Vector operations, tiled operations, distributed execution, task graphs, ...

What next ?

Samuel Thibault, University of Bordeaux
Inria STORM Team

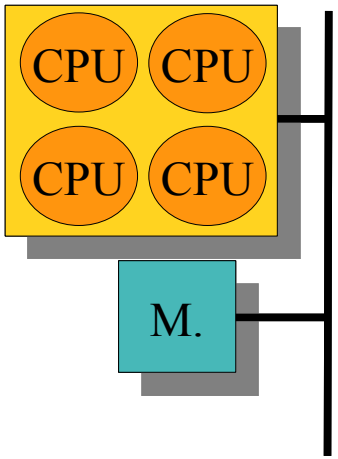
High-Performance Computing

- Frontier supercomputer
- #1 on top500.org LINPACK benchmark
- 1.102 EFlop/s
- 9472 nodes
  - 1 AMD Epyc CPU
  - 4 AMD Instinct 250X GPUs
  - 4TB flash memory
  - Infinity Fabric interconnect
- Slingshot 100GB/s network
- 21.1 MW
- Also #1 Green 500

High-Performance Computing

Classical parallel programming

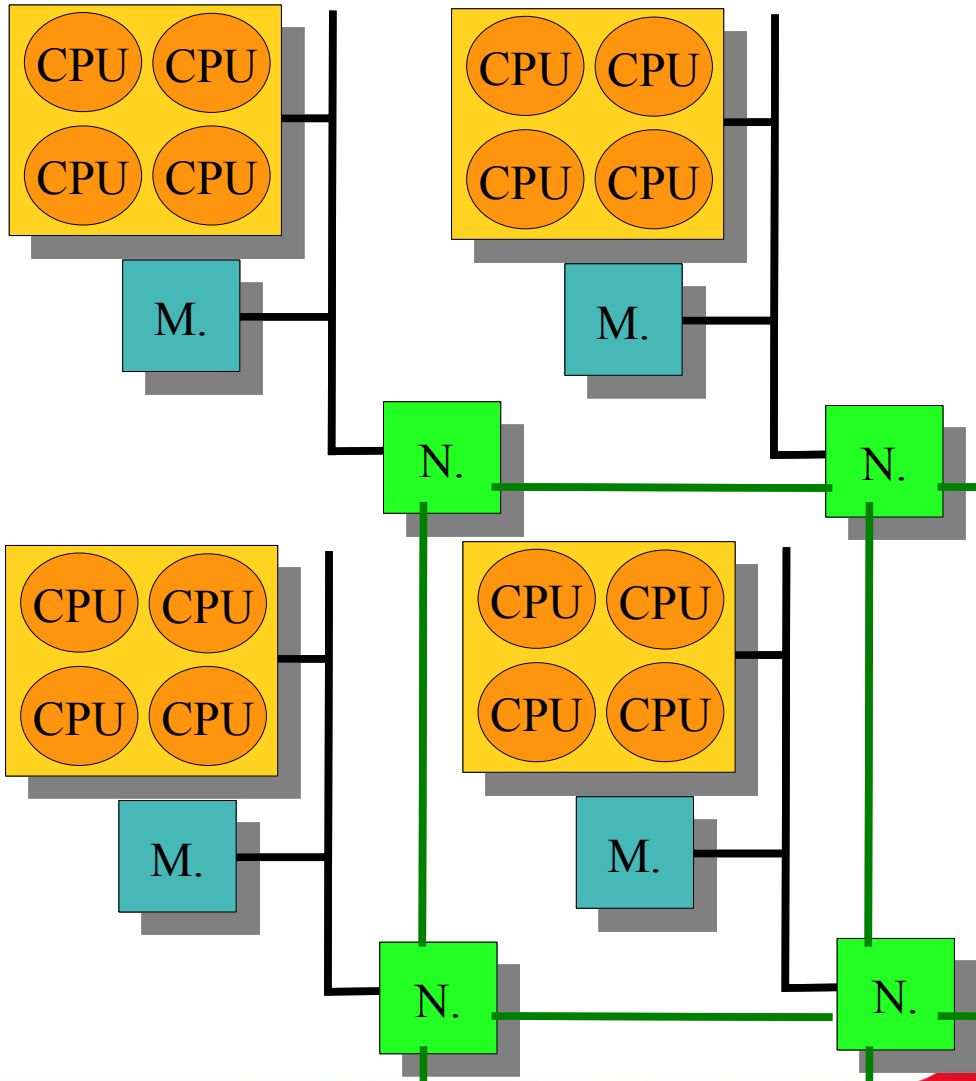
- threads



High-Performance Computing

Classical parallel programming

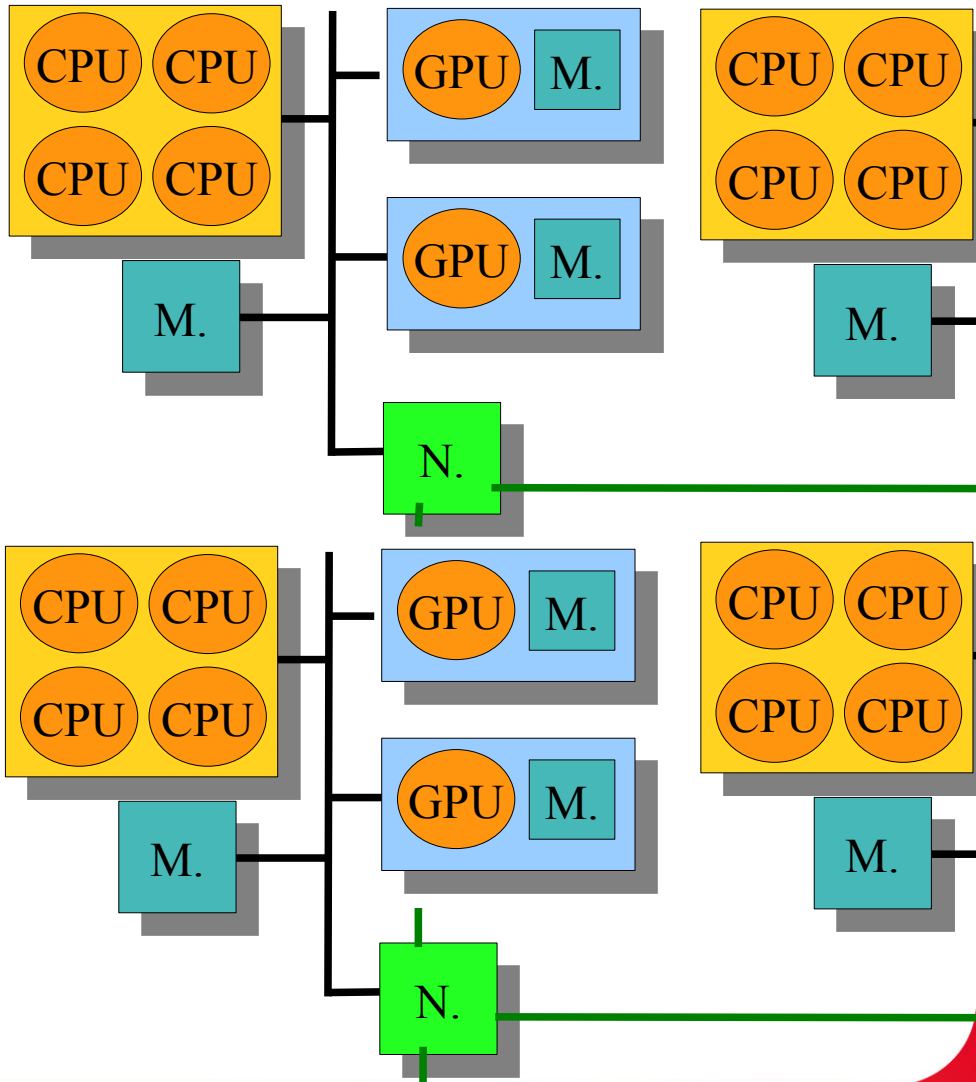
- threads
- MPI+threads



High-Performance Computing

Classical parallel programming

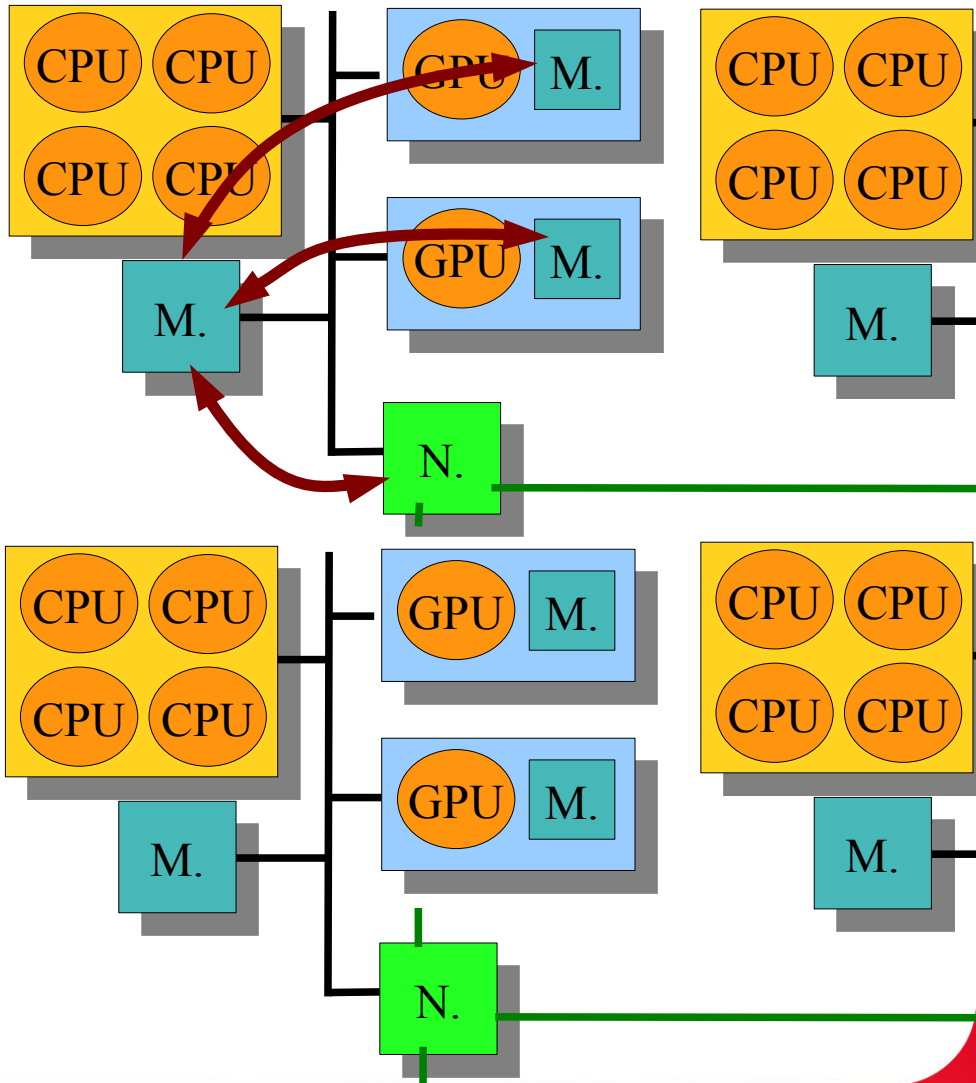
- threads
- MPI+threads
- MPI+threads+GPU



High-Performance Computing

Classical parallel programming

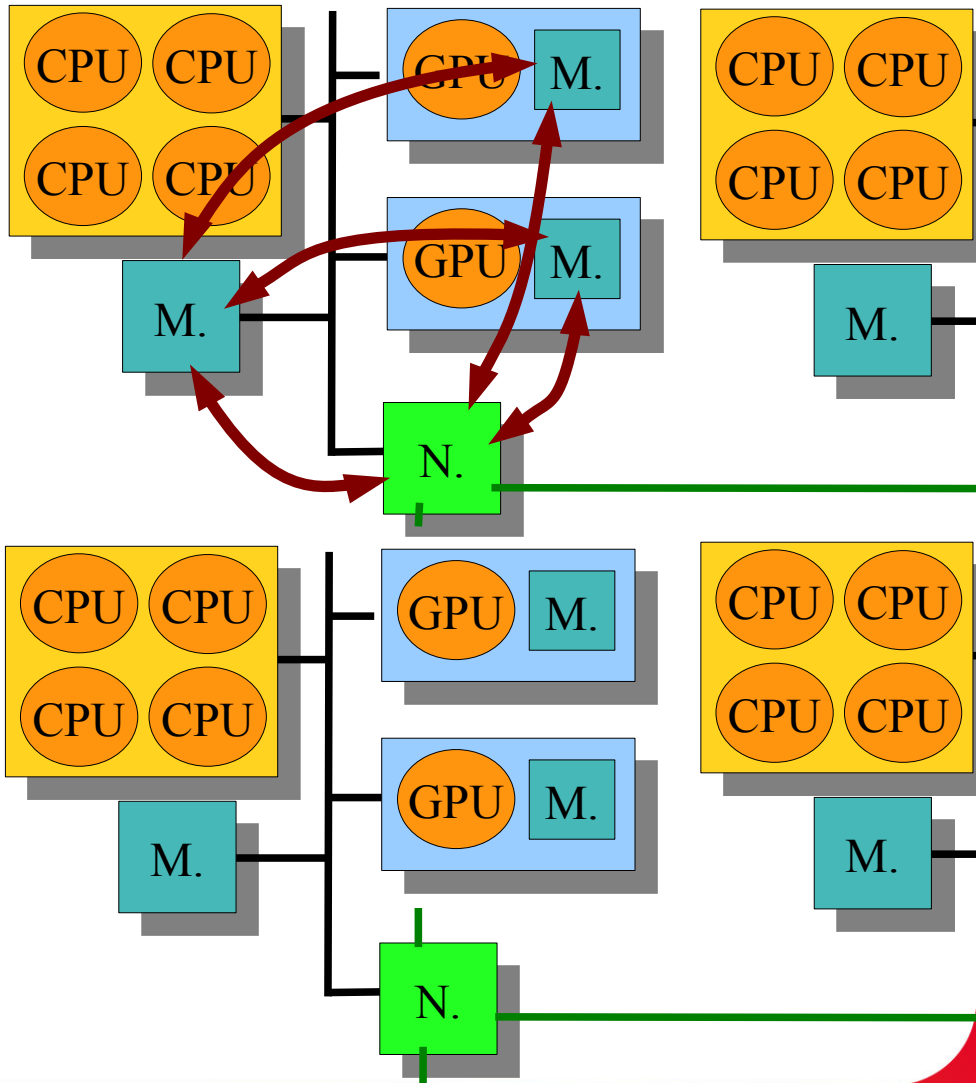
- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers



High-Performance Computing

Classical parallel programming

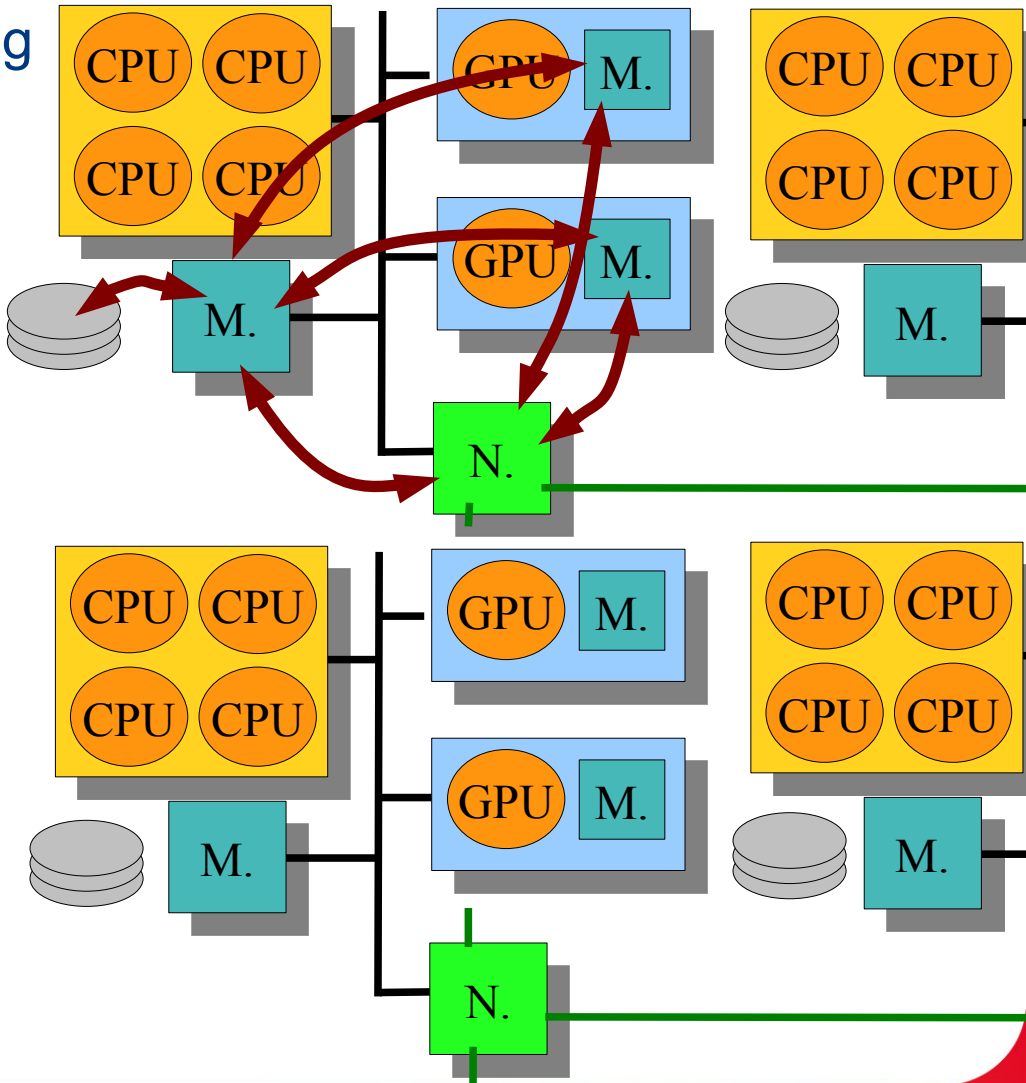
- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers



High-Performance Computing

Classical parallel programming

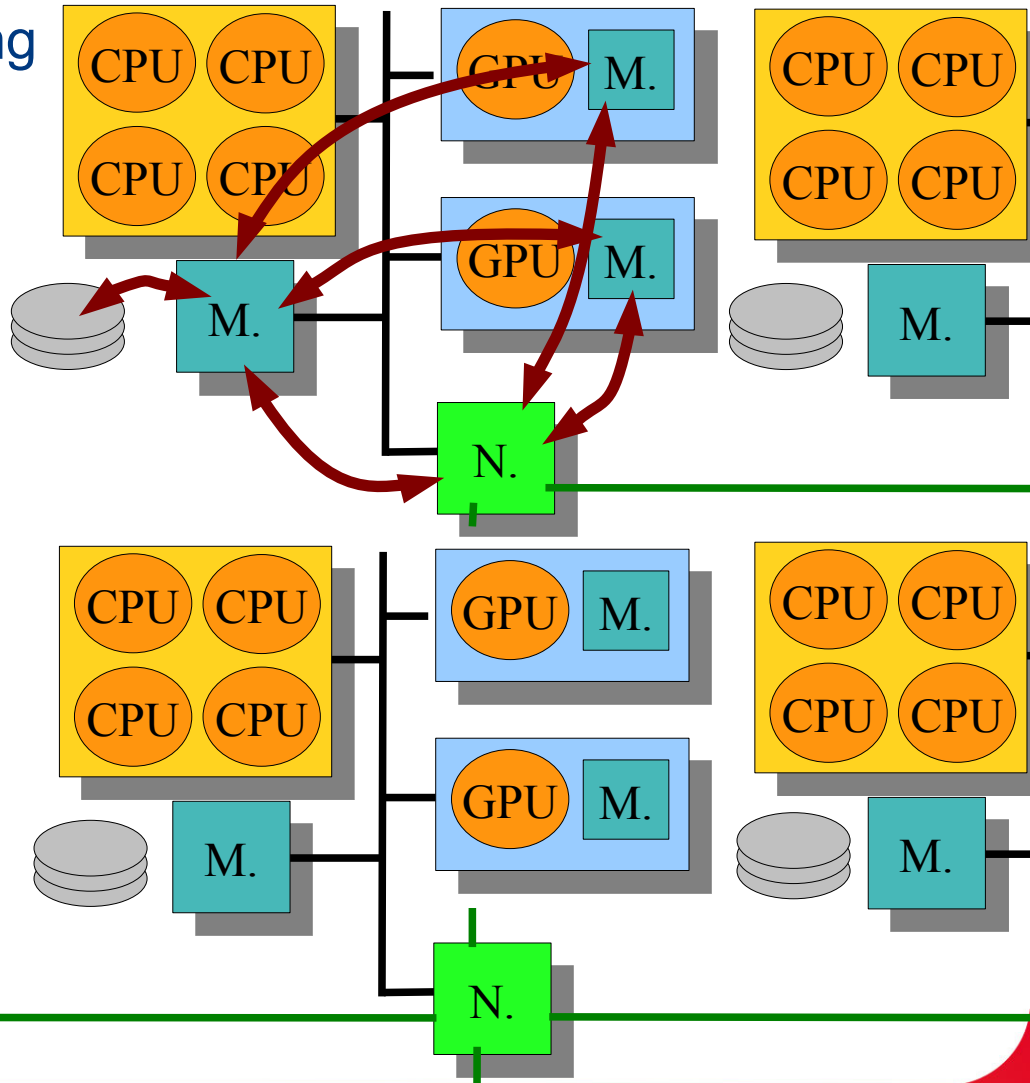
- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers
- Disk storage



High-Performance Computing

Classical parallel programming

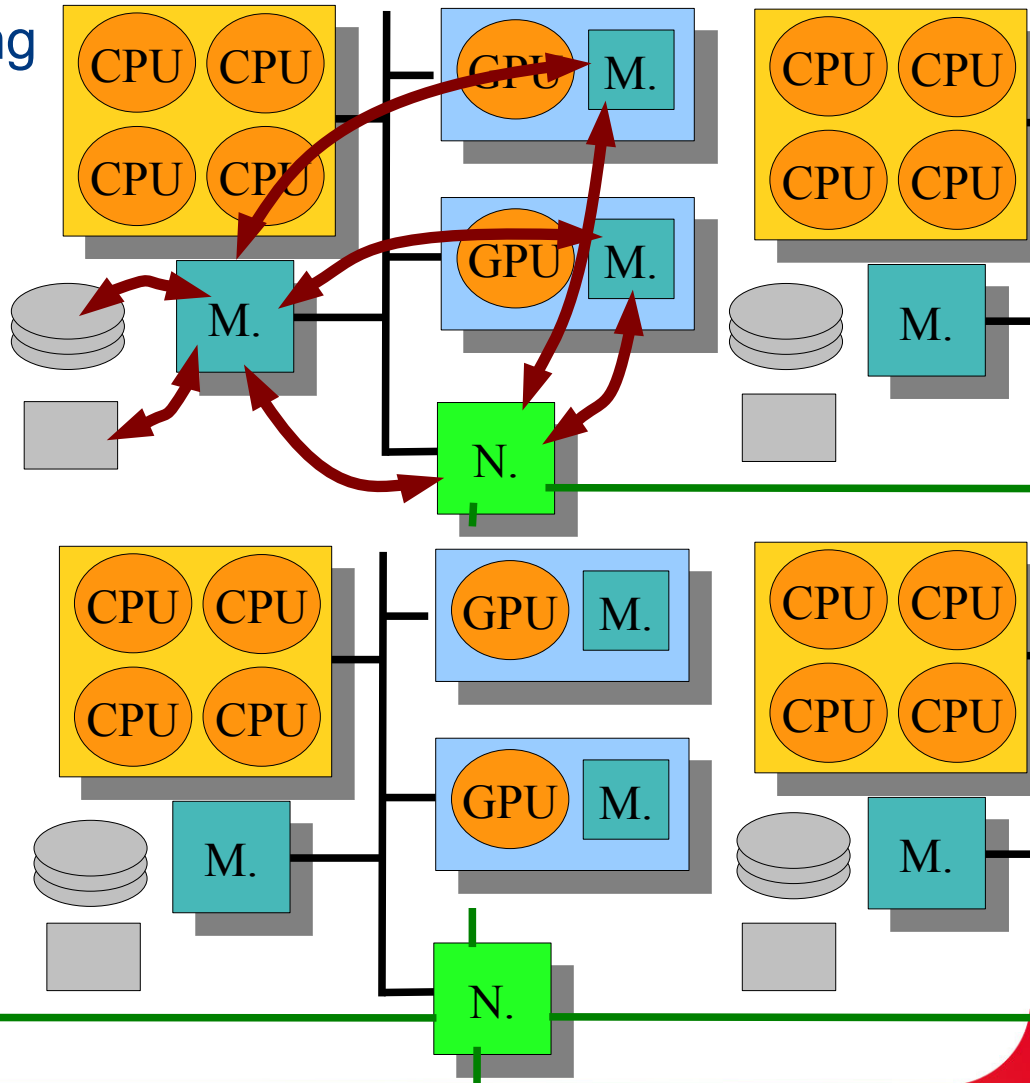
- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers
- Disk storage



High-Performance Computing

Classical parallel programming

- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers
- Disk storage

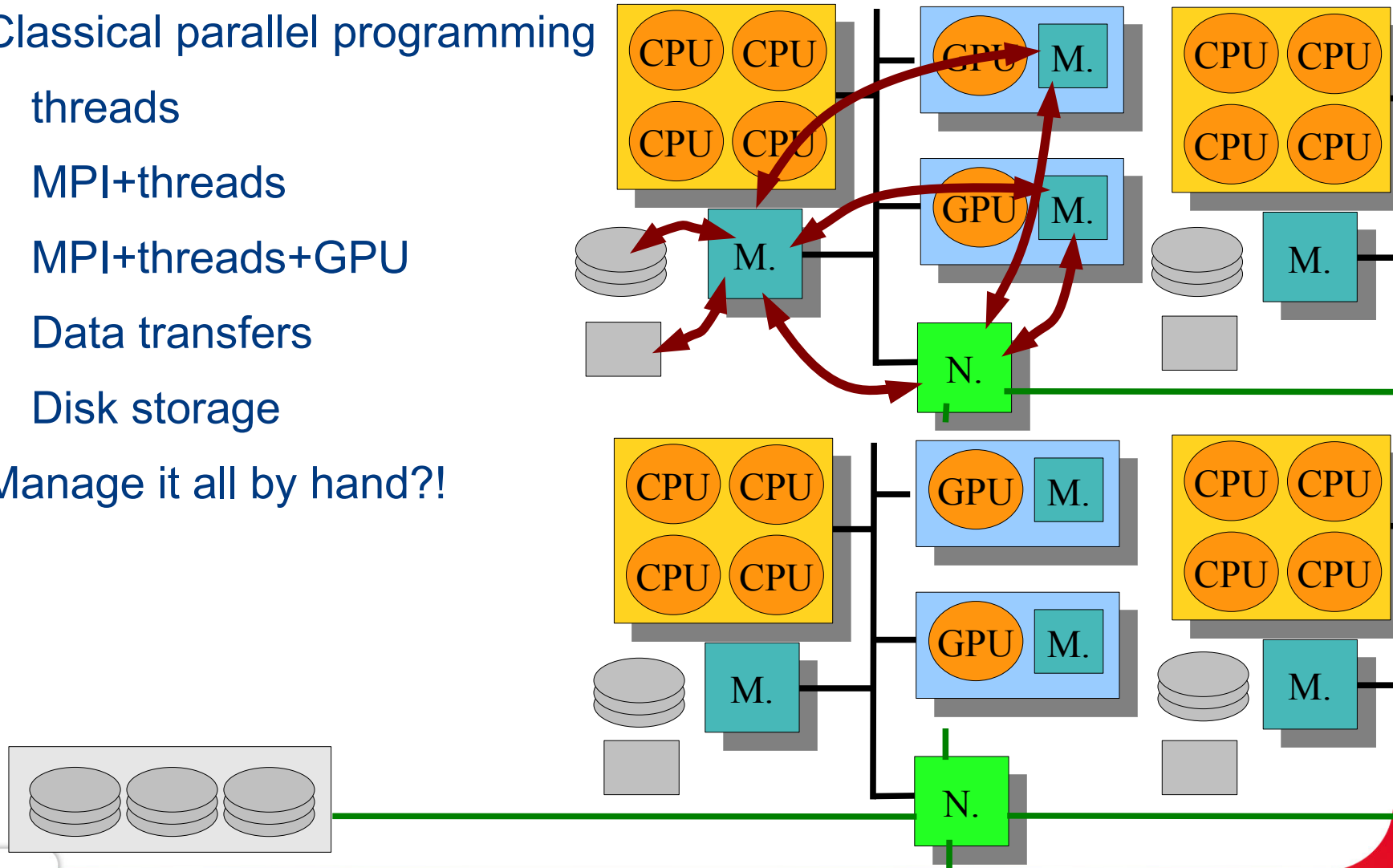


High-Performance Computing

Classical parallel programming

- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers
- Disk storage

Manage it all by hand?!



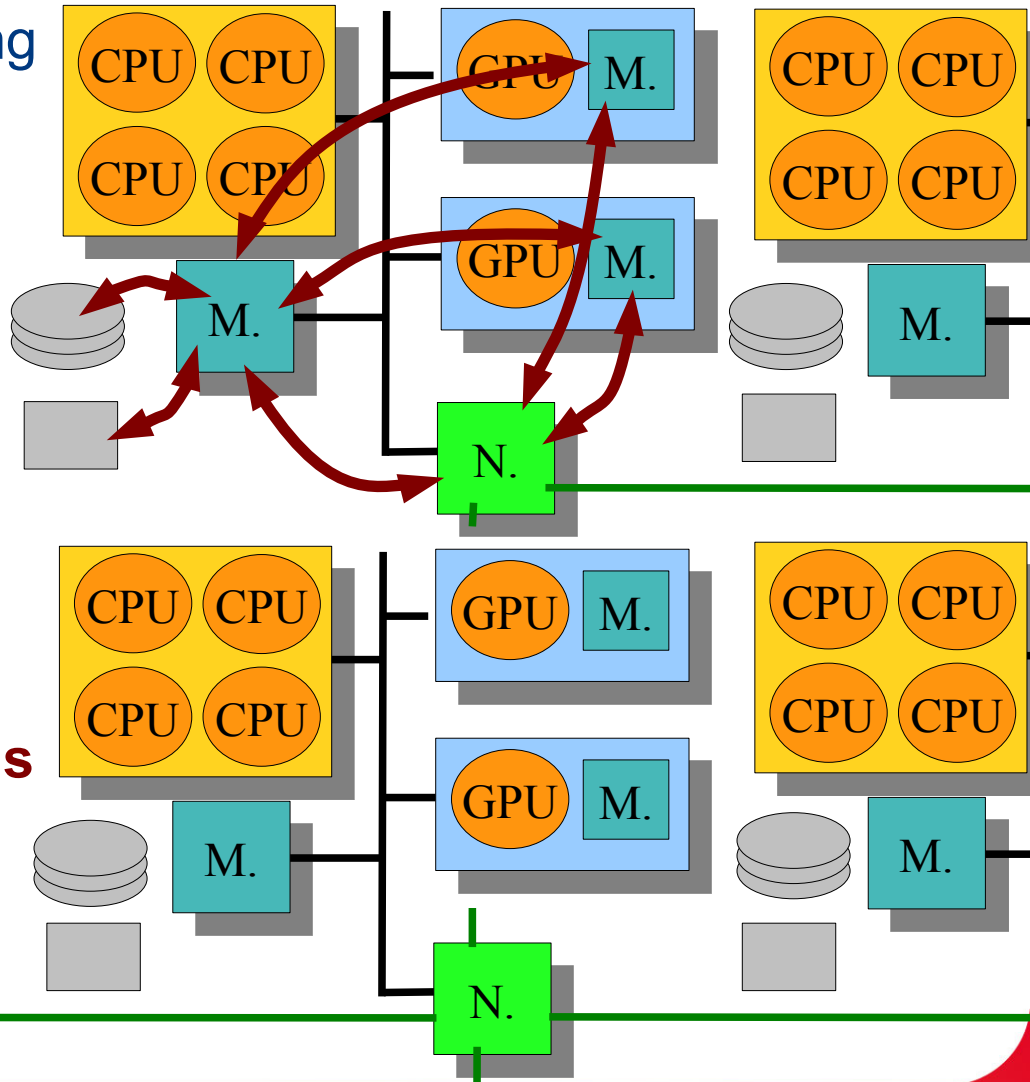
High-Performance Computing

Classical parallel programming

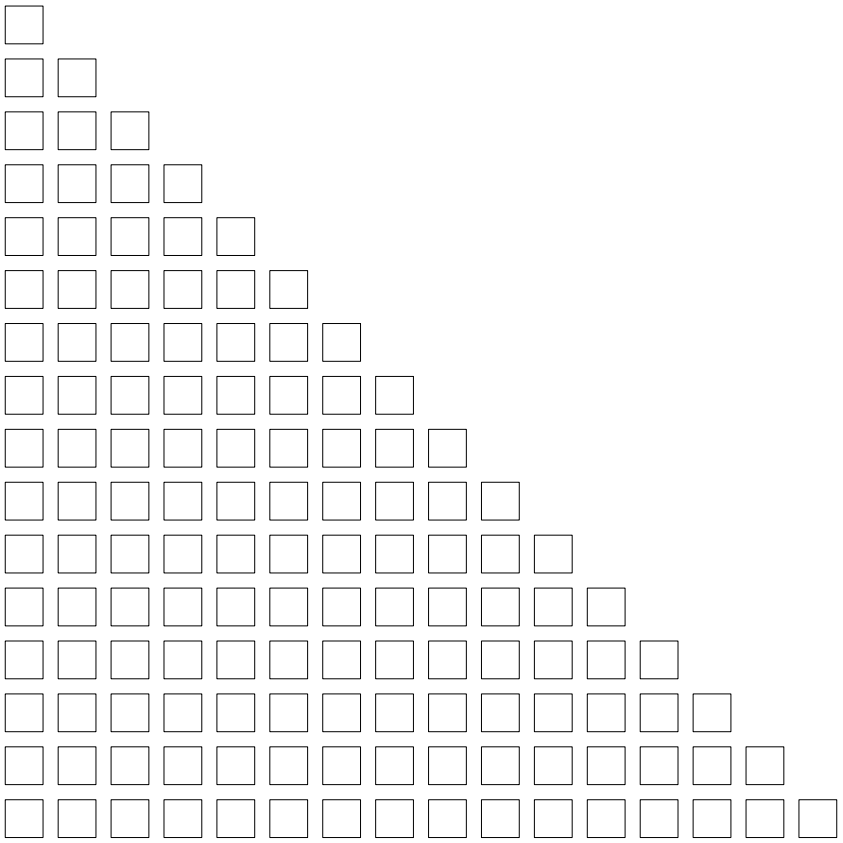
- threads
- MPI+threads
- MPI+threads+GPU
- Data transfers
- Disk storage

Manage it all by hand?!

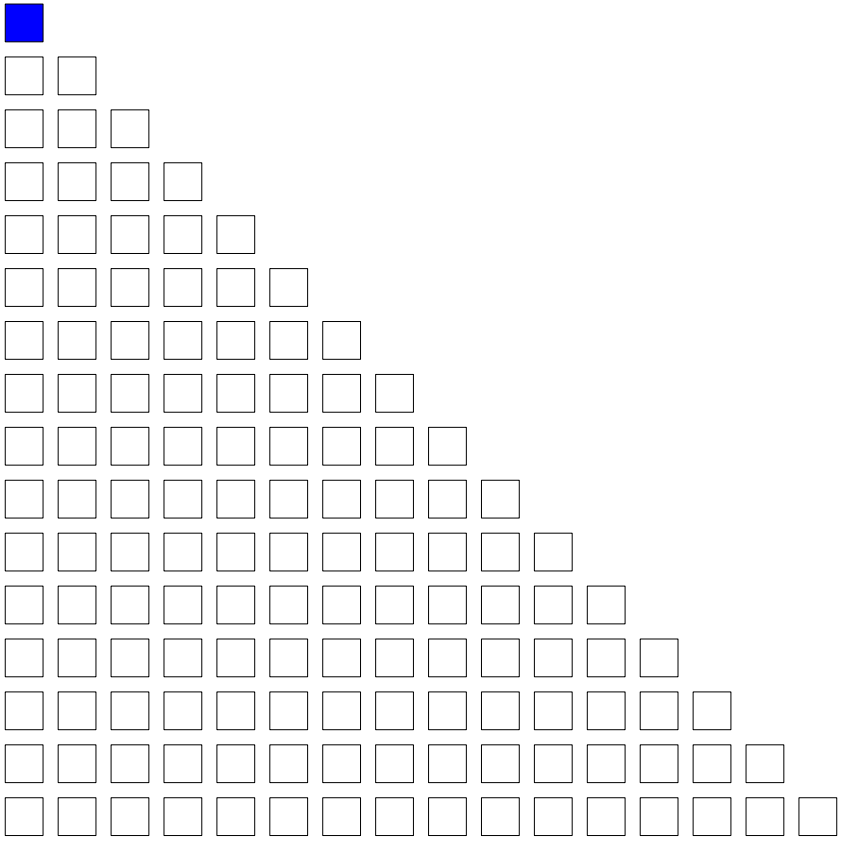
Need high-level abstractions



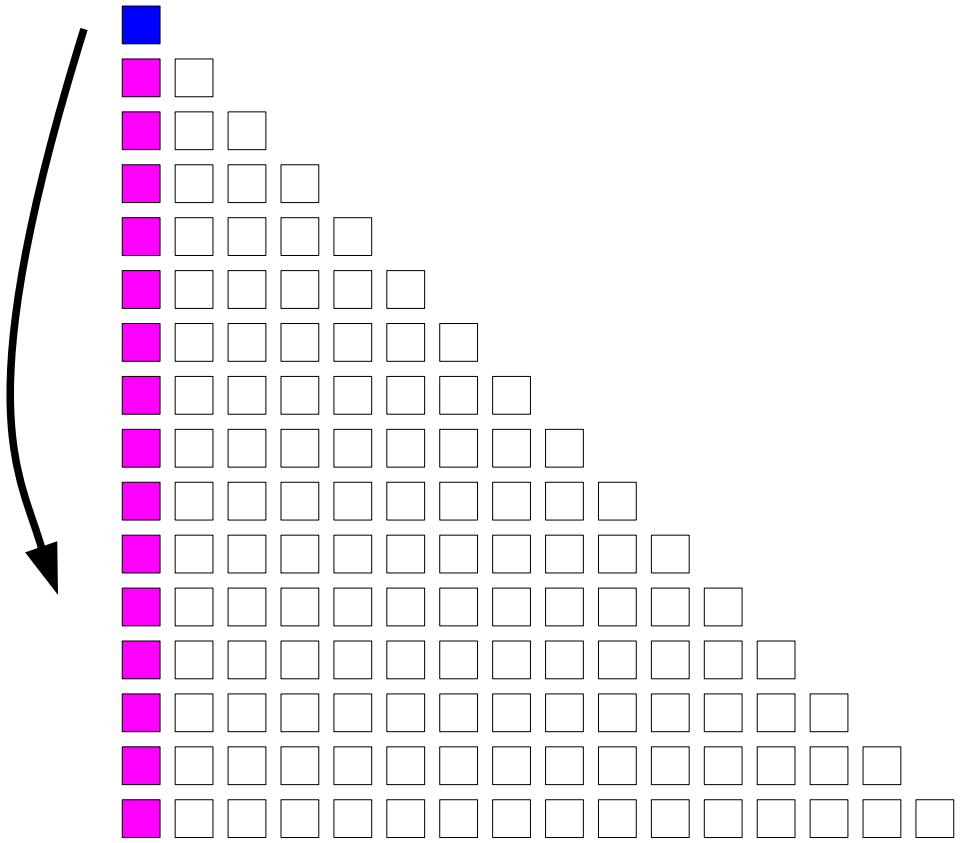
Cholesky ($A = L.L^T$)



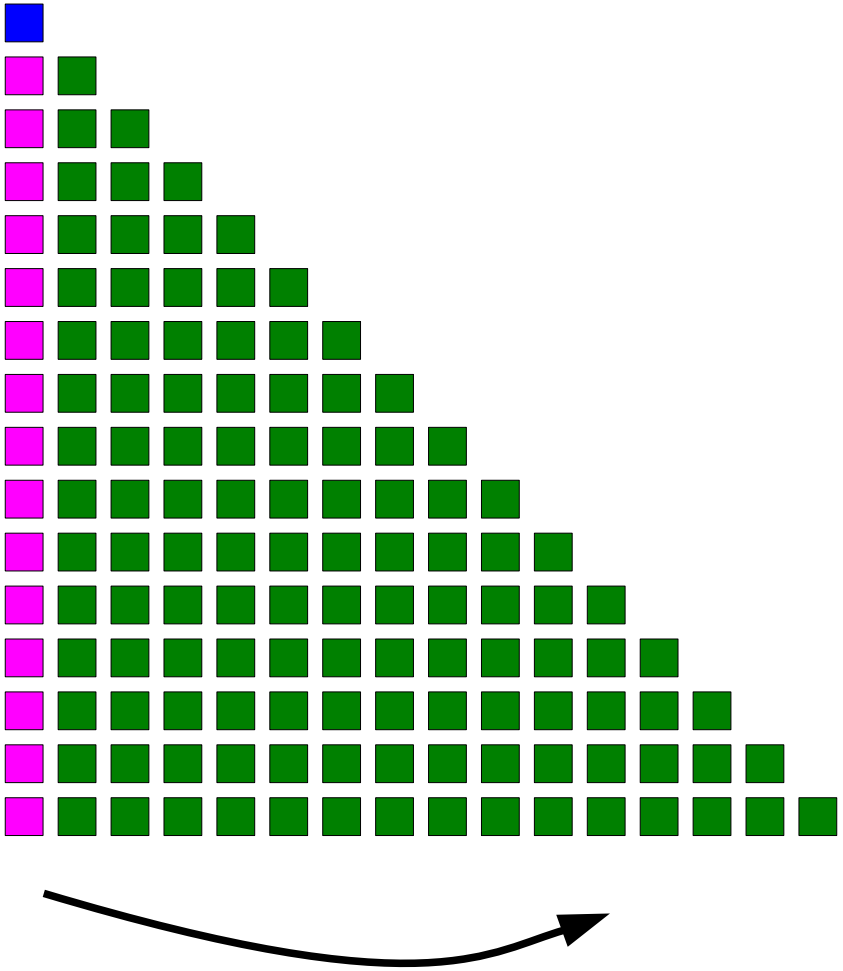
Cholesky ($A = L.L^T$)



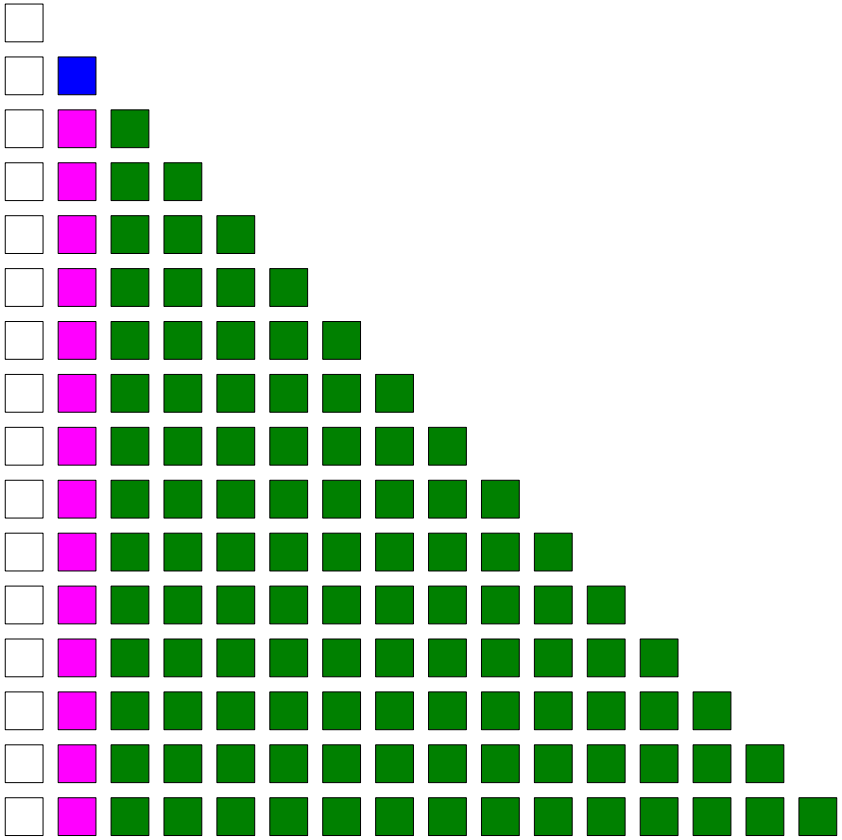
Cholesky ($A = L.L^T$)



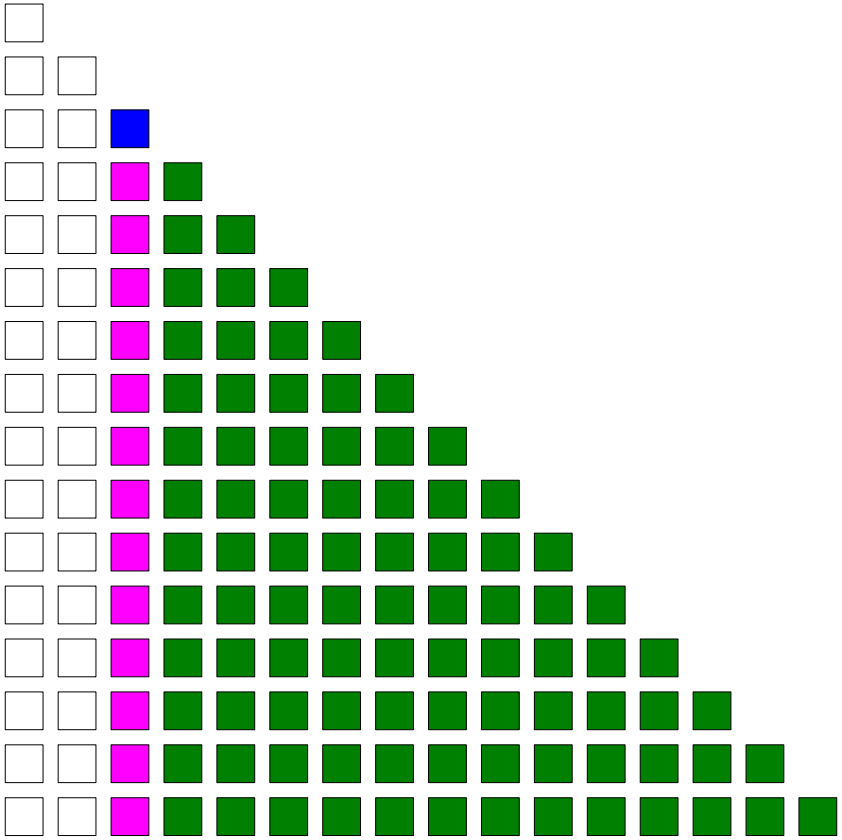
Cholesky ($A = L.L^T$)



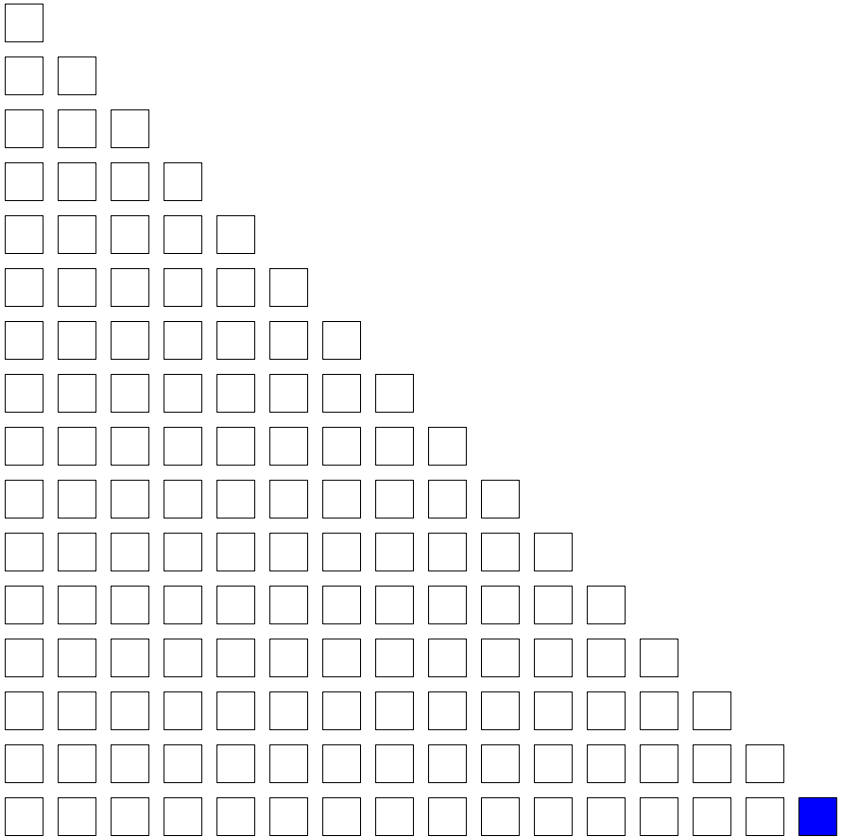
Cholesky ($A = L.L^T$)



Cholesky ($A = L.L^T$)

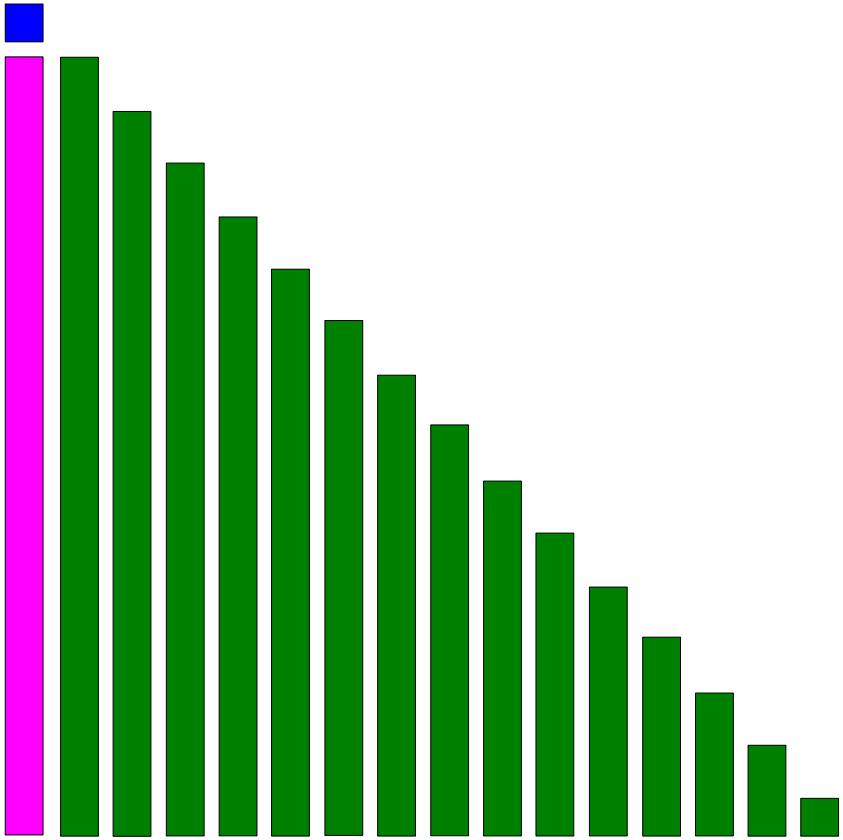


Cholesky ($A = L.L^T$)



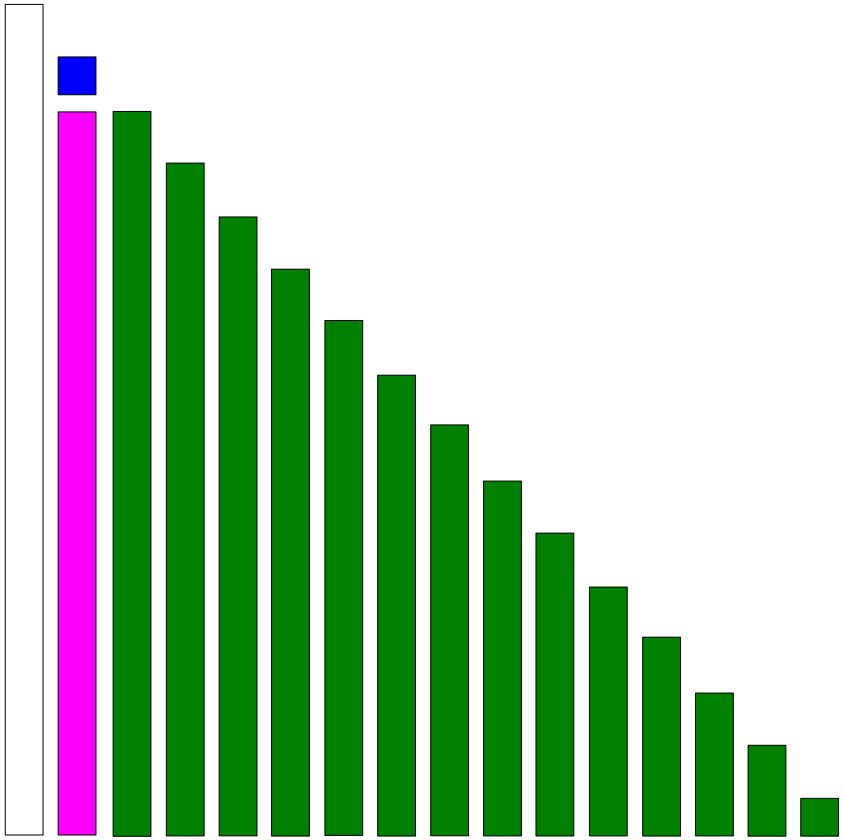
Cholesky, LINPACK

Vector computers (end ‘70s, ‘80s)



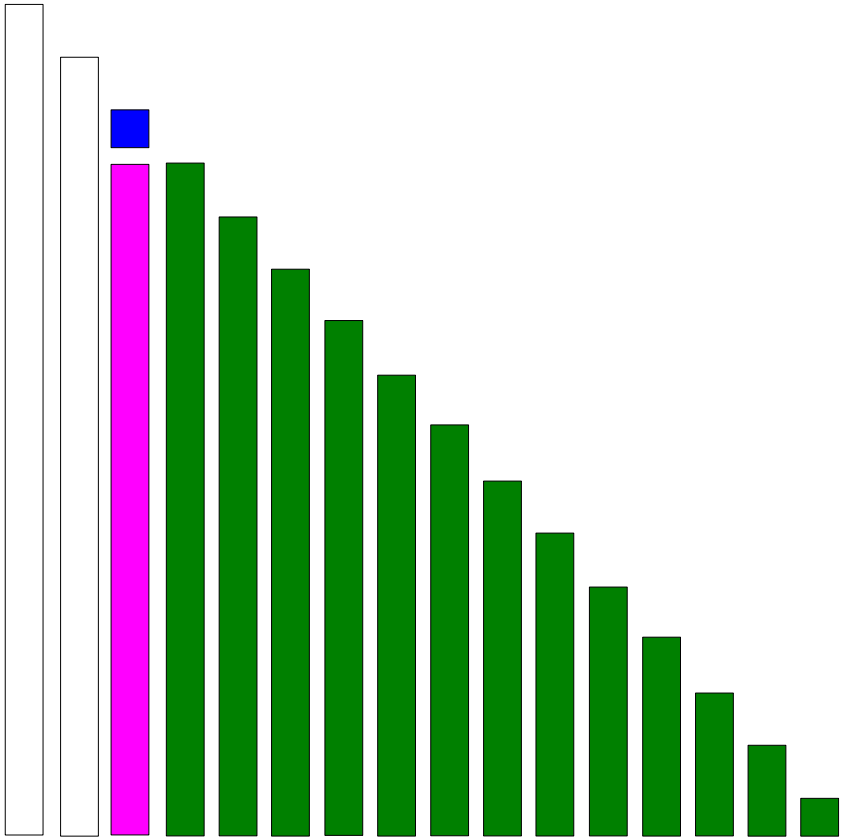
Cholesky, LINPACK

Vector computers (end ‘70s, ‘80s)



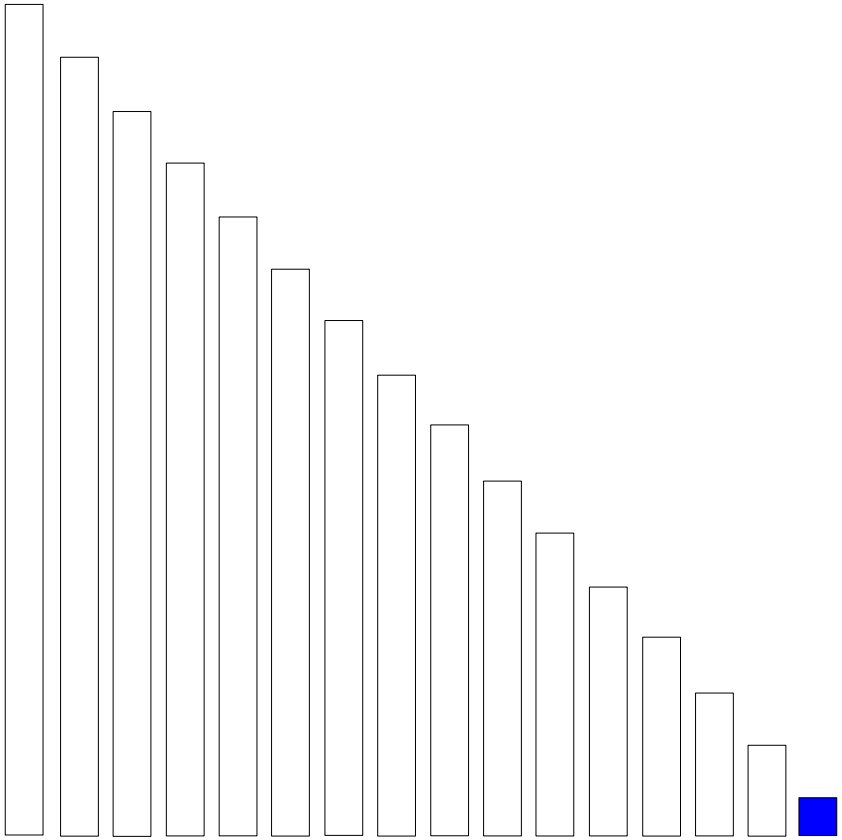
Cholesky, LINPACK

Vector computers (end ‘70s, ‘80s)



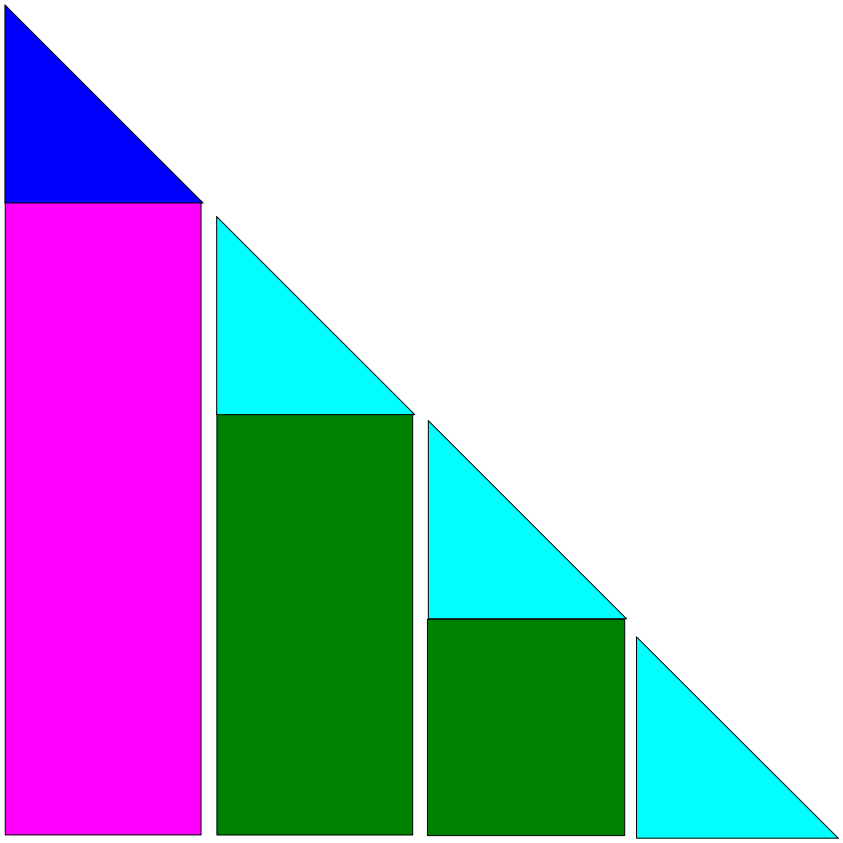
Cholesky, LINPACK

Vector computers (end ‘70s, ‘80s)



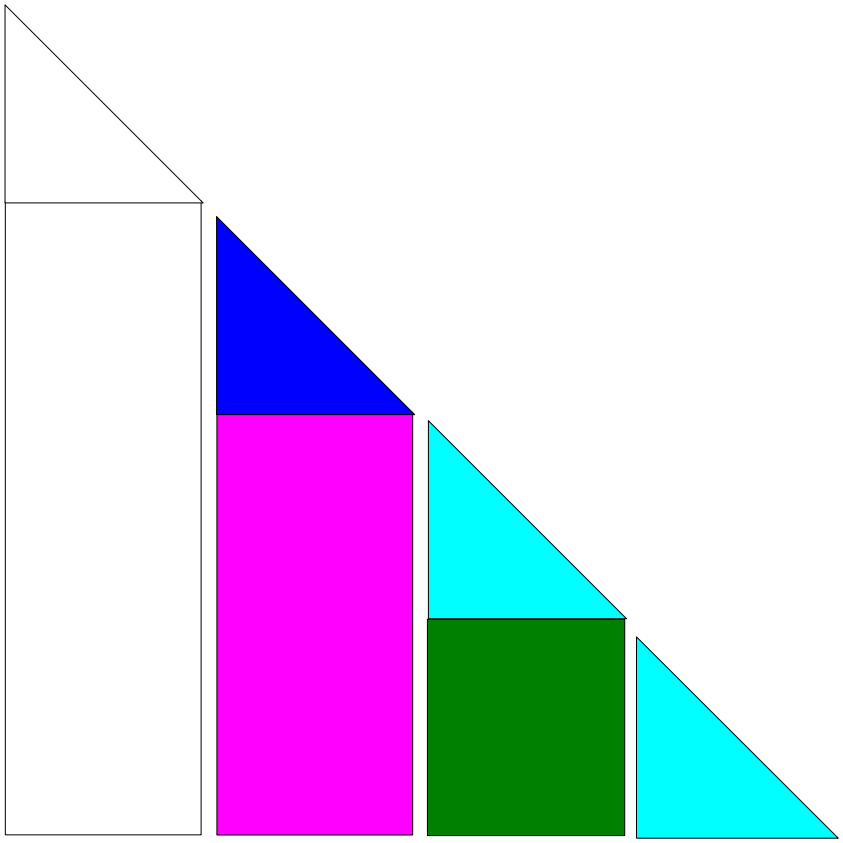
Cholesky, LAPACK

Computers with cache (early ‘90s-today), blocked operations : BLAS



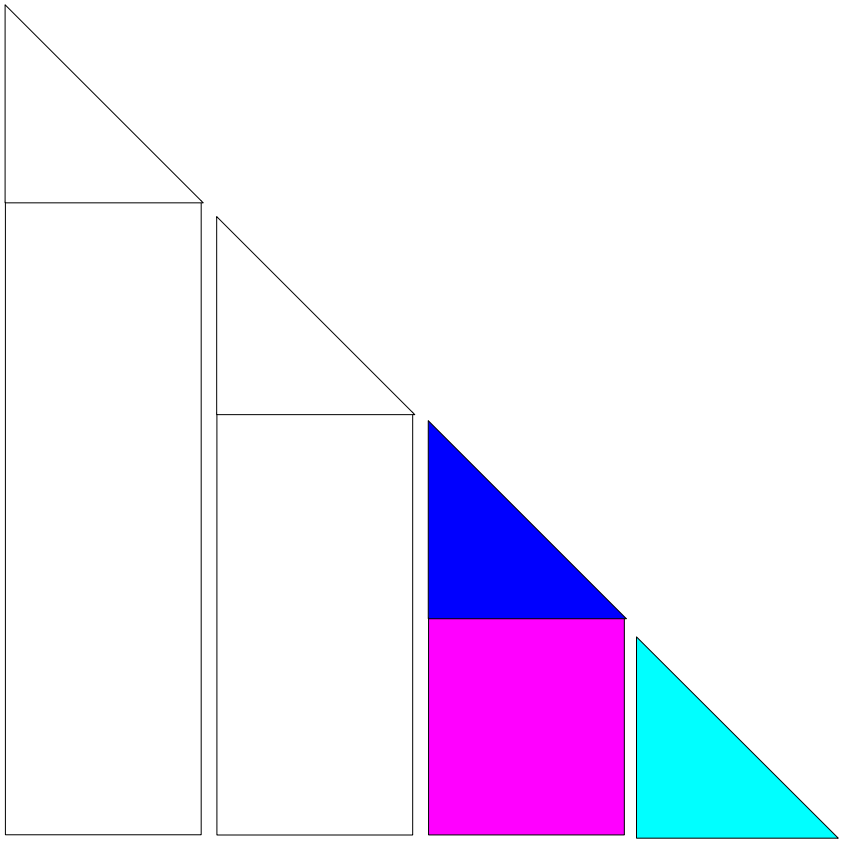
Cholesky, LAPACK

Computers with cache (early ‘90s-today), blocked operations : BLAS



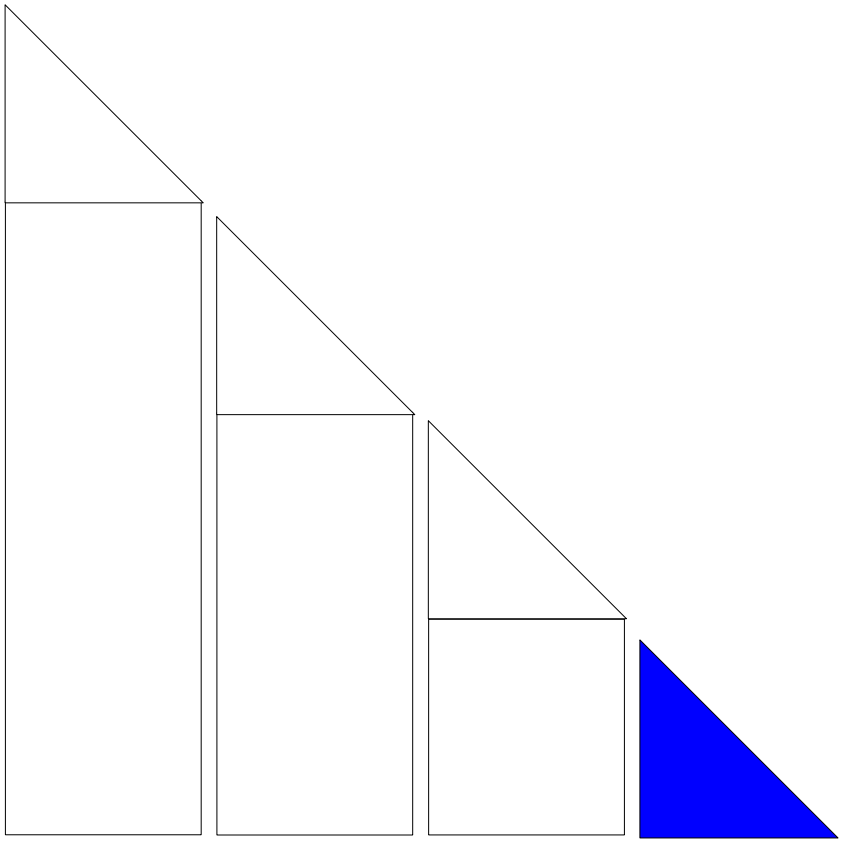
Cholesky, LAPACK

Computers with cache (early ‘90s-today), blocked operations : BLAS



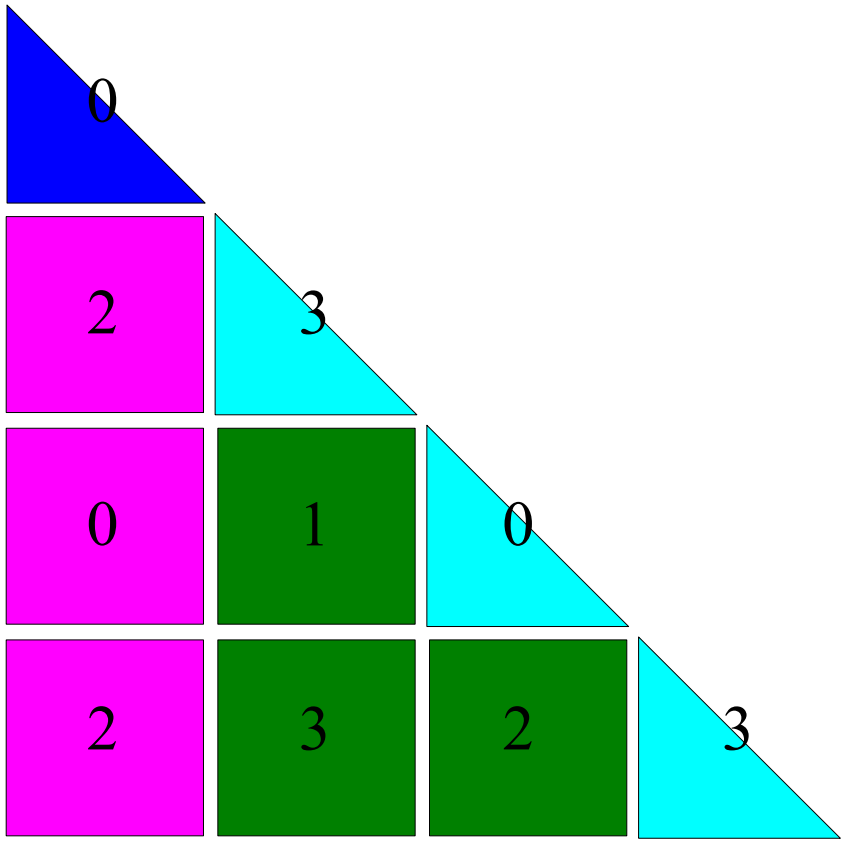
Cholesky, LAPACK

Computers with cache (early ‘90s-today), blocked operations : BLAS



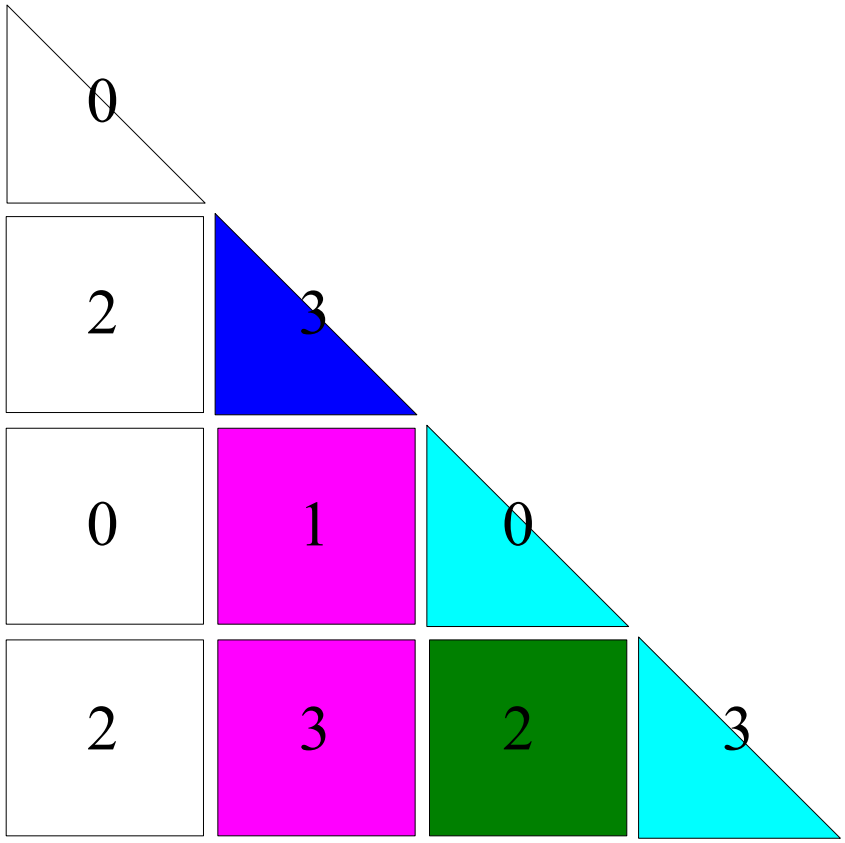
Cholesky, ScaLAPACK

Distributed Computers (late ‘90s-today), 2D-block-cyclic distribution



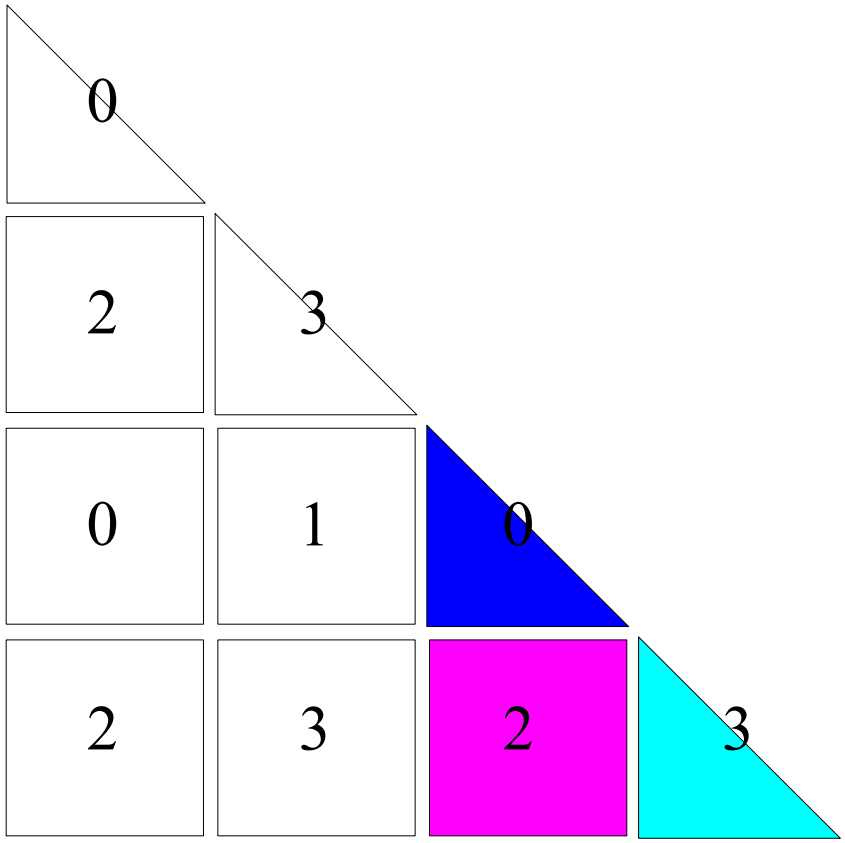
Cholesky, ScaLAPACK

Distributed Computers (late ‘90s-today), 2D-block-cyclic distribution



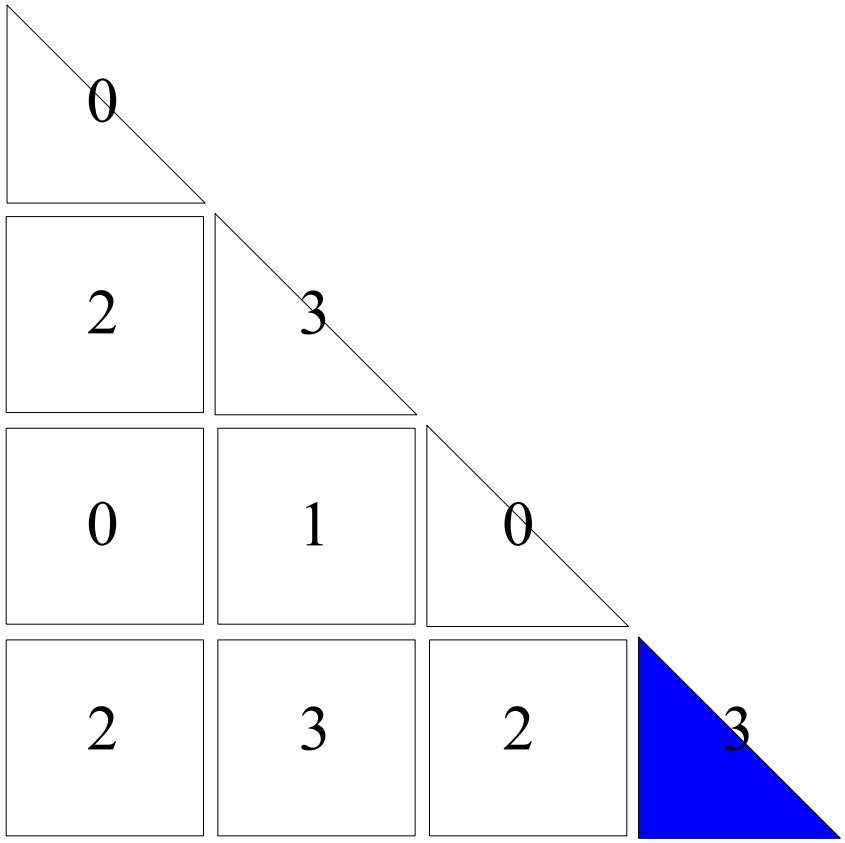
Cholesky, ScaLAPACK

Distributed Computers (late ‘90s-today), 2D-block-cyclic distribution



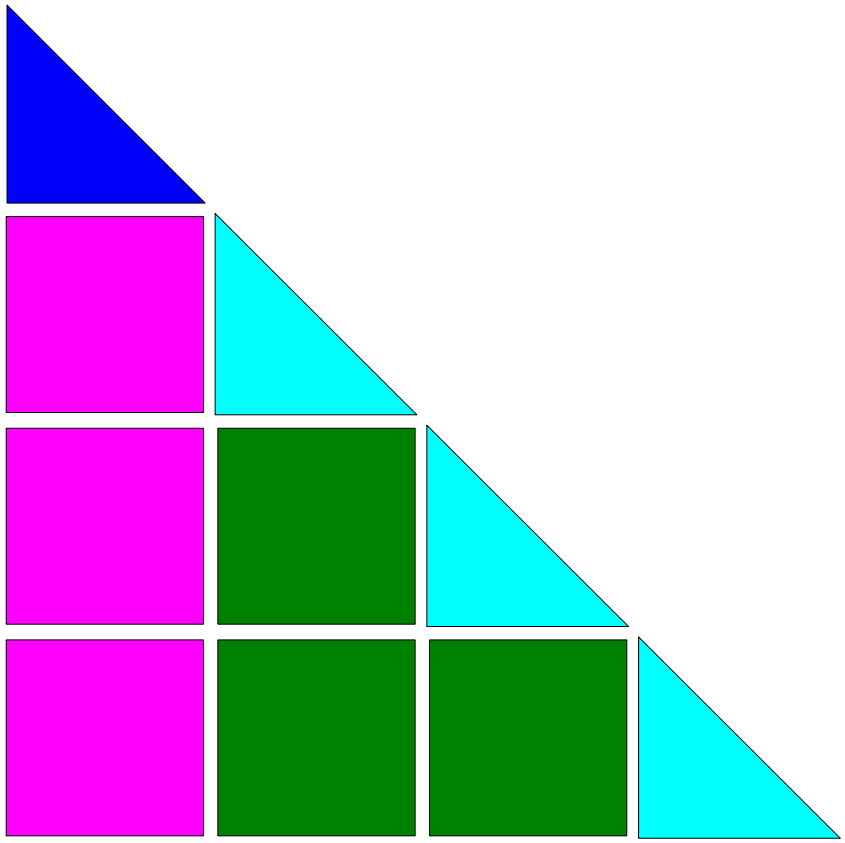
Cholesky, ScaLAPACK

Distributed Computers (late ‘90s-today), 2D-block-cyclic distribution



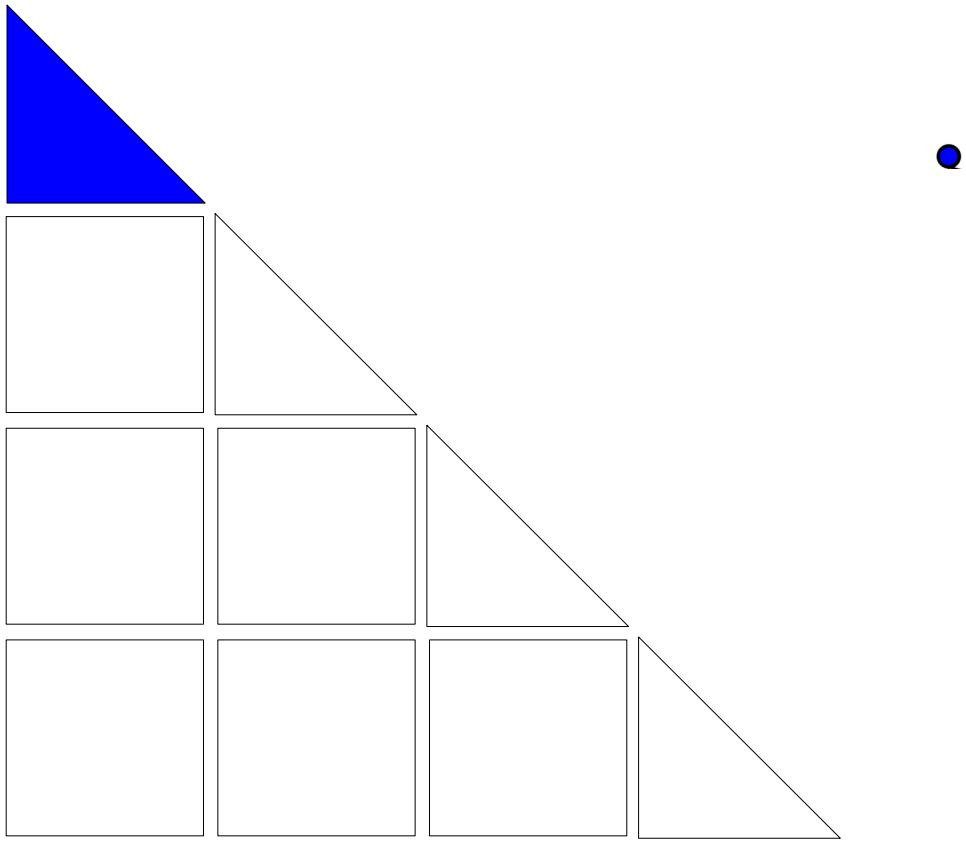
Cholesky, PLASMA

Task graph (~'08-today)



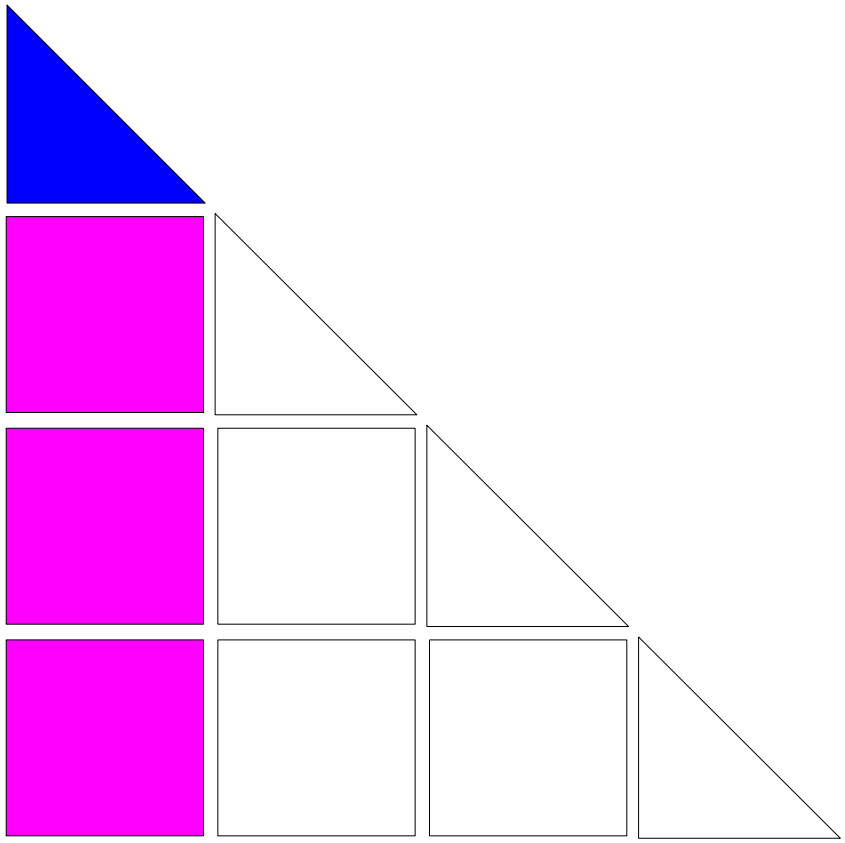
Cholesky, PLASMA

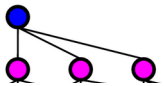
Task graph (~'08-today)



Cholesky, PLASMA

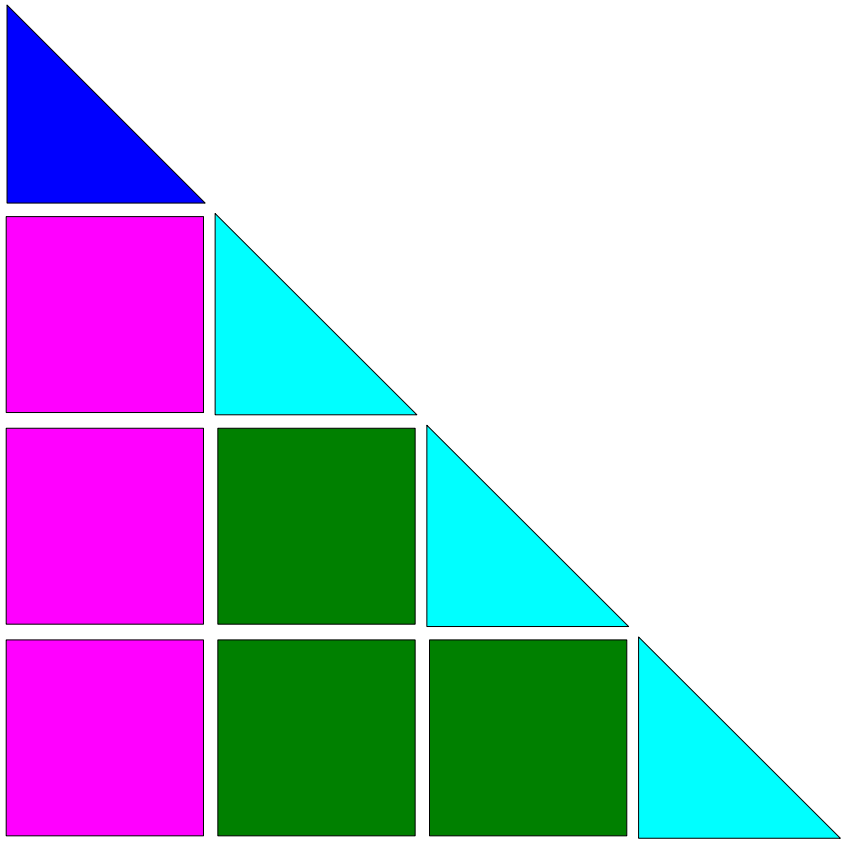
Task graph (~'08-today)

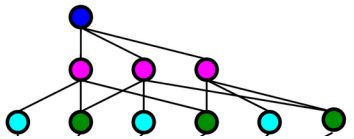




Cholesky, PLASMA

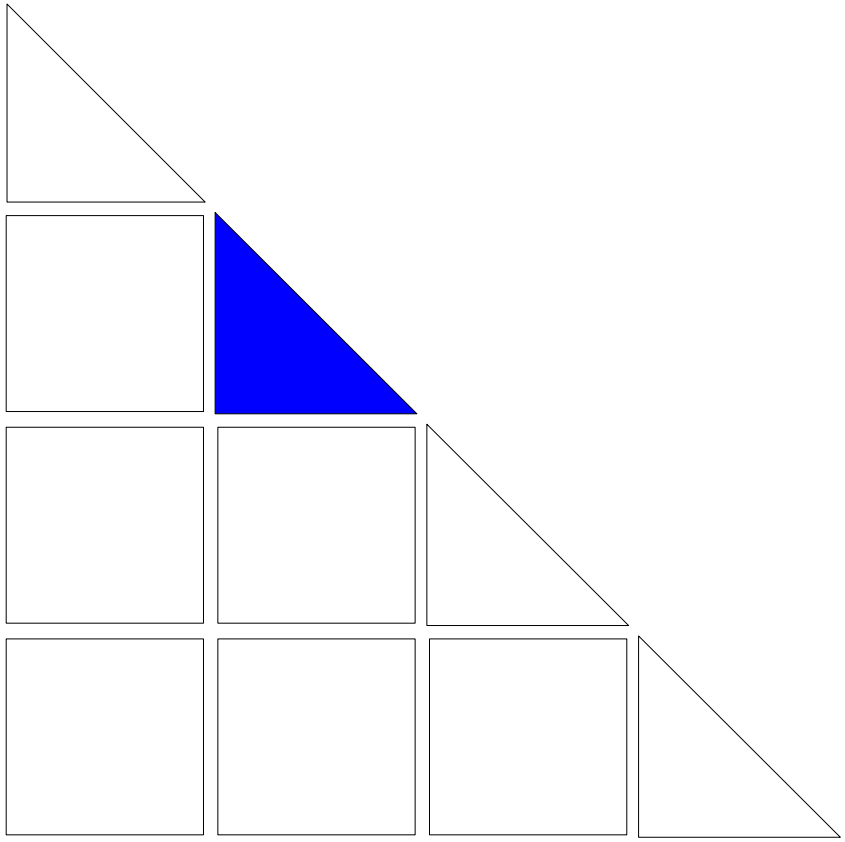
Task graph (~'08-today)

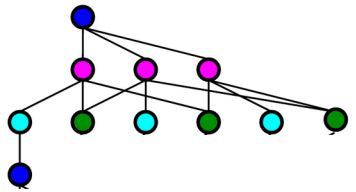




Cholesky, PLASMA

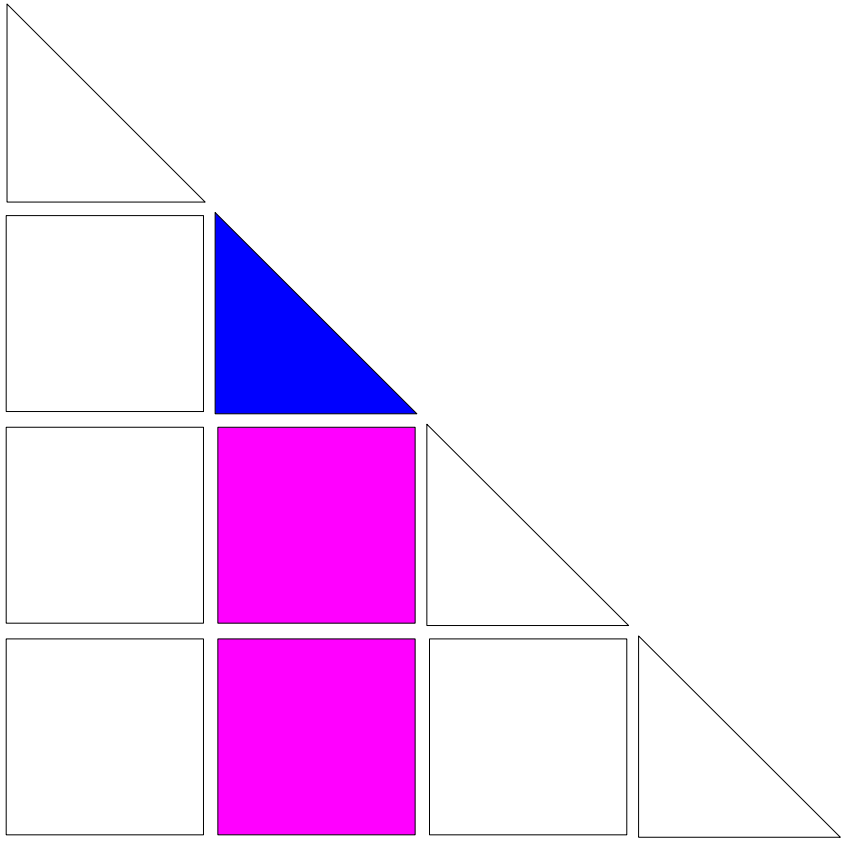
Task graph (~'08-today)

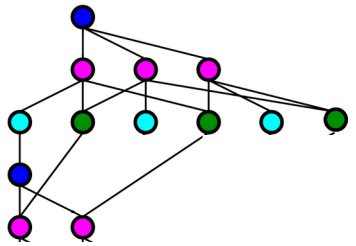




Cholesky, PLASMA

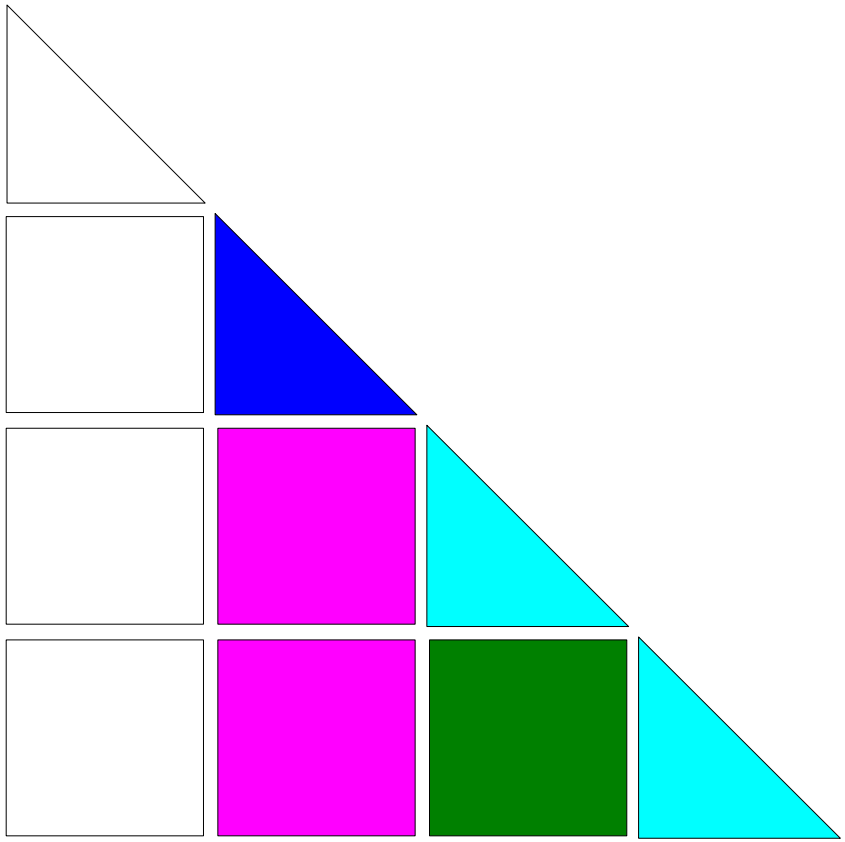
Task graph (~'08-today)

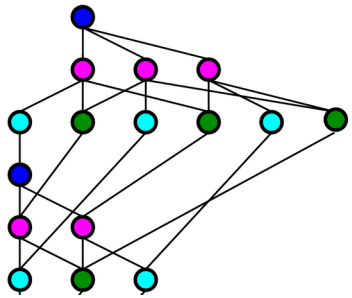




Cholesky, PLASMA

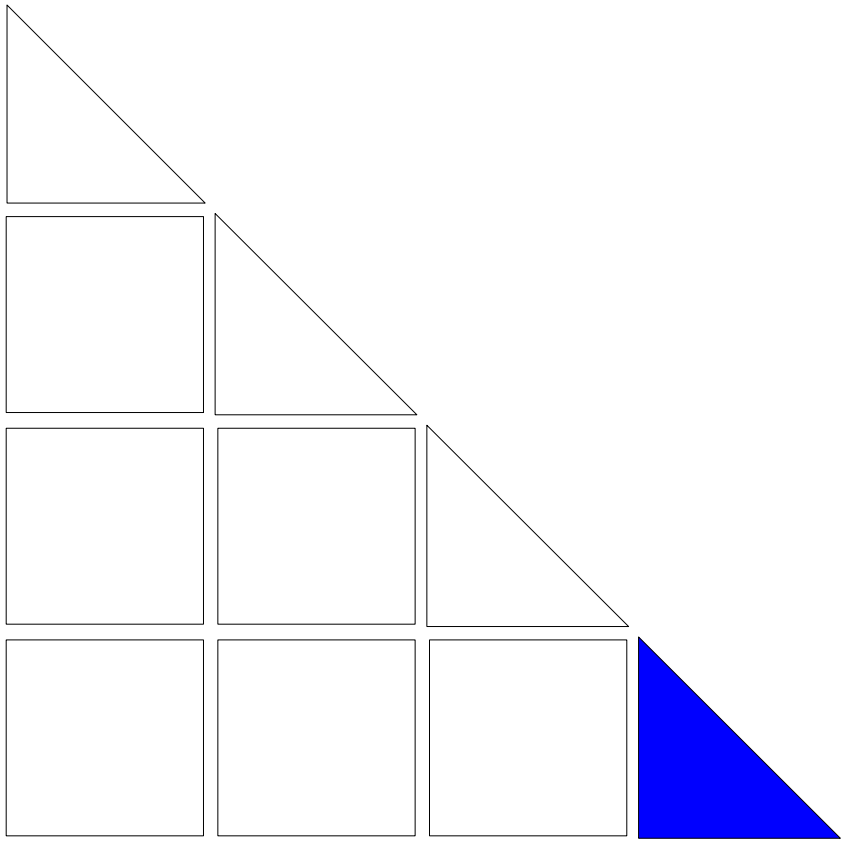
Task graph (~'08-today)

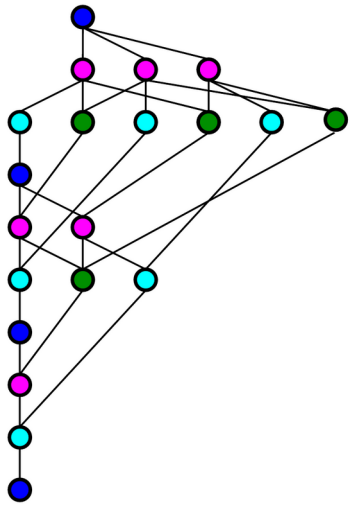




Cholesky, PLASMA

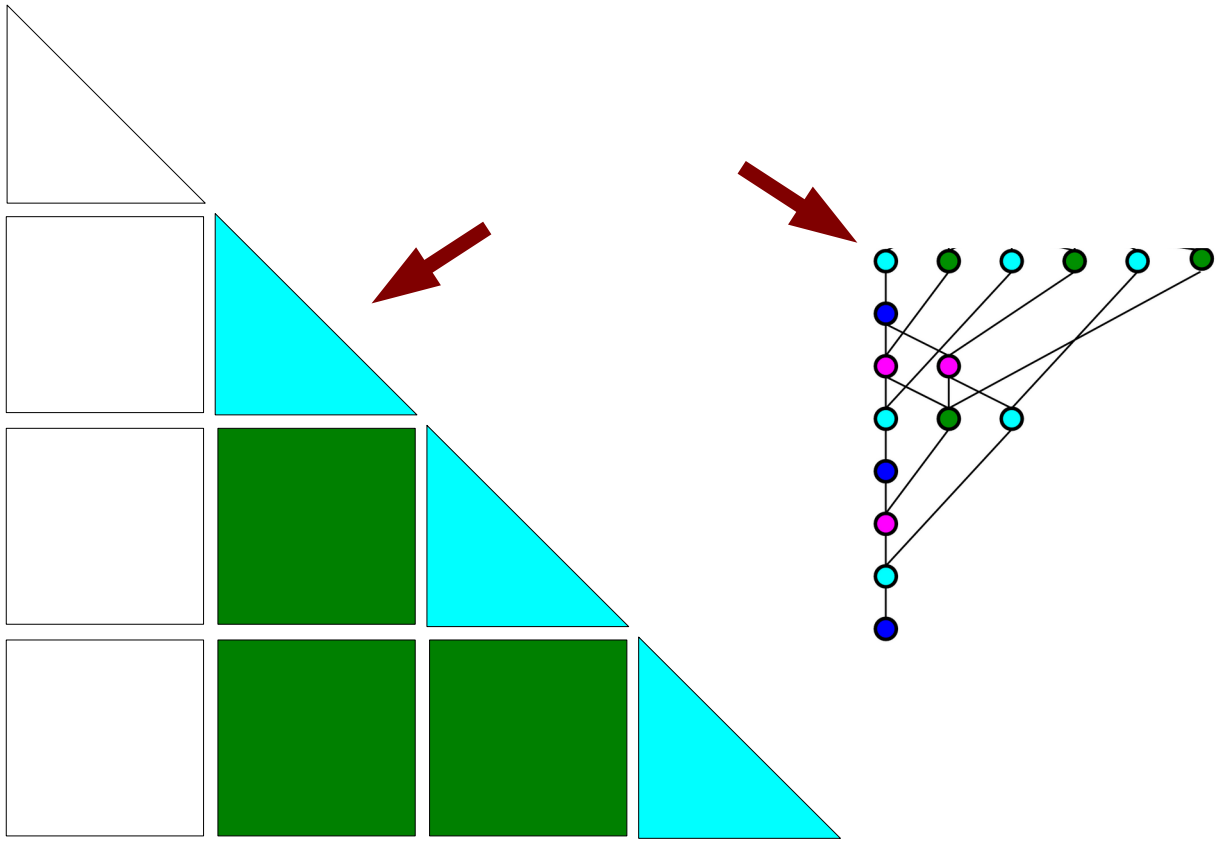
Task graph (~'08-today)





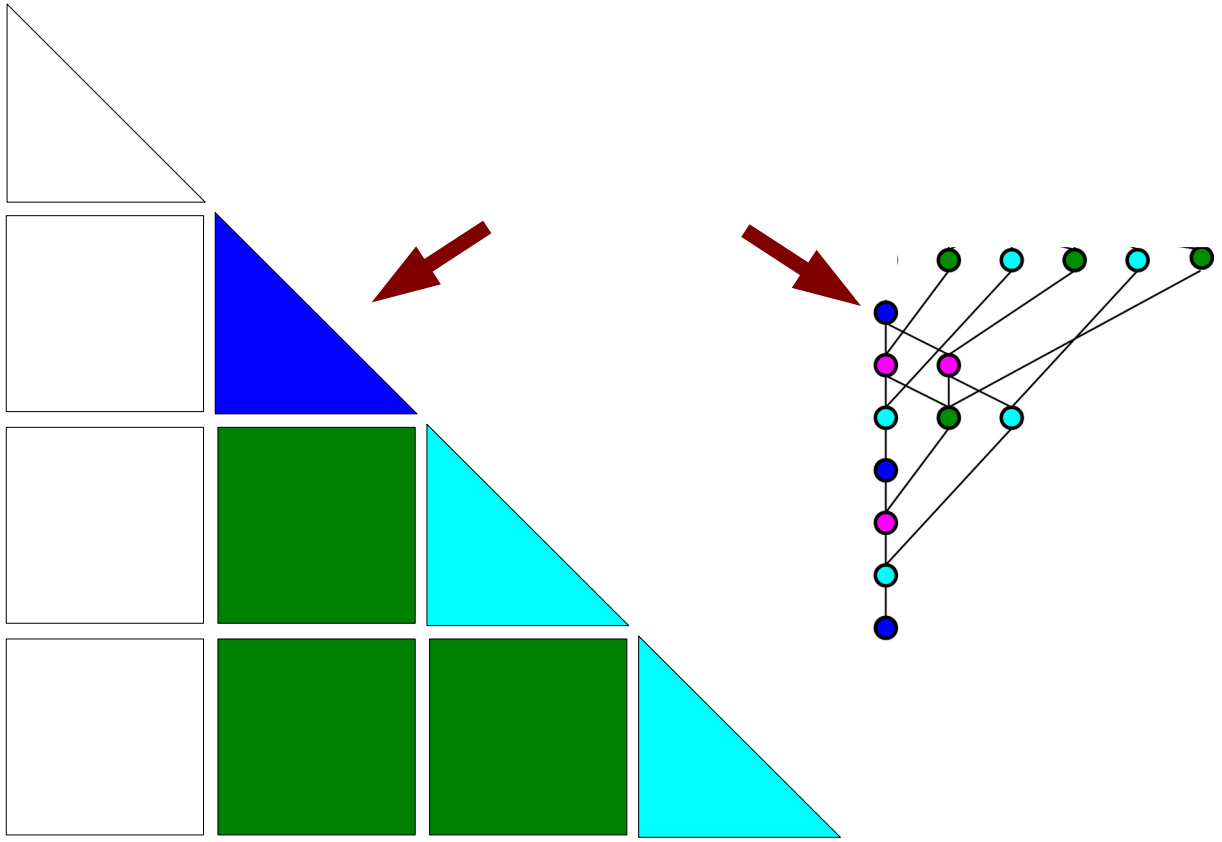
Cholesky, PLASMA

Task graph (~'08-today), **priorities**



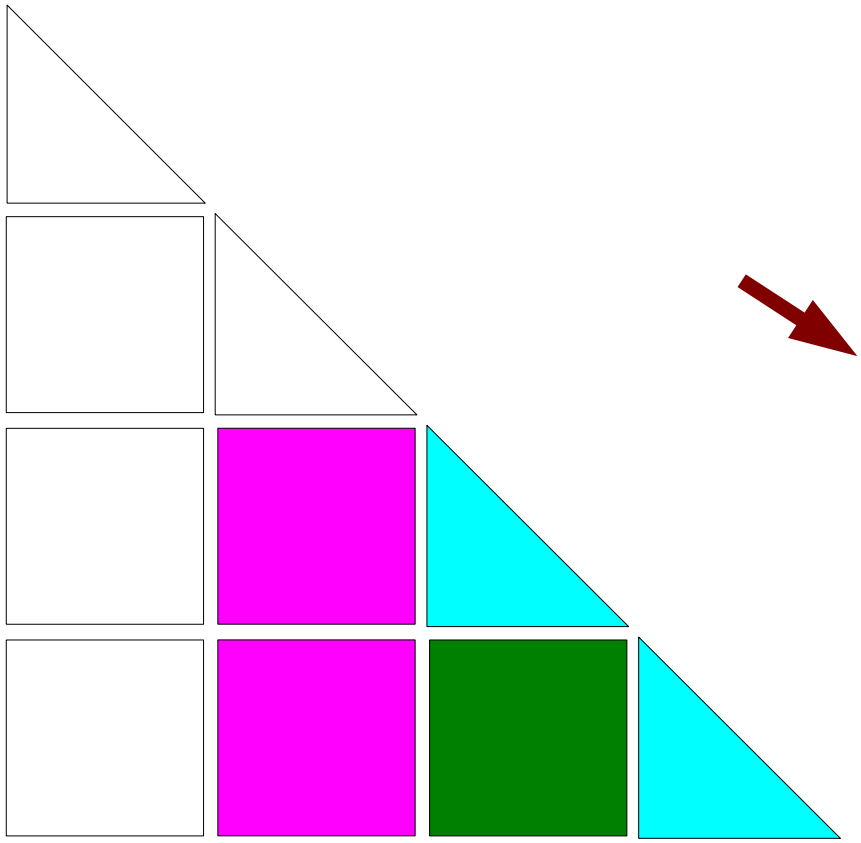
Cholesky, PLASMA

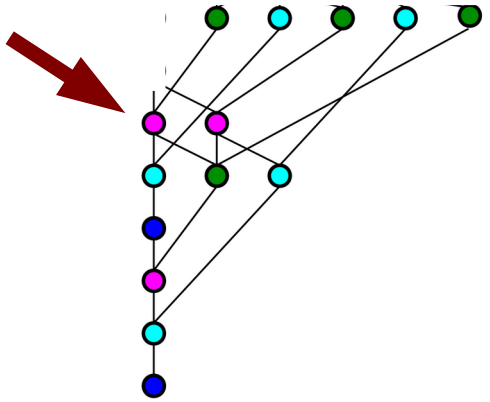
Task graph (~'08-today), **priorities**



Cholesky, PLASMA

Task graph (~'08-today), **priorities**





Task graphs in HPC

Runtime systems

- OmpSs, PARSEC, StarPU, SuperGlue/DuctTeip, XKaapi...

Standards

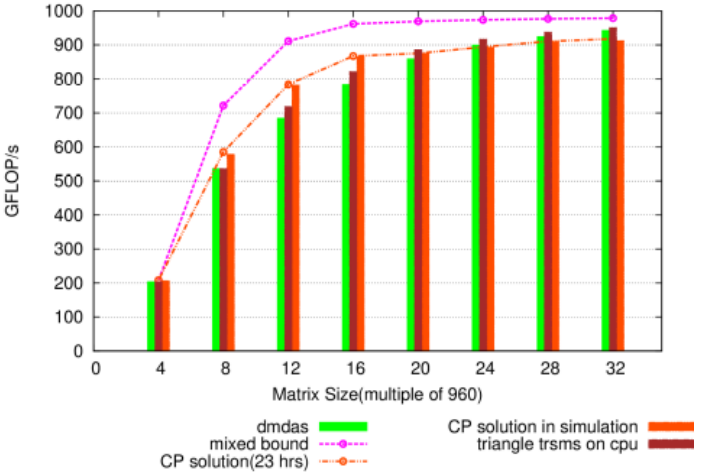
- OpenMP4.0 introduced task dependencies

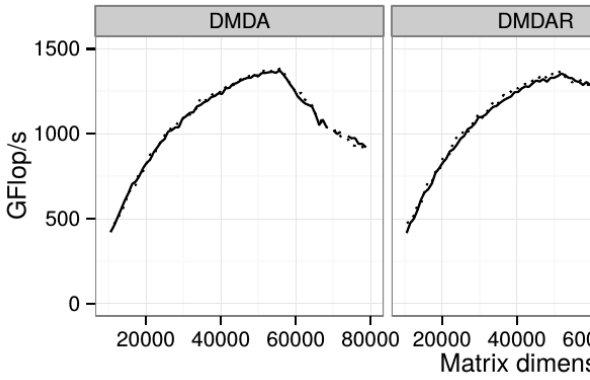
Applications

- Chameleon, DPLASMA, SLATE for dense linear algebra
- qr_mumps, PaStiX for sparse linear algebra
- ScalFMM for FMM
- …

Task-based support

- Task/data scheduling
  - Load balancing
  - Data prefetching
  - Pipelining
  - GPU memory limitation management
- Distributed execution through MPI
- Out-of-core: optimized swapping to disk
- High-level performance analysis
- Performance bounds
- Debugging sequential execution
- Reproducible performance simulation

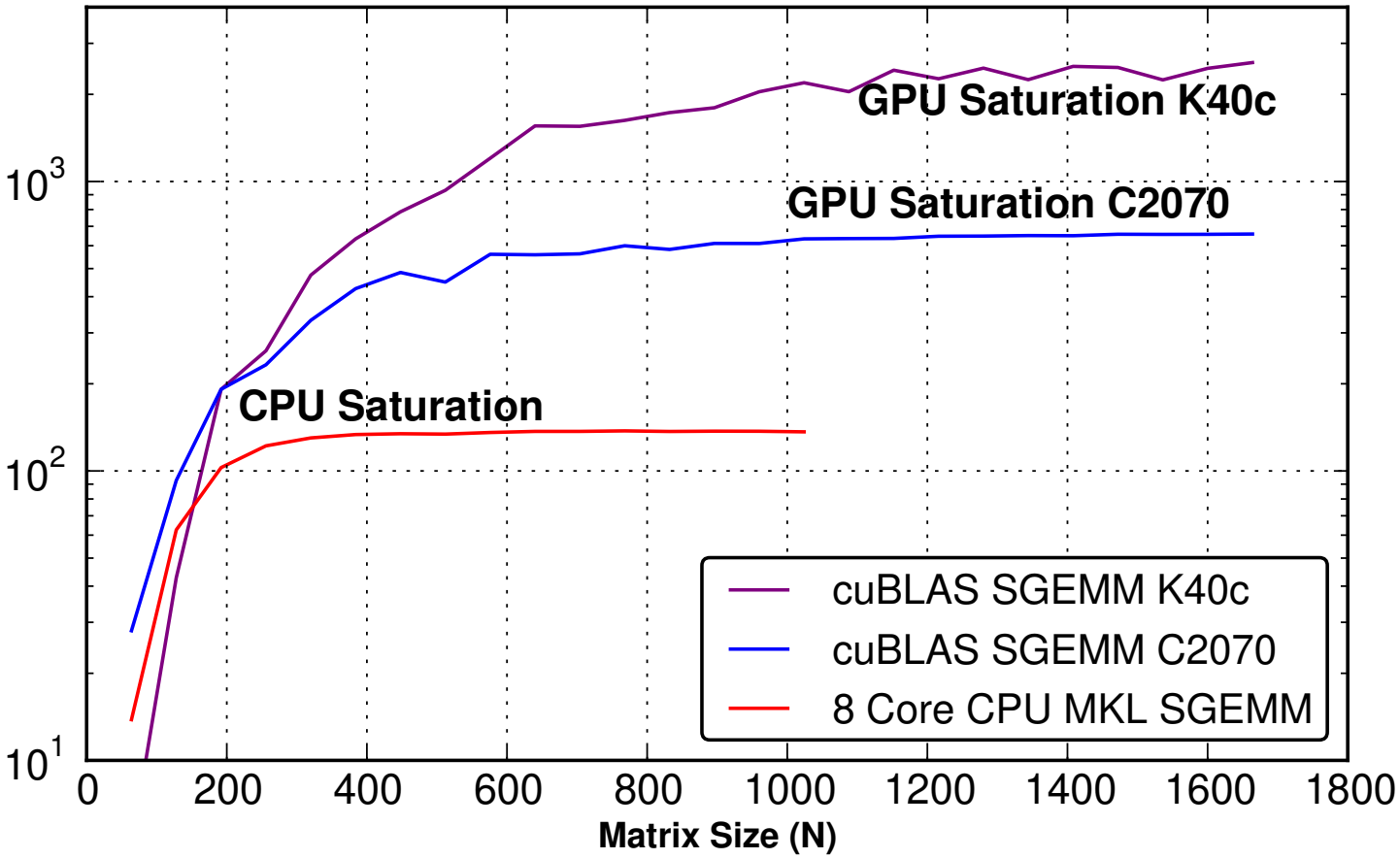




How big should a task be?

- Small enough to get parallelism to **feed** all processing units
- Large enough to **efficiently** use the processing units

How big should a task be?



From PARSEC : « Hierarchical DAG Scheduling for Hybrid Distributed Systems », Wu, Bouteiller, Bosilca, Faverge, Dongarra

How big should a task be?

GPUs

- Efficient only with large tile sizes

CPUs

- Need many tasks

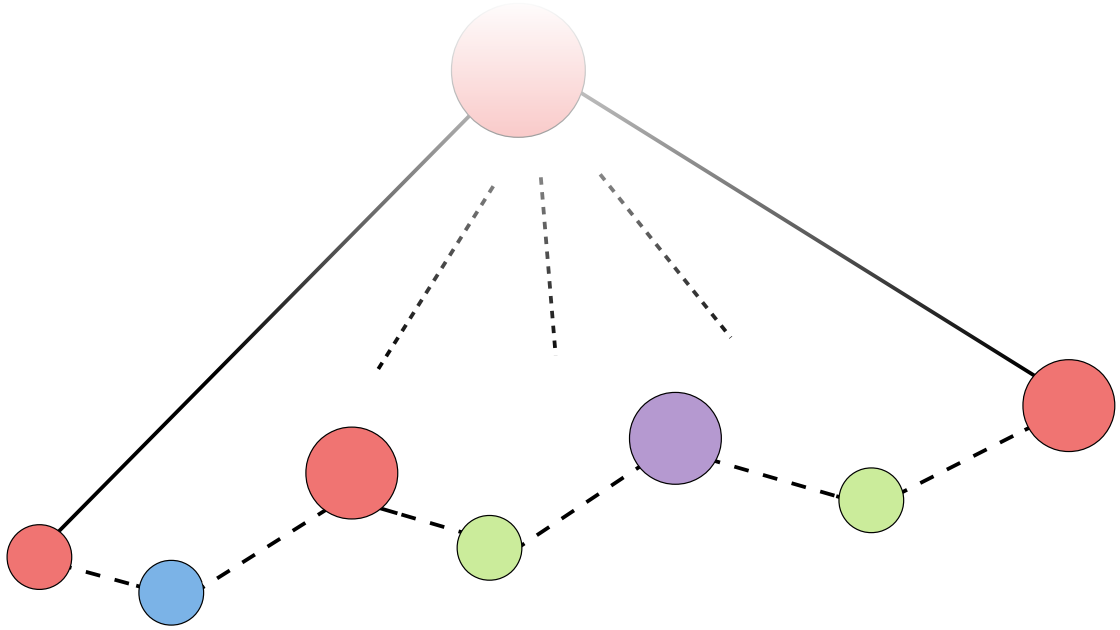
→ **Hybrid** task size

More generally, **hierarchical** task graphs

- Seen in CEA, OmpSs, PaRSEC, StarPU

Outline

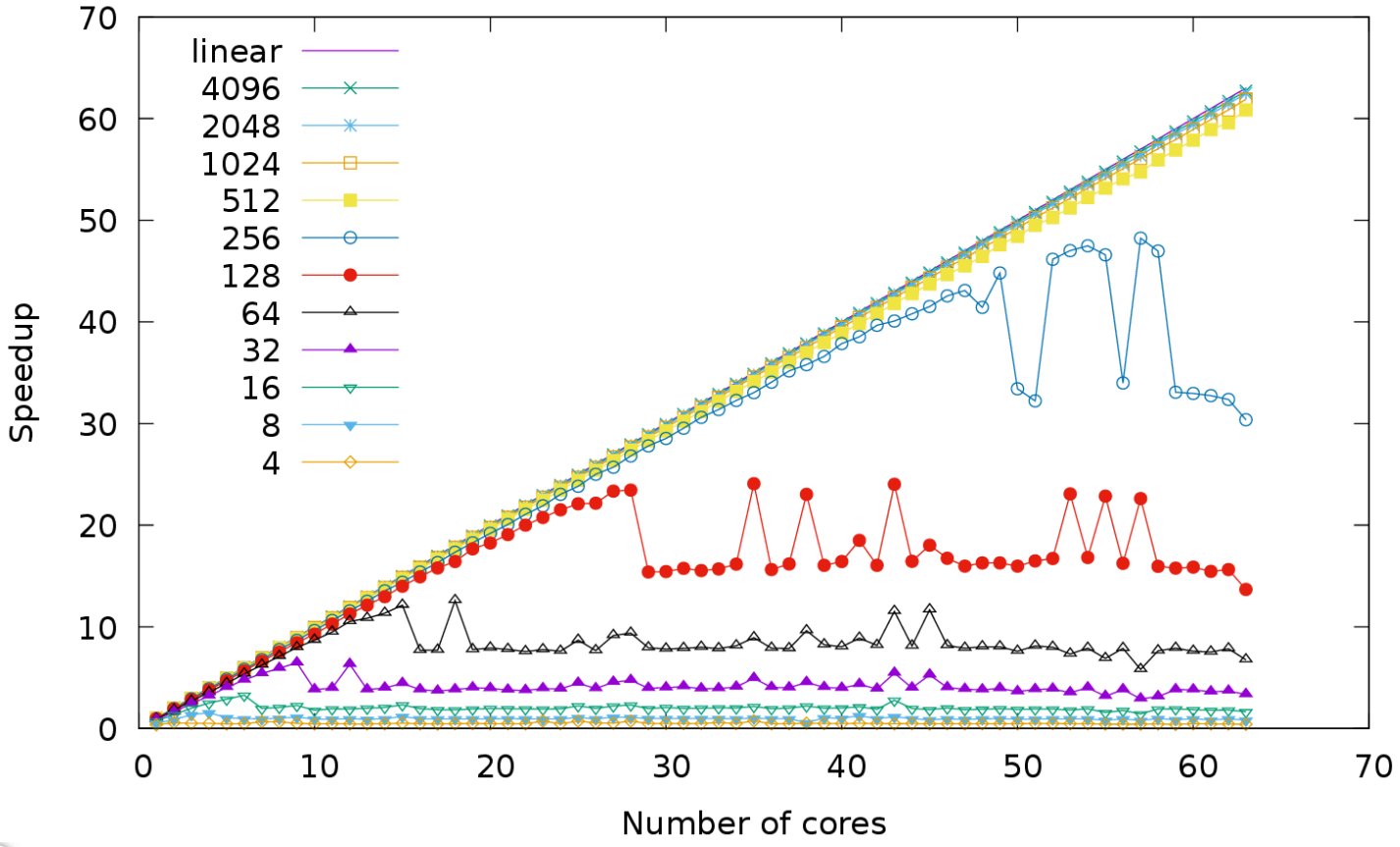
- How big should a task be?
- Hierarchical task graphs
- Opportunities



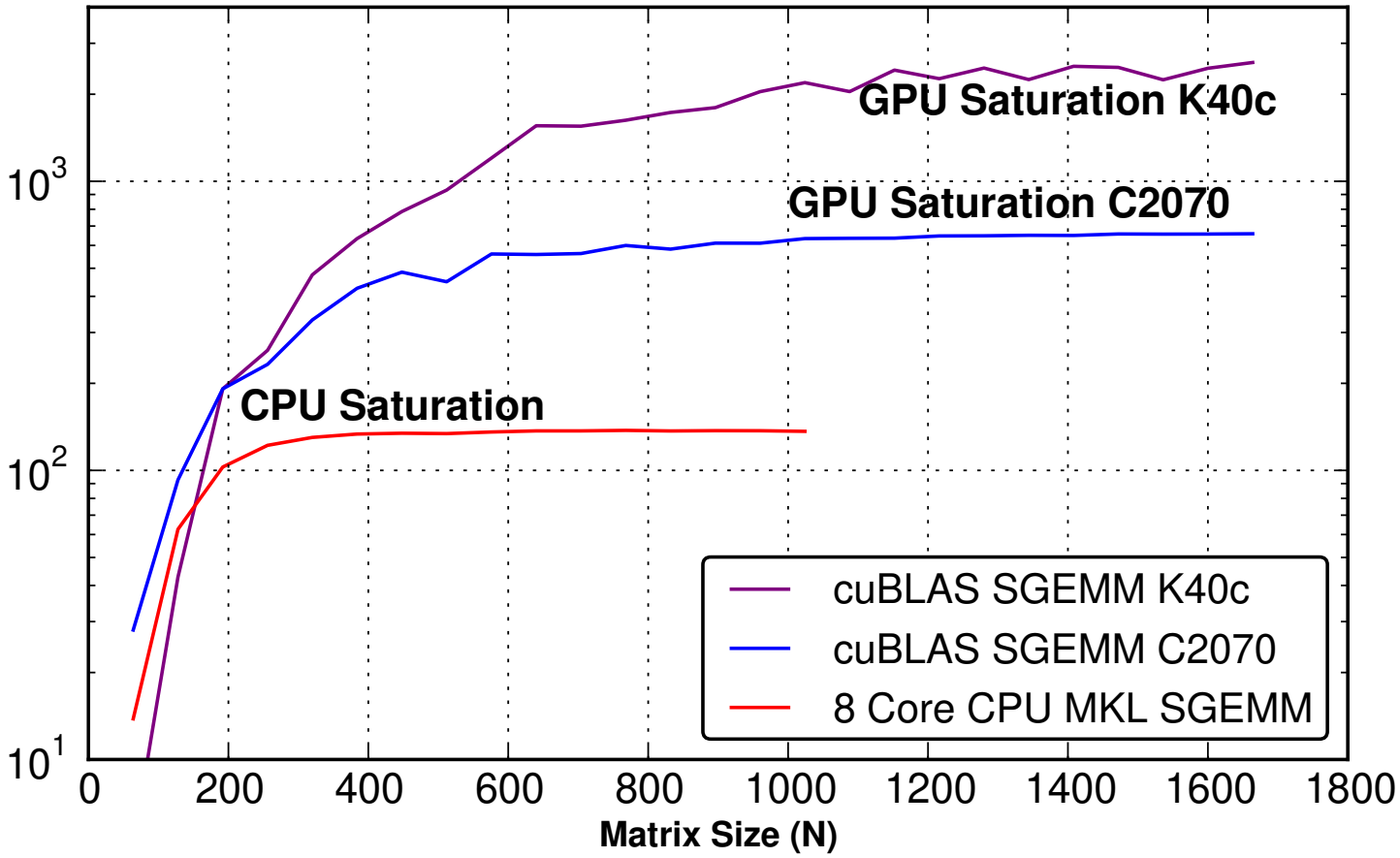
How big should a task be?

How big should a task be?

- Lower bound due to runtime overhead
  - Proposed by Martin Tillenius for DuctTeip (U. Uppsala)



How big should a task be?



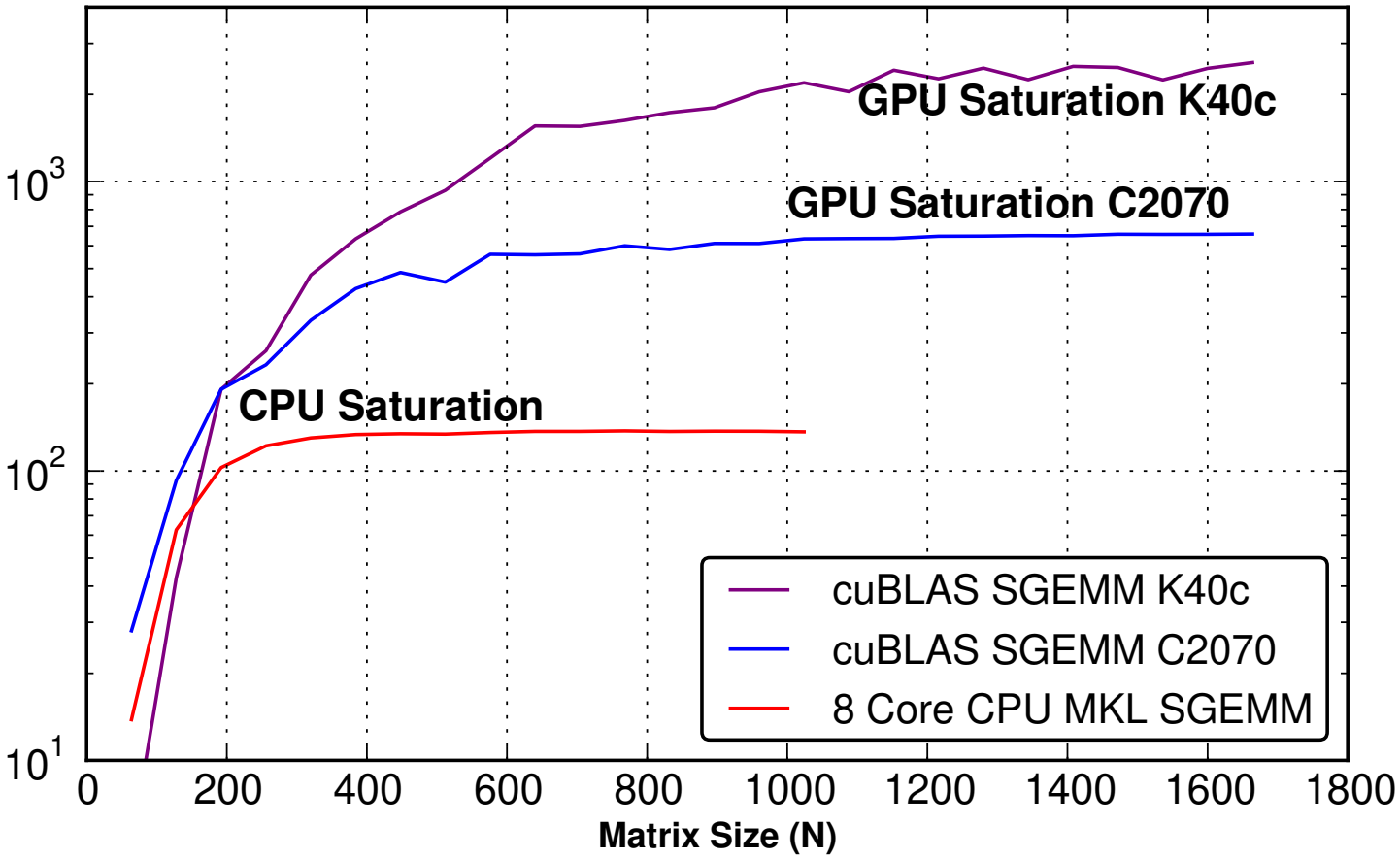
From PARSEC : « Hierarchical DAG Scheduling for Hybrid Distributed Systems », Wu, Bouteiller, Bosilca, Faverge, Dongarra

How big should a task be?

GPUs

- Have **thousands** for cores to feed
- Newer generations require yet larger sizes
- Can run several kernels at the same time
  - Still limited

How big should a task be?



From PARSEC : « Hierarchical DAG Scheduling for Hybrid Distributed Systems », Wu, Bouteiller, Bosilca, Faverge, Dongarra

How big should a task be?

CPUs

- Have many independent cores

  → Need many tasks
- Can use parallel implementations (e.g. from MKL)
  - But better have subtasks to interleave them

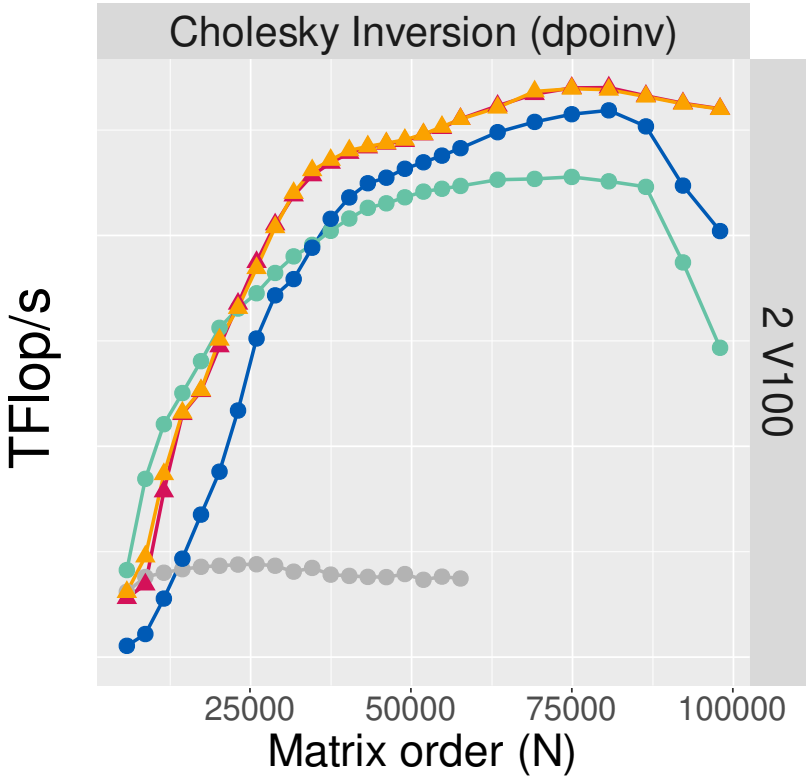
How big should a task be?

How size does not fit all

Gwenolé Lucas experimented with Chameleon

- Cholesky inversion
- 2 NVIDIA V100
- 2 Xeon Gold 6142
- Different matrix tile sizes : 2880 / 960 / 320

Hybrid tile sizes



Version: Tile sizes

- Non-Hierarchical: 2880
- Non-Hierarchical: 960
- Non-Hierarchical: 320
- Hierarchical: 2880 / 960
- Hierarchical: 2880 / 960 / 320

How big should a task be?

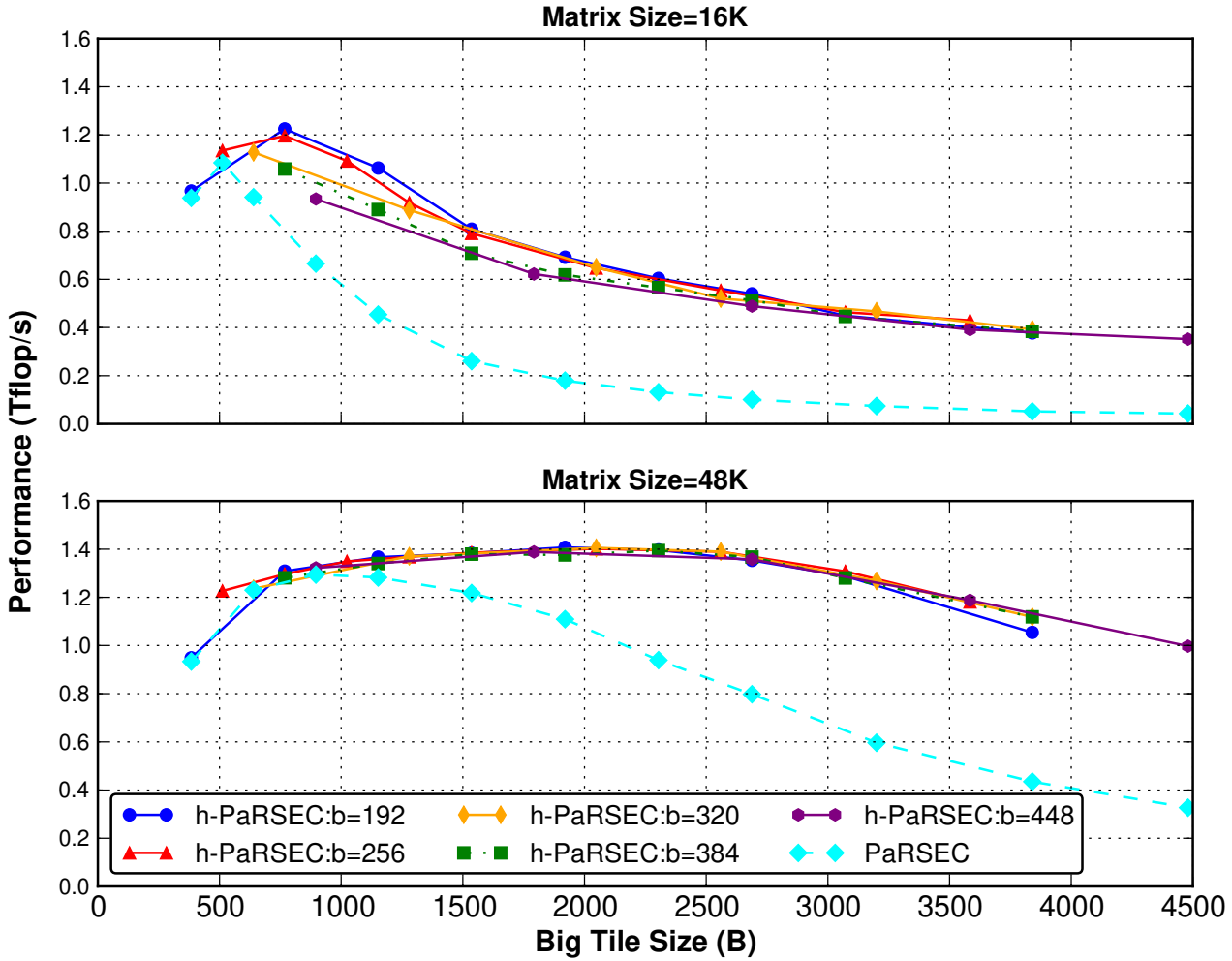
Tuning the tile size?

- B = “big” tile size, for GPUs
  - Large enough for GPUs
  - Not too large for parallelism

- b = “small” tile size, for CPUs
  - Large enough for CPUs
  - Not too large for parallelism

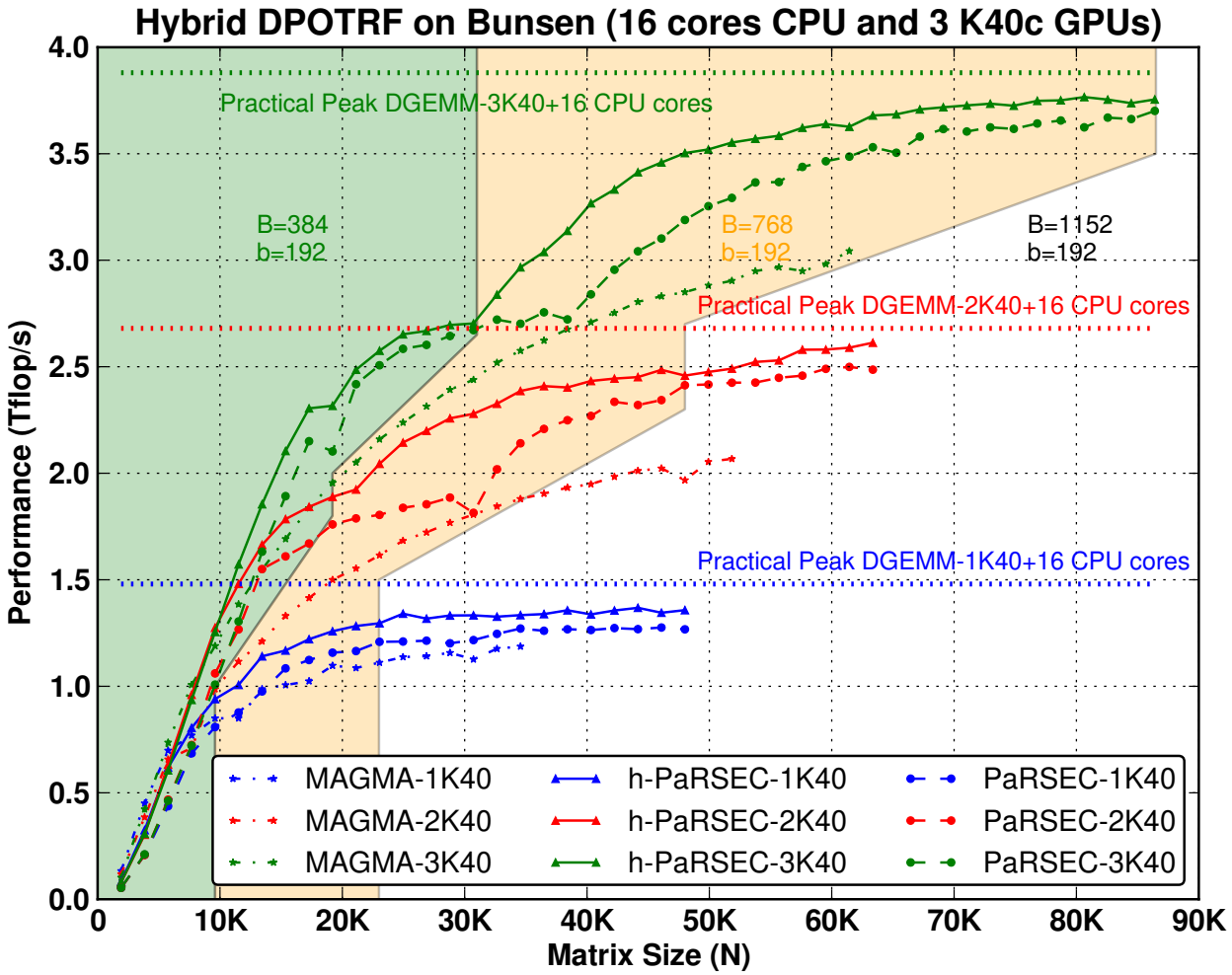
With Parsec, Wu, Bouteiller, Bosilca, Faverge and Dongarra experimented tuning

How big should a task be?



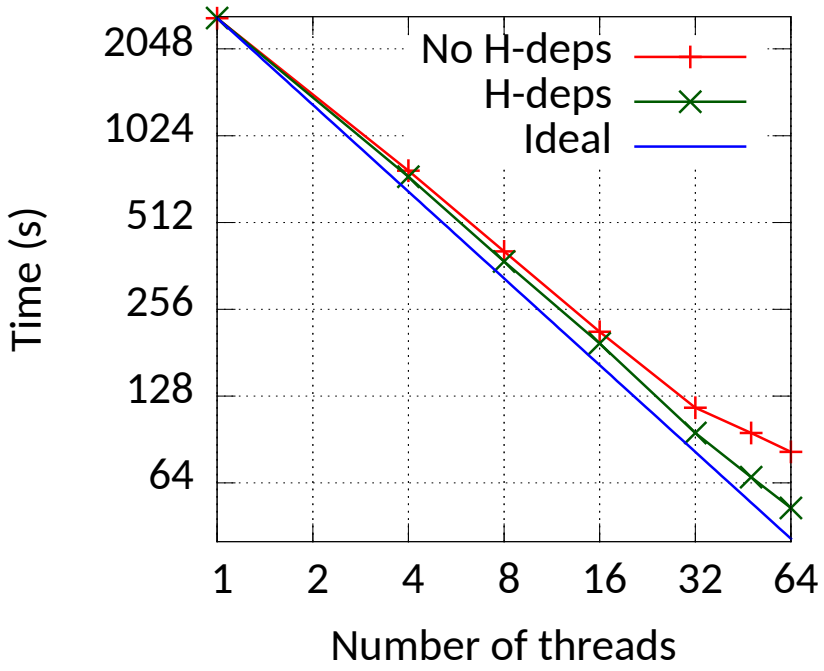
From PARSEC : « Hierarchical DAG Scheduling for Hybrid Distributed Systems », Wu, Bouteiller, Bosilca, Faverge, Dongarra

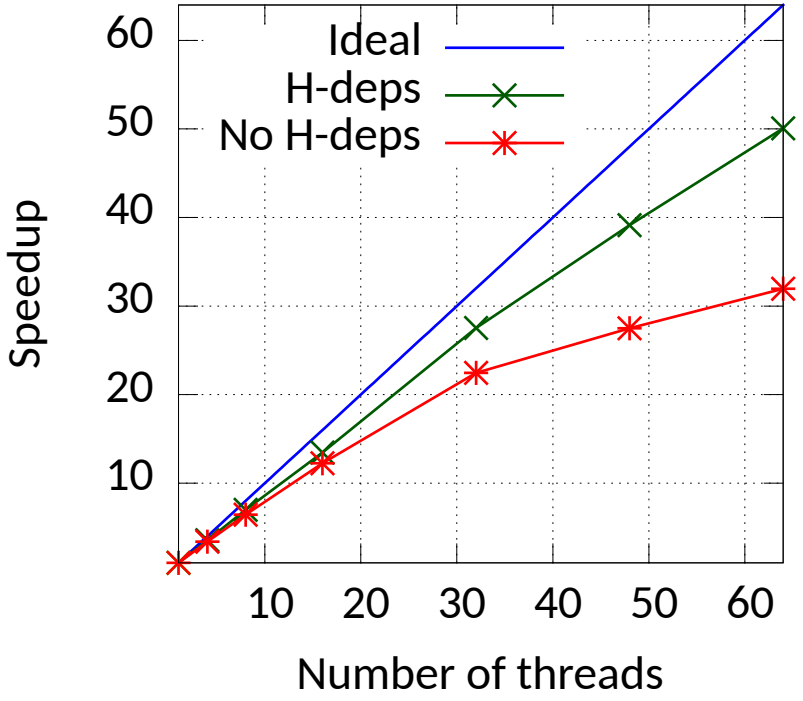
How big should a task be?



From PARSEC : « Hierarchical DAG Scheduling for Hybrid Distributed Systems », Wu, Bouteiller, Bosilca, Faverge, Dongarra

How big should a task be?





From CEA : « A hierarchical fast direct solver for distributed memory machines with manycore nodes », Augonnet, Goudin, Kuhn, Lacoste, Namyst, Ramet

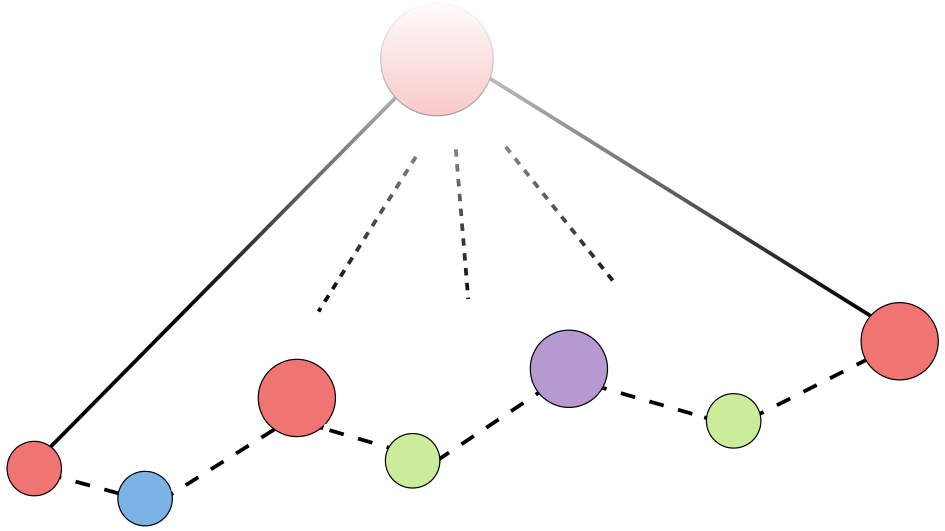
How big should a task be?

Multiple answers

- Depends on available platform parallelism
- Depends on available application parallelism
- Depends on application phases

Automatically adapt?

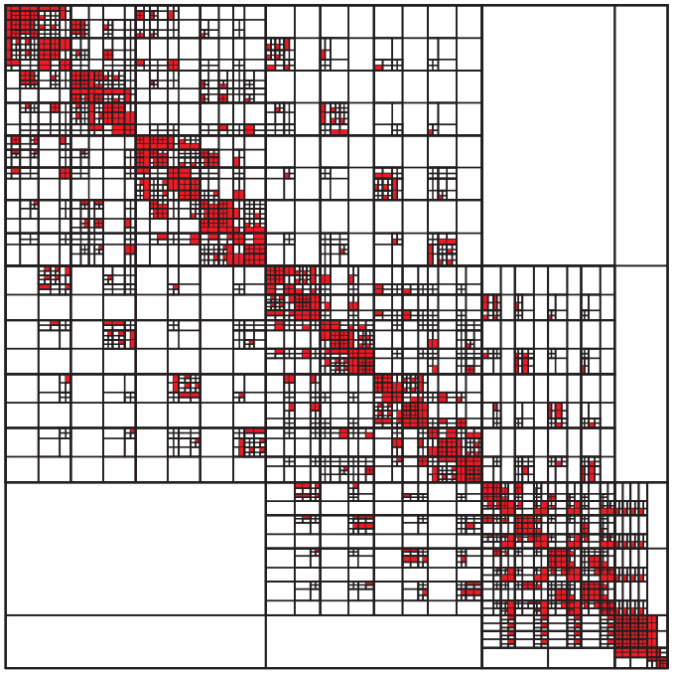
Hierarchical task graphs



Hierarchical task graphs

Applications themselves are recursive

- e.g. h-matrices

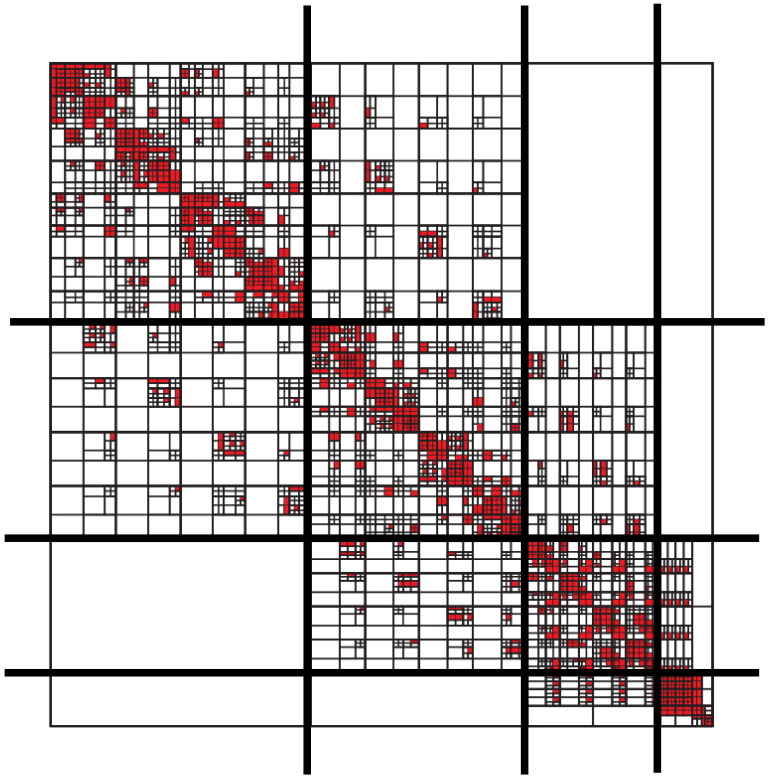


From Airbus Group

Hierarchical task graphs

Applications themselves are recursive

- e.g. h-matrices

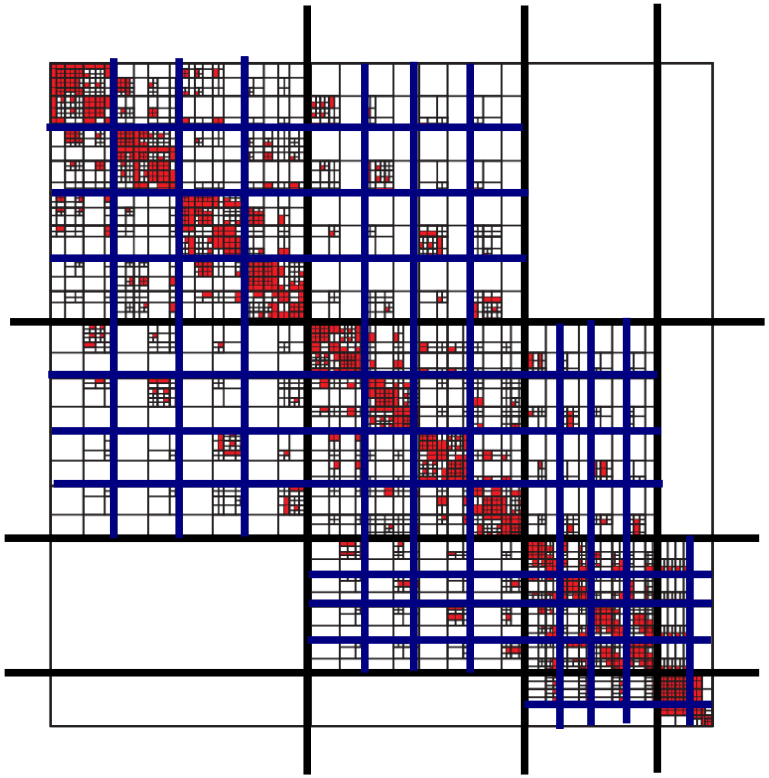


From Airbus Group

Hierarchical task graphs

Applications themselves are recursive

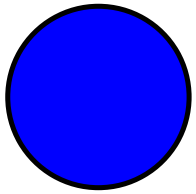
- e.g. h-matrices



From Airbus Group

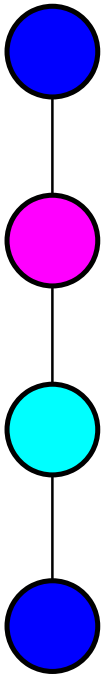
Hierarchical task graphs

Ideally, should just start with one huge task, and split



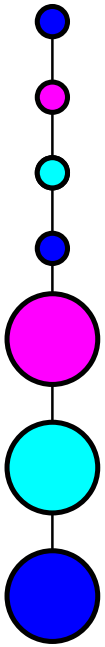
Hierarchical task graphs

Ideally, should just start with one huge task, and split



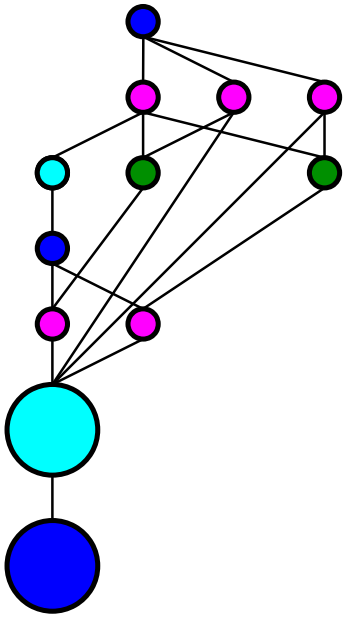
Hierarchical task graphs

Ideally, should just start with one huge task, and split



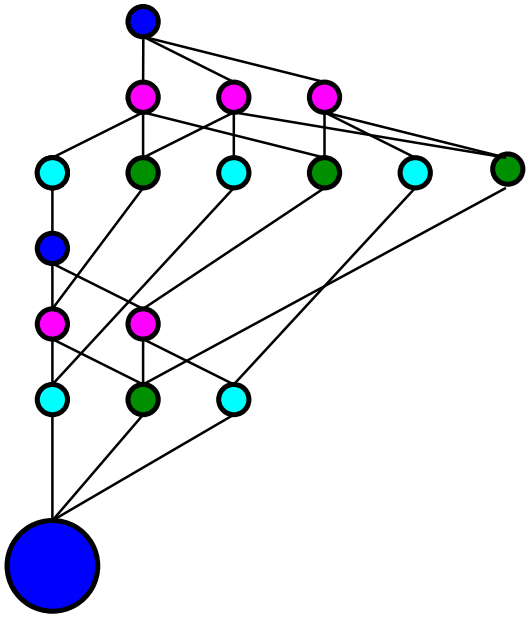
Hierarchical task graphs

Ideally, should just start with one huge task, and split



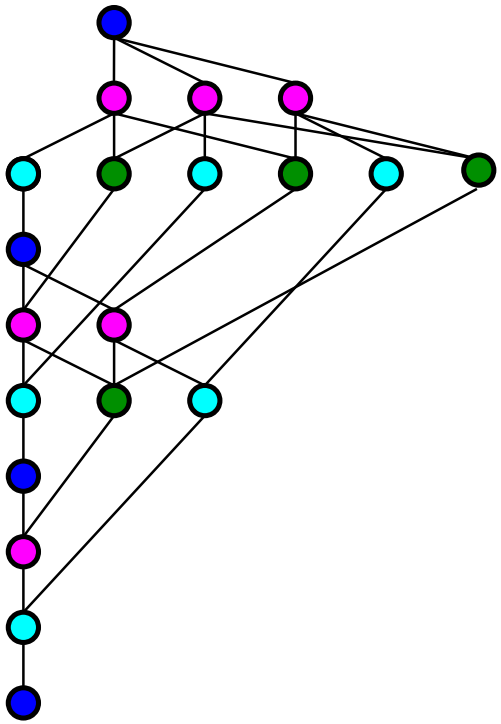
Hierarchical task graphs

Ideally, should just start with one huge task, and split



Hierarchical task graphs

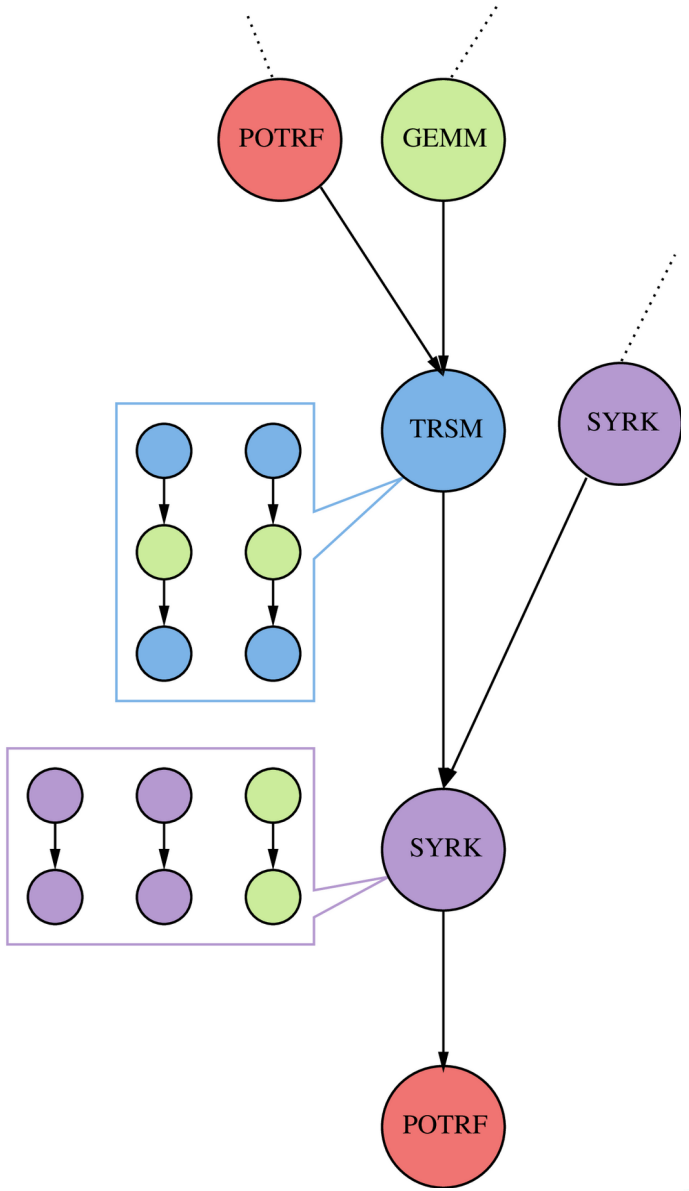
Ideally, should just start with one huge task, and split



Hierarchical task graphs

- In OpenMP?

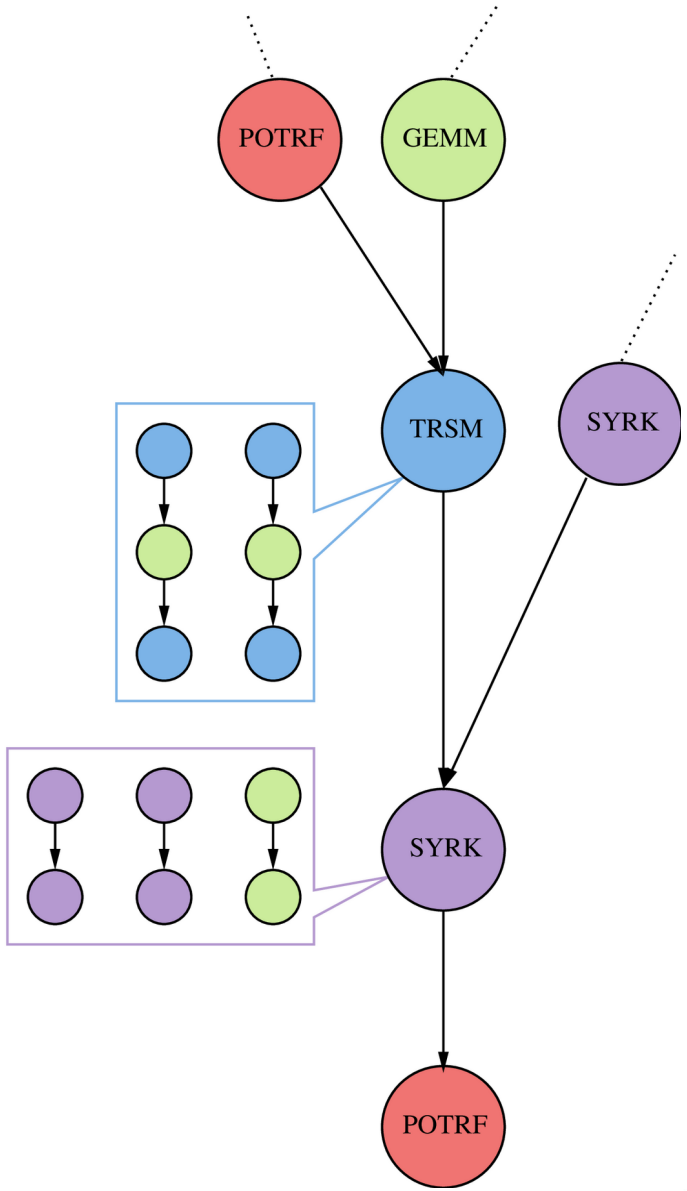
```
TRSM(A, B) {
#pragma omp task depend(in:A, inout:B) wait
{
    #pragma omp task \
            depend(in:A[0][0], inout:B[0][0])
        TRSM(A[0][0], B[0][0])
    #pragma omp task \
            depend(in:A[0][0], inout:B[1][0])
        TRSM(A[0][0], B[1][0])
    #pragma omp task \
            depend(in:A[1][0], in:B[0][0], inout:B[0][1])
        GEMM(A[1][0], B[0][0], B[0][1])
    ...
```



Hierarchical task graphs

Synchronization concerns

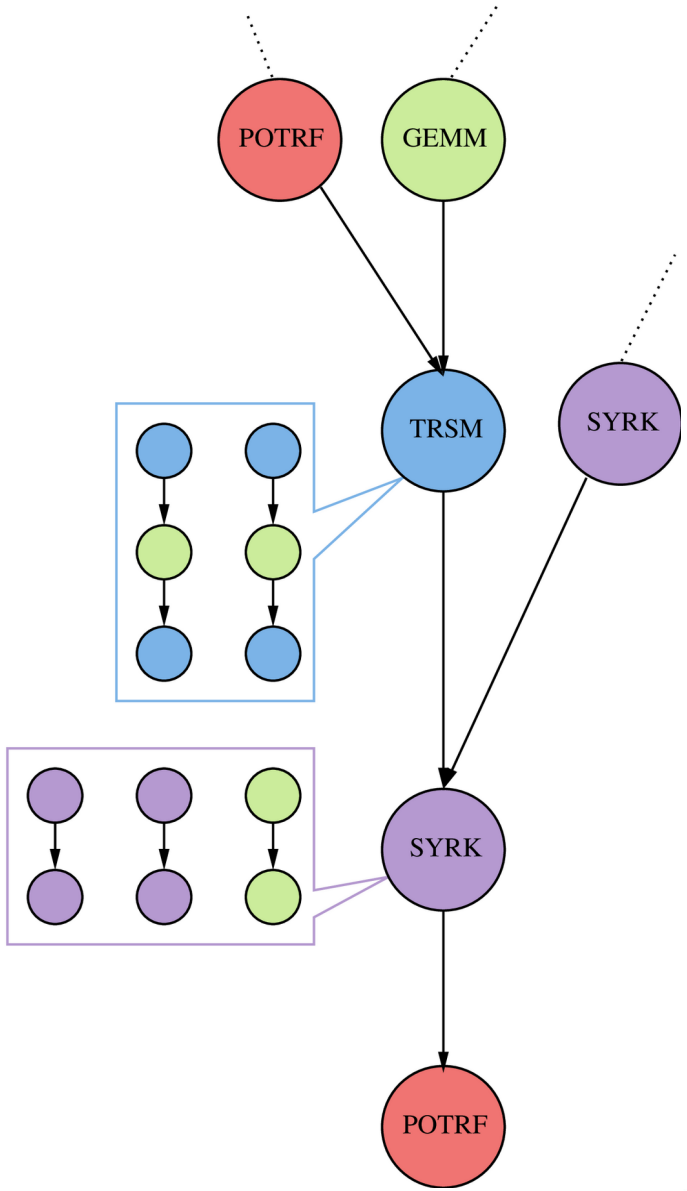
- Fork-join parallelism
- Hindered by synchronization induced by task graph



Hierarchical task graphs

- OmpSs introduced weak dependencies

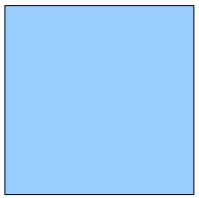
```
TRSM(A, B) {
#pragma omp task depend(weakin:A, weakinout:B) \
                    weakwait
{
   #pragma omp task \
         depend(in:A[0][0], inout:B[0][0])
        TRSM(A[0][0], B[0][0])
   #pragma omp task \
        depend(in:A[0][0], inout:B[1][0])
         TRSM(A[0][0], B[1][0])
   #pragma omp task \
        depend(in:A[1][0], in:B[0][0], inout:B[0][1])
         GEMM(A[1][0], B[0][0], B[0][1])
   ...
```



Hierarchical task graphs

In StarPU

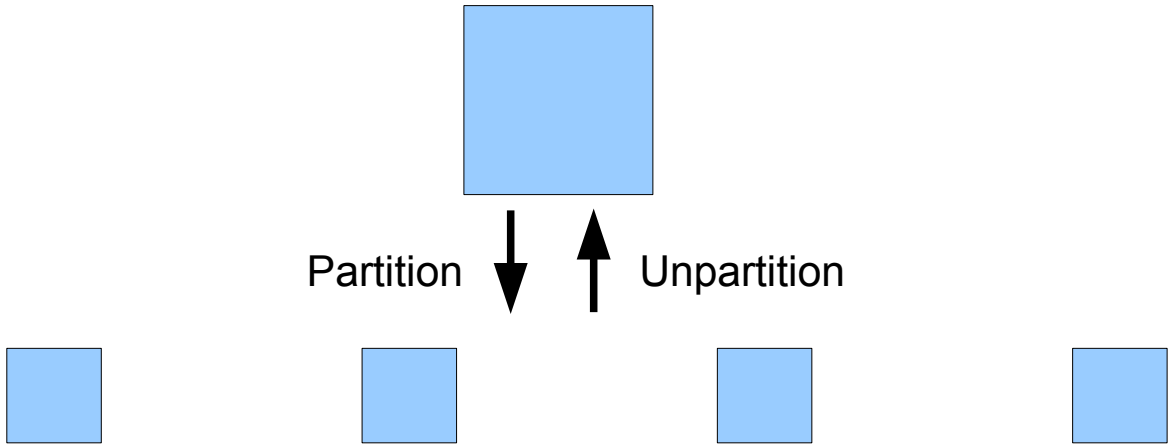
- Explicit data registration
- Recursive partitioning



Hierarchical task graphs

In StarPU

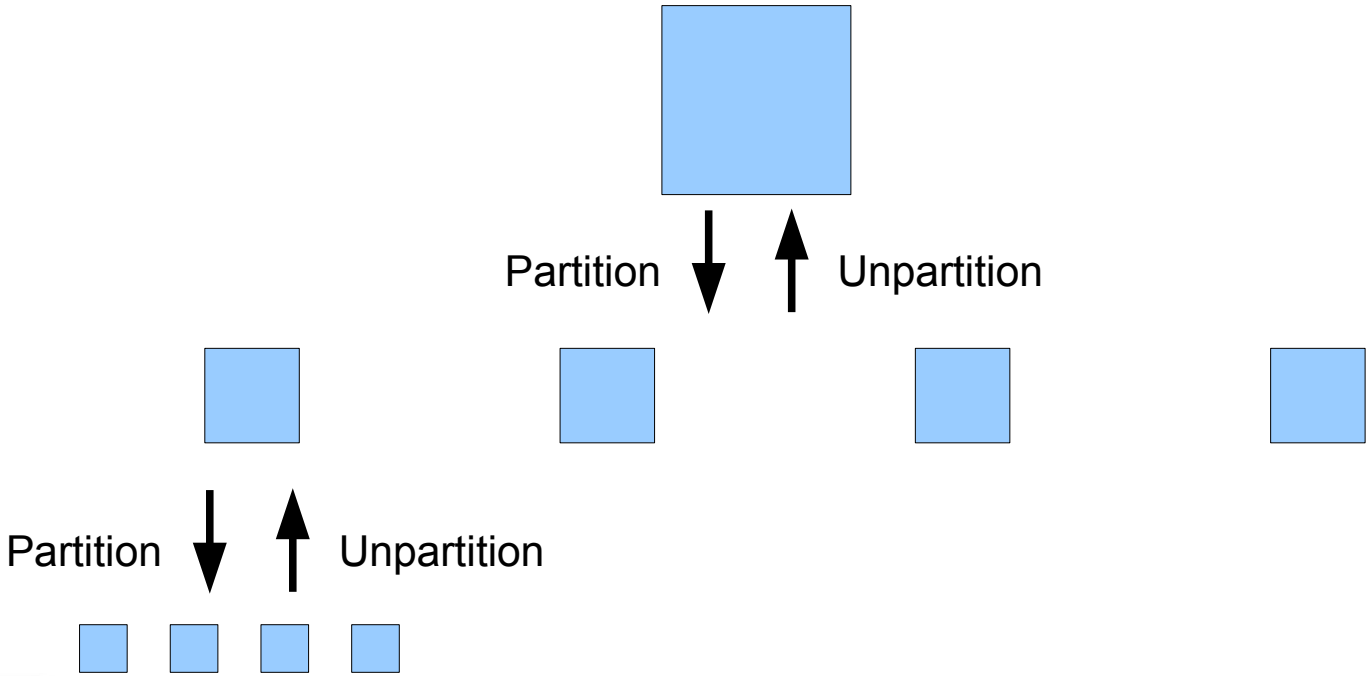
- Explicit data registration
- Recursive partitioning



Hierarchical task graphs

In StarPU

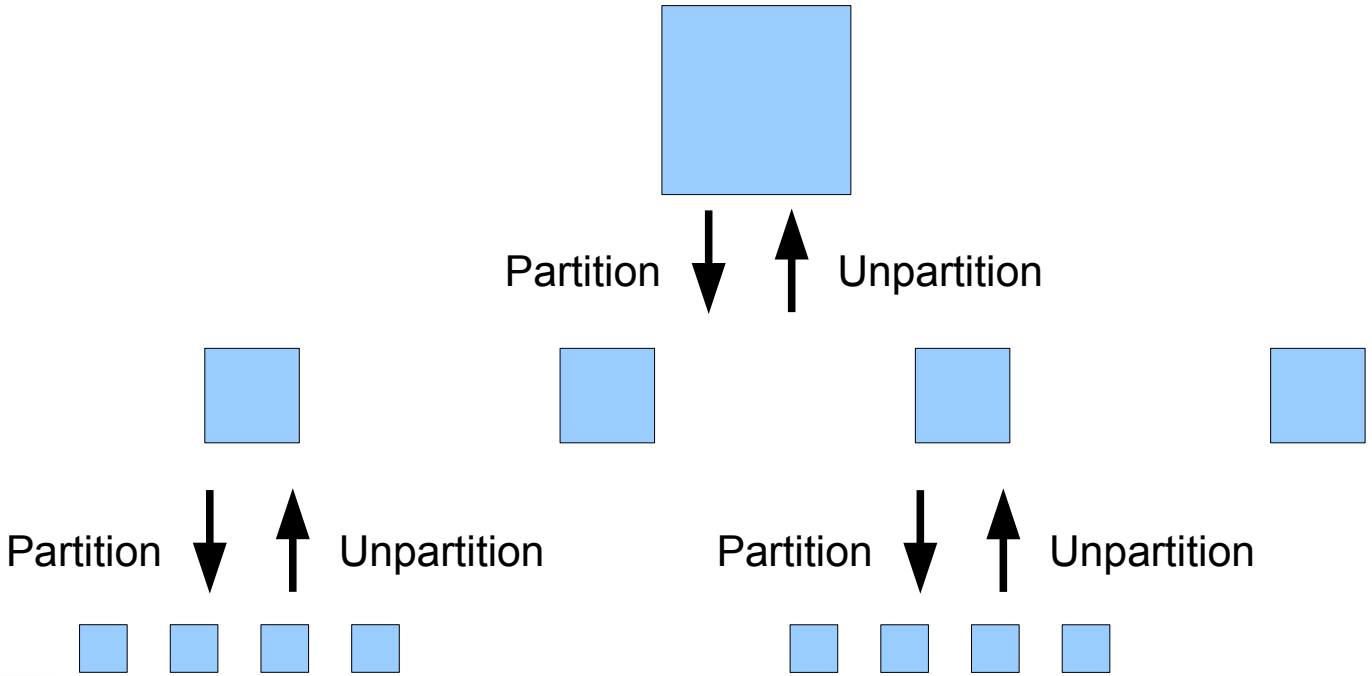
- Explicit data registration
- Recursive partitioning



Hierarchical task graphs

In StarPU

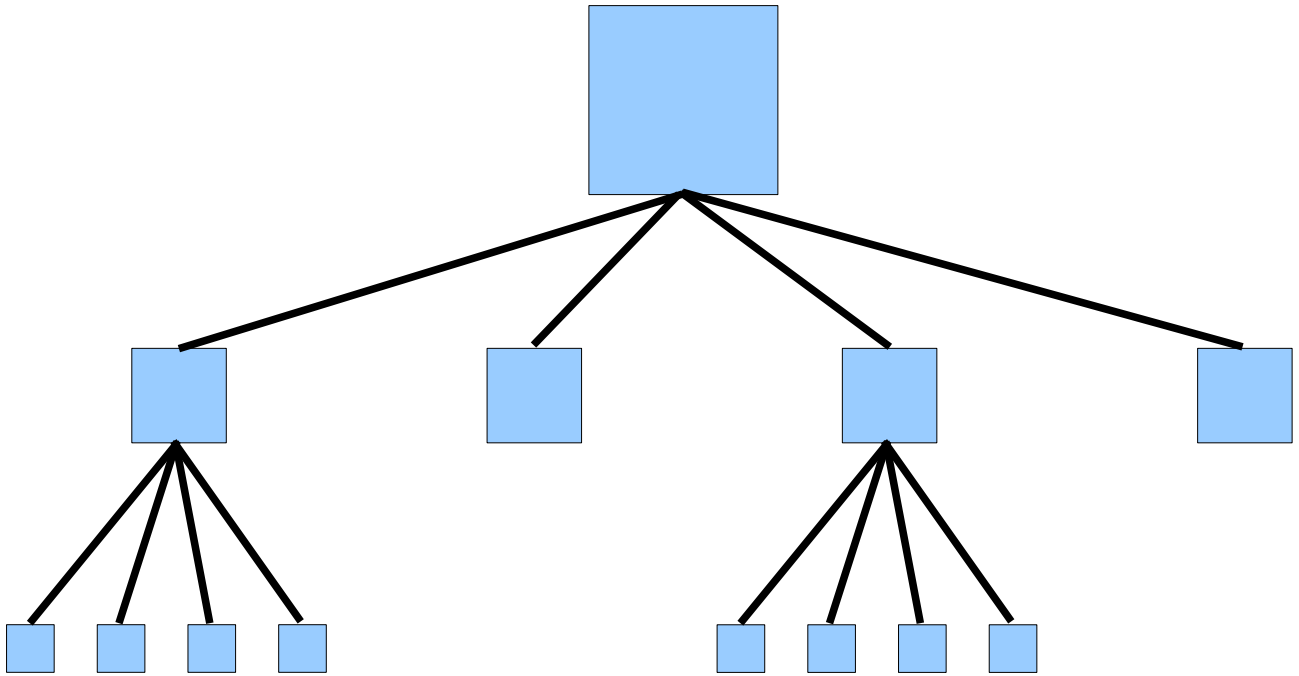
- Explicit data registration
- Recursive partitioning



Hierarchical task graphs

In StarPU

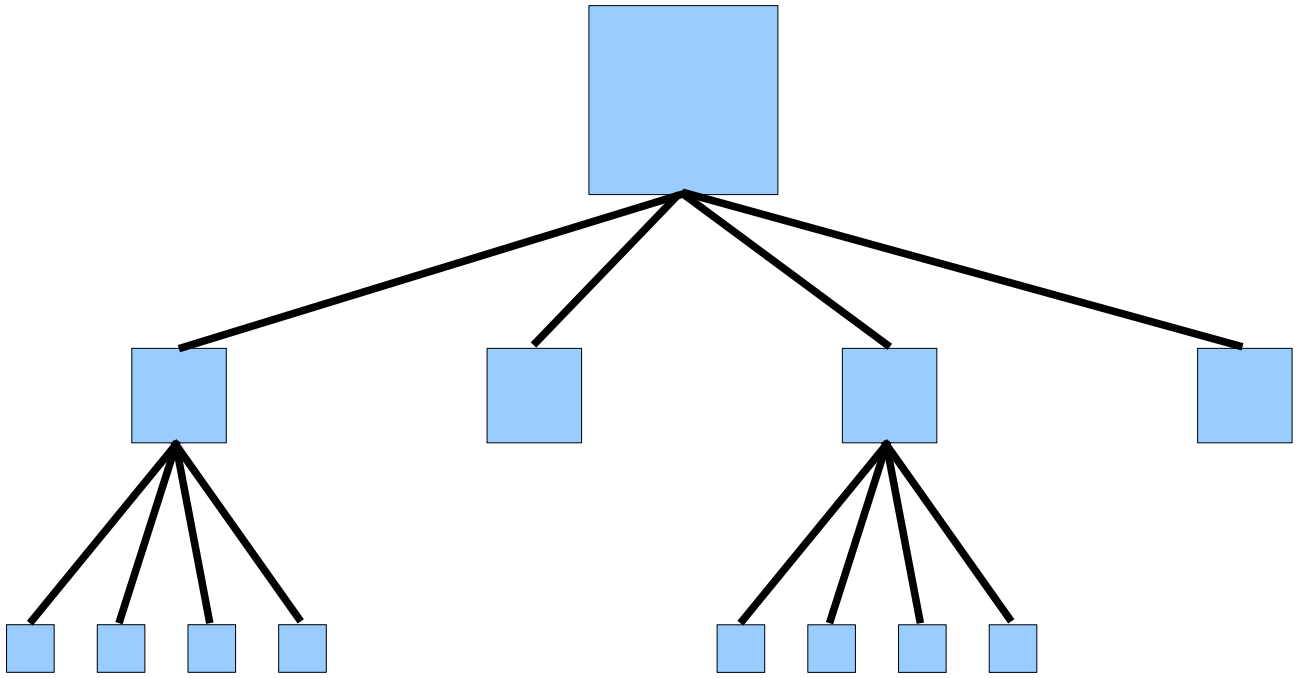
- Explicit data registration
- Recursive partitioning



Hierarchical task graphs

In StarPU

- Multi-level data coherency, among CPUs-GPUs-disk
- Enables seamless recursive tasks
- More details in Gwenolé Lucas' talk Thursday 14:40

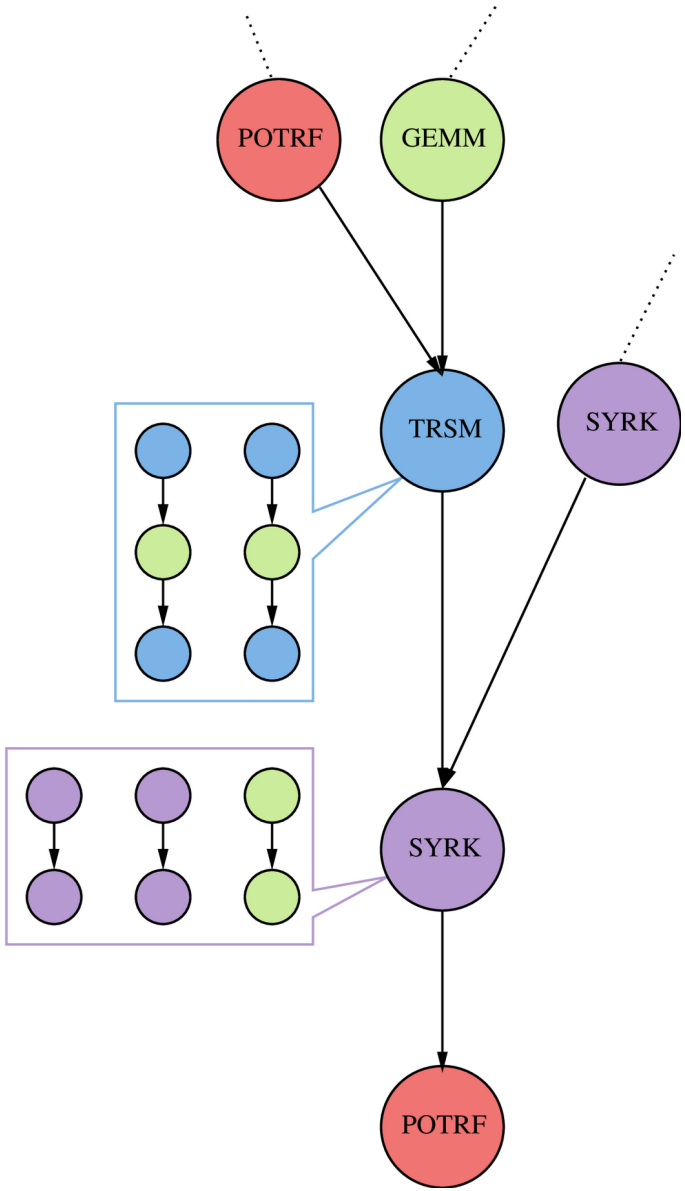


Hierarchical task graphs

Multi-level data coherency

- Synchronization pseudo-tasks
  - Only when needed
- Result is exactly as appropriate

→ Just split at will!

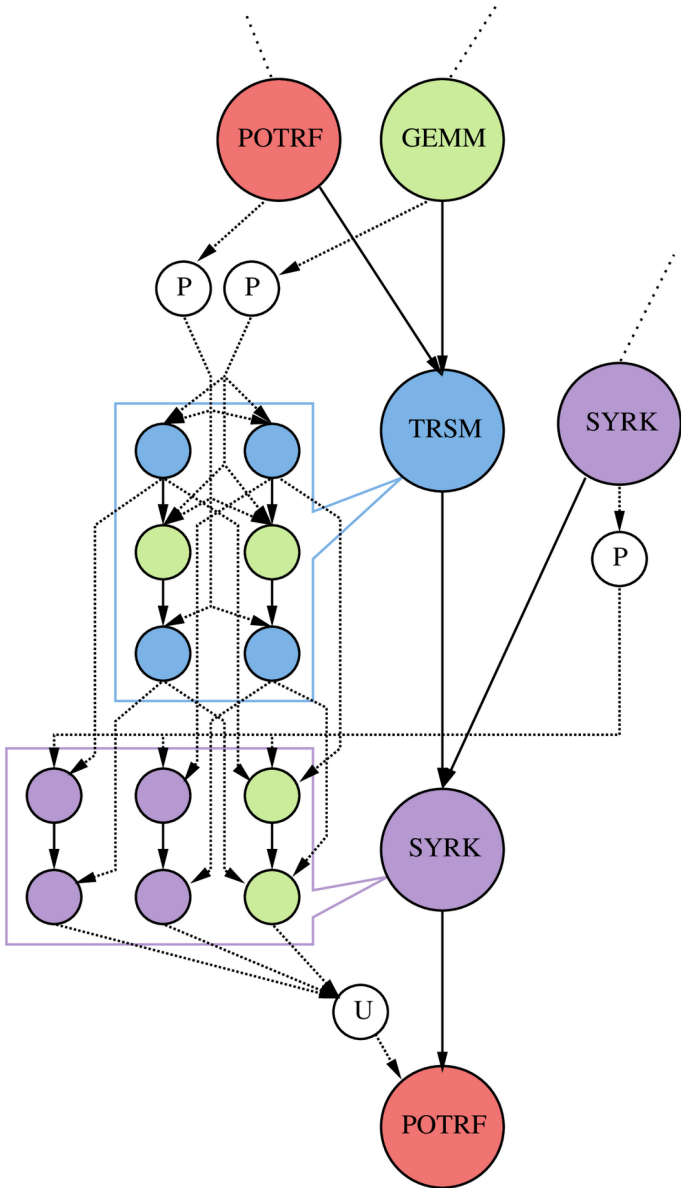


Hierarchical task graphs

Multi-level data coherency

- Synchronization pseudo-tasks
  - Only when needed
- Result is exactly as appropriate

→ Just split at will!



Dividing tasks

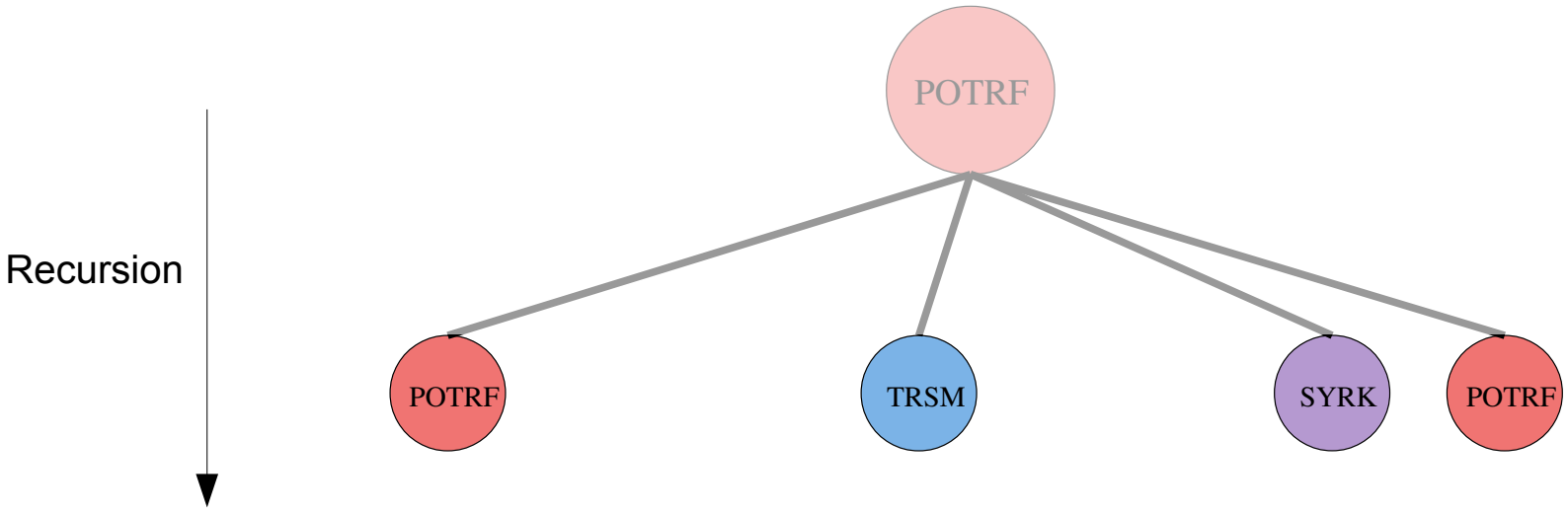
No extra synchronization

→ Can consider task graph subdivision as a tree

Dividing tasks

No extra synchronization

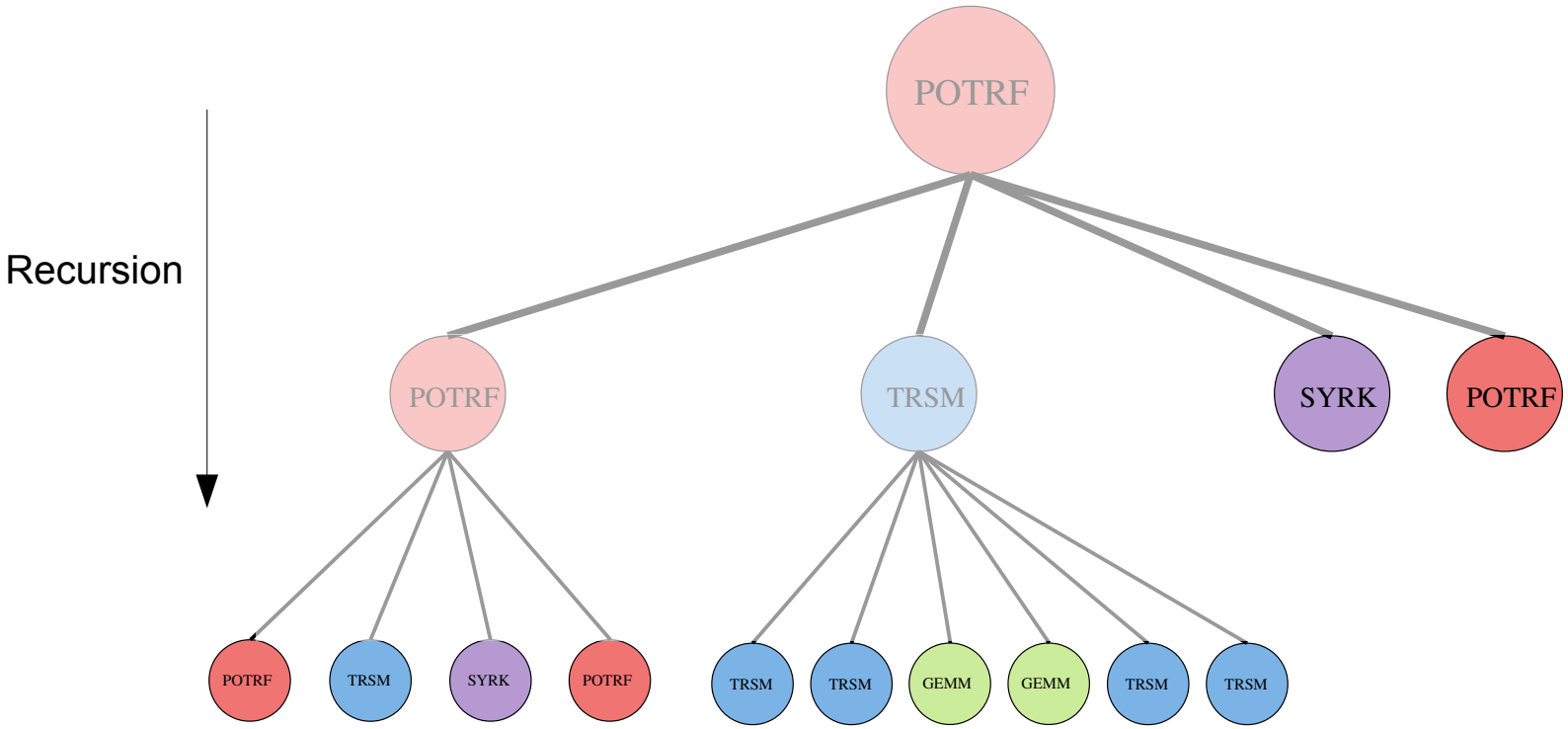
→ Can consider task graph subdivision as a tree



Dividing tasks

No extra synchronization

→ Can consider task graph subdivision as a tree

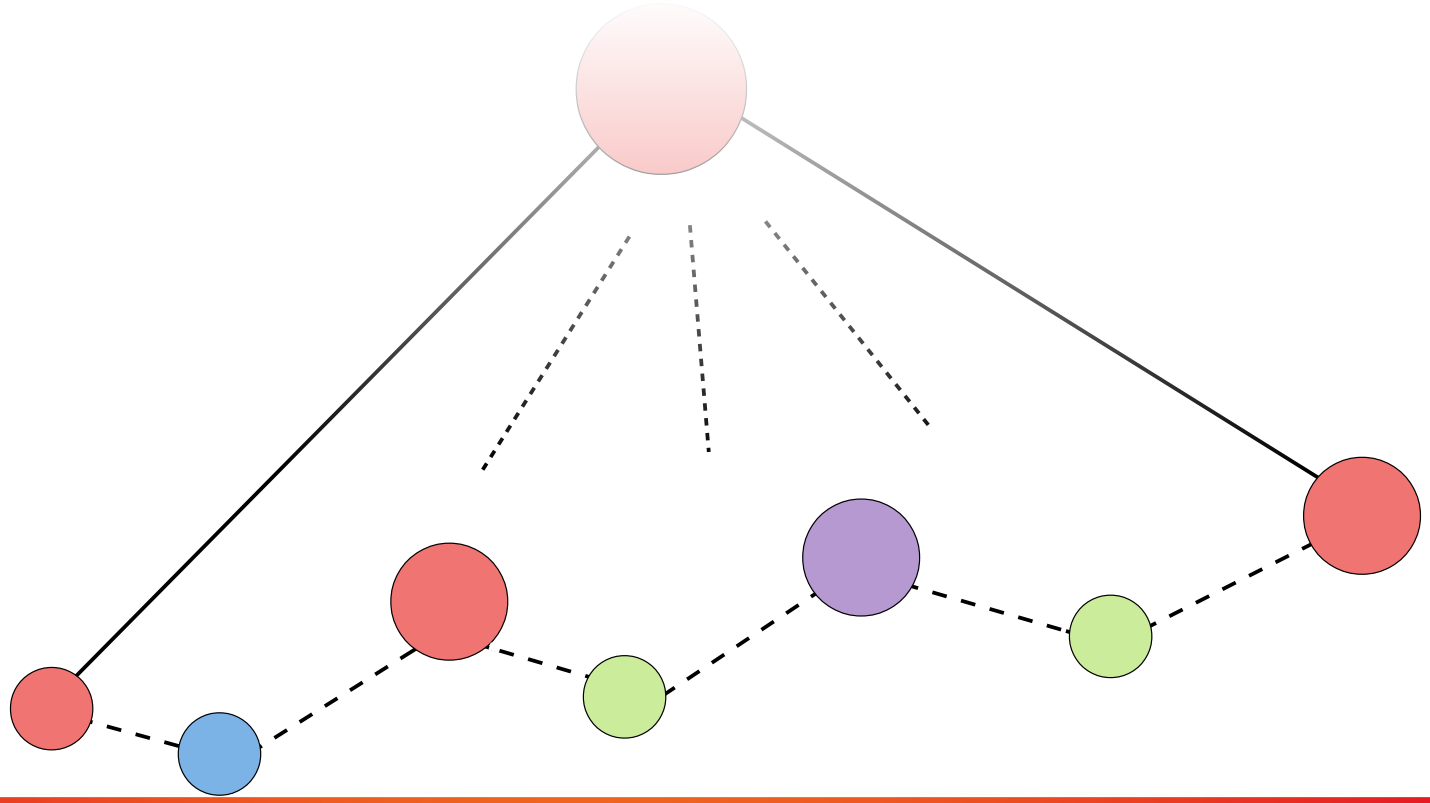


Dividing tasks

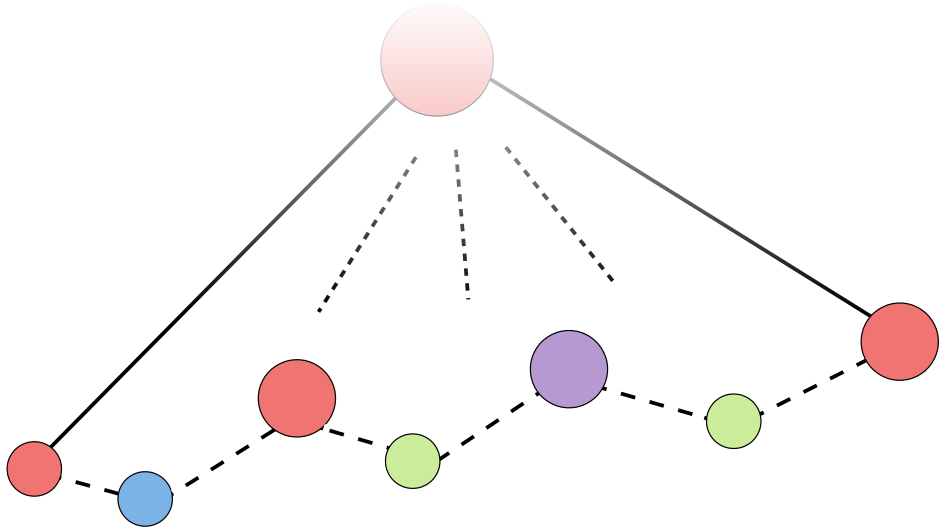
No extra synchronization

→ Can consider task graph subdivision as a tree

→ Decide at will where and when to stop recursing



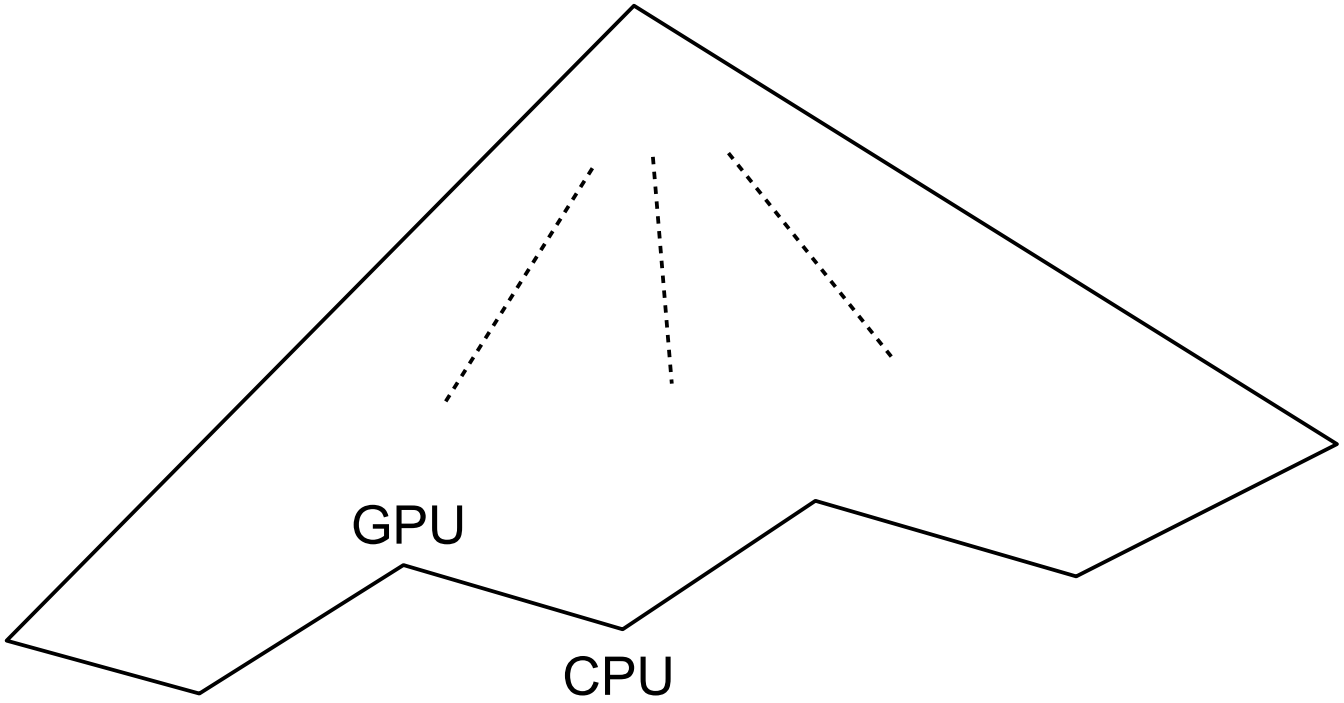
Opportunities



Opportunities

GPU / CPU efficiency management

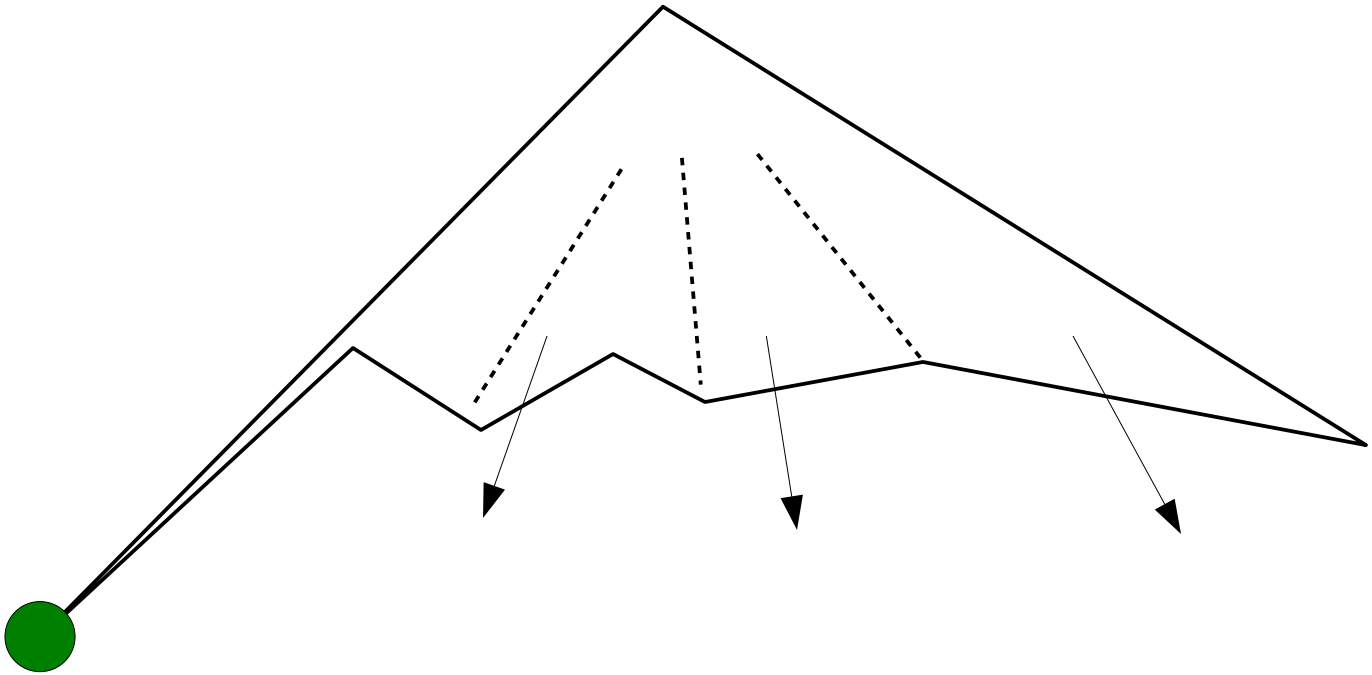
- Stop recursing at desired tile size
- Care for latency of the task graph critical path



Opportunities

Parallel submission

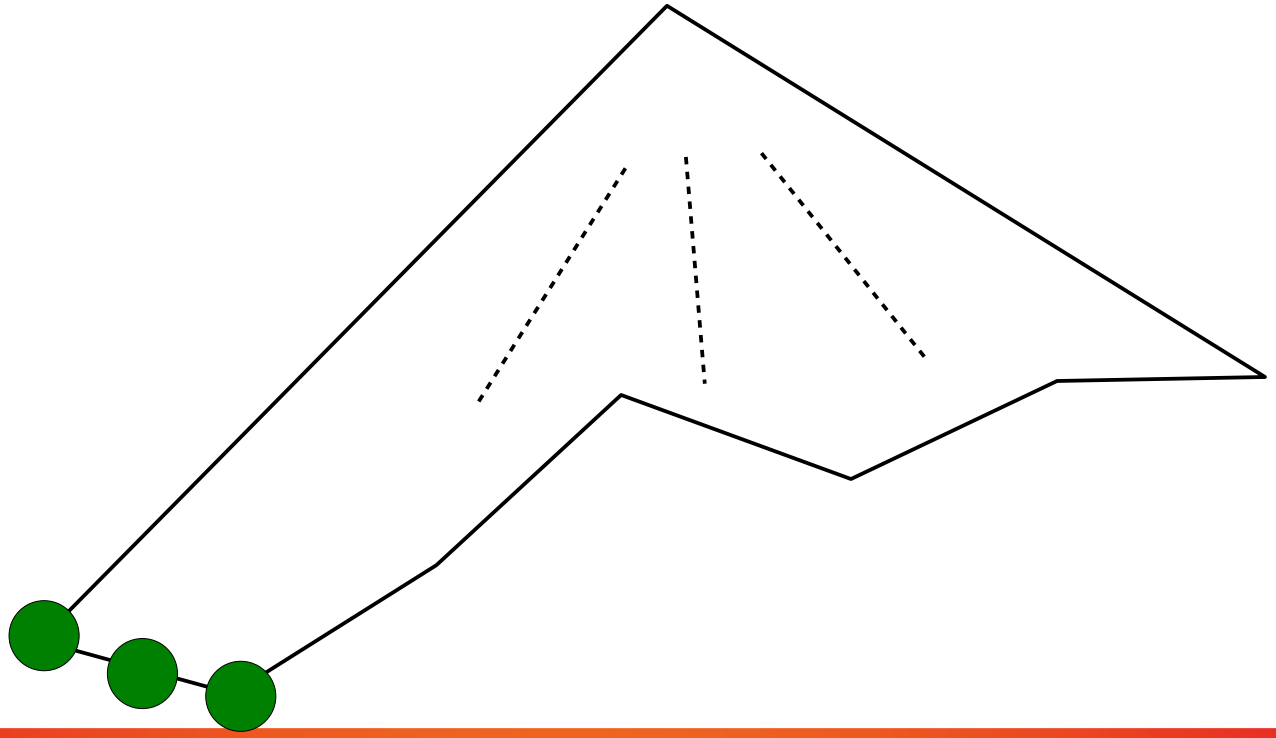
- Unroll the graph in parallel
- While one PU is computing the first task



Opportunities

Delay unrolling

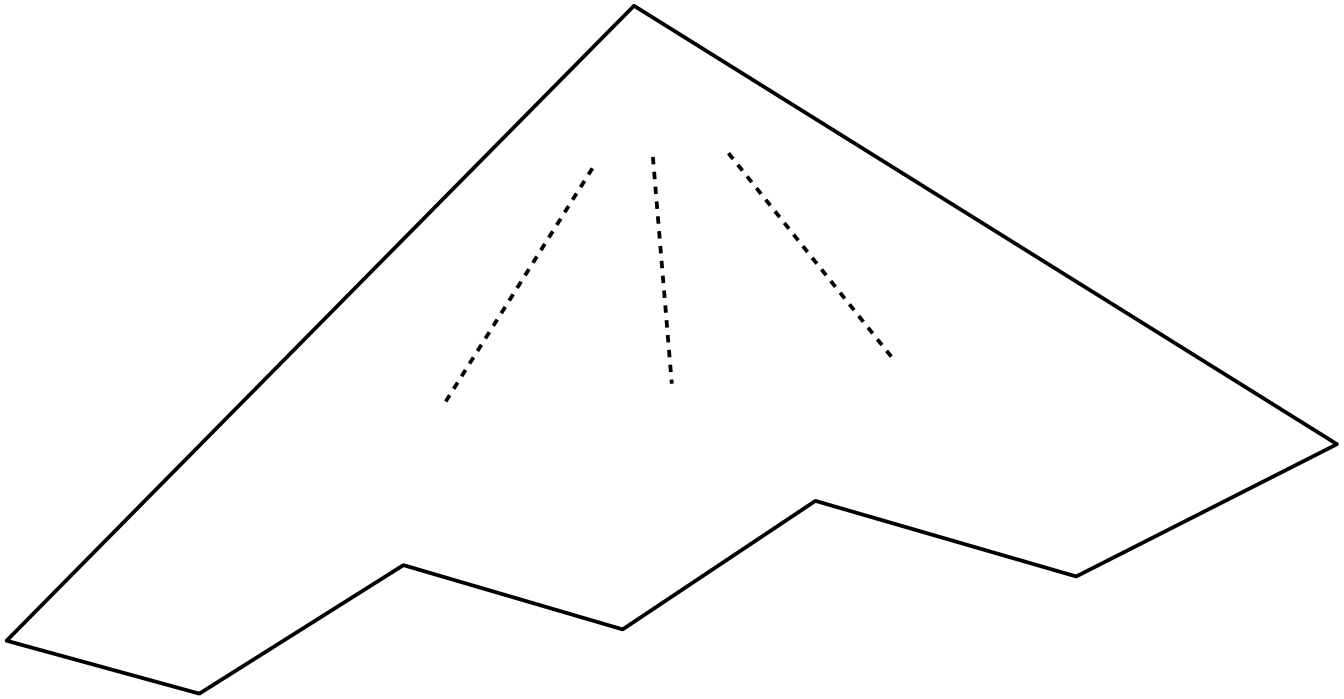
- Observe behavior before unrolling the rest accordingly
- Decorrelate submission and execution



Opportunities

Delay unrolling

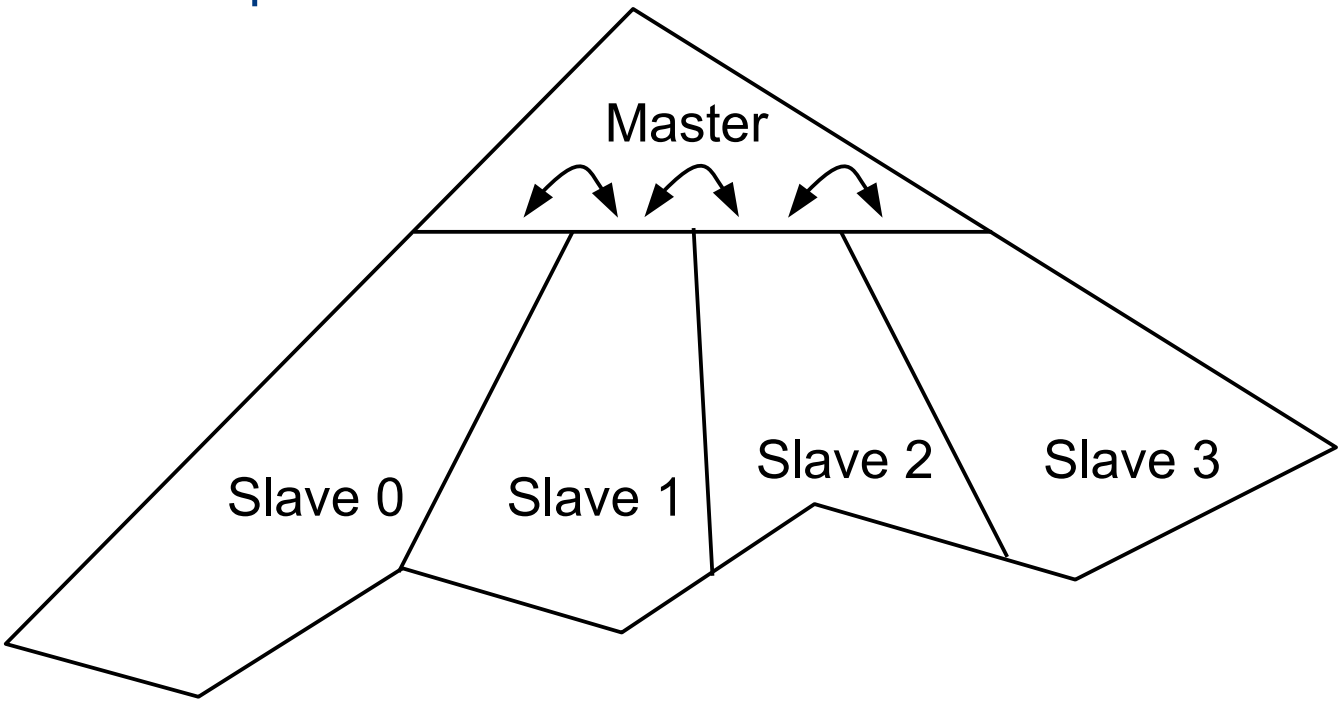
- Observe behavior before unrolling the rest accordingly
- Decorrelate submission and execution



Opportunities

Scaling at large

- Master unrolls higher recursion levels, schedules result
- Slaves unroll the rest
- Master still contention point

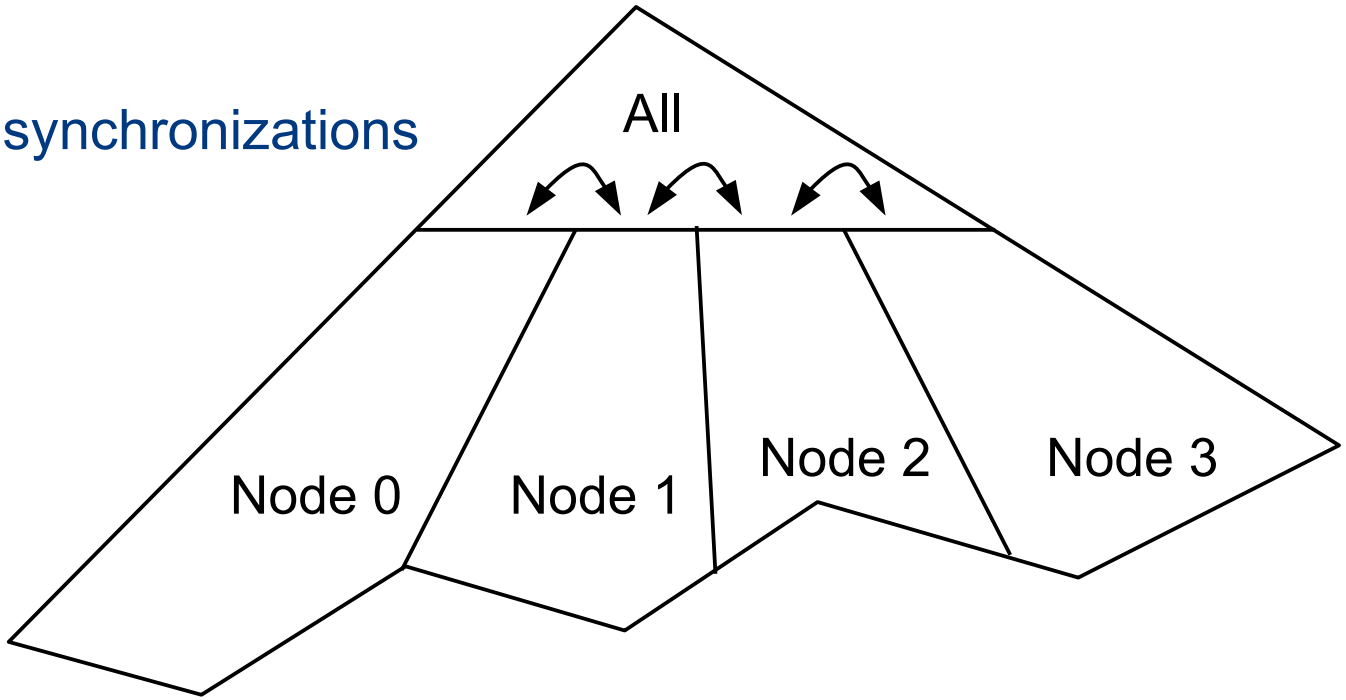


Opportunities

Scaling at large

- All nodes unroll higher recursion levels, determine task mapping
- Nodes unroll their own remaining recursion
- Network communication quite coarse

  → Spurious synchronizations

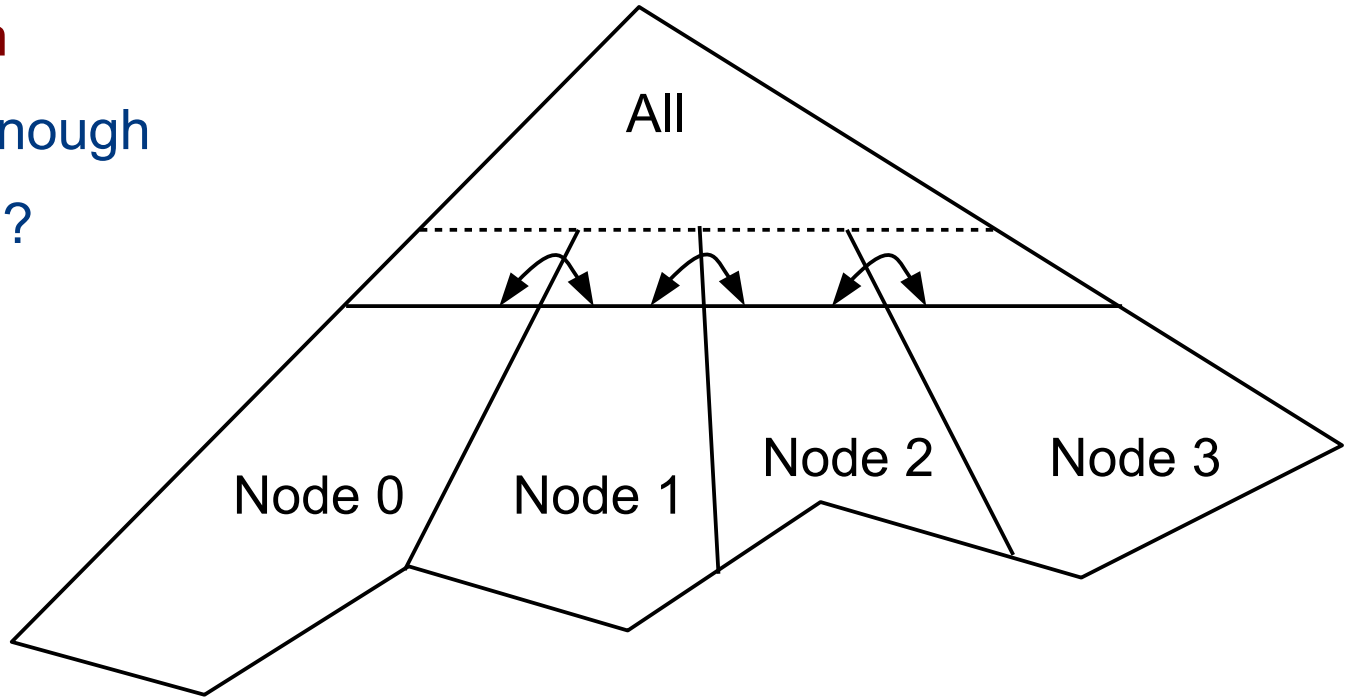


Opportunities

Scaling at large

- All nodes unroll higher recursion levels, determine task mapping
- Nodes unroll their own remaining recursion
- Network communication at finer grain
- Fine-grain enough

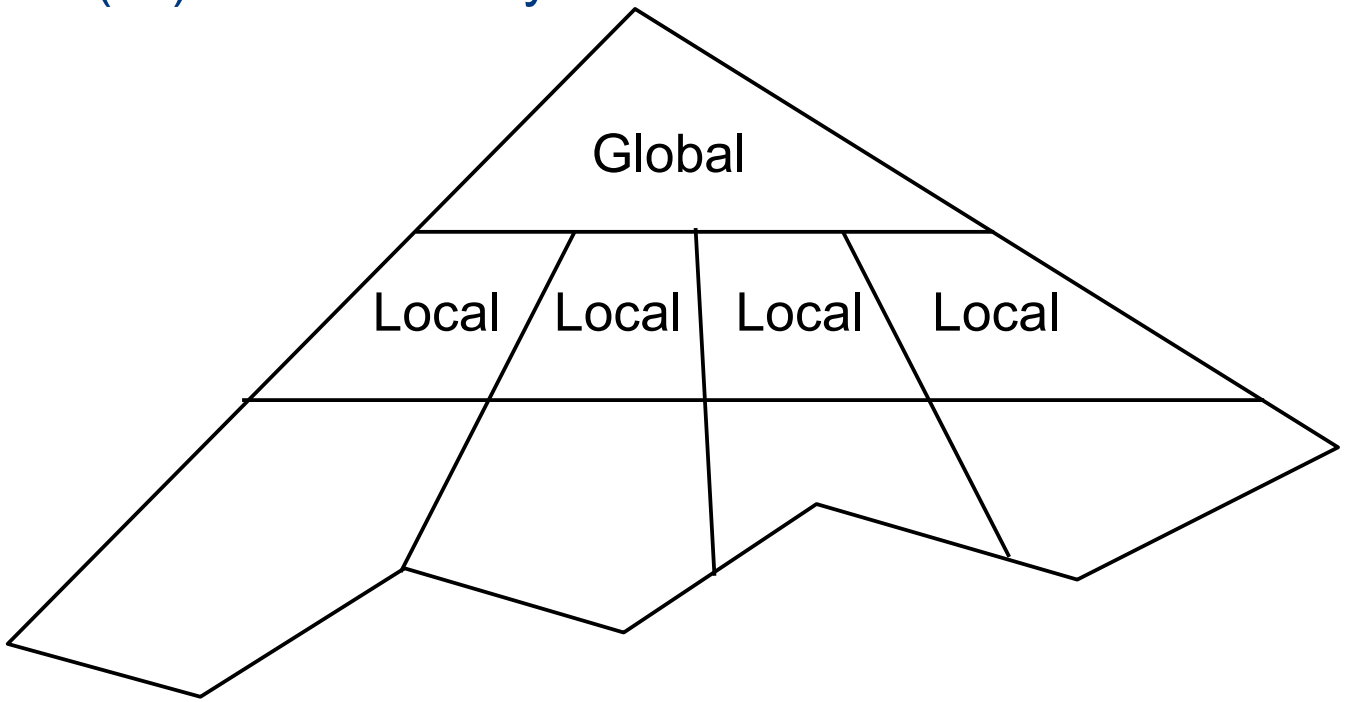
  vs overhead?



Opportunities

Scheduling at large

- High-level global task graph scheduling, e.g. mapping, critical path
- High-level local task graph scheduling, e.g. memory-based ordering
- $O(N^2)$ or even $O(N^3)$ not that costly



Opportunities

- Computation efficiency
- Scaling at large
- Scheduling at large

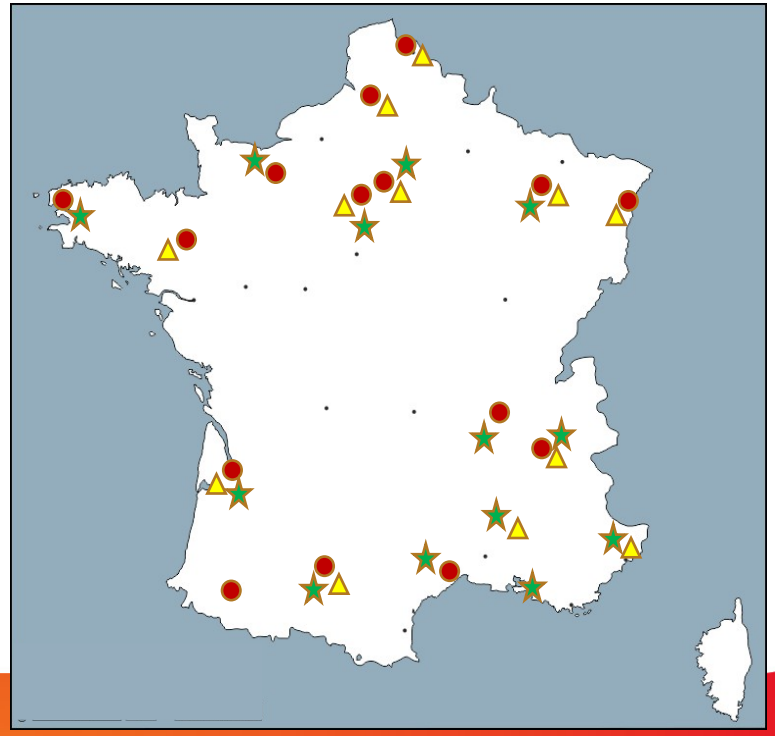
NumPEx PEPR project

NumPEx project

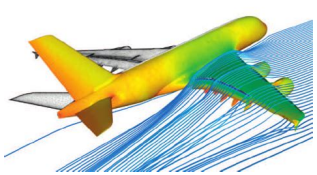
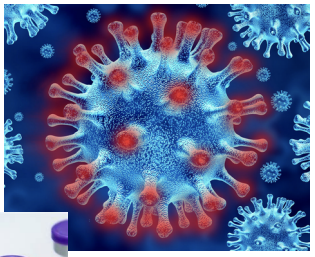
Building a national ecosystem for the future exaflops machine hosted in France

- Facilitate conception, deployment, and monitoring of composite applications at large scale
- Efficiently exploit a heterogeneous architecture with many accelerators
- Control the energy envelope

80 teams in France



NumPEx project






PC1-ExaMA

Methods/Algorithms for the Exascale

PC2-ExaSoft

HPC softwares/tools for the Exascale

PC5-ExaDIP

Application co-design Integration and productivity

PC3-ExaDost

Data-oriented softwares/tools for the Exascale

PC4-ExaATaW

Architectures/tools for large-scale workflows

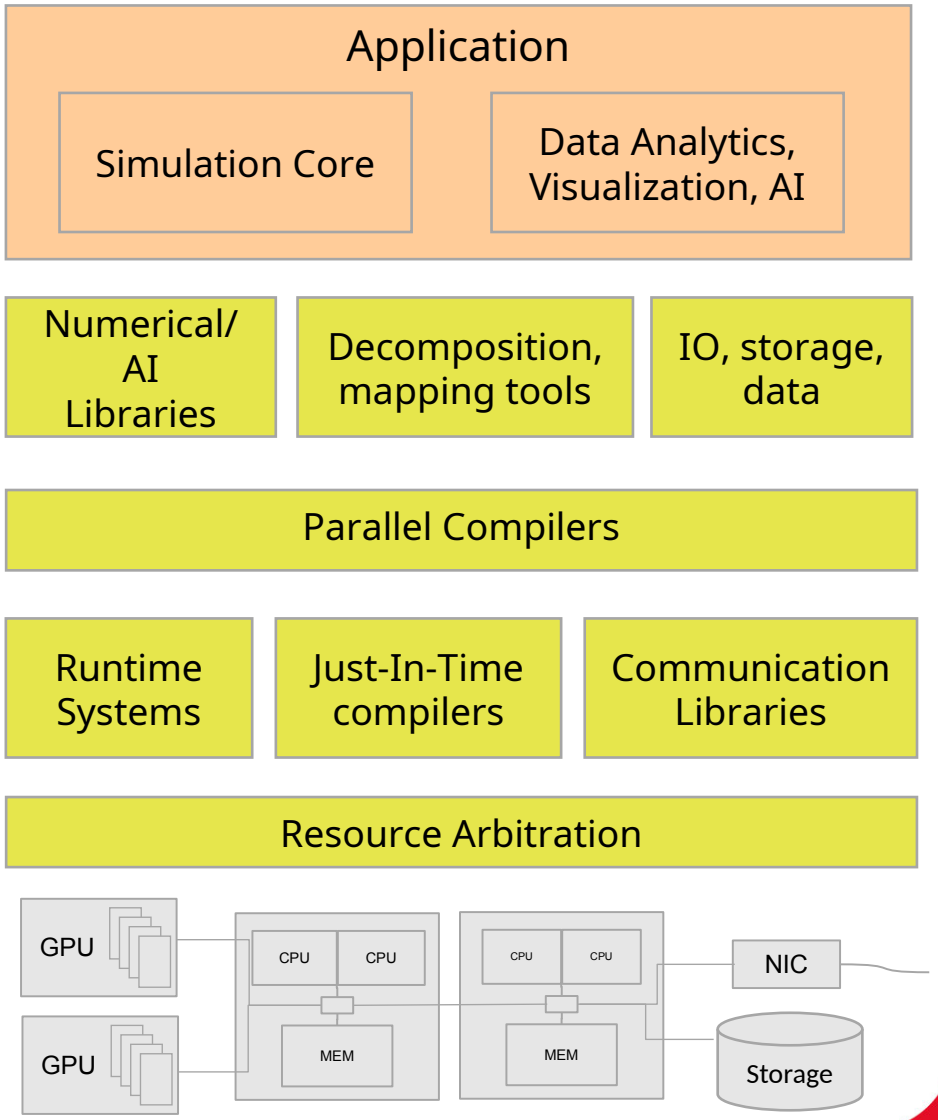
A Software Forge

Integration and productivity

NumPEx project

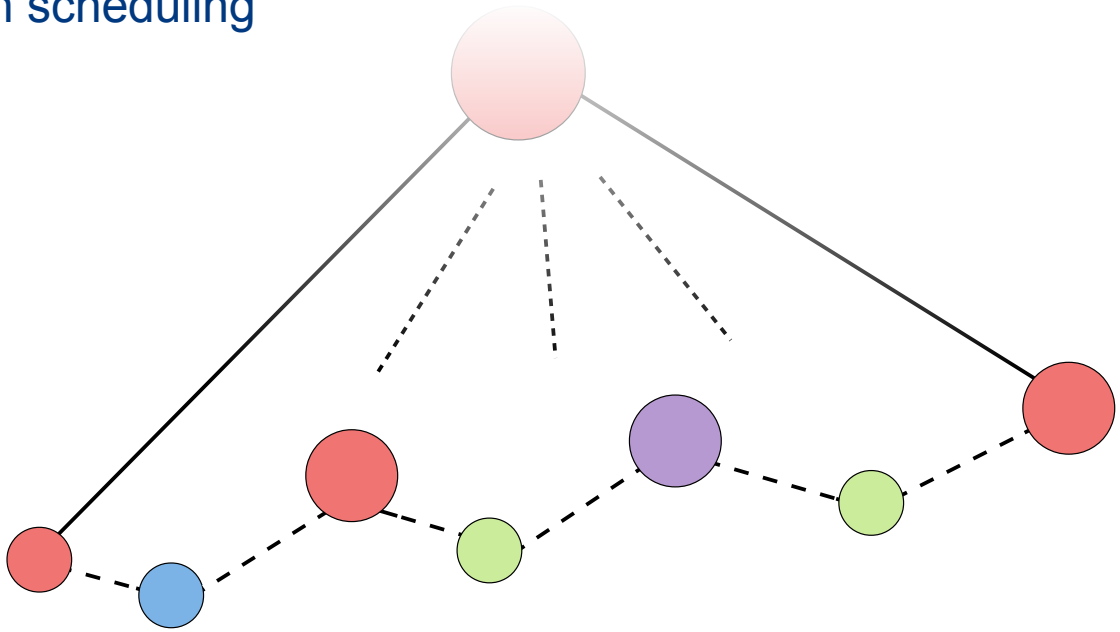
PC2-ExaSoft

- Holistic approach
  - Contribute to a sound, consistent software stack
  - Most components should fit together!
  - Bridge the gap between existing languages/software/tools
  - Integrate state-of-the-art research results
  - Demonstrate relevance on representative applications
- WP3 : Runtime Systems at Exascale



Conclusion

- Programming large systems requires abstraction
- Recursive applications are there
- Hierarchical task graphs seems an opportunity for
  - Controlling task size
  - Controlling task submission
  - High-level task graph scheduling



The StarPU runtime system

Development context

- History
  - Started in 2008
    - PhD Thesis of Cédric Augonnet
  - StarPU main core ≈ 70k lines of code
  - Written in C
- Open Source
  - Released under LGPL
  - Sources freely available
    - git repository and nightly tarballs
    - See https://starpu.gitlabpages.inria.fr/
  - Open to external contributors
- [HPPC'08]
- [Europar'09] – [CCPE'11],... >1500 citations

Features

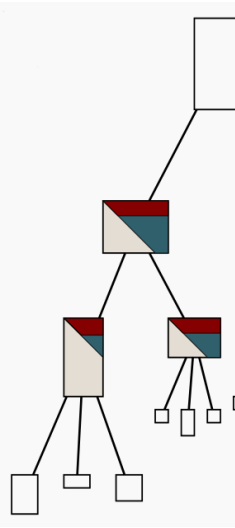
- C, OpenMP, OpenCL APIs
- CPUs, GPUs, Phi
- Advanced task mapping & scheduling
- Optimized data transfers
- Cluster Support through MPI
- Out-Of-Core support
- Simulation support
- Performance analysis tools

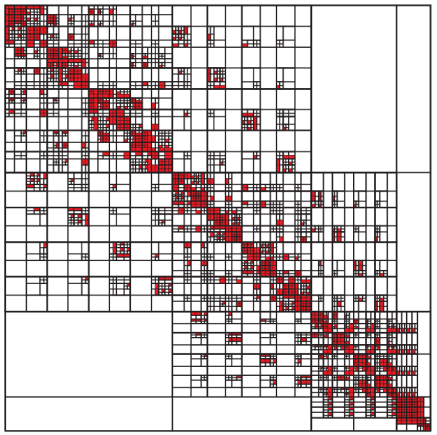
The StarPU runtime system

Success stories

Task-based programming actually makes things easier!

- QR-Mumps (sparse linear algebra)
  - Non-task version: only 1D decomposition
  - Task version: 2D decomposition, flurry of parallelism
    - With seamless memory control
- H-Matrices (compressed linear algebra, AirBus)
  - Out-of-core support
    - Could run cases unachievable before
    - e.g. 1600 GB matrix with 256 GB memory
  - Shipped to AirBus customers
- Implemented CFD, FMM, CG, stencils, …





The StarPU runtime system

Supported platforms

- Supported architectures
  - Multicore CPUs (x86, PPC, ...)
  - NVIDIA GPUs
  - OpenCL devices (eg. AMD cards)
  - HIP
  - FPGA (ongoing)
  - Old Intel Xeon Phi (MIC), SCC, Kalray MPPA, Cell (decommissioned)
- Supported Operating Systems
  - Linux
  - Mac OS
  - Windows

Applications on top of StarPU

Using CPUs, GPUs, distributed, out of core, ...

- Dense linear algebra
  - Cholesky, QR, LU, ... : Chameleon (based on Plasma/Magma)
- Sparse linear algebra
  - QR_MUMPS
  - PaStiX
- Compressed linear algebra
  - BLR, h-matrices
- Fast Multipole Method
  - ScalFMM
- Conjugate Gradient
- Other programming models : Data flow, skeletons
  - SignalPU, SkePU
- ...