A Case for Malleable Thread-Level Linear Algebra Libraries: The LU Factorization with Partial Pivoting

Sandra Catalán, Jose R. Herrero, Enrique S. Quintana-Ortí,
Rafael Rodríguez-Sánchez, Robert van de Geijn

BLIS Retreat, 19-20th September 2016, Austin (Texas)

Motivation

Increase number
of threads

BLAS → TLP

Nested TLP + TP

LAPACK → TP (runtime)

Why malleability

Ta *3th*

.

.

Ta

Tb *5th*

.

.

.

.

.

Tb

Why malleability

Ta *3th* Tb *5th* Ta *3th* Tb *5th*

. . . .

. . . .

Ta . Ta .

. .

. Tb *8th*

Tb

DLA library modification to allow number of threads expansion

LU as an example

Algorithm: $[A] := \mathrm{LU\_BLK}(A)$

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$$

where A_{TL} is 0×0

while $n(A_{TL}) < n(A)$ do

Determine block size b

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

A1. $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} := \mathrm{LU\_UNB}\left(\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}\right)$

A2. $A_{12} := \mathrm{TRILU}(A_{11})^{-1} A_{12}$

A3. $A_{22} := A_{22} - A_{21} A_{12}$

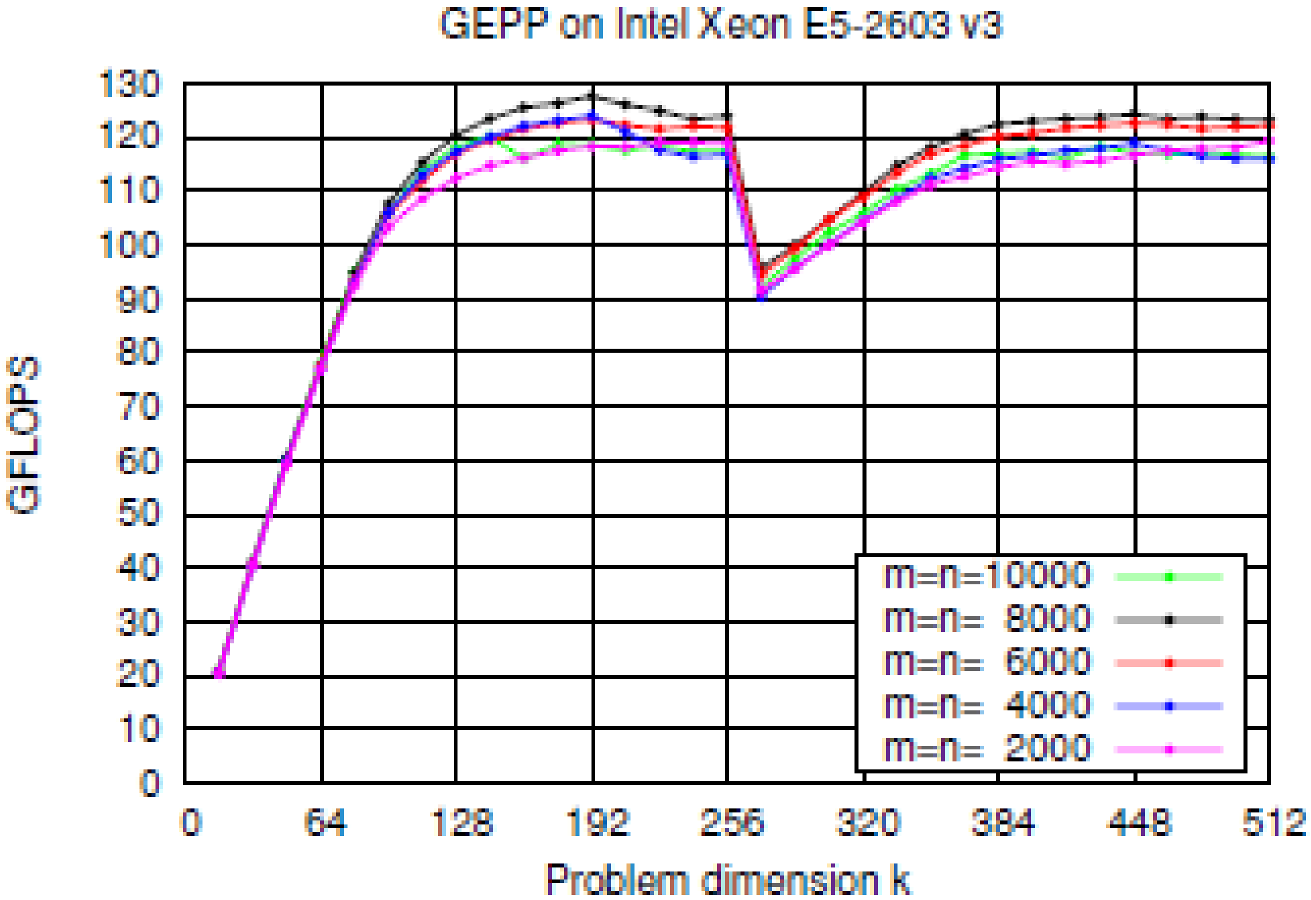
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

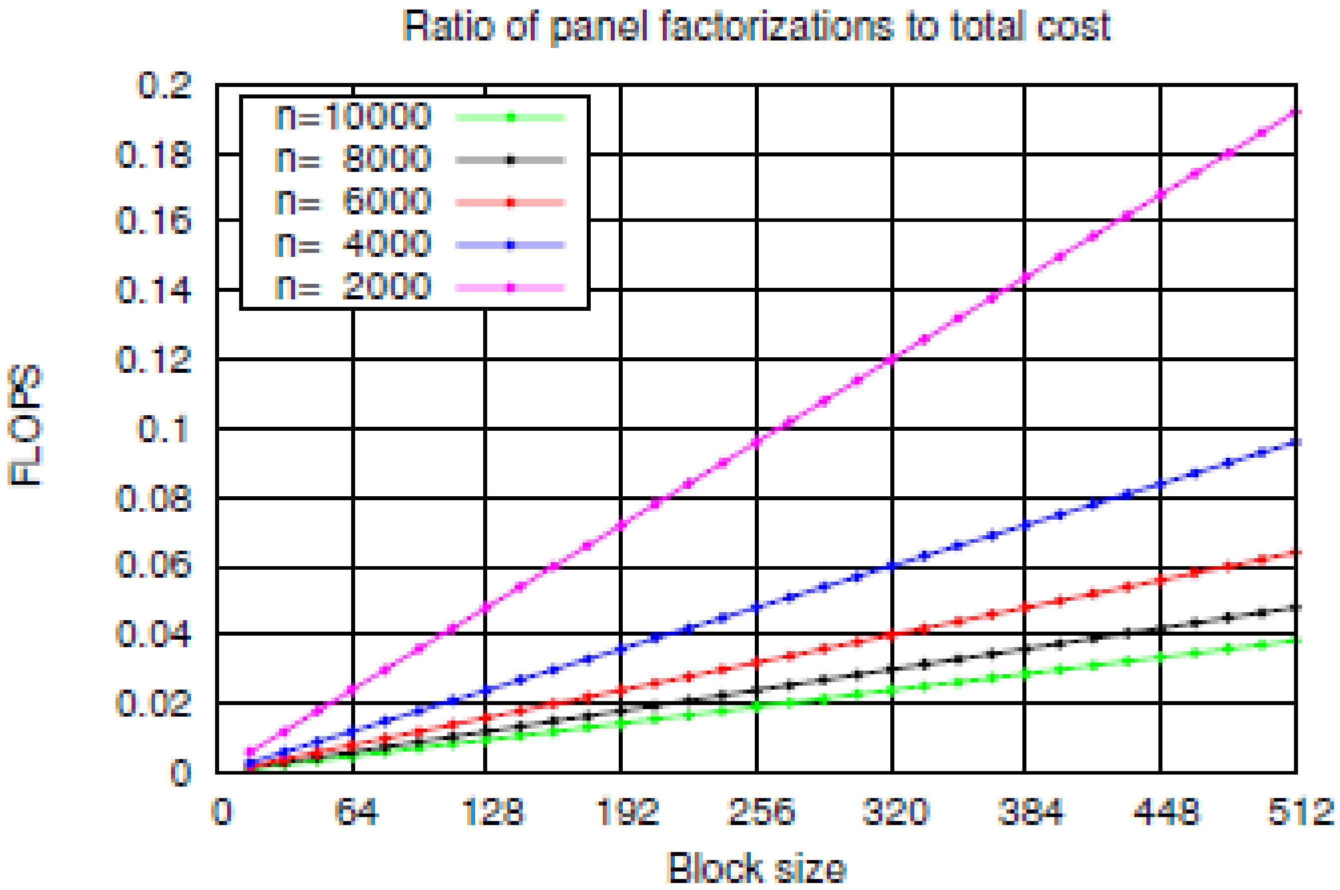
b size is important:

- Too small → Low GEMM performance
- Too large → Too many panel factorization flops

Optimal block size

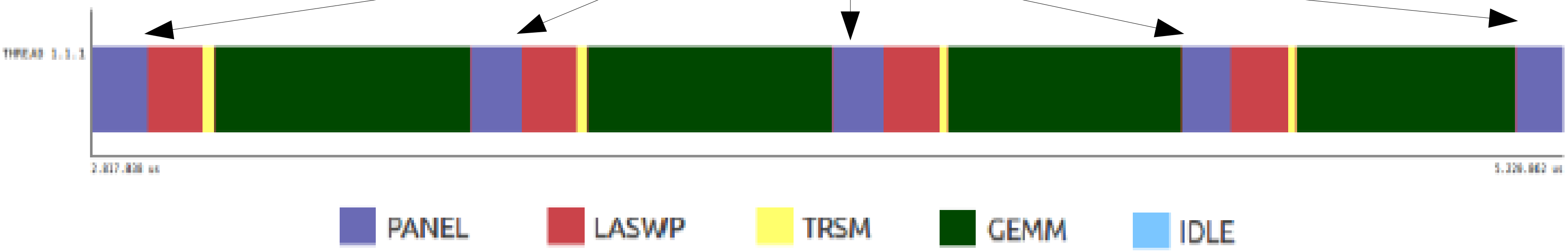


Optimal block size



The panel factorization relevance

Less than 2% of the flops
17.5% of the time



Execution trace of the first four iterations of the blocked RL LU factorization with partial pivoting, applied to a square matrix of order 10,000.

Dealing with the panel factorization

Algorithm: $[A] := \text{LU\_LA\_BLK}(A)$

Determine block size b

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), \; A_{BR} \rightarrow \left(A_{BR}^{P} \,\middle|\, A_{BR}^{R} \right)$$

where A_{TL} is 0×0, A_{BR}^{P} has b columns

$A_{BR}^{P} := \text{LU\_UNB}\left(A_{BR}^{P}\right)$

while $n(A_{TL}) < n(A)$ do

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

Determine block size b

% Partition into panel factorization and remainder

$$\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) \rightarrow \left(\begin{array}{c|c} A_{12}^{P} & A_{12}^{R} \\ \hline A_{22}^{P} & A_{22}^{R} \end{array}\right)$$

where both A_{12}^{P}, A_{22}^{P} have b columns

% Panel factorization, T_P

P1. $A_{12}^{P} := \text{TRILU}(A_{11})^{-1} A_{12}^{P}$

P2. $A_{22}^{P} := A_{22}^{P} - A_{21} A_{12}^{P}$

P3. $A_{22}^{P} := \text{LU\_UNB}\left(A_{22}^{P}\right)$

% Remainder, T_R

R1. $A_{12}^{R} := \text{TRILU}(A_{11})^{-1} A_{12}^{R}$

R2. $A_{22}^{R} := A_{22}^{R} - A_{21} A_{12}^{R}$

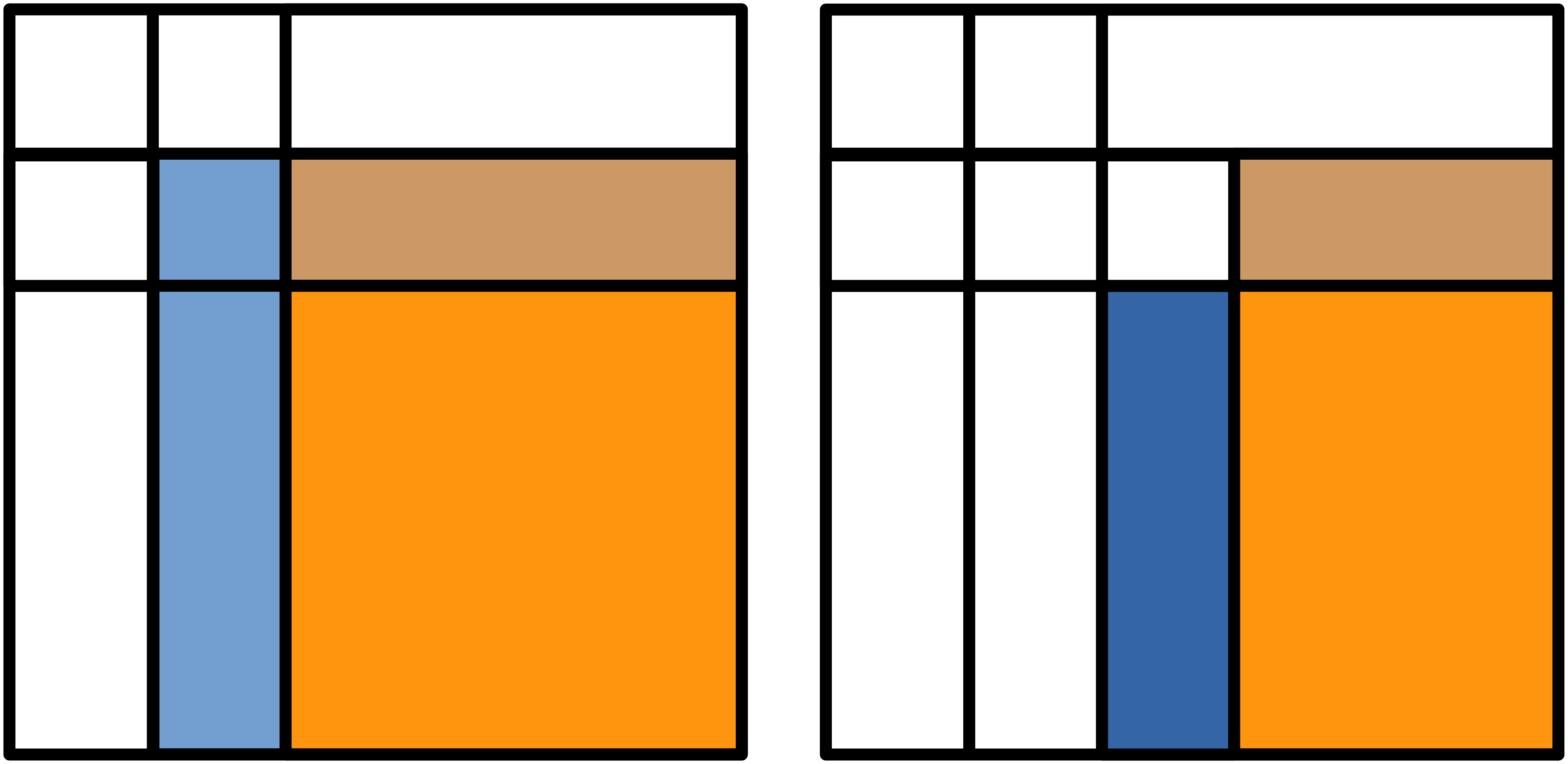
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

Look-ahead:

Overlap the factorization of the “next” panel with the update of the “current” trailing submatrix.

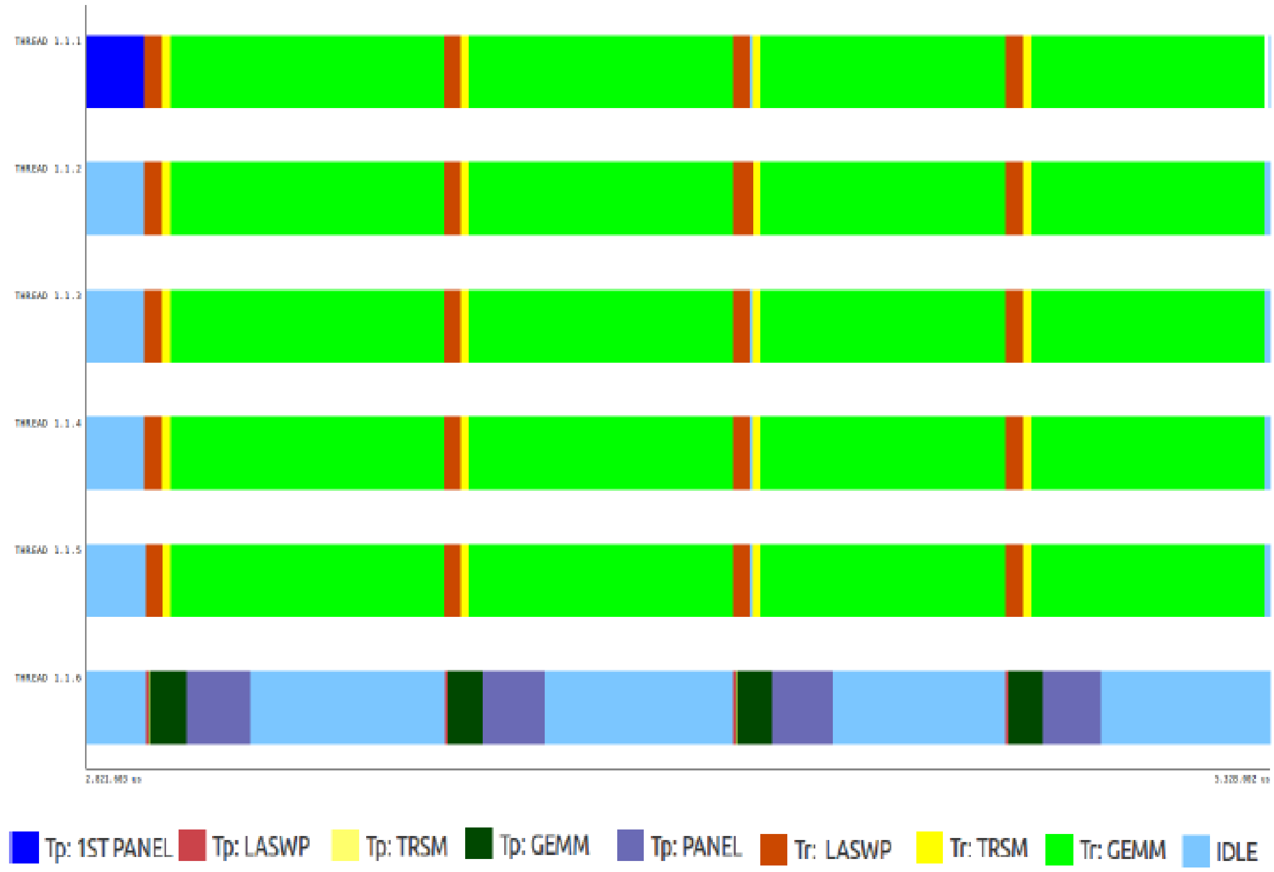
Look Ahead LU



Our setup

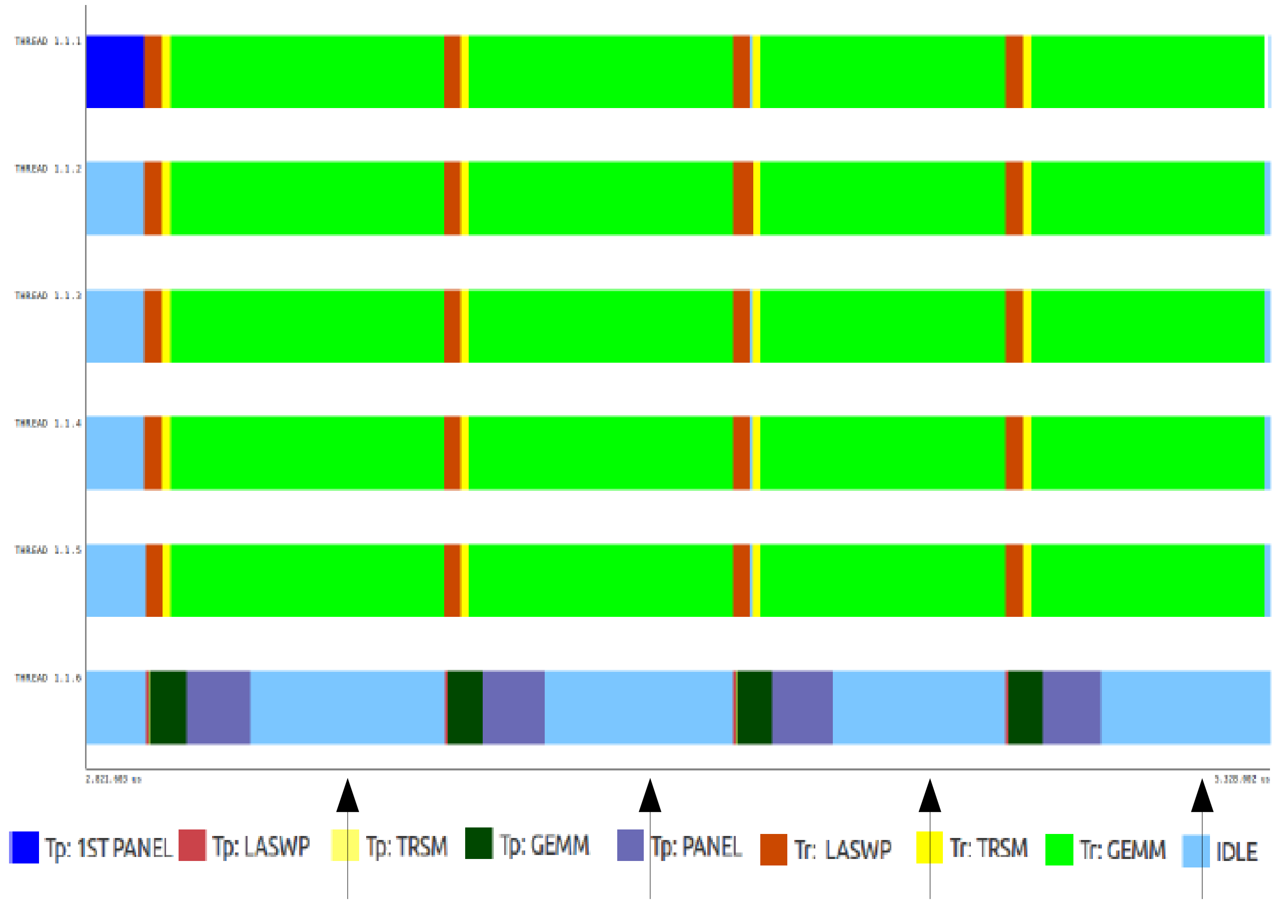
- Intel Xeon E5-2603 v3
- 6 cores at 1.6 Ghz
- BLIS 0.1.8
- BLIS Loop 4 (*jr*) parallelized
- Extrae 3.3.0
- Panel factorization via blocked algorithm
- Two block sizes b_o and b_i
- Inner LU involve small-grained computations and little parallelism

Look Ahead LU Performance



Execution trace of the first four iterations of the blocked RL LU factorization with partial pivoting, enhanced with look-ahead, applied to a square matrix of order 10,000.

Look Ahead LU Performance



Execution trace of the first four iterations of the blocked RL LU factorization with partial pivoting, enhanced with look-ahead, applied to a square matrix of order 10,000.

Towards malleability

- P threads in the panel factorization
- R threads in the update
- Panel factorization less expensive than update
  - P threads will join R team eventually
  - BLAS does not allow to modify the number of working threads

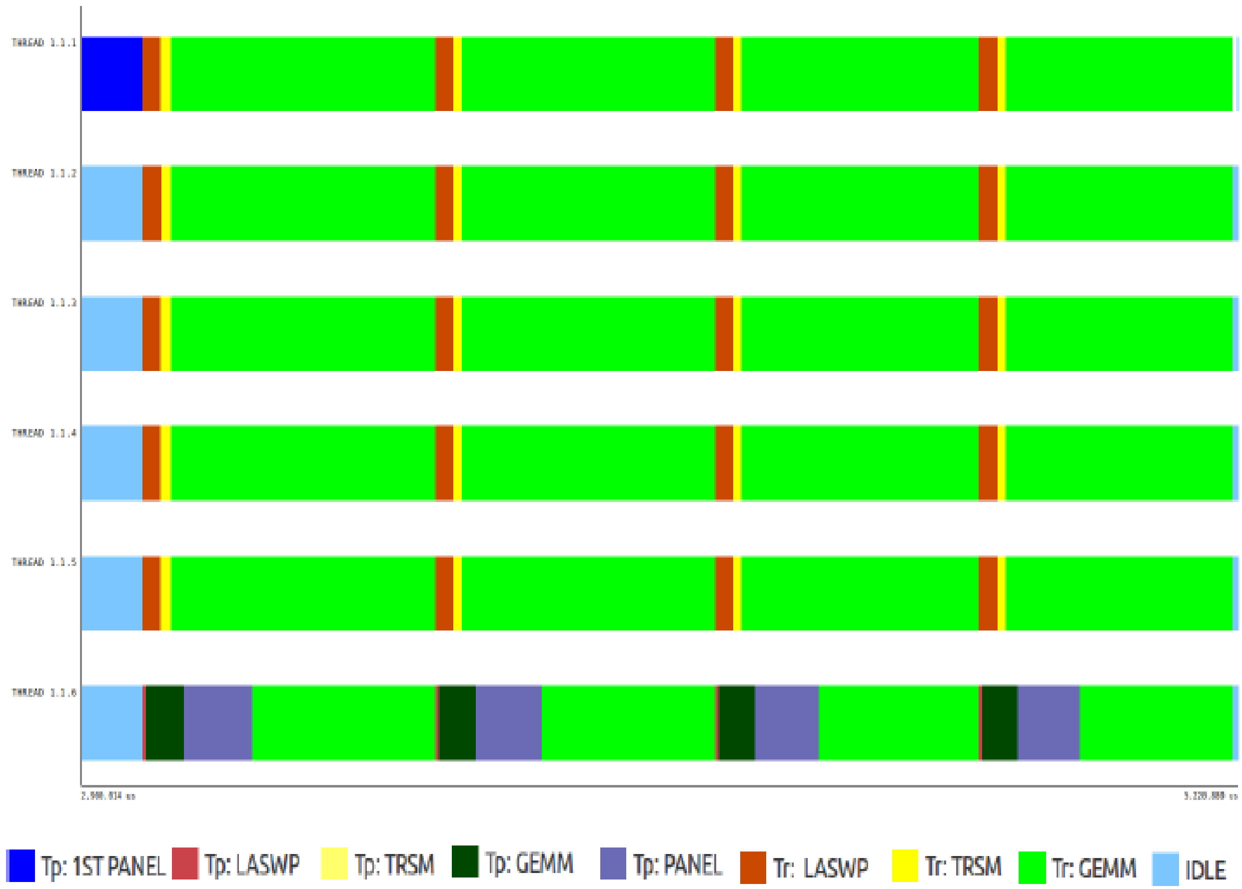
Static re-partitioning

- Workaround: split the update into several GEMM
- Drawbacks:
  - Lower GEMM throughput (packing and suboptimal blocks)
  - Decision on which loop to parallelize and the granularity of the partitioning

Malleable thread-level BLAS

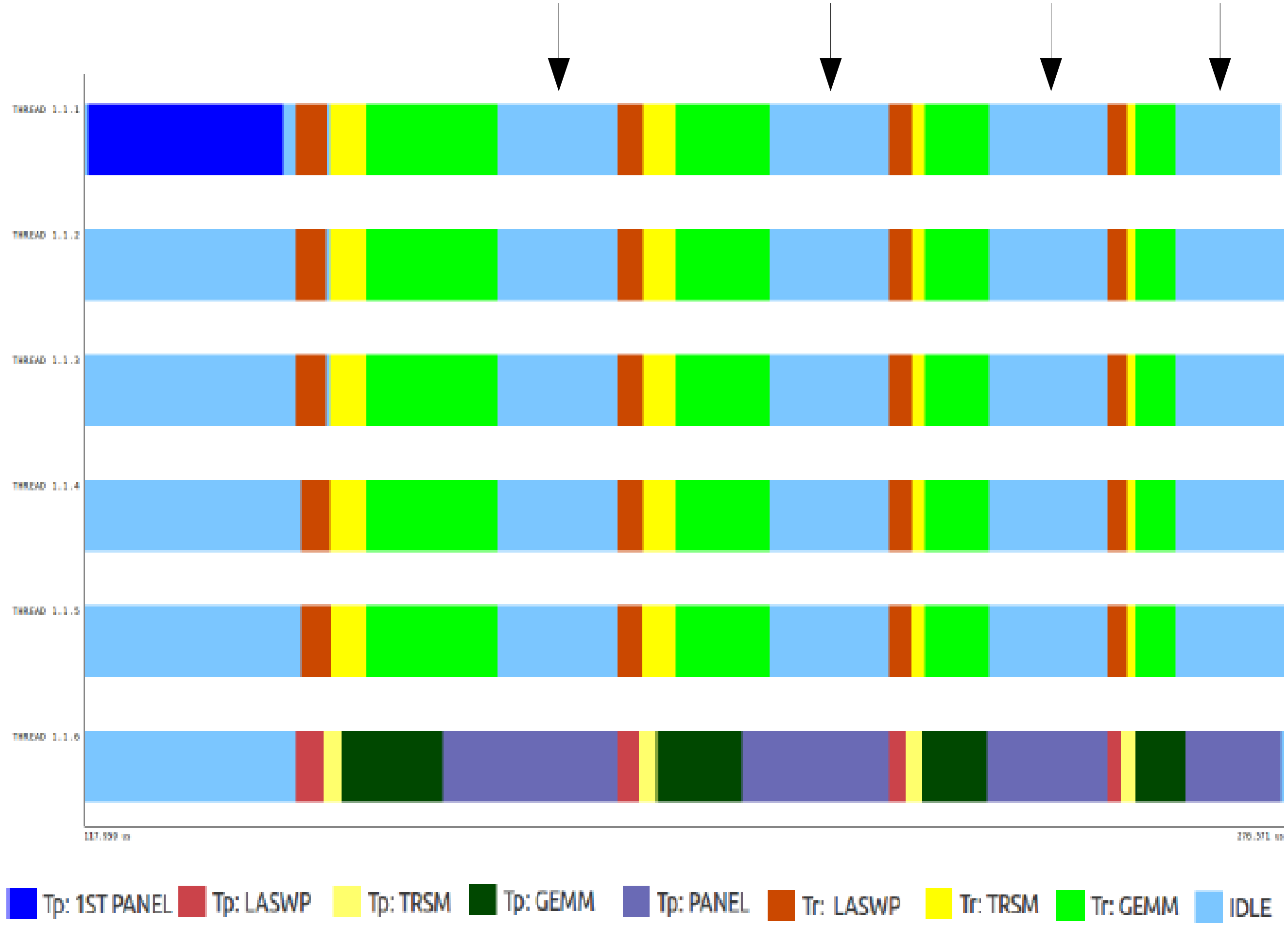
- Solving static partitioning issues:
  - Only one GEMM call → no extra data movements
  - BLIS takes care of the partitioning and granularity

How Malleability behaves



Execution trace of the first four iterations of the blocked RL LU factorization with partial pivoting, enhanced with look-ahead and *malleable BLIS*, applied to a square matrix of order 10,000.

And the small case...



Execution trace of the first four iterations of the blocked RL LU factorization with partial pivoting, enhanced with look-ahead, applied to a square matrix of order 2,000.

What if panel factorization is more expensive than the update

- If R finish before P → Stop panel factorization
  - RL LU. Keep a copy of the panel
  - Use LL LU. Sincronization among threads follows the same idea

Look ahead via runtimes

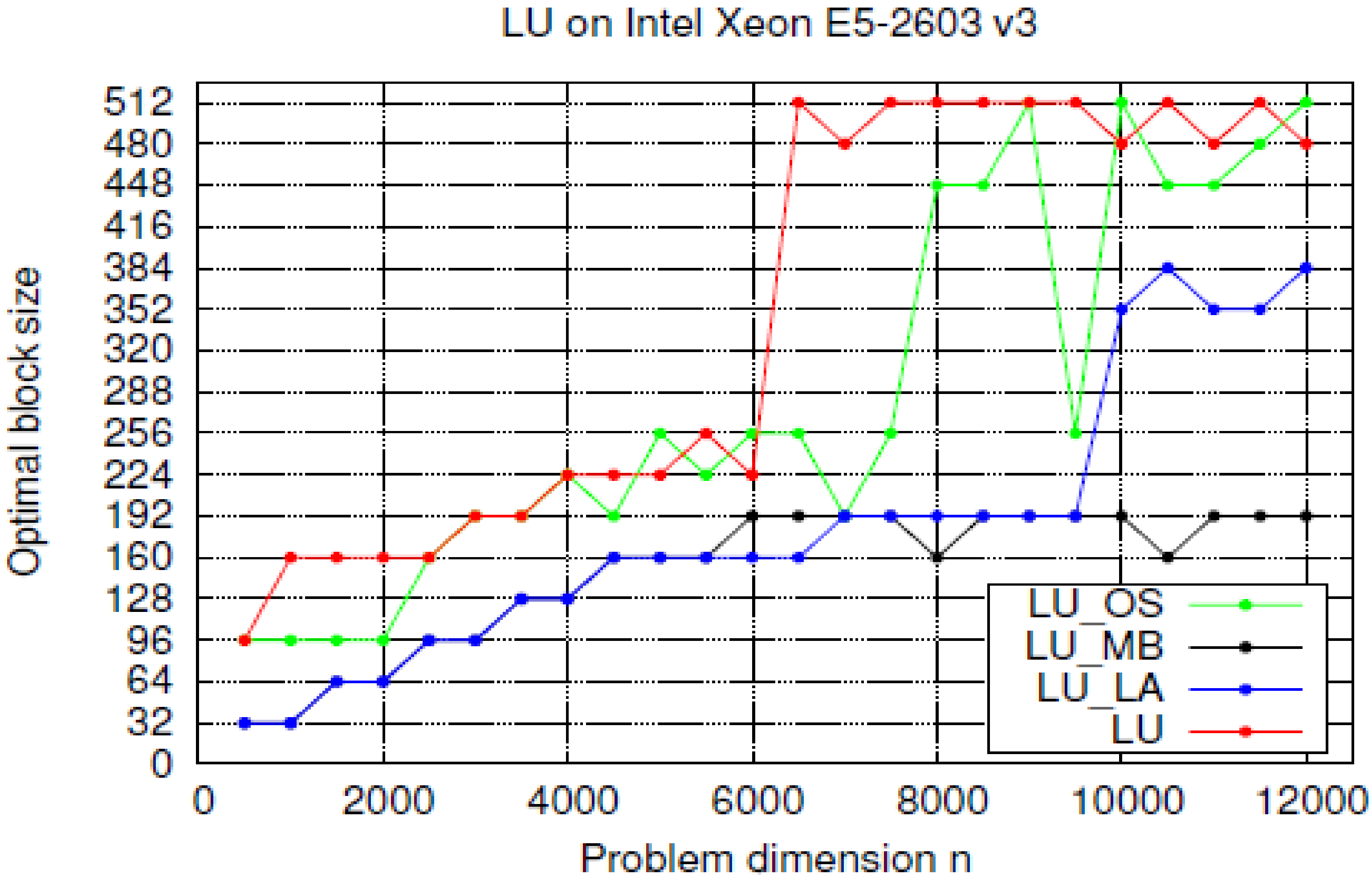
- ✔ TP execution
- ✔ Adaptative-depth look-ahead

- ✗ Re-packing and data movements (many GEMM calls)
- ✗ Block size fixes the granularity of the tasks
- ✗ Rarely exploit TP+TLP

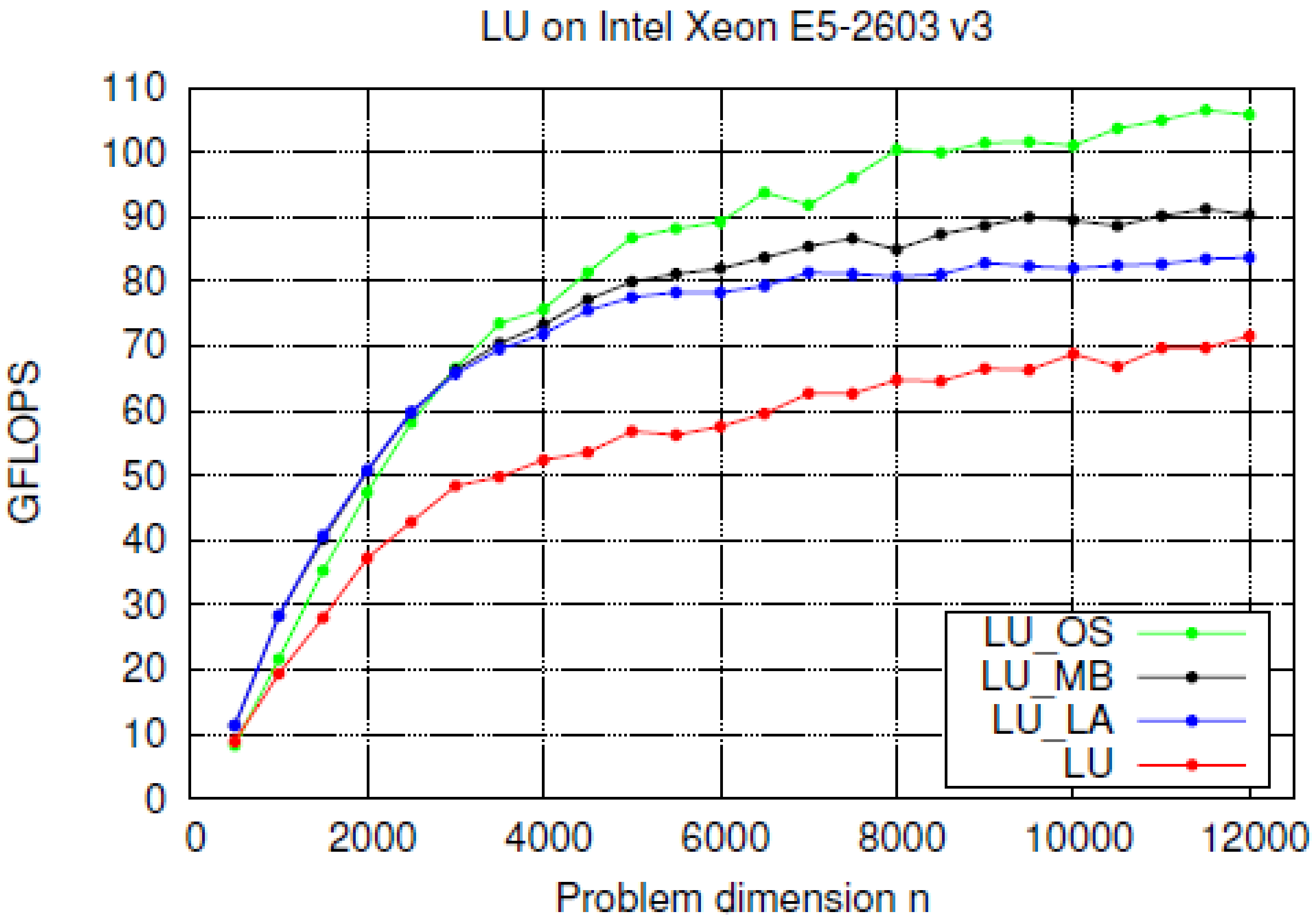
Experimental results

- LU, LU_LA, LU_MB, LU_OS
- Square matrices from n=500 to n=12,000
- b_o was tested for values from 32 to 512 in steps of 32
- b_i was evaluated for 16 and 32

Performance comparison



Performance comparison



Conclusions

- Malleable implementation of DLA library
- Competitive results (small matrices)
- Pending strategies to be applied (Early termination)

THANK YOU