Programmation par tâches et ses applications

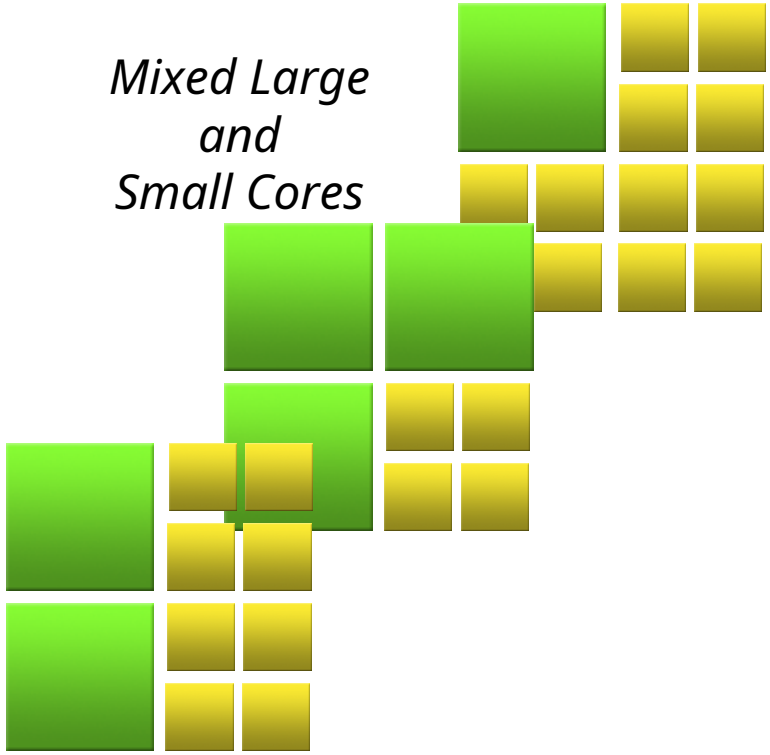
Samuel Thibault

INRIA Bordeaux - Sud-Ouest -- STORM Team

Introduction

Toward heterogeneous multi-core architectures

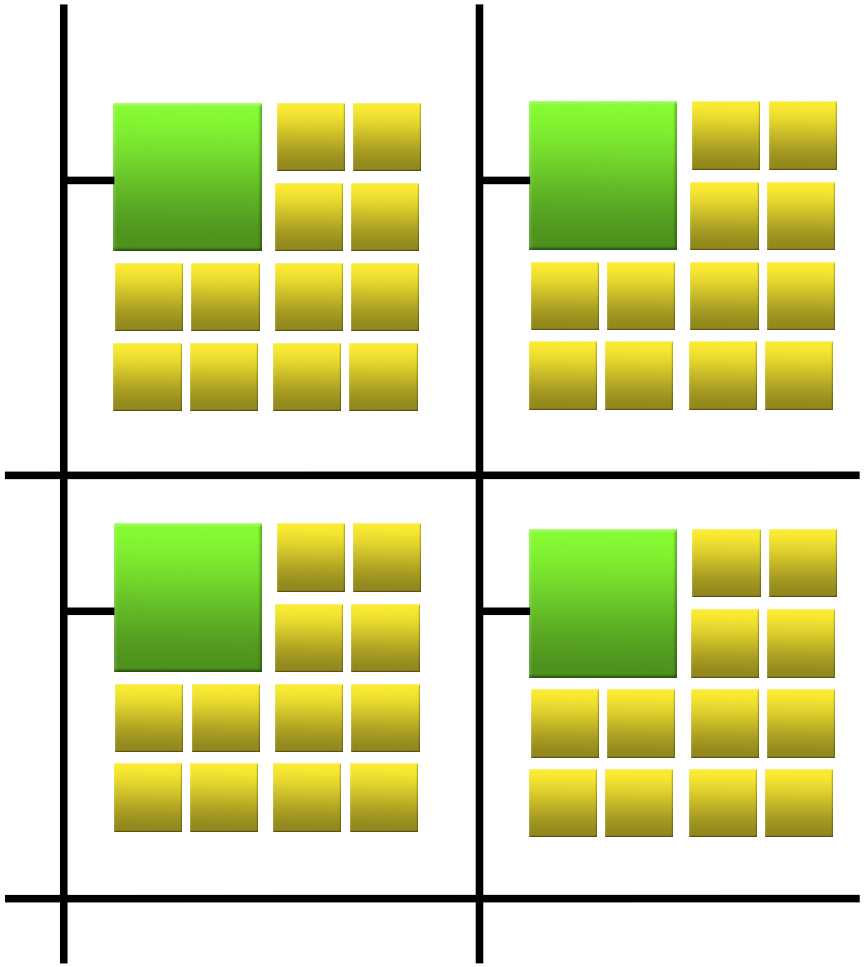
- Multicore is here
  - Hierarchical architectures
  - Manycore
  - Heterogeneous systems
- Architecture specialization
  - Now
    - Accelerators (GPGPUs, FPGAs)
  - In the near Future
    - Many simple cores
    - A few full-featured cores



Introduction

Toward heterogeneous multi-core clusters

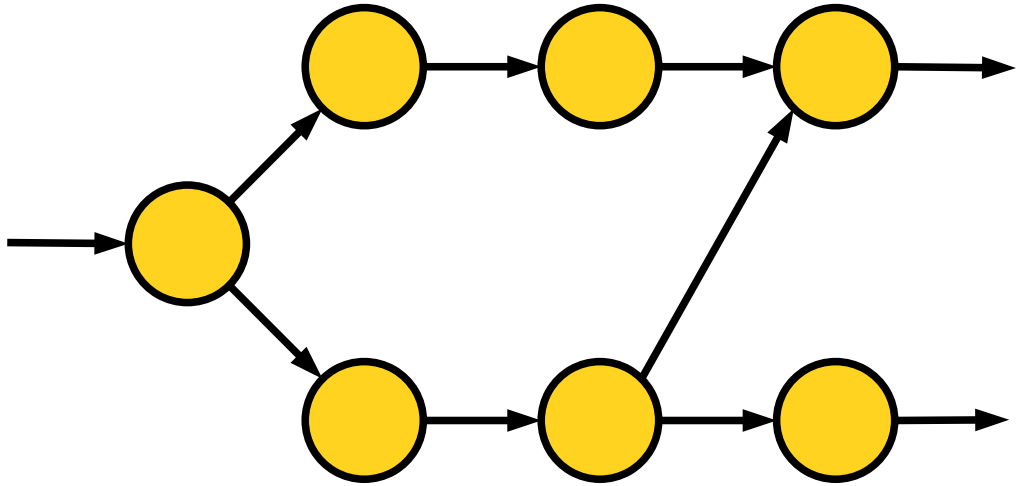
- Multicore is here
  - Hierarchical architectures
  - Manycore
  - Heterogeneous systems
- Clusters thereof
  - High-speed network
  - Network topology
  - Towards exascale



Task graphs

- Well-studied for scheduling parallelism (since 60's!)
- But only recent trend in HPC
- Departs from usual sequential programming

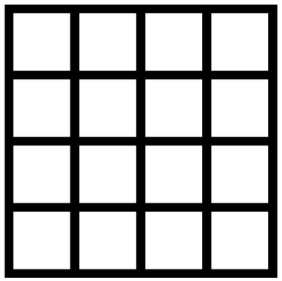
Really ?



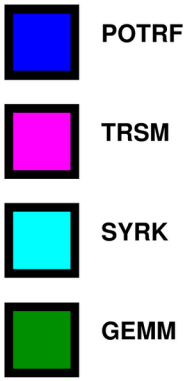
Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)



```
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();
```

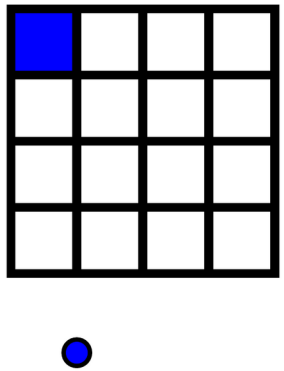


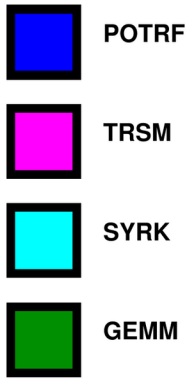
Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```



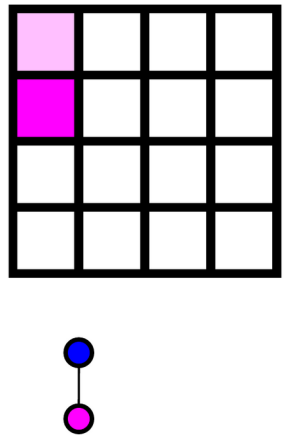


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

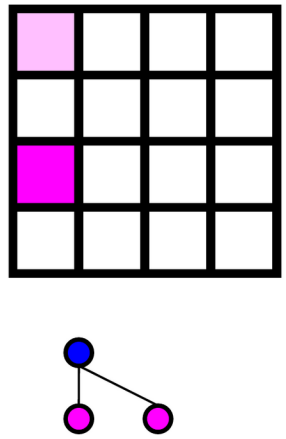


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

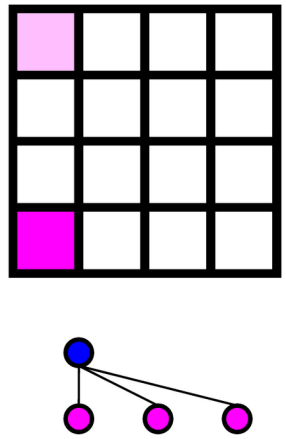


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

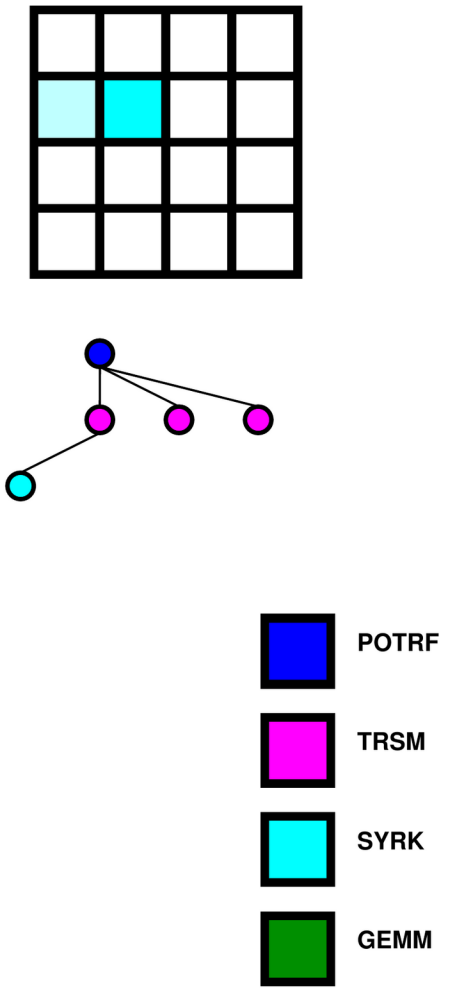


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

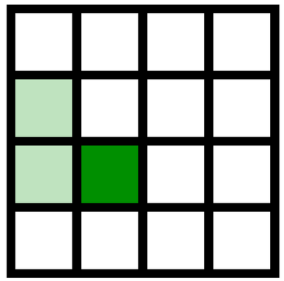


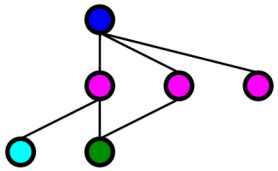
Expressing a task graph

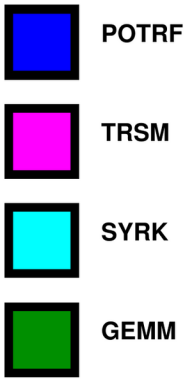
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```





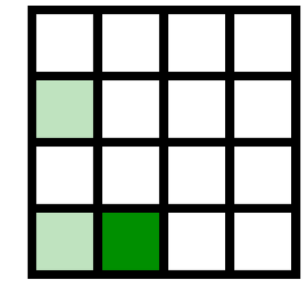


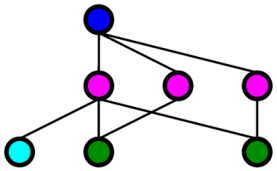
Expressing a task graph

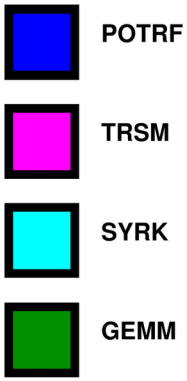
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```





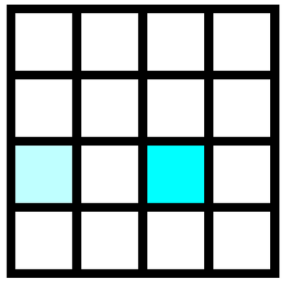


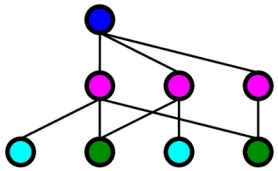
Expressing a task graph

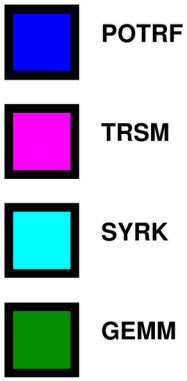
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```





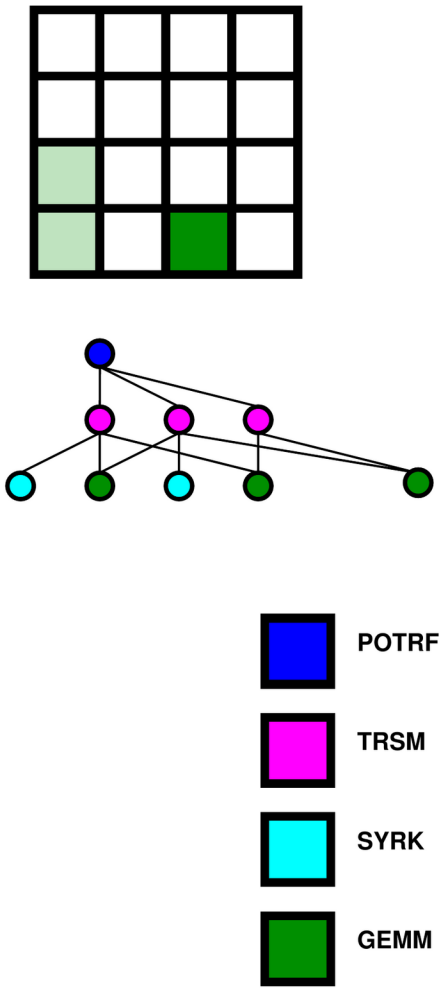


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

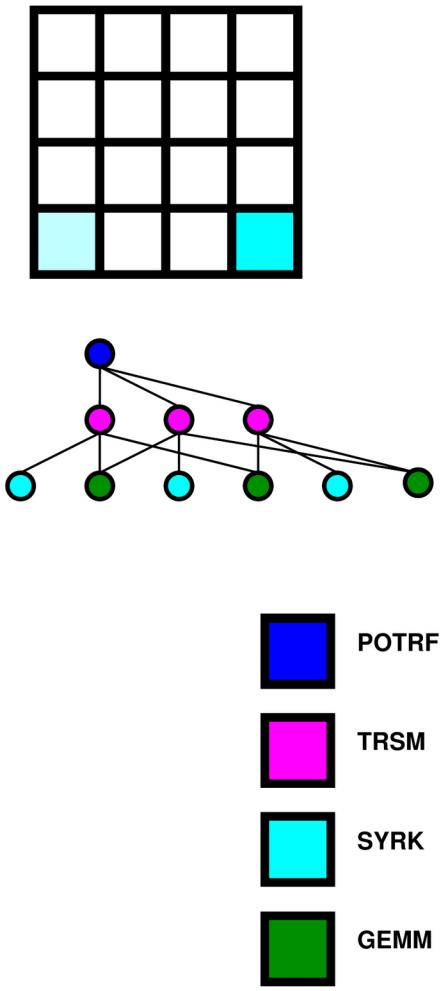


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

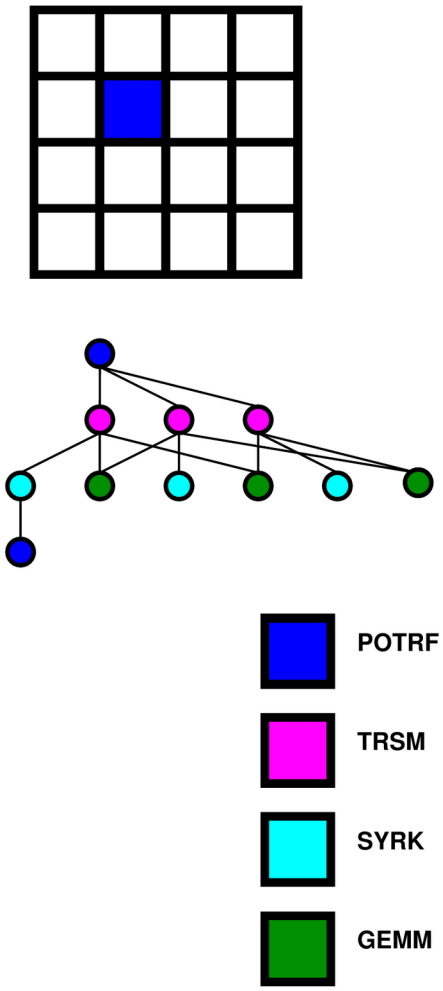


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```

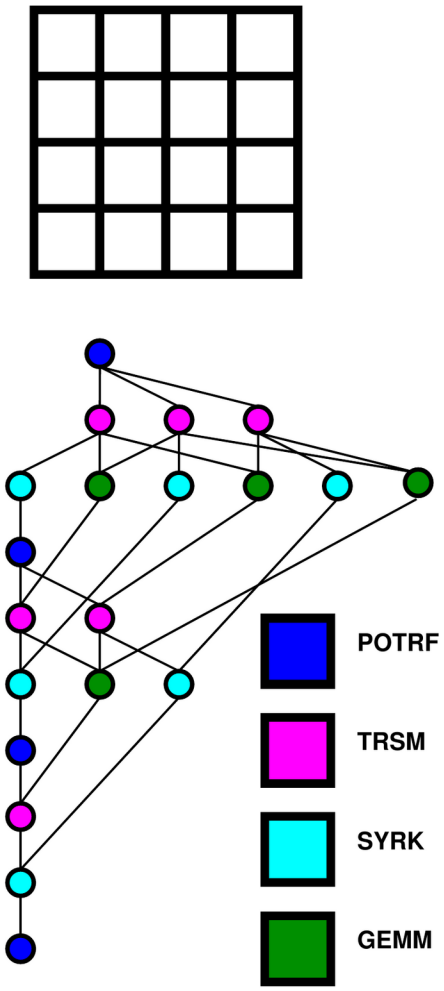


Expressing a task graph

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
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();
```



Write your application as a task graph

Even if using a sequential-looking source code

➔ Portable performance

Sequential Task Flow (STF)

- Algorithm remains the same on the long term
- Can debug the sequential version.
- Only kernels need to be rewritten
  - BLAS libraries, multi-target compilers
- Runtime will handle parallel execution

Task-based programming

- Needs code restructuring
  - Split computation into tasks
    - BLAS, typically
    - Supposed to have “stable” performance
- Constraining
  - No global variables
    - Mandatory for GPUs
- Actually… functional programming

So a good move, in the end ☺

- Have to accept constraints and losing control

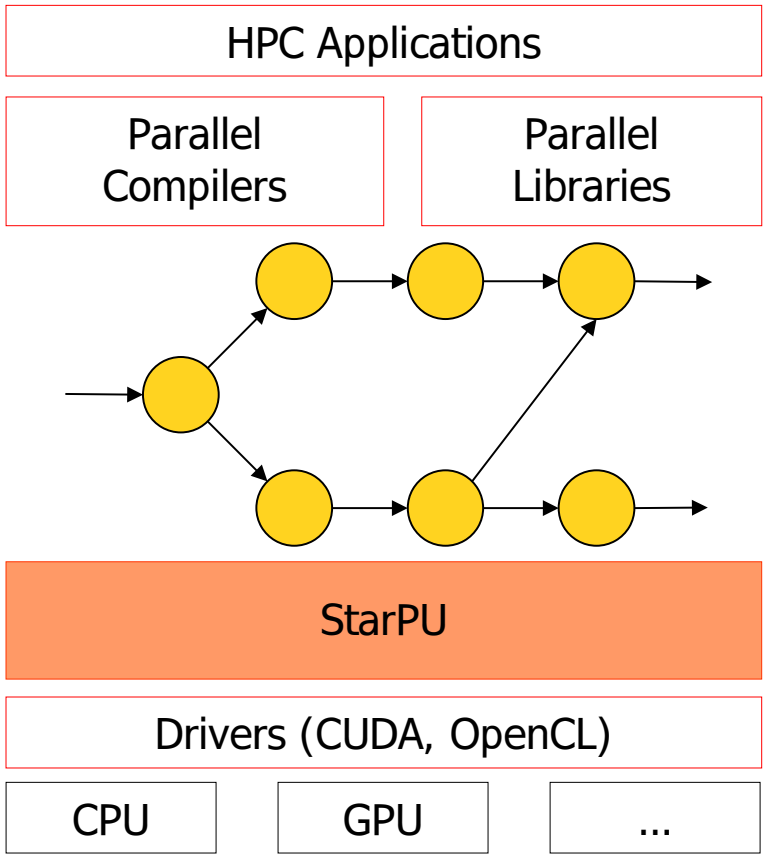
Just like we did when moving from assembly to high-level languages

Overview of StarPU

The StarPU runtime system

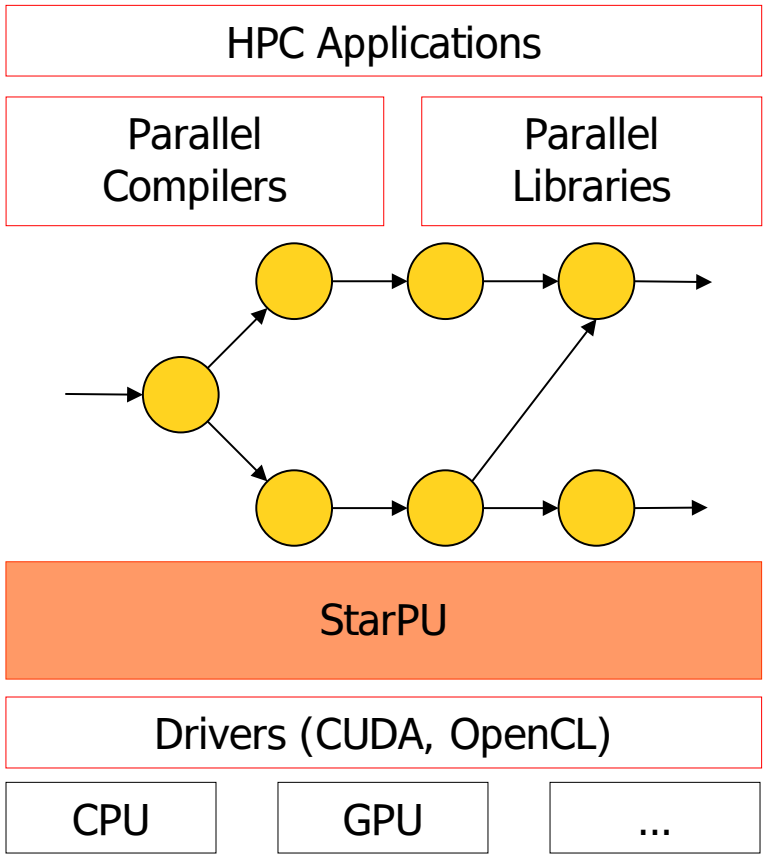
The need for runtime systems

- "do dynamically what can't be done statically anymore"
- Compilers and libraries generate (graphs of) tasks
  - Additional information is welcome!
- StarPU provides
  - Task scheduling
  - Memory management



Data management

- StarPU provides a Virtual Shared Memory (VSM) subsystem (aka DSM)
  - Replication
  - Consistency
  - Single writer
    - Or reduction, ...
- Input & ouput of tasks = reference to VSM data

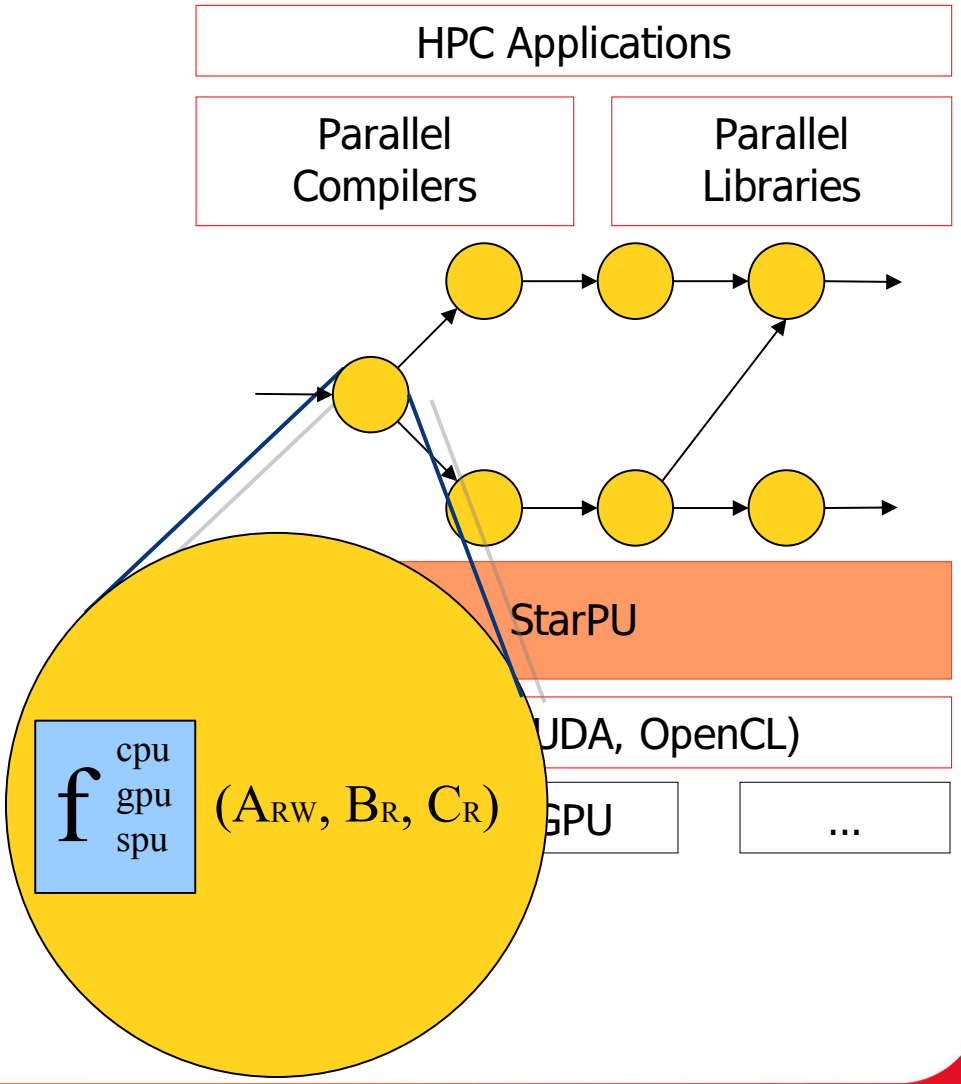


The StarPU runtime system

Task scheduling

- Tasks =
  - Data input & output
    - Reference to VSM data
  - Multiple implementations
    - E.g. CUDA + CPU implementation
  - Non-preemptible
  - Dependencies with other tasks

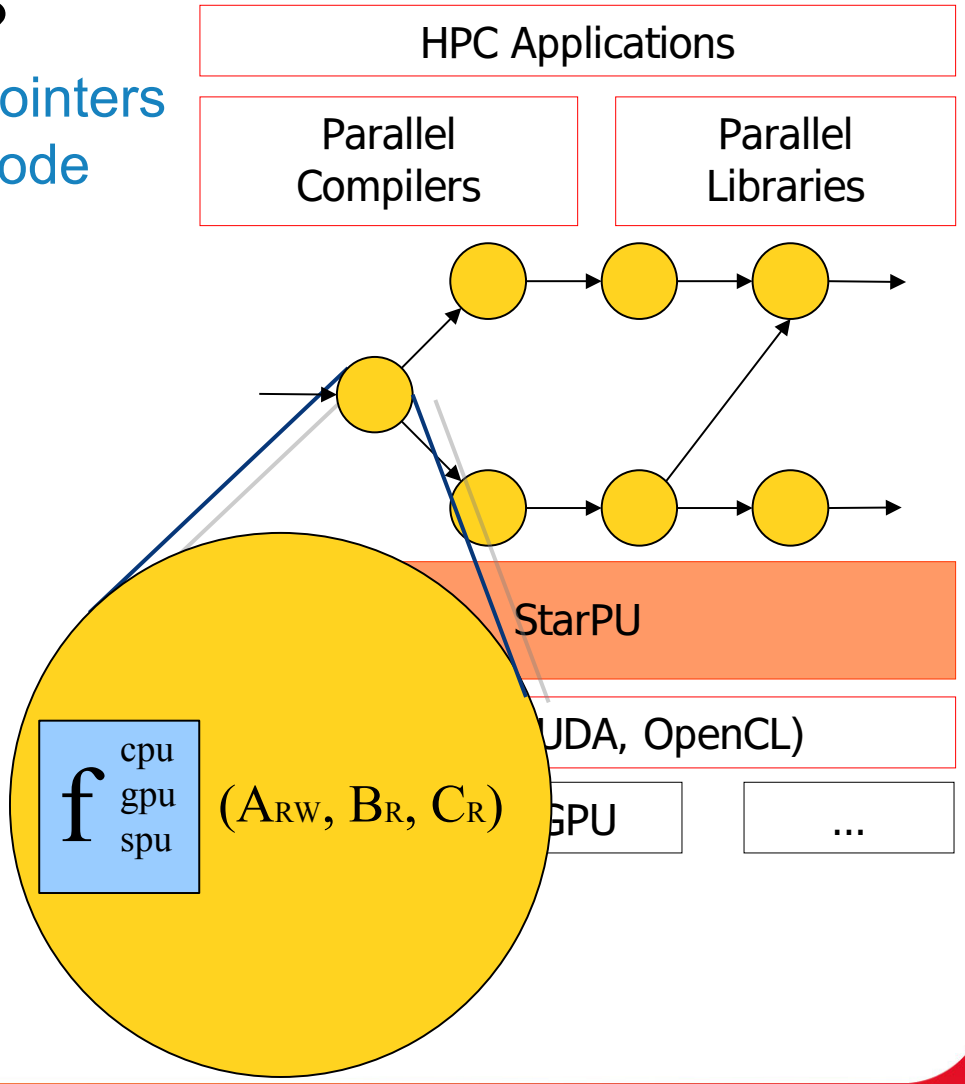
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



The StarPU runtime system

Task scheduling

- Who generates the code ?
  - StarPU Task ~= function pointers
  - StarPU doesn't generate code
- Libraries era
  - PLASMA + MAGMA
  - FFTW + CUFFT…
  - Variants management
- Rely on compilers
  - E.g. Kokkos



The StarPU runtime system

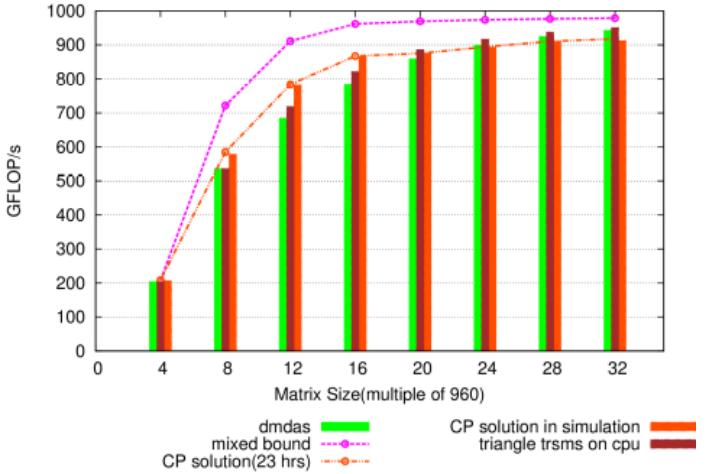
Supported platforms

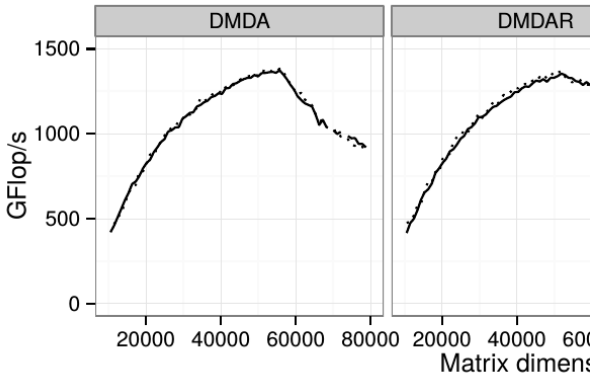
- Supported architectures
  - Multicore CPUs (x86, PPC, ...)
  - NVIDIA GPUs
  - OpenCL devices (eg. AMD cards)
  - HIP
  - SYCL (being merged)
  - FPGA (ongoing)
  - Old Intel Xeon Phi (MIC), SCC, Kalray MPPA, Cell (decommissioned)
- Supported Operating Systems
  - Linux
  - Mac OS
  - Windows

The StarPU runtime system

Task-based support

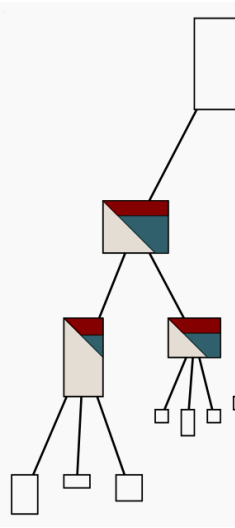
- Task/data scheduling
  - Pipelining
  - Load balancing
  - GPU memory limitation management
  - Data prefetching
- Distributed execution
- Out-of-core: optimized swapping to disk
- High-level performance analysis
- Debugging sequential execution
- Performance bounds
- Reproducible performance simulation





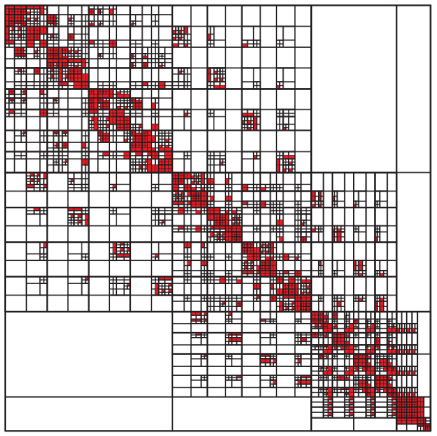
The StarPU runtime system

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, …



Applications on top of StarPU

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

- Dense linear algebra
  - Cholesky, QR, LU (without/with partial pivoting), SUMMA, ... : Chameleon
- Sparse linear algebra
  - QR_MUMPS
  - PaStiX
- Compressed linear algebra
  - BLR, h-matrices
- Fast Multipole Method
  - ScalFMM, Composyx
- Conjugate Gradient
- Other programming models : Data flow, skeletons
  - SignalPU, SkePU
- ...

Kokkos?

Parallel_for constructs

- fork/join
- Can be a limitation for irregular computation

Tasks

- Not very developped

Graphs

- Similar to cuda graphs : record graph of tasks

- Koktails project plans to interface with StarPU
  - Write application main code with Kokkos
  - Schedule with StarPU
  - Write computation kernels with Kokkos (CPU/GPU transparency)

Kokkos+MPI?

Explicit MPI calls

- Can be overlapped by using MPI_Isend
- But requires managing synchronization by hand
  - Kokkos-comm under way to automatize synchronization

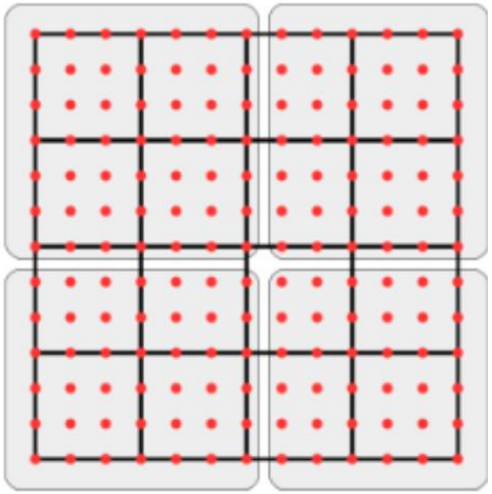
Vs task-based graphs

- Either explicit taskified-MPI calls
  - Integrated within the task graph
  - Will automatically synchronize with other tasks, optimize GPU-NIC, ...
- Or implicit MPI communications inferred from task graph
  - No difference between distribution to GPUs and distribution over MPI

Structured mesh applications?

Structured mesh applications?

- Domain decomposed into subdomains
- Iterations of
  - Update/compute
  - Transmit frontier (ghost cells) to neighbours

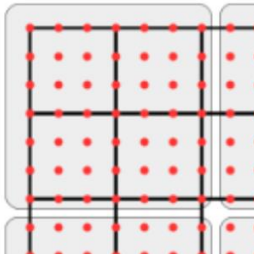


- How can it look like in task-based programming?

Structured mesh applications?

Update

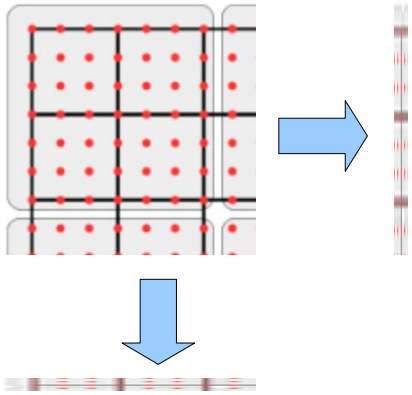
- In-place
- In-kernel
- Provide StarPU with CPU/GPU implementations



Structured mesh applications?

Extract

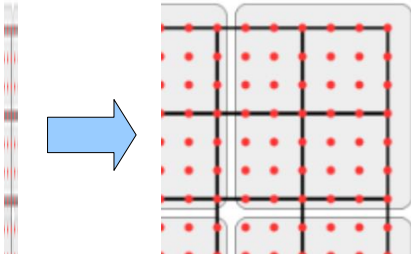
- A copy of the frontier into a small buffer

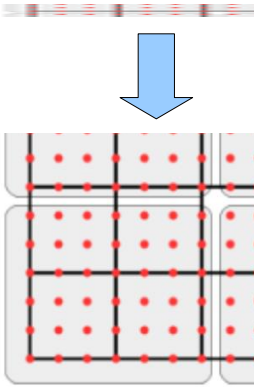


Structured mesh applications?

Import

- A copy of the frontier from the small buffer

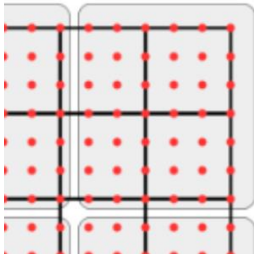


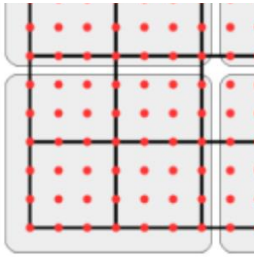


Structured mesh applications?

Update

- In-place





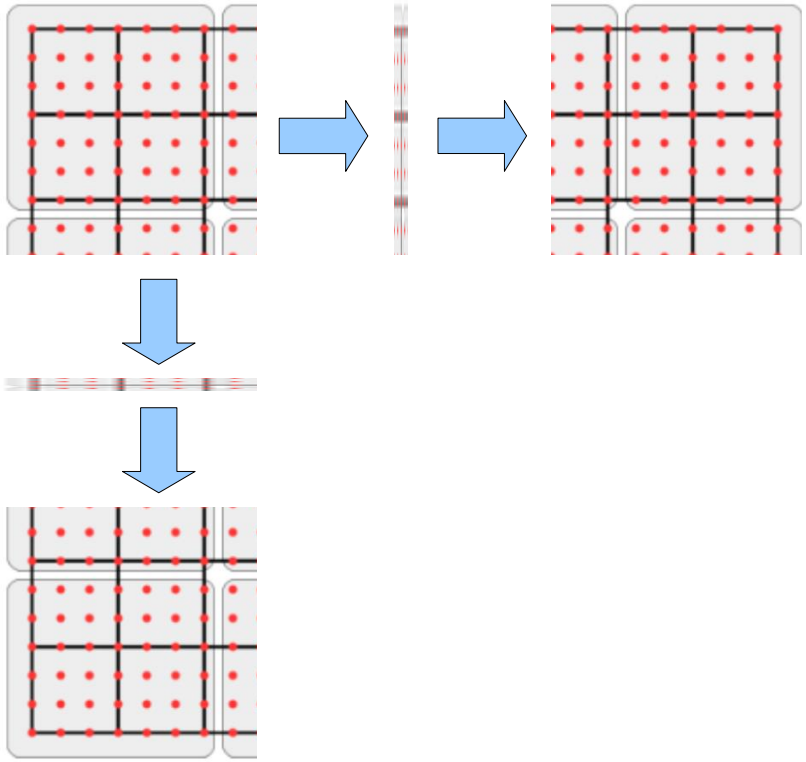
Structured mesh applications?

Update/Extract/Import/Update/...

- Will fit both multi-GPU and MPI approaches
- Asynchronous

StarPU example

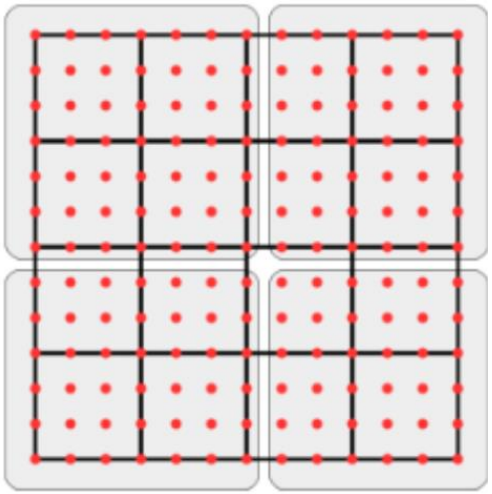
- examples/stencil



Structured mesh applications?

Subdomain decomposition

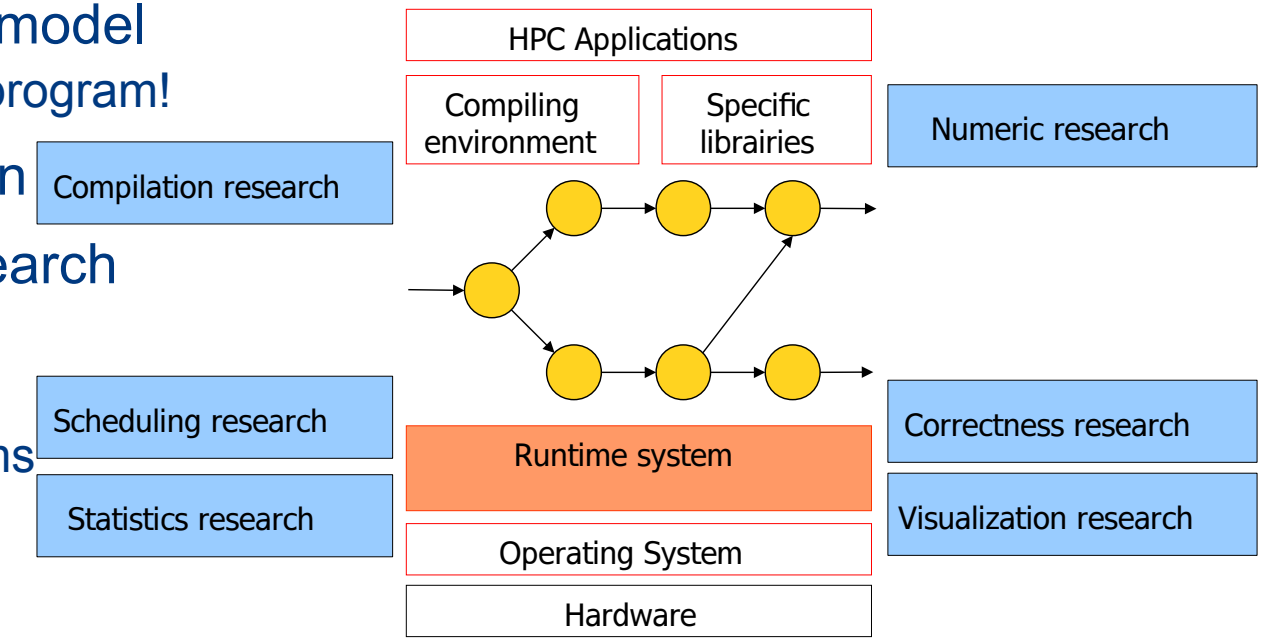
- Not too small : tasks need to be at the very least 1ms
- Not too large : expose parallelism



Conclusion

Task graphs

- Nice programming model
  - Keep sequential program!
- Optimized execution
- Playground for research
  - Runtime
  - Scheduling
  - Numeric algorithms
  - Statistics
  - Correctness
- Used for various real-world computations
  - Cholesky/QR/LU (dense/sparse/compressed), stencil, CG, CFD, FMM…



http://starpu.gitlabpages.inria.fr/tutorials/