Exploiting Parameterized Task-graph in Sparse Direct Solvers

February 27, 2019 - SIAM CSE'19

Mathieu Faverge[a], Grégoire Pichon[a], Pierre Ramet[a]

[a]Inria, Bordeaux INP, CNRS, University of Bordeaux

Introduction

Problem: Solve Sparse $Ax = b$ with runtime systems

- Using PaSTiX solver
- Cholesky, LDL^t, or LU
- Exploit symmetric pattern of $(A + A^t)$ to generalize the solution

Objective

Describe how (to try) to get an efficient sparse direct factorization using parameterized task graph (and sequential task flow) on runtime systems.

Runtime systems supported by PaStiX

StarPU

- Inria Storm Team
- **Sequential Task Flow**
- Multiple kernels on the accelerators
- GPU multi-stream enabled
- Computes cost models on the fly
- Multiple scheduling strategies: Minimum Completion Time, Local Work Stealing, user defined...

PaRSEC

- ICL – University of Tennessee, Knoxville
- **Parameterized Task Graph**
- Multiple kernels on the accelerators
- GPU multi-stream enabled
- Scheduling strategy based on static performance model

Runtime systems supported by PaStiX

StarPU

- Inria Storm Team
- **Sequential Task Flow**
- Multiple kernels on the accelerators
- GPU multi-stream enabled
- Computes cost models on the fly
- Multiple scheduling strategies: Minimum Completion Time, Local Work Stealing, user defined...

PaRSEC

- ICL – University of Tennessee, Knoxville
- **Parameterized Task Graph**
- Multiple kernels on the accelerators
- GPU multi-stream enabled
- Scheduling strategy based on static performance model

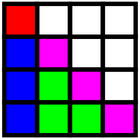
1

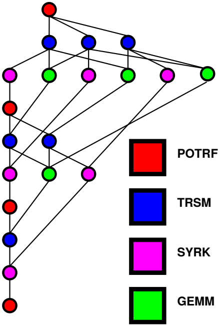
Quick introduction to both programming models

Sequential Task Flow

```
for (k = 0; k < N; k++) {
  POTRF( RW, A[k][k] );
  for (m = k+1; m < N; m++)
    TRSM( R, A[k][k], RW, A[m][k] );
  for (n = k+1; n < N; n++) {
    SYRK( R, A[n][k], RW, A[n][n] );
    for (m = n+1; m < N; m++)
      GEMM( R, A[m][k], R, A[n][k], RW, A[m][n] );
  }
}
__wait__();
```

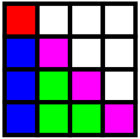
- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependencies...
- ... and builds a graph of tasks
- The graph of tasks is executed

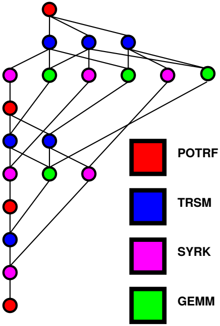




PTG: Example of the TRSM in Cholesky Decomposition

```
potrf_trsm(k, m)
```

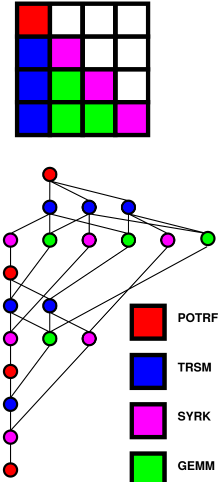




- Tasks dependencies are self-described
- Data references are annotated
- PaRSEC follows the dependencies from tasks to tasks ...
- ... and never builds the graph of tasks

PTG: Example of the TRSM in Cholesky Decomposition

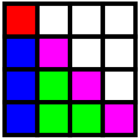
```
potrf_trsm(k, m)




// Parameters
READ  A <- A potrf_potrf(k)
RW    C <- (k == 0) ? dataA(m, k)
```

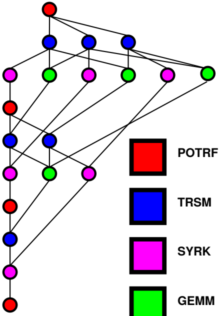


- Tasks dependencies are self-described
- Data references are annotated
- PaRSEC follows the dependencies from tasks to tasks ...
- ... and never builds the graph of tasks

PTG: Example of the TRSM in Cholesky Decomposition

```
potrf_trsm(k, m)



// Parameters
READ  A <- A potrf_potrf(k)
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C potrf_gemm(k-1, m, k)
        -> A potrf_syrk(k, m)
        -> A potrf_gemm(k, m, k+1..m-1)
        -> B potrf_gemm(k, m+1..descA.mt-1, m)
        -> dataA(m, k)
```

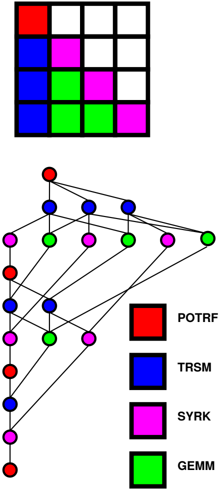




- Tasks dependencies are self-described
- Data references are annotated
- PaRSEC follows the dependencies from tasks to tasks ...
- ... and never builds the graph of tasks

PTG: Example of the TRSM in Cholesky Decomposition

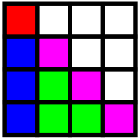
```
potrf_trsm(k, m)

// Execution space
k = 0   .. MT-1
m = k+1 .. MT-1


// Parameters
READ  A <- A potrf_potrf(k)
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C potrf_gemm(k-1, m, k)
        -> A potrf_syrk(k, m)
        -> A potrf_gemm(k, m, k+1..m-1)
        -> B potrf_gemm(k, m+1..descA.mt-1, m)
        -> dataA(m, k)
```

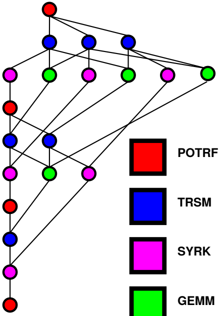


- Tasks dependencies are self-described
- Data references are annotated
- PaRSEC follows the dependencies from tasks to tasks ...
- ... and never builds the graph of tasks

PTG: Example of the TRSM in Cholesky Decomposition

```
potrf_trsm(k, m)

// Execution space
k = 0   .. MT-1
m = k+1 .. MT-1

// Task Mapping
: dataA(m, k)

// Parameters
READ  A <- A potrf_potrf(k)
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C potrf_gemm(k-1, m, k)
        -> A potrf_syrk(k, m)
        -> A potrf_gemm(k, m, k+1..m-1)
        -> B potrf_gemm(k, m+1..descA.mt-1, m)
        -> dataA(m, k)
```

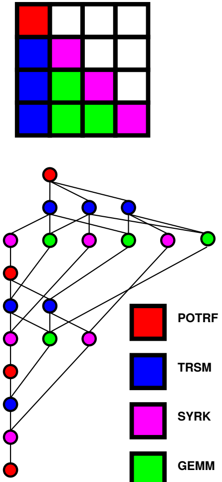




- Tasks dependencies are self-described
- Data references are annotated
- PaRSEC follows the dependencies from tasks to tasks ...
- ... and never builds the graph of tasks

PTG: Example of the TRSM in Cholesky Decomposition

```
potrf_trsm(k, m)

// Execution space
k = 0   .. MT-1
m = k+1 .. MT-1

// Task Mapping
: dataA(m, k)

// Parameters
READ  A <- A potrf_potrf(k)
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C potrf_gemm(k-1, m, k)
        -> A potrf_syrk(k, m)
        -> A potrf_gemm(k, m, k+1..m-1)
        -> B potrf_gemm(k, m+1..descA.mt-1, m)
        -> dataA(m, k)

BODY
  trsm(A, C);
END
```



- Tasks dependencies are self-described
- Data references are annotated
- PaRSEC follows the dependencies from tasks to tasks ...
- ... and never builds the graph of tasks

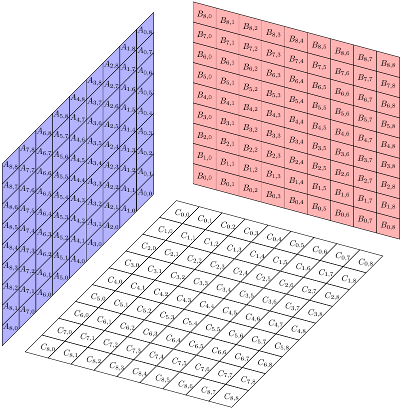
2

How to describe a sparse solver with PTG?

Sparse solver with PTG

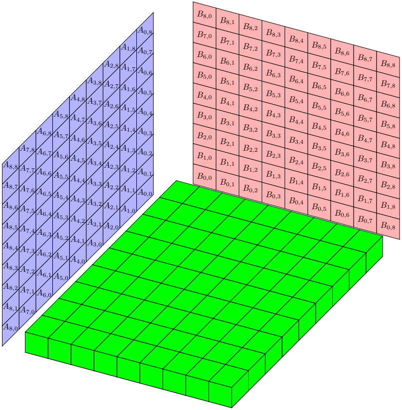
1. How to describe the domain space of each type of task?
2. How to describe in a linear space the dependencies between the tasks?

3D Visualization of the domain space (Dense GEMM)



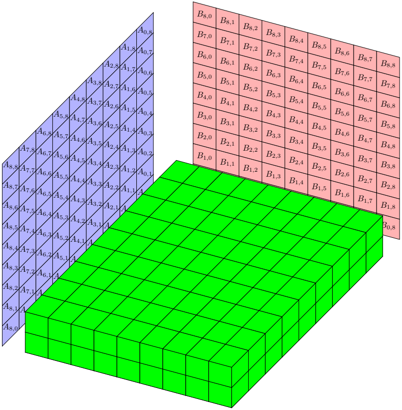
- Pretty simple to construct data dependencies
- Stack of GEMM tasks on each tile of C
- Broadcast of A, and B, tiles to rows, and columns, of C

3D Visualization of the domain space (Dense GEMM)



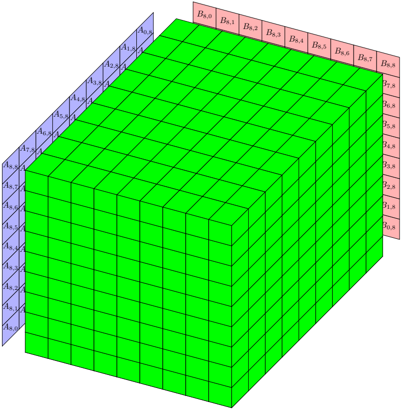
- Pretty simple to construct data dependencies
- Stack of GEMM tasks on each tile of C
- Broadcast of A, and B, tiles to rows, and columns, of C

3D Visualization of the domain space (Dense GEMM)



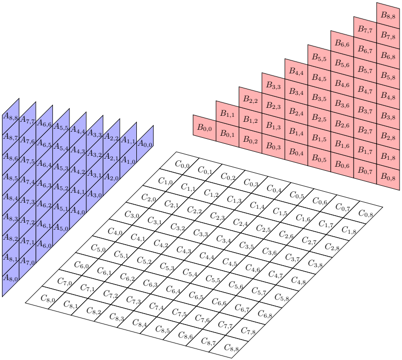
- Pretty simple to construct data dependencies
- Stack of GEMM tasks on each tile of C
- Broadcast of A, and B, tiles to rows, and columns, of C

3D Visualization of the domain space (Dense GEMM)



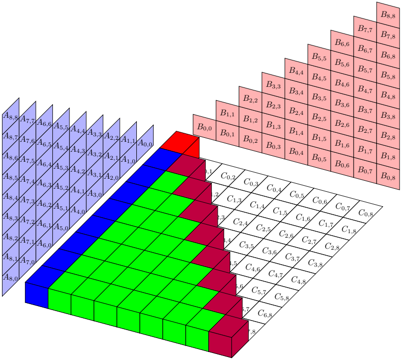
- Pretty simple to construct data dependencies
- Stack of GEMM tasks on each tile of C
- Broadcast of A, and B, tiles to rows, and columns, of C

3D Visualization of the domain space (Dense Cholesky)



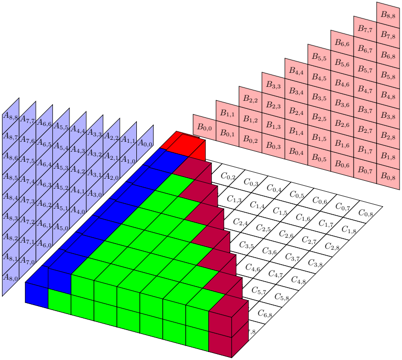
- A little more complicated than GEMM, but still very simple
- Stack of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to rows, and columns for update.

3D Visualization of the domain space (Dense Cholesky)



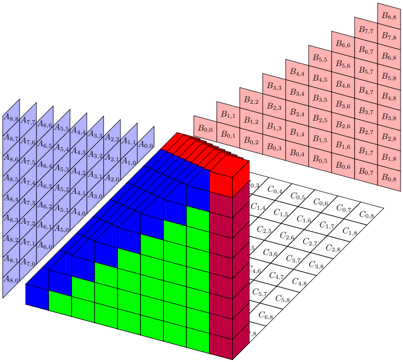
- A little more complicated than GEMM, but still very simple
- Stack of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to rows, and columns for update.

3D Visualization of the domain space (Dense Cholesky)



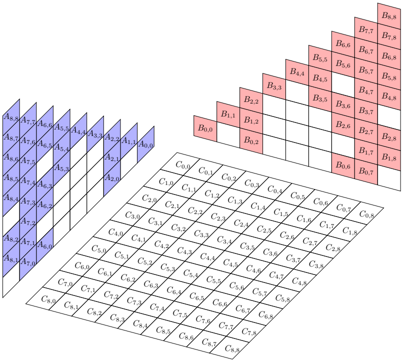
- A little more complicated than GEMM, but still very simple
- Stack of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to rows, and columns for update.

3D Visualization of the domain space (Dense Cholesky)



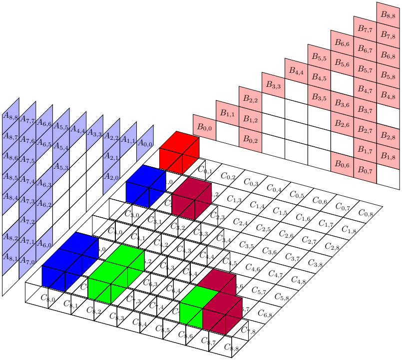
- A little more complicated than GEMM, but still very simple
- Stack of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to rows, and columns for update.

3D Visualization of the domain space (Sparse Cholesky)



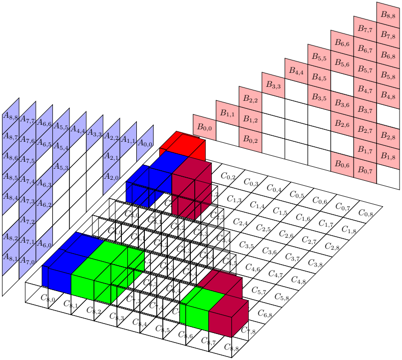
- Do not respect the continuous linear space required by the PTG model
- Stack **with** *holes* of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to **portion of** rows, and columns for update.

3D Visualization of the domain space (Sparse Cholesky)



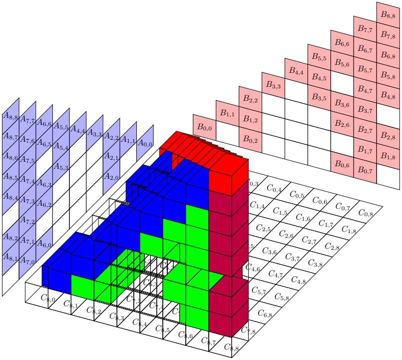
- Do not respect the continuous linear space required by the PTG model
- Stack **with** *holes* of GEMM, or SYRK, tasks on each tile of C ended respectively by a TRSM, or POTRF task.
- Broadcast of A and A^t tiles to **portion of** rows, and columns for update.

3D Visualization of the domain space (Sparse Cholesky)



- Do not respect the continuous linear space required by the PTG model
- Stack **with** *holes* of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to **portion of** rows, and columns for update.

3D Visualization of the domain space (Sparse Cholesky)

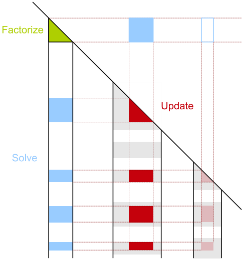


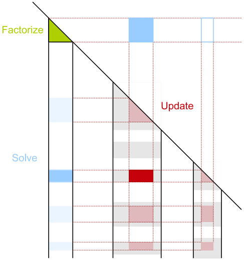
- Do not respect the continuous linear space required by the PTG model
- Stack **with** *holes* of `GEMM`, or `SYRK`, tasks on each tile of C ended respectively by a `TRSM`, or `POTRF` task.
- Broadcast of A and A^t tiles to **portion of** rows, and columns for update.

Summary

Issues

- We can not represent the tasks as in dense computation due to too many holes.
  → Need for a more compact representation of the domain space
- Task granularity is smaller than in dense
  → Need to rethink what is an elementary task





Exploit the CSR/CSC formats at the block level (1D tasks)

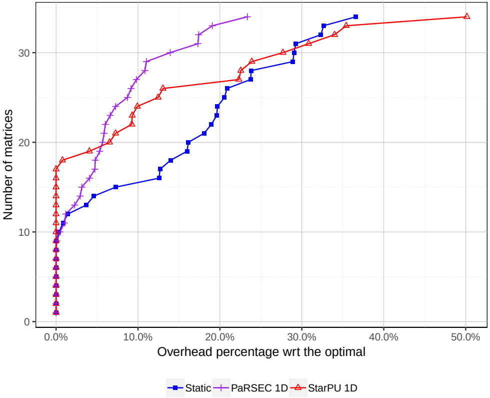
	0	1	2	3	4	5	6	7	8
0	X	-	X	-	-	-	-	-	X
1	-	X	-	-	-	X	X	-	-
2	X	-	X	-	-	-	-	-	X
3	-	-	-	X	-	X	-	-	X
4	-	-	-	-	X	-	X	-	X
5	-	X	-	X	-	X	X	-	X
6	-	X	-	-	X	X	X	X	X
7	-	-	-	-	-	-	X	X	-
8	X	-	X	X	X	X	X	X	X

Adjacency of the unknown 5:

$$\ldots \ \Big| \underbrace{\ 1\ \Big|\ 3\ }_{incoming} \Big|\ 5\ \Big| \underbrace{\ 6\ \Big|\ 8\ }_{outgoing} \Big| \ \ldots$$

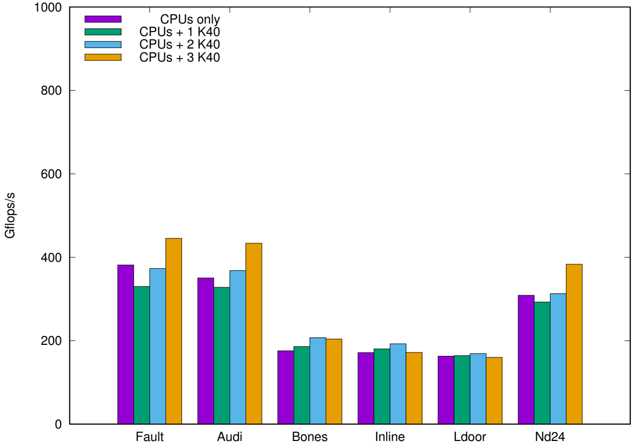
- One GEMM task per off-diagonal block
- The CSR of lower$(A + A^t)$ matches exactly the incoming edges of the POTRF/TRSM task.
- The CSC of lower$(A + A^t)$ matches exactly the outgoing edges of the POTRF/TRSM task.

Comparison on Shared memory architecture (24 CPUs)



- 34 matrices from SuiteSparse Collection
- Cost: from 1.38 to 318 Gflops
- Performance: from 133 to 558 Gflop/s
- Better average performance for PaRSEC
- More cases to the advantage of StarPU
- Both competitive with the internal static scheduling

Performance on Kepler architecture (24 CPUs + 3 K40)



3

How to express more complex parallelism

Exploit runtime systems to express more complex parallelism

Problem

- 1D parallelism is usually not enough to feed all resources
- Granularity is:
  - good for the bottom of the elimination tree
  - too large for the top levels of the elimination tree

Proposition

- Keep the 1D tasks for the smaller supernodes of the bottom of the elimination tree
- Refine to a smaller granularity and exploit 2D parallelism at the top of the tree
- Generate more parallelism with adequate granularity
- Should provide a better load balance for future distributed implementation

Hybrid 1D/2D with STF (STARPU)

```
for (k = 0; k < N; k++) {
  if ( 1D( A[k] ) ) {
    POTRF( RW, A[k] );
    TRSM( RW, A[k] );
    for (m = k+1; m < N; m++)
      if ( !empty(A[m][k]) ) {
         GEMM( R, A[k], RW, A[m] );
      }
    }
  }
  partition( RW, A[k] ); /* A[*][k] are now available */
}
for (k = 0; k < N; k++) {
  if ( 2D( A[k] ) ) {
    POTRF( RW, A[k][k] );
    for (m = k+1; m < N; m++)
      TRSM( R, A[k][k], RW, A[m][k] );
    for (m = k+1; m < N; m++)
      for (n = k+1; n <=m; n++)
         GEMM( R, A[m][k], R, A[n][k], RW, A[m][n] );
      }
    }
  }
}
__wait__();
```

- StarPU provides a partition tool to split the data during the task submission
- The system handles the dependencies between the large blocks (beginning of the algorithm) and the small sub-blocks (end).
- Data transfers are adapted to the selected granularity
- Partition is a task with no computation depending on the large data, and releasing all the small ones

Hybrid 1D/2D with PTG (PaRSEC)

- No automatic solution
- Exploit a task similarly to StarPU to change the dependencies
- Graph coherency is ensured through control dependencies

```
partition(k)

// Execution space
k = 0   .. MT-1
m = k+1 .. MT-1

// Task Mapping
: dataA(m, k)

// Parameters
READ  A <- data( m, k )
        -> C GEMM( m, k, 0 )

CTL  ctl <- ctl POTRF(k)
```

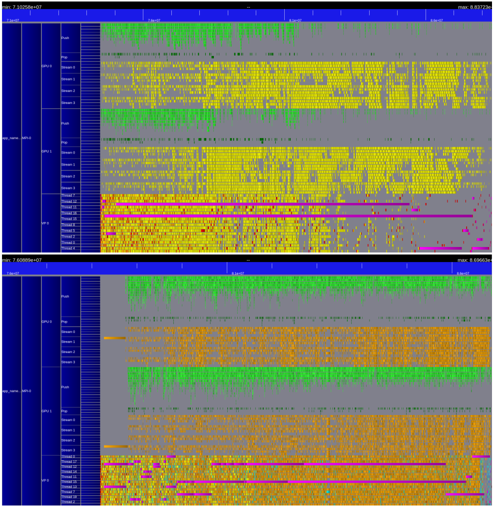
Hybrid 1D/2D with PTG (PaRSEC)

- No automatic solution
- Exploit a task similarly to StarPU to change the dependencies
- Graph coherency is ensured through control dependencies
- How to describe the new domain space of the GEMM?

```
partition(k)

// Execution space
k = 0   .. MT-1
m = k+1 .. MT-1

// Task Mapping
: dataA(m, k)

// Parameters
READ  A <- data( m, k )
        -> C GEMM( m, k, 0 )

CTL  ctl <- ctl POTRF(k)
```

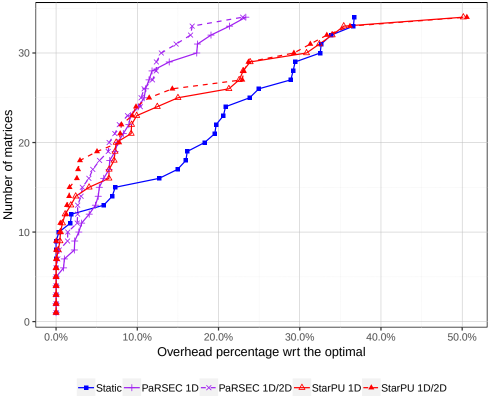
	0	1	2	3	4	5	6	7	8
0	X	-	-	-	-	-	-	-	-
1	-	X	-	-	-	-	-	-	-
2	X	-	X	-	-	-	-	-	-
3	-	-	-	X	-	-	-	-	-
4	-	-	-	-	X	-	-	-	-
5	-	X	-	X	-	X	-	-	-
6	-	X	-	0	X	X	X	-	-
7	-	-	-	-	-	-	X	X	-
8	X	0	X	X	X	X	X	X	X

Execution traces of 1D versus Hybrid 1D/2D



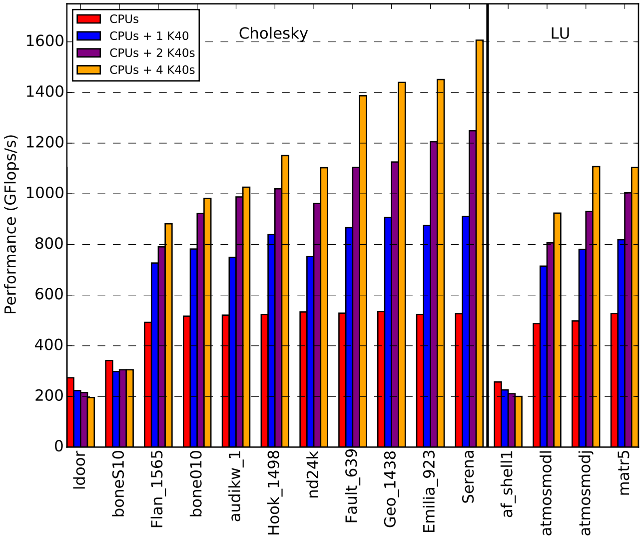
- `audi_kw1` matrix
- 24 cores + 2 Nvidia K40
- 1D on top
- Hybrid 1D/2D on bottom

Comparison on Shared memory architecture (24 CPUs)



- 34 matrices from SuiteSparse Collection
- Cost: from 1.38 to 318 GFlops
- Performance: from 133 to 582 GFLop/s
- Better average performance for PaRSEC
- More cases to the advantage of StarPU
- Hybrid is generally better than 1D with a small advantage

Performance on the Kepler architecture (24 CPUs + 3 K40)



4

Conclusion

Conclusion

- Both PTG and STF may be used in an *irregular* code
- Using the runtime systems allows to:
  - recover mistakes from the static prediction
  - test new strategies with some flexibility
- Brings the support of GPUs for free. . .
- Brings the support of distributed memory . . .
- Extremely useful in the case of irregular low-rank computations

Conclusion

- Both PTG and STF may be used in an *irregular* code
- Using the runtime systems allows to:
  - recover mistakes from the static prediction
  - test new strategies with some flexibility
- Brings the support of GPUs for free...
  The user may need to provide additional information to schedule the tasks
- Brings the support of distributed memory ...
- Extremely useful in the case of irregular low-rank computations

Conclusion

- Both PTG and STF may be used in an *irregular* code
- Using the runtime systems allows to:
  - recover mistakes from the static prediction
  - test new strategies with some flexibility
- Brings the support of GPUs for free...
  The user may need to provide additional information to schedule the tasks
- Brings the support of distributed memory ...
  The user needs to provide the data distribution
- Extremely useful in the case of irregular low-rank computations

Conclusion

- Both PTG and STF may be used in an *irregular* code
- Using the runtime systems allows to:
  - recover mistakes from the static prediction
  - test new strategies with some flexibility
- Brings the support of GPUs for free...
  The user may need to provide additional information to schedule the tasks
- Brings the support of distributed memory ...
  The user needs to provide the data distribution
- Extremely useful in the case of irregular low-rank computations

- New scheduling techniques for the GPUs in PaRSEC
- Exploit the flexiblity of the implementation to investigate various communication schemes in the distributed implementation
  Fan-in, Fan-out, Fan-both
- Dynamic re-scheduling for the low-rank solver

Thanks for your attention!

http://gitlab.inria.fr/solverstack/pastix