Empirical Modelling of Linear Algebra Shared-Memory Routines

Jesús Cámara
Javier Cuenca
Luis P. García
Domingo Giménez

Scientific Computing and Parallel Programming Group
University of Murcia, SPAIN

UMU

ICCS 2013

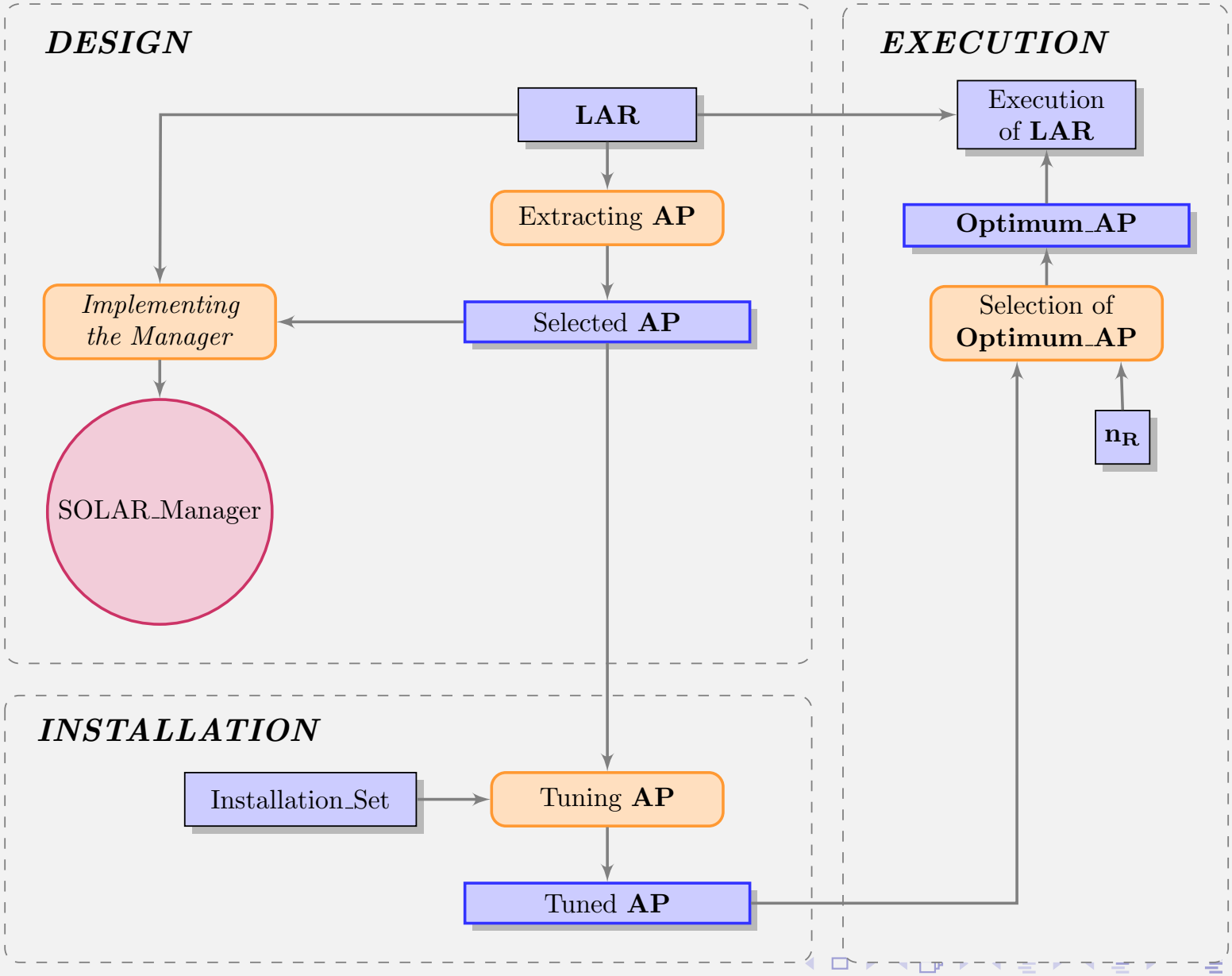
Introduction

- Multicore processor, cc-NUMA systems can offer performance improvements
  - Necessary software optimization techniques to benefit from the potential of the hardware
  - Modelling the execution time of the routine
  - Apply some empirical approach to study the behavior
- In this work:
  - Analysis of the behavior of multithread LAPACK routines on PLASMA and Intel MKL
  - Methodology for installation and modelling: take decisions at running time to reduce execution time
  - Typical decisions: number of threads, block or tile size, routine to use

Outline

- Introduction
- Auto-tuning methodology
- Motivation
- Routines and Computational Systems
- Empirical Modelling Method. PLASMA
- Comparison with Intel MKL LAPACK
- Conclusions and future work lines

Auto-tuning methodology



Motivation

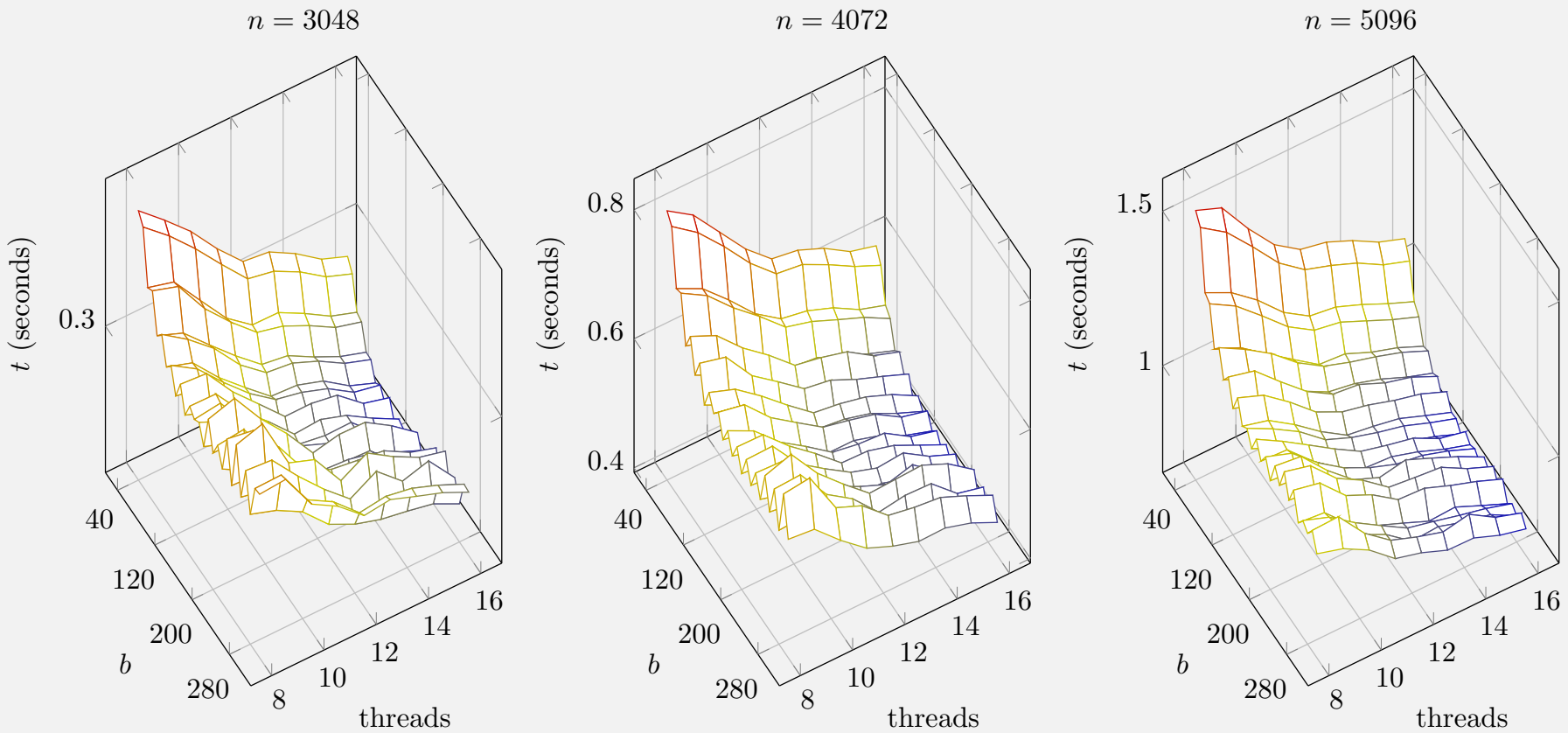
- PLASMA parallelism is not hidden inside BLAS
- PLASMA library relies on **TILE** algorithms
- Tile Algorithms: **OUTER BLOCK SIZE** and **INNER BLOCK SIZE**
- Auto-tuning PLASMA: finding the outer and inner block size pairs that maximize performance.
- BUT DEFAULT VALUES ARE USED FOR TILE SIZES

- Cholesky: 120
- LU: (200, 20)
- QR: (144, 48)

Motivation

- The optimum tile size depends on number of threads, matrix size and computing system

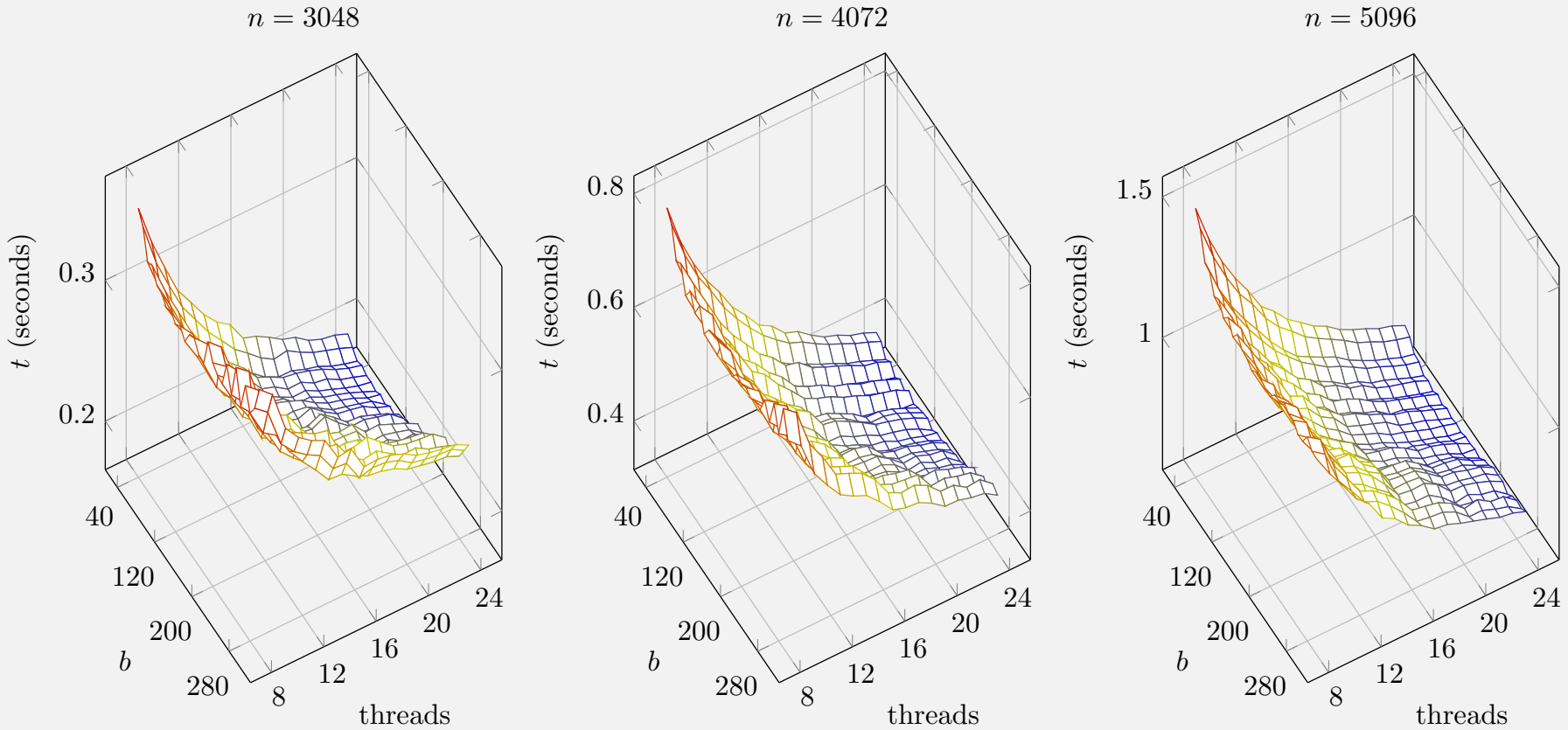
Cholesky factorization, 16 cores system



Motivation

- The optimum tile size depends on number of threads, matrix size and computing system

Cholesky factorization, 24 cores system



Routines and Computational Systems

LAPACK Routines

- Cholesky
- LU
- QR

Computational Systems

- **Hipatia**: System with 16 cores, 4 Intel Xeon Quad-Core, 2.93 GHz (4 cores). Linux 2.6.18, Intel `icc` (v12.0.0) and Intel MKL (v10.3.2)
- **Saturno**: NUMA system with 24 cores, 4 Intel Xeon X7542 (hexa-core) processors, 1.87 GHz, 32 GB of shared-memory. Linux 2.6.35, Intel `icc` compiler (v12.0.2) and Intel MKL (v10.3.2)
- **Joule**: NUMA system with 64 cores, 4 AMD Opteron 6276 (16 cores) processors, 2.3 GHz, 64 GB of shared-memory. Linux 2.6.32, Intel `icc` compiler (v12.1.3) and Intel MKL (v10.3.9)

Empirical Modelling Method

- Empirically estimated model of the execution time.
- Possible combinations: problem size and algorithm parameters (number of threads, block sizes, etc.)
- Algorithm parameter number of threads (t): $\{n^3, n^2, n, 1\} \times \{t, 1, \frac{1}{t}\}$

$$T(n,t) = k_1\frac{n^3}{t} + k_2 n^2 t + k_3 n^2 + k_4\frac{n^2}{t} + k_5 nt + k_6 n$$

- Experiments are performed for different values of n and t
- Estimation of the values of k_i with LS or NNLS
- LS: all coefficients have non zero values (positive or negative)
- NNLS: values obtained for the coefficients will be positive or zero

Empirical Modelling Method. PLASMA

- Algorithm parameters:
  - Number of threads (t). Outer block size (b). Inner block size (l)
  - Possible combinations: $\{n^3, n^2, n, 1\} \times \{t, 1, \frac{1}{t}\} \times \{b^2, b, 1, \frac{1}{b}\} \times \{l^2, l, 1, \frac{1}{l}\}$

$$
\begin{aligned}
T(n,t,b,l) = & k_1\frac{n^3}{t} + k_2\frac{n^3}{bl} + k_3\frac{n^3}{l} + k_4\frac{n^3}{b} + k_5 n^2 tb + k_6 n^2 t + k_7\frac{n^2 t}{bl} + \\
& k_8\frac{n^2 t}{l} + k_9\frac{n^2 t}{b} + k_{10} n^2 tl + k_{11} n^2 b + k_{12} n^2 l + k_{13}\frac{n^2}{bl} + k_{14}\frac{n^2}{l} + \\
& k_{15}\frac{n^2}{b} + k_{16} n^2 + k_{17}\frac{n^2}{t} + k_{18}\frac{n^2 b}{t} + k_{19}\frac{n^2 l}{t} + k_{20} ntb^2 + \\
& k_{21} ntbl + k_{22} ntb + k_{23} ntl^2 + k_{24}\frac{nt}{l} + k_{25}\frac{nt}{b} + k_{26} ntl + \\
& k_{27} nt + k_{28} nb^2 + k_{29} nbl + k_{30} nb + k_{31} nl^2 + k_{32} nl + k_{33} n
\end{aligned}
$$

Empirical Modelling Method. PLASMA

Installation

- Executing the routine with values of the parameters in an *Installation_Set*
- Varying t, b and l to some possible preselected values
- Estimation of 33 or 17 k_i coefficients with LS or NNLS
- Obtain `Mod-LS` or `Mod-NNLS` models for the execution time of the routine
- The model and the different possible values for the algorithm parameters are stored
- At execution time: the number of threads and tile sizes are selected for each problem size with the information provided by the model

Example. Empirically modelling of Cholesky

Installation_Set. Cholesky

- $n = \{500, 1500, 2500, 3500, 4500, 5500, 6500, 7500, 8500, 9500\}$
- b ranging from 40 to 300, $b\_inc = 40$
- t ranging from 4, 6 or 16 to the number of available cores, $t\_inc = 4, 6, 16$

`Mod-NNLS`

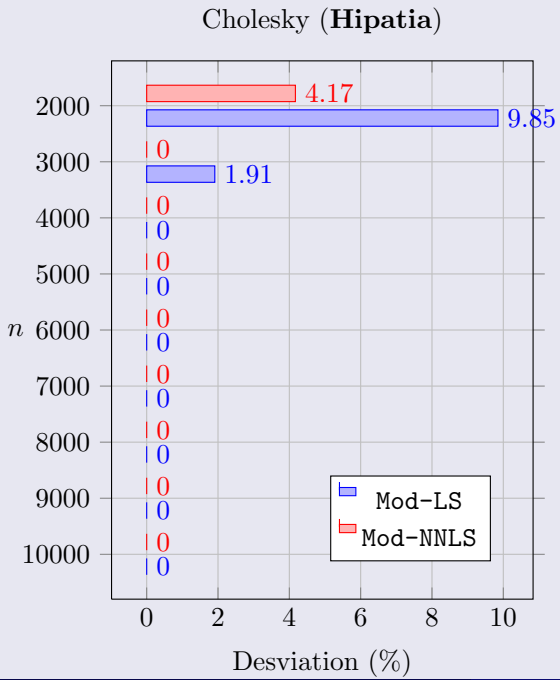
- Hipatia. $T(n,t,b) = k_1 \frac{n^3}{t} + k_2 \frac{n^3}{b}$
- Saturno. $T(n,t,b) = k_1 \frac{n^3}{t} + k_2 \frac{n^3}{b} + k_3 n^2 tb + k_{11} ntb^2$
- Joule. $T(n,t,b) = k_1 \frac{n^3}{t} + k_2 \frac{n^3}{b} + k_3 n^2 tb + k_6 n^2 b + k_{11} ntb^2$

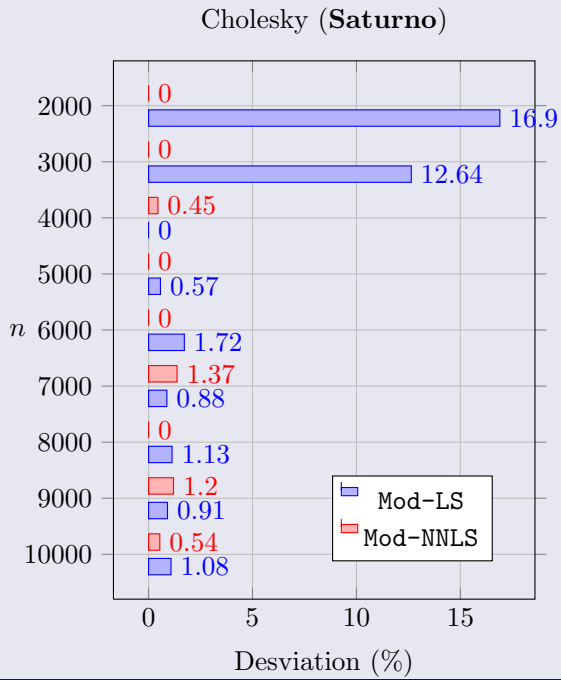
With the `Mod-NNLS` the non-zero coefficients change with the execution platform. Insight about the contribution of the value of the algorithmic parameters.

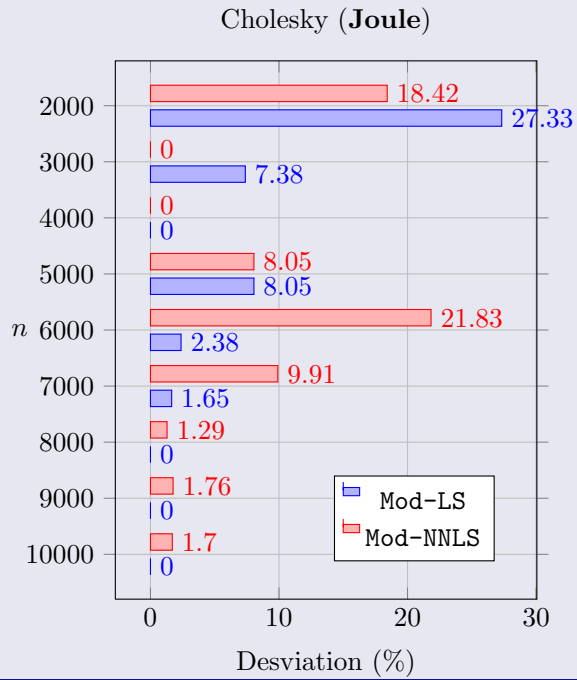
Example. Empirically modelling of Cholesky

Validation_Set ≠ *Installation_Set*

	hipatia			Saturno			Joule		
size	Optimum	Mod-LS	Mod-NNLS	Optimum	Mod-LS	Mod-NNLS	Optimum	Mod-LS	Mod-NNLS
2000	(12,200)	(8,280)	(16,280)	(24,80)	(12,80)	(24,80)	(32,40)	(16,200)	(64,40)
3000	(16,280)	(12,280)	(16,280)	(24,80)	(18,80)	(24,80)	(64,60)	(48,40)	(64,60)
4000	(16,280)	(16,280)	(16,280)	(24,120)	(24,120)	(24,80)	(64,60)	(64,60)	(64,60)
5000	(16,280)	(16,280)	(16,280)	(24,120)	(24,160)	(24,120)	(64,140)	(64,60)	(64,60)
6000	(16,280)	(16,280)	(16,280)	(24,120)	(24,160)	(24,120)	(64,140)	(64,60)	(64,80)
7000	(16,280)	(16,280)	(16,280)	(24,160)	(24,200)	(24,120)	(64,140)	(64,60)	(64,80)
8000	(16,280)	(16,280)	(16,280)	(24,160)	(24,240)	(24,160)	(64,200)	(64,200)	(64,100)
9000	(16,280)	(16,280)	(16,280)	(24,200)	(24,240)	(24,160)	(64,200)	(64,200)	(64,100)
10000	(16,280)	(16,280)	(16,280)	(24,200)	(24,280)	(24,160)	(64,200)	(64,200)	(64,100)







Example. Empirically modelling of QR

Installation_Set. QR

- $n = \{512, 1024, 1536, 2048, 2560, 3072, 3584, 4096, 4608, 5120\}$
- b ranging from 24 to 304, $b\_inc = 40$
- l ranging form 28 to 208, $l\_inc = 20$
- t ranging from 4, 6 or 16 to the number of available cores, $t\_inc = 4, 6, 16$

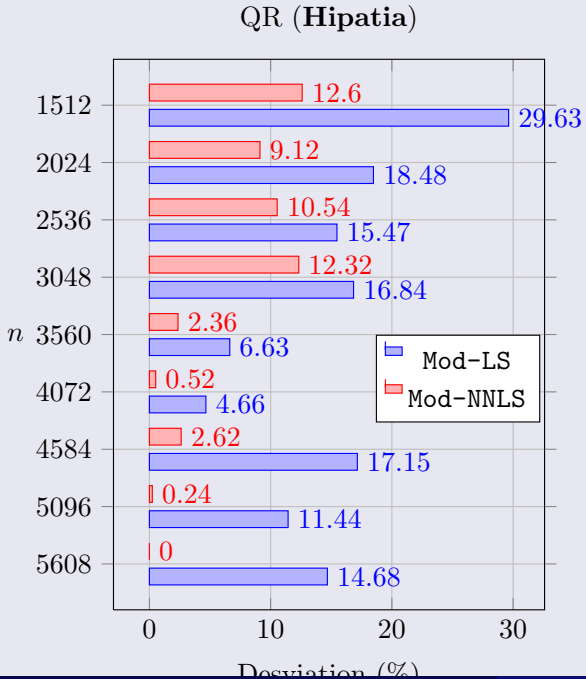
Mod-NNLS

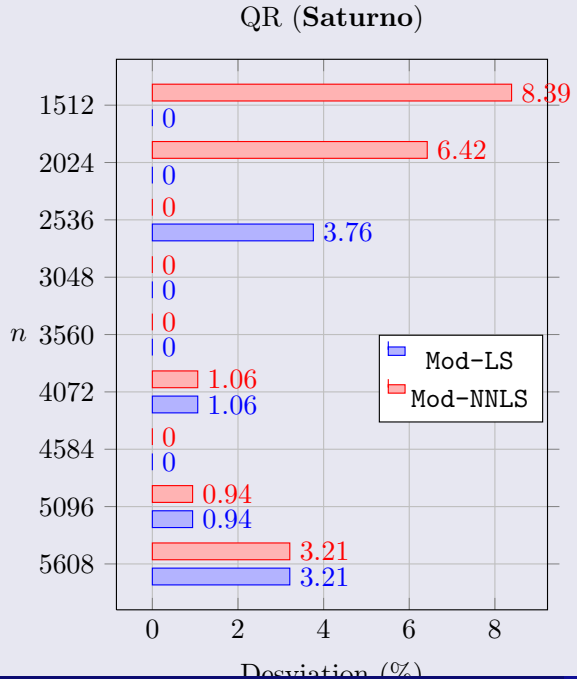
- Hipatia. $T(n,t,b,l) = k_1\frac{n^3}{t} + k_2\frac{n^3}{bl} + k_{19}\frac{n^2 l}{t}$
- Saturno. $T(n,t,b,l) = k_1\frac{n^3}{t} + k_4\frac{n^3}{b} + k_{10}n^2tl + k_{20}ntb^2 + k_{23}ntl^2 + k_{24}\frac{nt}{l}$
- Joule. $T(n,t,b,l) = k_1\frac{n^3}{t} + k_4\frac{n^3}{b} + k_{12}n^2 l + k_{13}\frac{n^2}{bl} + k_{17}\frac{n^2}{t} + k_{19}\frac{n^2 l}{t} + k_{20}ntb^2 + k_{21}ntl^2 + k_{28}nb^2$

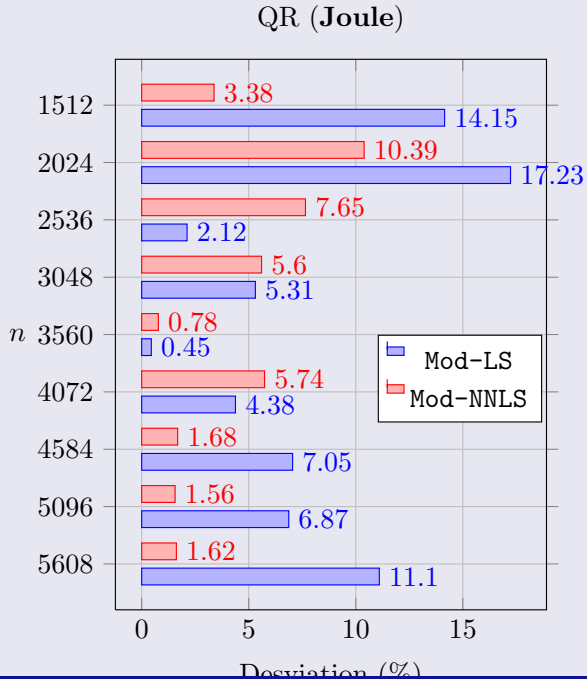
Example. Empirically modelling of QR

Validation_Set ≠ *Installation_Set*

	hipatia			Saturno			Joule		
size	Optimum	Mod-LS	Mod-NNLS	Optimum	Mod-LS	Mod-NNLS	Optimum	Mod-LS	Mod-NNLS
1512	(16,224,88)	(12,304,188)	(16,304,68)	(24,104,28)	(24,104,28)	(24,104,68)	(64,64,28)	(32,104,28)	(64,64,48)
2024	(16,264,88)	(16,304,188)	(16,304,68)	(24,104,28)	(24,104,28)	(24,104,68)	(48,64,28)	(64,104,48)	(64,64,48)
2536	(16,264,88)	(16,304,188)	(16,304,68)	(24,144,48)	(24,144,28)	(24,144,48)	(64,64,28)	(64,104,28)	(64,104,48)
3048	(16,264,88)	(16,304,168)	(16,304,88)	(24,144,48)	(24,144,48)	(24,144,48)	(64,64,28)	(64,104,28)	(64,104,48)
3560	(16,264,88)	(16,304,168)	(16,304,88)	(24,144,48)	(24,184,48)	(24,144,48)	(64,64,28)	(64,104,28)	(64,104,48)
4072	(16,304,108)	(16,304,168)	(16,304,88)	(24,184,48)	(24,184,48)	(24,184,48)	(64,144,48)	(64,104,28)	(64,104,48)
4584	(16,264,88)	(16,304,28)	(16,304,108)	(24,184,48)	(24,224,48)	(24,184,48)	(64,144,48)	(64,104,28)	(64,144,28)
5096	(16,304,88)	(16,304,28)	(16,304,108)	(24,184,48)	(24,224,48)	(24,224,48)	(64,144,48)	(64,104,28)	(64,144,28)
5608	(16,304,108)	(16,304,28)	(16,304,108)	(24,144,48)	(24,224,48)	(24,224,48)	(64,144,48)	(64,104,28)	(64,144,28)







Global results. Empirically modelling of Cholesky and QR

Cholesky

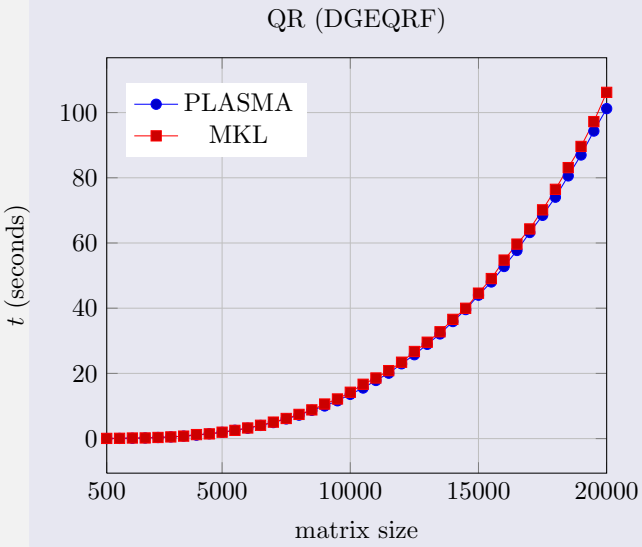
- The mean deviation of the optimum:
  - `Mod-LS`: 1 % (Hipatia), 4 % (Saturno), 5 % (Joule)
  - `Mod-NNLS`: 0.5 % (Hipatia), 0.4 % (Saturno), 7 % (Joule)
- The mean improvement with respect to the Default execution is 27 % in Hipatia, 1 % in Saturno and 13 % in Joule.

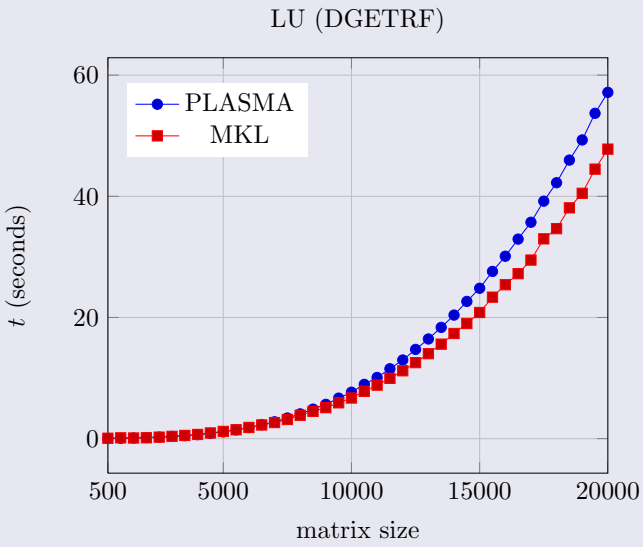
QR

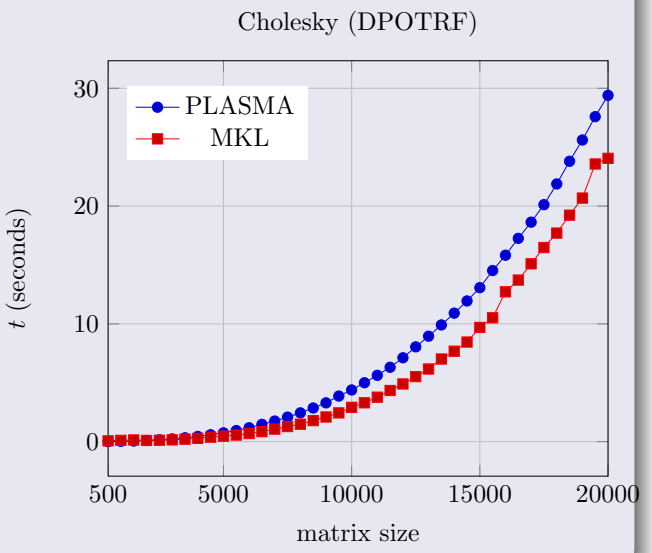
- The mean deviation of the optimum:
  - `Mod-LS`: 15 % (Hipatia), 1 % (Saturno), 8 % (Joule)
  - `Mod-NNLS`: 6 % (Hipatia), 2 % (Saturno), 4 % (Joule)
- The mean improvement with respect to the Default execution is 34 % in Hipatia, 2 % in Saturno and 7 % in Joule.

Comparison with Intel MKL LAPACK

Hipatia

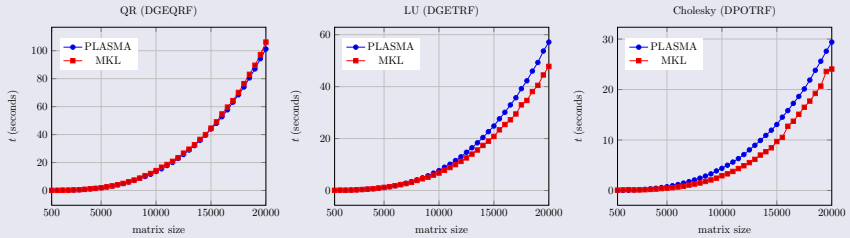




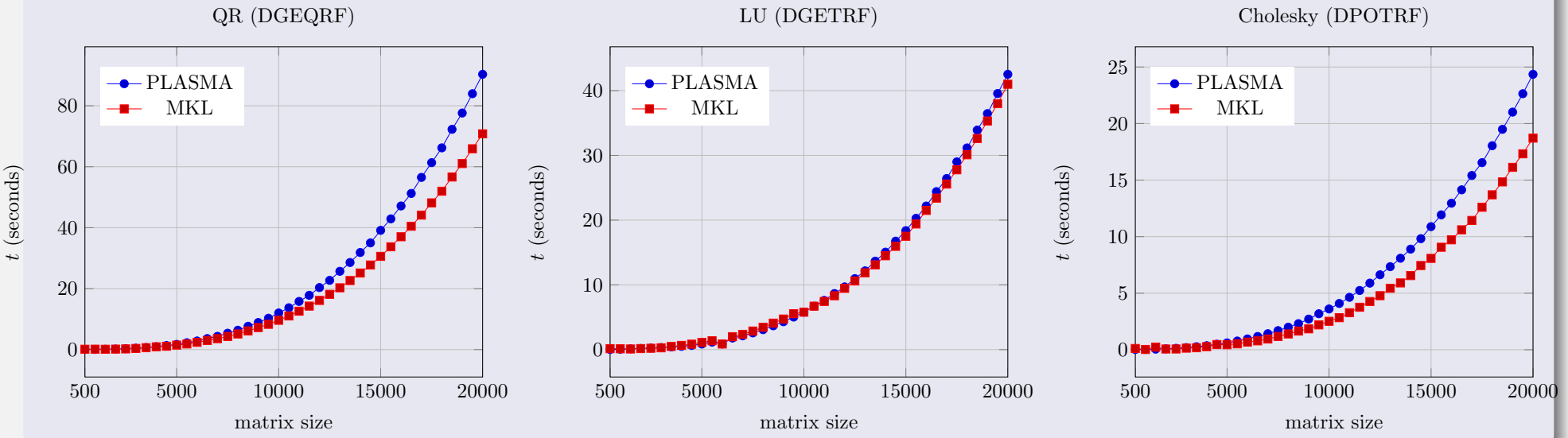


Comparison with Intel MKL LAPACK

Hipatia

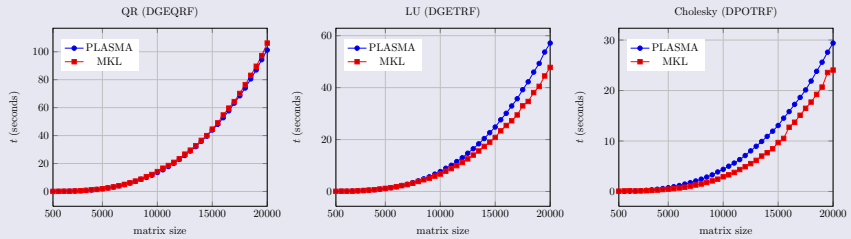


Saturno

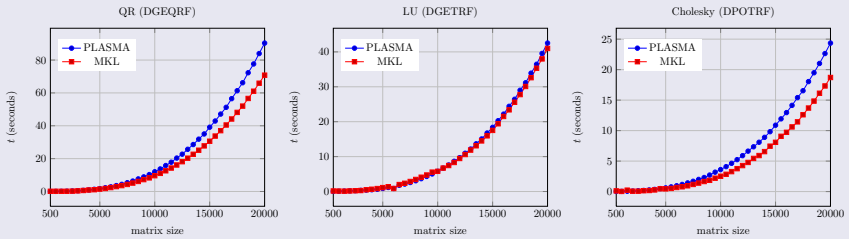


Comparison with Intel MKL LAPACK

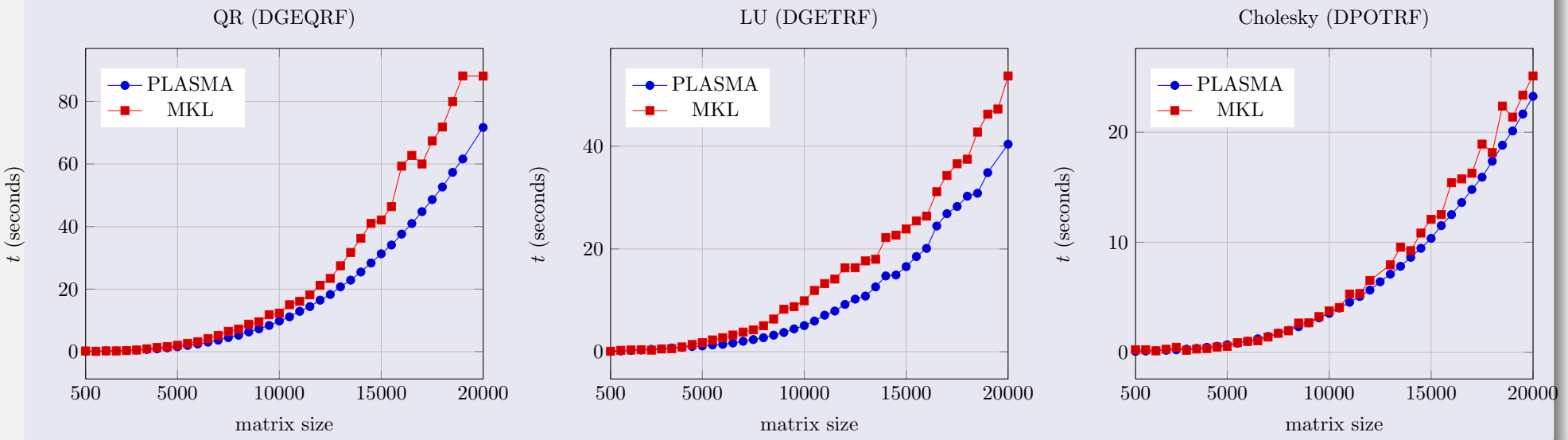
Hipatia



Saturno

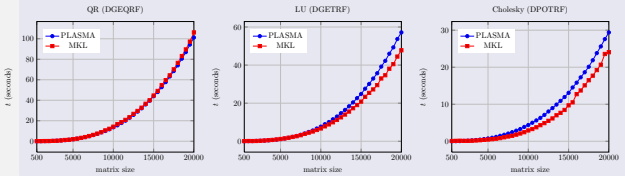


Joule

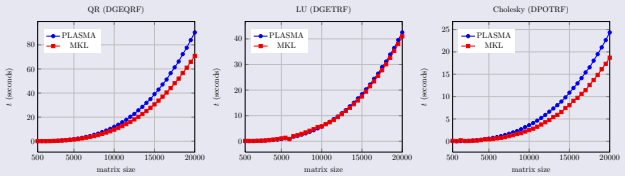


Comparison with Intel MKL LAPACK

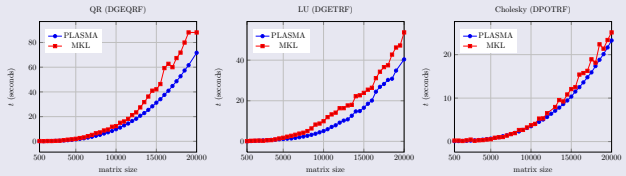
Hipatia



Saturno



Joule



- Intel MKL **outperform** PLASMA for large matrices, except QR or in Joule.
- Impossible to draw general conclusions about the **advantage** of using MKL
- PLASMA can **compete** against MKL. Correct selection of parameters
- The auto-tuning methodology can be used to decide the **implementation** to use and the correct **selection** of the tile sizes and number of threads

Comparison with Intel MKL LAPACK. LU

PLASMA LU vs MKL LU. Hipatia

size	Mod-LS	Mod-NNLS	Default	MKL
2000	0.176 (16,120,20)	0.239 (16,280,40)	0.190	**0.147**
3000	0.433 (16,120,20)	0.447 (16,280,60)	**0.377**	0.378
4000	0.726 (16,280,100)	0.729 (16,280,60)	**0.669**	0.695
5000	1.124 (16,280,120)	**1.121** (16,280,80)	1.127	1.196
6000	**1.745** (16,280,240)	1.777 (16,280,80)	1.820	1.853
7000	2.621 (16,280,20)	**2.619** (16,280,80)	2.779	2.659
8000	**3.797** (16,280,20)	3.821 (16,280,40)	4.088	3.863
Total	**10.622**	10.753	11.05	10.791

- The improvement with Mod-LS and Mod-NNLS are close
- Best times with Mod-LS. Preferred installation method
- Difference of 4 % with respect to the Default

Comparison with Intel MKL LAPACK. LU

PLASMA LU vs MKL LU. Saturno

size	Mod-LS	Mod-NNLS	Default	MKL
3000	**0.214** (24,120,20)	0.464 (24,280,60)	0.324	0.279
4000	**0.383** (24,160,60)	0.647 (24,280,60)	0.504	0.653
5000	**0.692** (24,160,60)	0.859 (24,280,60)	0.865	1.121
6000	**1.144** (24,160,60)	1.184 (24,280,80)	1.156	1.853
7000	1.777 (24,160,120)	**1.770** (24,280,80)	2.131	2.359
8000	2.508 (24,200,20)	**2.498** (24,280,80)	2.508	3.452
9000	3.521 (24,200,20)	**3.464** (24,280,100)	3.521	4.727
Total	**10.239**	10.886	11.09	14.444

- The improvements with Mod-LS and Mod-NNLS are close
- Best times with Mod-LS. Preferred installation method
- Difference of 7 % with respect to the Default

Comparison with Intel MKL LAPACK. LU

PLASMA LU vs MKL LU. Joule

size	Mod-LS	Mod-NNLS	Default	MKL
3000	0.750 (64,120,20)	0.828 (64,280,80)	0.596	**0.536**
4000	**0.822** (64,160,160)	1.164 (64,280,60)	0.848	0.938
5000	**1.137** (64,200,200)	1.672 (64,280,60)	1.168	1.775
6000	**1.458** (64,200,200)	2.122 (64,280,60)	1.493	2.736
7000	**1.939** (64,200,200)	2.817 (64,280,80)	2.029	3.826
8000	**2.730** (64,200,20)	2.758 (64,280,80)	**2.730**	5.066
9000	**3.738** (64,200,20)	3.983 (64,280,80)	**3.738**	8.259
Total	**12.574**	15.344	12.602	23.136

- The auto-tuning selects satisfactory values of the parameters
- Similar results to those obtained without parameters tuning: difference of 0.22 % with respect to the Default

Conclusions

- Empirical auto-tuning approach for PLASMA implementation of LAPACK routines: Provide a **set** of empirically obtained **models**
- The models facilitate a **satisfactory** selection of the algorithmic parameters
- This work focus on the Cholesky, QR and LU factorizations. But it is representative of the process to be done for **auto-tuning** the **whole** library
- The methodology is useful to obtain execution times close to the lowest achievable without the need for user **expertise**
- The results have shown that PLASMA can, with an autotuning methodology, be **competitive** compared to the Intel MKL library