Running PEPPHER benchmarks on top of the StarPU runtime system

Cédric Augonnet
Nicolas Collin
Nathalie Furmento
Raymond Namyst
Samuel Thibault

INRIA Bordeaux, LaBRI, Université de Bordeaux

22th January 2011

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche BORDEAUX - SUD-OUEST

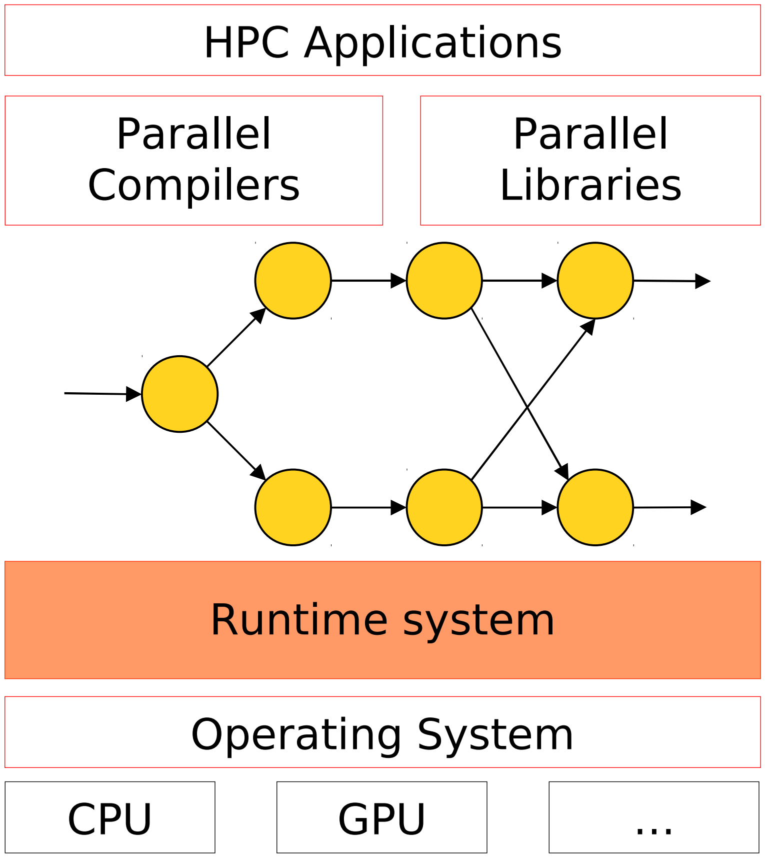
The StarPU runtime system

Motivations

- “do dynamically what can’t be done statically”

- Typical duties
  - Task scheduling
  - Memory management

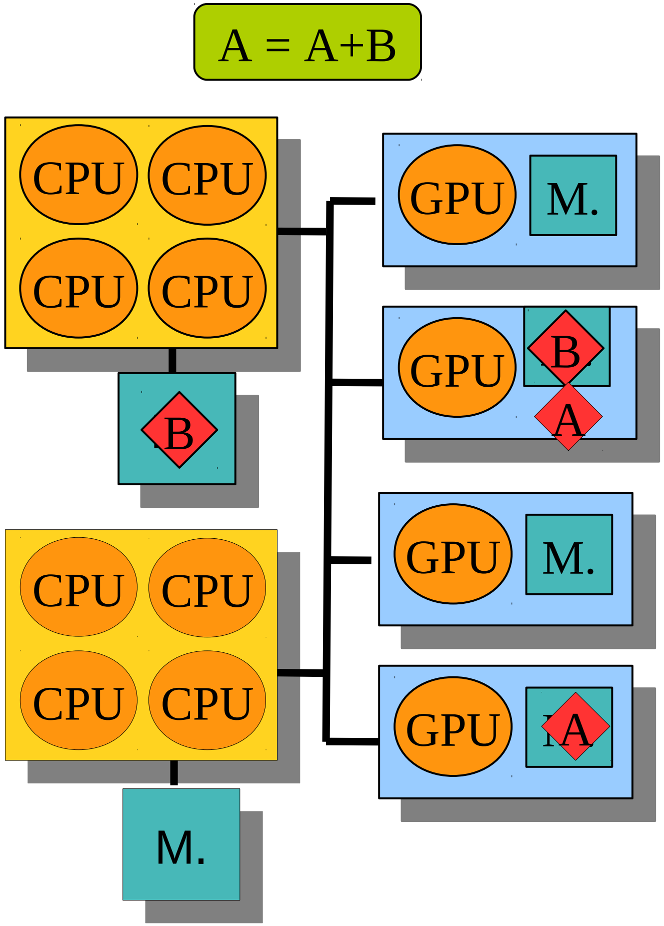
- Compilers and libraries generate (graphs of) parallel tasks
  - Additional information is welcome!



The StarPU runtime system

Motivations

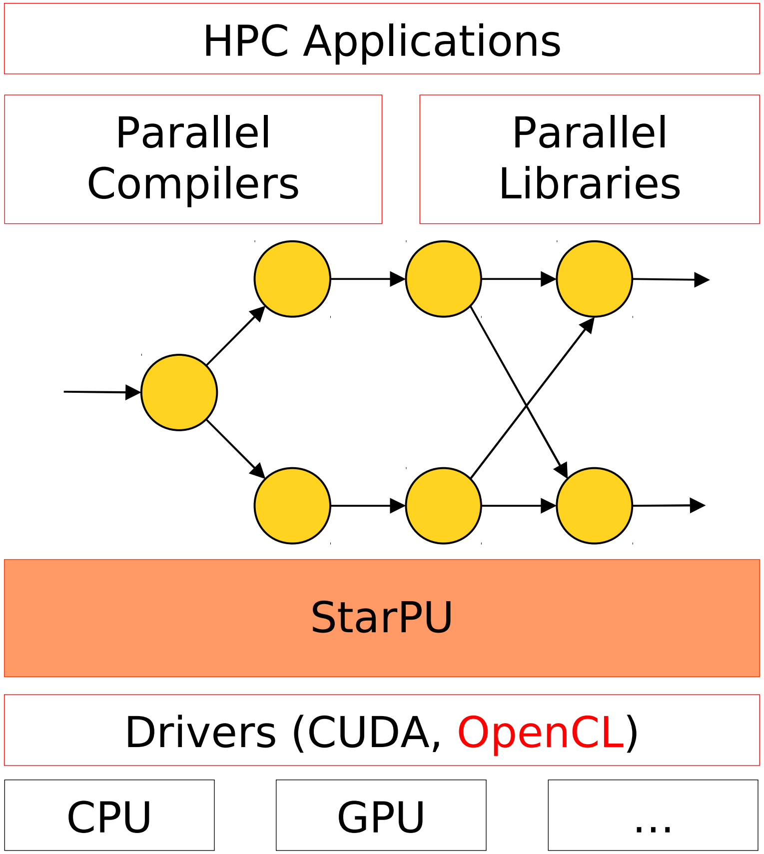
- Main Challenges
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous cores
    - Scheduling ≠ offloading
  - Avoid unnecessary data transfers between accelerators
    - Need to keep track of data copies



The StarPU runtime system

Memory Management

- StarPU provides a Virtual Shared Memory subsystem
    - Weak Consistency
    - Replication
    - Single writer
    - High level API
- Application registers data
- Input & ouput of tasks = reference to registered data

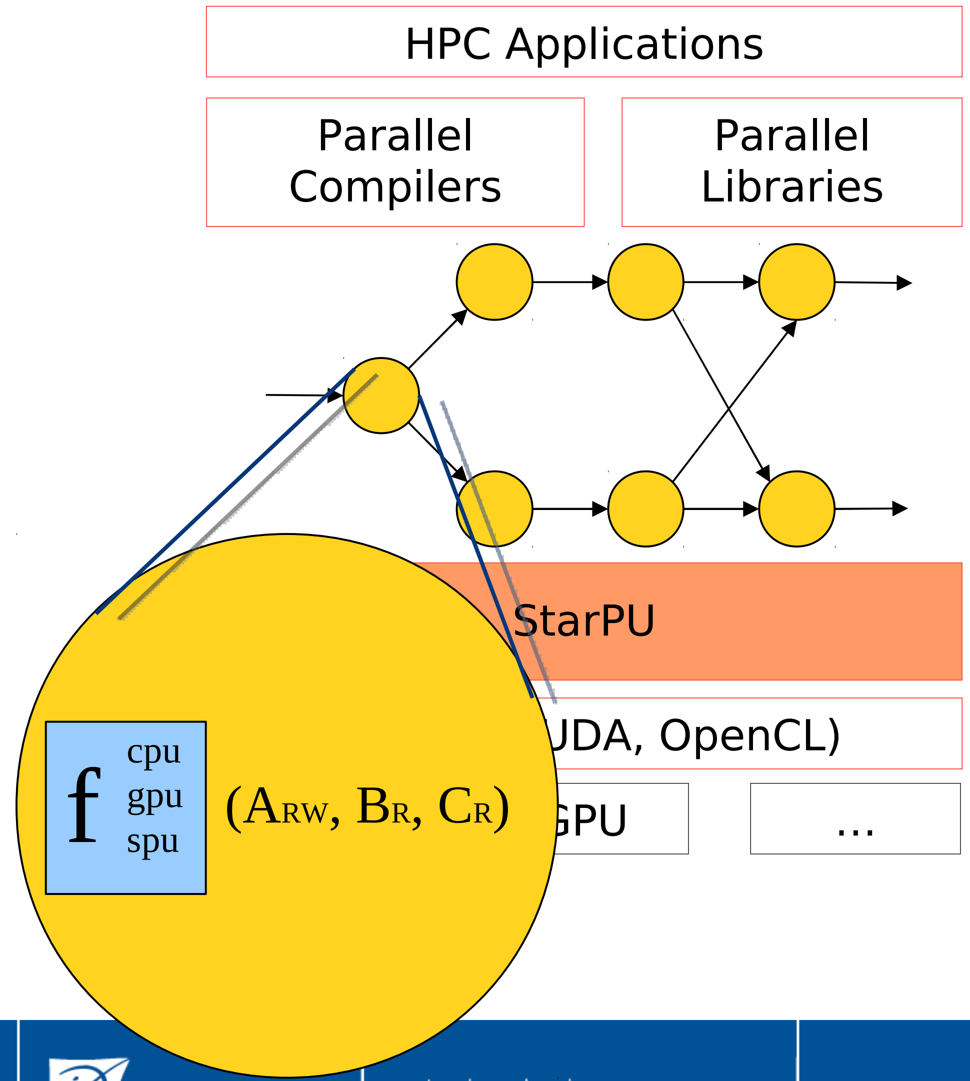


The StarPU runtime system

Task scheduling

- Tasks =
  - Data input & output
  - Dependencies with other tasks
  - Multiple implementations
    - e.g. CUDA and/or CPU
  - Scheduling hints

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



Pepphe Benchmarks

- Fast Fourier Transform (FFT)
  - Mixing FFTW and CUFFTW
- Dense Linear Algebra
  - Mixing PLASMA and MAGMA
- Computational Fluid Dynamic (CFD)
  - Porting Rodinia's CFD

Dense Linear Algebra

Mixing PLASMA and MAGMA

(Collaboration with UTK)

Mixing PLASMA and MAGMA with StarPU

Background

- Background
  - Cholesky/LU/QR: Solve dense linear systems
  - UTK : ~ leaders for Dense Linear Algebra for 20 years
  - Need performance portability

- State of the art libraries
  - PLASMA (Multicore CPUs)
  - MAGMA (Multiple GPUs)

- Our approach
  - Use PLASMA algorithms
  - PLASMA kernels on CPUs, MAGMA kernels on GPUs
  - Schedule tasks with StarPU

Mixing PLASMA and MAGMA with StarPU

Productivity

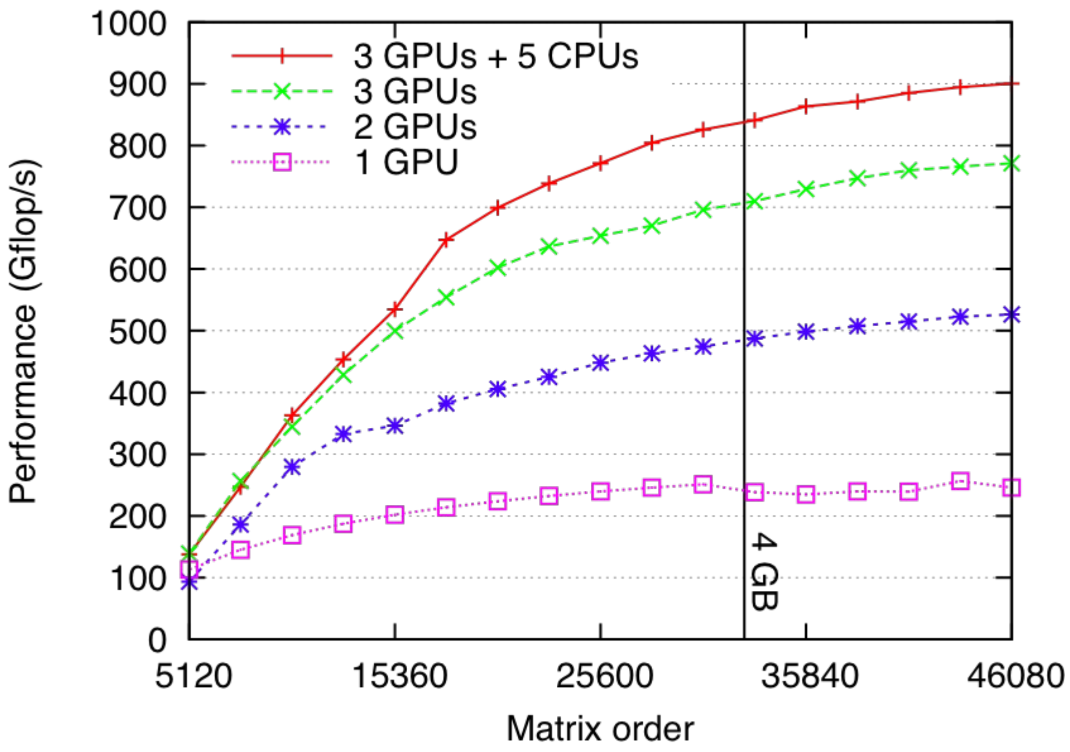
- Programmability
  - Cholesky: ~half a week, QR: ~2 days of works, LU : ~time to write new kernels
  - Quick algorithmic prototyping

```
// Sequential Tile Cholesky
FOR k = 0..TILES-1
  DPOTRF(A[k][k])
  FOR m = k+1..TILES-1
    DTRSM(A[k][k], A[m][k])
  FOR n = k+1..TILES-1
    DSYRK(A[n][k], A[n][n])
  FOR m = n+1..TILES-1
  DGEMM(A[m][k], A[n][k], A[m][n])
```

```
// Hybrid Tile Cholesky
FOR k = 0..TILES-1
    starpu_Insert_Task(DPOTRF, ...)
    FOR m = k+1..TILES-1
      starpu_Insert_Task(DTRSM, ...)
    FOR n = k+1..TILES-1
      starpu_Insert_Task(DSYRK, ...)
   FOR m = n+1..TILES-1
      starpu_Insert_Task(DGEMM, ...)
```

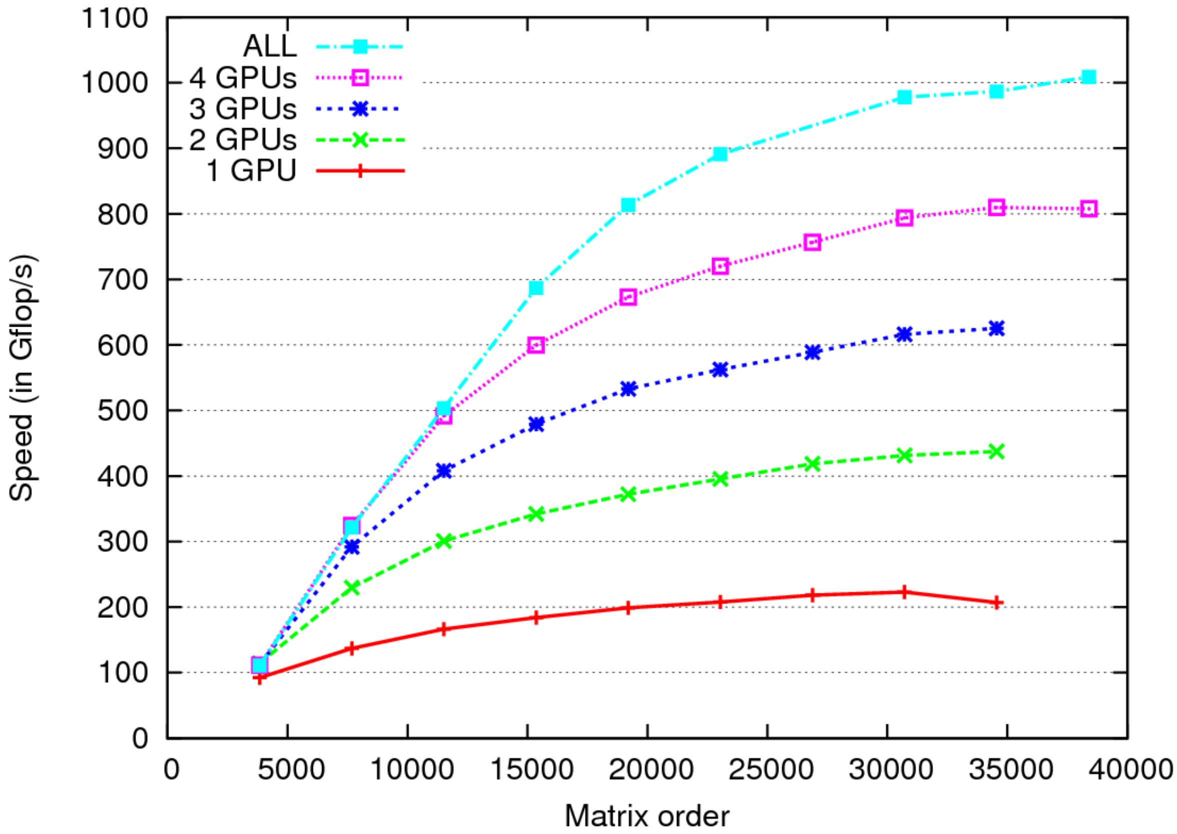
Mixing PLASMA and MAGMA with StarPU

- Cholesky decomposition
  - Hannibal: 8 CPU cores (Nehalem) + 3 GPUs (NV FX5800)



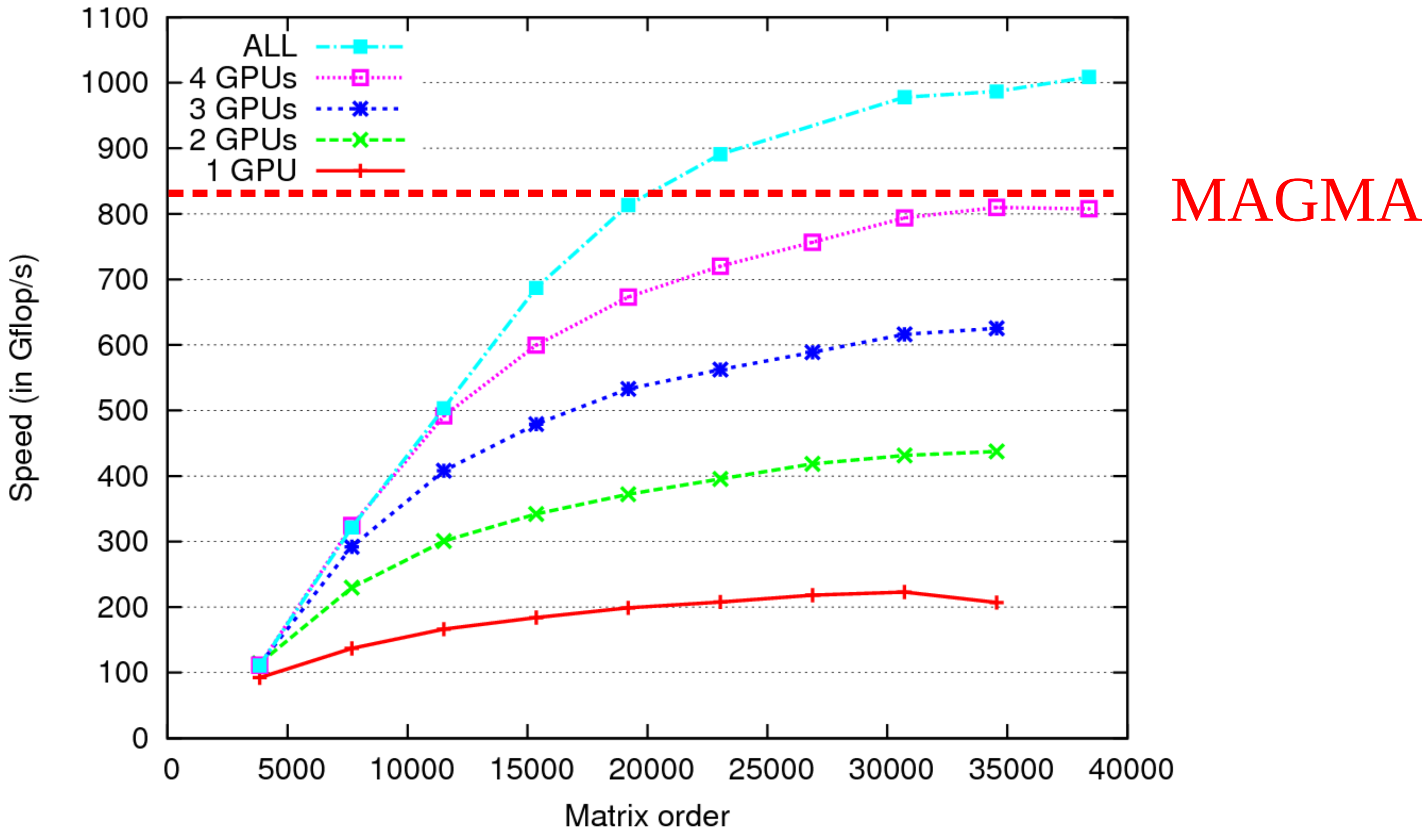
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



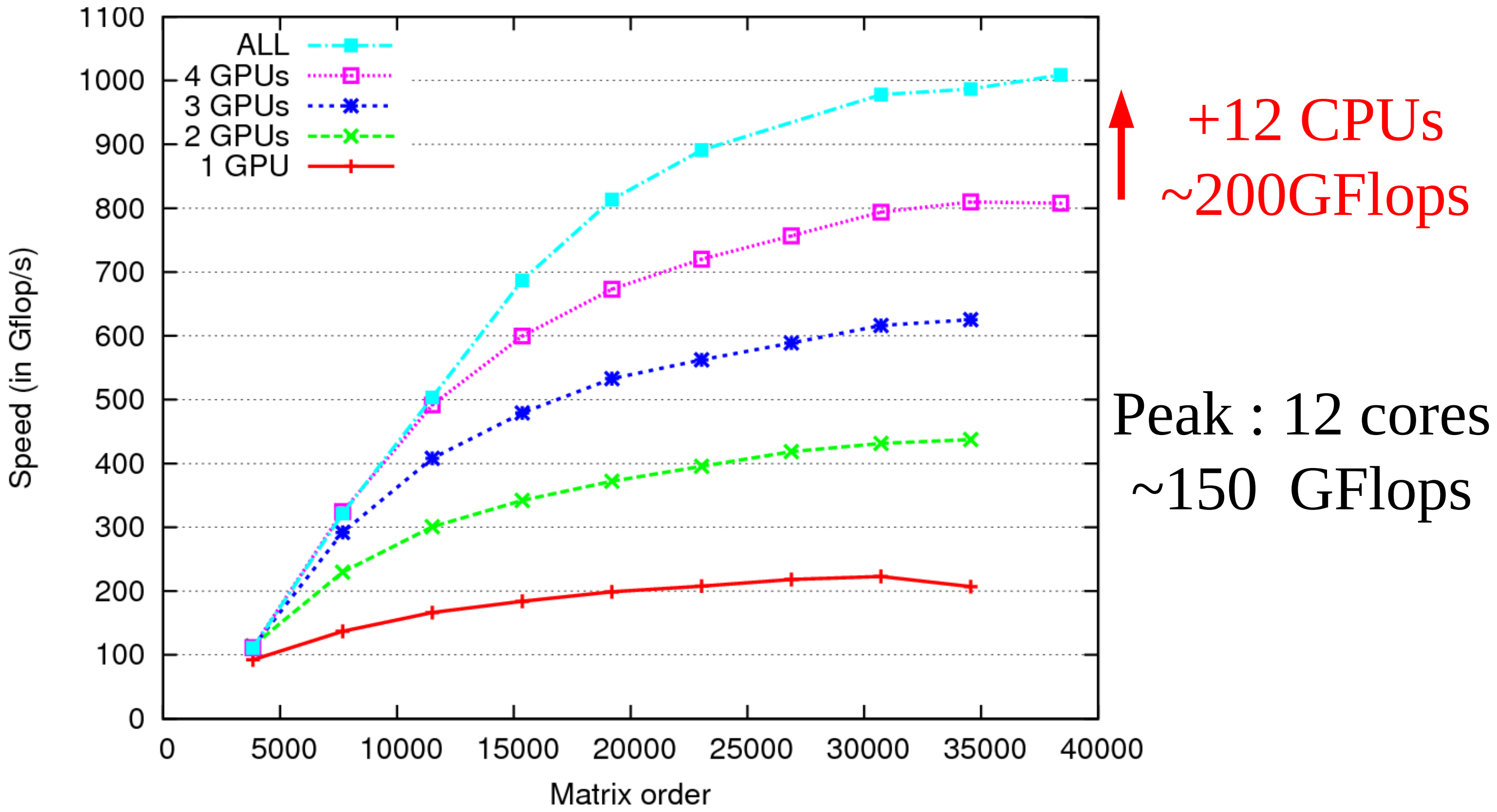
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



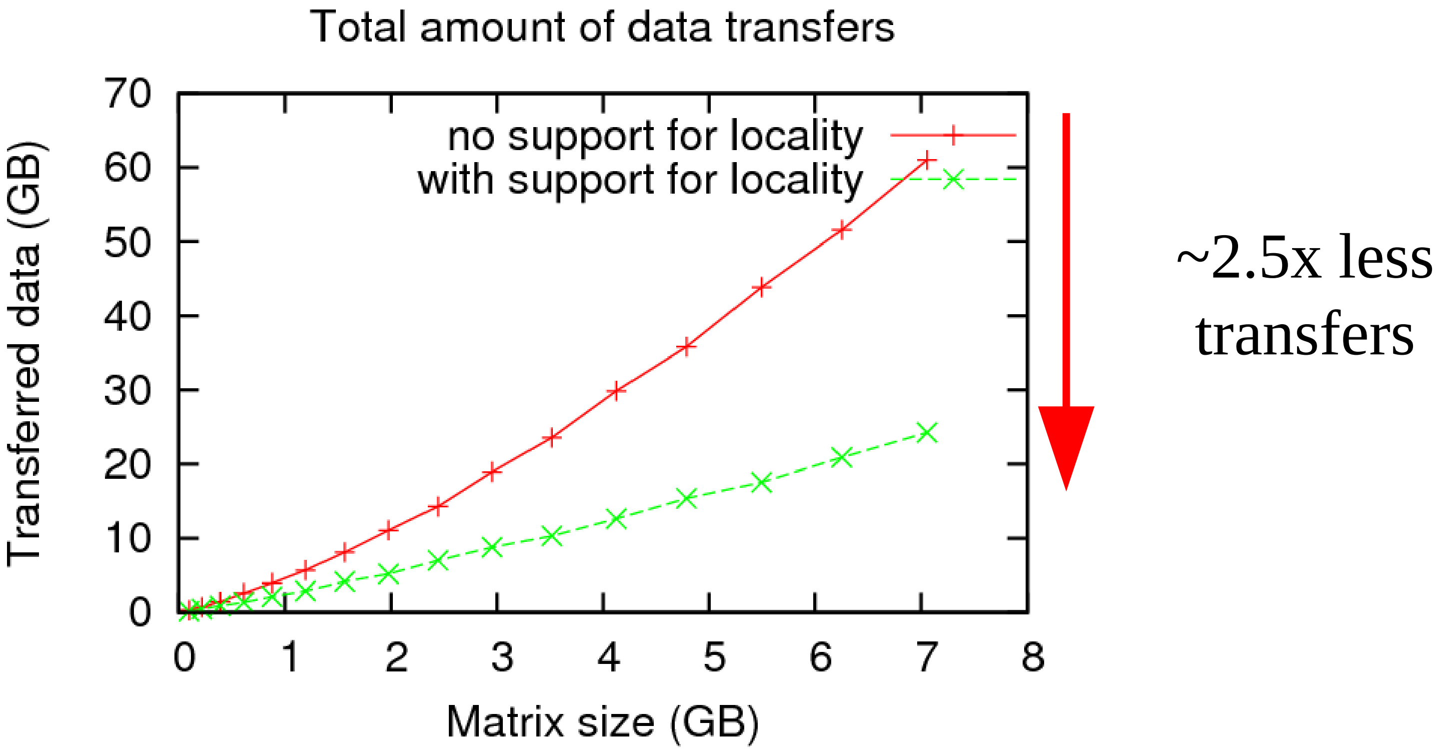
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



Mixing PLASMA and MAGMA with StarPU

- Memory transfers during Cholesky decomposition



Mixing PLASMA and MAGMA with StarPU

Perspective

- Add more algorithms
  - 2-sided Factorizations (eg. Hessenberg)
  - Solvers
- Going to be released as a standalone library
  - Toward a complete LAPACK implementation for hybrid computing
  - Need autotuning facilities!
- Next step: integrate MPI
  - On-going work
  - Accelerated SCALAPACK ?

Rodinia's CFD Solver

Rodinia's CFD Solver

Background

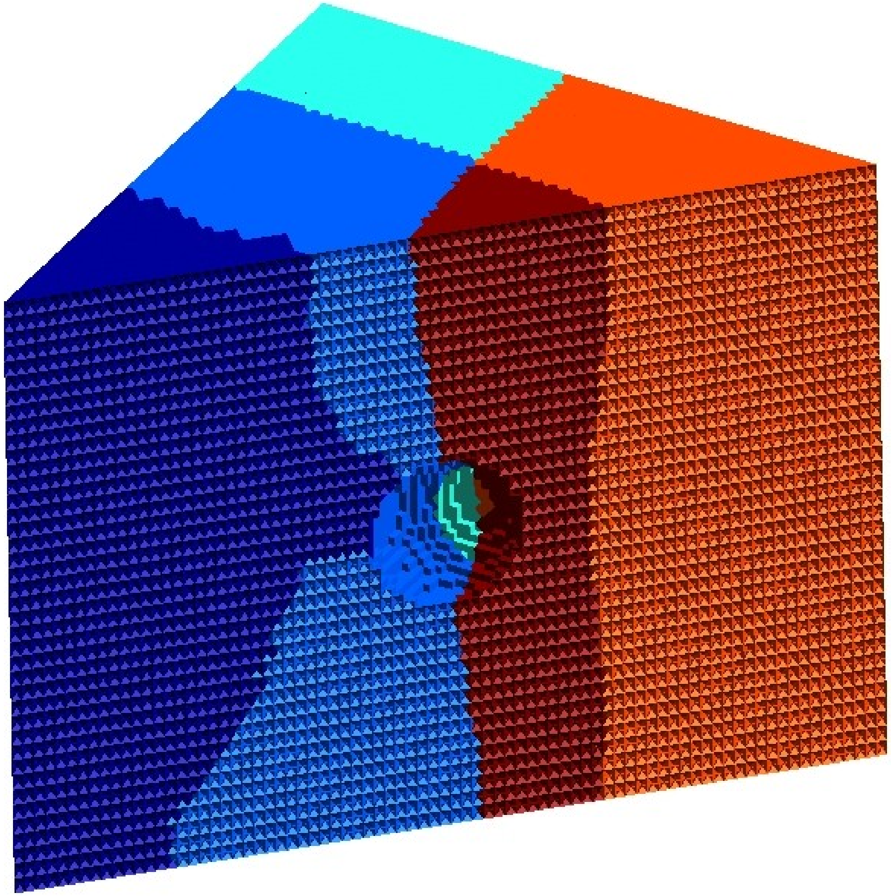
- The Rodinia benchmark suite
  - Cover the different « Berkeley Dwarves »
  - Available either in OpenMP or in CUDA
  - Neither multi-GPU nor hybrid systems

- Rodinia's CFD Solver benchmark
  - 3D Euler equations for incompressible flow
  - Unstructured Grid Finite Volumes
  - Memory intensive kernel
  - Pre-processing and Post-processing are not available
    - Need to create our own input meshes

Rodinia's CFD Solver

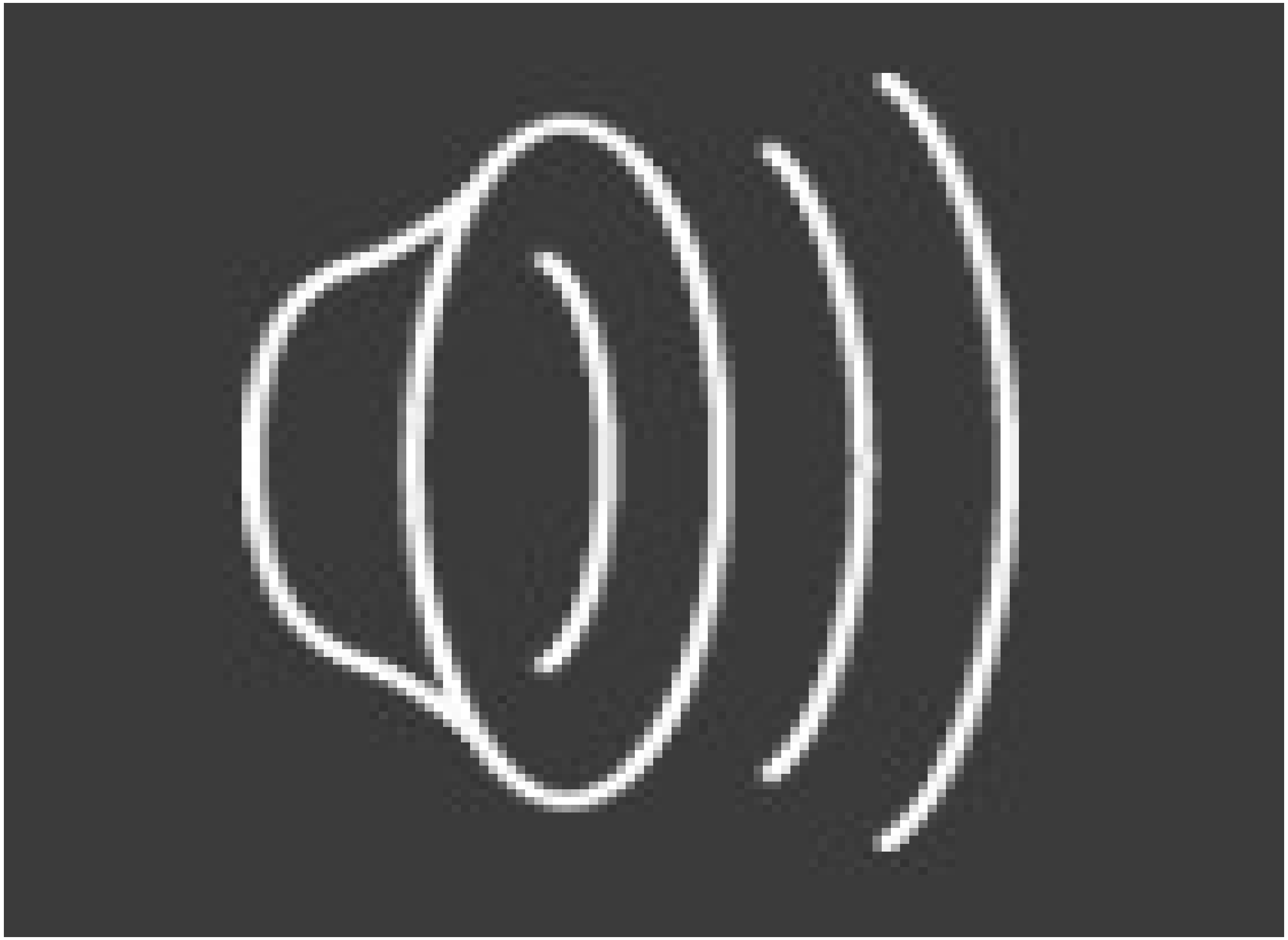
Methodology

- Pre-processing
  - Generated a mesh of the air around a sphere
  - Very simple yet !
- Parallelizing the problem
  - Partition the mesh using SCOTCH
  - 1 task = update 1 part
  - Redundant computation
  - Exchange part boundaries



Rodinia's CFD Solver

Post-processing



Rodinia's CFD Solver

Preliminary results

- Problem size
  - 64x64x64 grid, 1.3 Millions tetrahedrons
- Reference CPU performance
  - 1 core (Intel Westmere X5650)
    - 1.4s per iteration
  - 12 cores
    - 0.15s per iteration
- Preliminary performance with StarPU
  - 1 NVIDIA C2050
    - 53ms per iteration
  - 2 NVIDIA C2050
    - 28ms per iteration
  - We need large problems !

Rodinia's CFD Solver

Perspective

- Port in OpenCL
- Use hybrid platforms
  - GPUs are much faster than CPUs
    - Memory bound
    - Rather few tasks
  - Parallel CPU tasks
    - large granularity
- Heterogeneity-aware data layout
  - CPUs : Arrays of Structures (cache friendly)
  - GPUs : Structures of Arrays (SIMD friendly)

Conclusion

- StarPU
  - Data management & Task scheduling
  - Freely available under LGPL on Linux, Mac and Windows
- Adapted 3 PEPPHER benchmarks
  - FFTW + CUFFTW
  - MAGMA + PLASMA
  - Rodinia's CFD Solver

Conclusion

- Productive approach
  - Rely on existing kernels for CPU/GPU
  - Architecture independent task model
  - Higher-level front-ends would help
    - StarSs, HMPP, Codeplay's Offload

- Autotuning will be required
  - Need to find optimal granularity
    - Parallel tasks
    - Divisible tasks
  - Select code variants
    - eg. with SkePU