OpenMP 4.X to programming HPC applications on heterogeneous architectures?

Thierry Gautier
thierry.gautier@**inrialpes.fr**
MOAIS, INRIA, Grenoble

C2S@Exa IPL

Outline

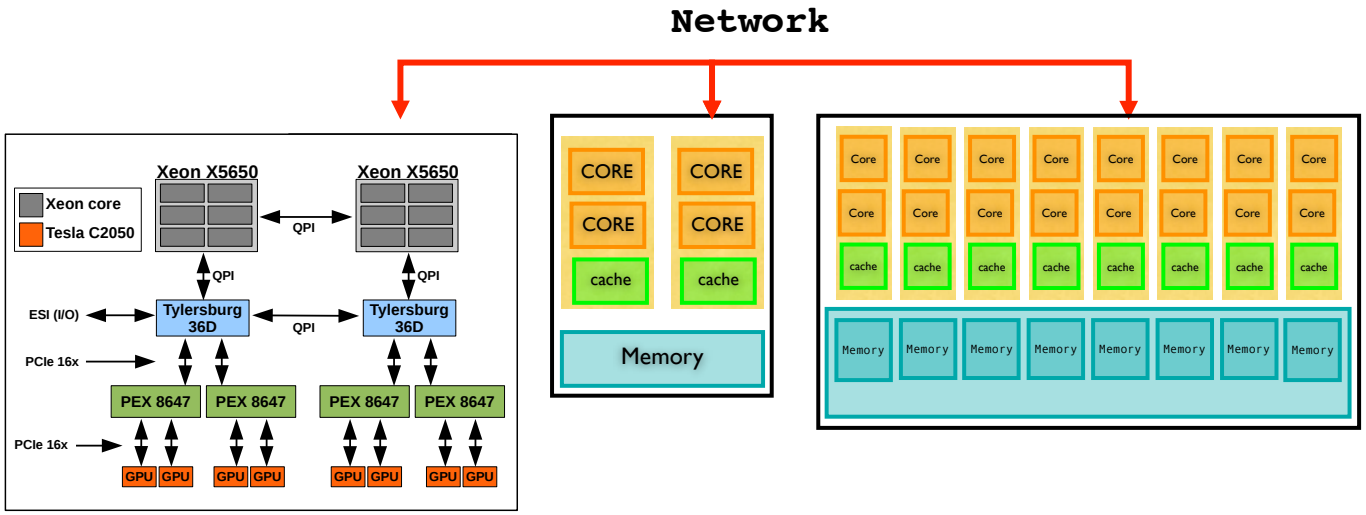
- Introduction
- OpenMP 3.1
- Improving overhead in OpenMP task management
- Extending loop scheduler
- ADT K'STAR & OpenMP4.X
- Conclusions

Parallel heterogeneous architecture

- **Complex architecture**
  - **Computing resources**
    - SIMD Units, CPU Core, GPU, Intel Xeon Phi,..., MPPA, FPGA...
  - **Memory**
    - hierarchical memory
      - registers, cache L1, L2, L3, local memory bank, global memory, remote memory
    - private / shared cache
  - **Interconnection networks**
    - between cores & memory = memory network (HyperTransport, QuickPath, PCI, ...)
    - between machines (Ethernet, InfiniBand,...)

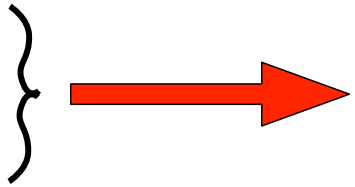
➡ **High complexity**

- **million of components**
- **heterogeneous**
  - memory access
  - computing capability



Role of a parallel programming model

- **Abstraction of the hardware**
  - **Hide complexity**
    - Numerous computing resources
      - SIMD Units, Core
    - NUMA (heterogeneous) architecture
      - Latency + contention in “remote” memory access
  - **Good compromise performance/simplicity**
- **Supercomputer = high performance network + multicores + accelerators**
  - **Network: MPI**
  - **Multicores: OpenMP**
  - **Accelerators: Cuda, OpenCL, OpenACC**
  - **SIMD Units: Compiler or extension**



OpenMP 4.X ?

- **Main Difficulty = scheduling**
  - **Scheduling is a hard problem**
    - heterogenous architecture
      - 1 GPU may be about 100 times faster than 1 CPU core
      - Taking into account communication

- **Which programming model ?**
  - **Currently at least 3 / 4 different models...**

OpenMP

- **Explicit Parallelism**
  - **Code annotation for Fortran, C, C++**

- **OpenMP 3.1**
  - **Task parallelism**
    - #pragma omp task [...]
    - Creation of task is a non blocking operation
      - completion guaranteed by synchronization
    - Task execution may be concurrent with the thread that creates it
  - **Data parallelism = parallel loop**
    - #pragma omp parallel for / #pragma omp for in a parallel region

- **OpenMP 4.X**
  - **SIMD Units**
    - Public beta, SC 2012
  - **Dependencies between tasks**
  - **Accelerator**
    - ~ OpenACC: Offloading code to accelerator

Some Limitations

- **Programming Model**
  - **Reduction operators are assumed to be associative & commutative**
  - **Parallel loop is only for independent loop**
    - automatic parallelization has some limitation
  - **Scheduling parallel loop: static, dynamic, guided**
    - ”chunk size” decision !
  - **Becomes more and more complex...**
- **Runtime support**
  - **Overhead in task management**
    - Intel Compiler has lower overhead over GCC. GCC does not work with lot of fine grain tasks.
  - **Internal algorithms (scheduling, termination, ...) may not scale well on NUMA**
- **OpenMP Standard evolves slowly but it does!**

- **Extensions**

1. **Improving overhead in task management**
2. **New NUMA aware loop schedule for irregular iterations**
3. **Future work around OpenMP4.X features**

Improving OpenMP task implementation

- [IWOMP 2012]
- Barcelona OpenMP Task Suite
  - Using libKOMP = our libGOMP implementation (on top of XKaapi)
  - A set of representative benchmarks to evalute OpenMP tasks implementations

Name	*Arguments used*	*Domain*	*Summary*
Alignment	prot100.aa	Dynamic programming	Aligns sequences of proteins
FFT	n=33,554,432	Spectral method	Computes a Fast Fourier Transformation
Floorplan	input.20	Optimization	Computes the optimal placement of cells in a floorplan
NQueens	n=14	Search	Finds solutions of the N Queens problem
MultiSort	n=33,554,432	Integer sorting	Uses a mixture of sorting algorithms to sort a vector
SparseLU	n=128 m=64	Sparse linear algebra	Computes the LU factorization of a sparse matrix
Strassen	n=8192	Dense linear algebra	Computes a matrix multiply with Strassen's method
UTS	medium.input	Search	Computes the number of nodes in an Unbalanced Tree

- Evaluation platforms
  - AMD48: 4x12 AMD Opteron (6174) cores
  - Intel32: 4x8 Intel Xeon (X7560) cores
- Softwares
  - gcc 4.6.2 + libGOMP
  - gcc 4.6.2 + libKOMP
  - icc 12.1.2 + Intel OpenMP runtime (KMP)

Running OpenMP BOTS with libKOMP

Speed-Up of BOTS kernels on the AMD48 platform

kernel	***libGOMP***	***libKOMP***	***Intel***
Alignment	38.8	**40.0**	37.0
FFT	0.5	**12.2**	12.0
Floorplan	27.6	**32.7**	29.2
NQueens	43.7	**47.8**	39.0
MultiSort	0.6	**13.2**	11.3
SparseLU	44.1	**44.4**	35.0
Strassen	20.8	**22.4**	20.5
UTS	0.9	**25.3**	15.0

- **Evaluation platforms**
  - **AMD48: 4x12 AMD Opteron (6174) cores**
  - **Intel32: 4x8 Intel Xeon (X7560) cores**

- **Softwares**
  - **gcc 4.6.2 + libGOMP**
  - **gcc 4.6.2 + libKOMP**
  - **icc 12.1.2 + Intel OpenMP runtime (KMP)**

Automatic chunk size decision

- **Parallel loop**
  - **which OpenMP schedule: static, dynamic, guided ?**
  - **which chunk size ? depend on the application or the instance or the architecture!**
    - To “big” = less parallelism, difficult to schedule
    - To “small” = overhead

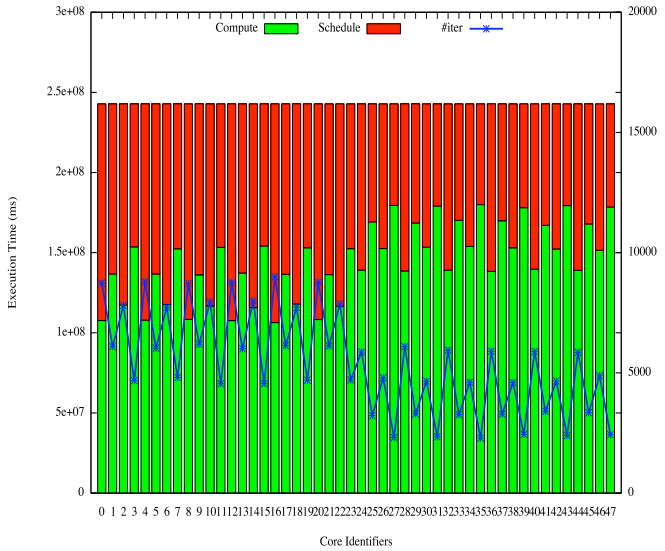
➡**Automatic selection of the “chunk size”**

- **Proposed solution: new adaptive OpenMP loop scheduler**
  - **Runtime selection of the grain size**
  - **NUMA aware scheduling**
  - **Extension of libGOMP (runtime support of GCC)**
  - **[submitted to IWOMP2013]**

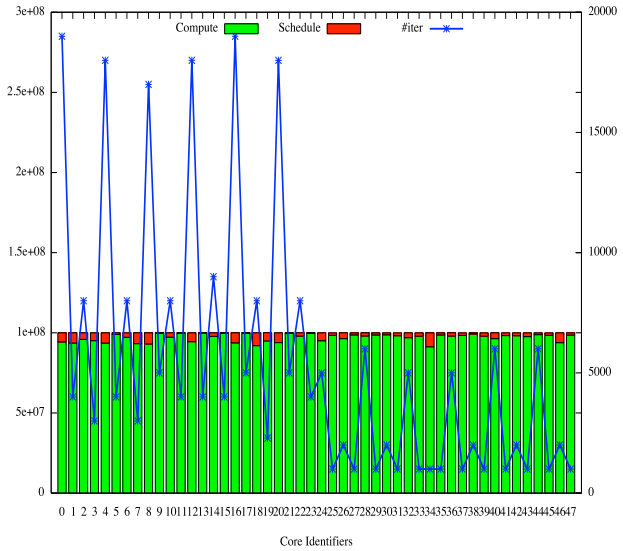
Particle collision detection

- Iterative application
  - One step of the code ~ parallel loop (2.9e6 particles)
    - irregular computation
- GCC / OpenMP libGOMP + instrumentation
- Report activity (#ticks) per core
  - AMD 48 cores machine
  - compute (green): user code
  - schedule (red): loop initialization, load balancing, loop termination, ...
  - #iterations per core (blue curve)

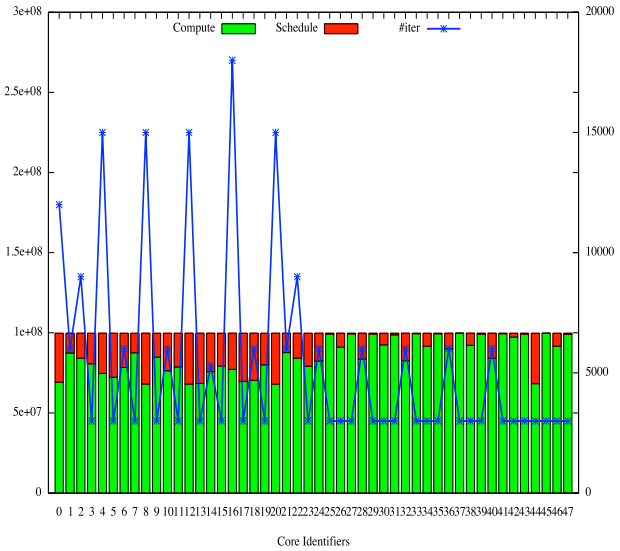
Dynamic schedule



Chunk size=1
time = 98.3ms



Chunk size=1000

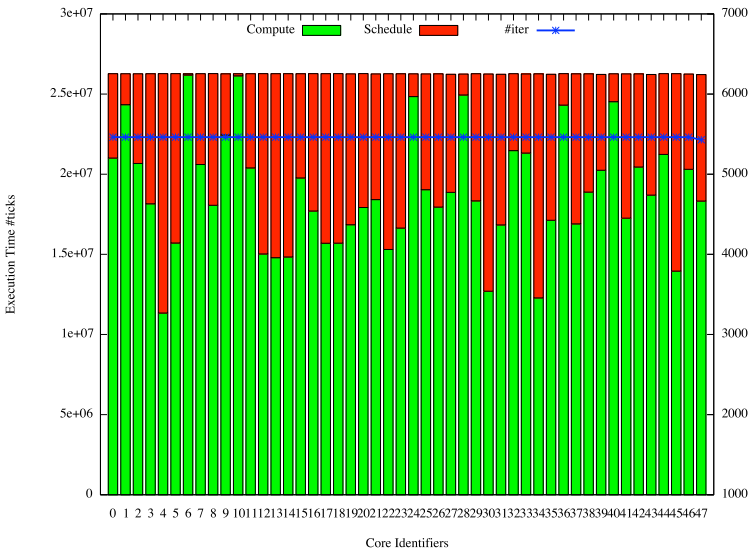


Best Chunk size=3000
time = 46ms

Particle collision detection

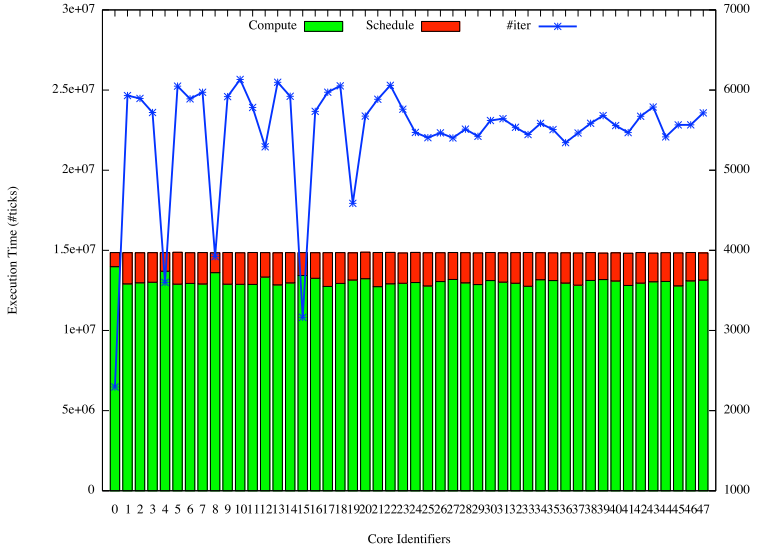
- 1 step of the code ~ parallel loop (2.9e6 particles)
  - irregular computation
- GCC / OpenMP libGOMP + instrumentation + new loop schedule
- Report activity (#ticks) per core
  - AMD 48 cores machine
  - compute (green): user code
  - schedule (red): loop initialization, load balancing, loop termination, ...
  - #iterations per core (blue curve)

Static & Adaptive schedule



Static OpenMP schedule

time = 12.41ms



New Adaptive schedule

time = 6.7ms

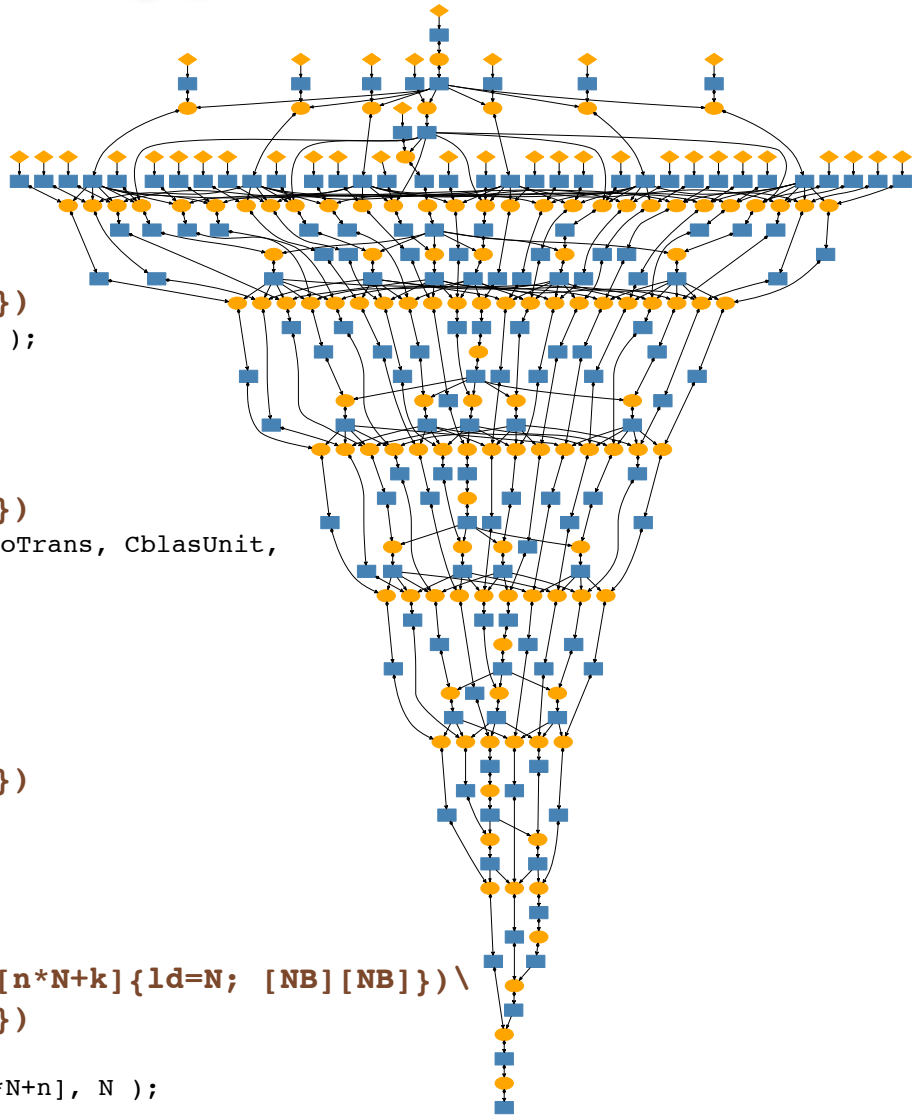
INRIA ADT K'START

- **OpenMP 4.X as a good candidate for programming heterogeneous multicores**
  - **SIMD Units, CPUs, GPUs or Intel Xeon Phi (MIC)**
  - **Standard not yet publicity available but access to discussions & proposals is possible**
- **What's new / under discussion ?**
  - **SIMD annotation**
    - SSE unit (Intel Xeon & Xeon Phi), Nvidia SM, Multicore / multi DSP (Texas Instrument), ...
  - **Dependent tasks with data flow dependencies**
  - **Accelerator: integrate OpenACC proposition into OpenMP standard**

- **ADT K'START (~2014-2015)**
  - **EPI INRIA MOAIS (Grenoble, T.Gautier) and RUNTIME (Bordeaux, O. Aumage)**
  - **Goal first implementation of OpenMP annotation for accelerator & dependent tasks**
    - based on the current discussion of the standard
    - Runtime will be based on StarPU and XKaapi
  - **Engineer position funded**
    - **if you have student or engineer with some experience!**
    - **skills: compiler design, Fortran, C, C++,...**

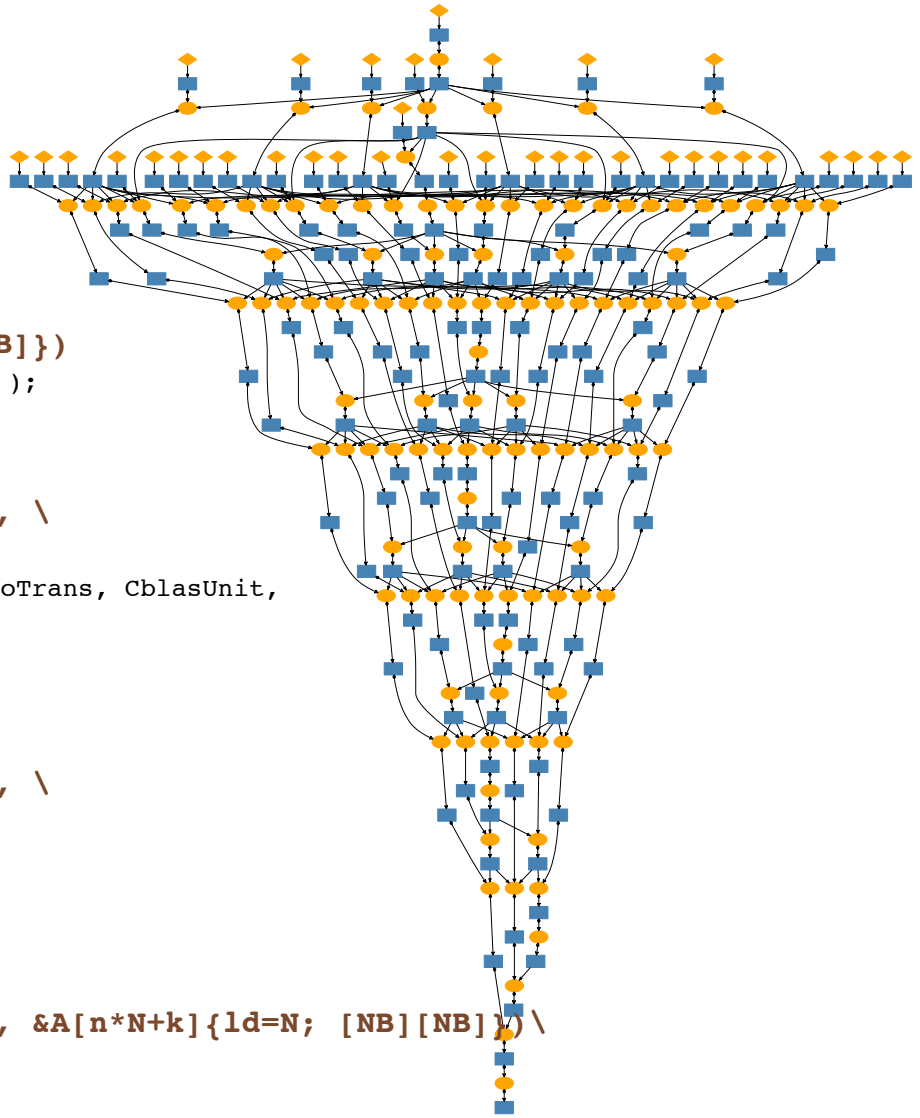
XKaapi' compiler prototype

```
#include <cblas.h>
#include <clapack.h>
void Cholesky( double* A, int N, size_t NB )
{

  for (size_t k=0; k < N; k += NB)
  {
#pragma kaapi task readwrite(&A[k*N+k]{ld=N; [NB][NB]})
    clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma kaapi task read(&A[k*N+k]{ld=N; [NB][NB]}) \
                   readwrite(&A[m*N+k]{ld=N; [NB][NB]})
      cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit,
        NB, NB, 1., &A[k*N+k], N, &A[m*N+k], N );
    }

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma kaapi task read(&A[m*N+k]{ld=N; [NB][NB]}) \
                   readwrite(&A[m*N+m]{ld=N; [NB][NB]})
     cblas_dsyrk ( CblasRowMajor, CblasLower, CblasNoTrans,
        NB, NB, -1.0, &A[m*N+k], N, 1.0, &A[m*N+m], N );

      for (size_t n=k+NB; n < m; n += NB)
      {
#pragma kaapi task read(&A[m*N+k]{ld=N; [NB][NB]}, &A[n*N+k]{ld=N; [NB][NB]})\
                   readwrite(&A[m*N+n]{ld=N; [NB][NB]})
        cblas_dgemm ( CblasRowMajor, CblasNoTrans, CblasTrans,
          NB, NB, NB, -1.0, &A[m*N+k], N, &A[n*N+k], N, 1.0, &A[m*N+n], N );
      }
    }
  }
}
```



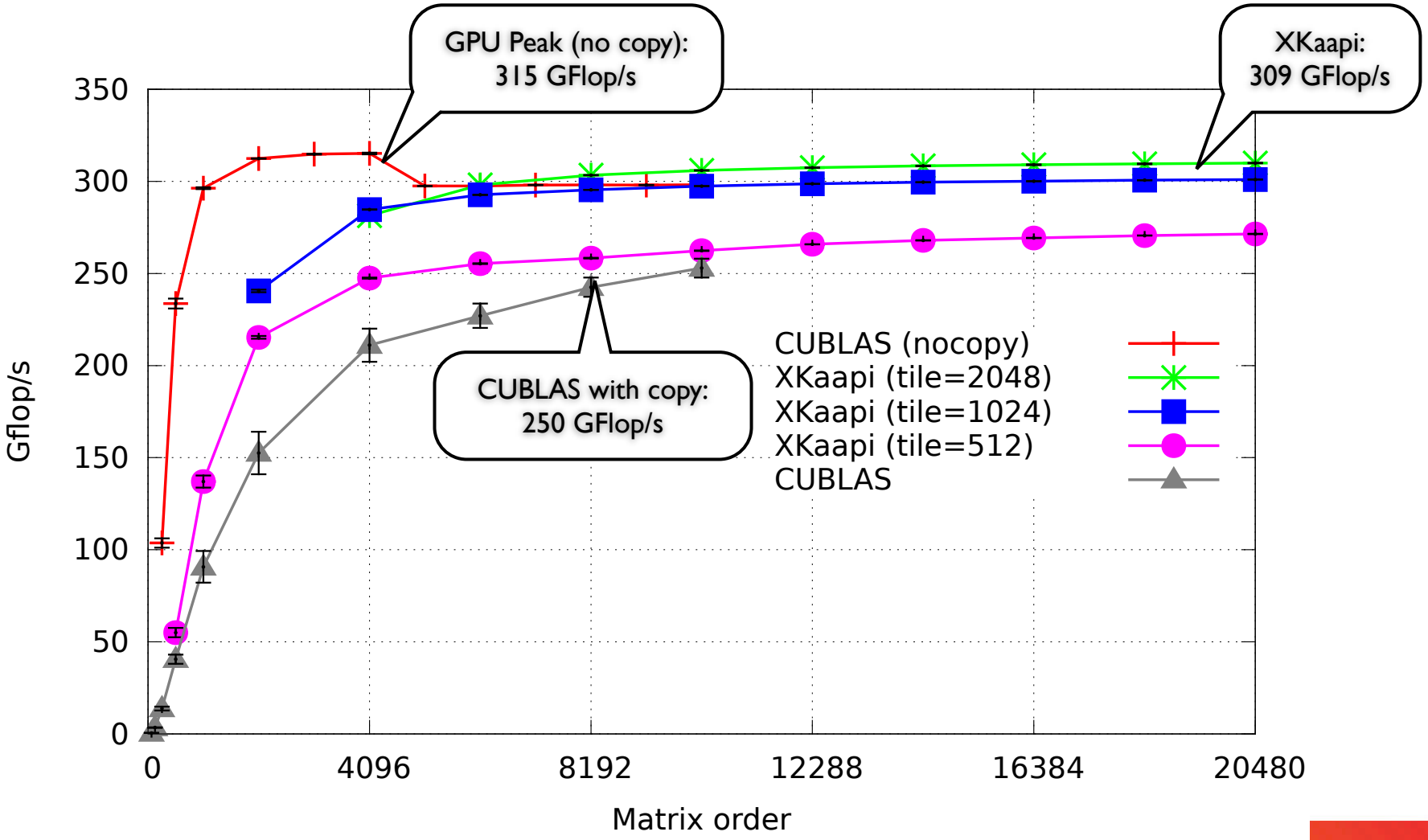
OpenMP version

```
#include <cblas.h>
#include <clapack.h>
void Cholesky( double* A, int N, size_t NB )
{
#pragma omp parallel
  for (size_t k=0; k < N; k += NB)
#pragma omp single
  {
#pragma omp task depend(inout: &A[k*N+k]{ld=N; [NB][NB]})
    clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: &A[k*N+k]{ld=N; [NB][NB]}, \
                    inout: &A[m*N+k]{ld=N; [NB][NB]})
      cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit,
        NB, NB, 1., &A[k*N+k], N, &A[m*N+k], N );
    }

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: &A[m*N+k]{ld=N; [NB][NB]}, \
                    inout: &A[m*N+m]{ld=N; [NB][NB]})
     cblas_dsyrk ( CblasRowMajor, CblasLower, CblasNoTrans,
        NB, NB, -1.0, &A[m*N+k], N, 1.0, &A[m*N+m], N );

      for (size_t n=k+NB; n < m; n += NB)
      {
#pragma omp task depend(in: &A[m*N+k]{ld=N; [NB][NB]}, &A[n*N+k]{ld=N; [NB][NB]})\
                    inout: &A[m*N+n]{ld=N; [NB][NB]})
        cblas_dgemm ( CblasRowMajor, CblasNoTrans, CblasTrans,
          NB, NB, NB, -1.0, &A[m*N+k], N, &A[n*N+k], N, 1.0, &A[m*N+n], N );
      }
    }
  }
}
```

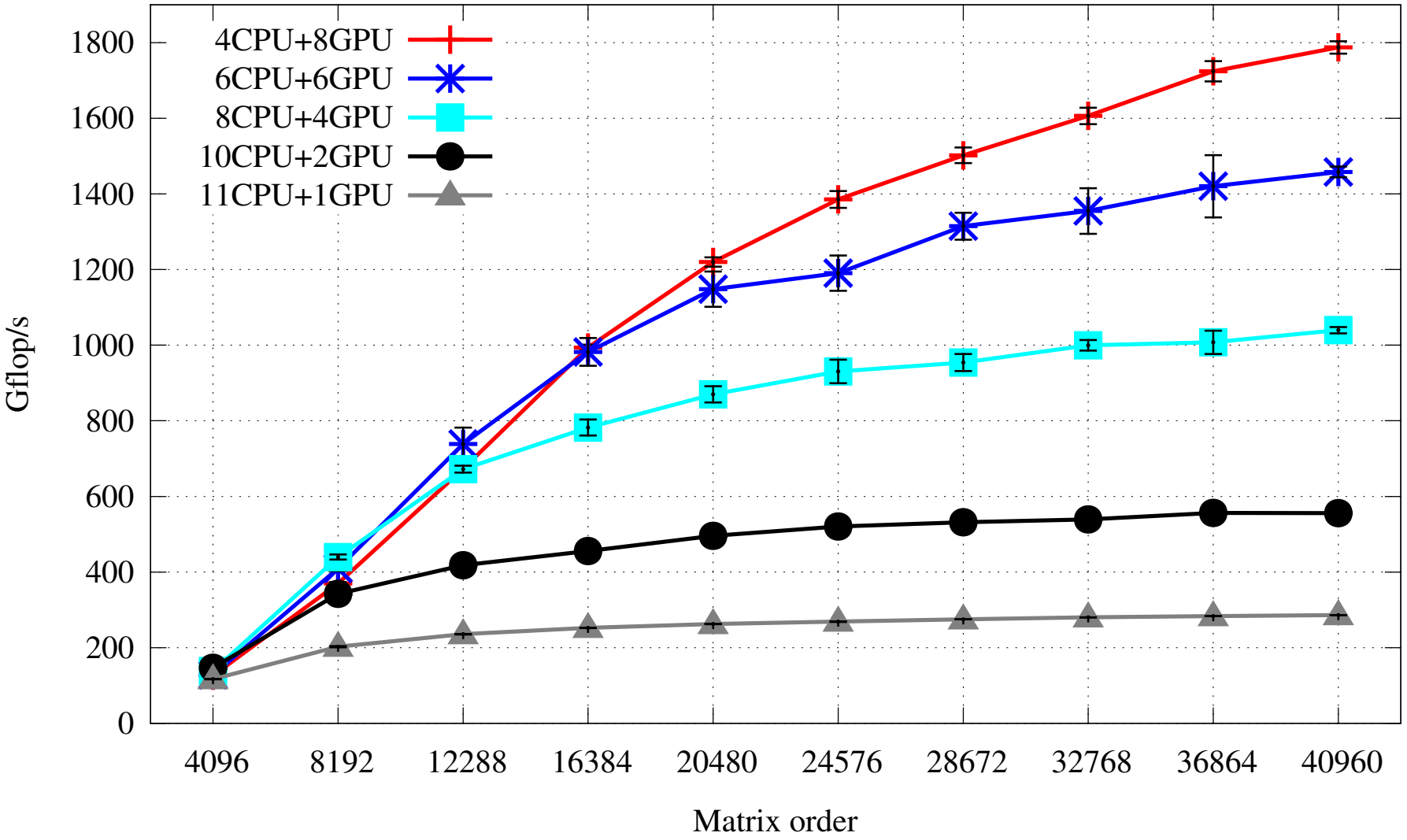


Overlapping GPU communication

- **DGEMM: 1GPU**
  - **CUBLAS DGEMM with and without taking into account data transfers**
  - **XKaapi DGEMM: parallel block version**



Cholesky, multi-CPUs/multi-GPUs



Conclusions

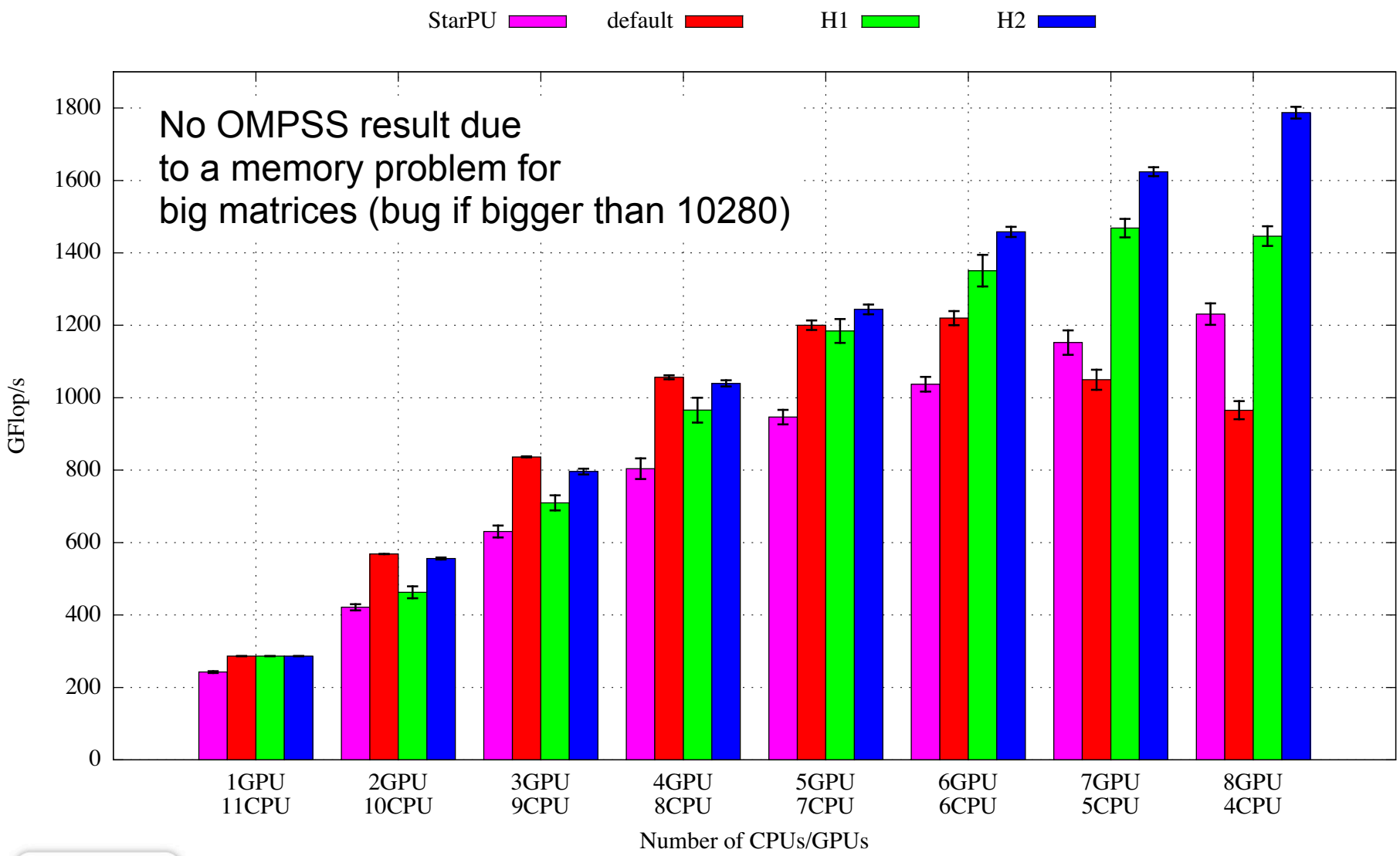
- **OpenMP is good opportunity for research on runtime**
  - GCC libGOMP is public, easy to extend, modify
  - Intel has release public version for its OpenMP runtime
  - good opportunity to promote research development through Intel compiler
- **OpenMP task**
  - improvement to provide fine grain implementation
  - important to simplify the scheduling
- **OpenMP loop**
  - existing schedulers cannot make abstraction of the chunk_size
  - NUMA aware scheduling
- **INRIA ADT K'STAR**
  - to promote runtime (StarPU, XKaapi) through a future OpenMP 4.X standard

Thank you for your attention !

http://kaapi.gforge.inria.fr

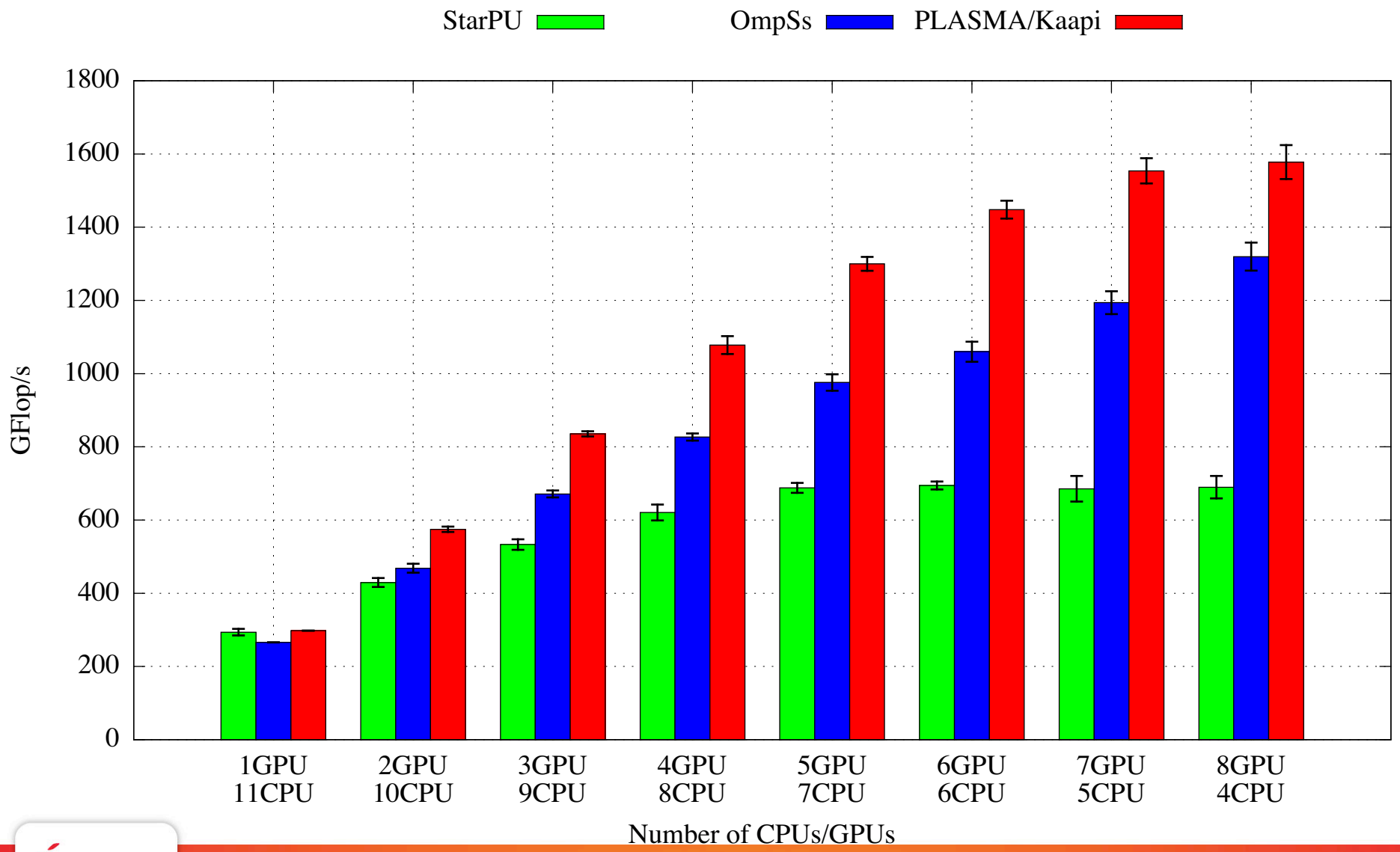
Comparison with OMPSS, StarPU

- DPOTRF matrix size 40960, BS 1024



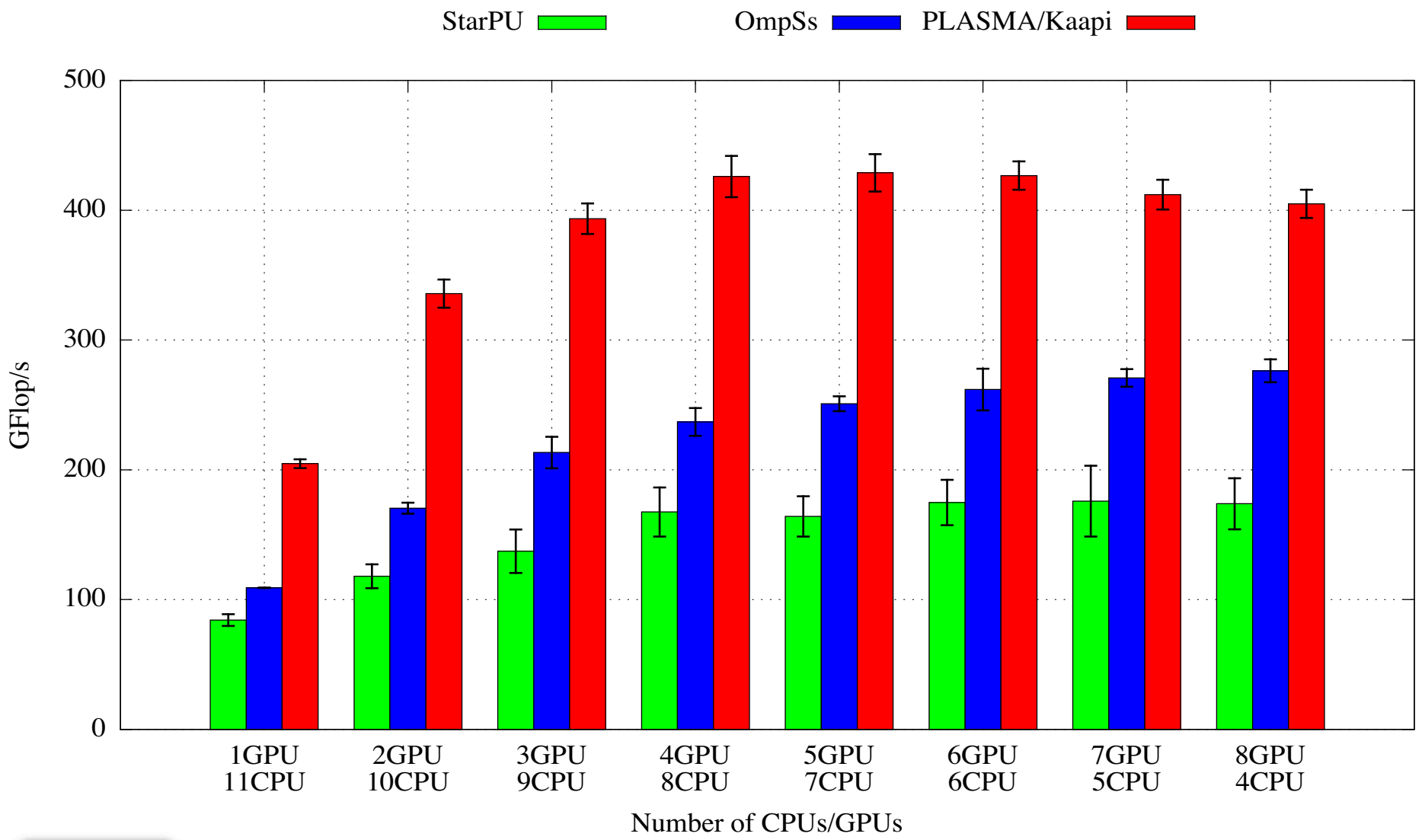
Comparison with OMPSS, StarPU

- DGEMM matrix size 10240, block size 1024



Comparison with OMPSS, StarPU

- DPOTRF matrix size 10240, block size 1024

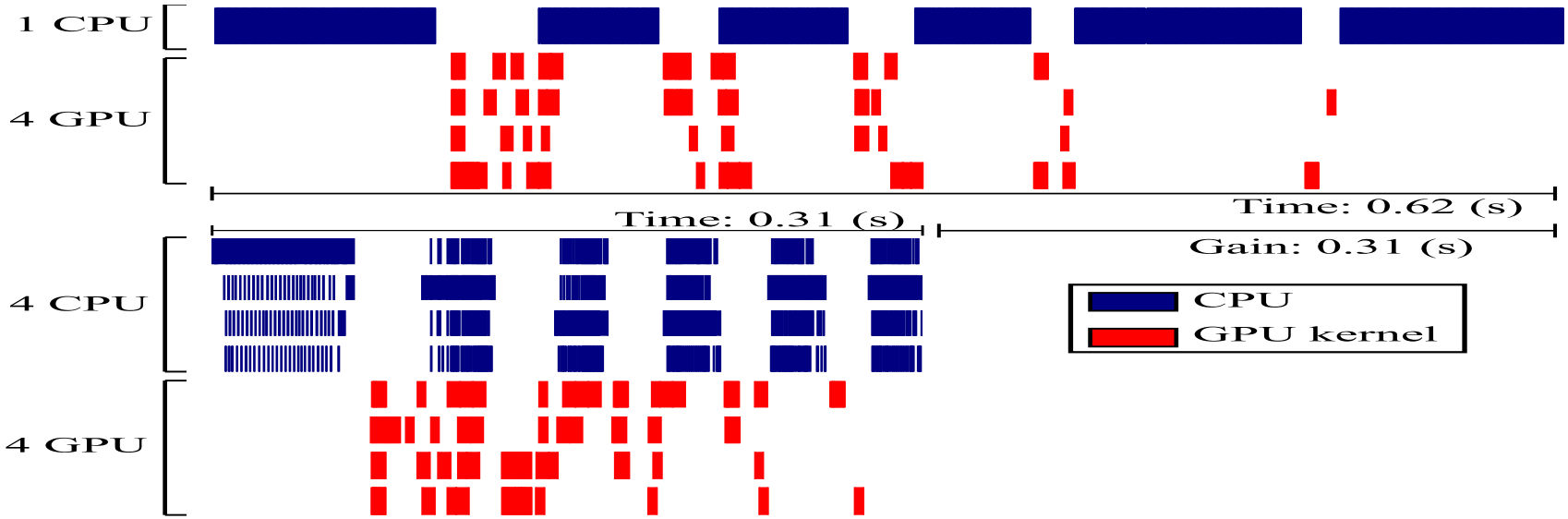


Impact of CPU number

- DPOTRF. Parallel panel factorization on x-CPUs, 4 GPUs.
  - Matrix size 40960, BS=1204

	Matrix dimension				
#CPU	4096	8192	16384	32768	40960
1	53.85 ±0.98	206.38 ±2.70	622.55 ±7.90	962.21 ±31.77	1052.58 ±20.53
4	115.16 ±1.02	391.05 ±2.64	755.91 ±6.89	1013.65 ±7.81	1022.45 ±37.55
8	138.34 ±1.06	439.70 ±3.38	782.21 ±10.51	999.46 ±6.90	1045.53 ±4.19

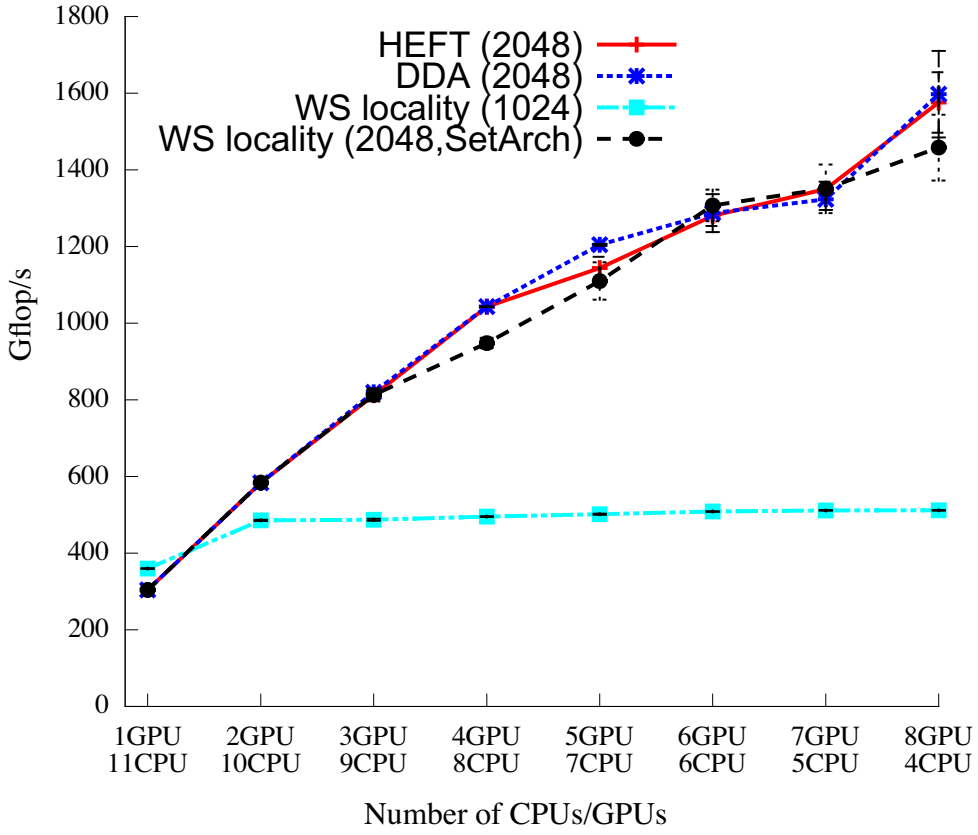
GFlops/s



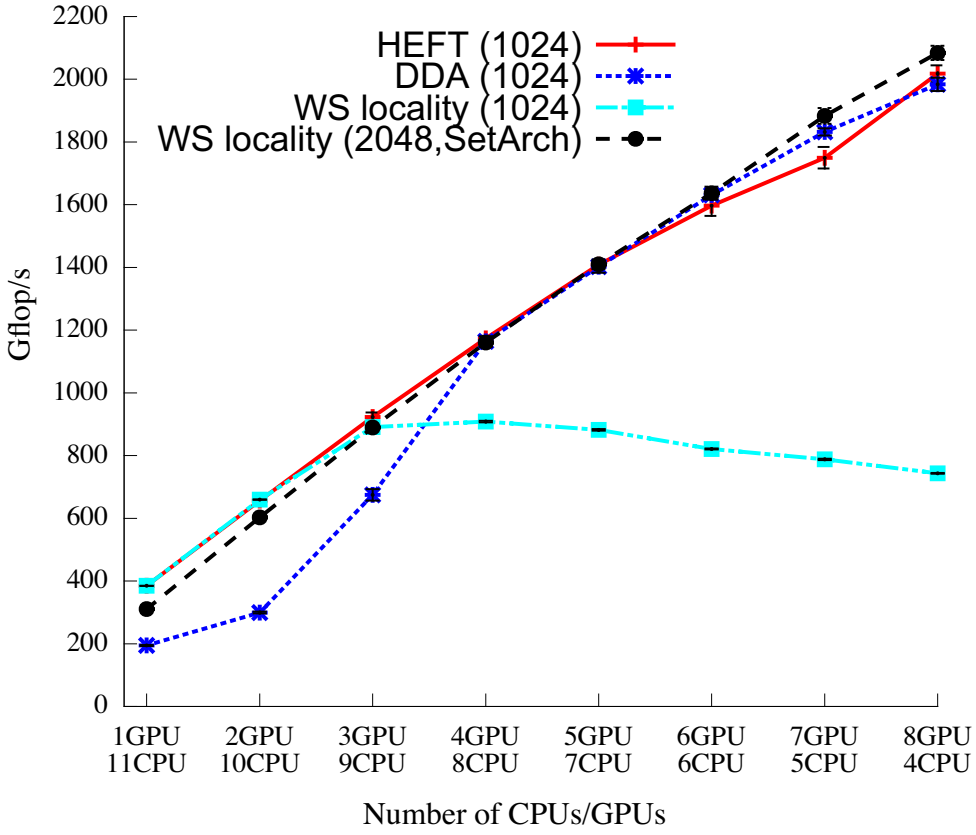
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks

dgemm



Matrix size 8192

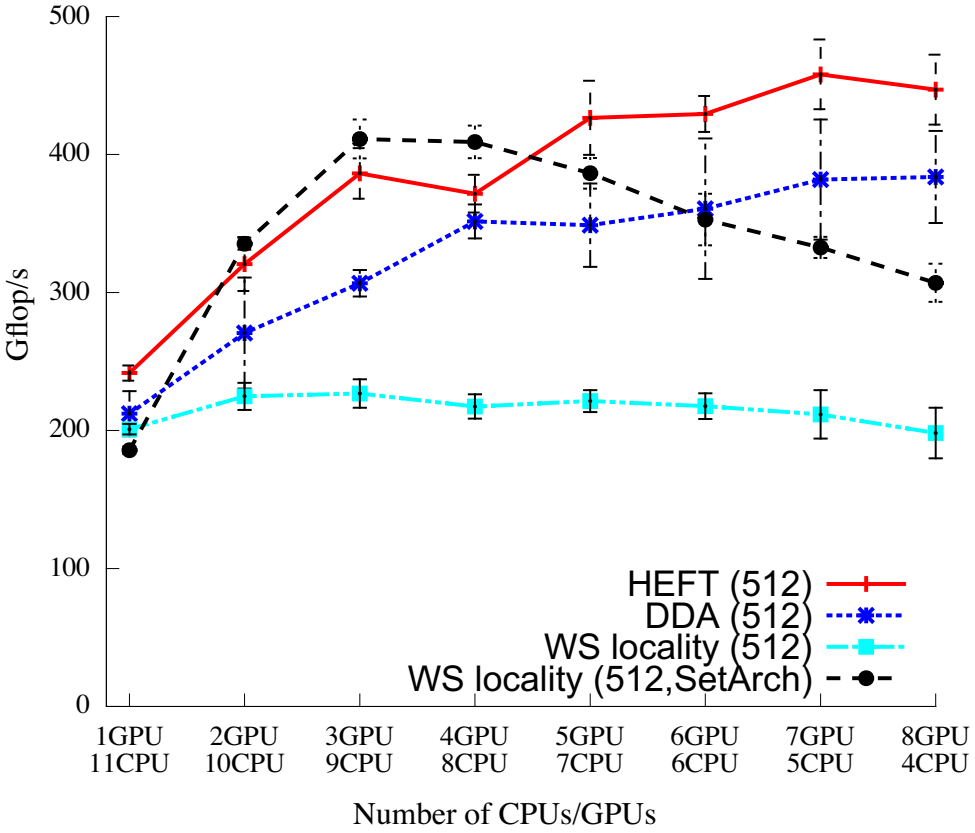


Matrix size 16384

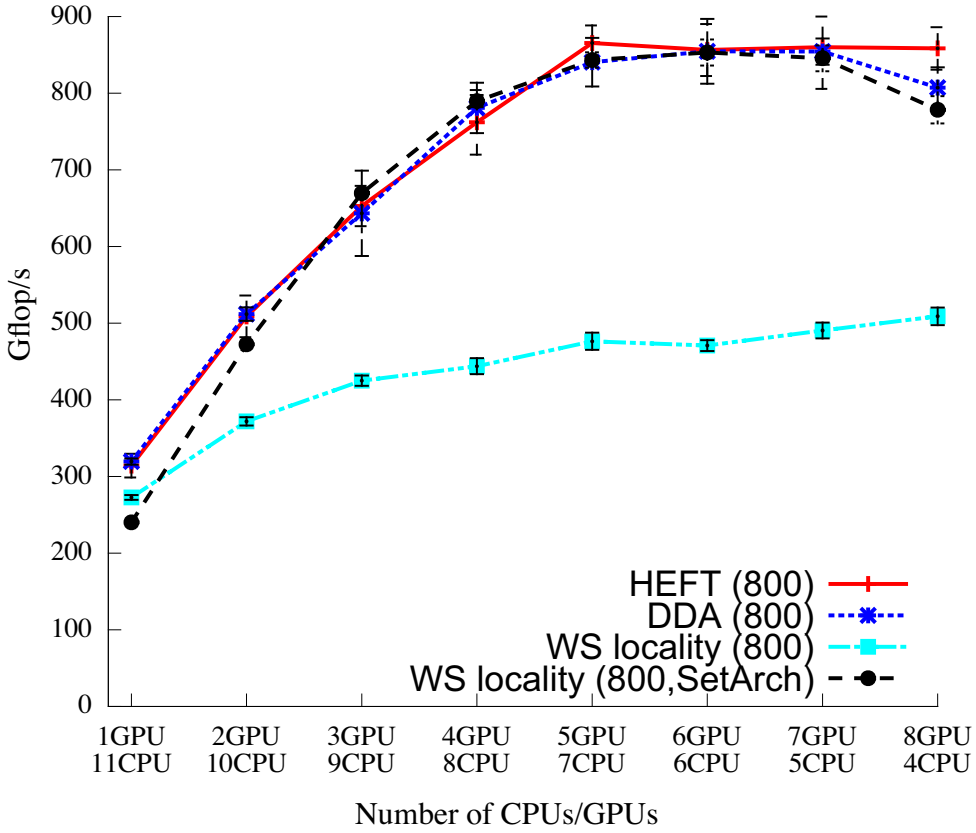
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks

dpotrf



Matrix size 8192

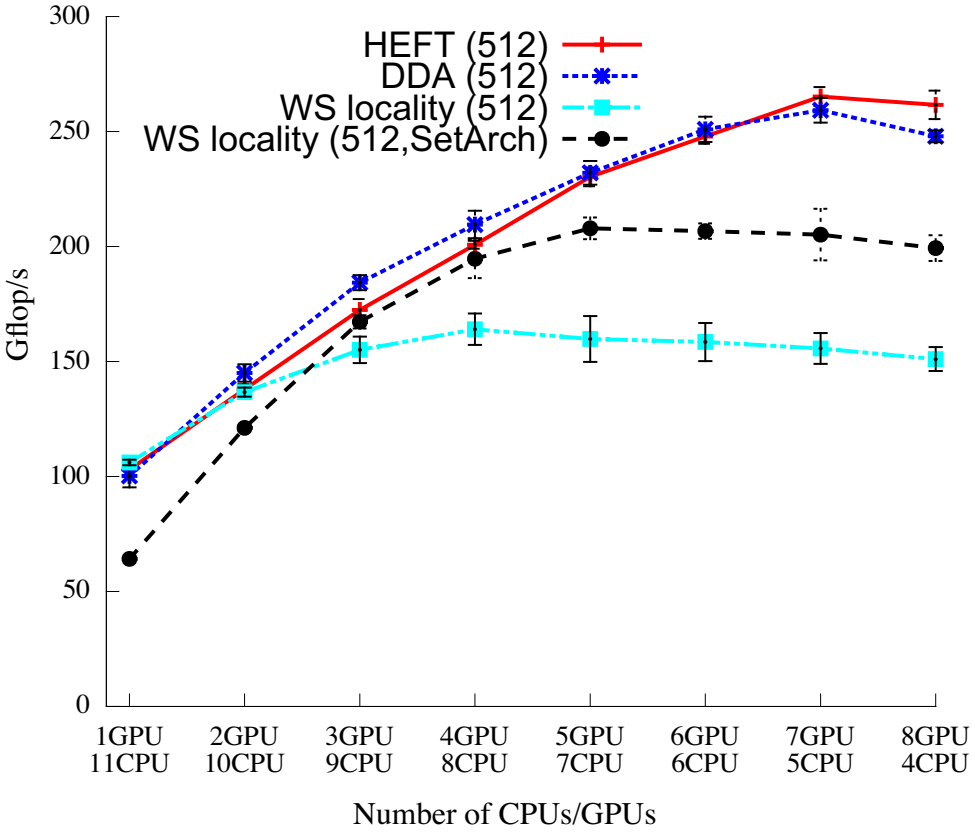


Matrix size 16384

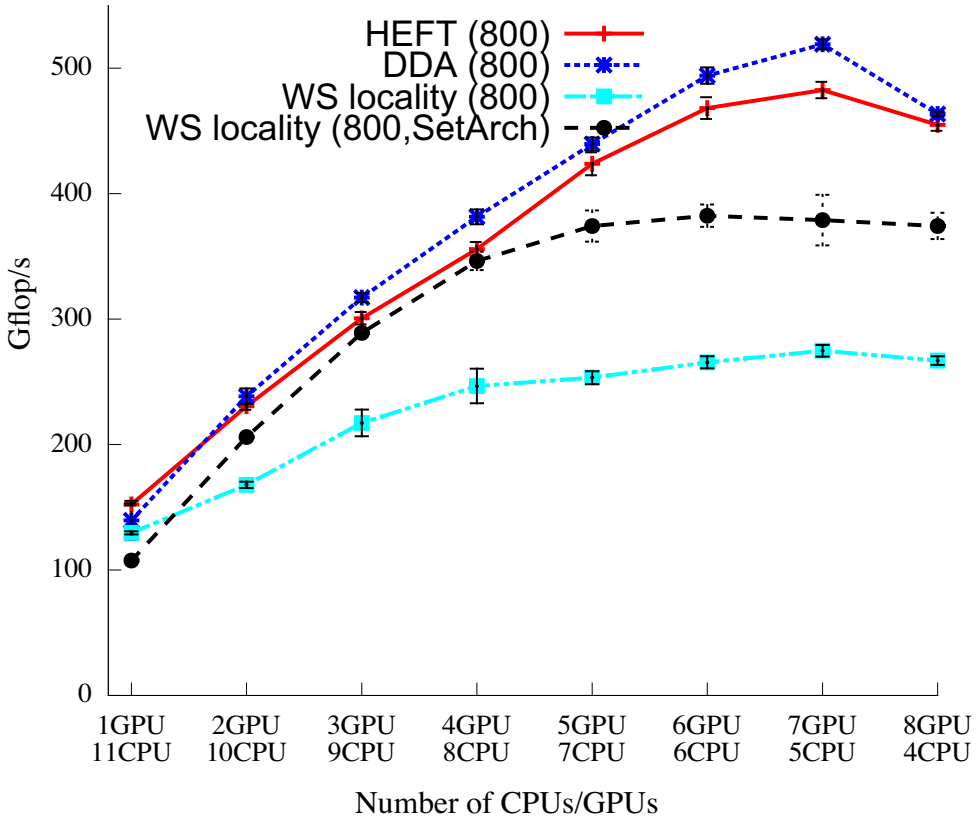
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks

dgetrf



Matrix size 8192

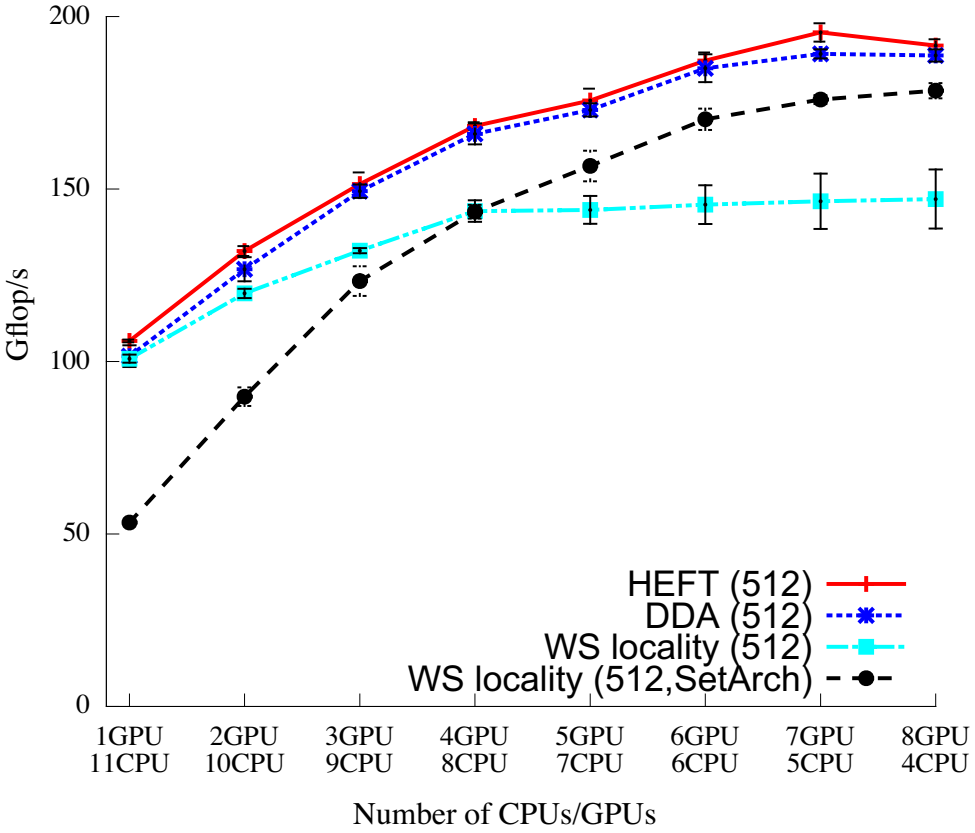


Matrix size 16384

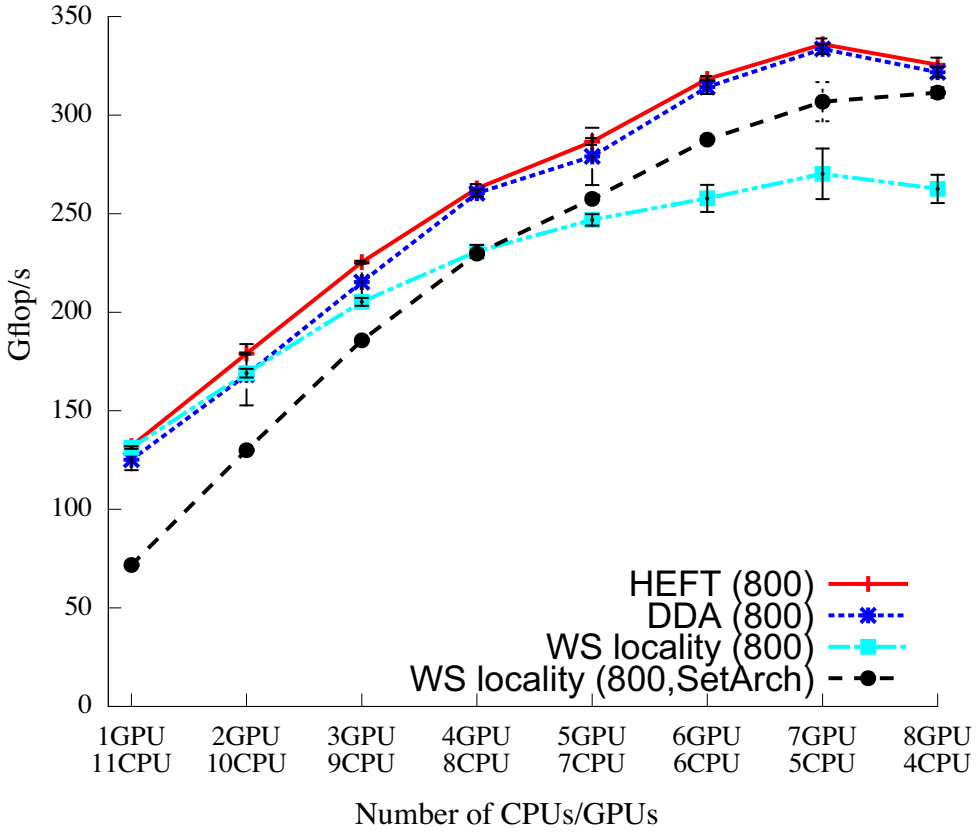
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks

dgeqrf



Matrix size 8192



Matrix size 16384