Overview of Distributed Linear Algebra on Hybrid Nodes over the StarPU Runtime

Emmanuel Agullo, Olivier Aumage, Mathieu Faverge, Nathalie Furmento, Florent Pruvost, Marc Sergent, Samuel Thibault

Overview of Distributed Linear Algebra on Hybrid Nodes over the StarPU Runtime

Emmanuel AGULLO, Olivier AUMAGE, Mathieu FAVERGE, Nathalie FURMENTO, Florent PRUVOST, Samuel THIBAULT, Marc SERGENT

MORSE Associated Team

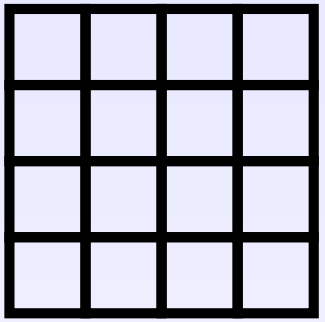
Outline

Introduction

- Runtime systems usually abstract a single node
  - Plasma/Quark, Flame/SuperMatrix, Morse/StarPU, Dplasma/Parsec ...
- How should nodes communicate?
  - Using explicit MPI user calls
  - Using a specific paradigm: Dplasma
- Can we keep the same paradigm and almost the same code, and leave runtime handle data transfers?
  - Example: **Cholesky** factorization (DPOTRF)

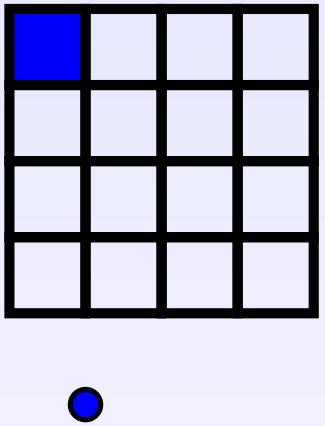
Sequential task-based Cholesky on a single node

```
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();
```



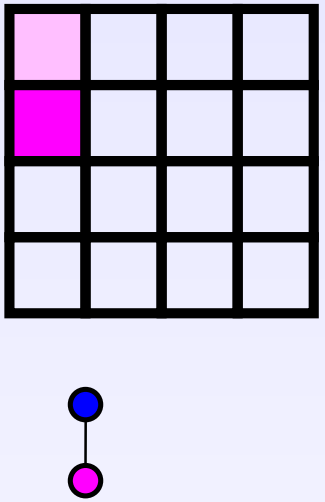
Sequential task-based Cholesky on a single node

```
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();
```



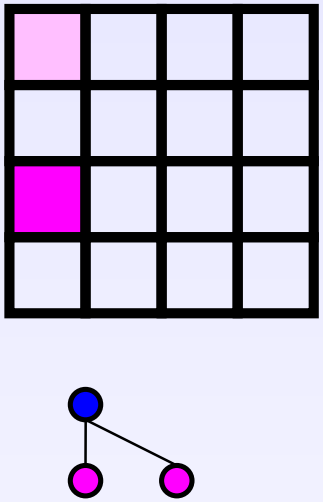
Sequential task-based Cholesky on a single node

```
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();
```



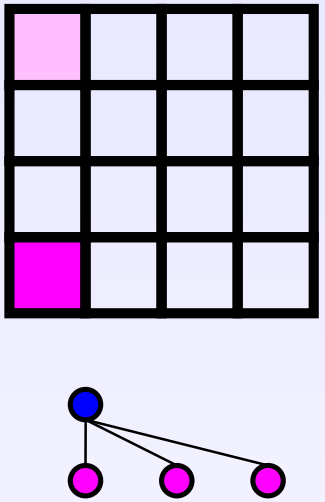
Sequential task-based Cholesky on a single node

```
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();
```



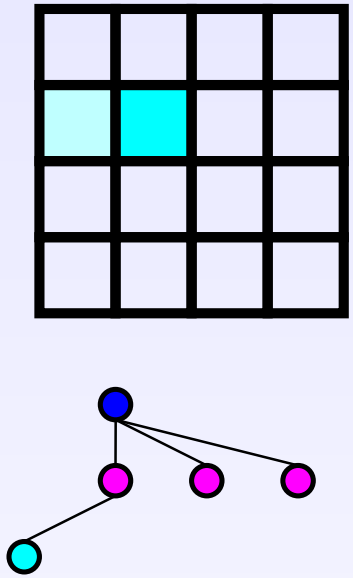
Sequential task-based Cholesky on a single node

```
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();
```



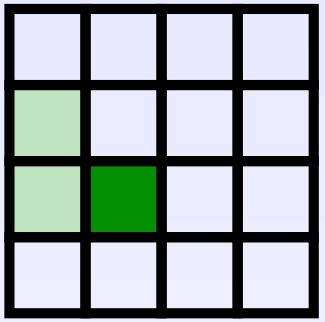
Sequential task-based Cholesky on a single node

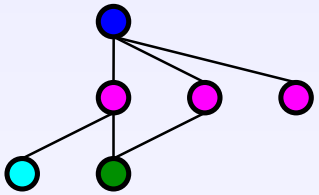
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Sequential task-based Cholesky on a single node

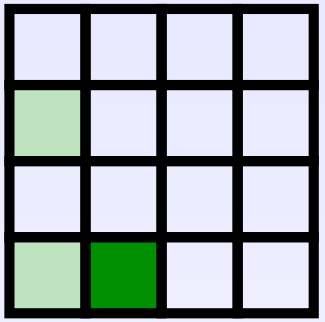
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

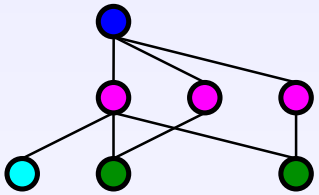




Sequential task-based Cholesky on a single node

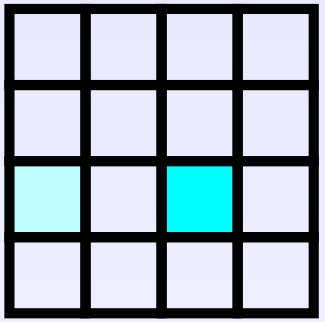
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

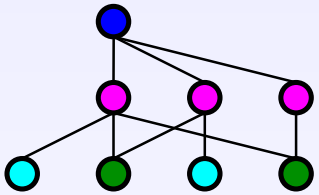




Sequential task-based Cholesky on a single node

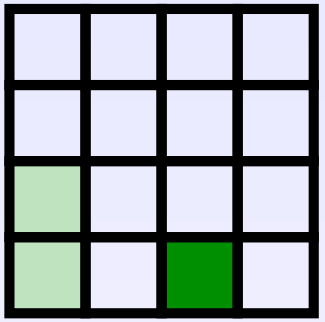
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

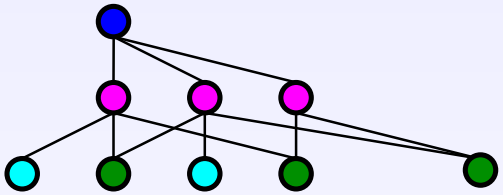




Sequential task-based Cholesky on a single node

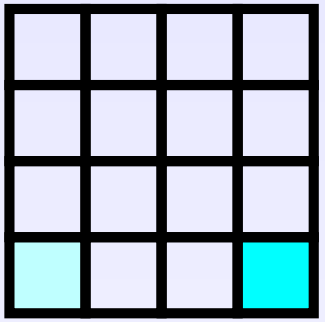
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

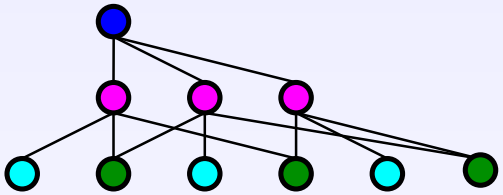




Sequential task-based Cholesky on a single node

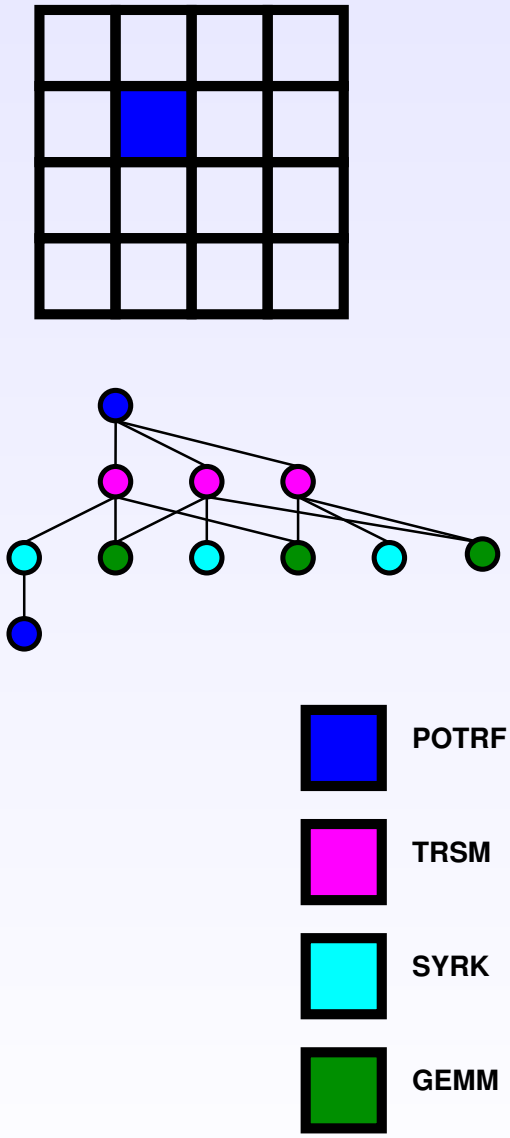
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```





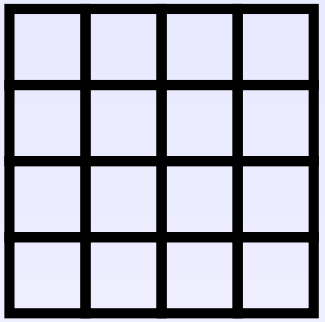
Sequential task-based Cholesky on a single node

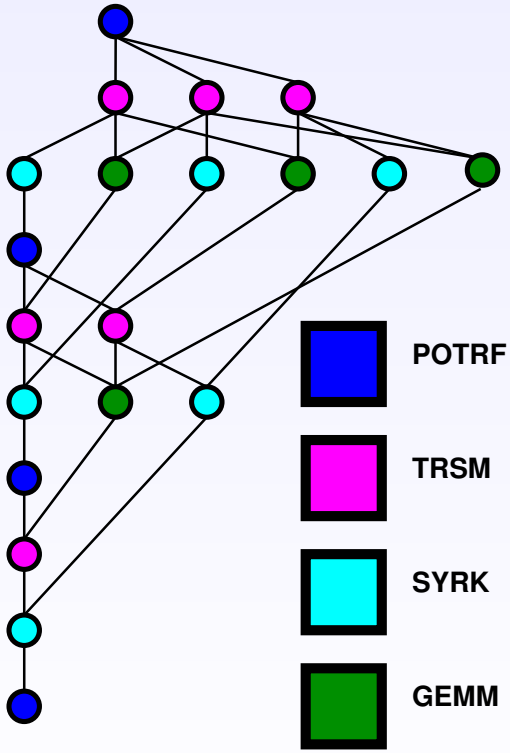
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Sequential task-based Cholesky on a single node

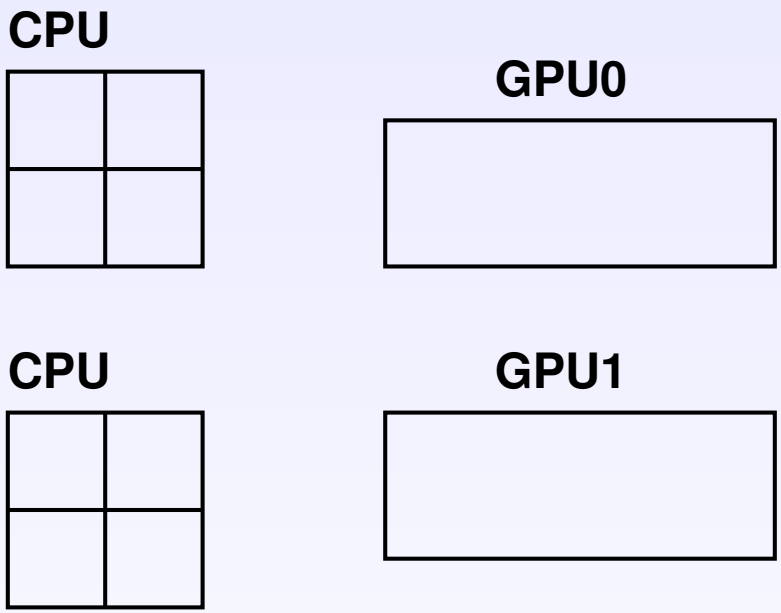
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

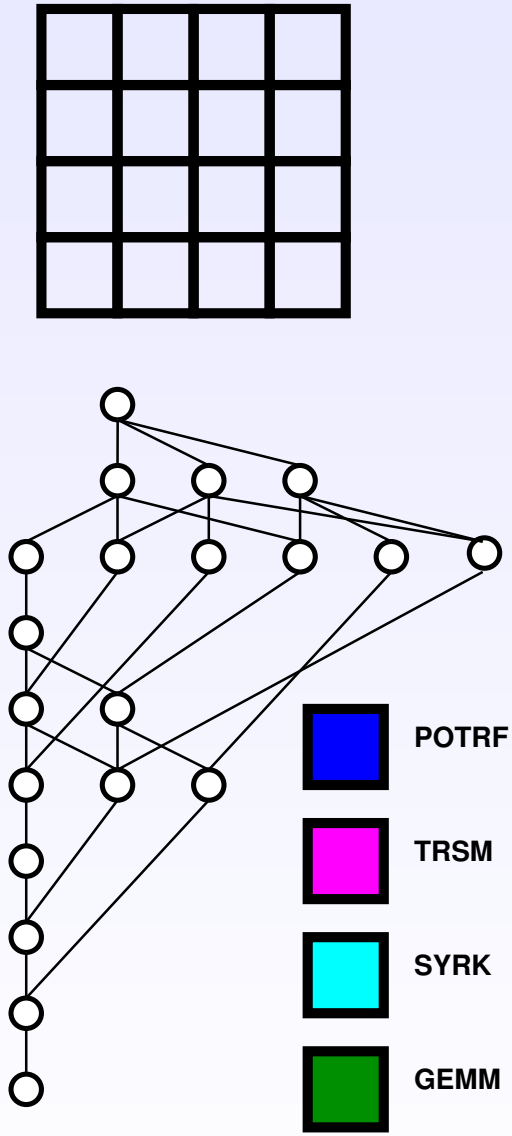




Runtime parallel execution on a heterogeneous node

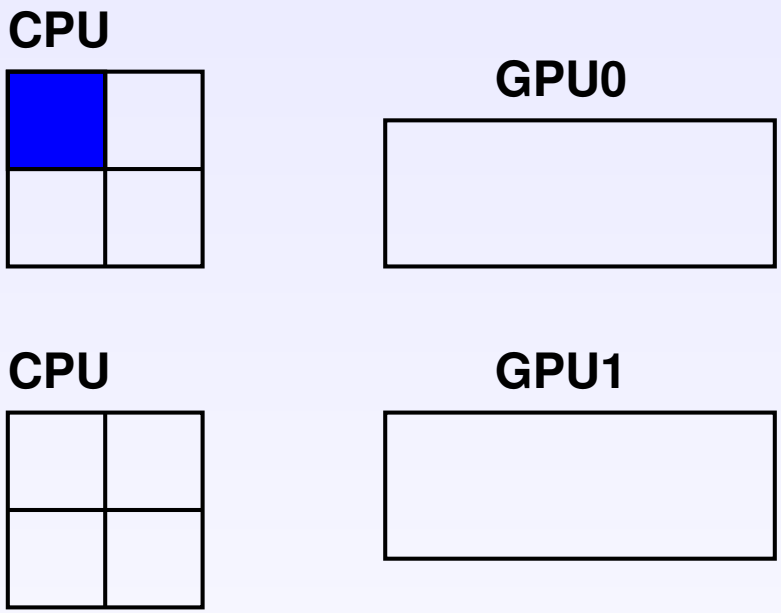
```
task_wait_for_all();
```

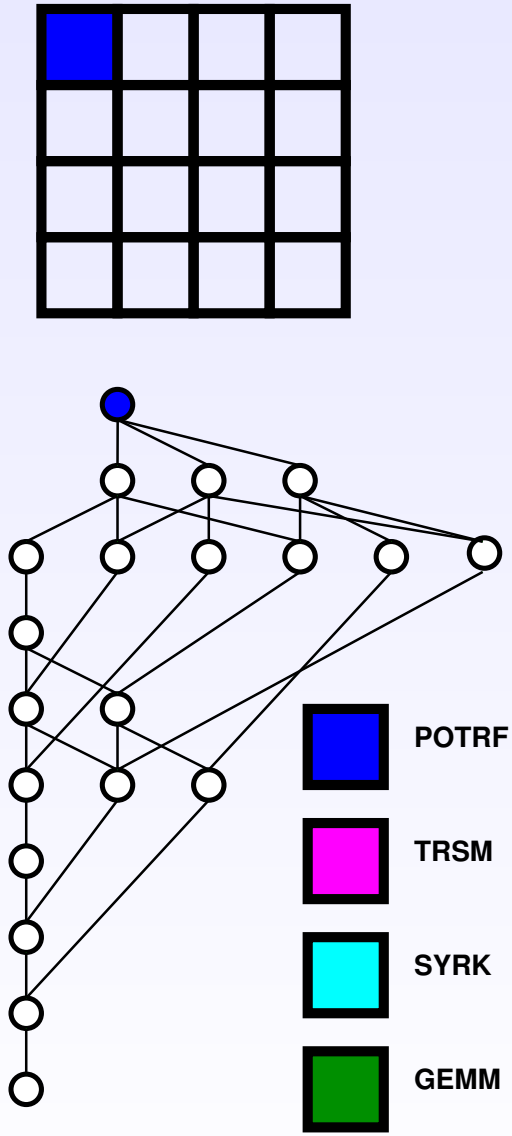




Runtime parallel execution on a heterogeneous node

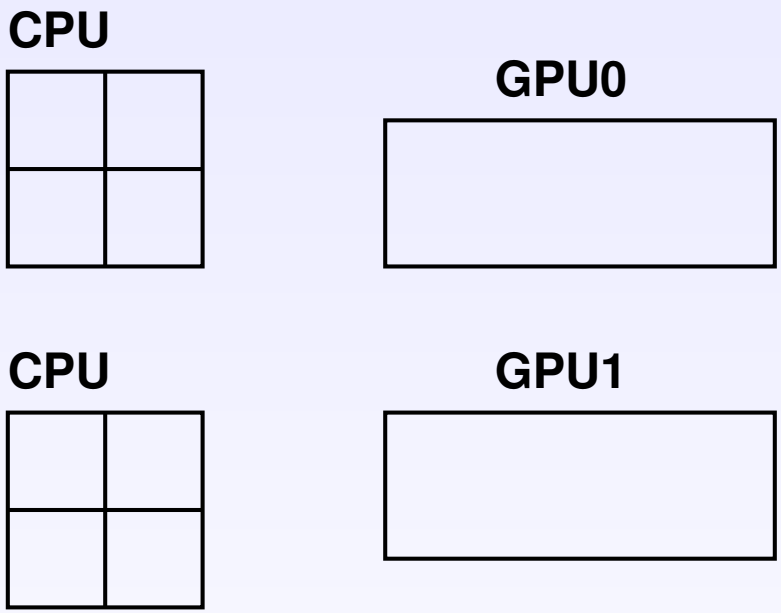
```
task_wait_for_all();
```

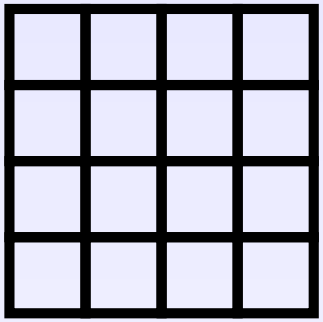


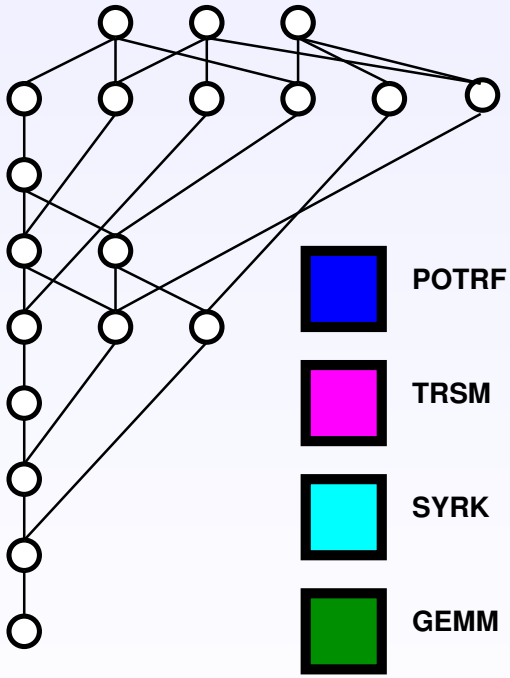


Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```

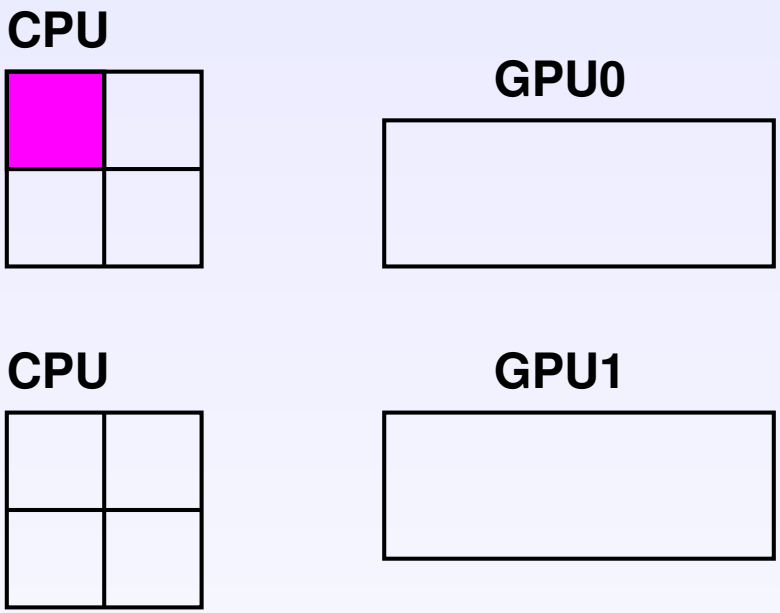


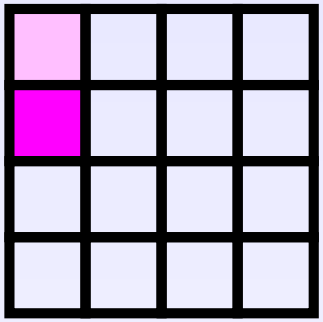


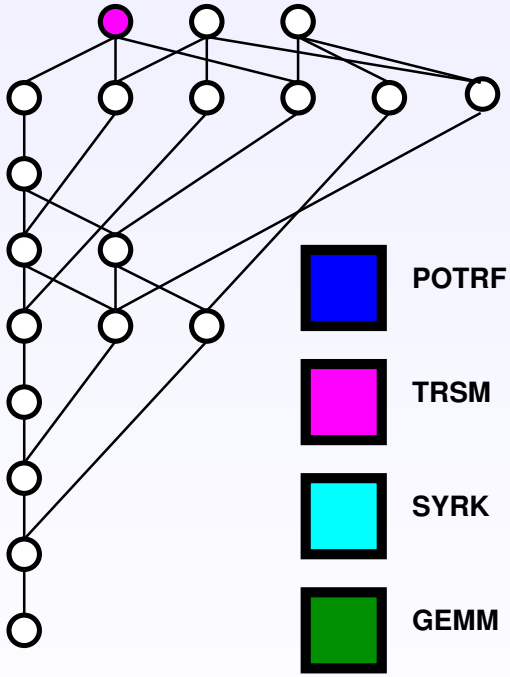


Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```

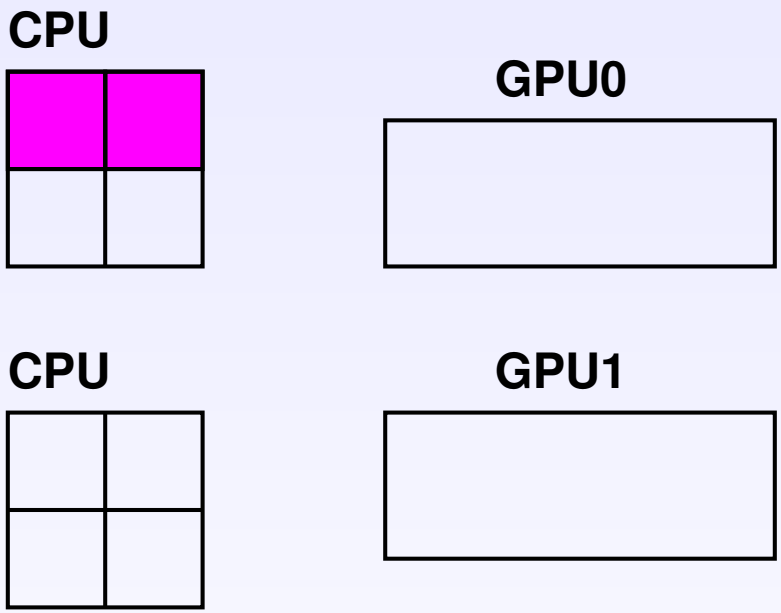


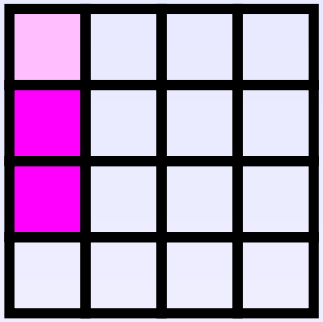


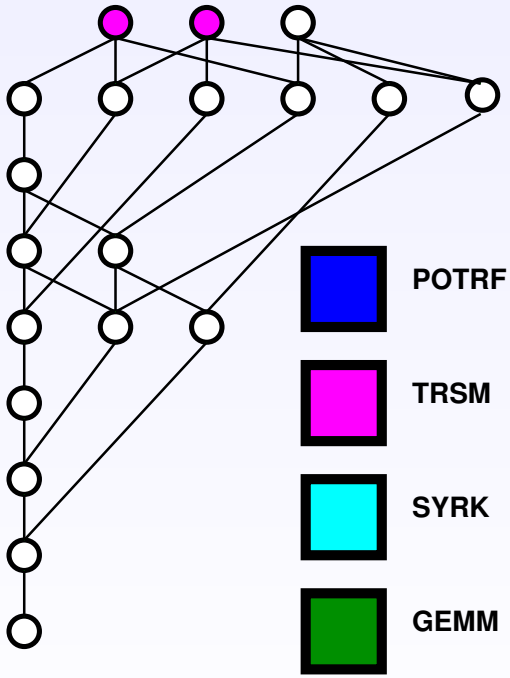


Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```

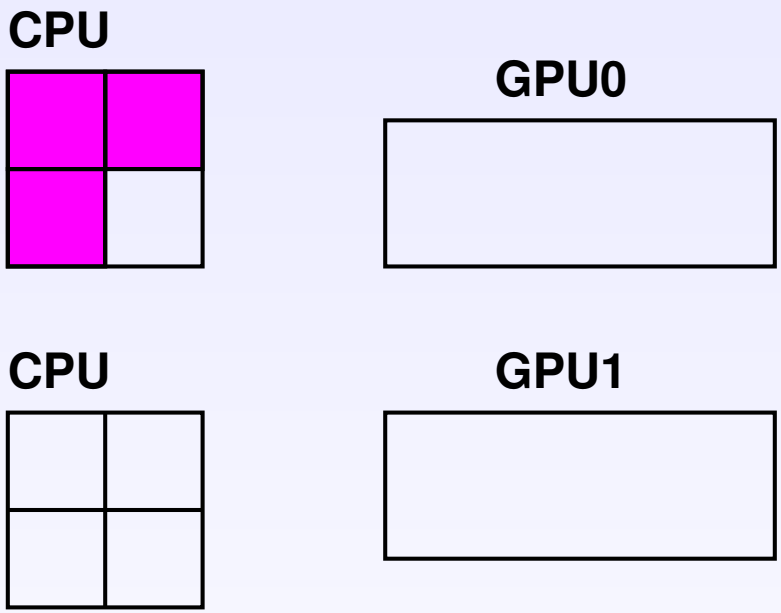


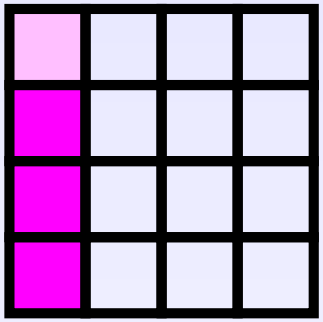


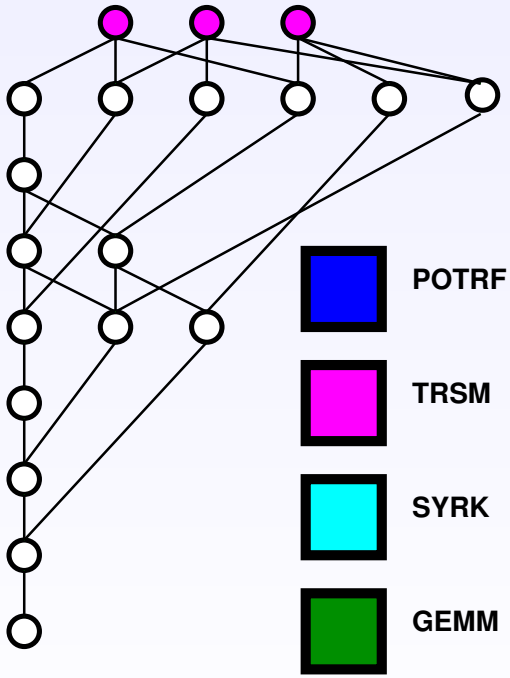


Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```

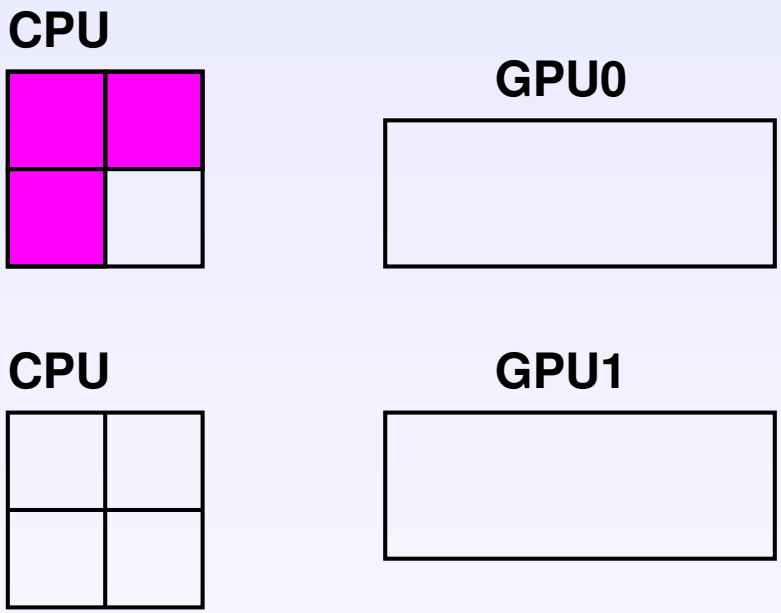


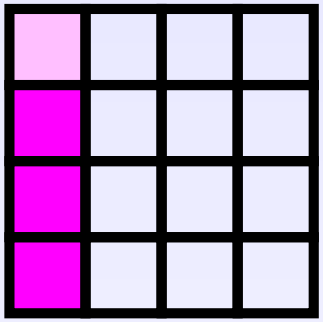


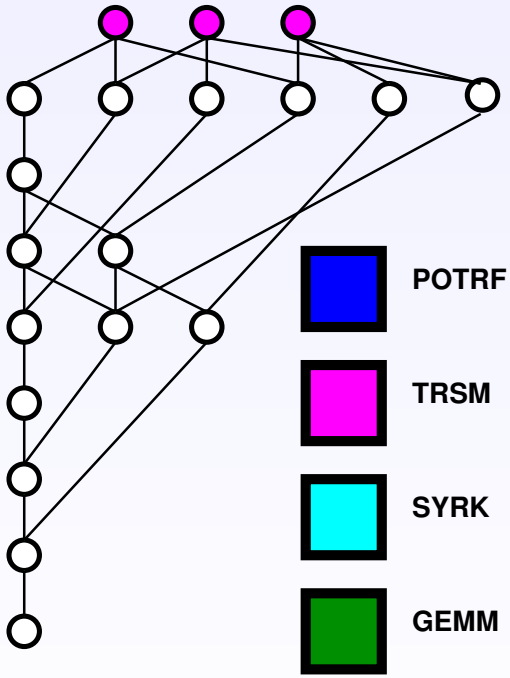


Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```



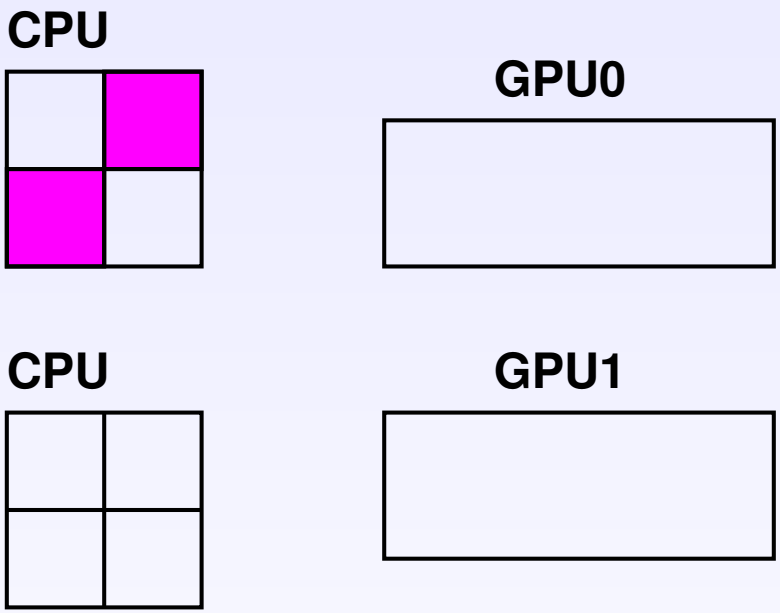


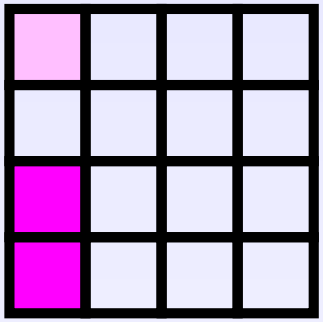


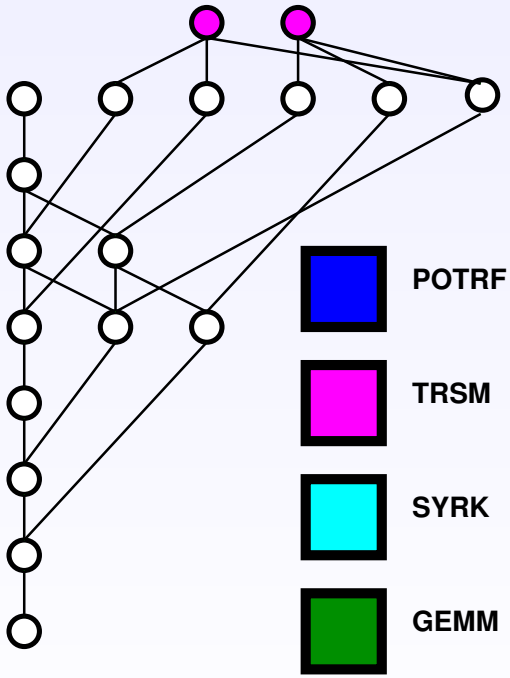
- Handles dependencies

Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```



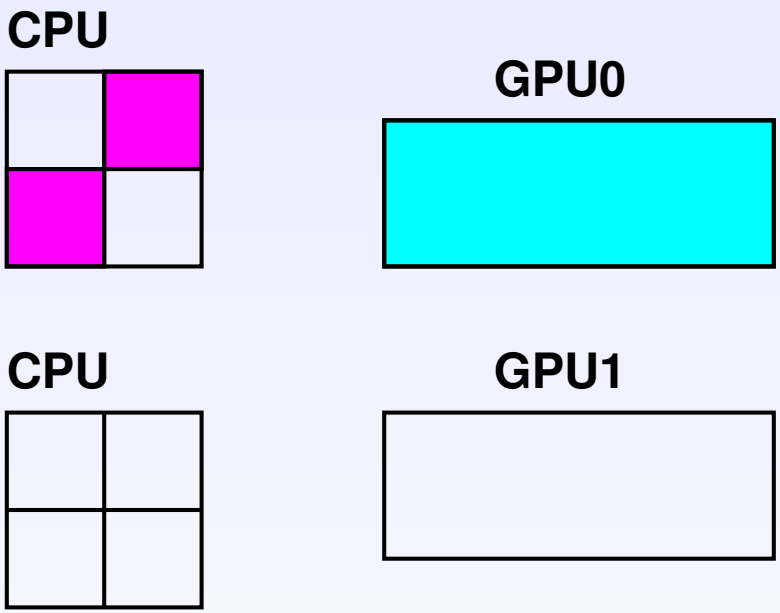


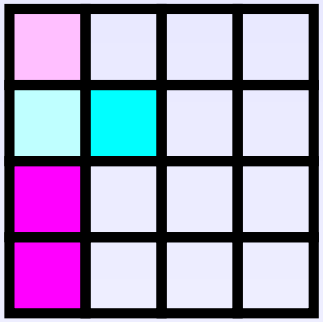


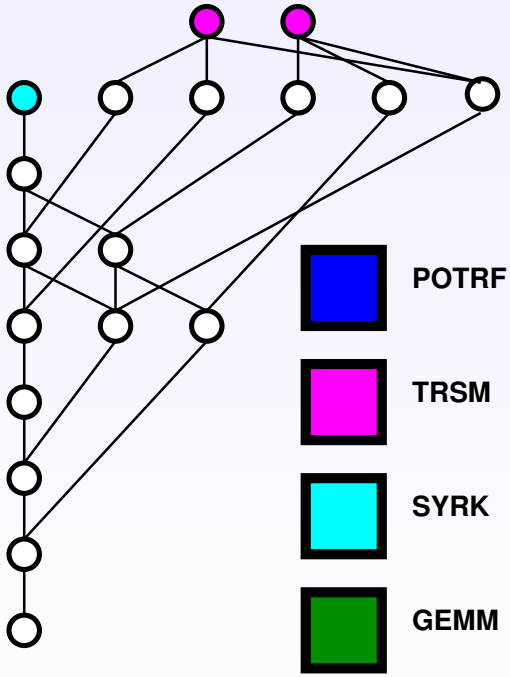
- Handles dependencies

Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```



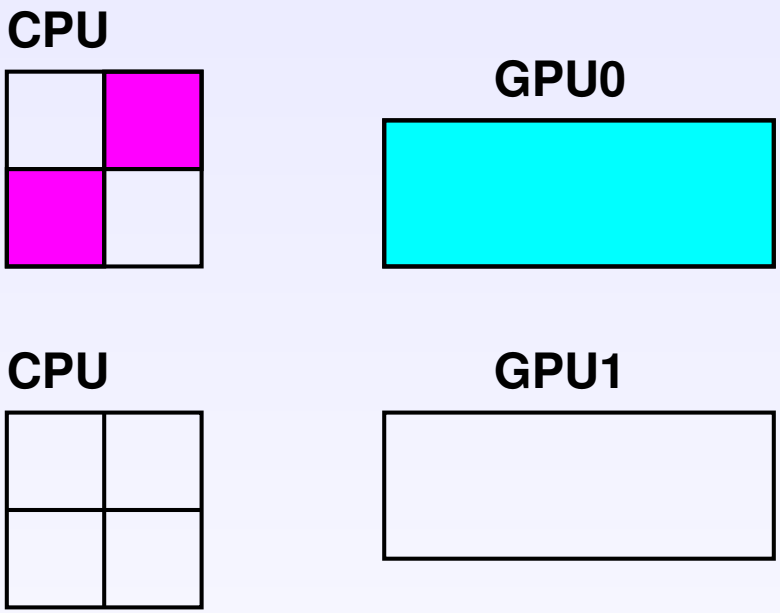


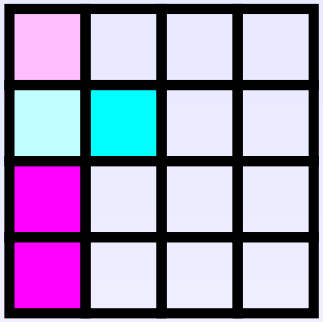


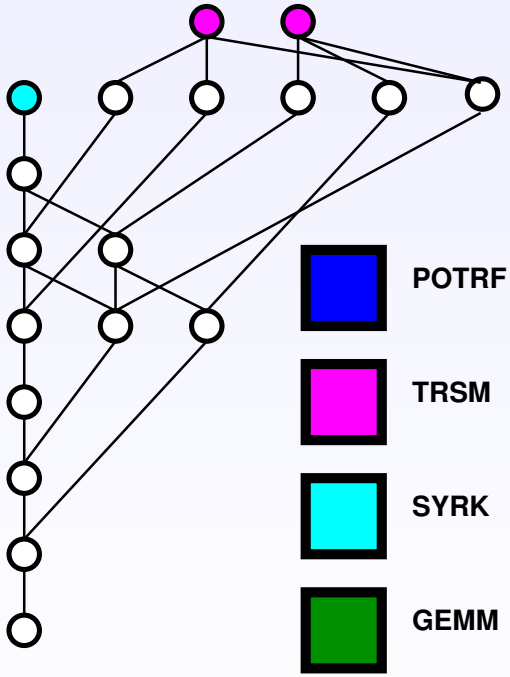
- Handles dependencies

Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```



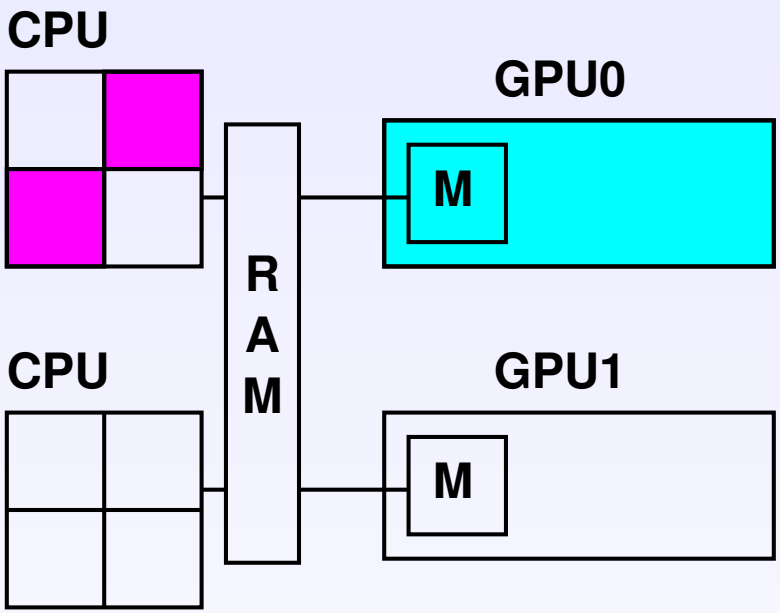


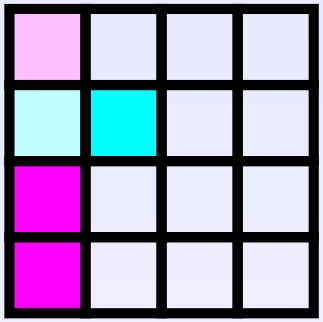


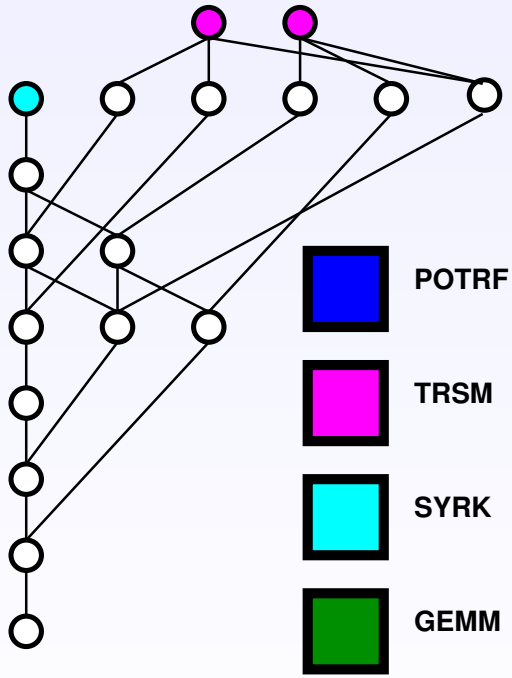
- Handles dependencies
- Handles scheduling (e.g. HEFT)

Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```



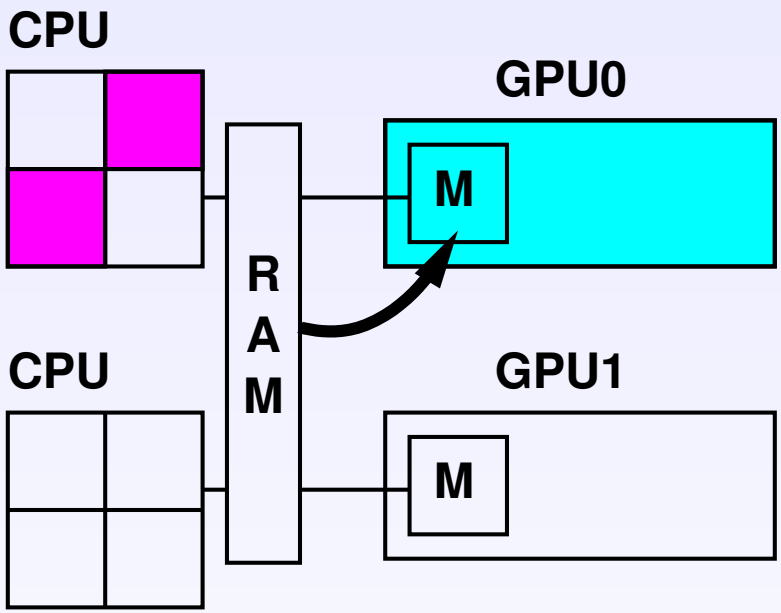


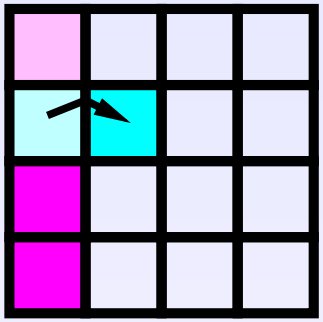


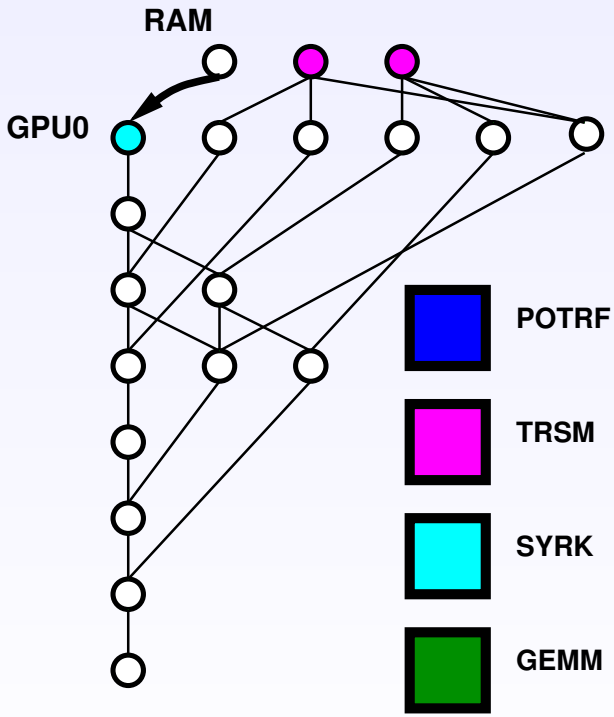
- Handles dependencies
- Handles scheduling (e.g. HEFT)

Runtime parallel execution on a heterogeneous node

```
task_wait_for_all();
```







- Handles dependencies
- Handles scheduling (e.g. HEFT)
- Handles data consistency (MSI protocol)

Summary

Sequential task-based paradigm for single node

- Sequential source code
- Runtime infers task dependencies from data dependencies
- Runtime drives and optimizes execution

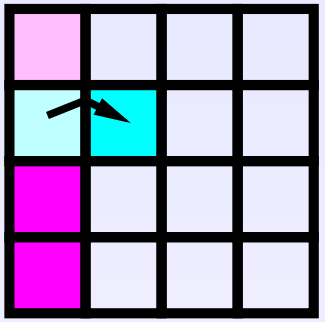
Summary

Sequential task-based paradigm for single node

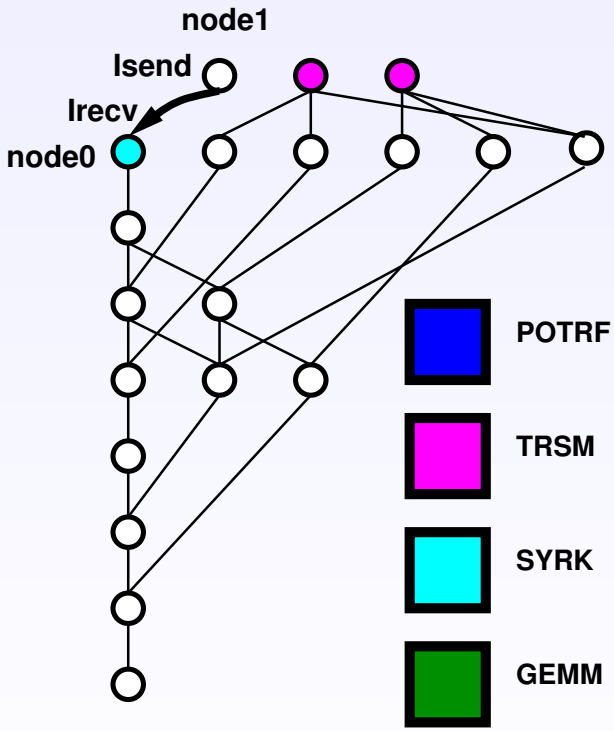
- Sequential source code
- Runtime infers task dependencies from data dependencies
- Runtime drives and optimizes execution

- How about clusters?
  - Do we really need a new programming paradigm?

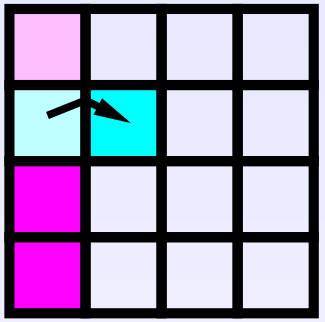
Do we need a new paradigm for clusters?



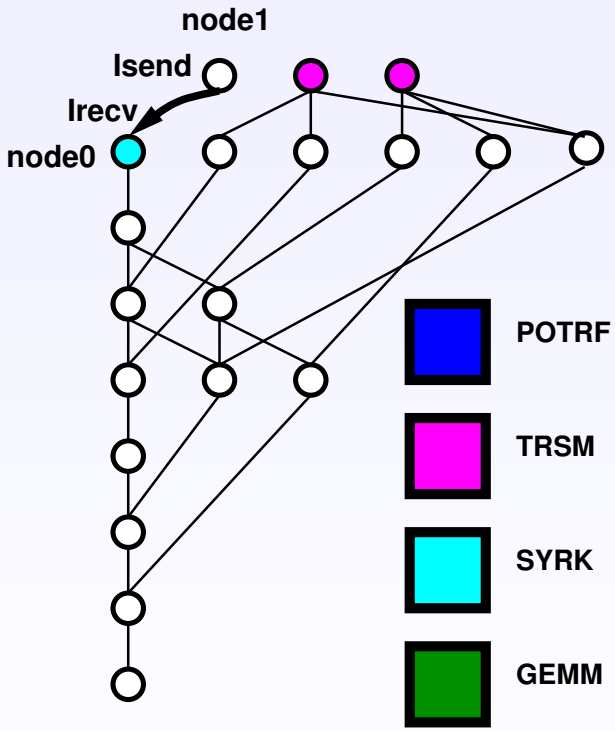
- How to express communications?



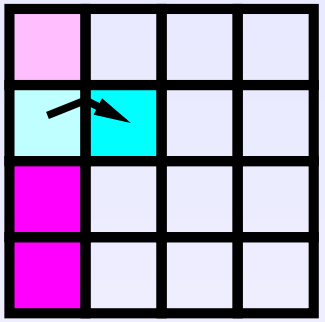
Do we need a new paradigm for clusters?



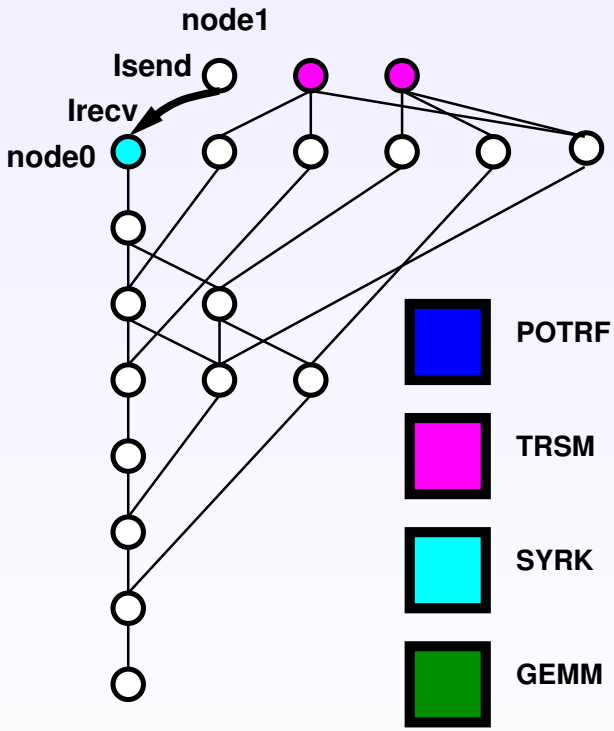
- How to express communications?
- How to establish the mapping?



Do we need a new paradigm for clusters?

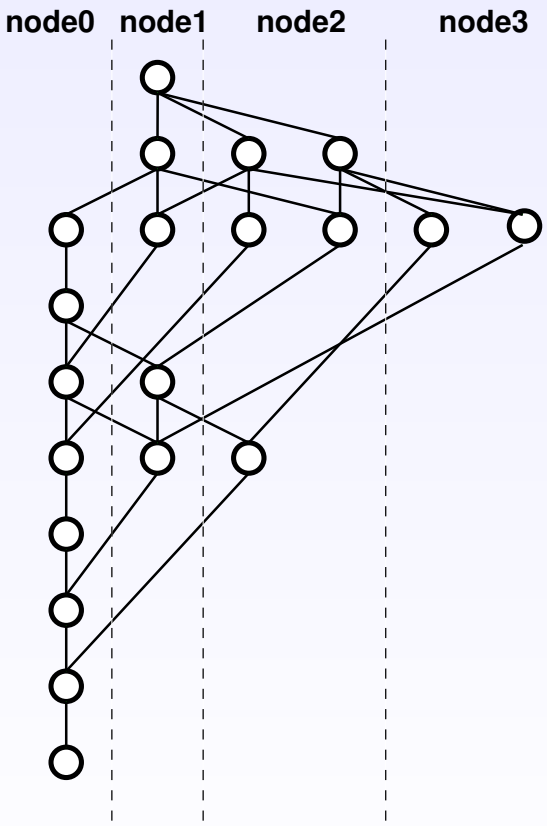


- How to express communications?
- How to establish the mapping?
- How communications will be initiated?



Mapping: Which node executes which tasks?

- The application provides the mapping



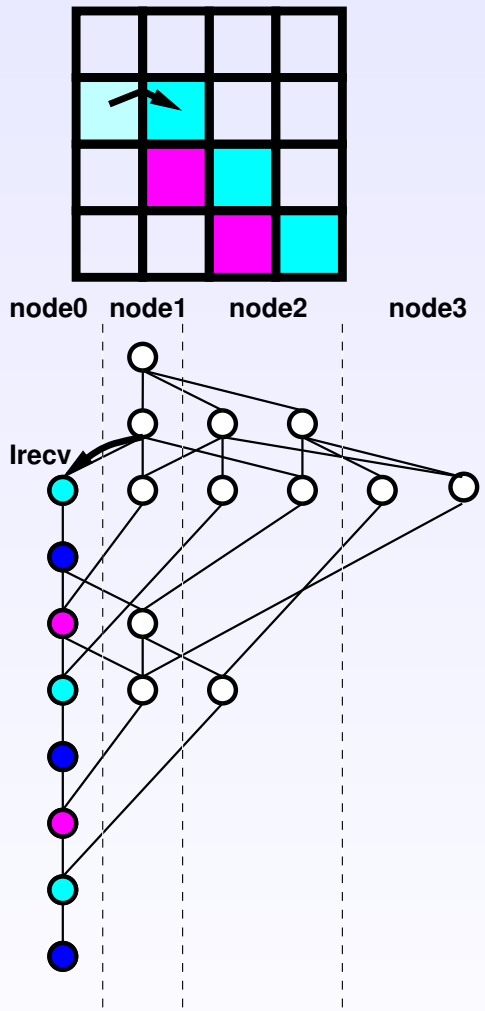
Data transfers between nodes

All nodes unroll the whole task graph

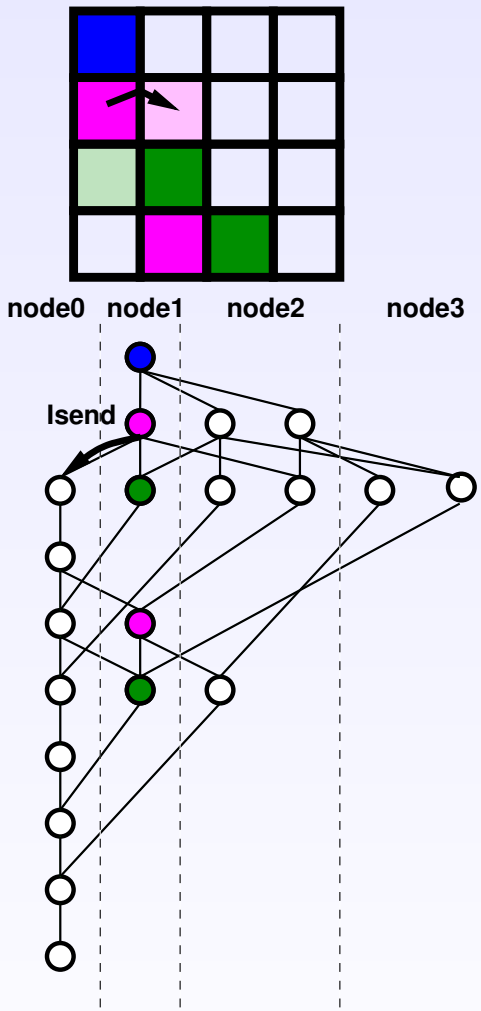
They determine tasks they will execute

They can infer required communications

No synchronization between nodes



Node 0 execution



Node 1 execution

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

Same paradigm for clusters (vs single node)

Almost same code

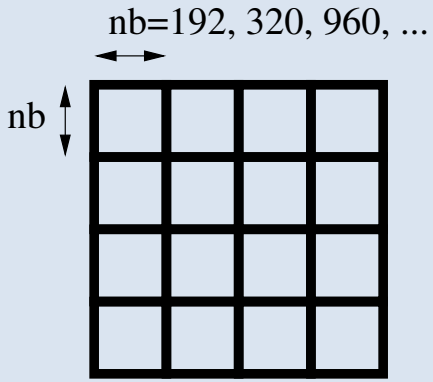
- MPI communicator
- Mapping function

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

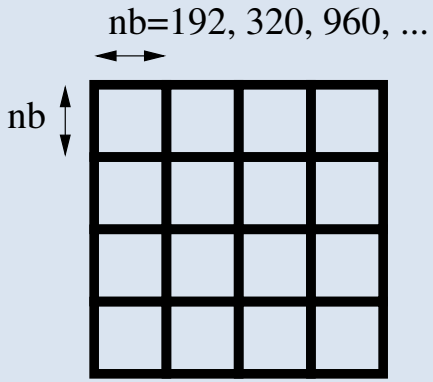
Experimental Setup on TGCC CEA Curie

- Double-precision **Cholesky**
  - Scalapack
  - Dplasma/Parsec
  - **Magma-morse/StarPU**
- 64 nodes



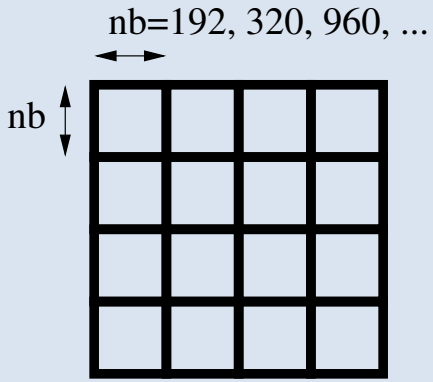
Experimental Setup on TGCC CEA Curie

- Double-precision **Cholesky**
  - Scalapack
  - Dplasma/Parsec
  - **Magma-morse/StarPU**
- 64 nodes
  - 2 Intel Westmere @ 2.66 GHz (8 cores per node)
- Homogeneous tile size: 192x192

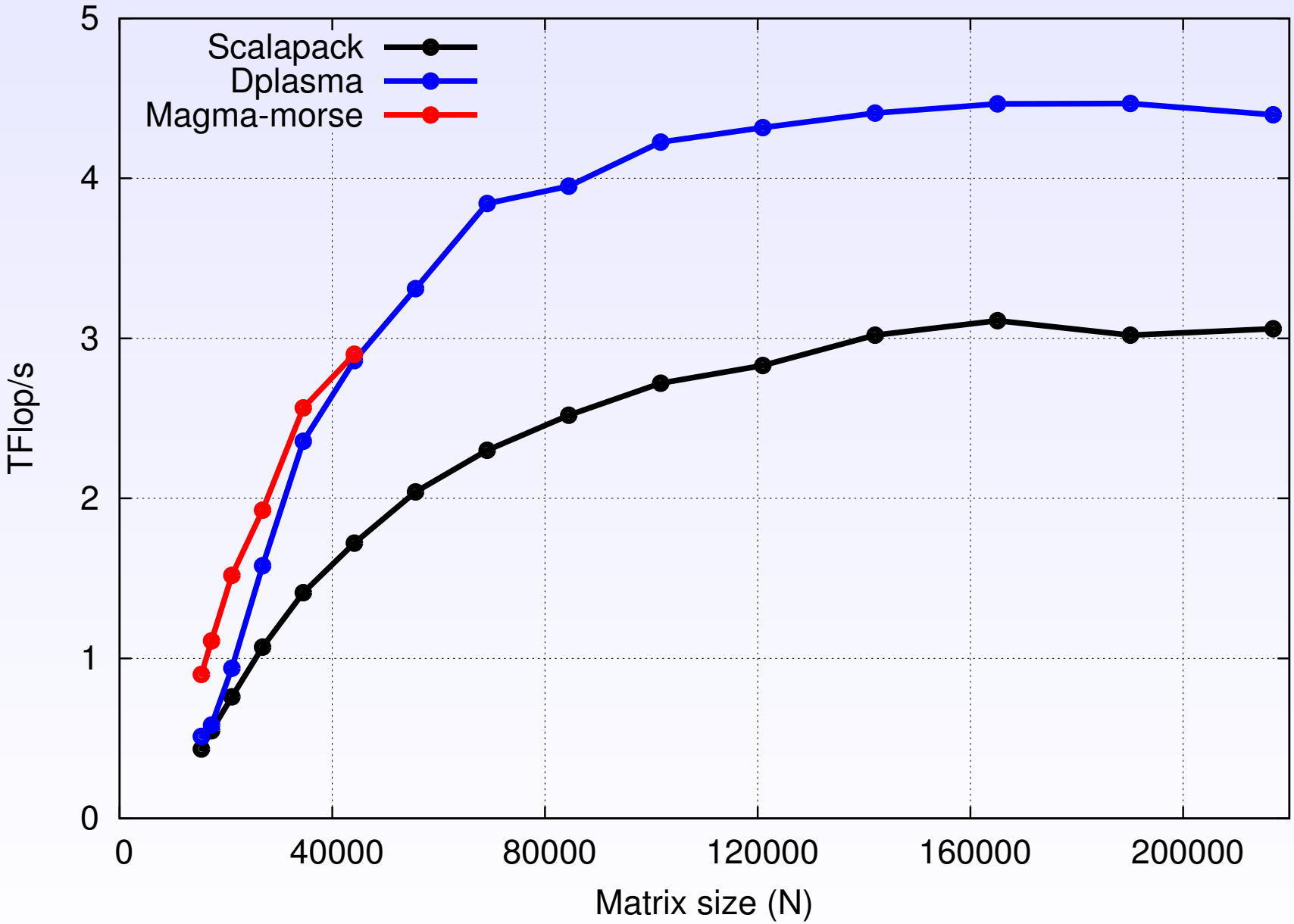


Experimental Setup on TGCC CEA Curie

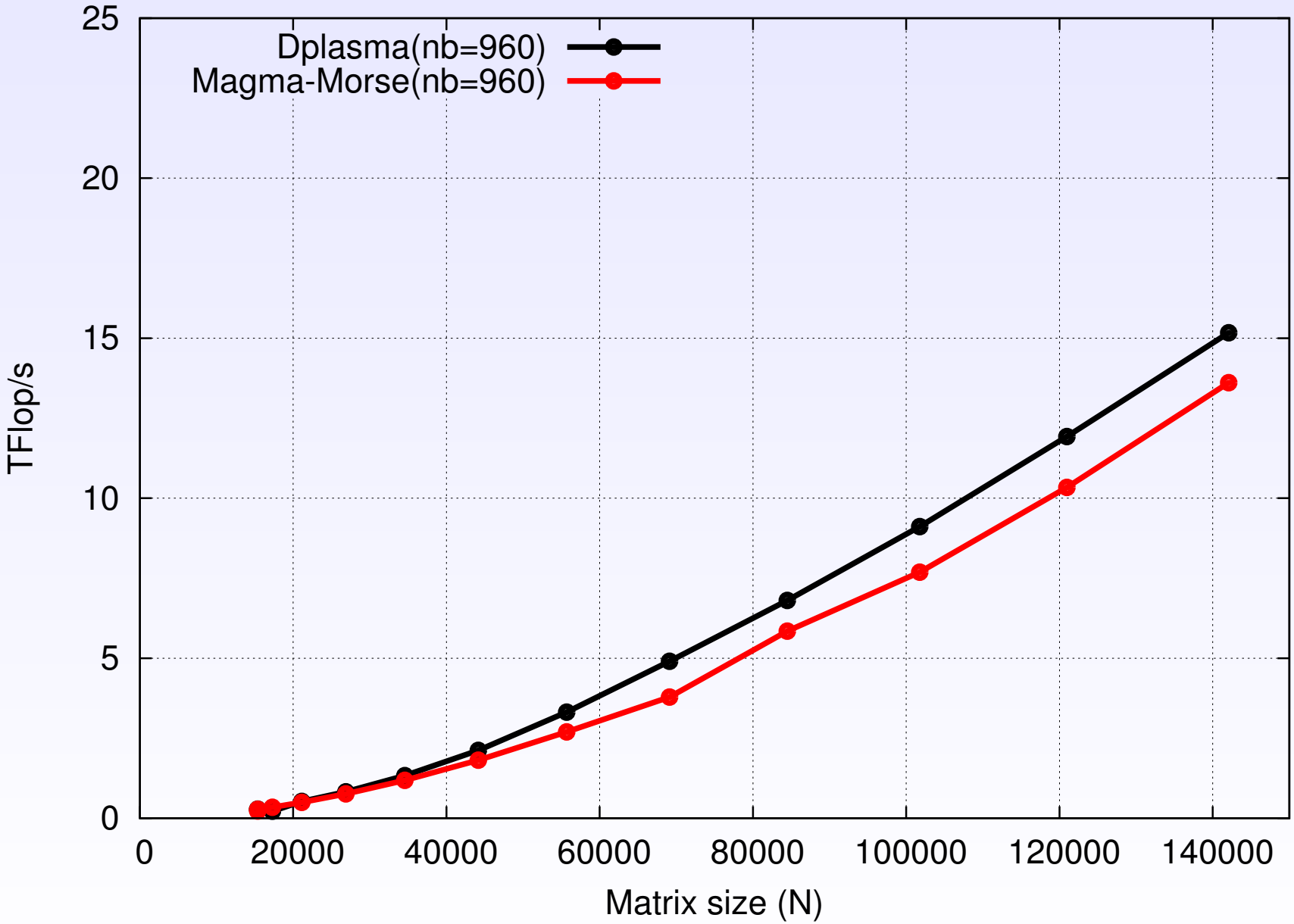
- Double-precision **Cholesky**
  - Scalapack
  - Dplasma/Parsec
  - **Magma-morse/StarPU**
- 64 nodes
  - 2 Intel Westmere @ 2.66 GHz (8 cores per node)
  - 2 Nvidia Tesla M2090 (2 GPUs per node)
- Homogeneous tile size: 192x192
- Heterogeneous tile sizes: 320x320 / 960x960



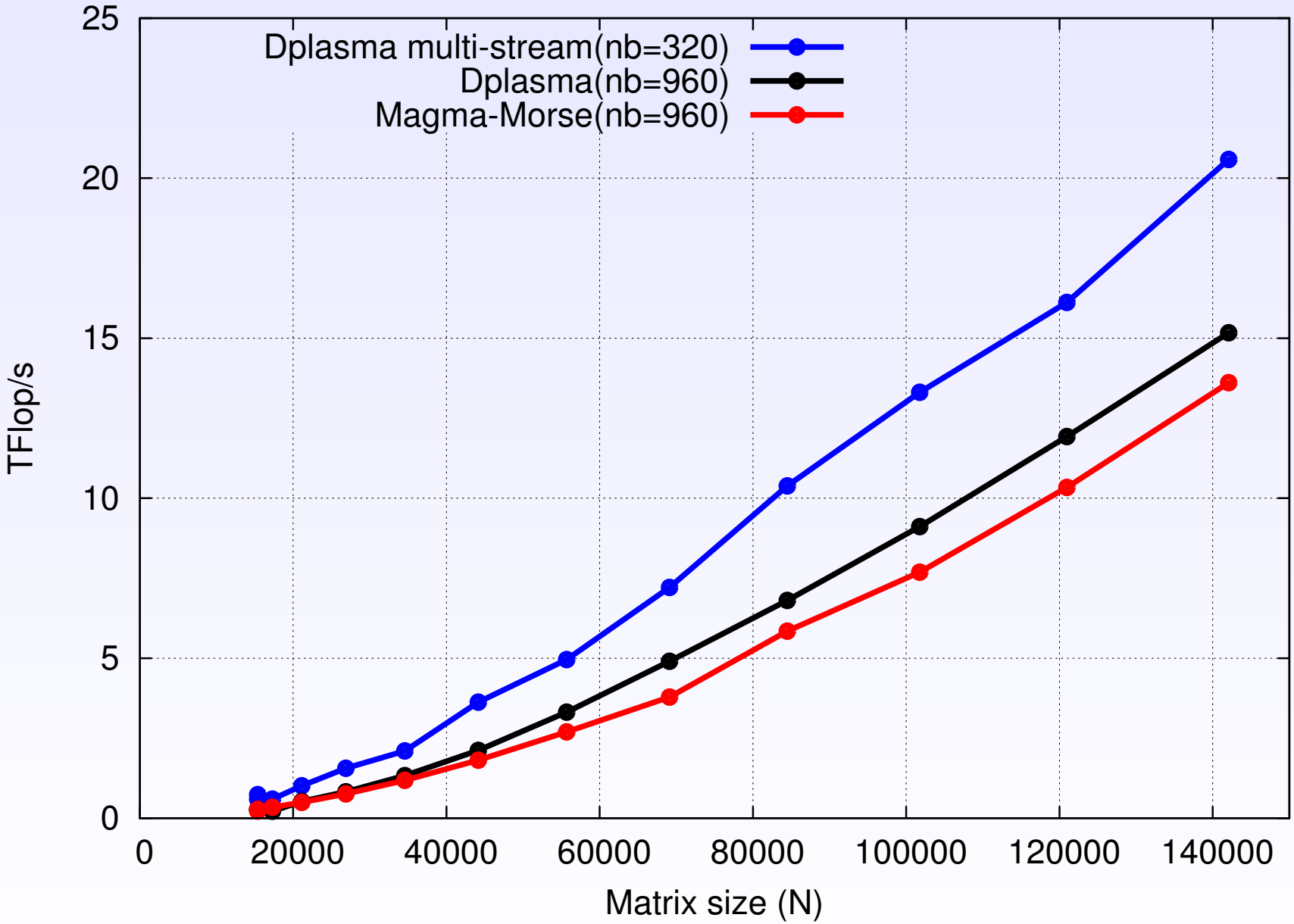
64 homogeneous nodes (8 cores per node)



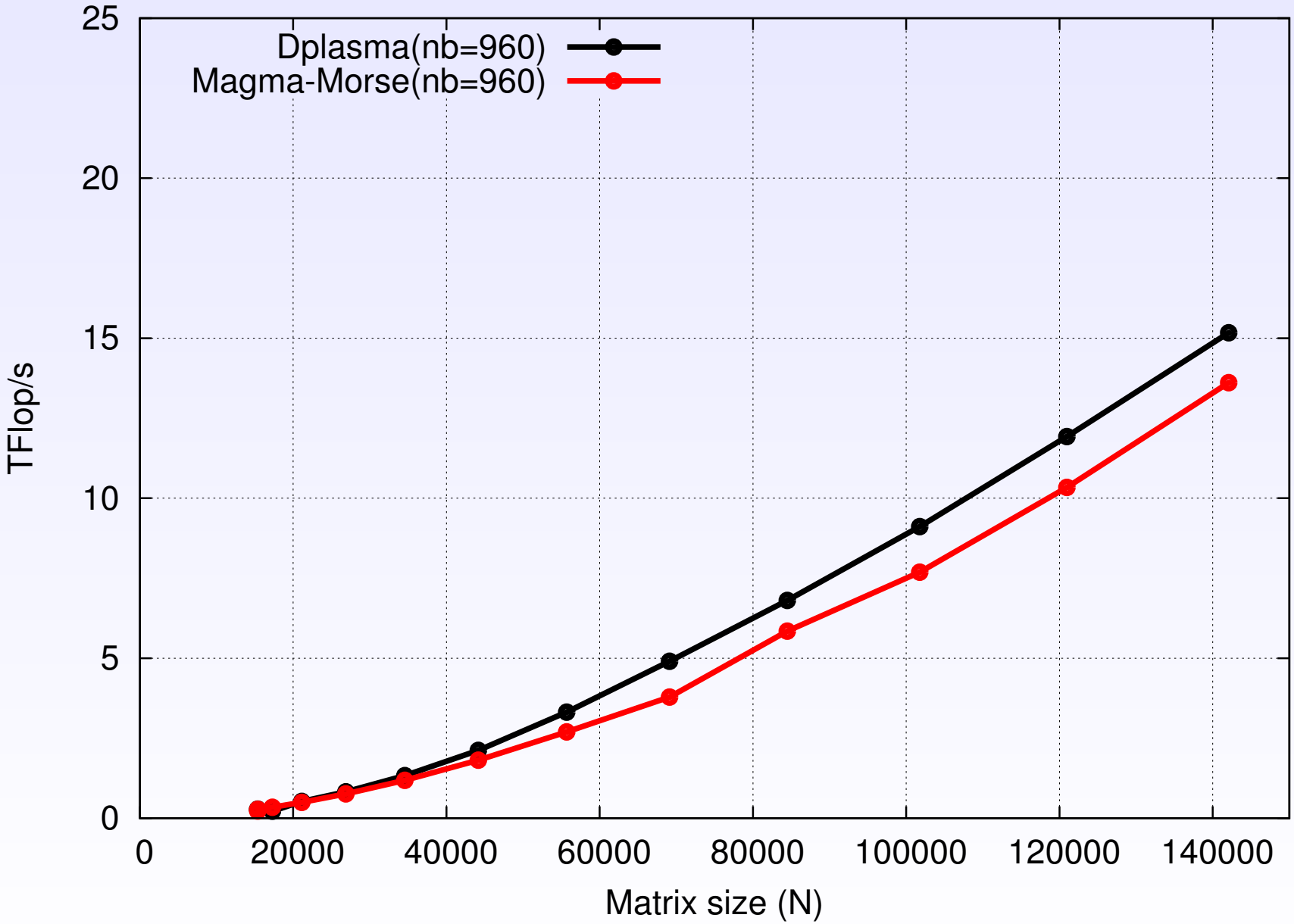
64 heterogeneous nodes (8 cores + 2 GPUs per node)



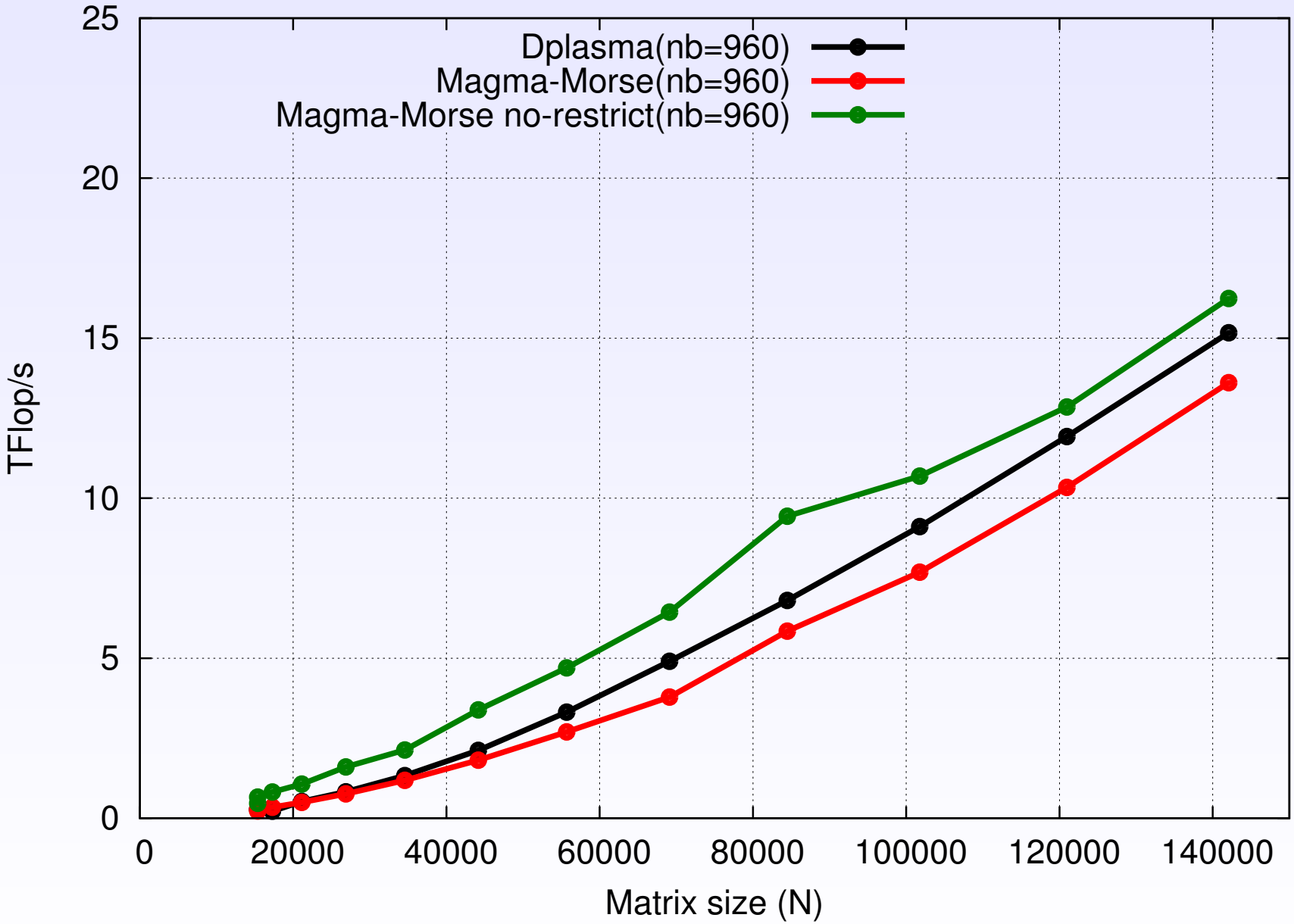
64 heterogeneous nodes (8 cores + 2 GPUs per node)



64 heterogeneous nodes (8 cores + 2 GPUs per node)



64 heterogeneous nodes (8 cores + 2 GPUs per node)



Conclusion

Contribution

- Harnessing cluster of hybrid nodes
- Sequential task-based paradigm
- **Almost no code changes vs single node**
- **Competitive performance**

Future work

- Extension to other LAPACK-like routines
- Release it into MAGMA library
- Dynamic inter-node load balancing

Morse: http://icl.cs.utk.edu/morse/

StarPU: http://runtime.bordeaux.inria.fr/StarPU/