Toward a supernodal sparse direct solver over DAG runtimes

Sparse Days 2013, Toulouse

X. Lacoste

Xavier LACOSTE
HiePACS team – Inria Bordeaux Sud-Ouest

November 27, 2012

Guideline

1
Context and goals

Context and goals

- Robust and efficient $Ax = b$ resolution using direct factorization
  - $\rightarrow$ PaStiX direct solver
- Factorization is time consuming, good performance required
- Emerging machines with many-cores and multiple GPUs
  - $\rightarrow$ use it all !

Possible solutions

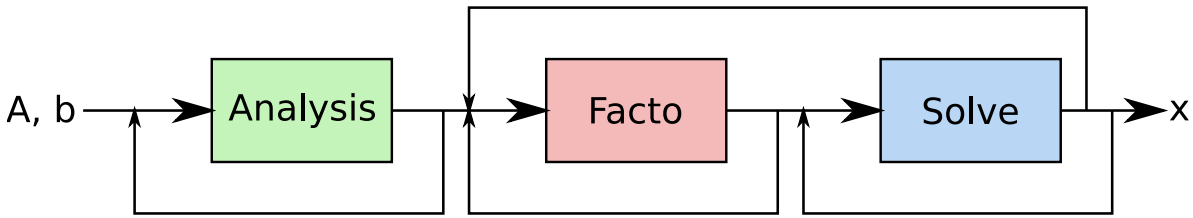
- Many-cores : PaStiX already finely tuned to using MPI and P-Threads;
- Multiple-GPU and many-cores, two solutions:
  - Manually handle GPUs:
    - lot of work;
    - heavy maintenance;
  - Use dedicated runtime:
    - May loose the performance obtained on many-core;
    - Easy to add new computing devices.

Elected solution, runtime:

- StarPU: RUNTIME – Inria Bordeaux Sud-Ouest;
- PaRSEC: ICL – University of Tennessee, Knoxville.

Major steps for solving sparse linear systems

1. Analysis: matrix is preprocessed to improve its structural properties ($A'x' = b'$ with $A' = P_n P D_r A D_c Q P^T$)
2. Factorization: matrix is factorized as $A = LU$, LL^T or LDL^T
3. Solve: the solution x is computed by means of forward and backward substitutions



2
Kernels

Panel factorization

- Factorization of the diagonal block (xxTRF);
- TRSM on the extra-diagonal blocks (ie. solves $X \times b_d = b_{i,i>d}$ – where b_d is the diagonal block).

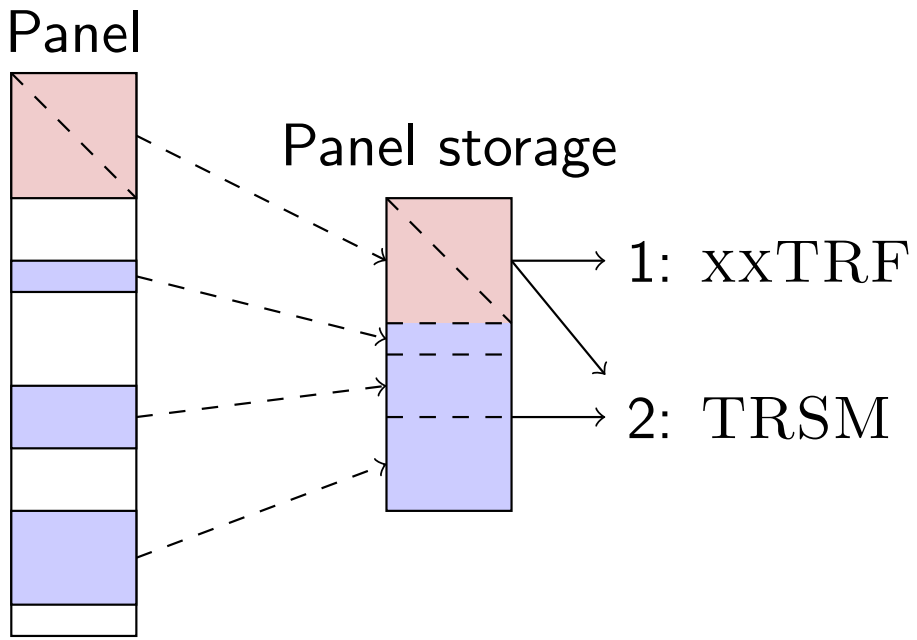


Figure: Panel update

Trailing supernodes update

- One global GEMM in a temporary buffer;
- Scatter addition (many AXPY).

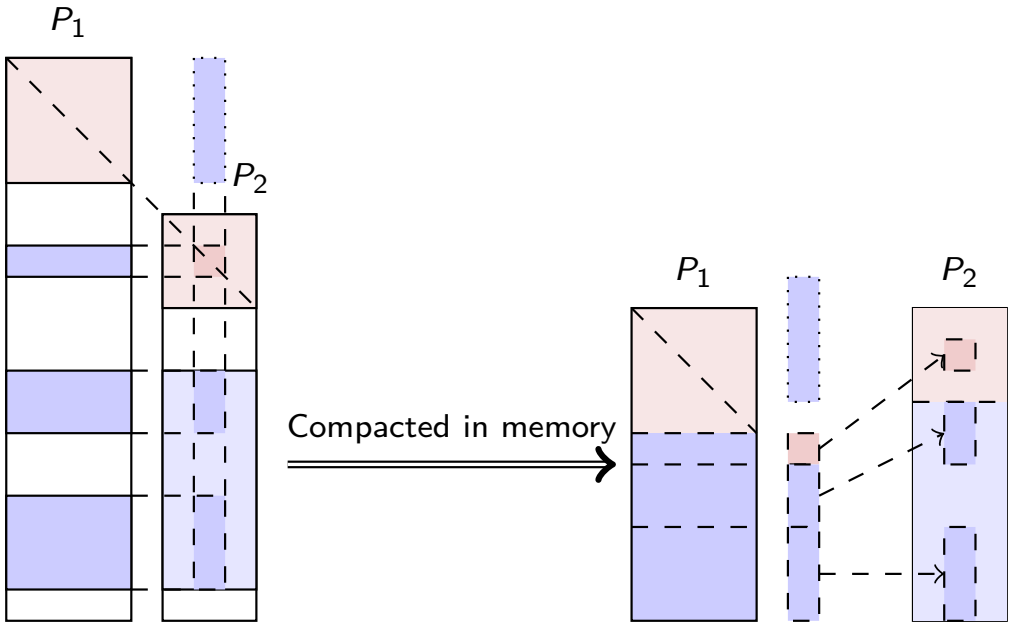


Figure: Panel update

Why a new kernel ?

- A BLAS call $\Rightarrow$ a CUDA startup paid;
- Many AXPY calls $\Rightarrow$ loss of performance.

$\Rightarrow$ need a GPU kernel to compute all the updates from P_1 on P_2 at once.

How ?

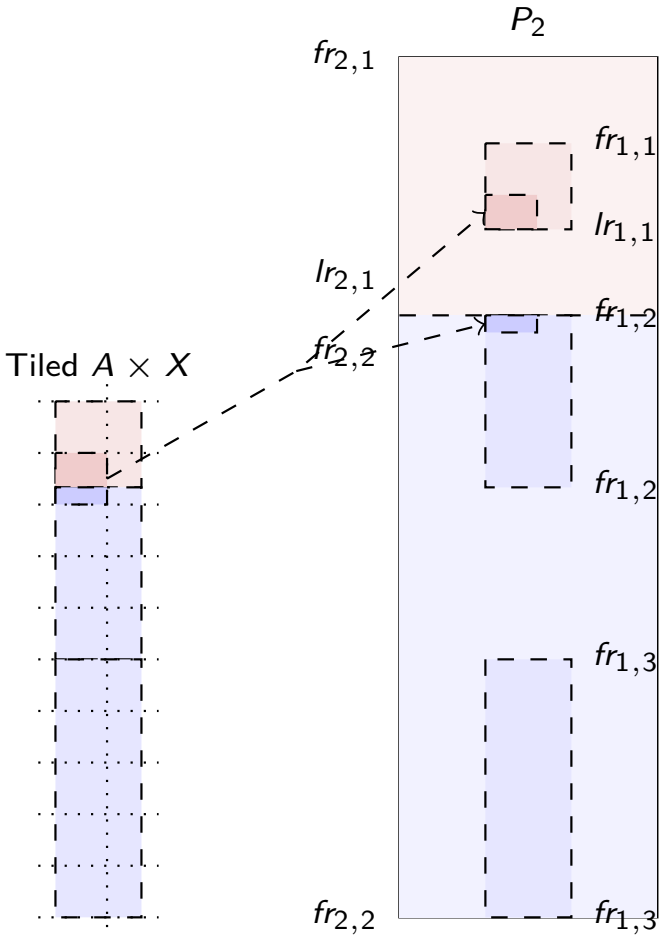
auto-tunning GEMM CUDA kernel

- Auto-tunning done by the framework ASTRA developped by Jakub Kurzak for MAGMA and inspired from ATLAS;
- computes $C \leftarrow \alpha AX + \beta B$, AX split into a 2D tiled grid;
- a block of threads computes each tile;
- each thread computes several entries of the tile in the shared memory and substract it from C in the global memory.

Sparse GEMM cuda kernel

- Based on auto-tuning GEMM CUDA kernel;
- Added two arrays giving first and last line of each blocks of P_1 and P_2;
- Computes an offset used when adding to the global memory.

Sparse GEMM on GPU



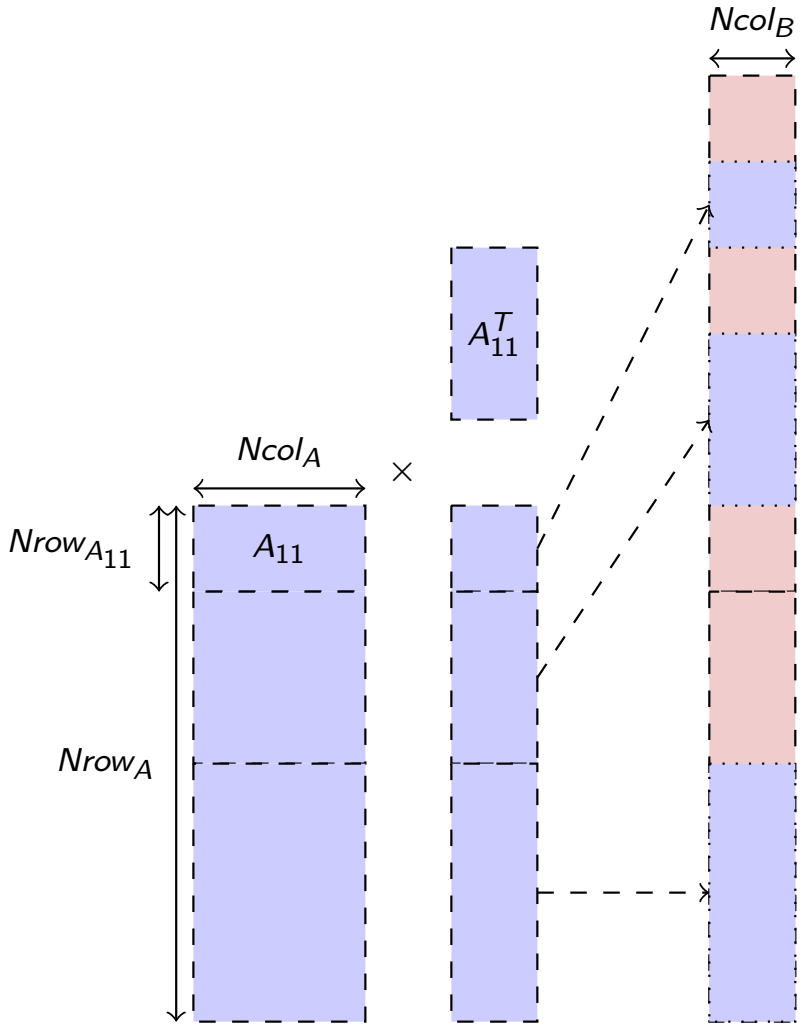
blocknbr = 3;
blocktab = [$fr_{1,1}$, $lr_{1,1}$,
$fr_{1,2}$, $lr_{1,2}$,
$fr_{1,3}$, $lr_{1,3}$];

fblocknbr = 2;
fblocktab = [$fr_{2,1}$, $lr_{2,1}$,
$fr_{2,2}$, $lr_{2,2}$];

```
sparse_gemm_cuda( char TRANSA, char TRANSB, int m, int n,
                  cuDoubleComplex alpha,
                  const cuDoubleComplex *d_A, int lda,
                  const cuDoubleComplex *d_B, int ldb,
                  cuDoubleComplex beta,
                  cuDoubleComplex *d_C, int ldc,
                  int blocknbr, const int *blocktab,
                  int fblocknbr, const int *fblocktab,
                  CUstream stream );
```

Figure: Panel update on GPU

GPU kernel experimentation



Parameters

- $Ncol_A = 100$;
- $Ncol_B = Nrow_{A_{11}} = 100$;
- $Nrow_A$ varies from 100 to 2000;
- Random number and size of blocks in A;
- Random blocks in B matching A;
- Get mean time of 10 runs for a fixed $Nrow_A$ with different blocks distribution.

Figure: GPU kernel experimentation

GPU kernel performance

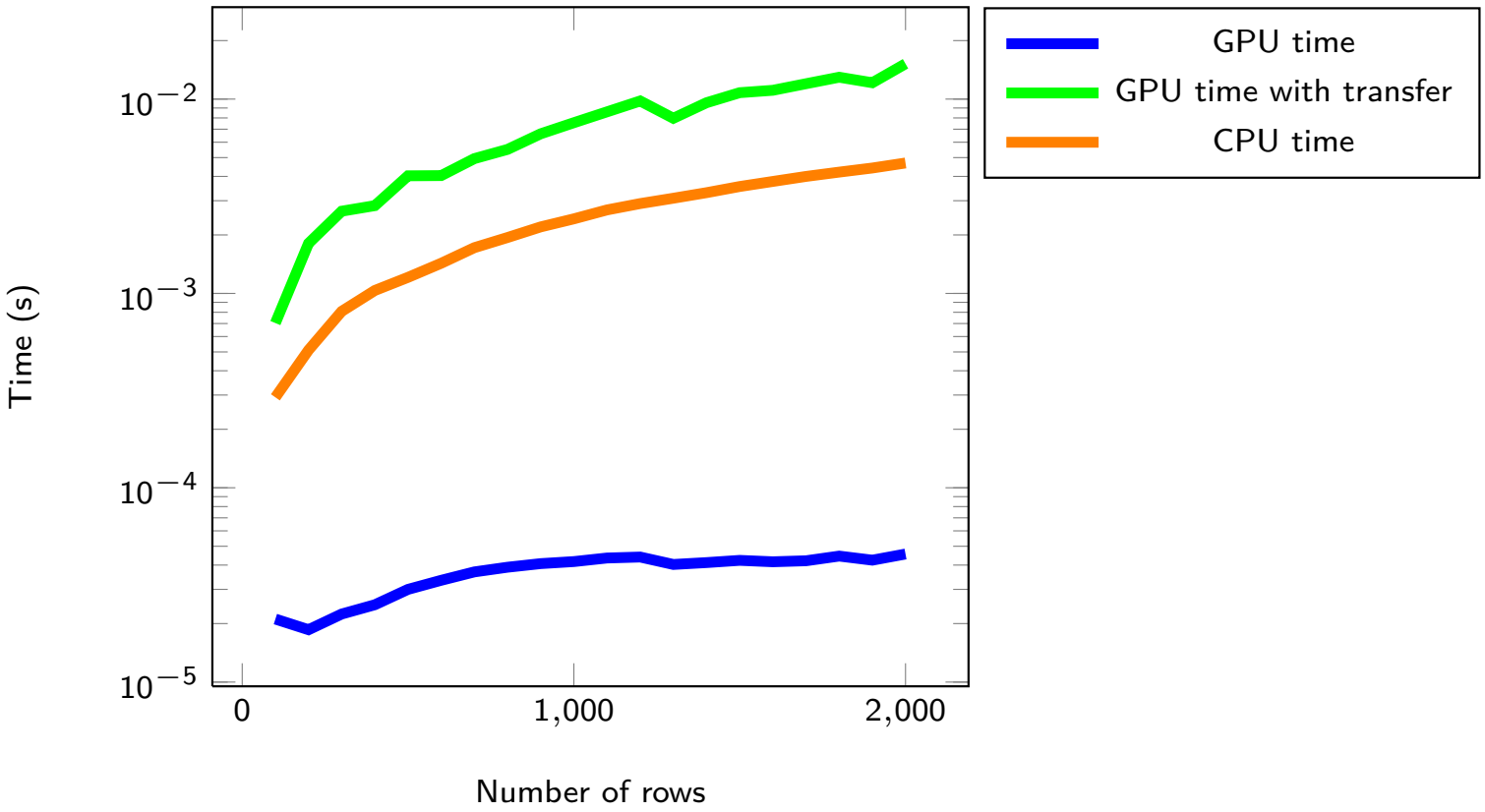


Figure: Sparse kernel timing with 100 columns.

3
Runtime

Runtimes

- Task-based programming model;
- Tasks scheduled on computing units (CPUs, GPUs, ...);
- Data transfers management;
- Dynamicaly build models for kernels;
- Add new scheduling strategies with plugins;
- Get informations on idle times and load balances.

StarPU Tasks submission

Algorithm 1: StarPU tasks submission

forall the *Supernode* S_1 **do**
  submit_panel (S_1);
  /* update of the panel */
  forall the *extra diagonal block* B_i *of* S_1 **do**
    $S_2 \leftarrow$ supernode_in_front_of (B_i);
    submit_gemm (S_1,S_2);
    /* sparse GEMM $B_{k,k \geq i} \times B_i^T$ substracted from S_2 */
  end
end

PaRSEC's parametrized taskgraph

```
panel(j) [high_priority = on]
/* execution space */
j = 0 .. cblknbr-1
/* Extra parameters */
firstblock = diagonal_block_of( j )
lastblock = last_block_of( j )
lastbrow = last_brow_of( j ) /* Last block generating an update on j */
/* Locality */
:A(j)
RW A ← leaf ? A(j) : C gemm(lastbrow)
       → A gemm(firstblock+1..lastblock)
       → A(j)
```

Figure: Panel factorization description in PaRSEC

Giving more information to the runtime

Definition of a new scheduler PaStiX *work stealing*

- Use PaStiX static tasks placement;
- steal tasks from other contexts when no more tasks are ready (based on StarPU work stealing policy).

Choose which GEMM will run on GPUs

- staticaly decide to place only some panels on GPUs following a given criterium :
  - panel size;
  - number of update on the panel;
  - number of flops for the panel update.
- PaRSEC can place task on a given GPU whereas it's more complicated with StarPU.

4
Results on DAG runtimes

Matrices and Machines

Matrices

Name	N	NNZ_A	Fill ratio	OPC	Fact
MHD	4.86×10^5	1.24×10^7	61.20	9.84×10^{12}	Float LU
Audi	9.44×10^5	3.93×10^7	31.28	5.23×10^{12}	Float LL^T
10M	1.04×10^7	8.91×10^7	75.66	1.72×10^{14}	Complex LDL^T

Machines

Machine	Processors	Frequency	GPUs	RAM
Romulus	AMD Opteron 6180 SE (4×12)	2.50 GHz	Tesla T20 ($\times 2$)	256 GiB
Mirage	Westmere Intel Xeon X5650 (2×6)	2.67 GHz	Tesla M2070 ($\times 3$)	36 GiB
Riri	Intel Xeon E7- 4870 (4×10)	2.40 GHz	None	1 TB

CPU only results on Audi

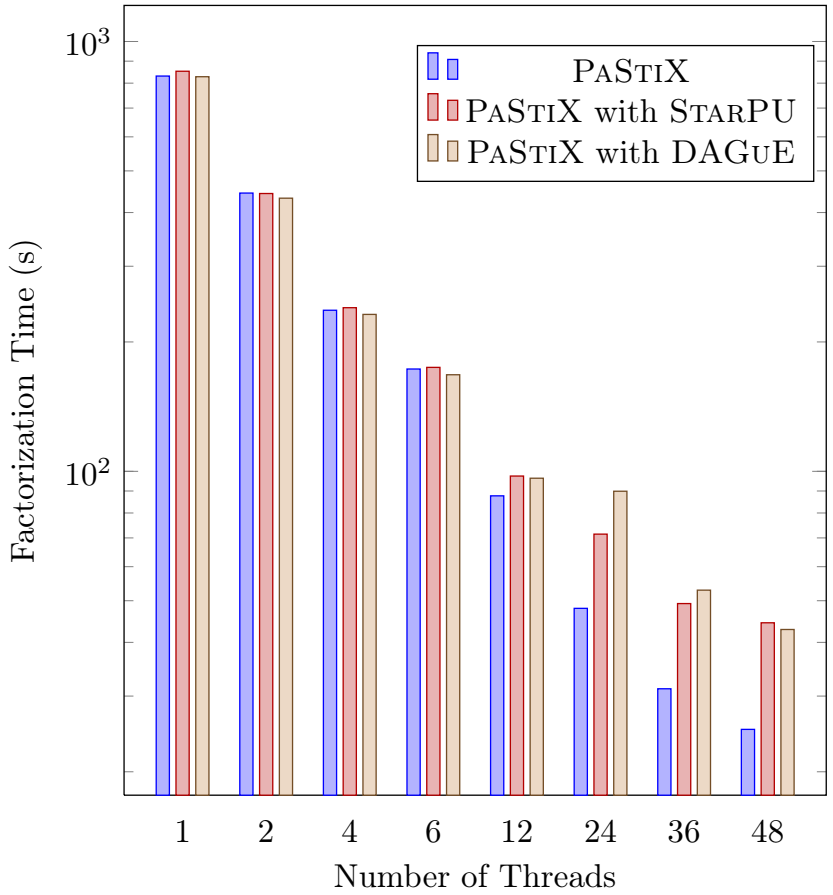


Figure: LL^T decomposition on Audi (double precision)

CPU only results on MHD

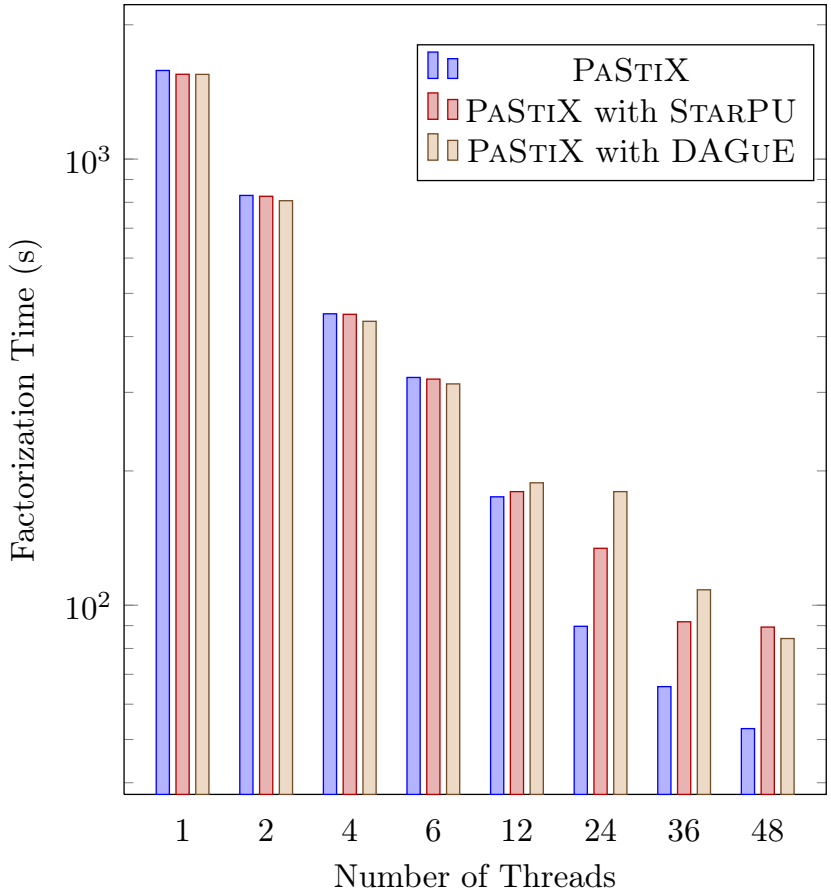


Figure: *LU* decomposition on MHD (double precision)

CPU only results on 10 Millions

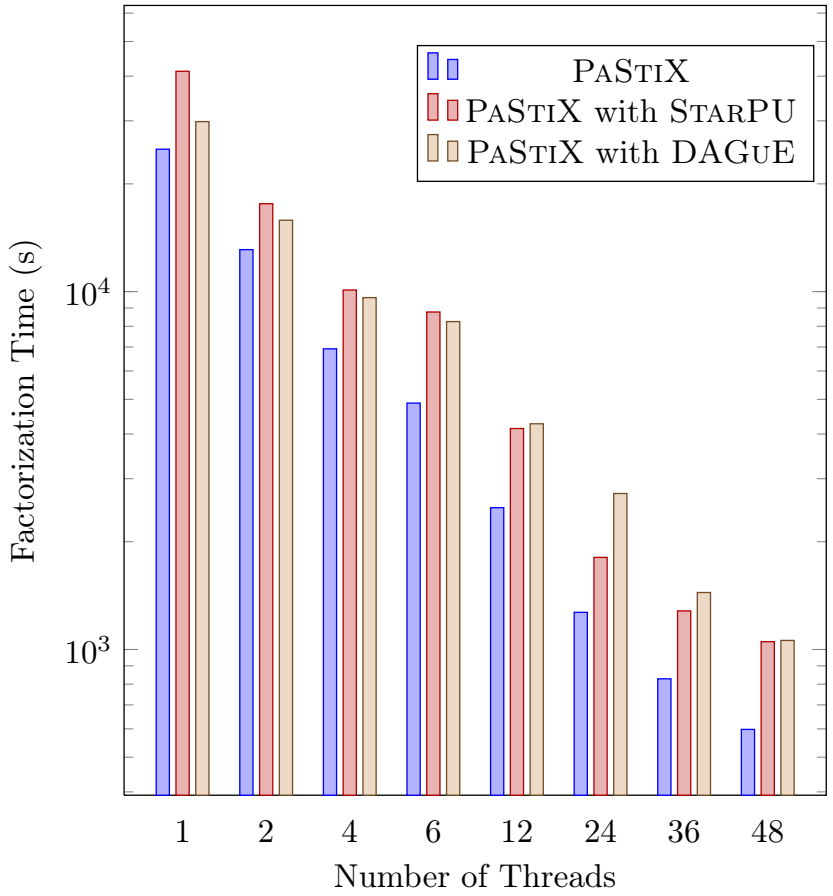


Figure: LDL^T decomposition on 10M (double complex)

GPU study on mirage

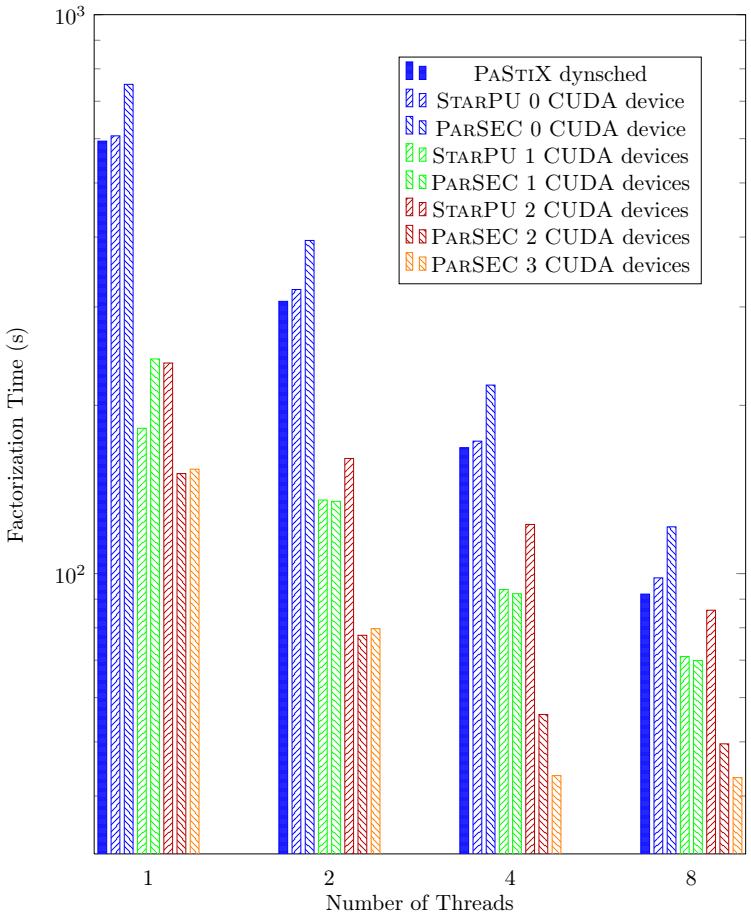


Figure: LL^T decomposition on Audi (double precision)

5
Improvement on granularity

Improvements on granularity

- Graph preprocessing minimal blocking $\rightarrow$ reduce number of tasks;
- Smarter panel splitting to suppress low flop tasks.

Panel splitting

Why splitting panels ?

- create more parallelism.

Drawback

- induce facing block splitting that can create many tiny blocks.

Solution

- smarter panel splitting;
- avoid tiny blocks creation which leads to inneficient BLAS.

A smarter split

- For each panel :
  - Construct a partition of the panel height with the number of facing blocks;
  - Decide to split where the number of splitted blocks is minimal.

A smarter split

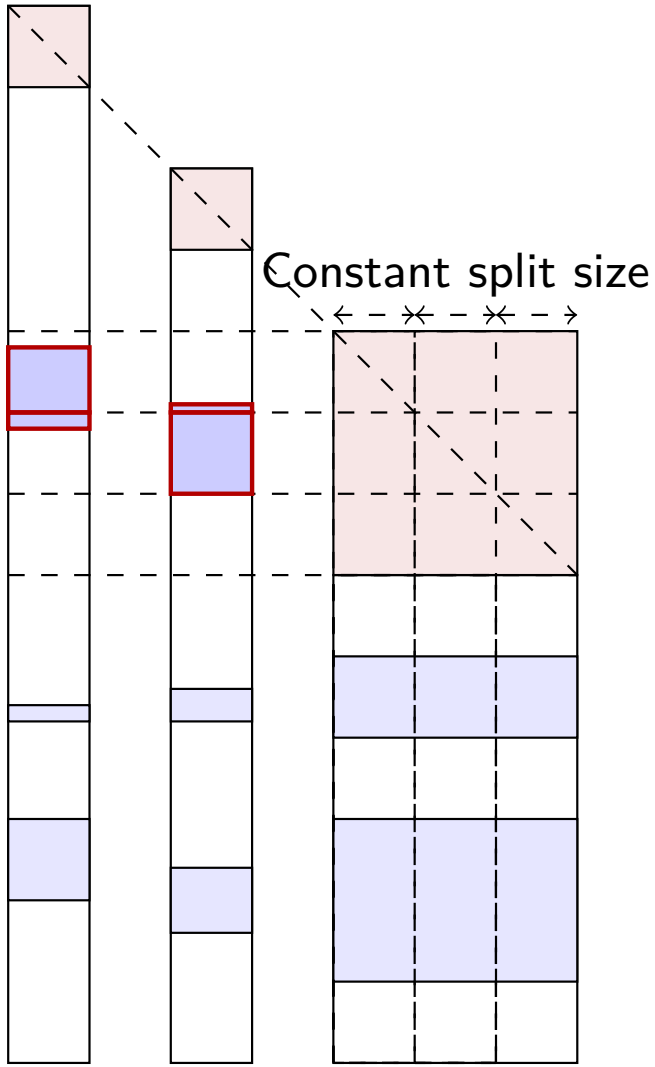
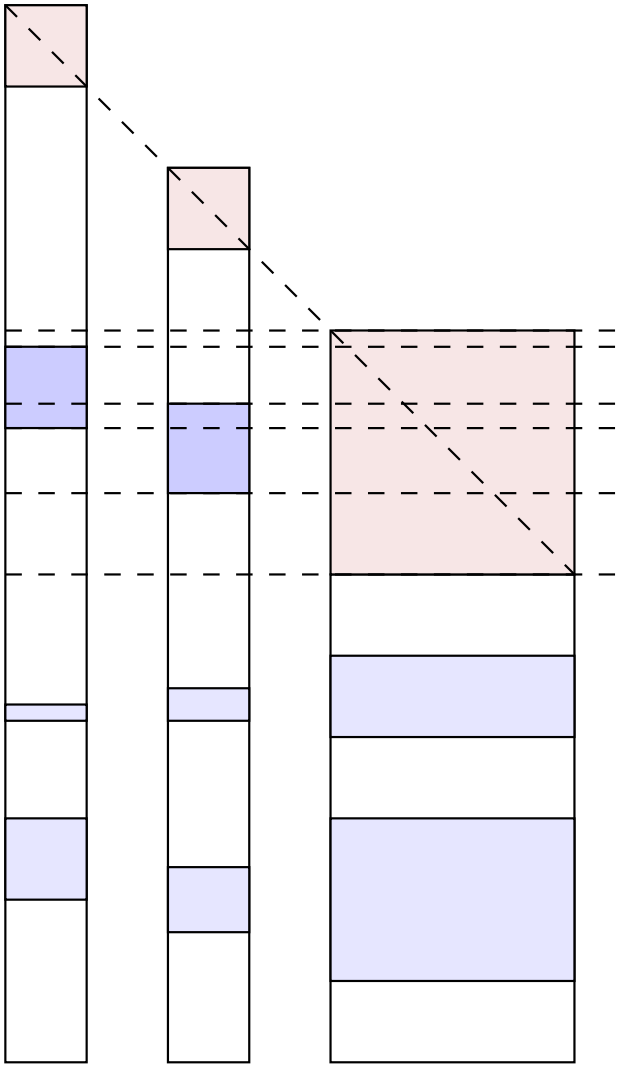


Figure: Classical equal splitting

A smarter split

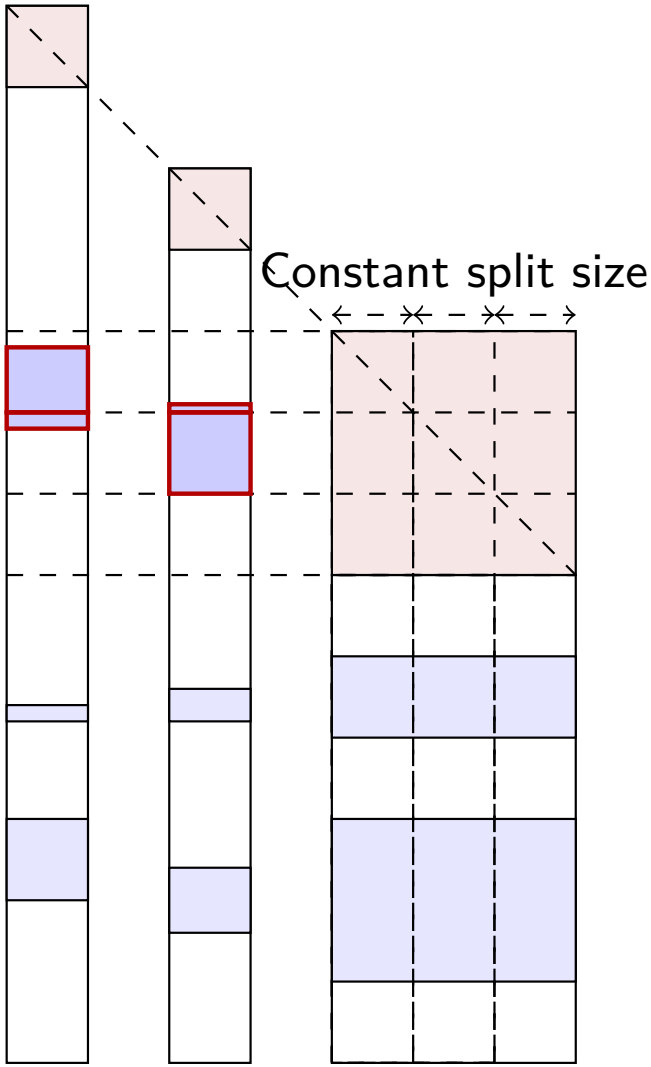


- 5 intervales partition :

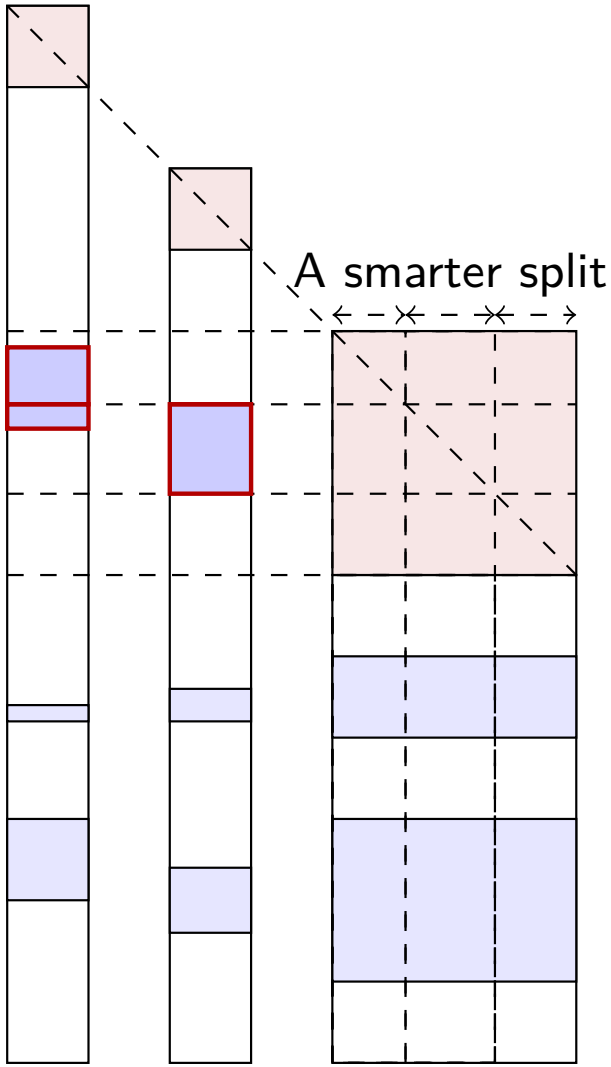
start	end	facing blocks
60	70	0
70	78	1
78	81	2
81	88	1
88	90	0

- better to split only on of the two facing blocks, on row 78 or 81.

A smarter split



(a) Classical equal splitting



(b) Smarter adapted splitting

6
Results about granularity

Preprocessing option comparaison on Audi, on Mirage

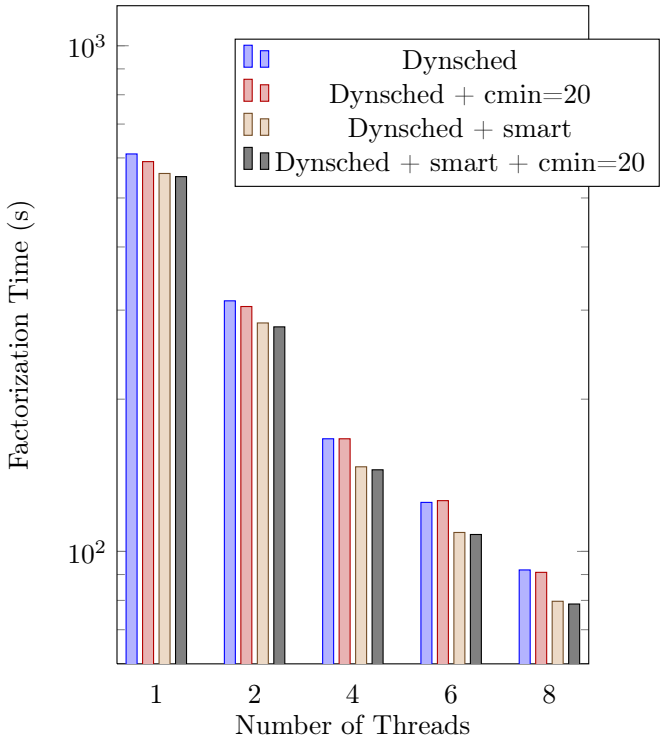


Figure: LL^T decomposition on Audi (double precision)

method	Dynsched		Dynsched + smart	
cmin	0	20	0	20
analyze time	1.95 s	0.35 s	2.56 s	0.42 s
number of panels	118814	10082	118220	9491
number of blocks	2283029	338493	2213497	280722
created by splitting	65147	48284	18072	13081
Avg. panel size	7.94262	93.602	7.98253	99.4305
Avg. block height	10.1546	29.2206	9.08452	24.5355
Memory usage	10.1 Go	10.7 Go	10.5 Go	11.1 Go

Smart panel splitting

- Factorization time reduction : 6-15%;
- Analyze time augmentation : 16-20%.

cmin 20

- Analyze time reduction : 80%;
- Less tasks may reduce runtime overhead, no effect on PaStiX fatorization time.

Study on SCOTCH minimal subblock parameter (cmin), on Riri

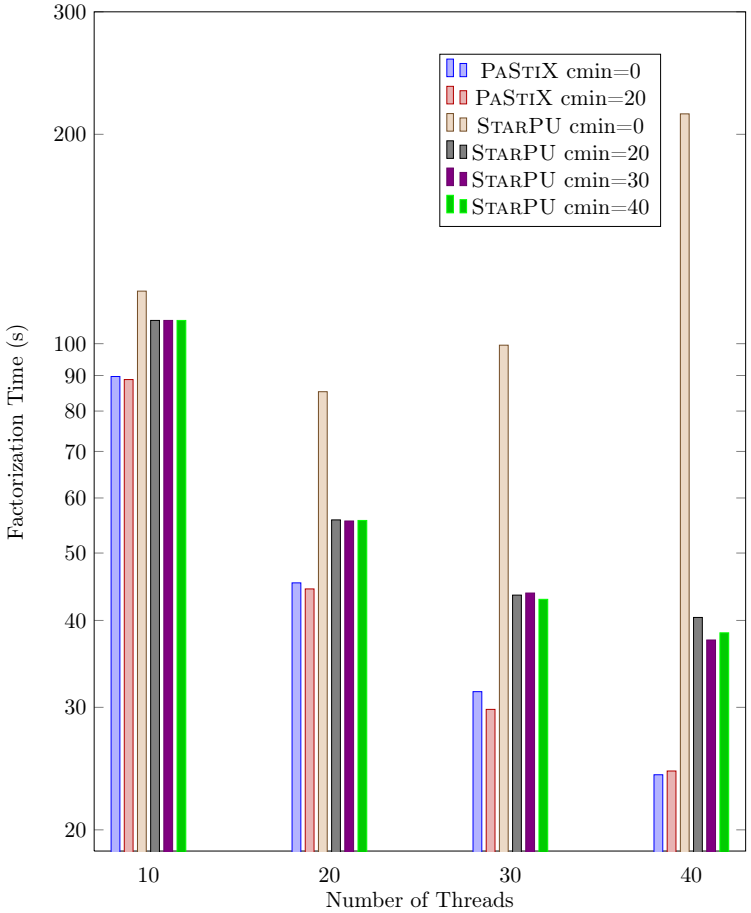


Figure: LL^T decomposition on Audi (double precision)

7
Conclusion and futur works

Conclusion

- Timing and scaling close to PaStiX;
- Speedup obtained with one (StarPU) or two (PaRSEC) GPUs and little number of cores;

Future works

- More locality :
  - StarPU : use contexts to attach tasks to a pool of processing units;
  - PaRSEC : Virtual processors : organize scheduling by socket;
- Streams : need streams to perform multiple kernel execution on a GPU at a time.
- Group tasks to reduce the runtime overhead (gather small tasks in PaStiX or let the runtime decide what is a small task);
- Distributed implementation (MPI), decide when to aggregate contribution or send FANIN or let the runtime decide.

Thanks !

Xavier LACOSTE
INRIA HiePACS team

Sparse Days - June 18, 2013