MORSE

Matrices Over Runtime Systems @ Exascale

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

Inria informatics mathematics

KAUST

University of Colorado Denver

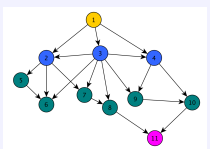
Linear algebra

$$AX = B$$

Sequential-Task-Flow

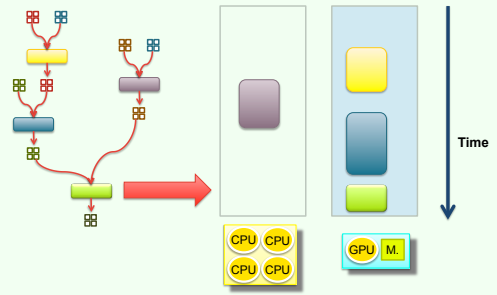
```
for (j = 0; j < N; j++)
  Task (A[j]);
```

Direct Acyclic Graph



→

Runtime systems



→

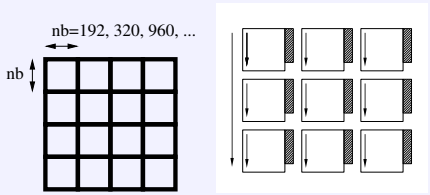
Heterogeneous platforms




Case study: dense linear algebra

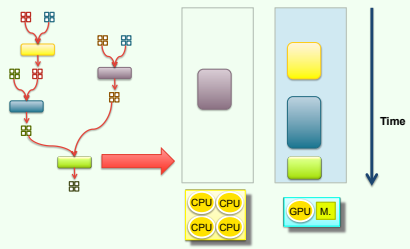
Chameleon = dense linear algebra tile algorithms (STF) on top of runtime systems

Tile matrix layout



Runtime systems

- QUARK
- StarPU



STF **PLASMA** Algorithms

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

Optimized kernels

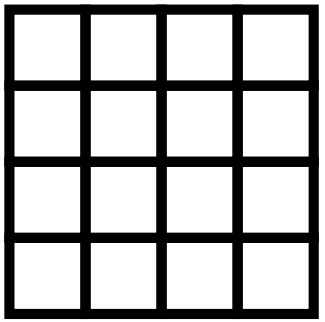
- BLAS, LAPACK
- cuBLAS, MAGMA

Outline

Outline

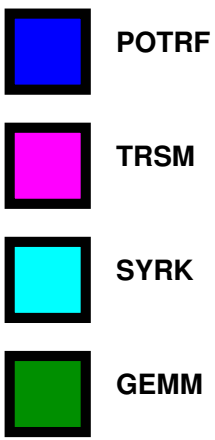
STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent



work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```

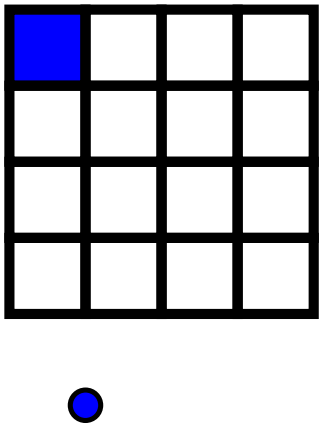


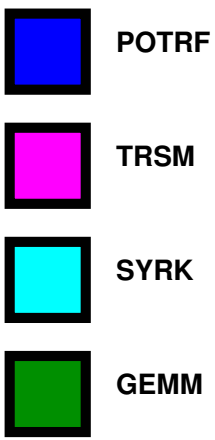
STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```



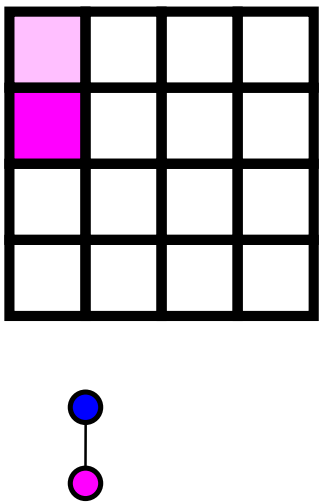


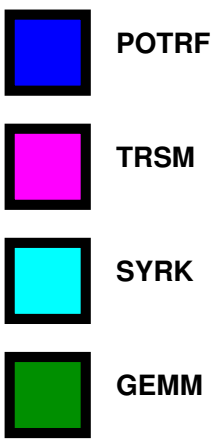
STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```



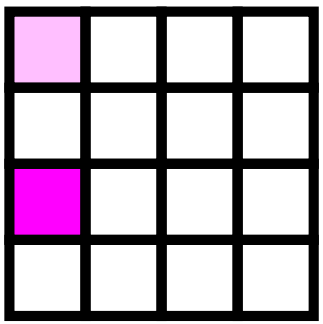


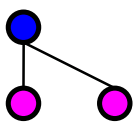
STF Cholesky Algorithm on homogeneous node

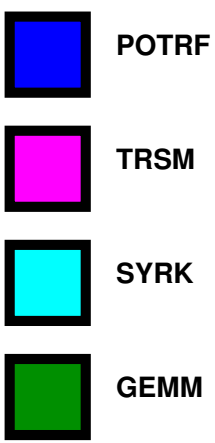
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





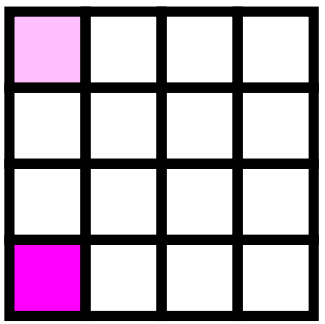


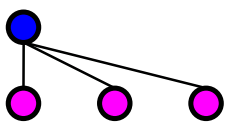
STF Cholesky Algorithm on homogeneous node

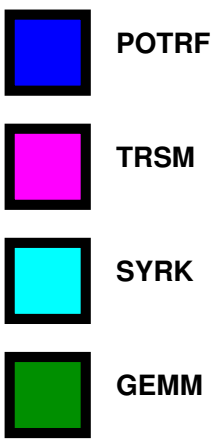
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





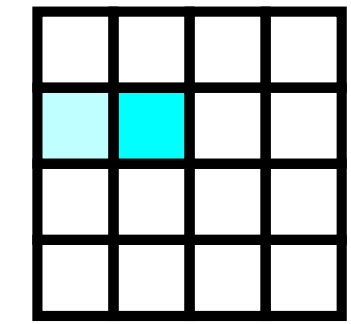


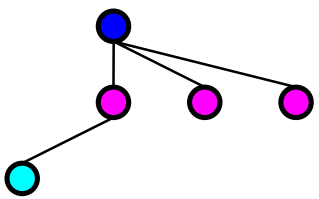
STF Cholesky Algorithm on homogeneous node

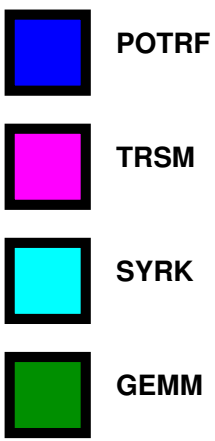
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





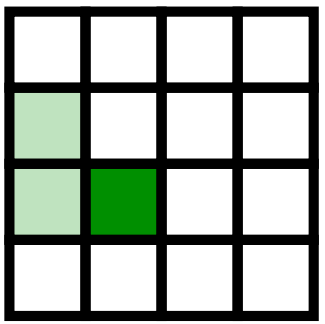


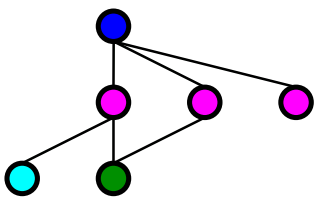
STF Cholesky Algorithm on homogeneous node

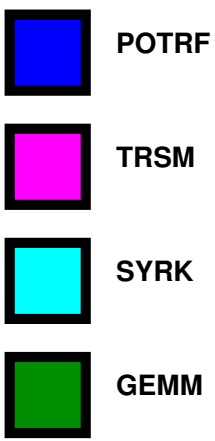
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





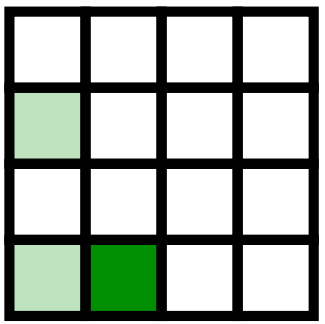


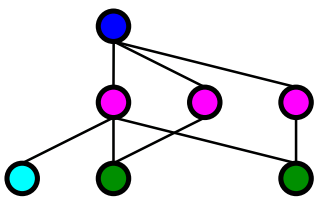
STF Cholesky Algorithm on homogeneous node

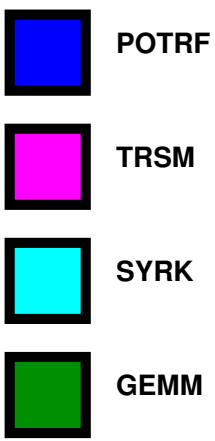
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





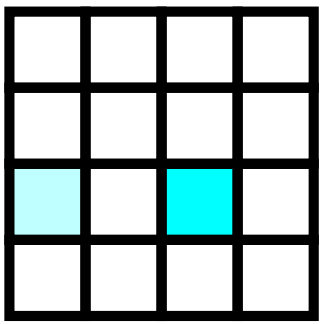


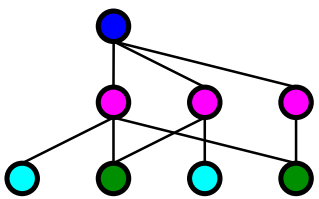
STF Cholesky Algorithm on homogeneous node

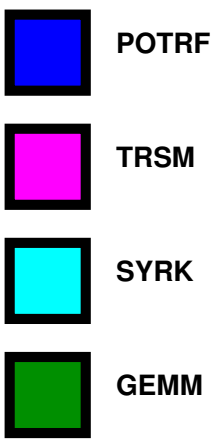
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





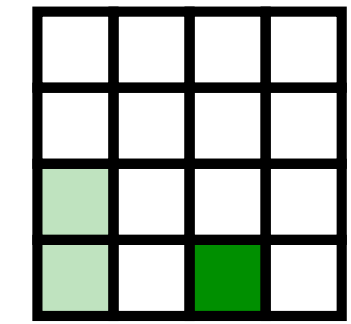


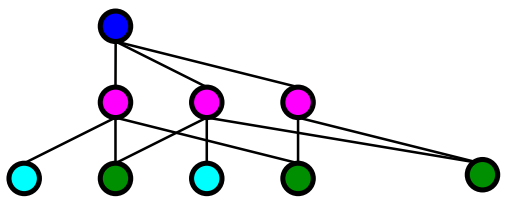
STF Cholesky Algorithm on homogeneous node

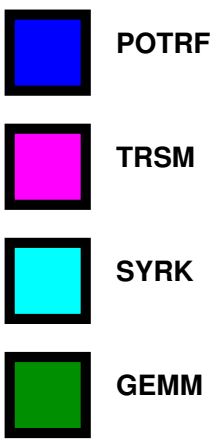
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





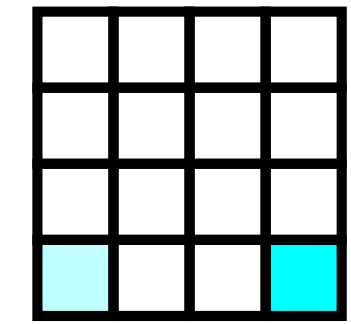


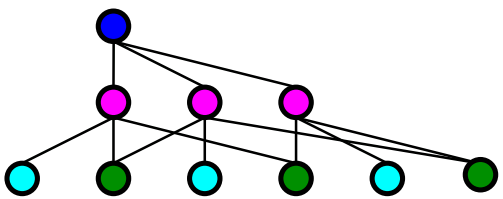
STF Cholesky Algorithm on homogeneous node

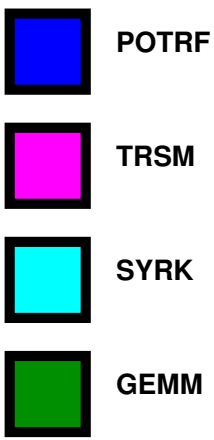
PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```





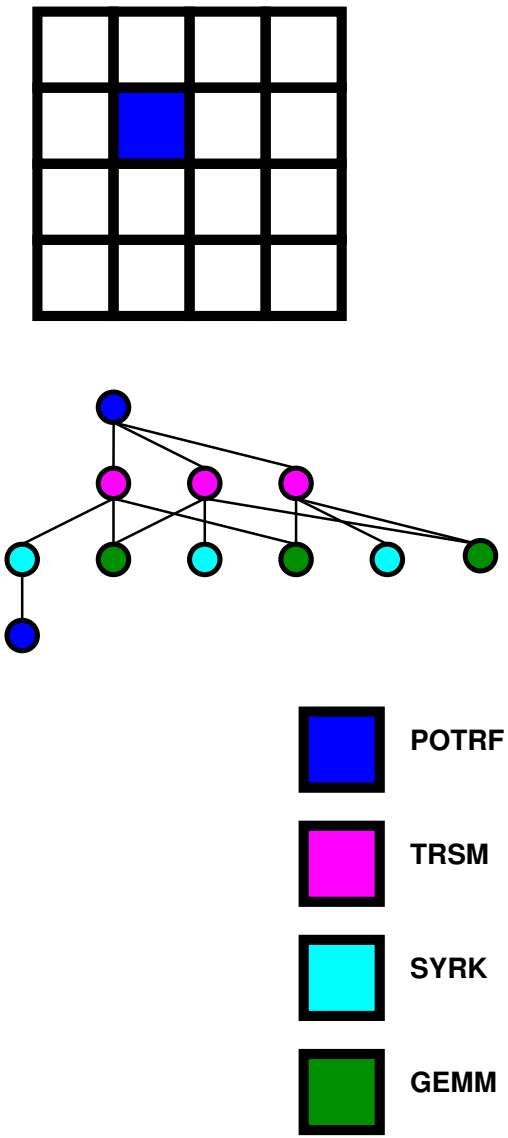


STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent

work on tiles → CPU kernels

```
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]);
  }
}
__wait__();
```

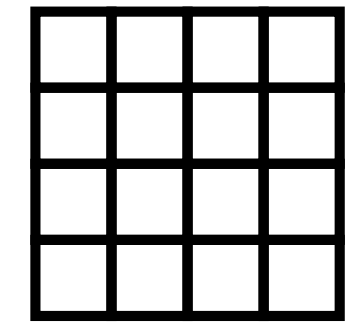


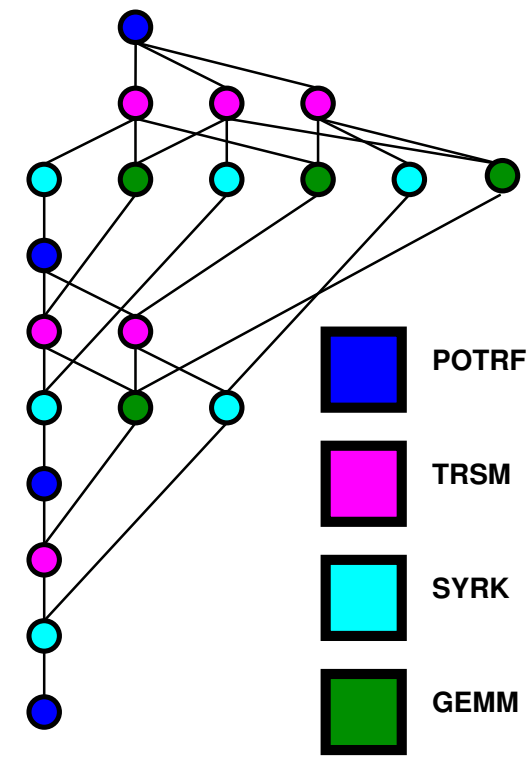
STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent

work on tiles → CPU kernels

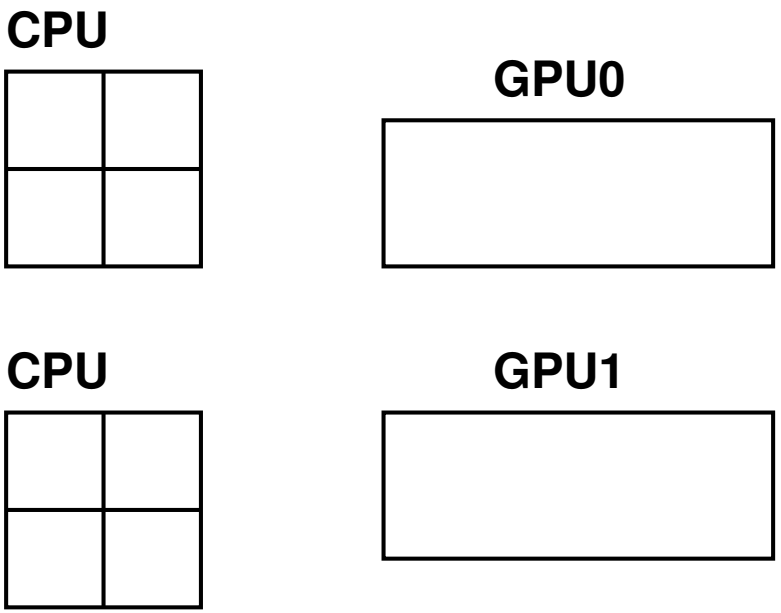
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

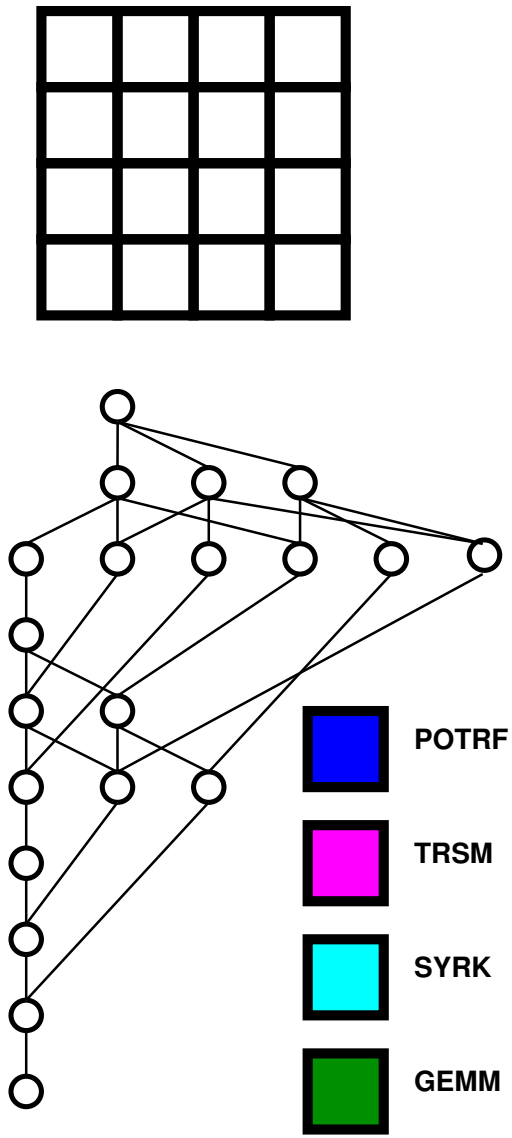




On heterogeneous node

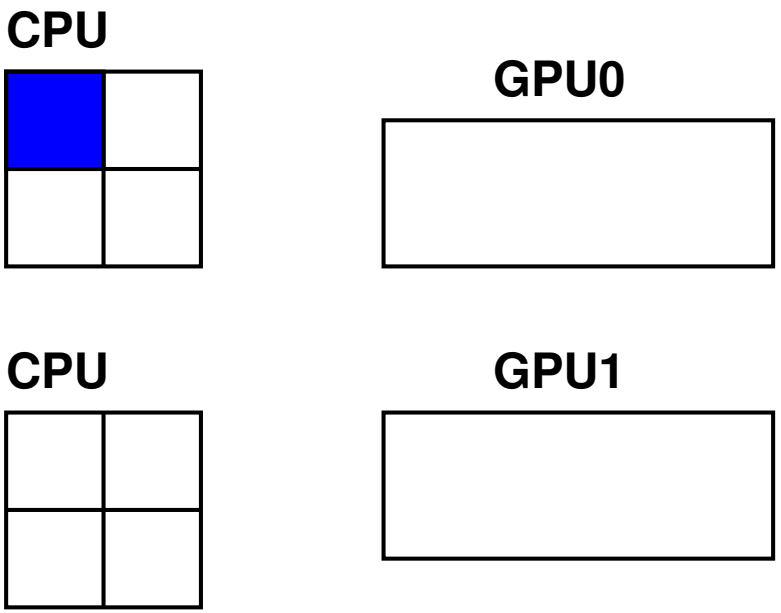
work on tiles → CPU + GPU kernels

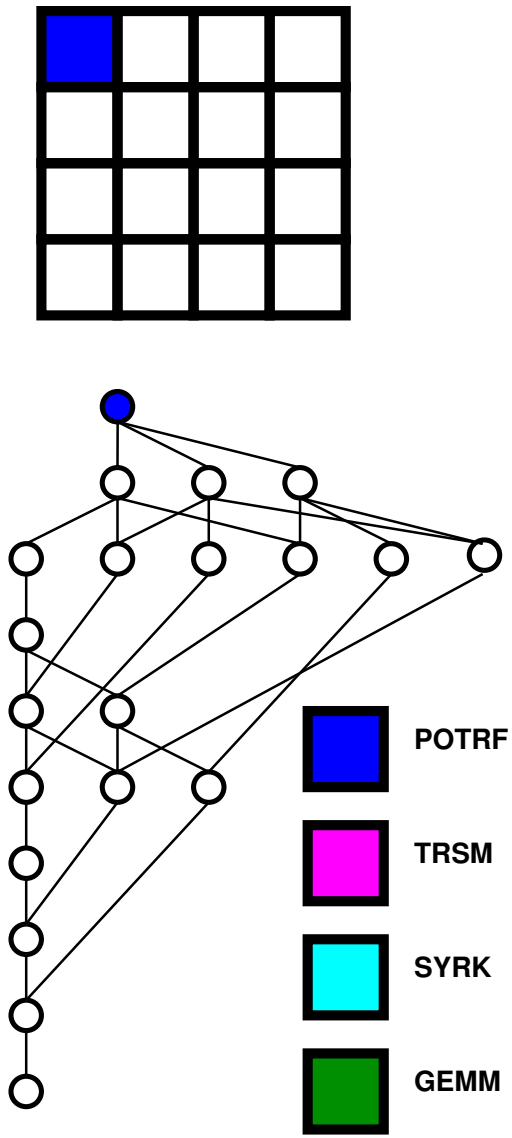




On heterogeneous node

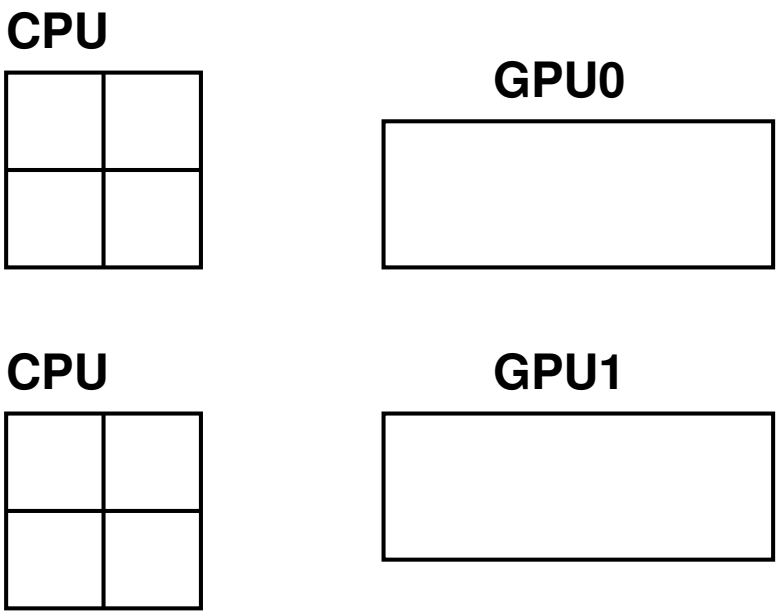
work on tiles → CPU + GPU kernels

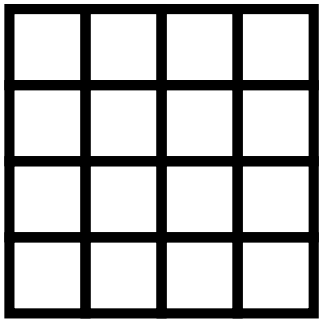


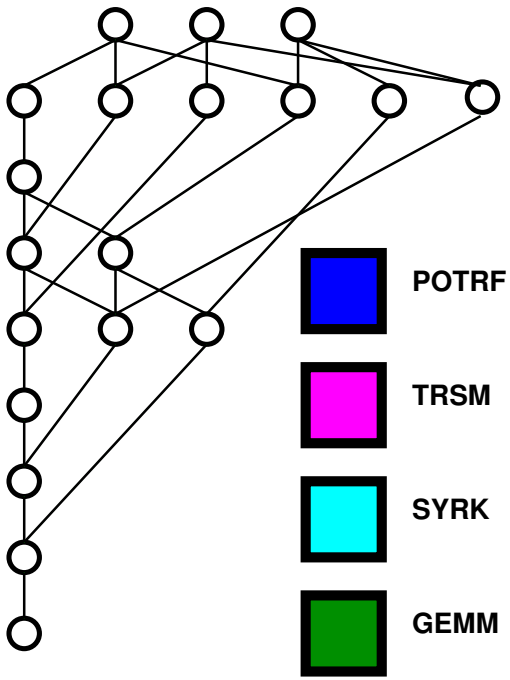


On heterogeneous node

work on tiles → CPU + GPU kernels

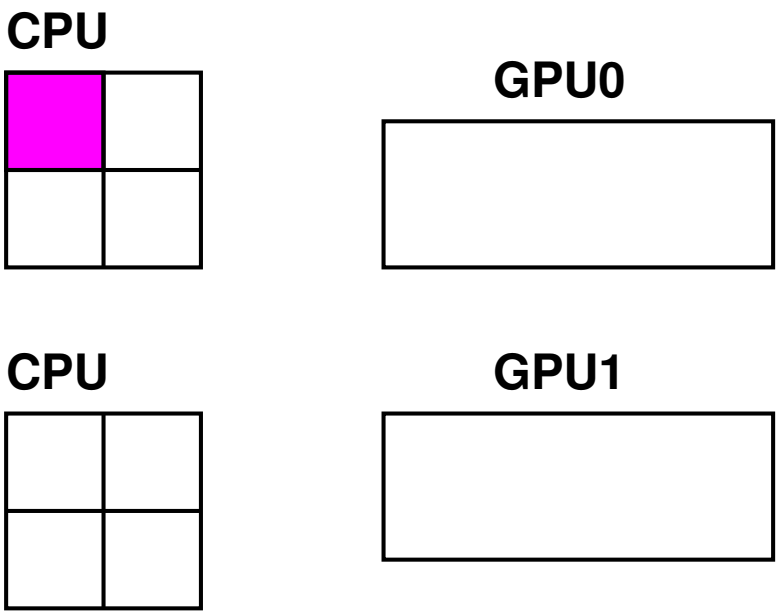


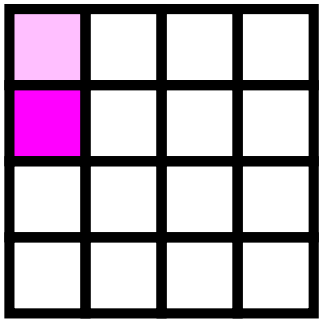


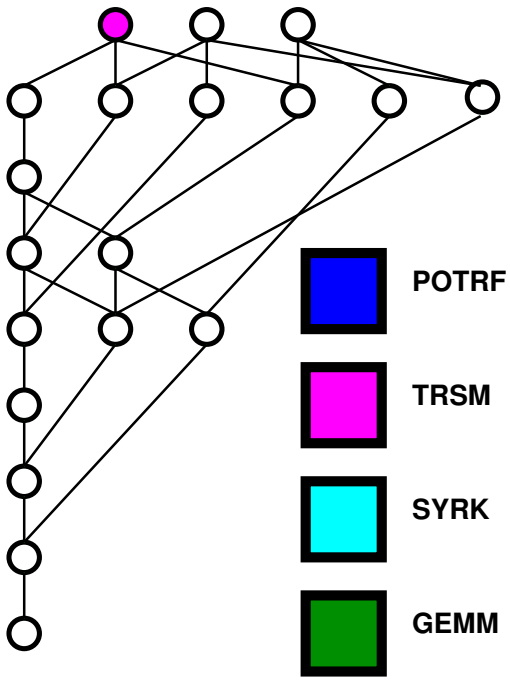


On heterogeneous node

work on tiles → CPU + GPU kernels

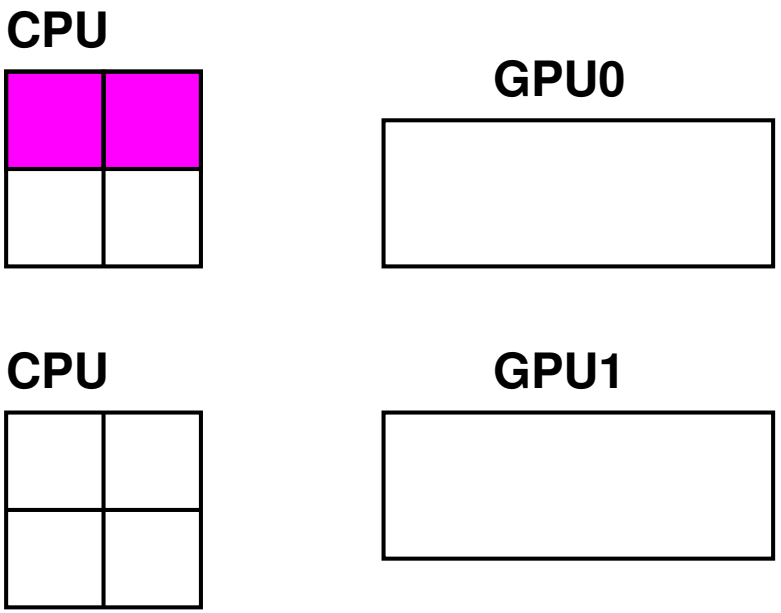


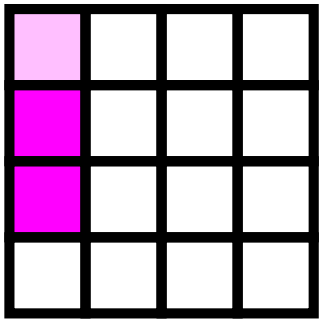


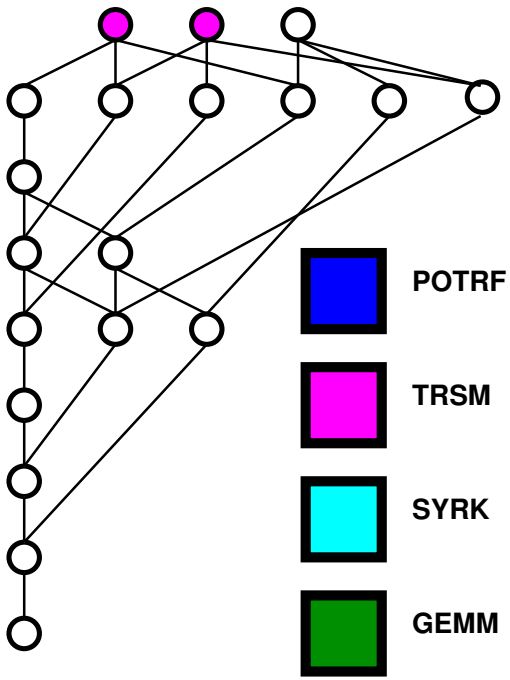


On heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

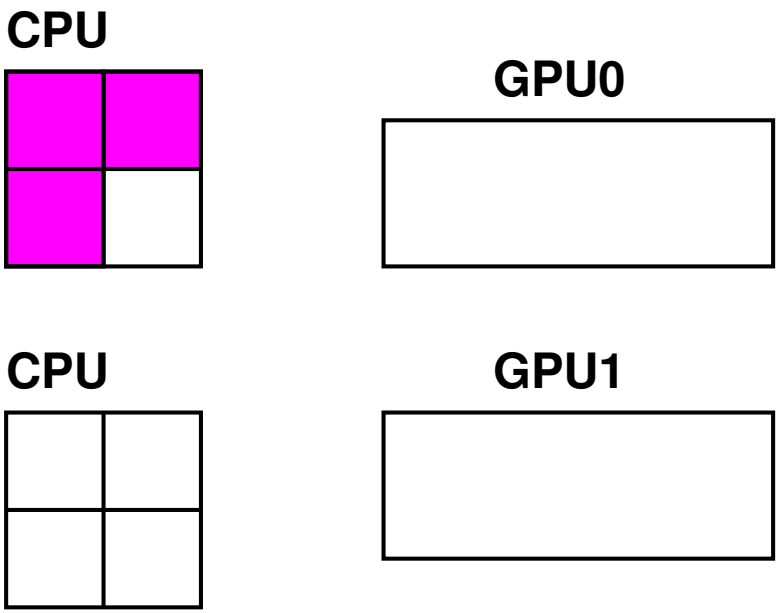


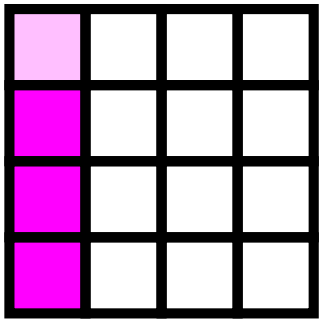


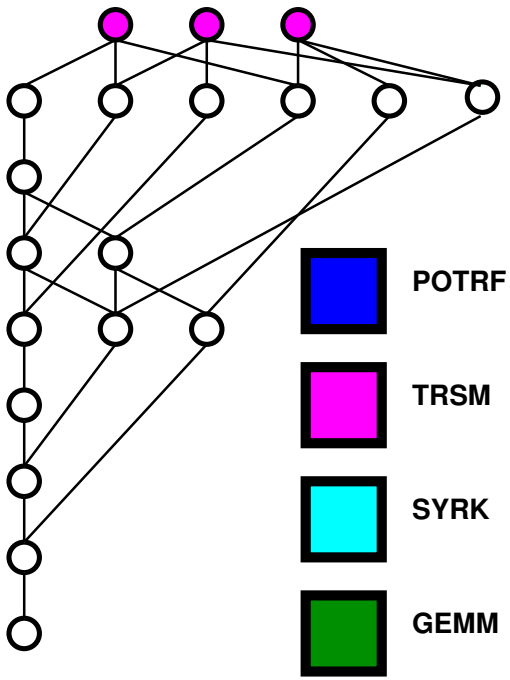


On heterogeneous node

work on tiles → CPU + GPU kernels

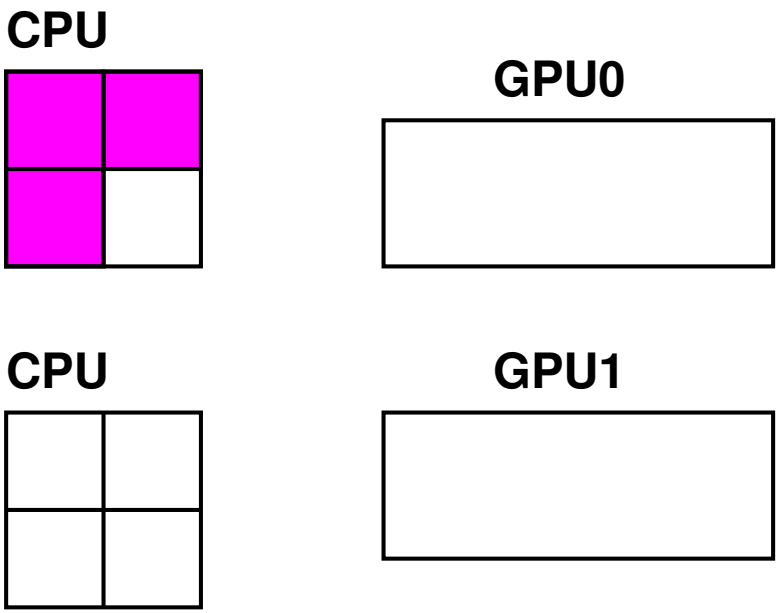


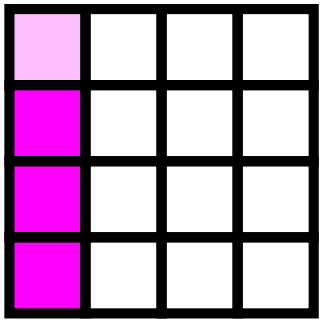


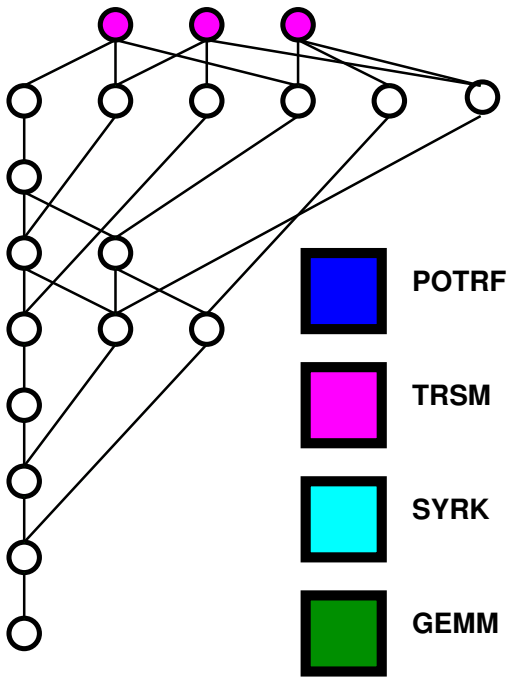


On heterogeneous node

work on tiles → CPU + GPU kernels



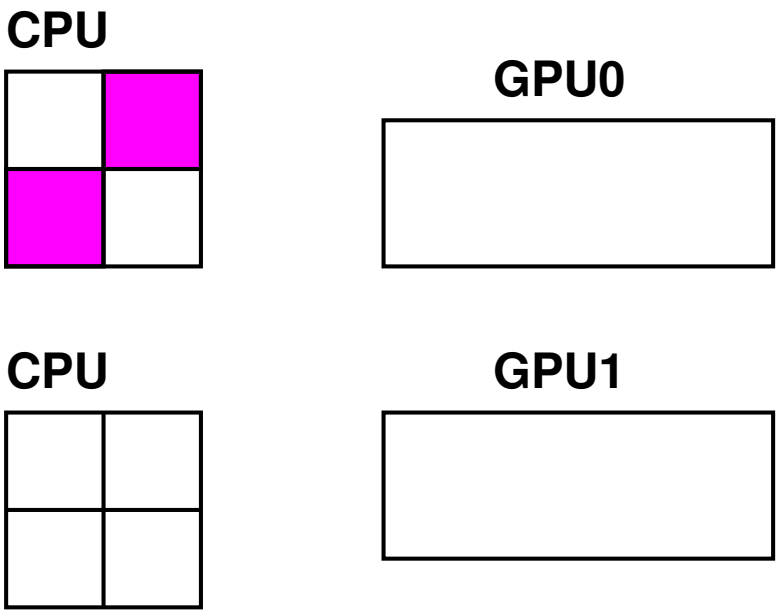


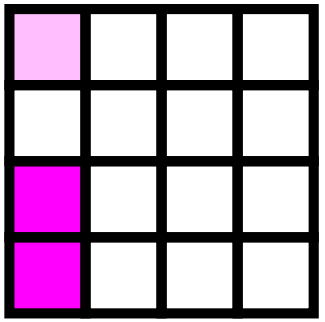


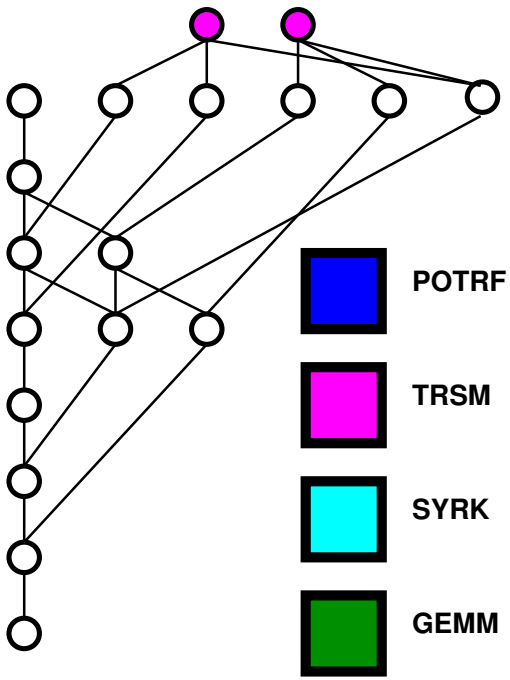
- Handles dependencies

On heterogeneous node

work on tiles → CPU + GPU kernels



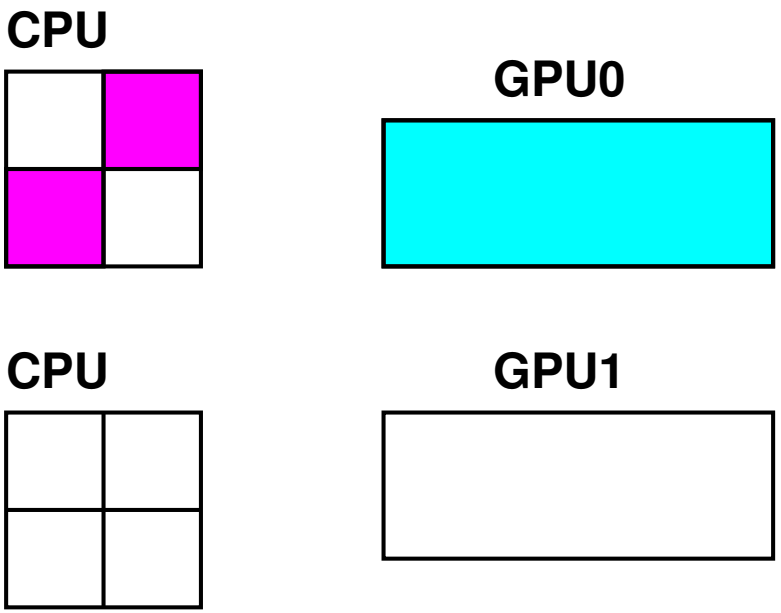


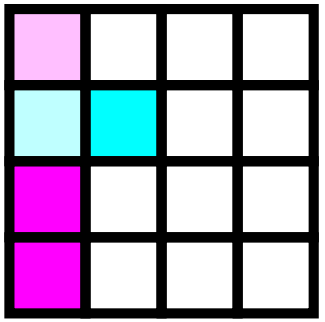


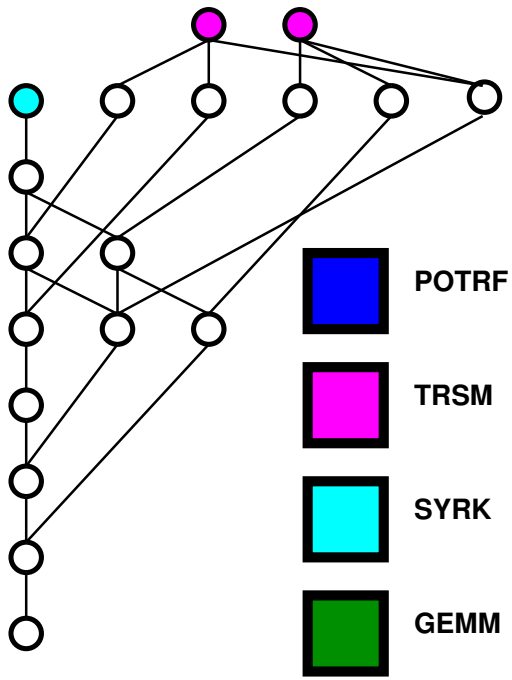
- Handles dependencies

On heterogeneous node

work on tiles → CPU + GPU kernels



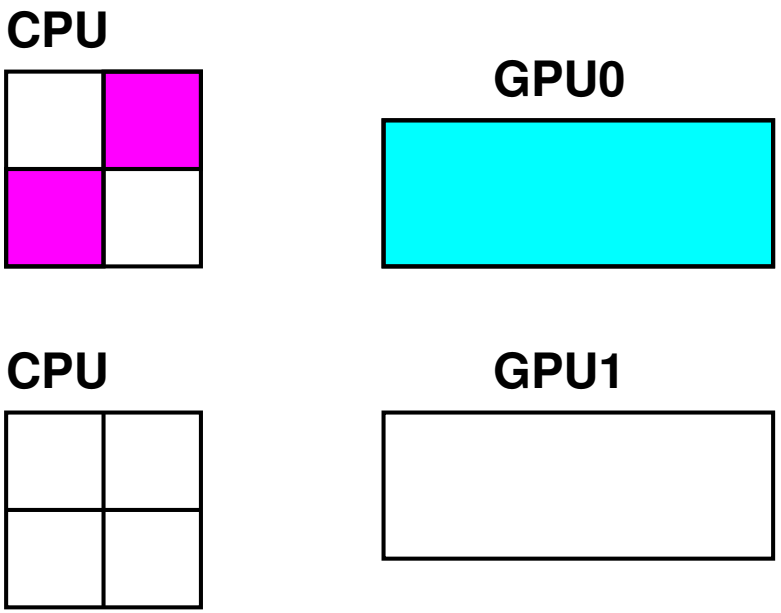


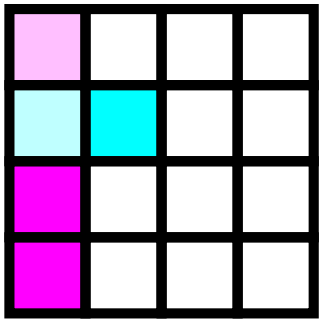


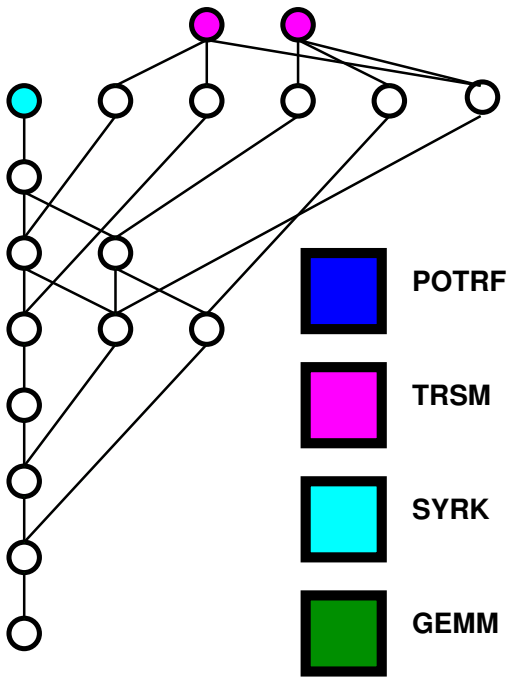
- Handles dependencies

On heterogeneous node

work on tiles → CPU + GPU kernels



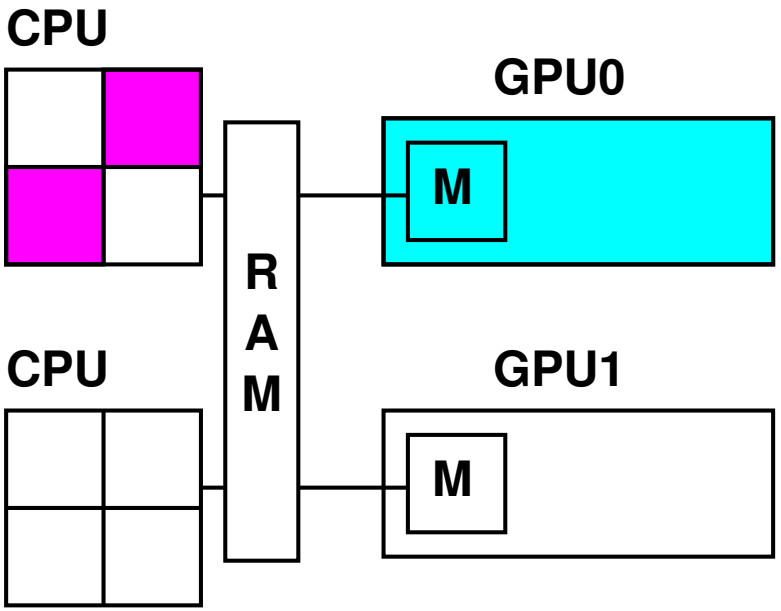


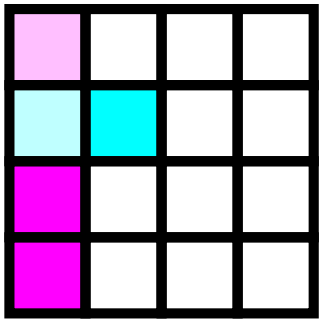


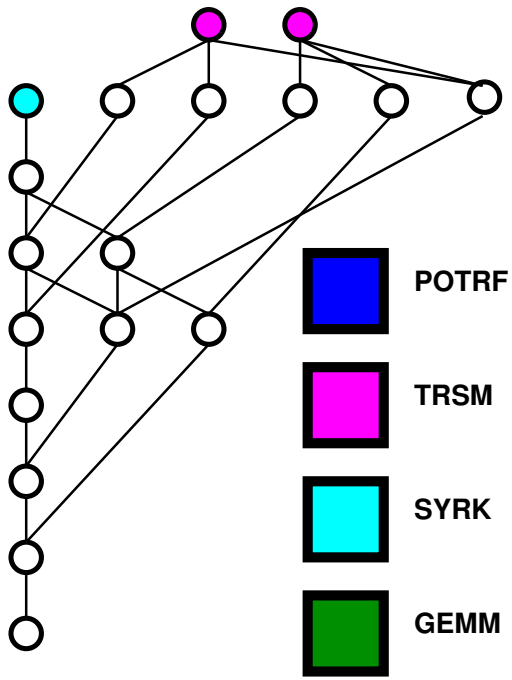
- Handles dependencies
- Handles scheduling

On heterogeneous node

work on tiles → CPU + GPU kernels



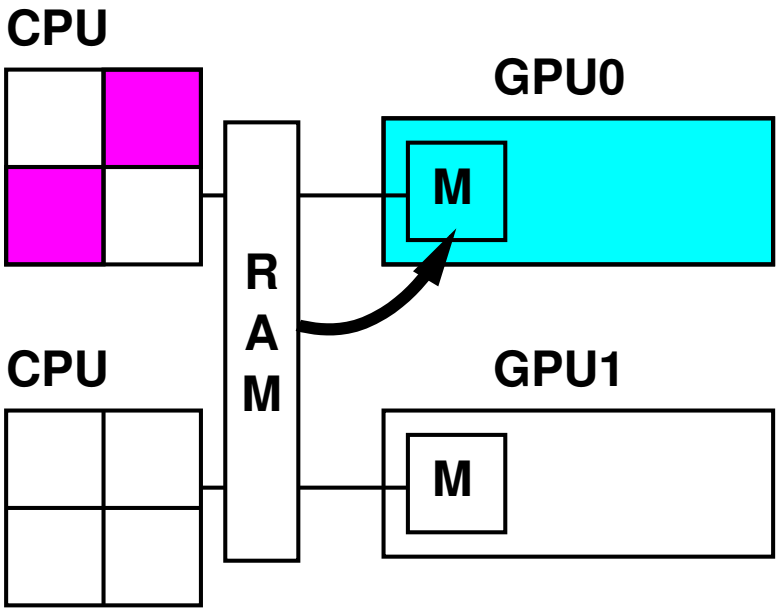


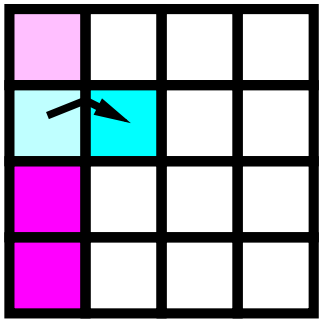


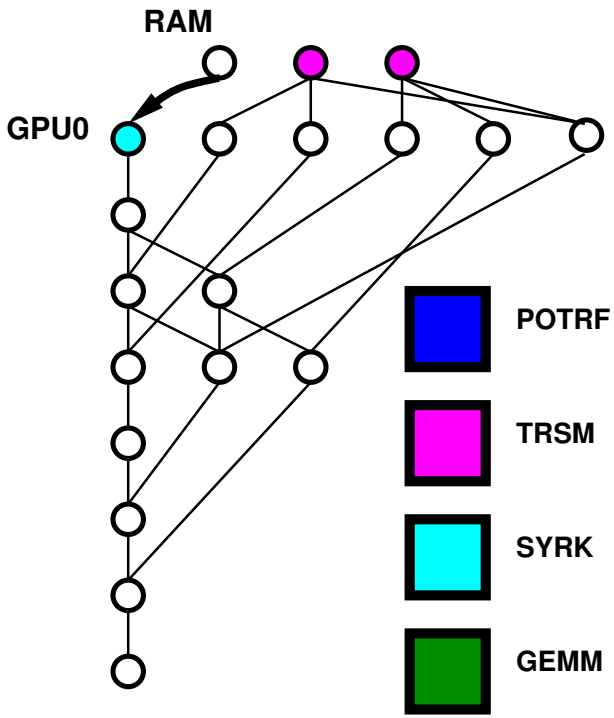
- Handles dependencies
- Handles scheduling

On heterogeneous node

work on tiles → CPU + GPU kernels



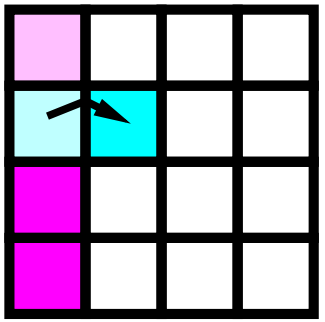




- Handles dependencies
- Handles scheduling
- Handles data consistency

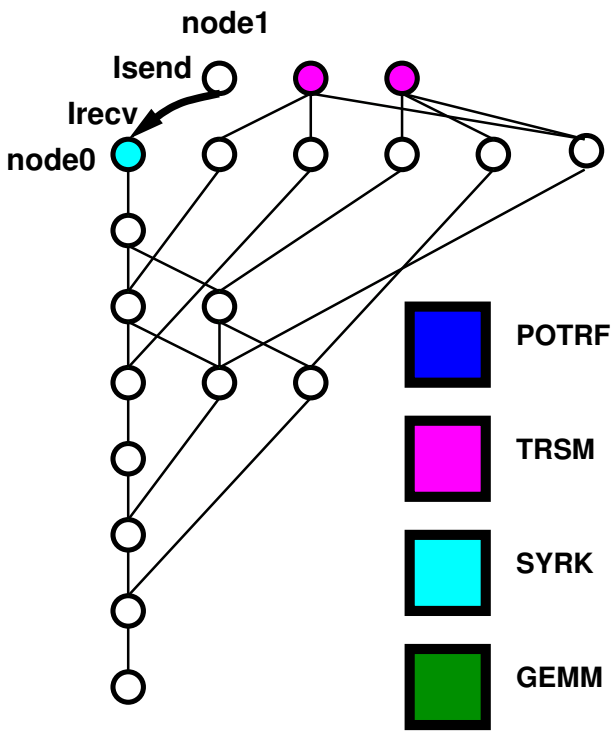
Address scalability

A new programming paradigm for clusters?



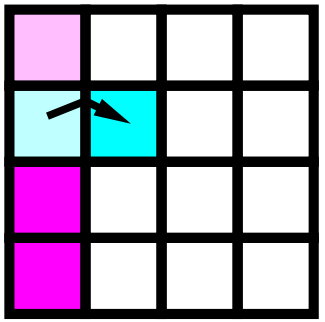
Questions

Existing methods



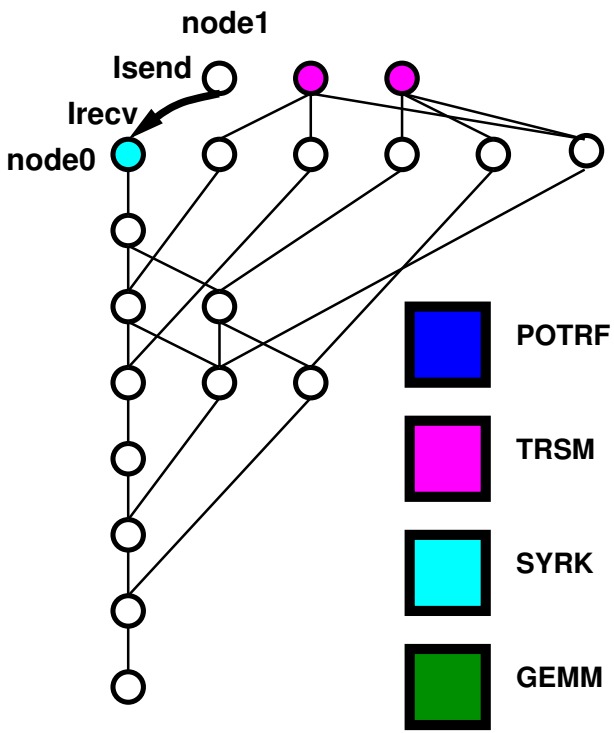
Address scalability

A new programming paradigm for clusters?



Questions

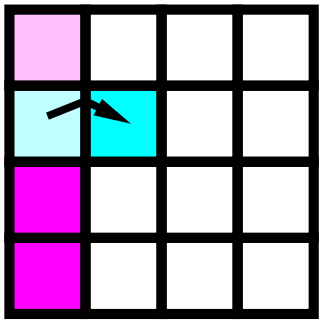
- How to establish the mapping?



Existing methods

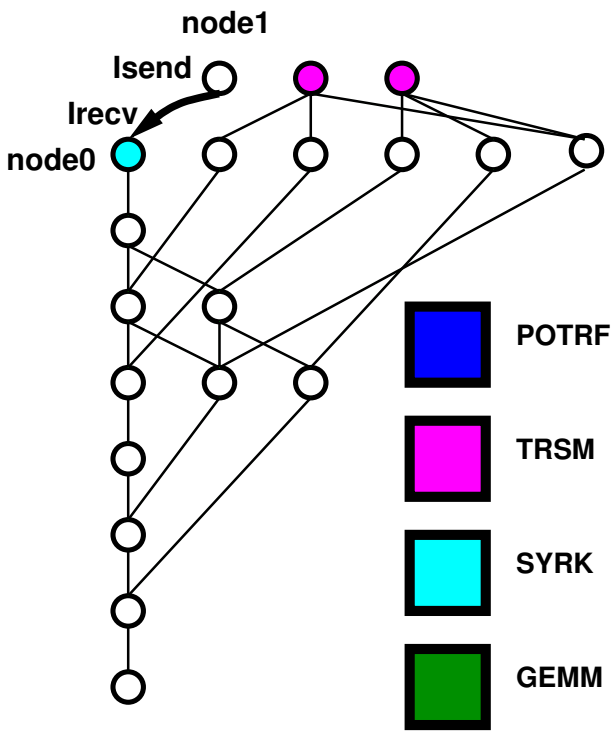
Address scalability

A new programming paradigm for clusters?



Questions

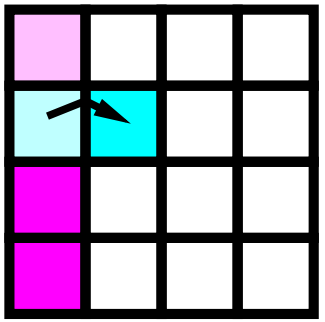
- How to establish the mapping?
- How to manage communications?



Existing methods

Address scalability

A new programming paradigm for clusters?

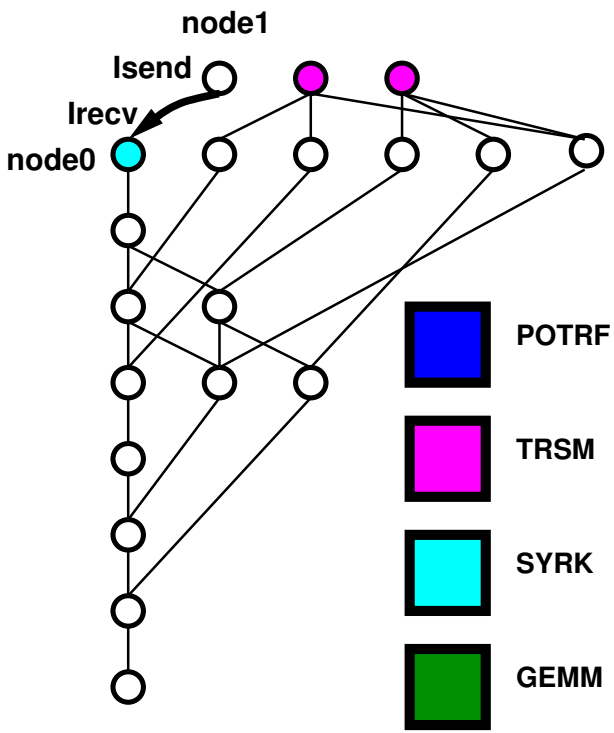


Questions

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

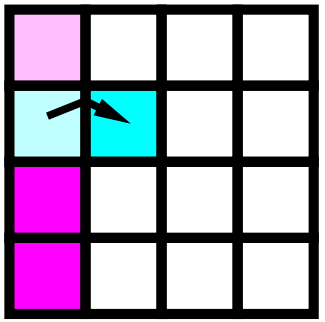
Existing methods

- Explicit MPI communications tasks



Address scalability

A new programming paradigm for clusters?

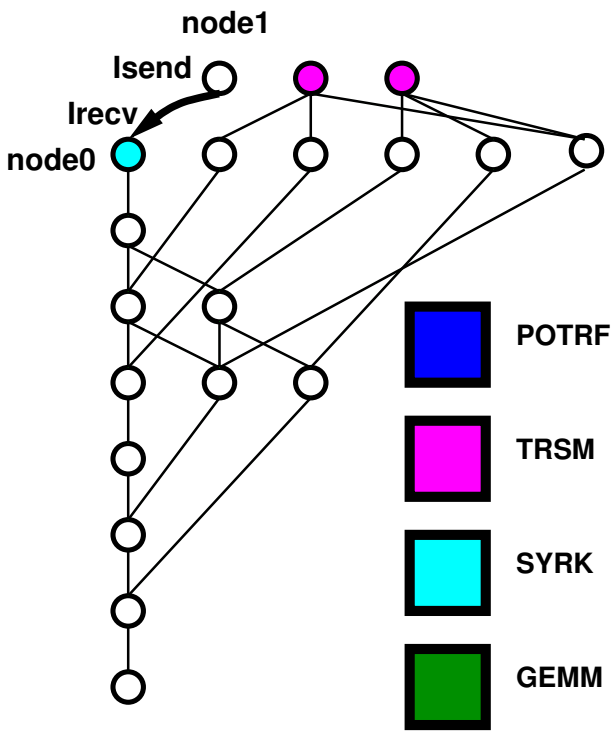


Questions

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

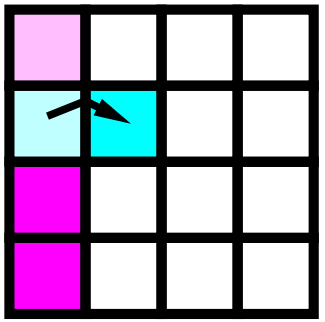
Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)



Address scalability

A new programming paradigm for clusters?

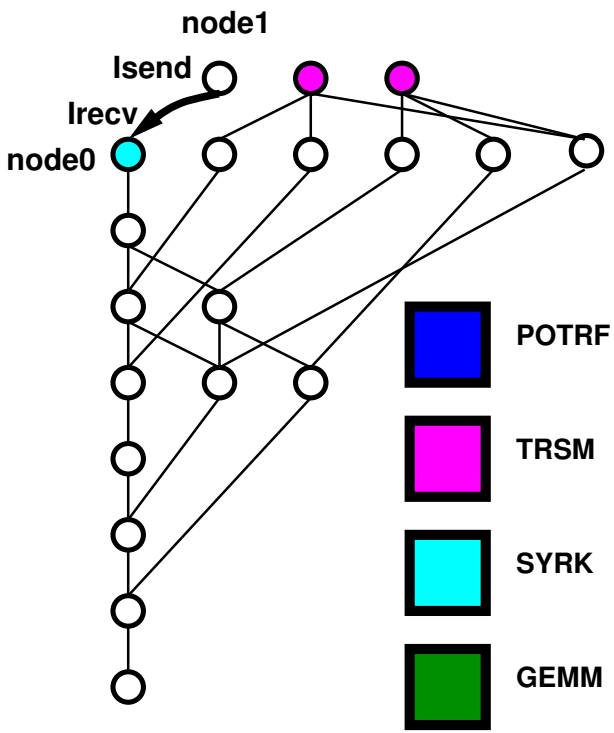


Questions

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

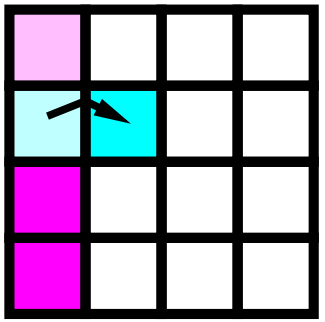
Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)



Address scalability

A new programming paradigm for clusters?

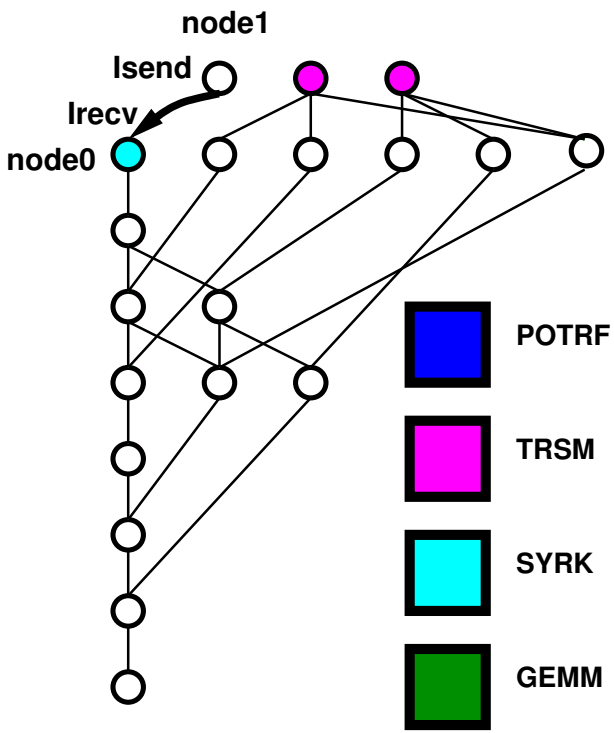


Questions

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

Existing methods

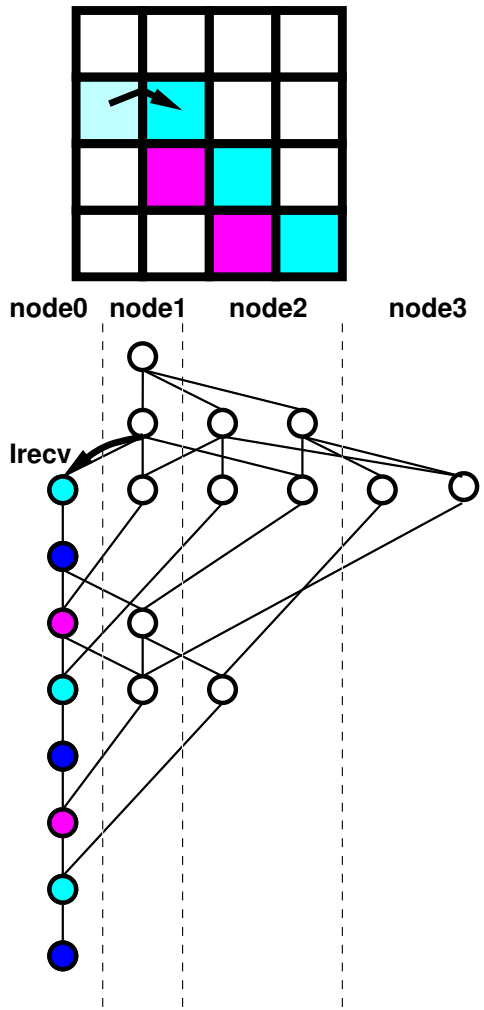
- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)
- STF model - Replicated unrolling (StarPU)



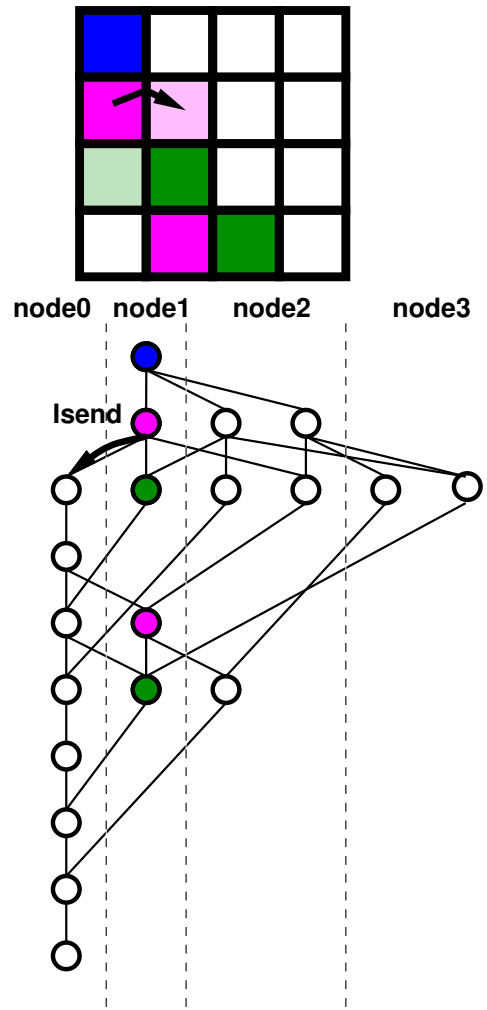
Data transfers between nodes

Method considered

- All nodes unroll the whole task graph
- They determine tasks they will execute
- They can infer required communications
- No negotiation between nodes (not master-slave)
- Unrolling can be pruned



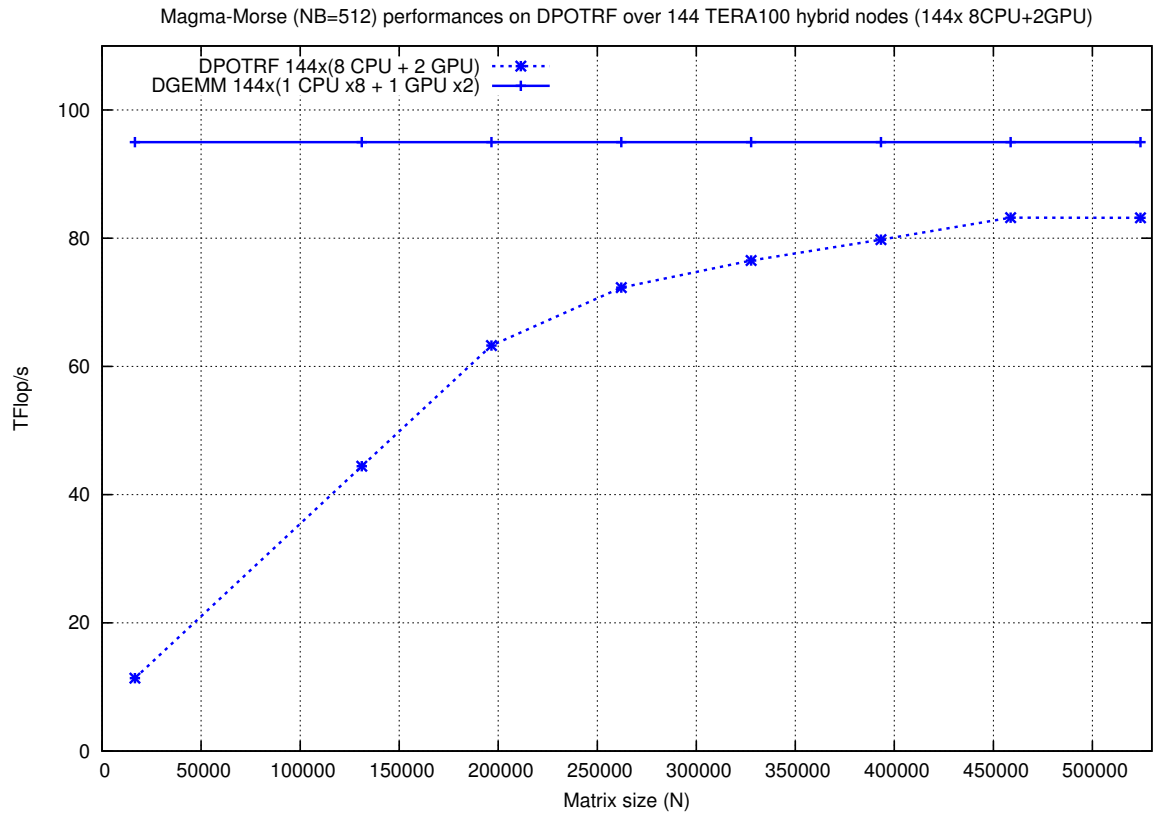
Node 0 execution



Node 1 execution

Performance

- 144 TERA-100 Hybrid nodes
  - collaboration with CEA-CESTA
- CPU: 2 Quad-core Xeon E5620
- GPU: 2 NVIDIA Tesla M2090

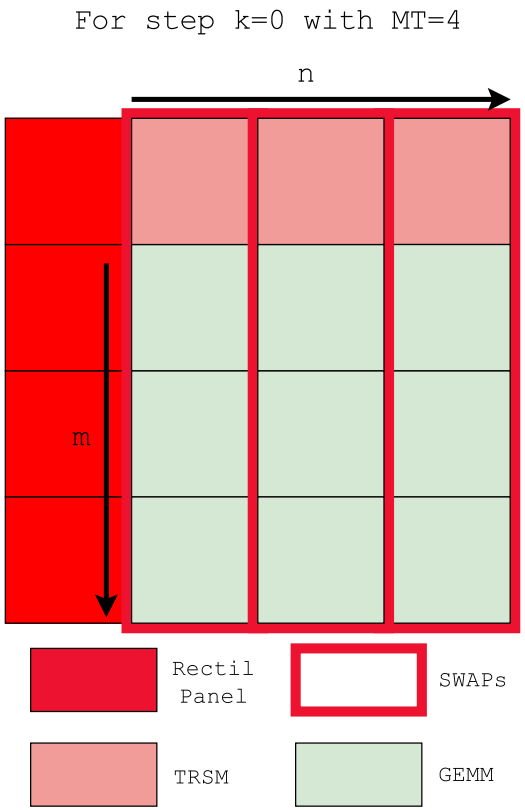


Outline

Tile LU factorization with numerical Partial Pivoting

PhDs A. Hugo and T. Cojean

```
1   //If A is a square matrix
2   for k : 0 → MT-1
3
      RECTIL_PANEL(submatrix(A_{k,k}, ..., A_{MT,k}))
4     //Waits for the pivots
      computed by this panel
5     data_acquire(pivs[k])
6
7     for n : k+1 → MT-1
8       //SWAP if required
      according to pivs[k] array
9       SWAPs(submatrix(A_{k,n}, ..., A_{MT,n}))
10      TRSM(A_{k,k}, A_{k,n})
11      for m : k+1 → MT-1
12      GEMM(A_{m,k}, A_{k,n}, A_{m,n})
13
14  for k : 0 → MT-1
15    for n : 0 → k
16      SWAPs(submatrix(A_{k,n}, ..., A_{MT,n}))
17
```



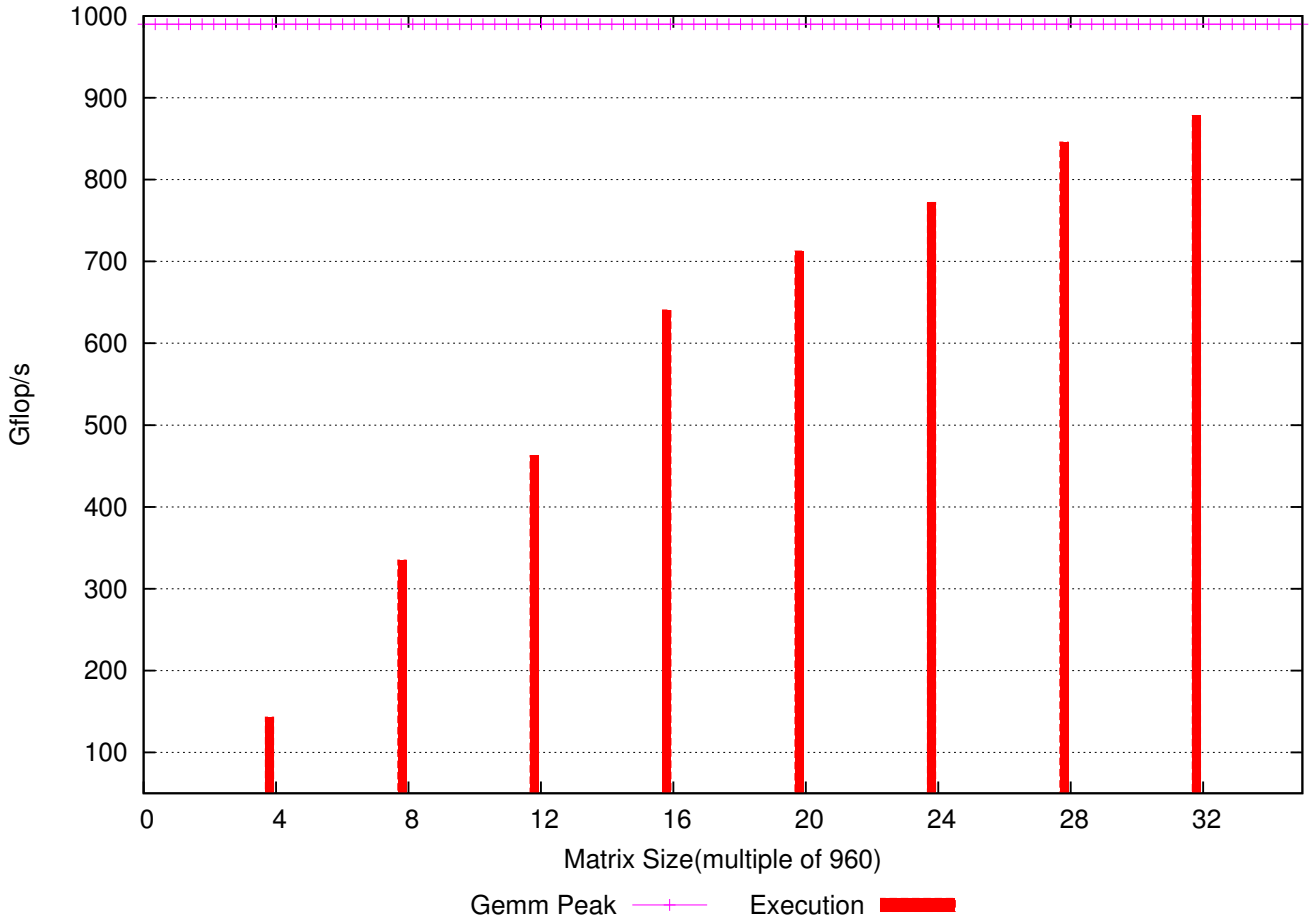
GETRF Partial Pivoting with rectil panel using

$MT = 4$

Outline

Improving Cholesky through theoretical analysis

PhD. S. Kumar (in collaboration with Combinatoire & Algorithmique)



Improving Cholesky through theoretical analysis

PhD. S. Kumar (in collaboration with Combinatoire & Algorithmique)