Resource aggregation in task-based applications over accelerator-based multicore machines

SIAM PP16

T. Cojean[1], A. Guermouche[1], A. Hugo[2], R. Namyst[1], P.-A. Wacrenier[1]

[1] INRIA Bordeaux, LaBRI, Université de Bordeaux

[2] University of Uppsala

Université Pierre et Marie Curie

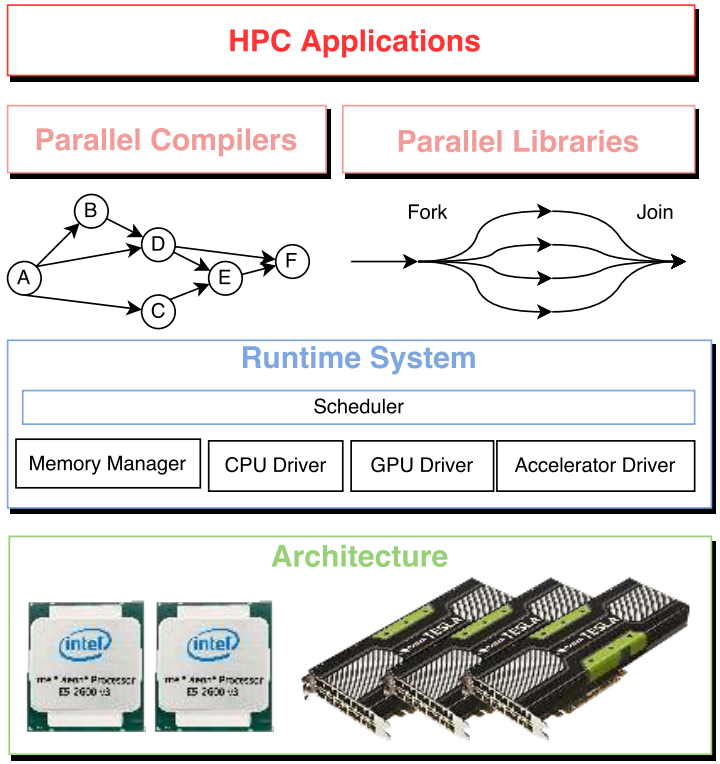
April 12, 2016

Current Computing Platforms and Runtime Systems

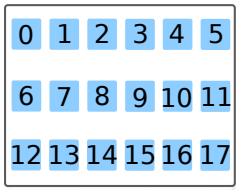
Clusters are increasingly:

- Using CPUs with more cores
- Using GPUs/Accelerators
- Heterogeneous

Design of runtime systems
For portability of performances



Current Computing Platforms and Compute Grain



- Few independent cores
- Low computing power

$\Rightarrow$ Need many small computation

- All at once
- High computing power

$\Rightarrow$ Need one big computation

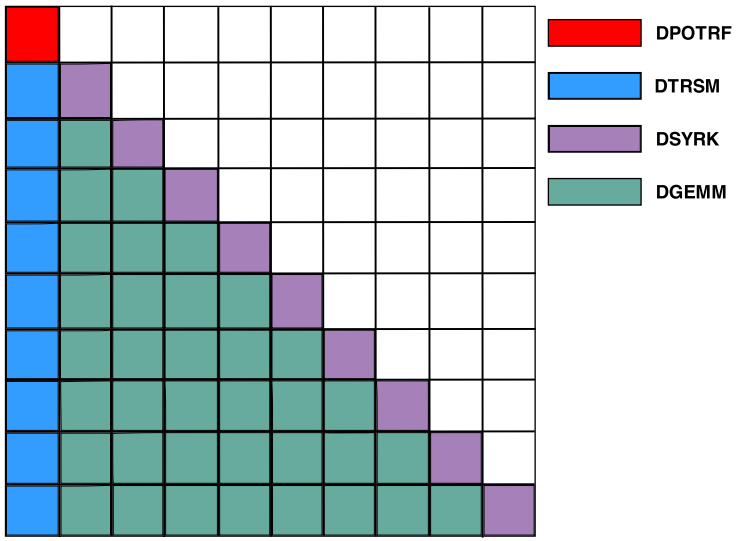
How to meet both CPU and GPU's needs when used concurrently?

Usual approach: trade-off between CPU and GPU cases

Example with a cholesky factorization

For CPUs

```
1  /* MORSE Code */
   for k : 0 → MT-1
3    DPOTRF(A_{k,k})
     for m : k+1 → MT-1
5      DTRSM(A_{k,k}, A_{m,k})
     for n : k+1 → NT-1
7      SYRK(A_{n,k}, A_{n,n})
       for m : n+1 → MT-1
9        DGEMM(A_{m,k}, A_{n,k}, A_{m,n})
```



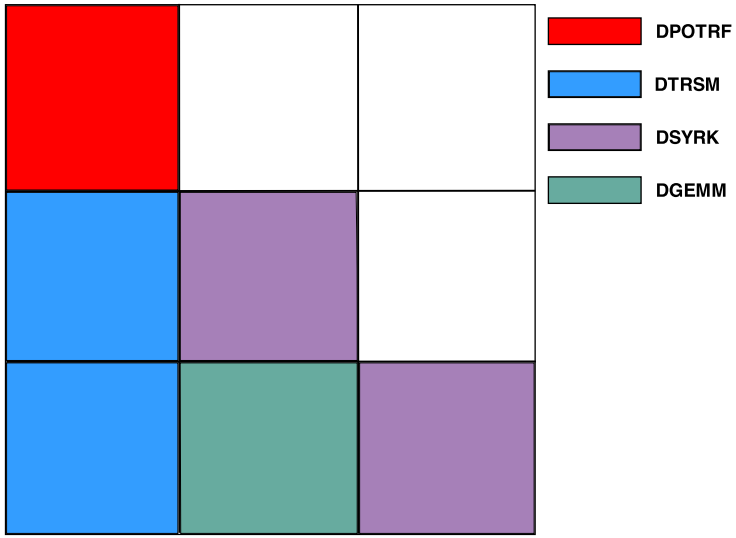
Cholesky Factorization with $k = 0$ and $MT = 10$

Usual approach: trade-off between CPU and GPU cases

Example with a cholesky factorization

For a GPU

```
1  /* MORSE Code */
   for k : 0 → MT-1
3    DPOTRF(A_{k,k})
     for m : k+1 → MT-1
5      DTRSM(A_{k,k}, A_{m,k})
     for n : k+1 → NT-1
7      SYRK(A_{n,k}, A_{n,n})
       for m : n+1 → MT-1
9        DGEMM(A_{m,k}, A_{n,k}, A_{m,n})
```



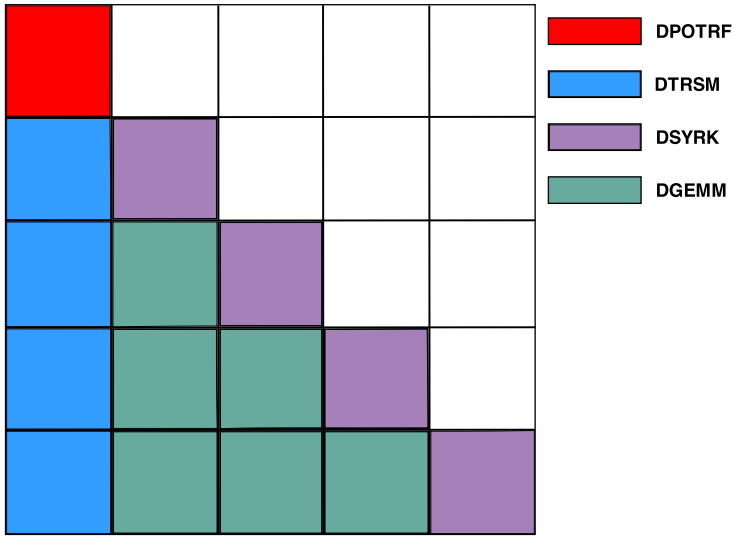
Cholesky Factorization with $k = 0$ and $MT = 3$

Usual approach: trade-off between CPU and GPU cases

Example with a cholesky factorization

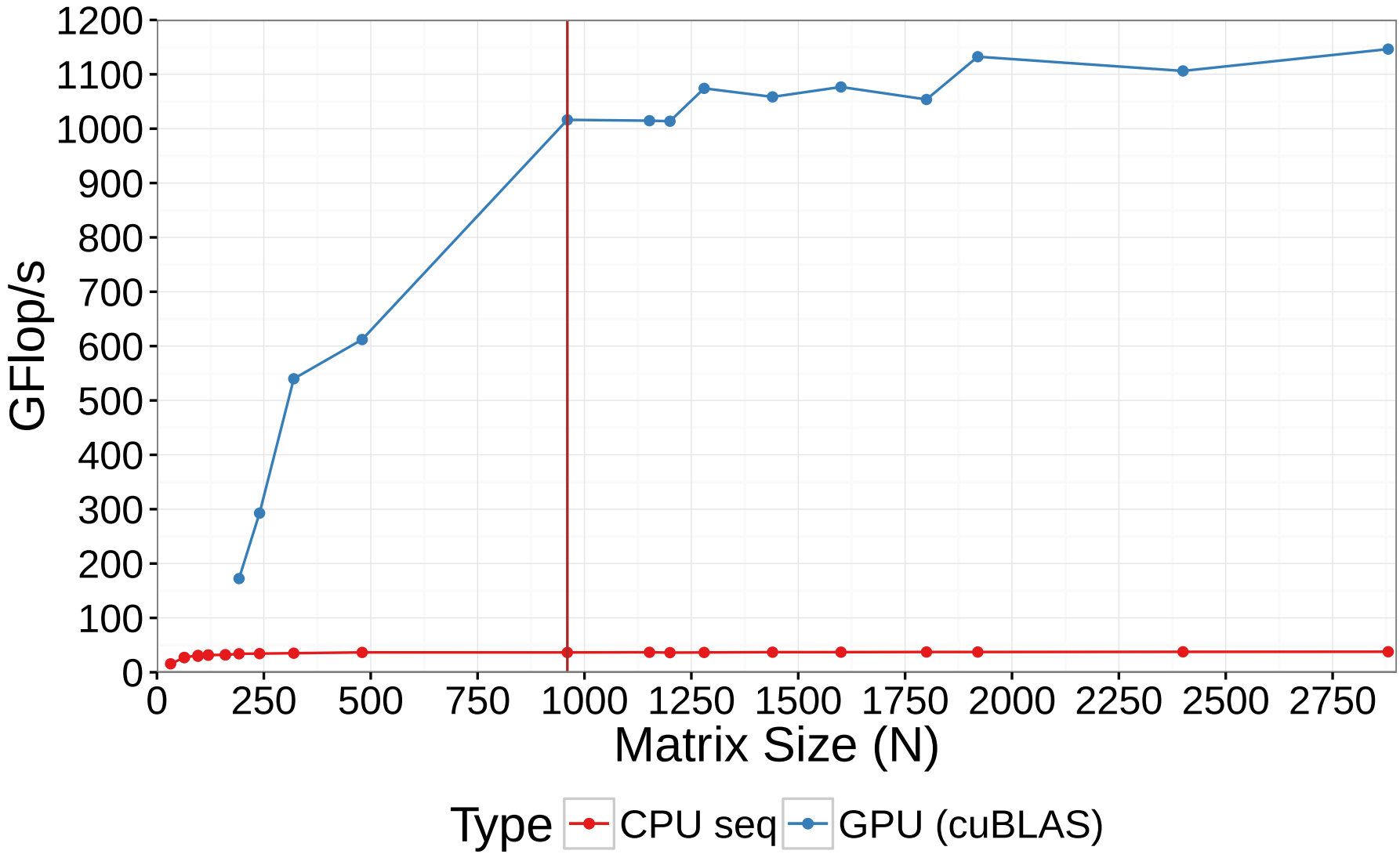
For both: use an intermediate block size

```
1   /* MORSE Code */
    for k : 0 → MT-1
3     DPOTRF(A_{k,k})
      for m : k+1 → MT-1
5       DTRSM(A_{k,k}, A_{m,k})
      for n : k+1 → NT-1
7       SYRK(A_{n,k}, A_{n,n})
        for m : n+1 → MT-1
9         DGEMM(A_{m,k}, A_{n,k}, A_{m,n})
```

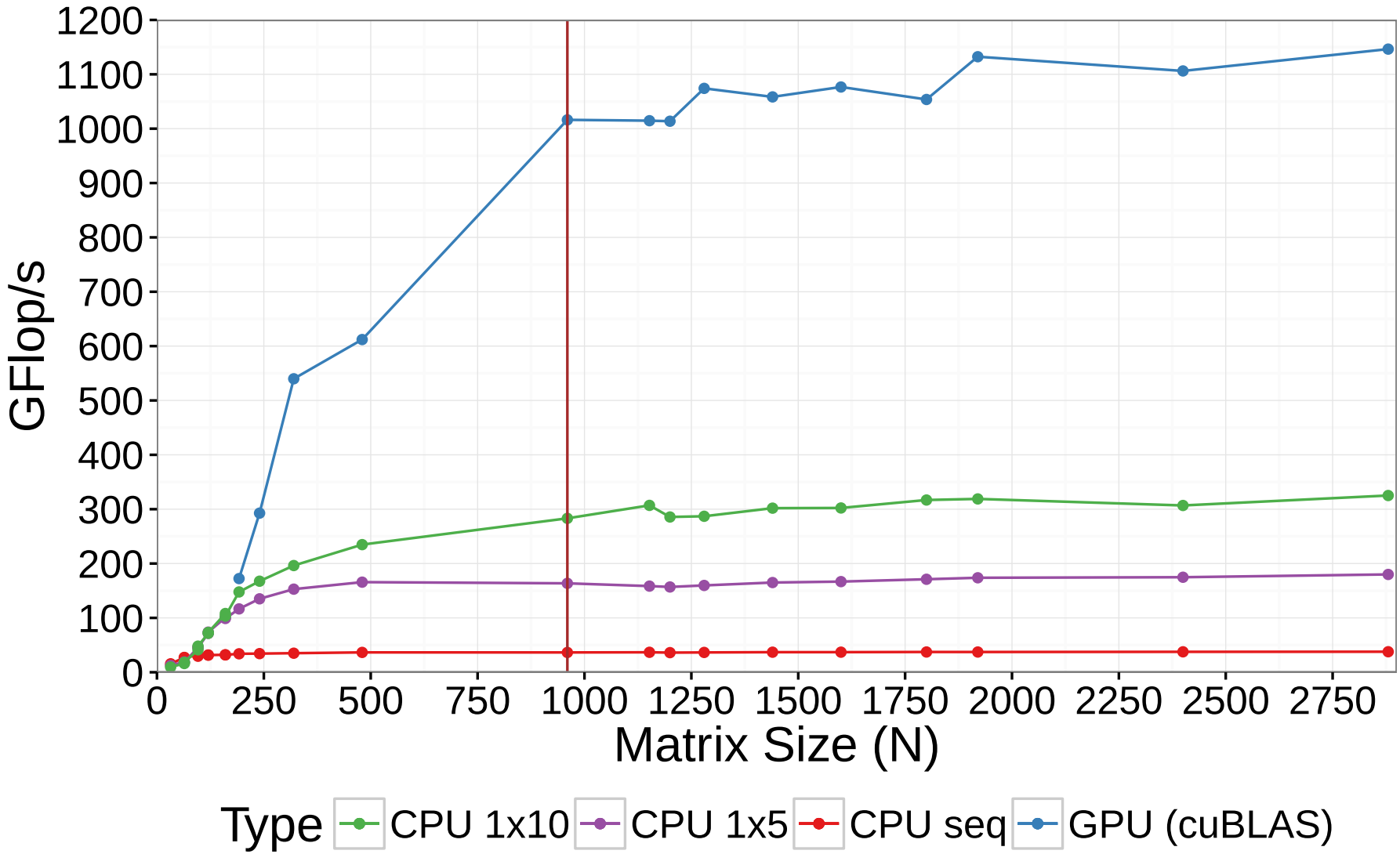


Cholesky Factorization with $k = 0$ and $MT = 5$

Case study: DGEMM CPU vs GPU performance

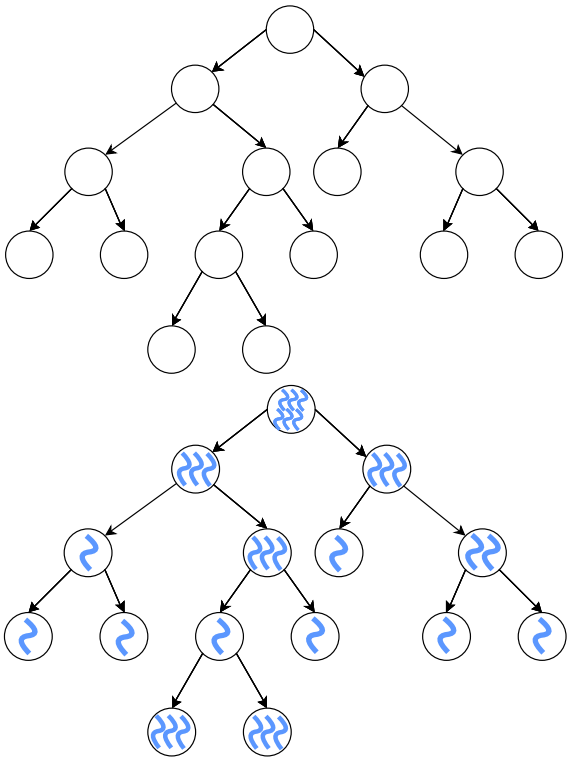


Case study: DGEMM CPU vs GPU performance



Another approach: parallel tasks

Proposed solution



Two parallelism levels

- Task parallelism
- Internal-task parallelism

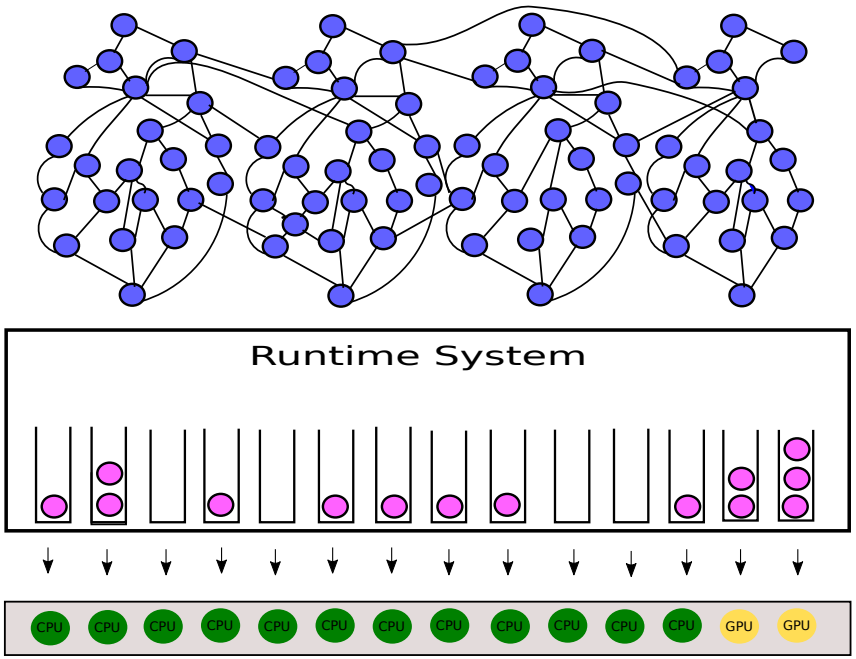
A task has a new parameter: the amount of threads it can run on.

Here:

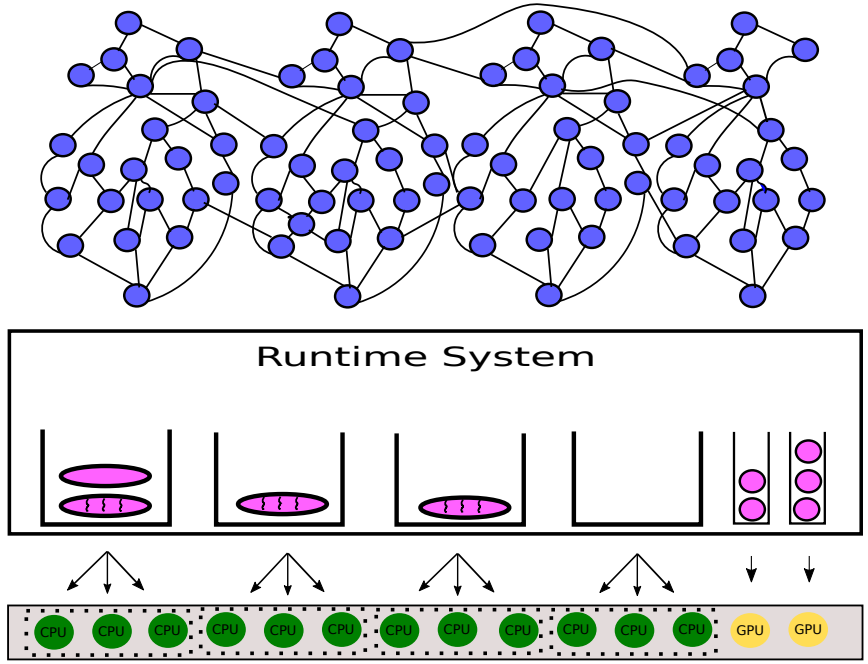
- Task parallelism: StarPU
- Internal parallelism: can be anything.
  Good candidate: OpenMP (Parallel MKL)

Clustering CPU cores

Usual StarPU use case

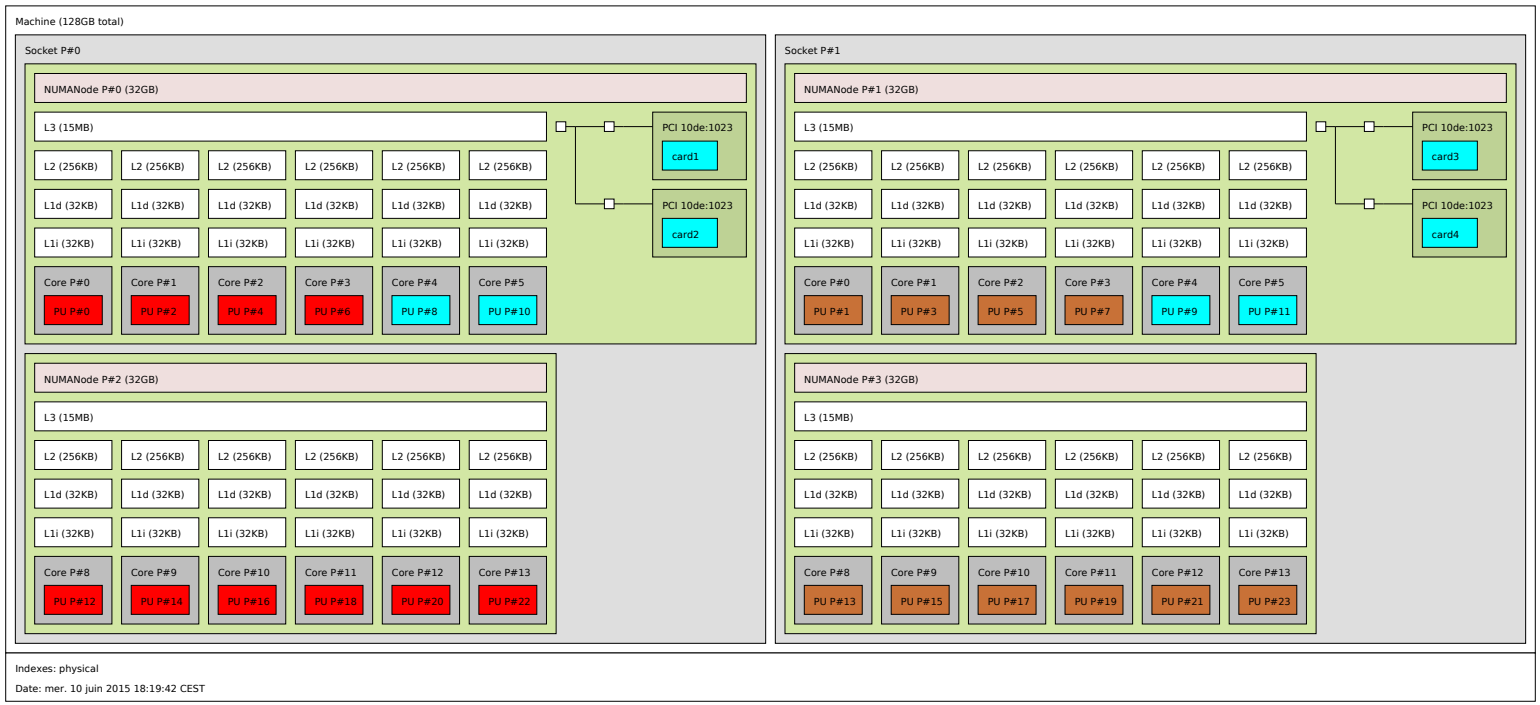


StarPU with clustered cores



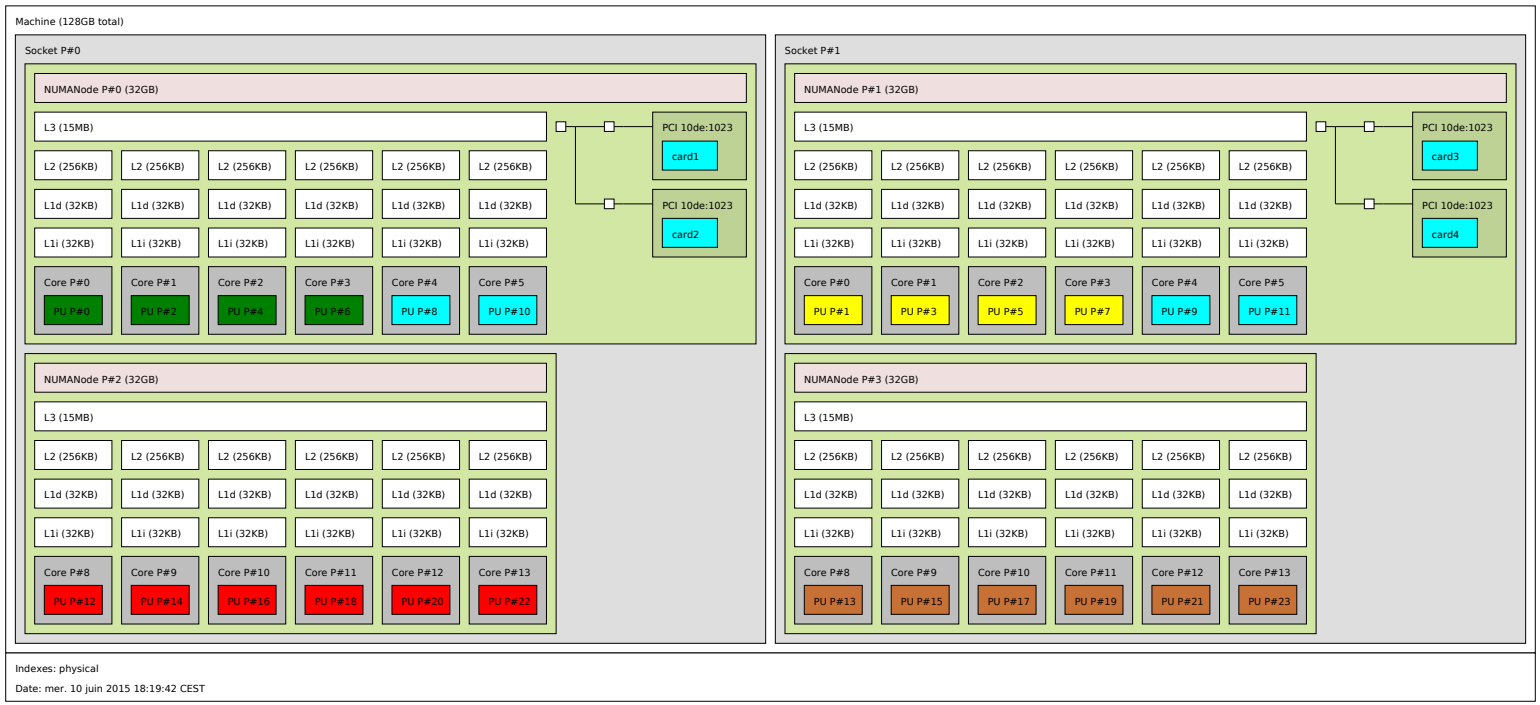
Example: clustering inside processors' sockets

```
starpu_clusters *clusters;
/* ... */
clusters = starpu_cluster_machine(HWLOC_OBJ_SOCKET,
                    STARPU_CLUSTER_TYPE, GNU_OPENMP_MKL,
                    0);
/* Parallel tasks submission and computation */
START_TIMING();
MORSE_zpotrf_Tile(uplo, descA);
STOP_TIMING();

starpu_uncluster_machine(clusters);
```



Example: clustering inside processors' NUMA nodes

```
1  starpu_clusters *clusters;
   /* ... */
3  clusters = starpu_cluster_machine(HWLOC_OBJ_NUMANODE,
                    STARPU_CLUSTER_TYPE, GNU_OPENMP_MKL,
5                   0);
   /* Parallel tasks submission and computation */
7  START_TIMING();
   MORSE_zpotrf_Tile(uplo, descA);
9  STOP_TIMING();

1  starpu_uncluster_machine(clusters);
```



Example : GNU OpenMP for internal parallelism

Aim: ensure the **internal runtime** uses the **StarPU assigned resources** for the task.

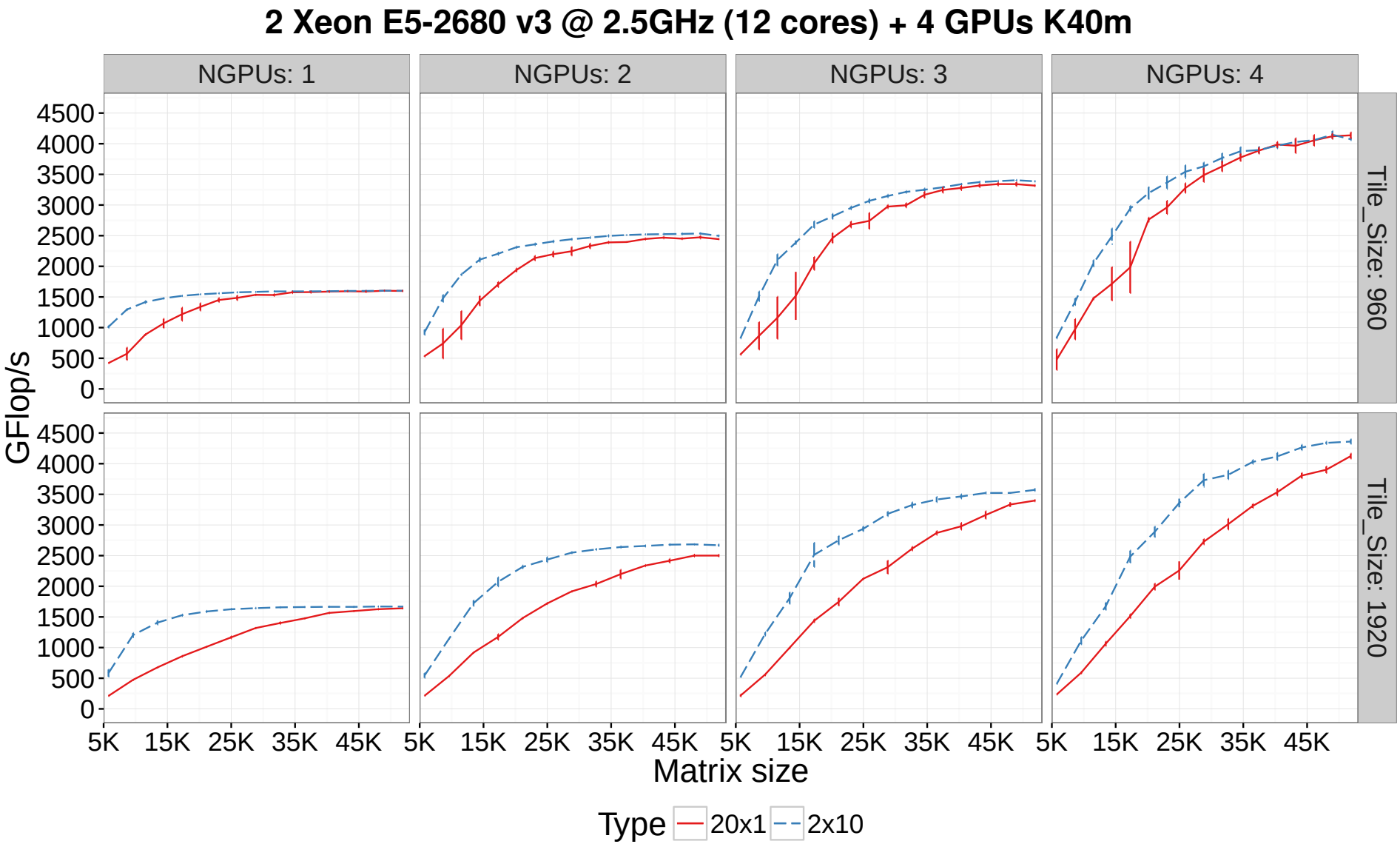
Before task execution

```
1  void starpu_gnu_openmp_mkl_prologue(void *ctx)
   {
3    /* get the CPUs of the current context */
     int cpus[MAX_WORKERS];
5    int ncpus = starpu_get_cpus(ctx, cpus);

7    /* bind openmp threads to CPUs */
   #pragma omp parallel num_threads(ncpus)
9    bind_to_cpu(cpus[omp_get_thread_num()]);
   }
```

Fetch available resources

Create and bind thread team

Exhaustive study



Why are we better?

Some answers or guesses come naturally:

		DPOTRF	DTRSM	DSYRK	DGEMM
960	1 core	1.8	8.8	25.4	28.6
	5 cores	0.42	2.0	5.4	6.3
	10 cores	0.34	1.3	3.7	3.6
1920	1 core	5.8	18.5	30.3	30.7
	5 cores	1.28	4.0	6.6	6.5
	10 cores	0.78	2.1	3.5	3.6

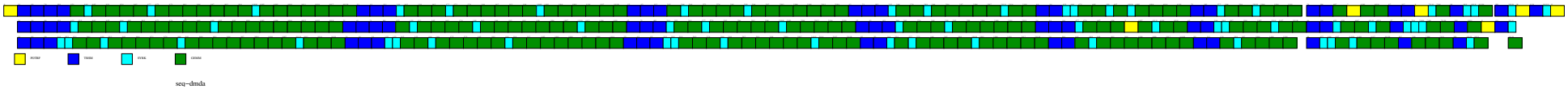
- Homogeneization of CPU and GPU **granularity of computations**
- New compromises on **tile size**

But to really understand, we tried several things.

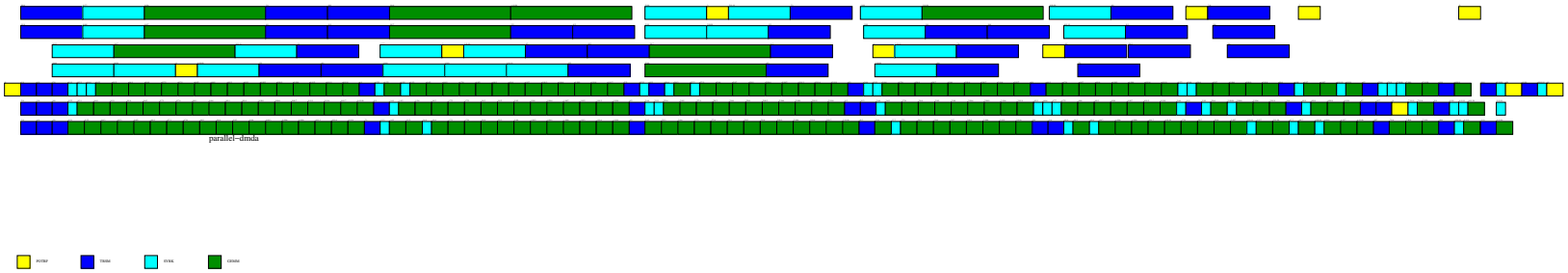
- Scheduling study
- Bounds computation

Scheduling: dmda with tile size of 1920 and 12 tiles

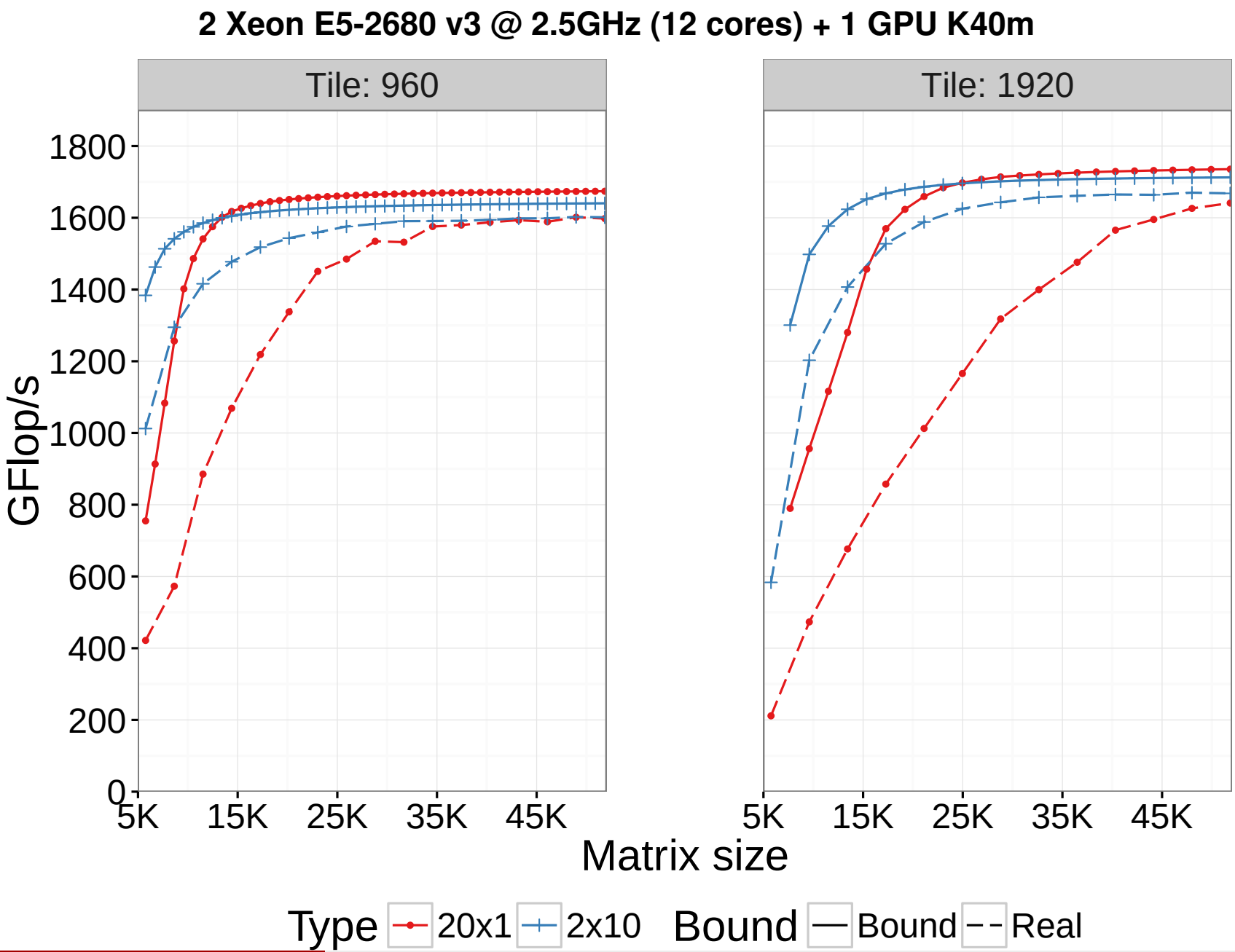
Sequential tasks



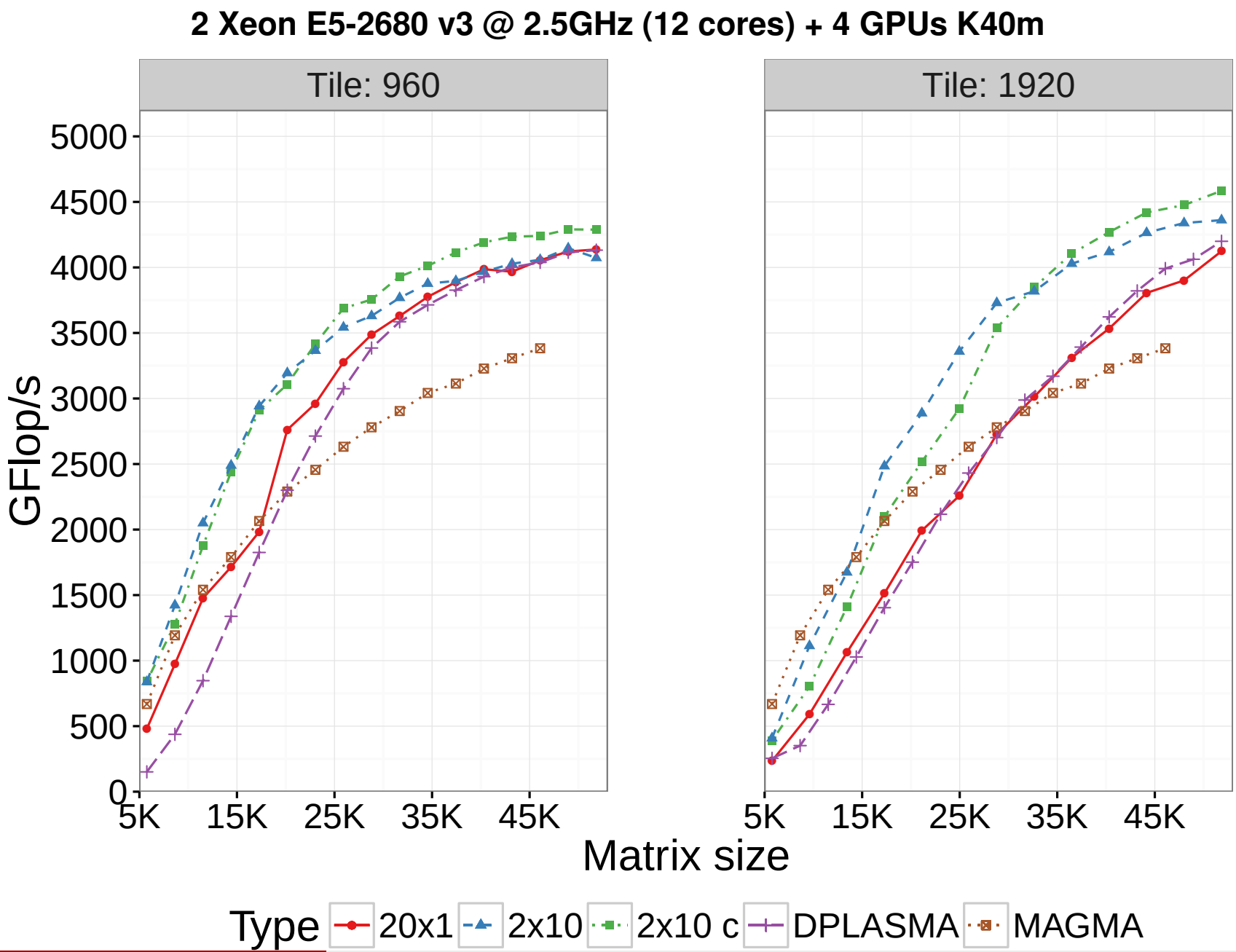
Parallel tasks



Bounds versus real execution



Comparison with existing frameworks



Conclusion

Summary

- Implementation of parallel tasks model in StarPU
- Interface to manage a static partition/clustering of the machine
- Good performance obtained on sirocco with cholesky
- Looked at several aspects to understand the gains
  - Scheduling study
  - Bounds computation

Future/Ongoing works

- Use Intel KNL as soon as available!
- Schedulers able to manage resource size?
- Tests on more algorithms/kernels to come!
- Explore the “opposite”: splitting tasks.