# Resilient algorithms in HPC and linear algebra for new architectures

Valentin Le Fèvre

Barcelona Supercomputing Center, Spain

January 27, 2022

# My CV

- Bachelor and Master degrees at ENS de Lyon
- PhD on *Resilient scheduling algorithms for large-scale platforms* at ENS de Lyon
- Post-doc on *Optimization of mathematical libraries* at Barcelona Supercomputing Center

# My CV

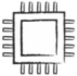
- Bachelor and Master degrees at ENS de Lyon
- PhD on *Resilient scheduling algorithms for large-scale platforms* at ENS de Lyon
  - Checkpointing techniques: checkpointing of a (linear) workflow, use of spare nodes to avoid restarts, . . .
  - Replication: which task should be replicated, efficient process replication strategy
- Post-doc on *Optimization of mathematical libraries* at Barcelona Supercomputing Center

# My CV

- Bachelor and Master degrees at ENS de Lyon
- PhD on *Resilient scheduling algorithms for large-scale platforms* at ENS de Lyon
  - Checkpointing techniques: checkpointing of a (linear) workflow, use of spare nodes to avoid restarts, ...
  - Replication: which task should be replicated, efficient process replication strategy
- Post-doc on *Optimization of mathematical libraries* at Barcelona Supercomputing Center
  - Project with Fujitsu for optimization on A64FX: **Cholesky**
  - EPI: based on Risc-V architecture, need new algorithms

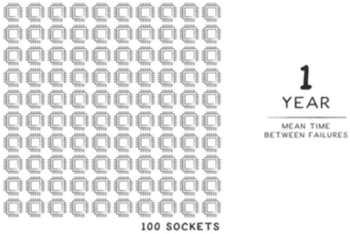
# Outline

# Introduction: scale is the enemy

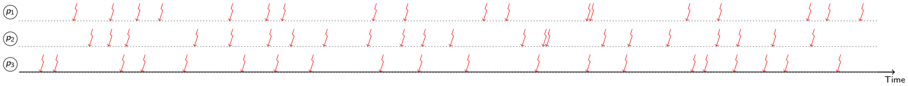


**100**
YEARS
MEAN TIME
BETWEEN FAILURES

# Introduction: scale is the enemy



# Introduction: scale is the enemy

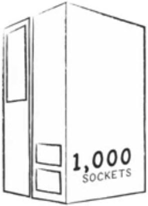



If three processors have around 20 faults during a time $t$ ($\mu = \frac{t}{20}$)...

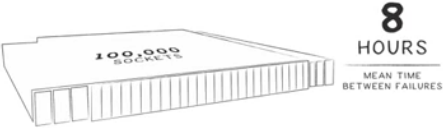

...during the same time, the platform has around 60 faults ($\mu_N = \frac{t}{60}$)

$$\mu_N = \frac{\mu}{N}$$

# Introduction: scale is the enemy




# Introduction: scale is the enemy



# Introduction: High Performance Computing

- High Performance Computing: gathering a large number of components to decrease execution time of applications
- Driving force: simulations/data-based in scientific research
- More components $\Rightarrow$ More failures

**Resilience [*The Top Ten Exascale Research Challenges*, ASCAC Subcommittee (2014)]**

Ensuring correct scientific computation in face of faults, reproducibility, and algorithm verification challenges.

# Two different types of faults

Fail-stop errors:

- Complete stop of the application
- Dead component, power failure, bug in the code, ...
- Easy to detect
- No correction possible as progress is lost

# Two different types of faults

Fail-stop errors:

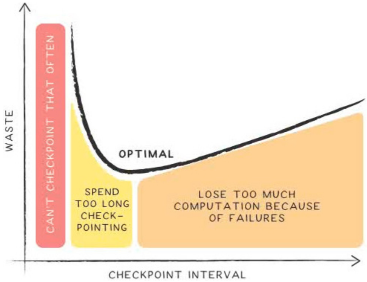
- Complete stop of the application
- Dead component, power failure, bug in the code, . . .
- Easy to detect
- No correction possible as progress is lost

Silent errors:

- Wrong results (bitflip for example)
- Cosmic radiations, faulty ALUs, . . .
- Unnoticed if a detection mechanism is not used
- Some detection mechanisms can be used to correct

# A first solution: checkpointing

- Regularly save the state of the application
- For silent errors: add a verification mechanism



# A first solution: checkpointing

Period $T$, minimize overhead $\mathbb{H}(T) = \frac{\mathbb{E}(T+C)}{T} - 1$

## Theorem - Young/Daly's formula

$$T_{opt} = \sqrt{\frac{2C}{\lambda_N}} = \sqrt{2C\mu_N} = \Theta(\lambda^{-\frac{1}{2}}) \quad (1)$$

$$\mathbb{H}_{opt} = \sqrt{2C\lambda_N} + o(\lambda^{\frac{1}{2}}) = \Theta(\lambda^{\frac{1}{2}}) \quad (2)$$

Recall that $\lambda_N = N\lambda = \frac{1}{\mu_N} = \frac{N}{\mu}$

# A second solution: replication

## Replication

Execute some (portion of) work several times to detect/correct errors.
Each execution is called a replica.

- For fail-stop errors
  - If one of the replicas works, we are done
  - More replicas $\Rightarrow$ More chance to succeed
- For silent errors
  - If two replicas have different outputs: an error is detected
  - With three replicas: majority rule used to correct

Replication decreases the rate of fatal failures but we need more resources.

# Outline

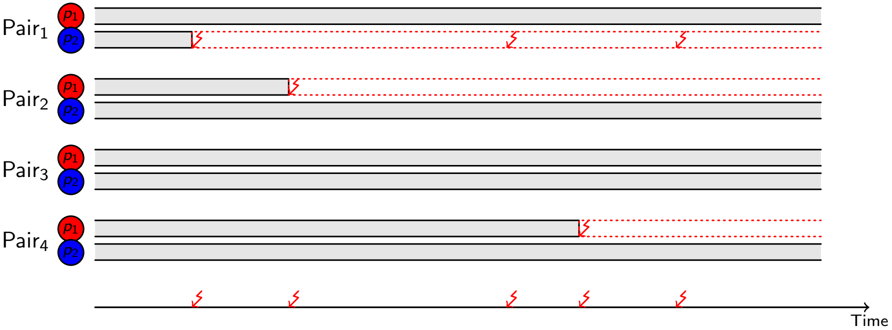
# Replication

- Full replication: efficiency $< 50\%$
- Can replication+checkpointing be more efficient than checkpointing alone?
- Study by Ferreira et al. [SC'2011]: yes
- Revisited by Hussain, Znati and Melhem [SC'2018]: yes

# Model by Ferreira et al. [SC' 2011]

- Platform with $N = 2b$ processors arranged into $b$ pairs
- Parallel application with $b$ processes, each replicated
- When a replica is hit by a failure, it is not restarted
- Application fails when both replicas in one pair have been hit

# Example



# Why Replication?

With $\mu = 5$ years, time to reach 90% chance of fatal failure:

| | |
|---|---|
| No replication | 24 minutes for $N = 100,000$ |
| No replication | 12 minutes for $N = 200,000$ |
| Replication | 85 hours for $N = 200,000$ ($b = 100,000$ pairs) |

# Checkpointing Period

- Replication combined with periodic checkpoint-restart à la Young/Daly
- Restart after interruption instead of after first failure
- Many failures needed to interrupt the application
  ⇒ checkpointing period much larger than without replication
- Optimal period?

# Mean Time To Interruption

- $N = 2b$, $b$ processor pairs
- $n_{\text{fail}}(2b)$ expected number of failures to interrupt the applications
- MTTI $M_N = M_{2b} =$ Mean Time to Interruption
  $\Rightarrow$ replaces MTBF from the application perspective

$$M_N = M_{2b} = n_{\text{fail}}(2b) \times \mu_{2b} = n_{\text{fail}}(2b) \times \frac{\mu}{2b} = \frac{n_{\text{fail}}(2b)}{2\lambda b} \qquad (3)$$

# Checkpointing

$$\text{No Replication} \quad T_{opt} = \sqrt{2\mu_N C} \tag{4}$$

$$\text{Full Replication} \quad T_{opt} = \sqrt{2M_N C} \tag{5}$$

# What's Wrong?

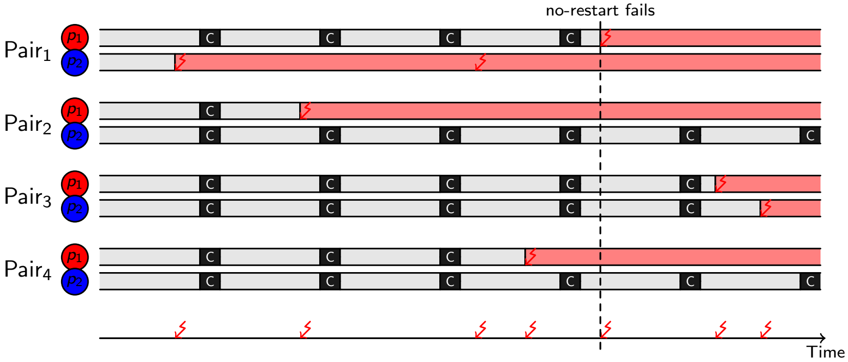
$$Topt = \sqrt{2M_N C}$$

- Just an approximation. How accurate?
- Risk is increasing as more and more processors die until application crash

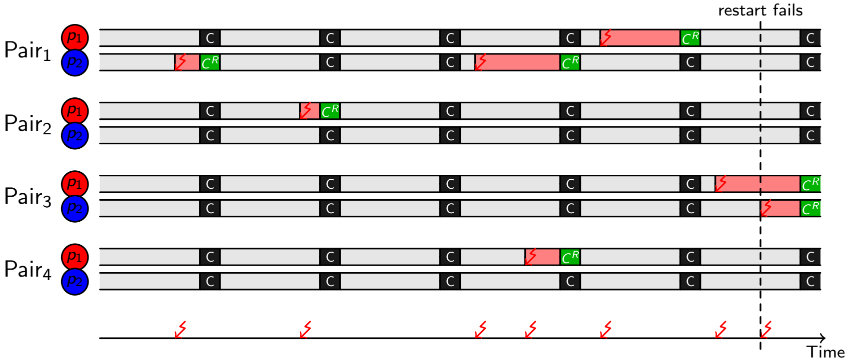
  $\Rightarrow$ Periodic checkpointing (most likely) not optimal ☹

# Outline

# *no-restart* vs. *restart*



# *no-restart* vs. *restart*



# restart

- Restart all failed processors (if any) after each checkpoint instead of only after interruption
- What is the additional cost?
- What is the optimal checkpointing period?

# Combined Checkpoint-Restart

**Cost of a checkpoint and restart wave** $C^R$

- one instance of each surviving process saves state (checkpoint)
- processes for missing replicas of the replicas allocated
- new processes load current (checkpointed) state and join system

**In-memory checkpoint replication**

- the buddy process and the replica are the same process
- surviving processes upload their checkpoint directly onto memory of newly spawned replicas

$\Rightarrow$ no exchange of checkpoints between pair of surviving buddies

**Worst case**: sequential approach, $C^R = 2C$
**Best case**: buddy checkpointing, negligible overhead, $C^R \approx C$

# Checkpointing Period

Periodic checkpointing is optimal for *restart*

$$T_{opt}^{\text{rs}} = \left(\frac{3C^R}{4b\lambda^2}\right)^{\frac{1}{3}} = \Theta(\lambda^{-\frac{2}{3}}). \tag{6}$$

$$\mathbb{H}^{\text{rs}}(T_{opt}^{\text{rs}}) = \left(\frac{3C^R\sqrt{b}\lambda}{\sqrt{2}}\right)^{\frac{2}{3}} + o(\lambda^{\frac{2}{3}}) = \Theta(\lambda^{\frac{2}{3}}) \tag{7}$$

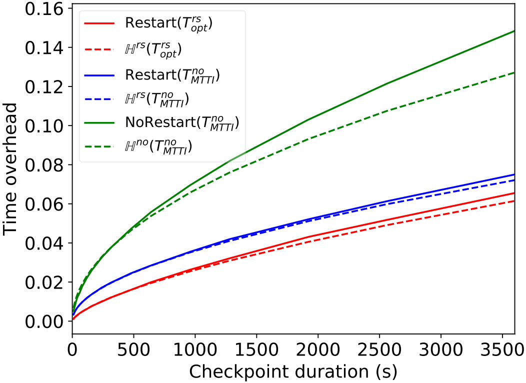
An order of magnitude longer!

# Outline

# Notations

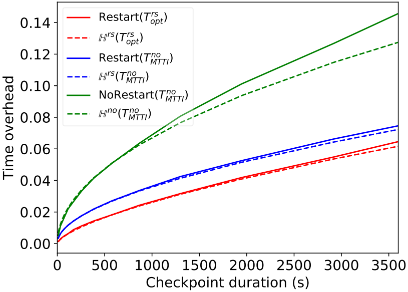
- *restart*
  $Restart(T)$ and overhead $\mathbb{H}^{\text{rs}}(T)$
  $T^{\text{rs}}_{opt}$ optimal period
- *no-restart*
  $NoRestart(T)$ and overhead $\mathbb{H}^{\text{no}}(T)$
  $T^{\text{no}}_{MTTI}$ used as 'optimal' period (analogy with Young/Daly)

# Model Accuracy

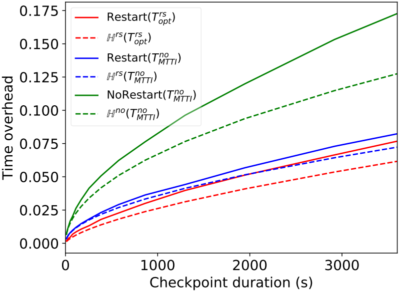



$\mu = 5$ years, $b = 10^5$ processor pairs, $C^R = C$.

# Model Accuracy With Trace Logs



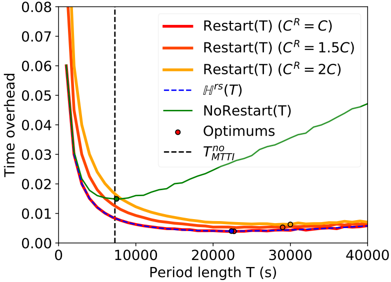

LANL#18



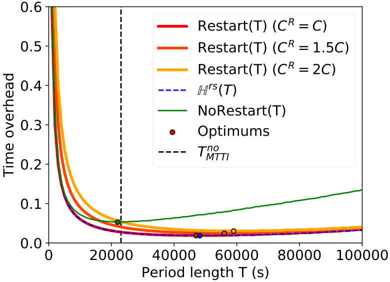

LANL#2

$\mu = 5$ years, $b = 10^5$ processor pairs, $C^R = C$.

# Impact of Checkpointing Period



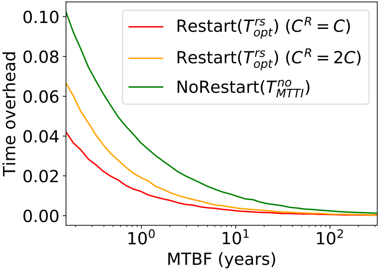

$C = 60$ seconds



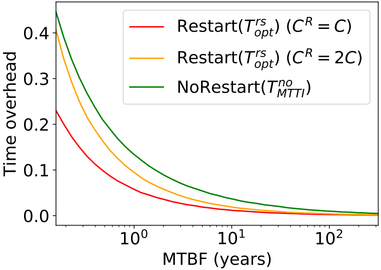

$C = 600$ seconds

$\mu = 5$ years, $b = 10^5$ processor pairs

# Impact of MTBF




$C = 60$ seconds

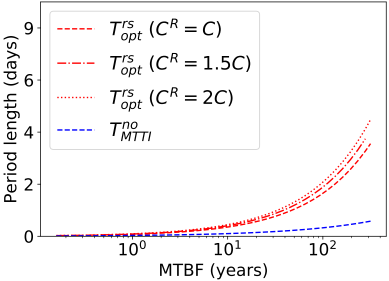



$C = 600$ seconds

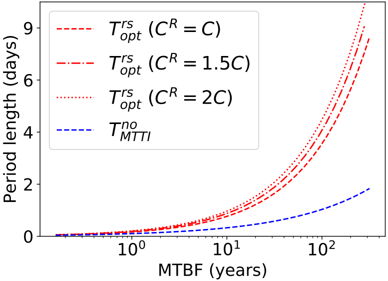
$b = 10^5$ processor pairs

# I/O Pressure

The I/O pressure decreases when the checkpointing period increases.




$C = 60$ seconds




$C = 600$ seconds

$b = 10^5$ processor pairs

# Time To Solution

**No replication, $N$ parallel processors**

$$T_{final} = (\mathbb{H}_{opt} + 1)\left(\gamma + \frac{1-\gamma}{N}\right) T_{seq}, \qquad \mathbb{H}_{opt} = \sqrt{\frac{2C}{\mu_N}}$$

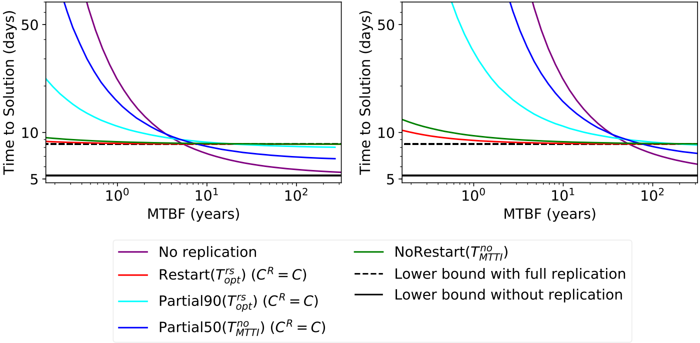
**Replication, $N = 2b$, $b$ replica pairs**

$$T_{final} = (\mathbb{H}_{opt} + 1)(1+\alpha)\left(\gamma + \frac{2(1-\gamma)}{N}\right) T_{seq}$$

$$\textit{no-restart} \quad \mathbb{H}_{opt} = \sqrt{\frac{2C}{M_N}}$$

$$\textit{restart} \quad \mathbb{H}_{opt} = \left(\frac{3C^R\sqrt{N}\lambda}{2\mu}\right)^{\frac{2}{3}}$$

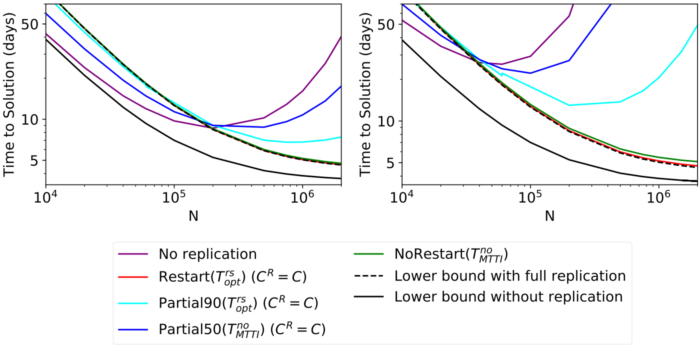
# Time To Solution




$C^R = C = 60$ seconds $\quad C^R = C = 600$ seconds

$$N = 200,000,\ \gamma = 10^{-5},\ \alpha = 0.2$$

# Replication Useful?




$C^R = C = 60$ seconds $\quad C^R = C = 600$ seconds

$\mu = 5\text{years}, \; \gamma = 10^{-5}, \; \alpha = 0.2$

# Conclusion

- Opinion is divided about replication
- Checkpoint/restart alone cannot ensure full reliability in heavily failure-prone environments
- When replication is needed (large $C$, short $\mu$, large $\gamma$), magic recipe:
  - use full replication
  - *restart* dead processors at each checkpoint (overlap if possible)
  - use $T_{opt}^{\text{rs}}$
- Not in this presentation: we can also minimize the energy consumption

# Outline

# Context: Cholesky Factorization

A: symmetric definite positive matrix

$$A = LL^T$$

$$Ax = b \Rightarrow LL^T x = b \Rightarrow x = (L^T)^{-1} L^{-1} b$$

Several solvers for sparse matrices: CHOLMOD, PaStiX, MUMPS, ...

# Context: Cholesky Factorization

A: symmetric definite positive matrix

$$A = LL^T$$

$$Ax = b \Rightarrow LL^T x = b \Rightarrow x = (L^T)^{-1} L^{-1} b$$

Several solvers for sparse matrices: **CHOLMOD**, PaStiX, MUMPS, ...
CHOLMOD is originally a sequential code.

# Structure of a solver

Usually 3 phases:

- Analyze
- Factorize
- Solve

# Structure of a solver

Usually 3 phases:

- Analyze $\Rightarrow$ perform some optimizations (reduce fill-in), build the elimination tree, aggregate nodes...
- Factorize
- Solve

# Structure of a solver

Usually 3 phases:

- Analyze $\Rightarrow$ perform some optimizations (reduce fill-in), build the elimination tree, aggregate nodes...
- Factorize $\Rightarrow$ main part of the solver, depends on the optimizations done in the previous phase.
- Solve

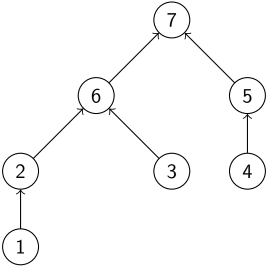
# Structure of a solver

Usually 3 phases:

- Analyze ⇒ perform some optimizations (reduce fill-in), build the elimination tree, aggregate nodes...
- Factorize ⇒ main part of the solver, depends on the optimizations done in the previous phase.
- Solve ⇒ simple: triangular solve.

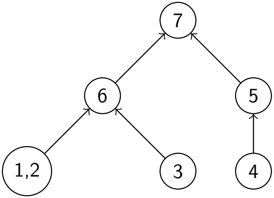
# Elimination tree

- Describes the structure of the sparse matrix
- One node = one column



# Elimination tree

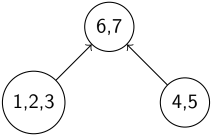
- Describes the structure of the sparse matrix
- One node = several contiguous columns




Node amalgamation: *supernodes*
Each supernode can be transformed with BLAS calls.

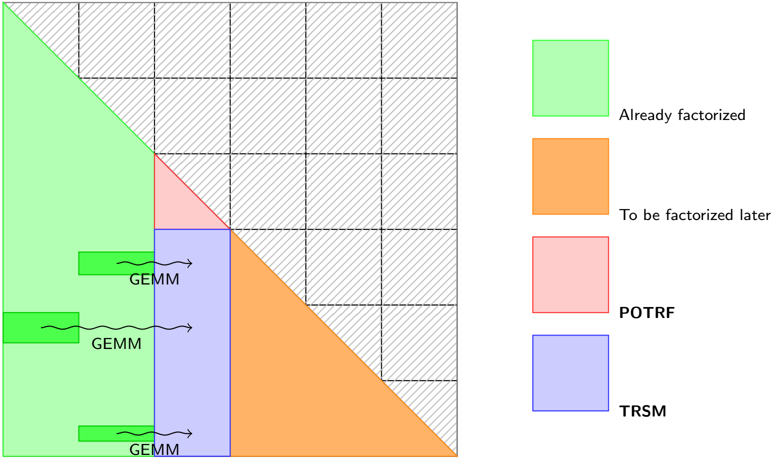
# Elimination tree

- Describes the structure of the sparse matrix
- One node = several contiguous columns




Node amalgamation: *supernodes*
Each supernode can be transformed with BLAS calls.

# BLAS kernels in Cholesky



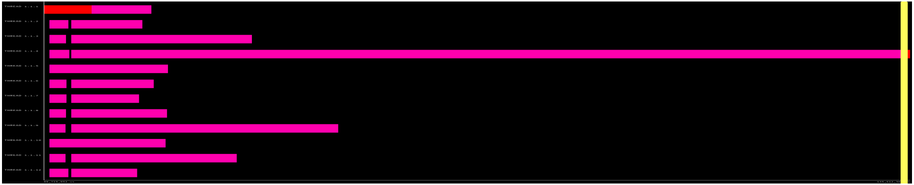
# Outline

# Task-based approach

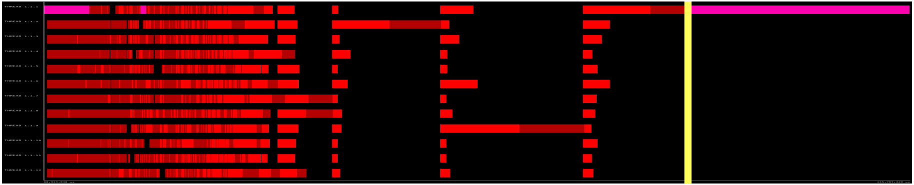
```
for (s = 0 ; s < nsuper ; s++)
{
   #pragma omp task in({*(dep_in[s][ii]), ii=0;num_in[s]}) out(*dep_out[s]) \
    default(none) shared(...) firstprivate(...) private(...) label(outer)
   {
      //Construction of the supernode
      for (idxS = STp [s] ; idxS < STp[s+1] ; idxS++) {
           d = STi[idxS] ;
           if (d==s) continue ;
           #pragma omp task default(none) shared(...) \
           private(...) firstprivate(...) private(...) label(inner)
           {
               //SYRK and GEMM
               omp_set_lock(&omp_lock);
               //Assembly of supernode
               omp_unset_lock(&omp_lock);
           }
      }
      #pragma omp taskwait
      //POTRF and TRSM
   }
}
```

# Selective Nesting

bone010



(a) Without nested tasks (NON-NESTED)

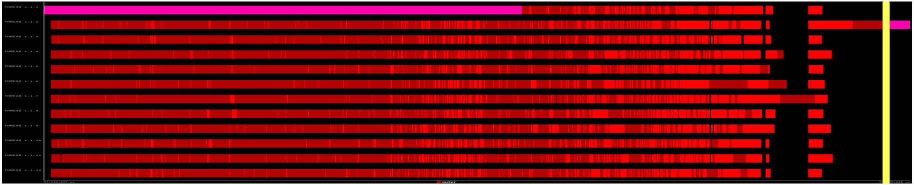


(b) With nested tasks (NESTED)

# Selective Nesting

inline_1

(c) Without nested tasks (NON-NESTED)



(d) With nested tasks (NESTED)

# Selective Nesting

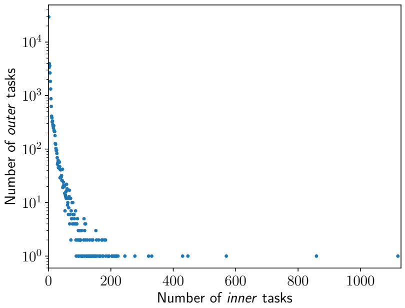
- **Trade-off between parallelism and overhead of task creation**
- If no task is nested: lack of parallelism, especially when reaching the root of the elimination tree

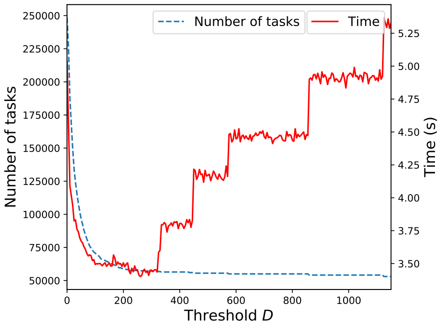
# Selective Nesting

- **Trade-off between parallelism and overhead of task creation**
- If no task is nested: lack of parallelism, especially when reaching the root of the elimination tree
- If every task is nested: too many tasks, overhead of task creation and destruction becomes a bottleneck

# Determining threshold $D$

bone010



# Determining threshold $D$

Analysis on a few matrices:

- Outer tasks with most *inner* tasks should be nested
- Optimal $D$ always less than 30% of max number of *inner* tasks
- Ratio between number of columns and number of tasks tend to be close (at optimal $D$)
- $\Rightarrow$ Algorithm OPT-D

# Determining threshold $D$

Analysis on a few matrices:

- Outer tasks with most *inner* tasks should be nested
- Optimal $D$ always less than 30% of max number of *inner* tasks
- Ratio between number of columns and number of tasks tend to be close (at optimal $D$)
- $\Rightarrow$ Algorithm Opt-D

If an *inner* task is small: keep it inside the *outer* loop $\Rightarrow$ Algorithm Opt-D-Cost

# Determining threshold $D$

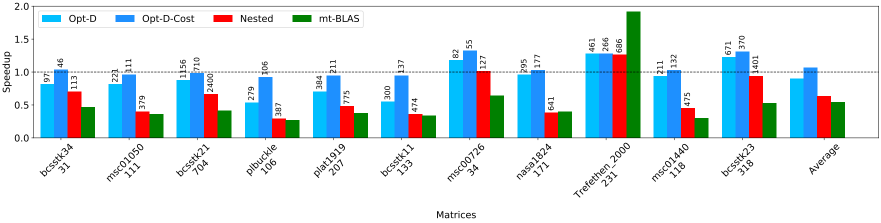
Analysis on a few matrices:

- Outer tasks with most *inner* tasks should be nested
- Optimal $D$ always less than 30% of max number of *inner* tasks
- Ratio between number of columns and number of tasks tend to be close (at optimal $D$)
- $\Rightarrow$ Algorithm Opt-D

If an *inner* task is small: keep it inside the *outer* loop $\Rightarrow$ Algorithm Opt-D-Cost
We will compare to mt-BLAS: sequential cholmod with multi-threaded BLAS.

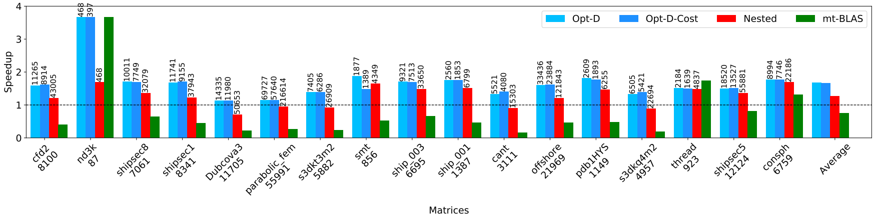
# Outline

# Results




Matrices with $10,000 < \#nnz < 50,000$

# Results




Matrices with $3,000,000 < \#nnz < 6,000,000$

# Conclusion

- Task-graph should depend on the data
- Finding a good granularity can drastically improve the performance
- Algorithm OPT-D-COST suitable for a large variety of matrices *without trying lots of configurations each time*
- mt-BLAS can be a good alternative for several matrices that can we can detect

# Conclusion

- Algorithm tuned from experiments on a given platform (here A64FX processors)
- For a few matrices in the evaluation, the performance is degraded

- Is there a way to find a good model, used to determine D afterwards? Hopefully, not platform-dependent
- What makes some matrices very different?
- Implementations of other algorithms (LU/QR) to try this strategy

# Outline

# European Processor Initiative

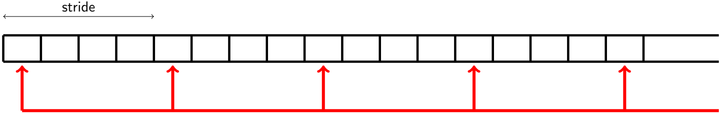
- Main goal: Develop European-based processors for ExaScale
- Rely on ARM architecture for main chips and Risc-V for accelerators

- Energy efficiency is one of the main challenges
- Risc-V initiative: to provide royalty-free ISA
- Needs to redesign algorithms using the vector extension

# Sparse Matrix Multiplication

Matrices are in Compressed Sparse Column format.

- SPA: SParse Accumulator
    - simple to implement
    - does not scale well $\Rightarrow$ one column by one
- ESC: Expand-Sort-Compress
    - More steps in the algorithm and use of sort
    - Most of the algorithm is fully parallelizable using *loop raking*



# BLIS

- BLIS is a portable BLAS library
- How to vectorize the kernels?
- Collaboration with University of Madrid
- BSC is responsible for evaluating the library using test chips