# StarPU

## Exploiting Heterogeneous Architectures Through Task-Based Programming

Olivier Aumage
RUNTIME group
INRIA – University of Bordeaux

European CoE in Lattice Field Theory
1st Workshop – April 2014

# TEAM RUNTIME

Efficient runtime systems for parallel architectures

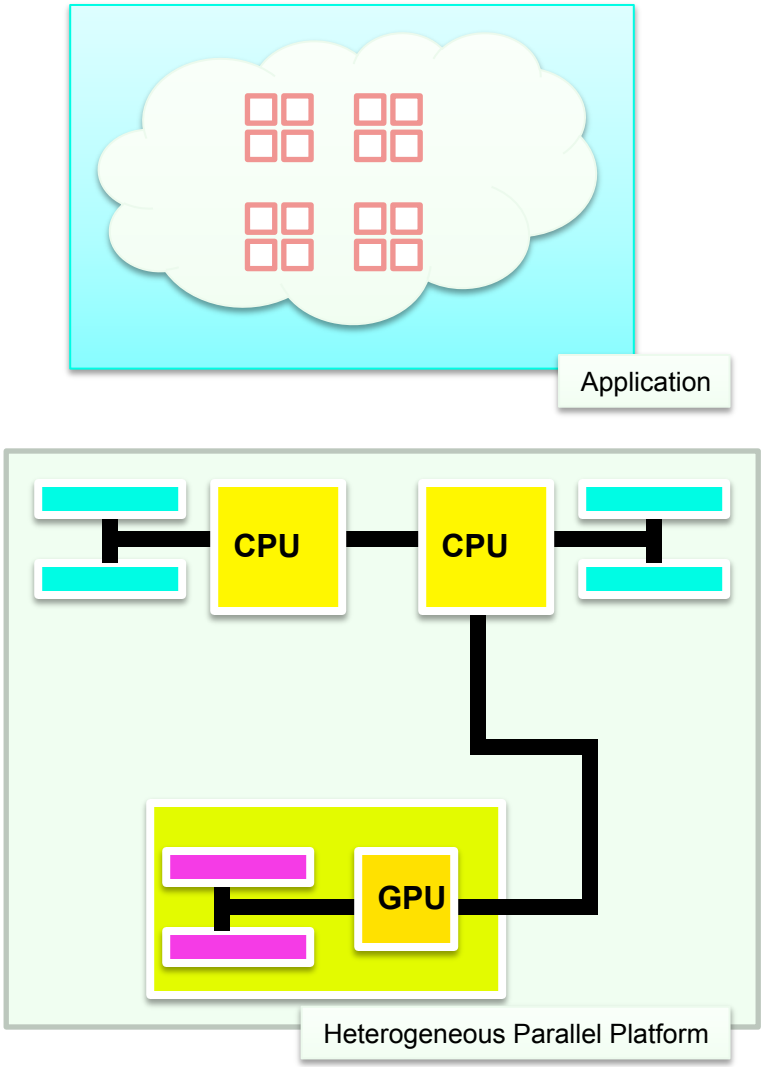
- Runtime systems for HPC
  - *Perform dynamically what cannot be done statically*
- Four directions
  - Multithreading
  - Communication
  - Integration
  - Compilers & analysis
- Our aim
  - Designing efficient runtime systems
    - approaching the raw performance of hardware
    - while preserving applications portability
  - Portability of performance

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

# Heterogeneous Parallel Platforms Programming

General purpose processors + specialized accelerators

- Combination of various units
  - Latency-optimized cores
  - Throughput-optimized cores
  - Energy-optimized cores
- Integrated cores
  - Intel IvyBridge
  - AMD Fusion
  - nVidia Tegra
- Distributed cores
  - Standalone GPUs
  - Intel Knights Corner (MIC)
    - Intel Xeon Phi
  - Intel Single-Chip Cloud (SCC)
    - Many-core without cache consistency

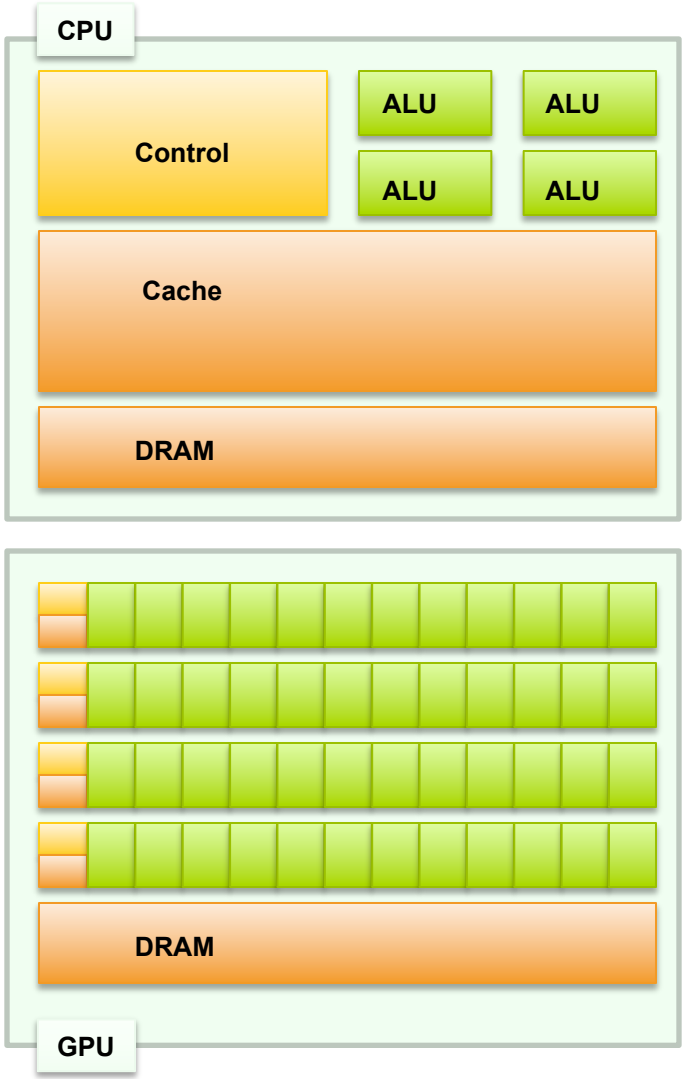


# Heterogeneous Hardware Programming Issue

## CPU vs GPU

Multiple strategies for multiple purposes

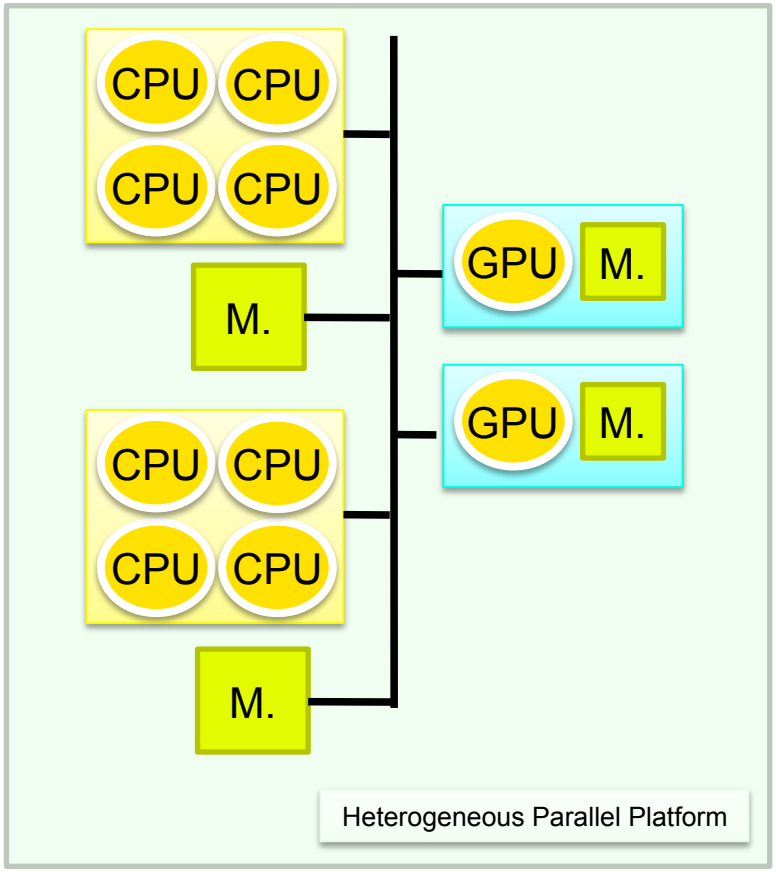
- CPU
  - Strategy
    - Large caches
    - Smart control
  - Purpose
    - Complex codes, branching
    - Complex memory access patterns
  - ~ World Rally Championship car
- GPU
  - Strategy
    - Lot of computing power
    - Simplified control
  - Purpose
    - Regular data parallel codes
    - Simple memory access patterns
  - ~ Formula One car



# Overview of StarPU

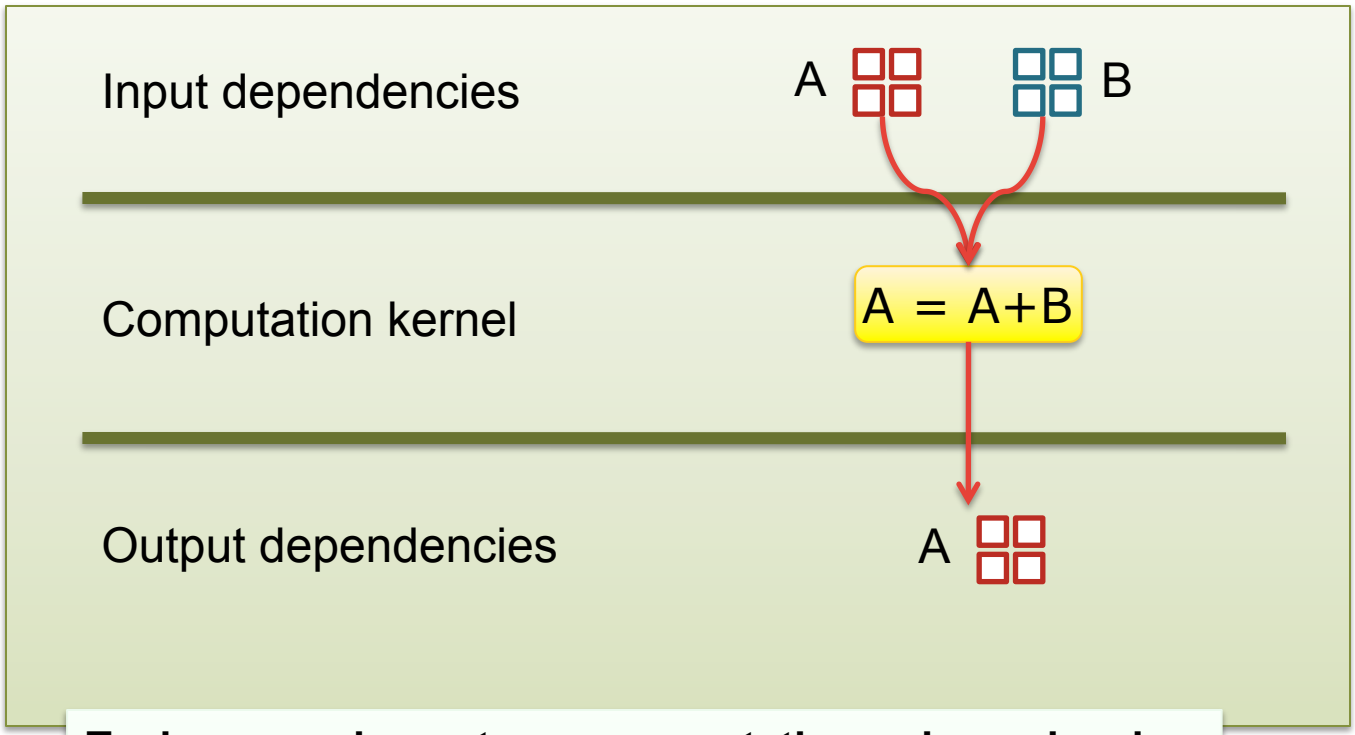
Maximize use of processing units, minimize data transfers

- Task scheduling
  - Dynamic
  - On every PU
    - General purpose
    - Accelerated/specialized
- Memory transfer management
  - Eliminate redundant transfers
  - Software Distributed Shared-Memory (DSM)



# Task Parallelism

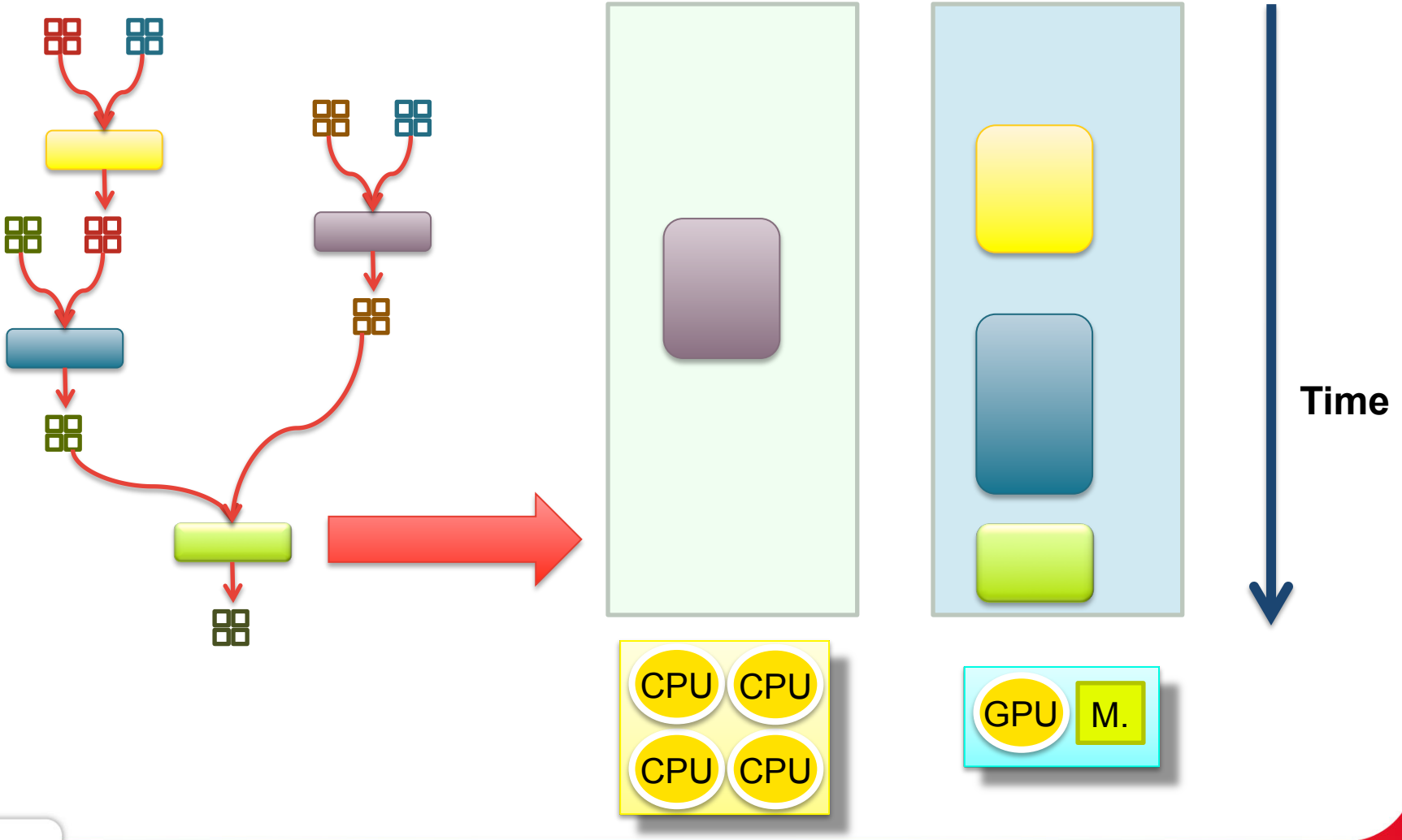





**Task = an « elementary » computation + dependencies**

# Task Scheduling

Mapping the graph of tasks (DAG) on the machine



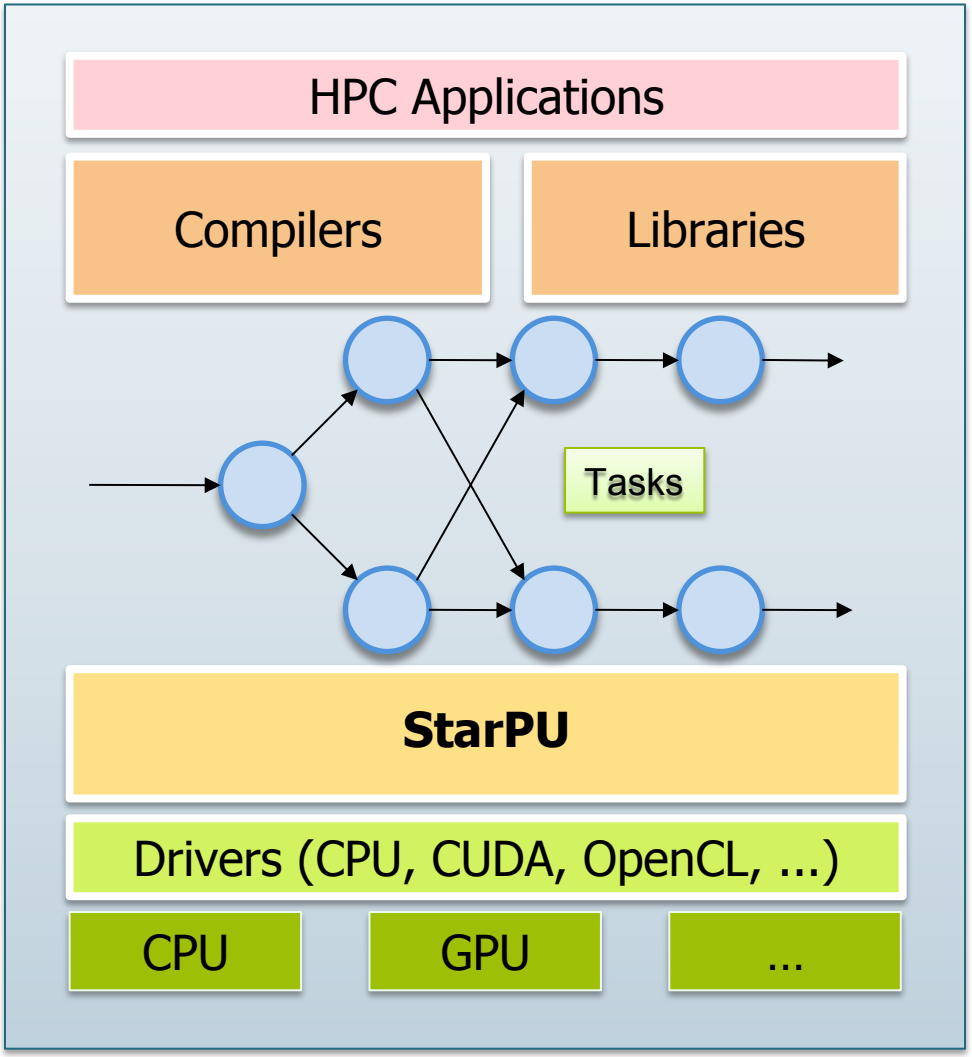
# Overview of StarPU

Information needed from applications

- Tasks
  - Implementation(s)
    - CPU

      Regular

      MMX, SSE, AVX, ...
    - Cuda, OpenCL
- Data
  - Type, layout
    - Vector, matrix, ...
  - Partioning
- Task/Data Relationships
  - Dependencies
    - R, W, R/W, reduction, ...
- (Optional) Custom Scheduler
  - Open Scheduling Platform

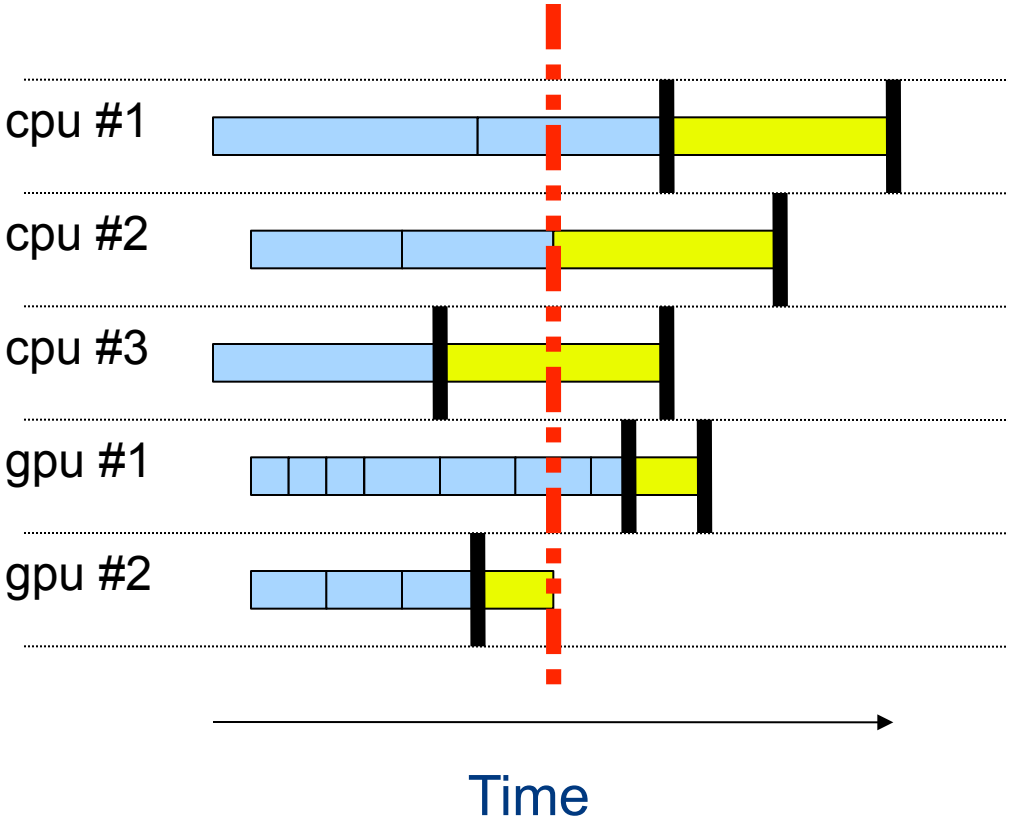


# Prediction-based scheduling

Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
- Can be used to implement scheduling
  - E.g. HEFT

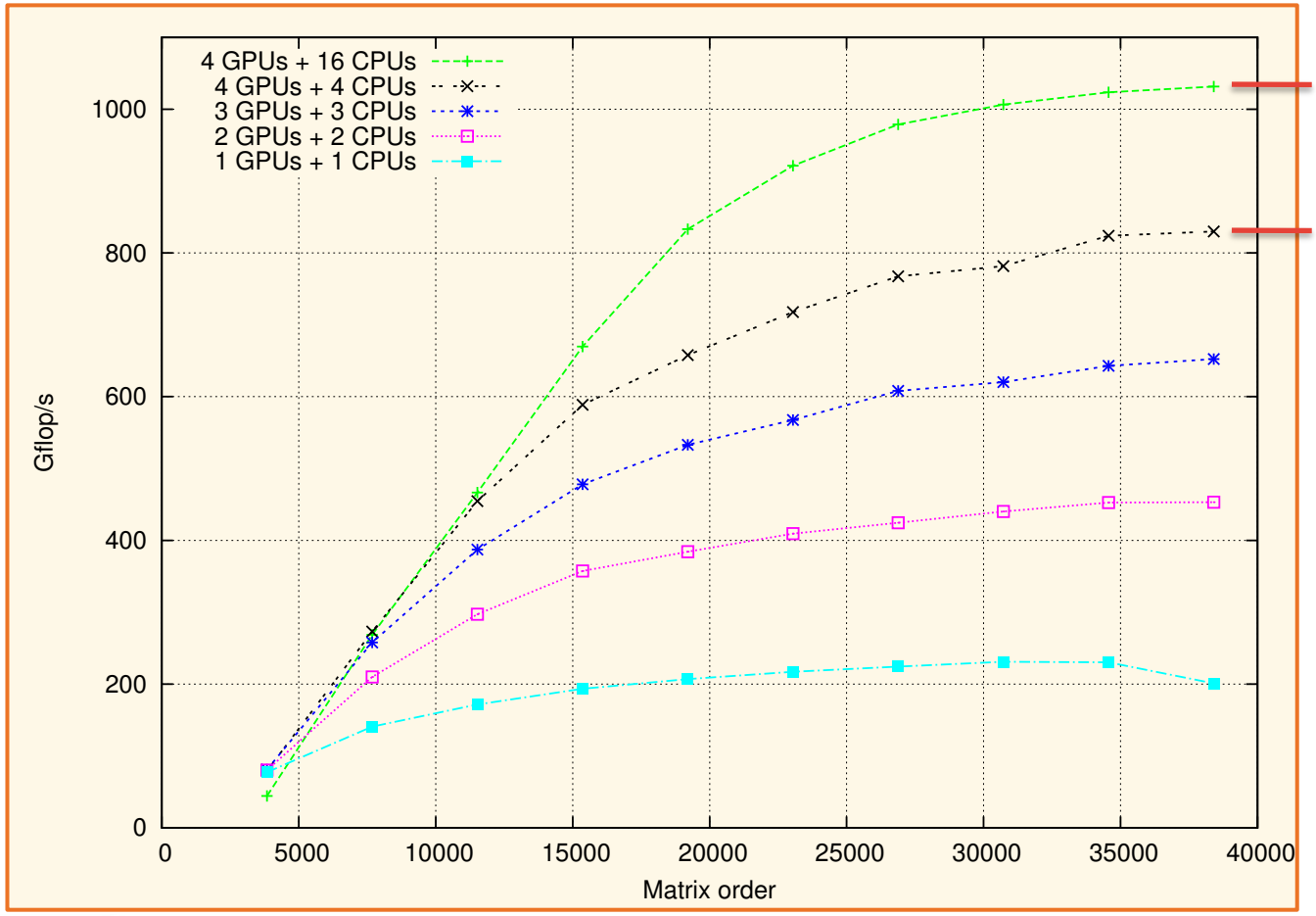
    Heterogeneous Earliest Finish Time



# Example : UTK Magma (lin. alg.) & StarPU

## University of Tennessee & INRIA HiePACS & INRIA Runtime

- QR factorization
  - 4x Nvidia C1060 GPUs + 16x AMD CPUs (12 cores for computation, 4 cores for driving GPUs)




Measured increase:
+12 CPUs
~200 GFlops

Expected increase:
+12 CPUs
~150 Gflops

# Example : UTK Magma (lin. alg.) & StarPU

- « Super-Linear » efficiency in QR?
  - Kernel efficiency
    - sgeqrt: CPU: 9 Gflops GPU: 30 Gflops Speedup: ~3
    - stsqrt: CPU: 12 Gflops GPU: 37 Gflops Speedup: ~3
    - somqr: CPU: 8.5 Gflops GPU: 227 Gflops Speedup: ~27
    - sssmqr: CPU: 10 Gflops GPU: 285 Gflops Speedup: ~28
  - Task distribution observed on StarPU
    - sgeqrt: 20% of tasks on GPUs
    - Sssmqr: 92.5% of tasks on GPUs
  - Taking advantage of heterogeneity!
    - Preferably dedicate GPUs to high potential kernels
    - Avoid slowing down GPUs with low potential kernels

# (Low-level) programming with StarPU

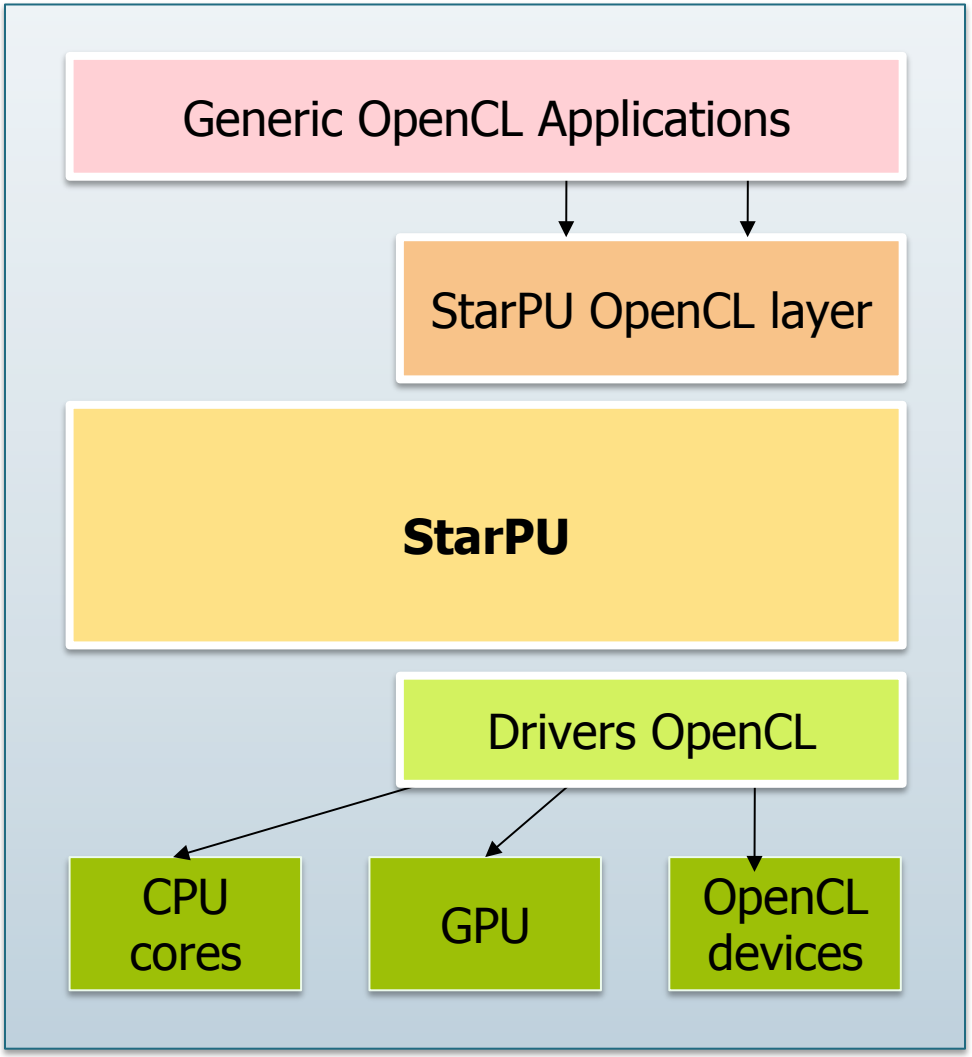
Library API

```
starpu_data_handle vector_handle;
starpu_vector_data_register(&vector_handle, 0,
                    (uintptr_t)vector,  NX, sizeof(vector[0]));

struct starpu_task *task = starpu_task_create();
task->cl = &scal_cl;
task->buffers[0].handle = vector_handle;
task->buffers[0].mode   = STARPU_RW;
float factor = 3.14;
task->cl_arg      = &factor;
task->cl_arg_size = sizeof(factor);

starpu_task_submit(task);
starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```

# SOCL – StarPU as backend for OpenCL

Portably programming StarPU through OpenCL

- Run generic OpenCL codes on top of StarPU
- Events can be used between all devices
  - Synchronizations
- Buffers can be shared by all devices
  - Data transfers handled by StarPU
- Compatibility with StarPU's contexts
  - Dedicated command queues
  - Dedicated schedulers



# StarPU – Additional features

- Interfacing with MPI
  - Automatically handles inter-node transfers
    - MPI communication for enforcing inter-node task dependencies
  - Automatically overlaps data transfers and computation

- Out-of-core
  - Large workloads
    - Enable StarPU to evict temporarily unused data to disk
  - Integration with StarPU's memory management layer

# Conclusion: A possible foundation for WP3?

WP3. Languages and programming models for heterogeneous architectures

- **StarPU**
  - Multicores
  - Intel MIC, CUDA/OpenCL devices
- Application integration
  - Full API
  - OpenCL compatibility
- Used in production
  - Linear algebra libraries
    - UTK's MAGMA, qrMUMPS.
  - Dept. of Energy, Airbus Group
- Training activities
  - European PRACE/PATC
  - European COST

# Conclusion: A possible foundation for WP3?

WP3. Languages and programming models for heterogeneous architectures

- **StarPU**
  - Multicores
  - Intel MIC, CUDA/OpenCL devices
- Application integration
  - Full API
  - OpenCL compatibility
- Used in production
  - Linear algebra libraries
    - UTK's MAGMA, qrMUMPS.
  - Dept. of Energy, Airbus Group
- Training activities
  - European PRACE/PATC
  - European COST

## Potential cooperation topics

- **Scheduling physics applications**
  - Porting OpenCL codes
  - Interfacing with languages/libraries
- **Improving task-based codes debugging/profiling tools**
  - Interoperability with
  - Allinea Software DDT/MAP?

# Thank you for your attention!

**StarPU**

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

LGPL License

Open to external contributors