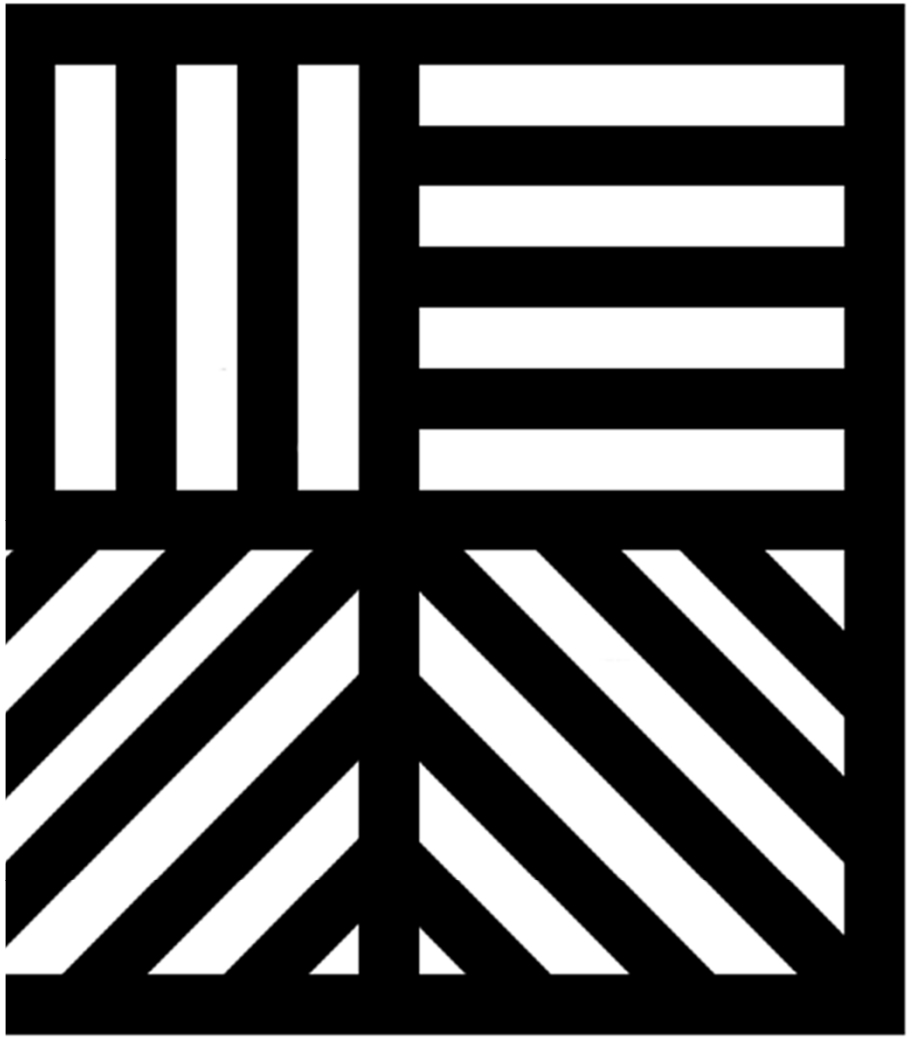
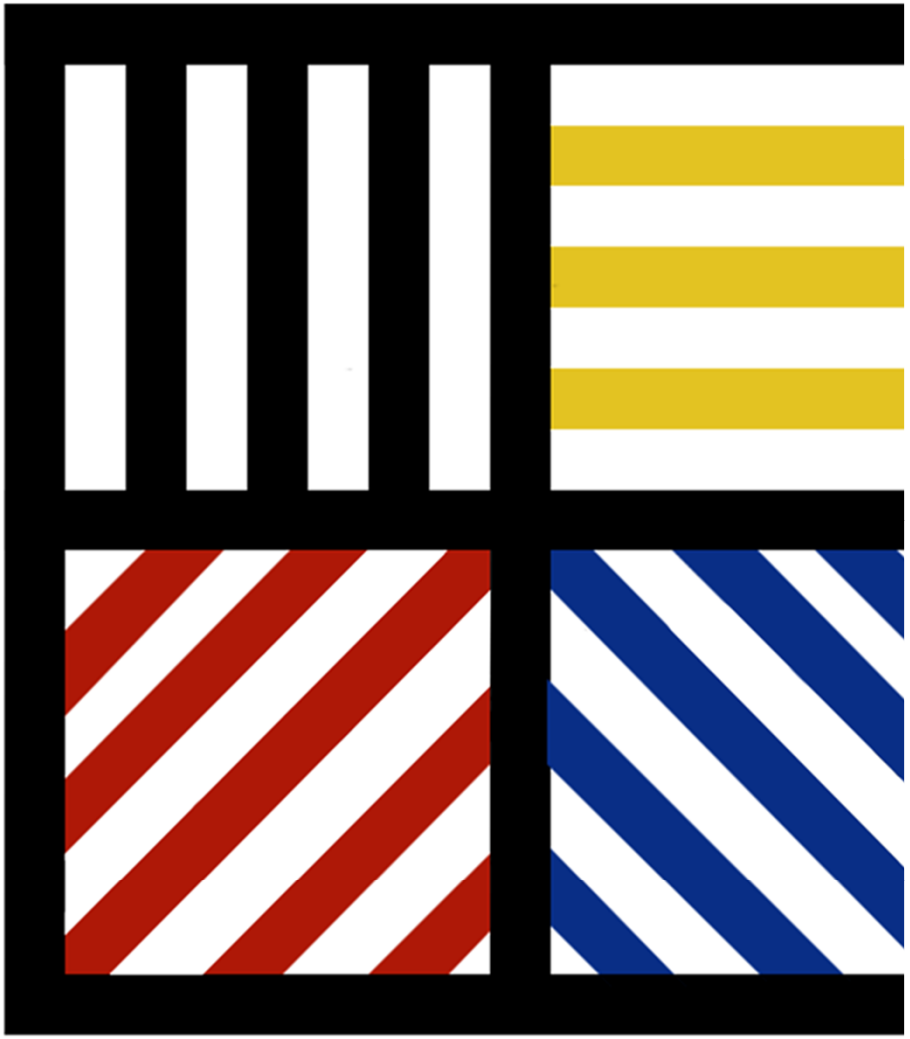
LinBox 2013





JNCF Luminy Avril 2013

LinBox C++ Library

- USA-France-Canada Free Software, LGPL
- 141 000 lines of code (sloccount) & documentation
- http://linalg.org
- Exact Linear Algebra
  - Efficient
  - Generic
- Now: linbox-1.3.2

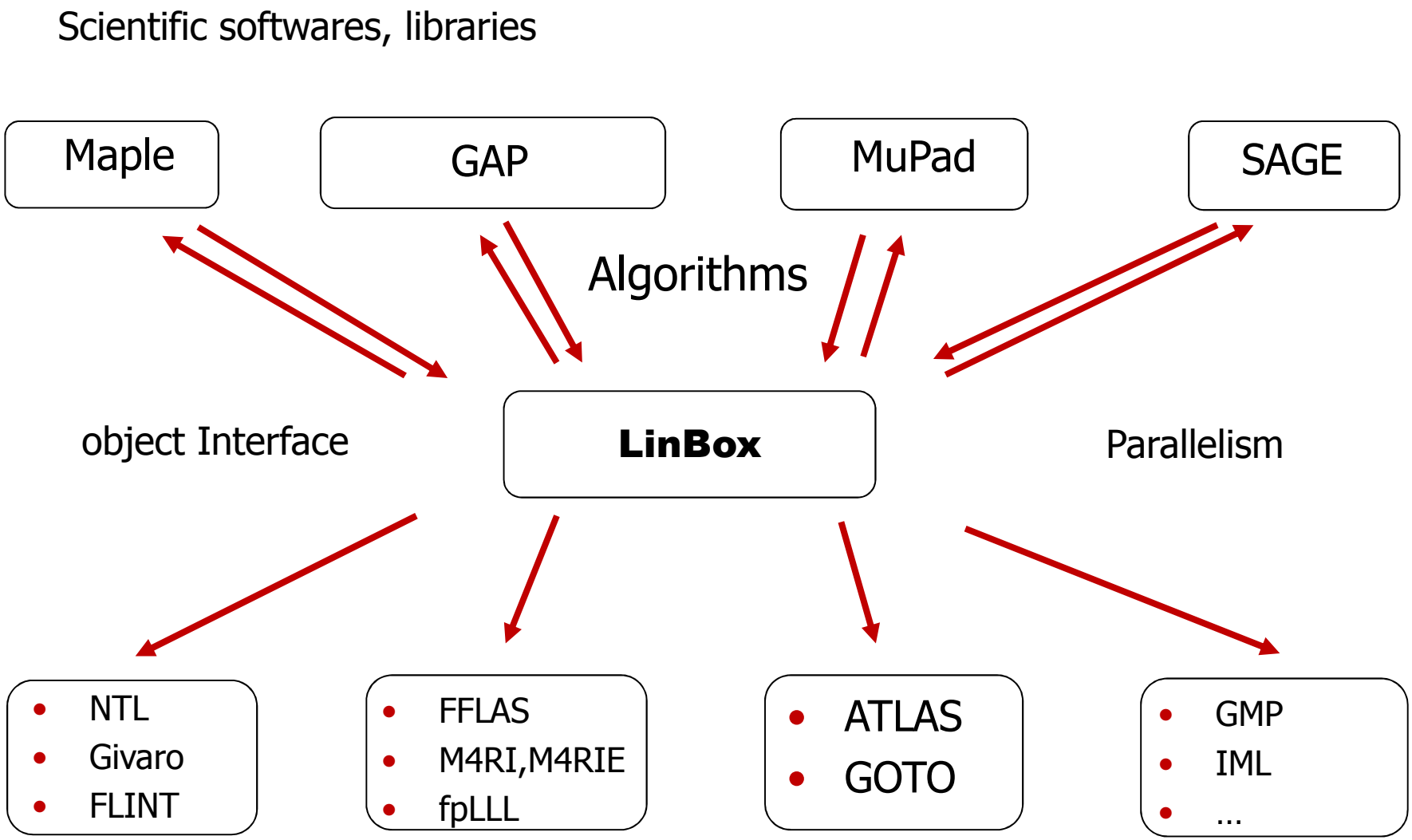
Generic Exact Linear Algebra

- Genericity with respect to the computational domain
  - Modular arithmetic
  - Finite Fields
  - Integers, Rationals
  - Polynomials

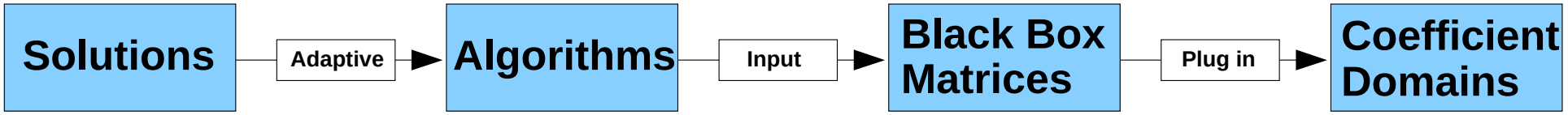
- Genericity with respect to the matrix type
  - Dense
  - Structured
  - Black box
  - Sparse

⇨ Remains efficient by using any new specialization ...

A middleware



Generic Design of The LinBox library



1. Adaptive solutions
   - Framework for automated adaptivity: DATA & RESOURCES aware
   - Object oriented modeling
2. Generic algorithms
   - Best complexities
3. Optimized blackbox / dense data structures
   - BLAS based dense matrices
   - Row, column, raw, sparse iterators
4. Numerous coefficient domains
   - Finite fields and rings (Zech, float, Montgomery, etc.)
   - Integers, rationals
   - Polynomials

LinBox

- Solutions for exact
  - Solving linear systems
  - Echelon forms
  - Rank
  - Determinant
  - Minimal Polynomial
  - Characteristic Polynomial
  - Frobenius normal form
  - Integer Smith normal form
  - Trace
  - Valence
  - Signature

Field, Blackbox, Solution

```
#include <linbox/field/modular.h>
#include <linbox/blackbox/sparse.h>
#include <linbox/solutions/det.h>
…
LinBox::Modular<> F(13);
LinBox::SparseMatrix< Modular<> > A(F);
…
LinBox::Modular<>::Element d;

// Chooses our best guessed determinant algorithm for
// this particular dimension and sparsity modular matrix
// → Both static (types) and dynamic (dimension, sparsity, resources …)
//    choices, hybridizations, adaptations
LinBox::det( d, A,Method::Wiedemann( Specifier::SYMMETRIC) );
```

LinBox: efficient reductions

- Dense

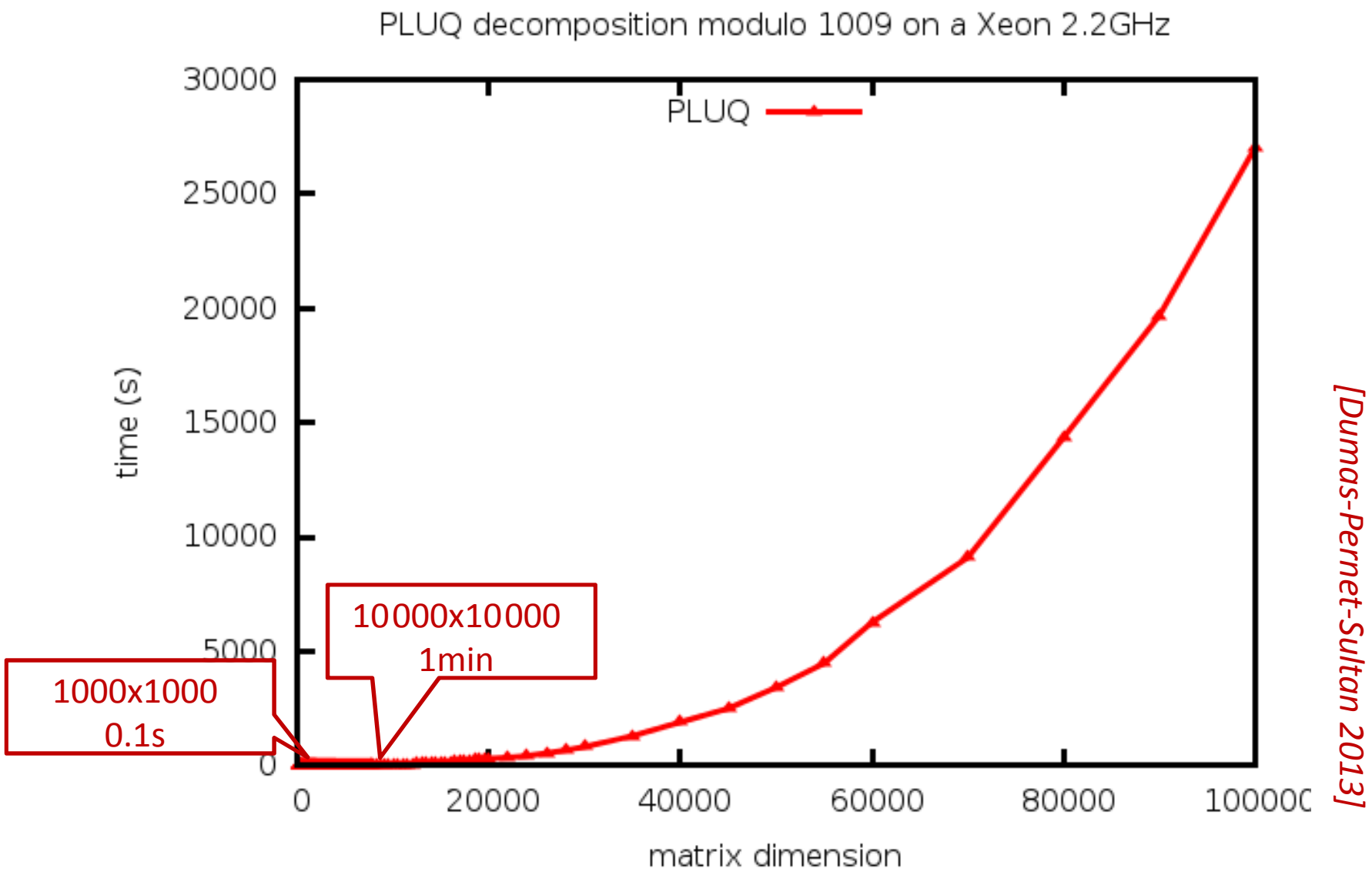
N = 10 000 $\mathbb{Z}/547909\,\mathbb{Z}$	MatMul (171s)	PLUQ	RHS-TRSM	Inverse (red. Ech.)	Charpoly
Xeon 2.53 GHz	**Ratio: 1**	**0.46**	**0.55**	**1.30**	**2.88**
~ G ff op/s	11.69	8.58	10.47	9.01	4.06

  - Characteristic polynomial: *[DPW05]* 2.66 n^3 & *[PS07]* $\approx 5n^{\omega}$

Athlon 1.8GHz	100	300	500	800	1000	1500	2000	3000	5000	7500	10 000
magma-2.11	0.010s	0.830s	3.810s	15.64s	29.96s	102.1s	238.0s	802.0s	3793s	MT	MT
$[DPW05]+[PS07]$	0.005s	0.105s	0.387s	1.387s	2.755s	7.696s	17.91s	61.09s	273.4s	991.4s	2080s

- Rang, forme normale de Smith de grandes matrices creuses
  - Dickson SRG7 : $4\,782\,969 \times 4\,782\,969$ ⇨ rang $32\,064$
  - GL7d19 : $1\,911\,130 \times 1\,955\,309$ ⇨ rang $1\,033\,568$
  - GL7d18 : $1\,955\,309 \times 1\,548\,650$, avec $33\,590\,540$ éléments non nuls
    - ⇨ rang $921\,740$
    - ⇨ Forme normale de Smith entière : $1\ (921\,637) \oplus 2(100) \oplus 42\ (2) \oplus 252\ (1)$
  - ...

Dense PLUQ



[Dumas-Pernet-Sultan 2013]

Adaptivity

- Challenge in Symbolic Software
  - Many variants for every problem
  - Recursive Cascading:
    - Should provide optimal choice for every data
  - Lower bound: the race !
    - Run all the algorithms at the same time, kill long runners
  - Examples that beats the race:
    - *[Saunders-Wan]* elimination/iterative rank
    - Cascade algorithms with tuned thresholds (TRSM, PLUQ, …)

- Challenge in nowadays environments (grid, multi-core)
  - Adapt to the volatile/unsafe resources
  - Parallel/Sequential degeneration

ANR HPAC (LinBox, FGb, …): some Kernels

- Dense: pFFPACK set of routines

A1. Parallel Matrix-Mul on top of parallel BLAS: coupling ?
A2. Parallel MM word size arith
A3. Parallel MM cryptographic size moduli
A4. Parallel hybrid dense-sparse MM

B1. Parallel Echelon form variants
  - row, column, reduced, LQUP, etc.
  - over $\mathbb{F}_p$

B2. Parallel Echelon form for cryptographic size moduli
B3. Parallel hybrid dense-sparse Gaussian elimination

…

Software abstraction layer for data parallelism

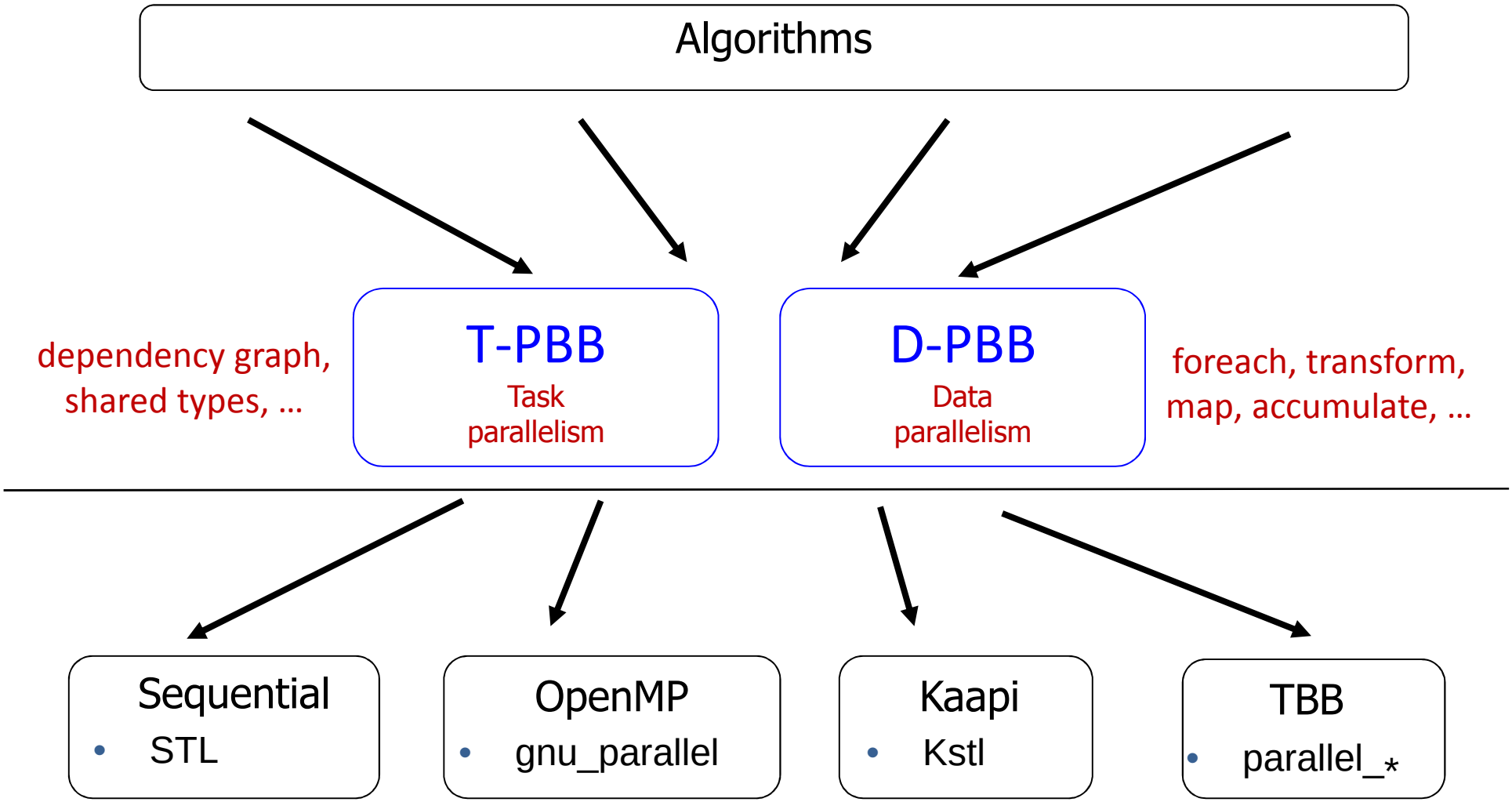
- Interface between linear algebra kernels and parallel algorithms *[Dumas-Gautier-Saunders 2010]*

⇨ «parallel building block» (parallel skeleton)

⇨ Data-PBB: a set of STL-like algorithms that may have parallel implementations

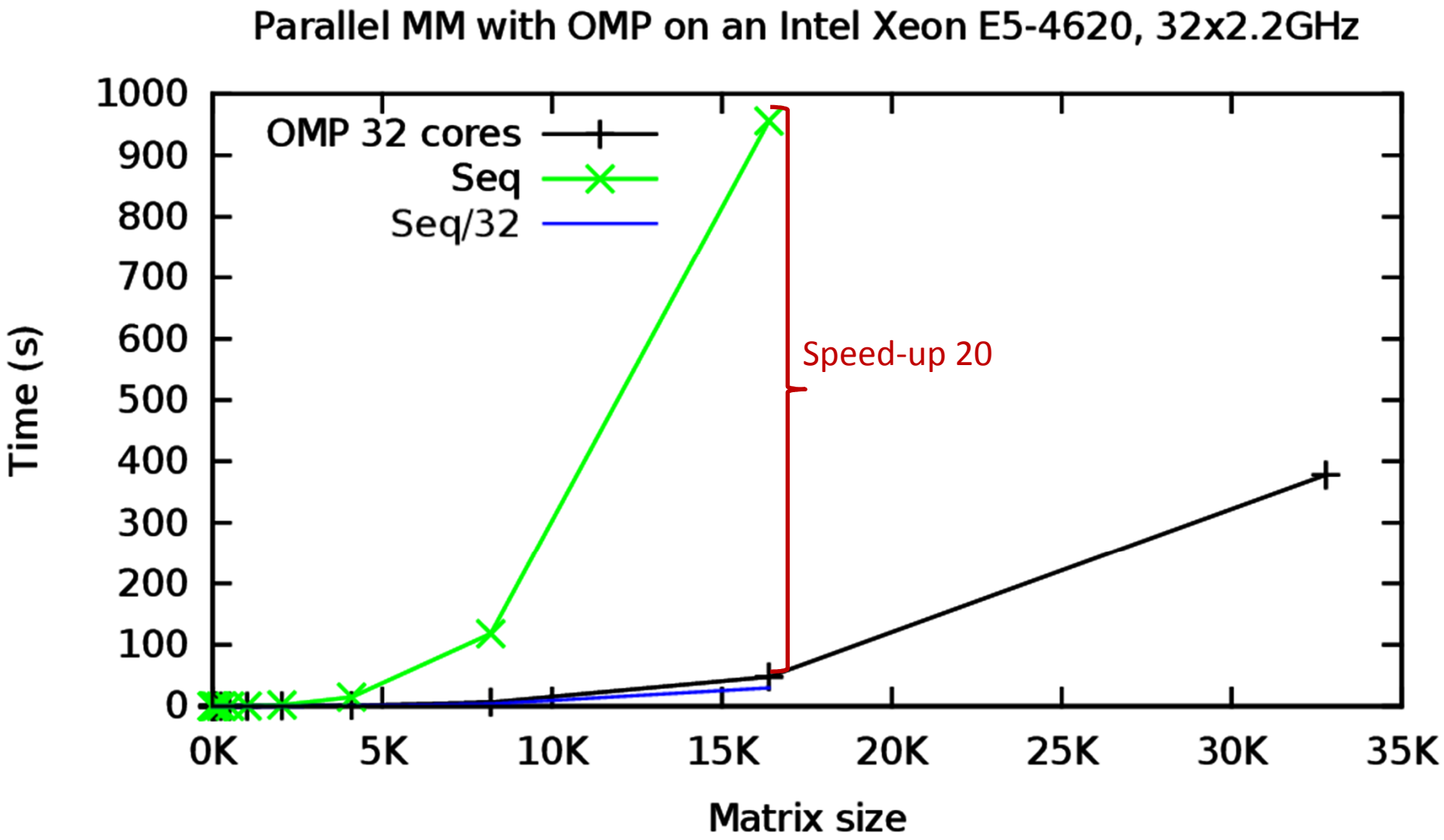
- *pbb::for_each*: $\forall_i$, f(input[i])
- *pbb::transform*: $\forall_i$, output[i] = f(input[i])
- *pbb::accumulate*: $\sum_i$ input[i]
- New ! *pbb::accumulate_while*: $\sum_i$ input[i] while cond($\sum_i$)

Coarse grain Transparent Data Parallelism

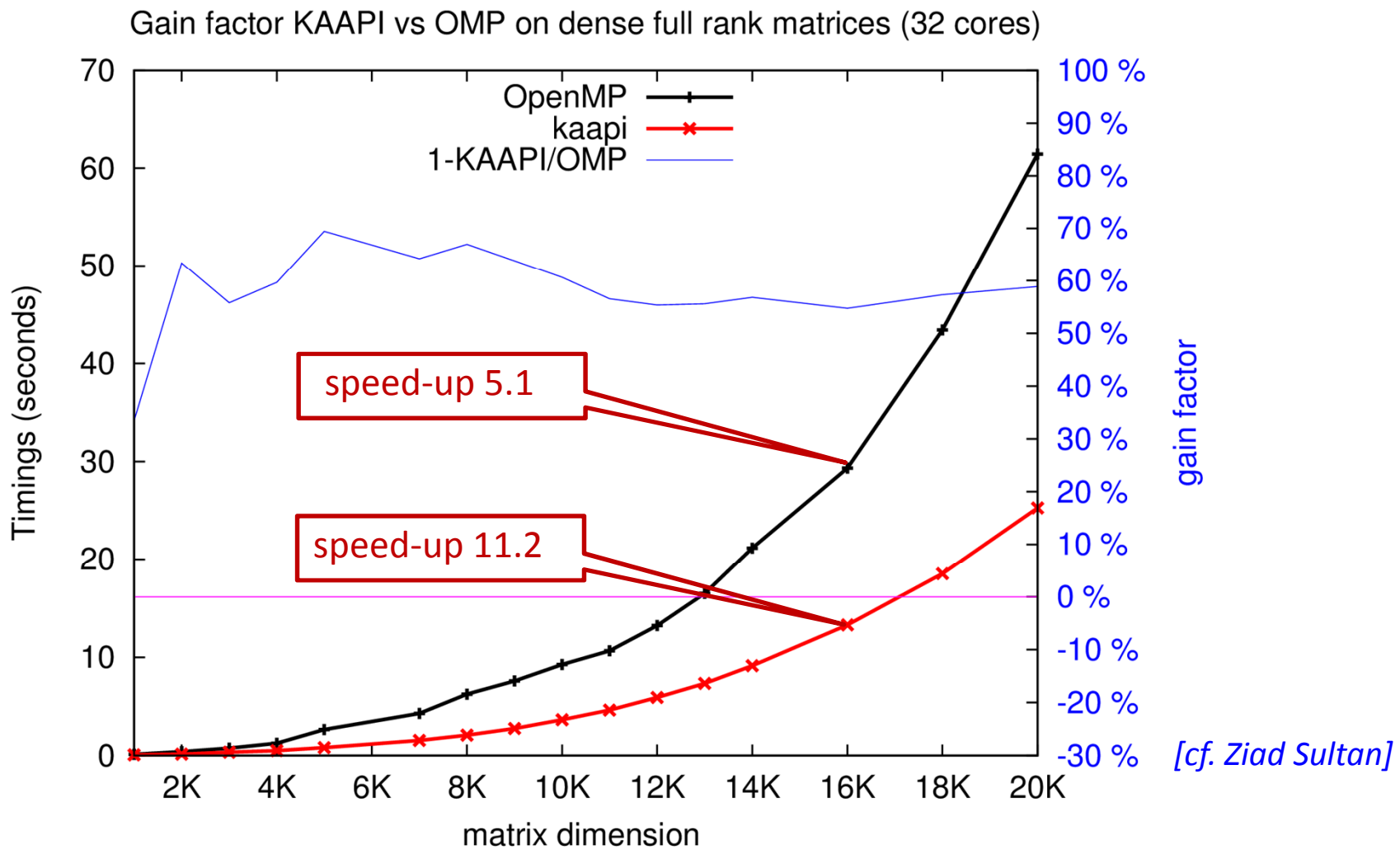


Specialized parallel kernels + Parallelism is a plug-in → portability, sustainability

Parallel loops are not sufficient



LUP: ex. 10000x10000 in 3s on 32 cores



[cf. Ziad Sultan]