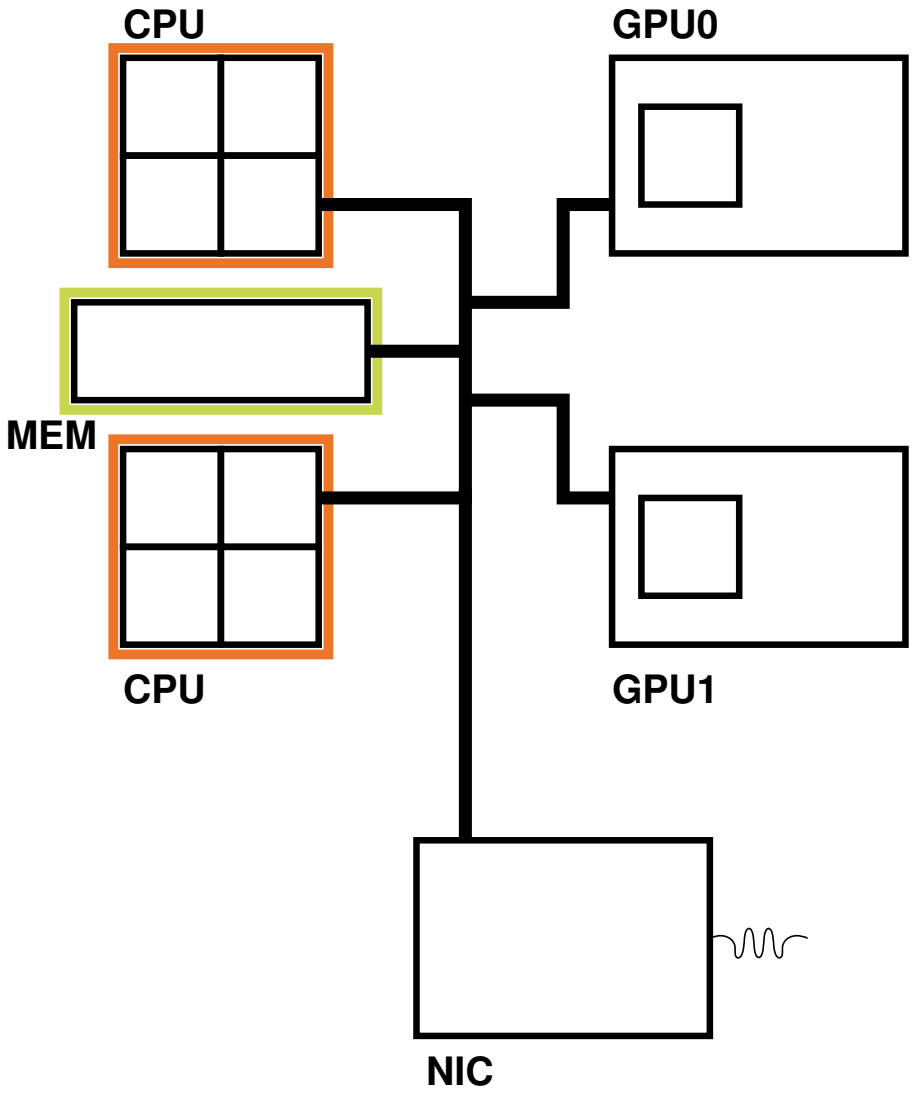
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

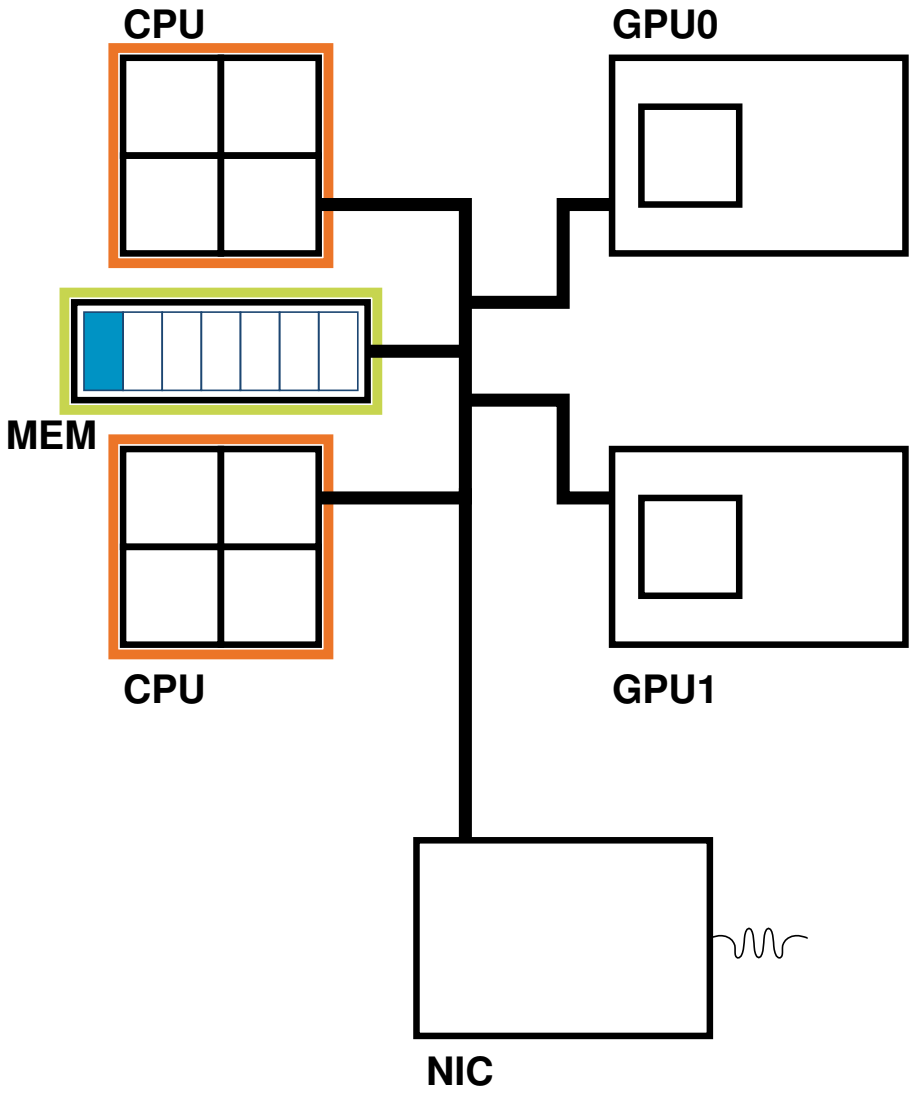
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

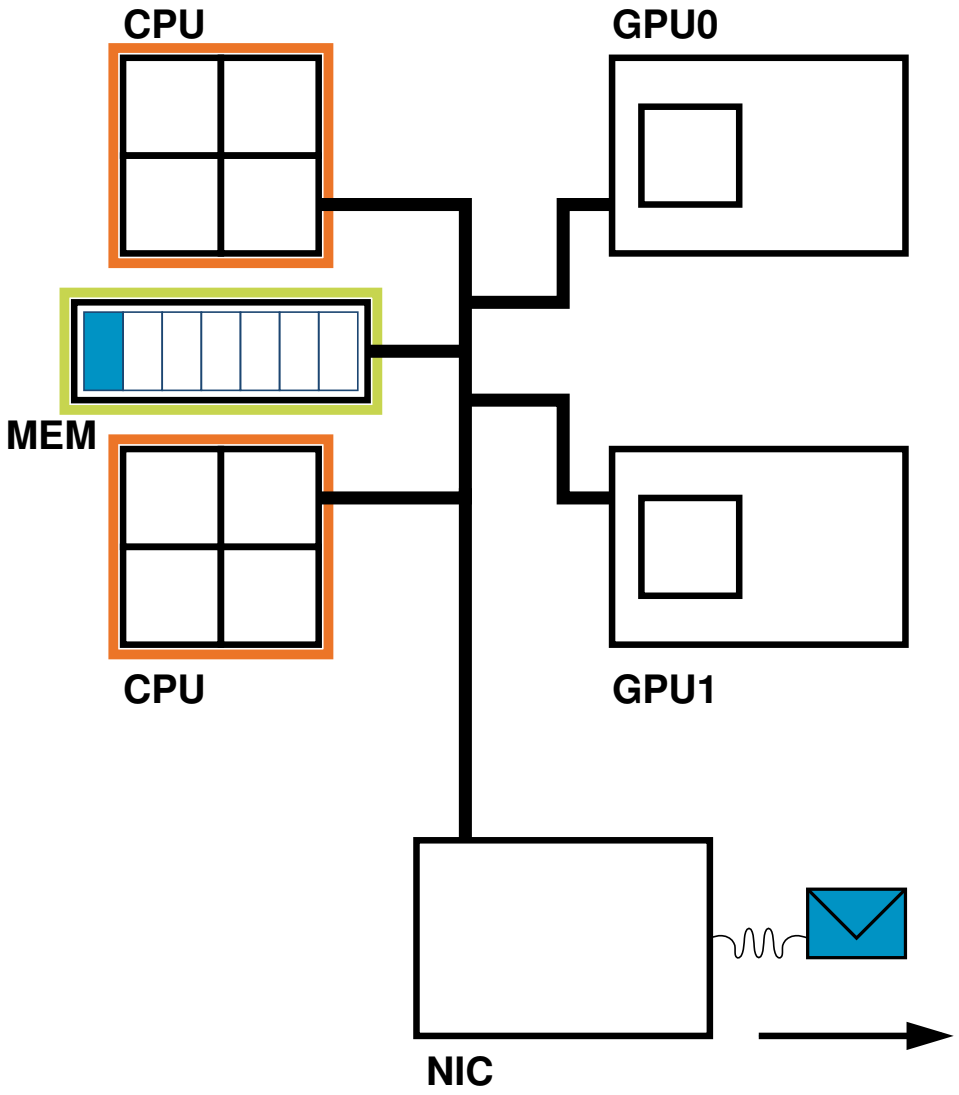
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

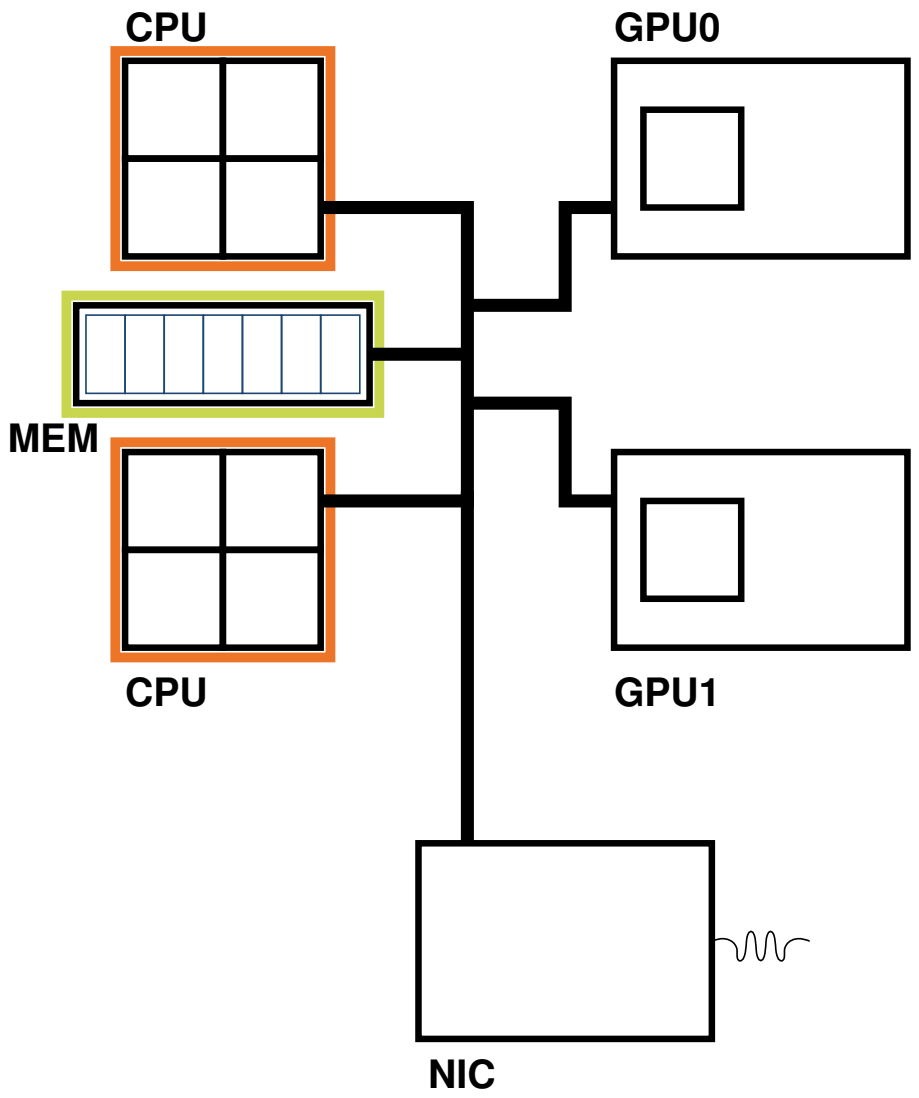
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

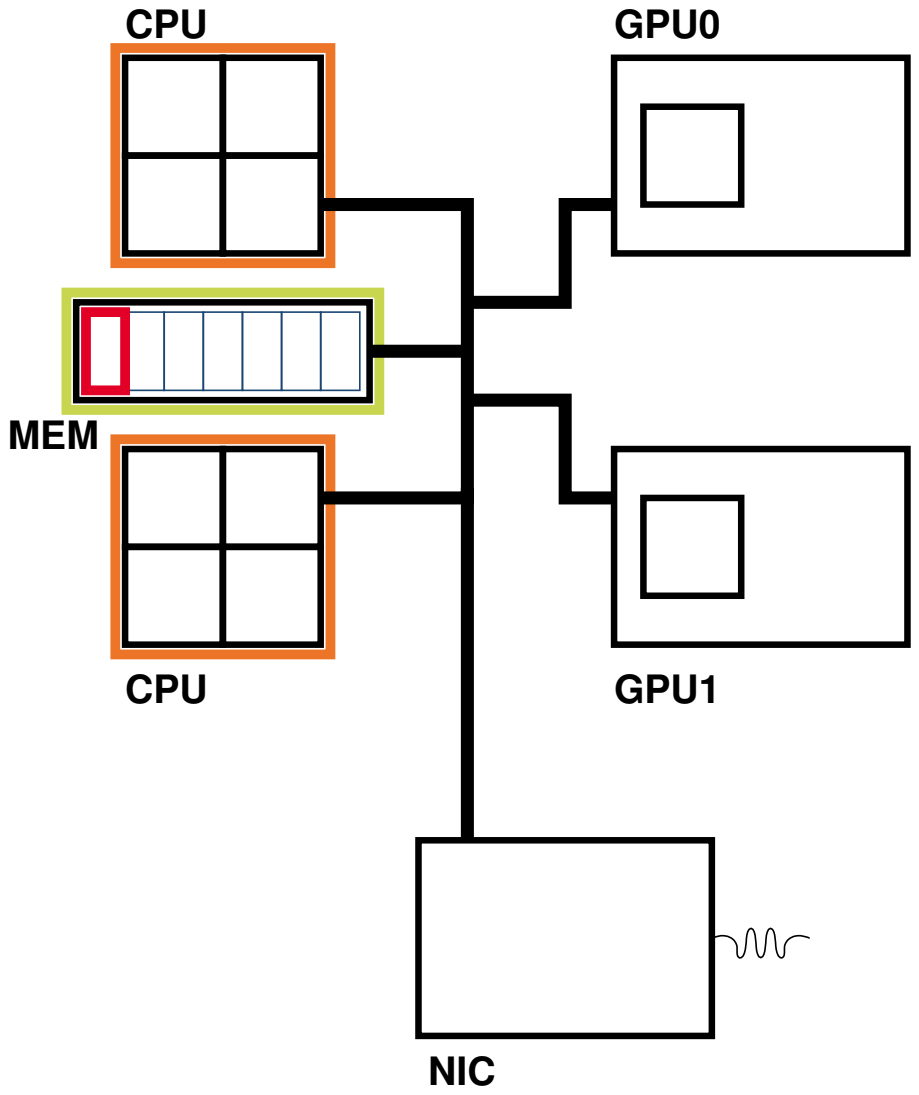
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

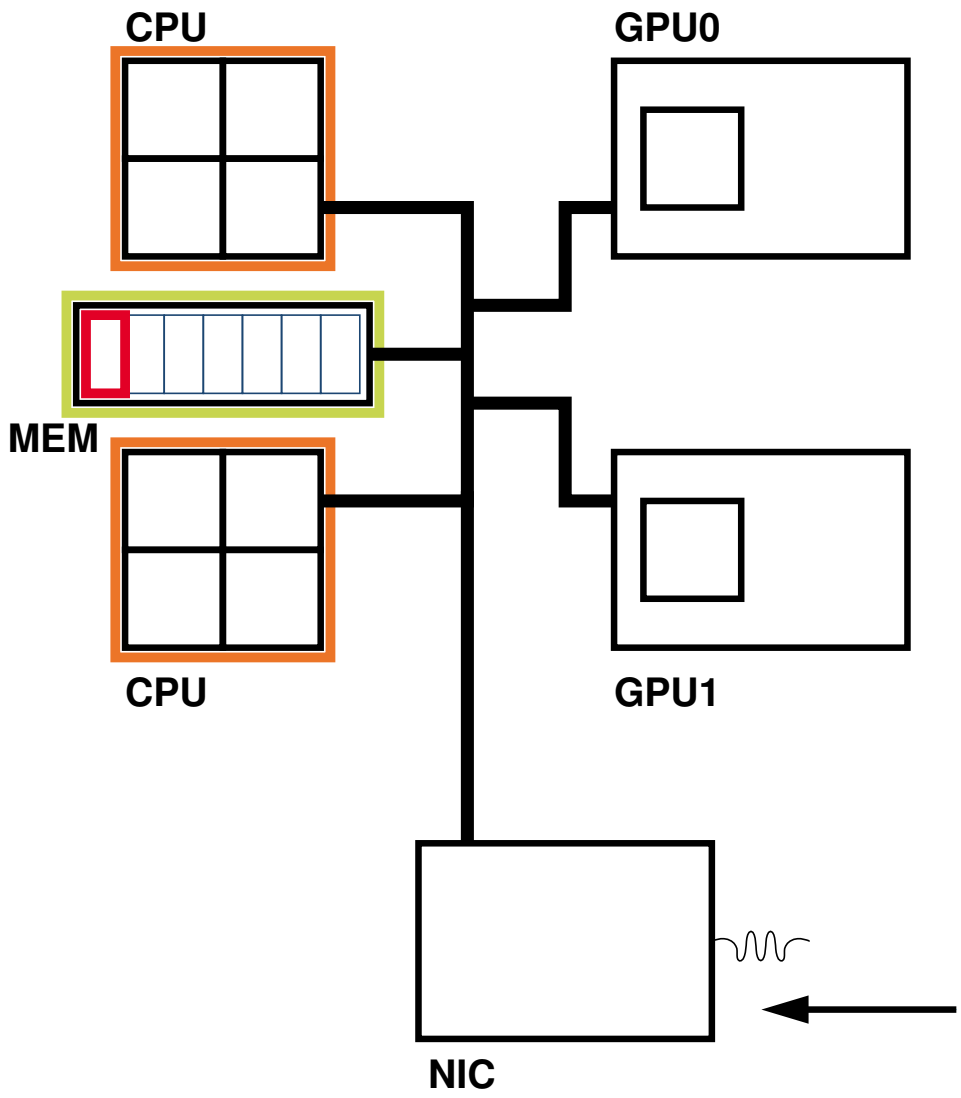
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

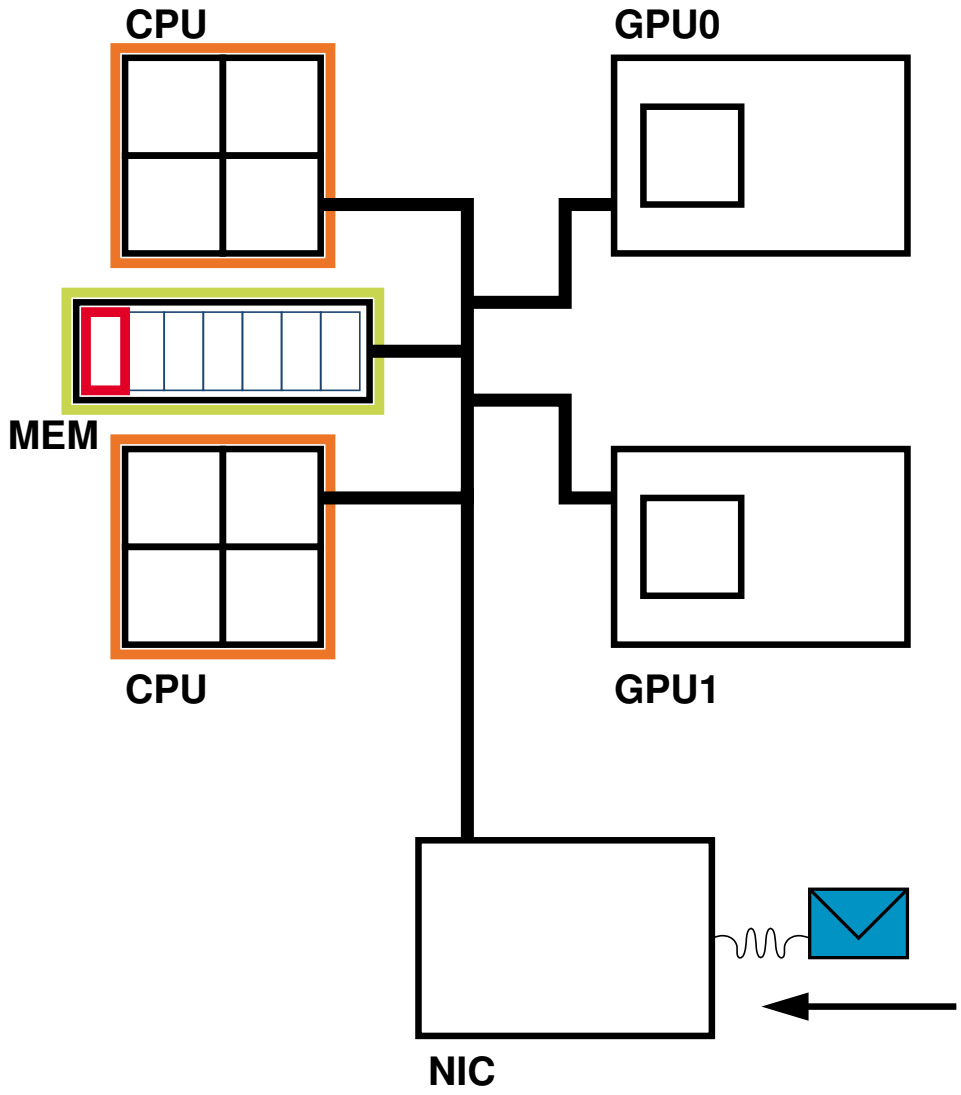
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

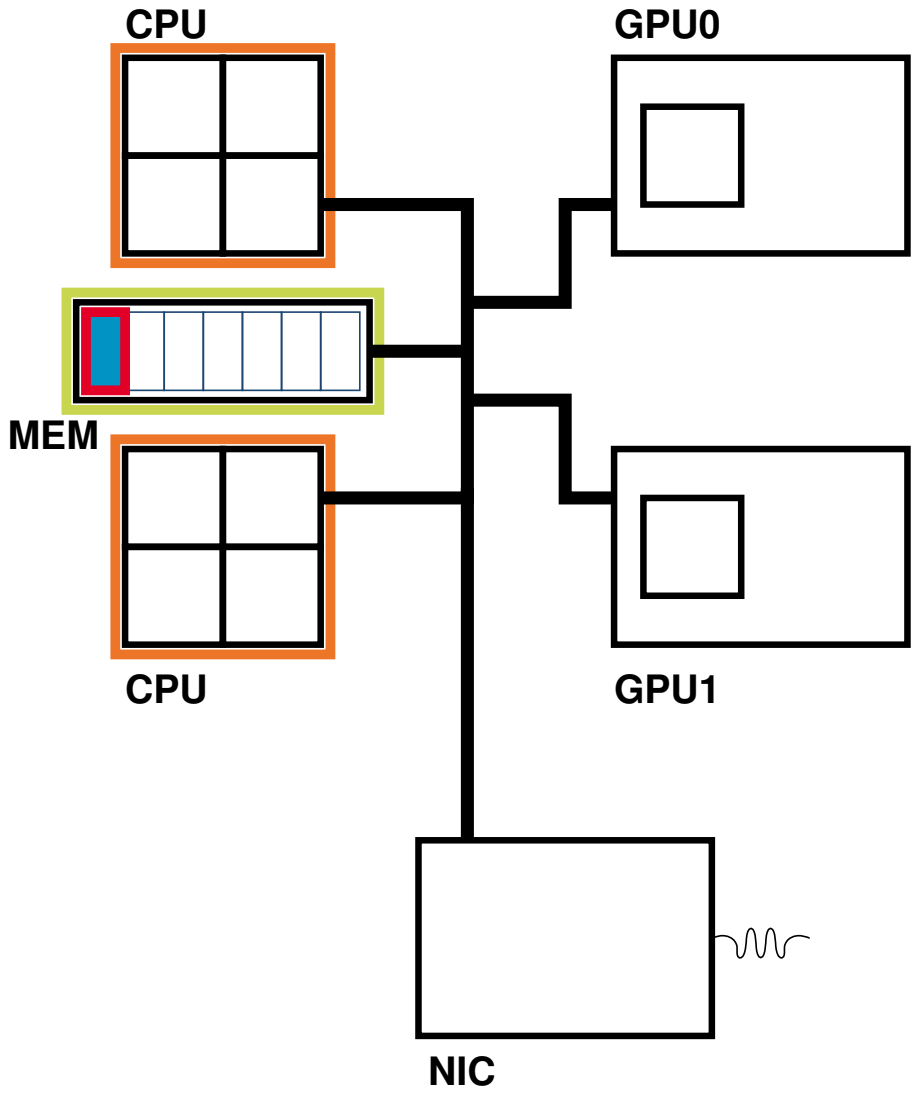
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Same Paradigm for Clusters (vs Single Node)

same code

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

Same Paradigm for Clusters (vs Single Node)

Almost same code

- MPI communicator

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD);
  }
}
task_wait_for_all();
```

Same Paradigm for Clusters (vs Single Node)

Almost same code

- MPI communicator
- Mapping function

```
int getnode(int i, int j) { return((i%p)*q + j%q); }


for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD, getnode(j,j));
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD, getnode(i,j));
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD, getnode(i,i));
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD, getnode(i,k));
  }
}
task_wait_for_all();
```

Same Paradigm for Clusters (vs Single Node)

Almost same code

- MPI communicator
- Mapping function

```
int getnode(int i, int j) { return((i%p)*q + j%q); }
set_rank(A, getnode);

for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD);
  }
}
task_wait_for_all();
```

Distributed STF Programming

Specific init/shutdown:

```
1       starpu_mpi_init_conf(&argc, &argv, 1, MPI_COMM_WORLD, NULL);
2       starpu_mpi_comm_rank(MPI_COMM_WORLD, &rank);
3       starpu_mpi_comm_size(MPI_COMM_WORLD, &size);
4
5       starpu_mpi_shutdown();
```

Distributed STF Programming

(Optional) distribution function:

```
1 int my_distrib(int x, int y, int nb_nodes) {
2     /* example: block distrib */
3     return ((int)(x / sqrt(nb_nodes)
4         + (y / sqrt(nb_nodes)) * sqrt(nb_nodes))) % nb_nodes;
5 }
```

Distributed STF Programming

Data registration:

```
1  unsigned matrix[X][Y];
2  starpu_data_handle_t data_handles[X][Y];
3  for(x = 0; x < X; x++) {
4      for (y = 0; y < Y; y++) {
5          int mpi_rank = my_distrib(x, y, size);
6
7          /* StarPU registration step */
8          if (mpi_rank == my_rank)
9              /* Owning data */
10             starpu_variable_data_register(&data_handles[x][y],
11                 STARPU_MAIN_RAM, (uintptr_t)&(matrix[x][y]), sizeof(unsigned));
12         else if (my_rank == my_distrib(x+1, y, size)
13                 my_rank == my_distrib(x-1, y, size)
14                 my_rank == my_distrib(x, y+1, size)
15                 my_rank == my_distrib(x, y-1, size))
16             /* Don't own this index, but will need it for computations */
17             starpu_variable_data_register(&data_handles[x][y],
18                                           -1, (uintptr_t)NULL, sizeof(unsigned));
19         else
20             /* Do not allocate anything for this */
21             data_handles[x][y] = NULL;
22
23         /* StarPU-MPI registration step */
24         if (data_handles[x][y]) {
25             starpu_mpi_data_register(data_handles[x][y], x*X+y, mpi_rank);
26         }
27     }
28 }
```

Distributed STF Programming

Task submission:

```
1  for(loop=0; loop<niter; loop++)
2      for (x = 1; x < X-1; x++)
3          for (y = 1; y < Y-1; y++)
4              starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
5                                     STARPU_RW, data_handles[x][y],
6                                     STARPU_R, data_handles[x-1][y],
7                                     STARPU_R, data_handles[x+1][y],
8                                     STARPU_R, data_handles[x][y-1],
9                                     STARPU_R, data_handles[x][y+1],
10                                    0);
11 starpu_task_wait_for_all();
```

Distributed STF Programming

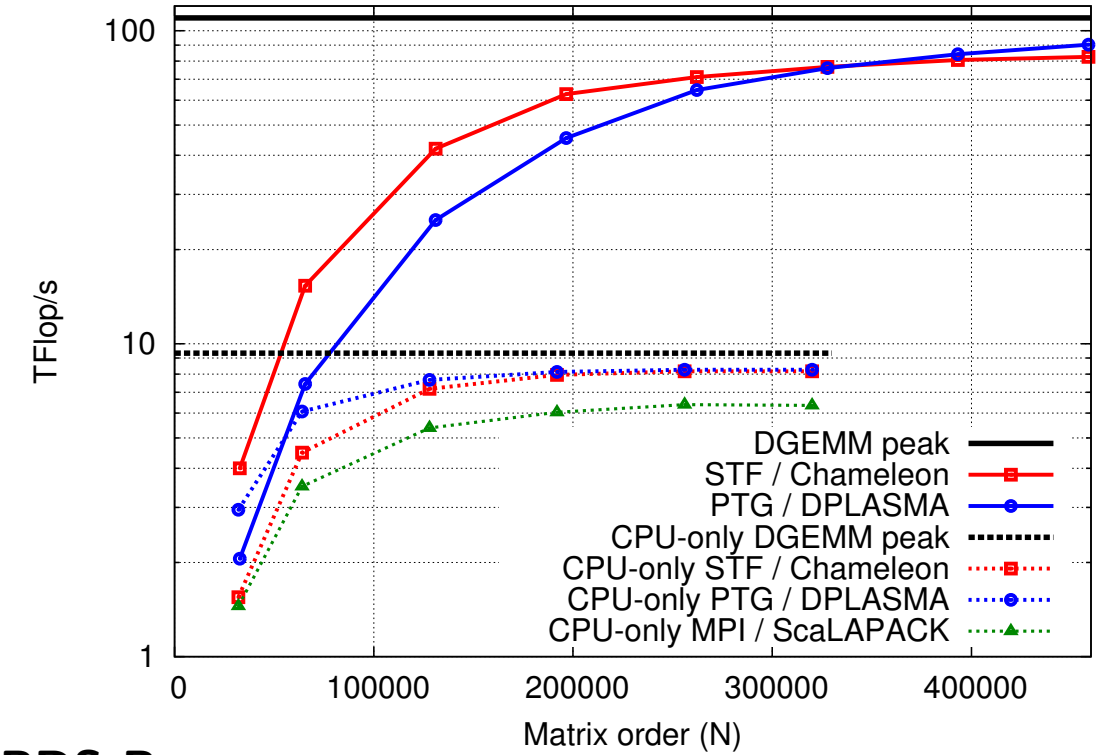
Task submission with pruning:

```
1  for(loop=0; loop<niter; loop++)
2      for (x = 1; x < X-1; x++)
3          for (y = 1; y < Y-1; y++)
4              if (my_distrib(x,y,size) == my_rank
5                  my_distrib(x-1,y,size) == my_rank
6                  my_distrib(x+1,y,size) == my_rank
7                  my_distrib(x,y-1,size) == my_rank
8                  my_distrib(x,y+1,size) == my_rank)
9                      starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
10                                            STARPU_RW, data_handles[x][y],
11                                            STARPU_R, data_handles[x-1][y],
12                                            STARPU_R, data_handles[x+1][y],
13                                            STARPU_R, data_handles[x][y-1],
14                                            STARPU_R, data_handles[x][y+1],
15                                            0);
16 starpu_task_wait_for_all();
```

Distributed Scalability Study Results

Chameleon linear algebra library (Inria Team HiePACS)

- Heterogeneous cluster: 1152 CPU cores+288 GPUs



IEEE TPDS Paper:
DOI: 10.1109/TPDS.2017.2766064 — https://hal.inria.fr/hal-01618526