Università della Svizzera italiana

Faculty of Informatics

Institute of Computational Science ICS

Platform for Advanced Scientific Computing

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Libraries for Scientific Computing

Patrick Sanan
Advanced Scientific Computing Group, USI Lugano

sananp@usi.ch

Including material from:

Dr. William Sawyer, Dr. Karl Rupp, Dr. Michael Heroux, Dr. Dimitar Lukarski, Prof. Stan Tomov, Dr. Peter Messmer

Pros

- Don’t reinvent (and reimplement) the wheel
  - Leverage the work of experts
- Focus on science
  - Experiment quickly
  - Avoid “lock in” from choices of data structures and algorithms
- Open source / community codes
  - Contribute to community projects
  - Code longevity and dissemination
- Use of algorithms and implementations which you don’t have deep knowledge of!

Cons

“Hell is other people’s code”

- learning curves
- versioning, changing APIs
- dependencies/building
- syntax, domain-specific languages, design choices
- Lack of perfect documentation (or local experts)
- Sales pitches, vaporware
- Use of algorithms and implementations which you don’t have deep knowledge of!

Disclaimers

- This presentation does not cover all domains for which useful libraries exist. In particular, the following are not covered extensively:
  - Optimization libraries
  - Meshing and graph partitioning libraries
  - I/O libraries
  - Communication Libraries
- Neither are all packages within a given domain represented.
- The hope is that you will be exposed to a subset of available tools related to linear algebra
- We will focus on PETSc in the next lecture as an example of a useful and widely-used library
- We set the threshold for “high level” here to be “just below domain-specific”.
  - Thus, we will note a multiphysics package like MOOSE, but not, say, an MD code like LAMMPS or GROMACS

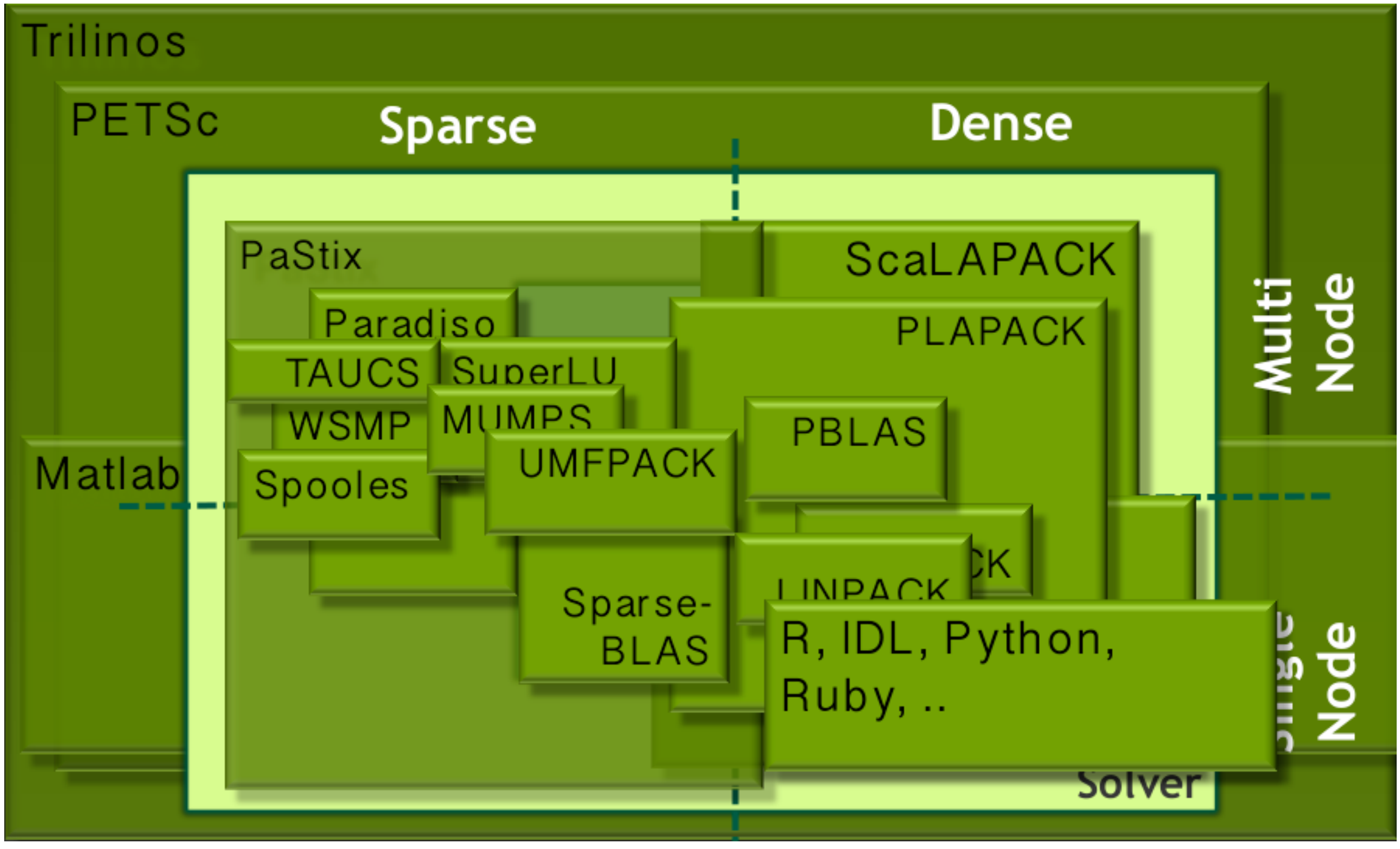
Linear Algebra

Many libraries, providing some or all of the following

- Dense matrix and vector arithmetic
- Matrix Factorization
- Eigenanalysis
- Sparse matrix operations
- Iterative linear solvers and preconditioners

In some our all of the following parallel environments

- Distributed memory (e.g. MPI)
- Shared memory (e.g. OpenMP)
- Accelerators/Coprocessors/Throughput-oriented devices (e.g. OpenCL)


Trilinos
PETSc
Sparse
Dense
PaStix
ScaLAPACK
Paradiso
PLAPACK
TAUCS
SuperLU
WSMP
MUMPS
PBLAS
Matlab
Spooles
UMFPACK
Sparse-
BLAS
R, IDL, Python,
Ruby, ..
Solver
Multi
Node
Single
Node

Dense Linear Algebra

You almost certainly use these libraries already!
(Except with the mini-app)

Some typical operations:

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^TQ = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization: $A = P^TLU \qquad P^TP = I$
- Forward/back-substitution: $Ax = y \Rightarrow LUx = y \Rightarrow w = L^{-1}y \Rightarrow x = R^{-1}w$
- Eigenvalue decomposition: $Ax = \lambda x \Rightarrow A = QDQ^T$
- Generalized eigen-problem: $Ax = \lambda Bx$
- Singular value decomposition: $A = U\Sigma V^T \quad U^TU = I \quad V^TV = I$

BLAS/LAPACK

Many, many, implementations

- Many of the libraries here implement these functions
- The difference between implementations usually comes down to what architecture(s) the implementation is optimized for
- At CSCS at the moment
  - Intel (MKL) - available with PrgEnv-intel
  - With the Cray/PGI/GNU environments, Cray's libsci contains heavily optimized implementations of BLAS, LAPACK, and SCALAPACK (distributed memory operations), amongst others

Elemental

- Modern C++ (C++11) linear algebra library
- libelemental.org
- Depends on BLAS/LAPACK, MPI
- Includes libFlame
  - Multithreaded dense linear algebra package
  - Independent of LAPACK

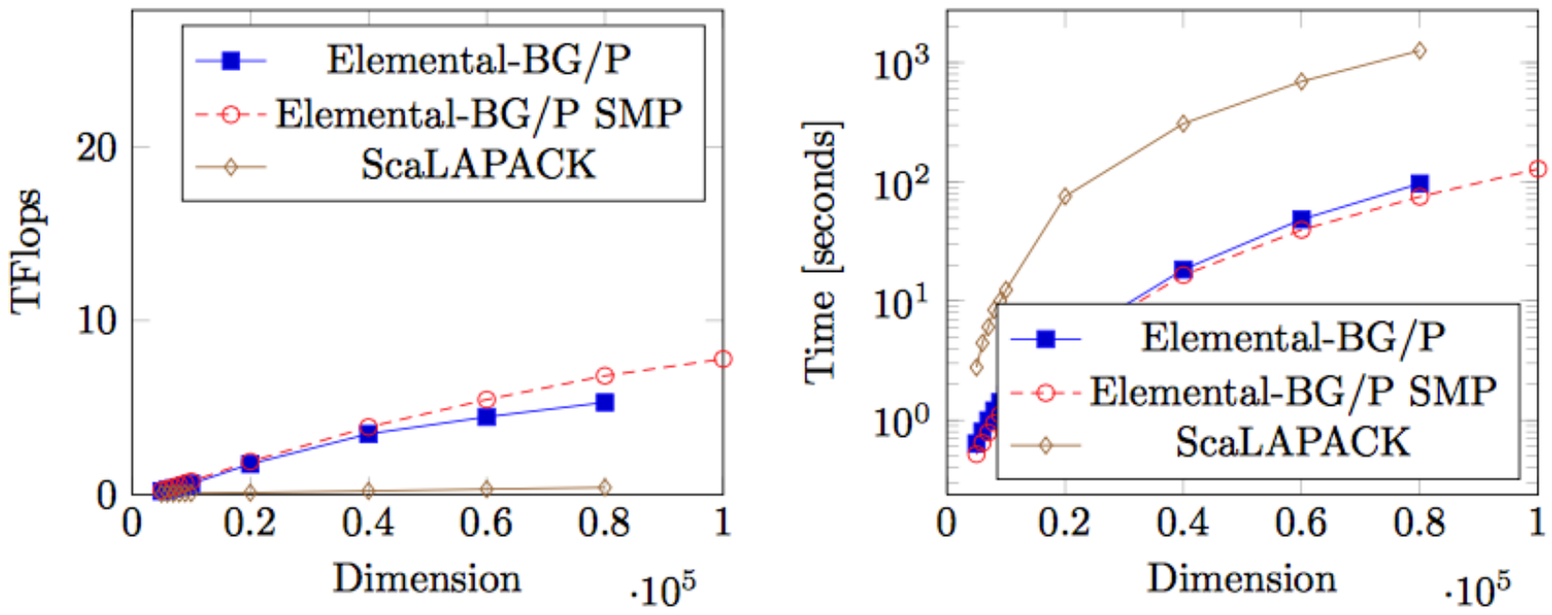


Figure 5: Real double-precision reduction of generalized eigenvalue problem to symmetric standard form on 8192 cores.

J. Poulson, B. Marker, R. A. v. d. Geijn, J. R. Hammond and N. A. Romero (2013). *Elemental: A new framework for distributed memory dense matrix computations*, ACM Transactions on Mathematical Software, 39(2), pp. 13:1-13:24, [doi: 10.1145/2427023.2427030].

Eigen

- eigen.tuxfamily.org
- A template-only C++ linear algebra library
  - Efficient, nice syntax, difficult to debug
- Interfaces with ViennaCL and many other libraries discussed here!

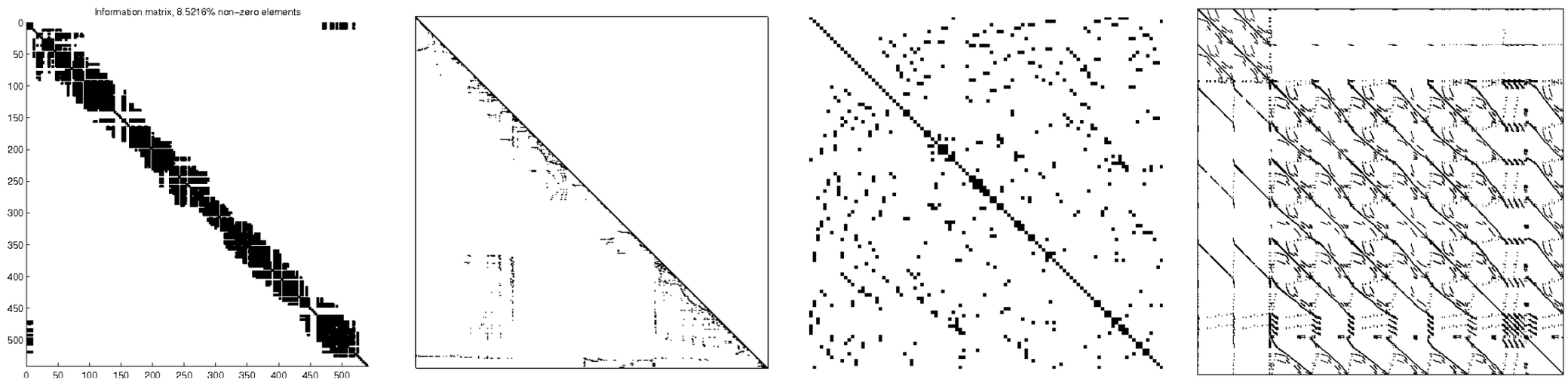
```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
int main()
{
  Matrix2d mat;
  mat << 1, 2,
         3, 4;
  Vector2d u(-1,1), v(2,0);
  std::cout << "Here is mat*mat:\n" << mat*mat << std::endl;
  std::cout << "Here is mat*u:\n" << mat*u << std::endl;
  std::cout << "Here is u^T*mat:\n" << u.transpose()*mat << std::endl;
  std::cout << "Here is u^T*v:\n" << u.transpose()*v << std::endl;
  std::cout << "Here is u*v^T:\n" << u*v.transpose() << std::endl;
  std::cout << "Let's multiply mat by itself" << std::endl;
  mat = mat*mat;
  std::cout << "Now mat is mat:\n" << mat << std::endl;
}
```

http://eigen.tuxfamily.org/dox/group__TutorialMatrixArithmetic.html

Sparse Linear Algebra

- Operations involving large sparse matrices often account for a large percentage of the compute time for large PDE problems
- Sparse matrix operations are much harder to vectorize than dense ones
- Sparse operations are almost always limited by memory bandwidth



math.nist.gov/MatrixMarket

Sparse Direct Solvers

$$A = LU, A = LDL^T$$

- Factor a matrix into a product of easy-to-solve factors
- Applicable to a wide range of matrices, often as a practical “black box”
- Suboptimal scaling and entry-dependent factorization time and storage
- Challenging to parallelize
- For arbitrarily-large systems, eventually beaten by optimally-scaling methods (iterative and/or multilevel solvers)

MUMPS

- http://mumps.enseeiht.fr
- Distributed multifrontal direct solver
- Fortran 90, with OpenMP directives for shared-memory parallelism

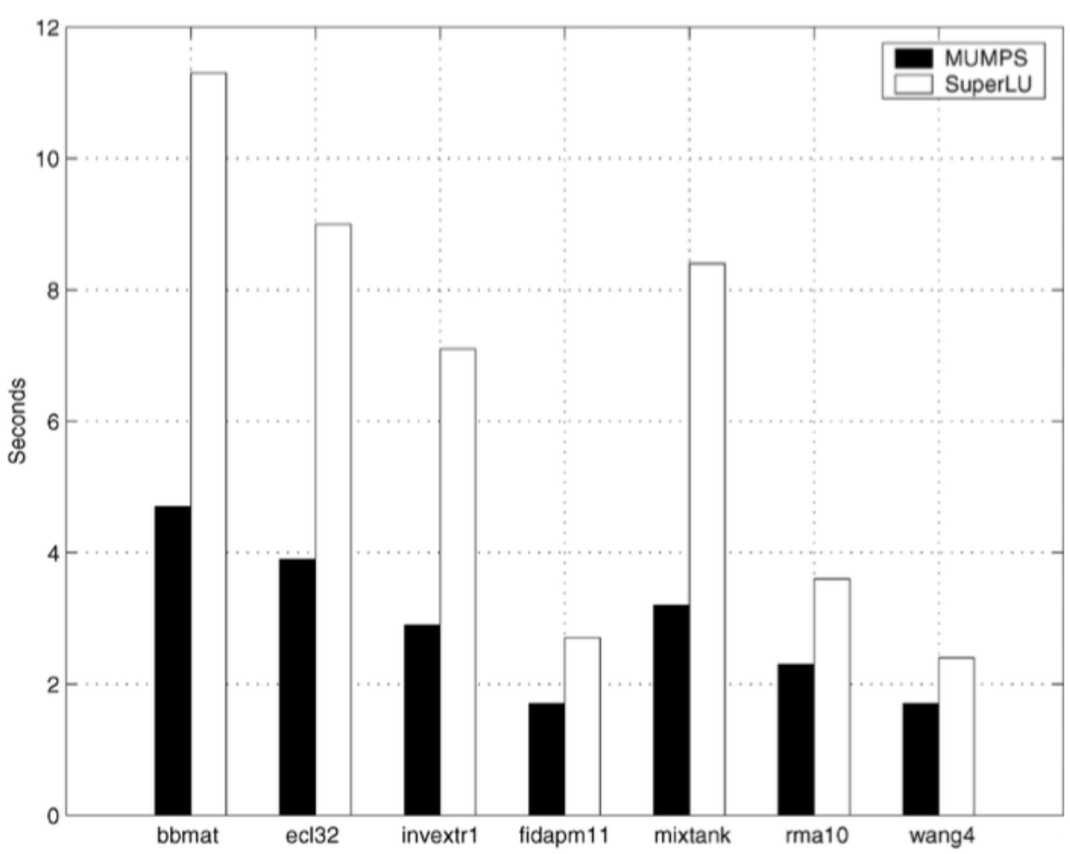


Fig. 5. Time comparison of the analysis phases of MUMPS and SuperLU. MC64 preprocessing is *not* used and AMD ordering *is* used.

SuperLU

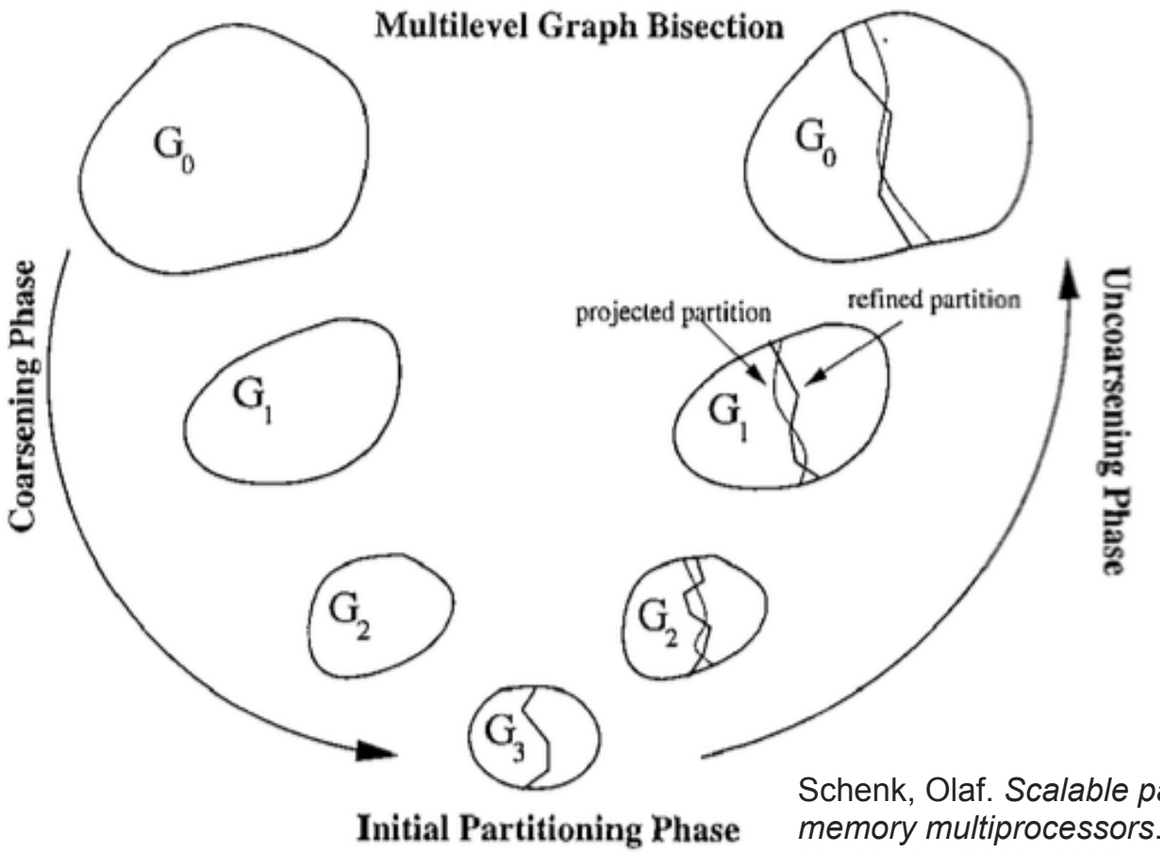
- http://acts.nersc.gov/superlu/
- C

- Both popular libraries
- Tradeoff analysis is complicated by the fact that the factorization may be more robust, due to a slower pivoting procedure

Amestoy, Patrick R., et al. "Analysis and comparison of two general sparse solvers for distributed memory computers." *ACM Transactions on Mathematical Software (TOMS)* 27.4 (2001): 388-421.)

PARDISO

- pardiso-project.org
- Fortran, using OpenMP and MPI
- Included in the Intel MKL, but be aware that this version does not contain recent additions to the library!
- Sparse direct solvers, along with related algorithms such as efficent evaluation of Schur complements
- By Olaf Schenk, my advisor at USI!



Schenk, Olaf. *Scalable parallel sparse LU factorization methods on shared memory multiprocessors*. Diss. Diss. Technische Wissenschaften ETH Zürich, Nr. 13515, 2000, 2000.

Iterative Solvers

A Krylov method (like Conjugate gradients) with a multilevel preconditioner is, currently, the only known O(N) algorithm for an important class of discretized PDEs.

1: **function** PCG(A, M^{-1}, b, x_0)
2: $r_0 \leftarrow b - Ax_0$
3: $u_0 \leftarrow M^{-1}r_0$
4: $p_0 \leftarrow u_0$
5: $s_0 \leftarrow Ap_0$
6: $\gamma_0 \leftarrow \langle u_0, r_0 \rangle$
7: $\eta_0 \leftarrow \langle s_0, p_0 \rangle$
8: $\alpha_0 \leftarrow \gamma_0 / \eta_0$
9: **for** $i = 1, 2, \ldots$ **do**
10: $x_i \leftarrow x_{i-1} + \alpha_{i-1} p_{i-1}$
11: $r_i \leftarrow r_{i-1} - \alpha_{i-1} s_{i-1}$
12: $u_i \leftarrow B(r_i)$
13: $\gamma_i \leftarrow \langle u_i, r_i \rangle$
14: $\beta_i \leftarrow \gamma_i / \gamma_{i-1}$
15: $p_i \leftarrow u_i + \beta_i p_{i-1}$
16: $s_i \leftarrow Ap_i$
17: $\eta_i \leftarrow \langle s_i, p_i \rangle$
18: $\alpha_i \leftarrow \gamma_i / \eta_i$

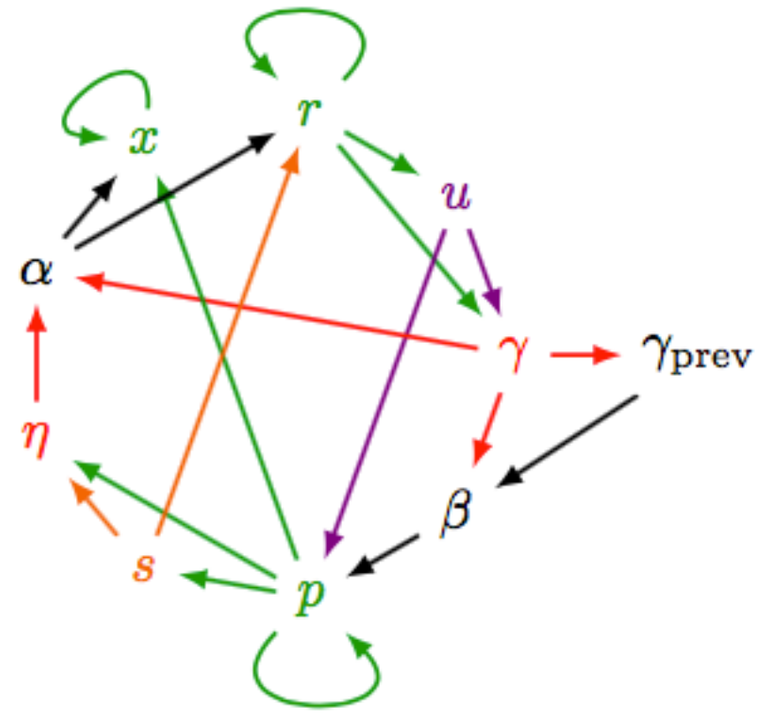
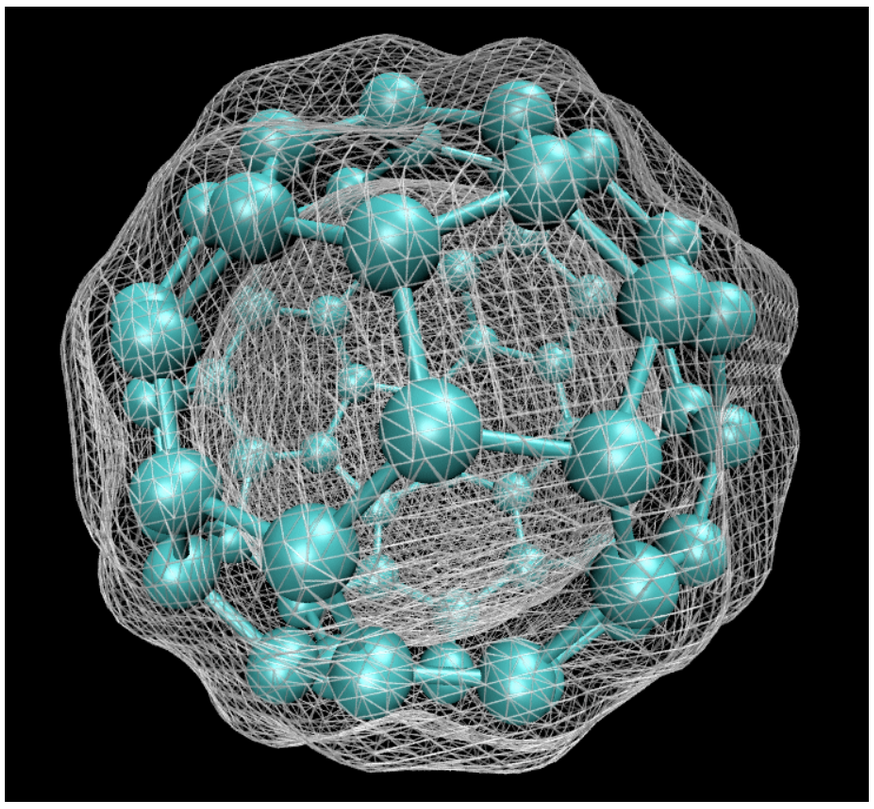


Figure 2: Schematic of the main loop of the preconditioned Conjugate Gradient (PCG) method

Eigensolvers

The bottleneck in many physics computations, computing large numbers of eigenvalues of large systems is a daunting computational task

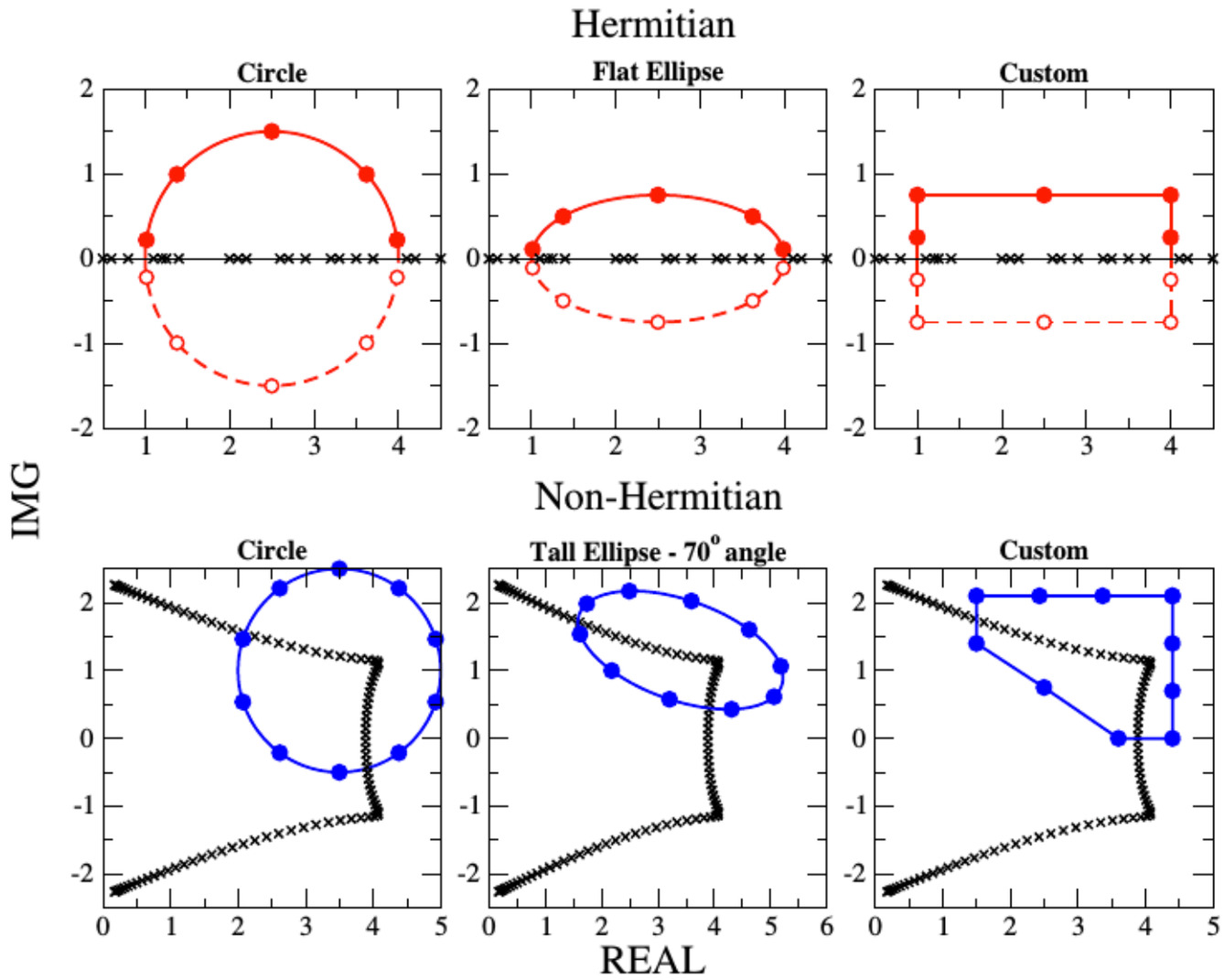
$$Ax = \lambda x$$



https://en.wikipedia.org/wiki/Density_functional_theory

FEAST

- http://www.ecs.umass.edu/~polizzi/feast/
- Uses the FEAST algorithm, based on contour integration
- Implemented for shared- and distributed-memory environments



http://www.ecs.umass.edu/~polizzi/feast/doc.htm

SLEPc

- Library for eigenanalysis of large, sparse, distributed linear systems
- slepc.upv.es
- Built very closely on top of PETSc
- Well-documented
- Sophisticated algorithms

Problem class	Model equation	Module
Linear eigenvalue problem	$Ax = \lambda x, \quad Ax = \lambda Bx$	`EPS`
Quadratic eigenvalue problem	$(K + \lambda C + \lambda^2 M)x = 0$	–
Polynomial eigenvalue problem	$(A_0 + \lambda A_1 + \cdots + \lambda^d A_d)x = 0$	`PEP`
Nonlinear eigenvalue problem	$T(\lambda)x = 0$	`NEP`
Singular value decomposition	$Av = \sigma u$	`SVD`
Matrix function (action of)	$y = f(A)v$	`MFN`

J. E. Roman, C. Campos, E. Romero and A. Tomas. *SLEPc Users Manual.* Tech. Rep. DSIC-II/24/02 - Revision 3.6, Universitat Politècnica de València, 2015.

GPU-enabled Linear Algebra Libraries

An active area of research!

There is promise (see the upcoming MAGMA plots), but this is still an area to approach with some caution.

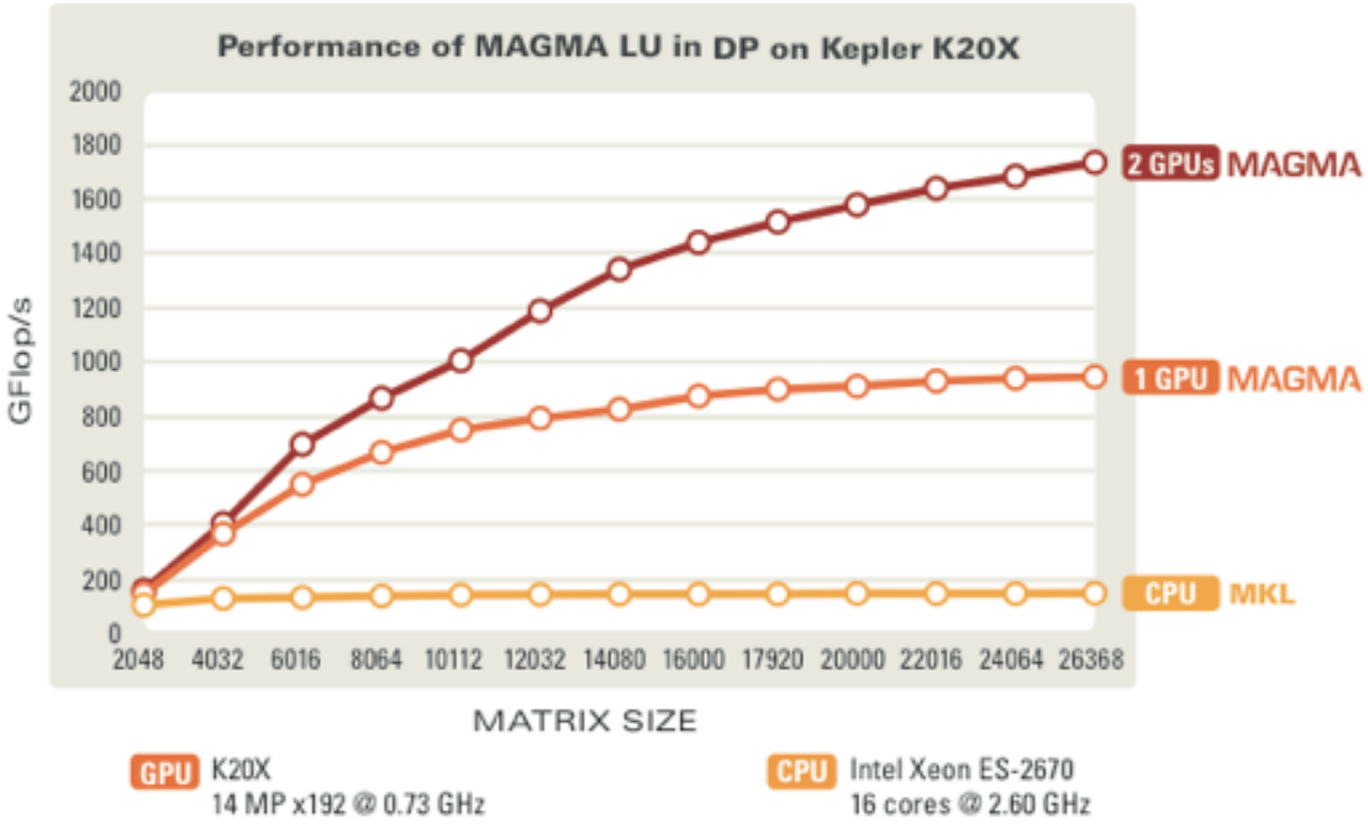
See last year's course for a full presentation on the then state-of-the-art, from Will Sawyer
http://github.com/fomics/SummerSchool2014

MAGMA

HYBRID ALGORITHMS

MAGMA uses a hybridization methodology where algorithms of interest are split into tasks of varying granularity and their execution scheduled over the available hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level 3 BLAS, are scheduled on the GPU.

PERFORMANCE

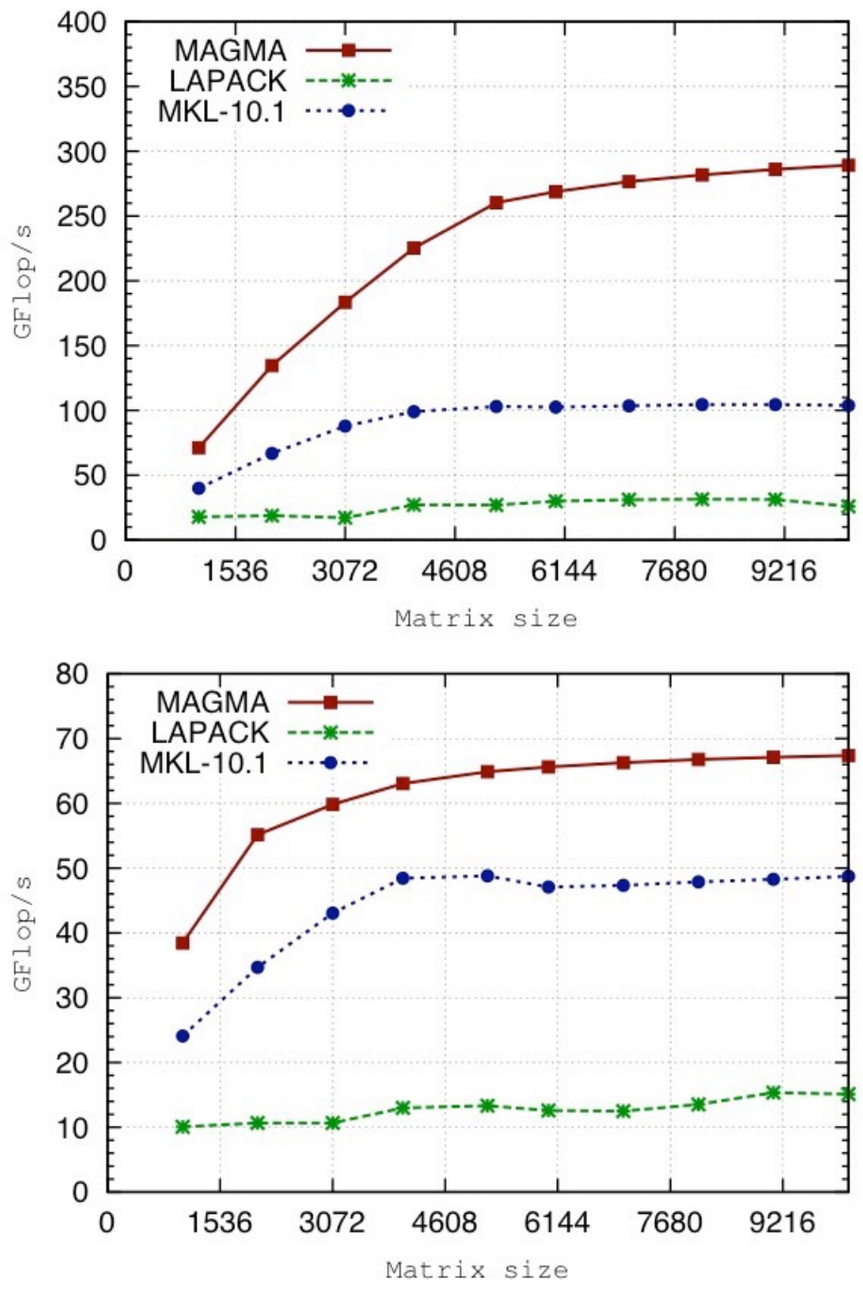


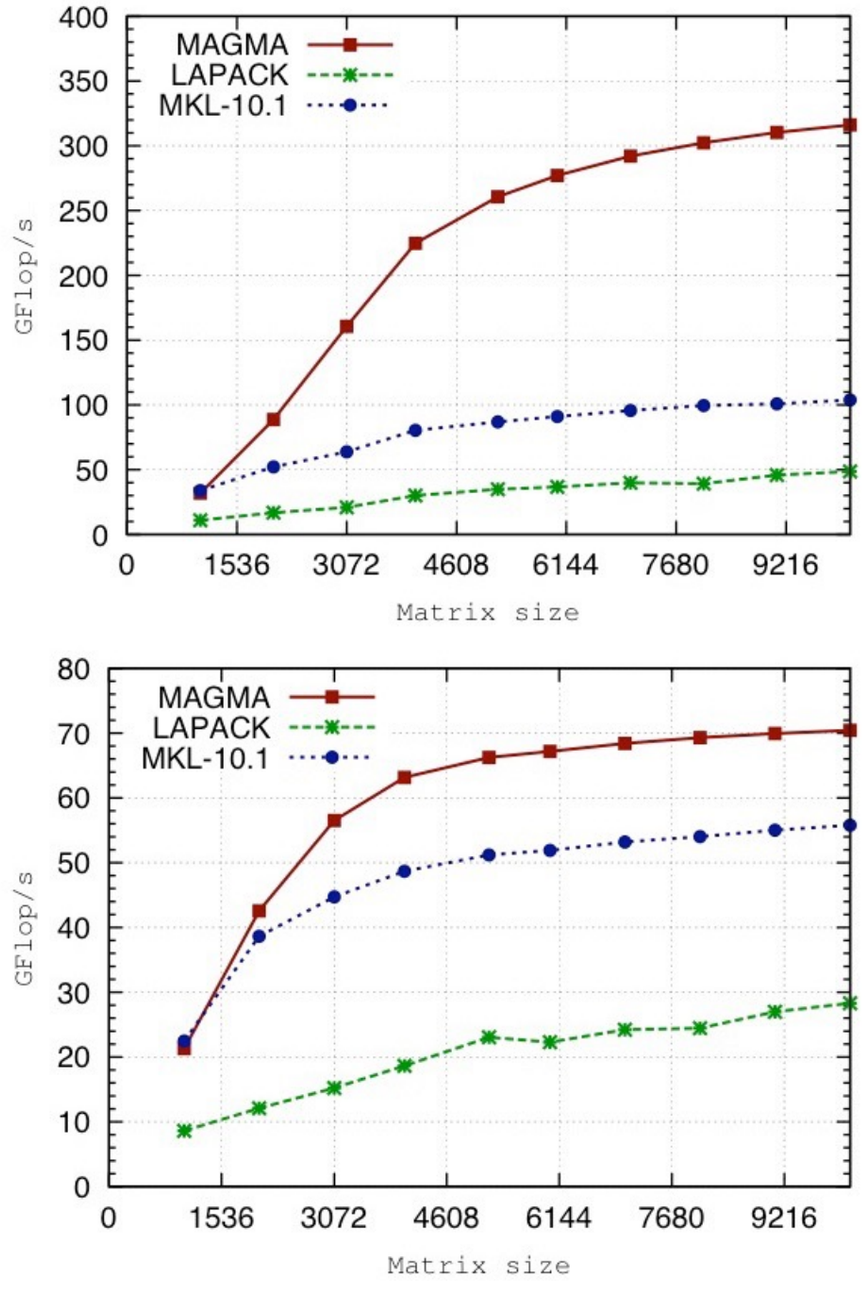
FEATURES AND SUPPORT

- **MAGMA 1.3** FOR **CUDA**
- **clMAGMA 1.0** FOR **OpenCL**
- **MAGMA MIC 0.3** FOR **Intel Xeon Phi**

CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●		Eigenvalue problem solvers
●			MAGMA BLAS
●			CPU Interface
●	●	●	GPU Interface
●	●	●	Multiple precision support
●			Non-GPU-resident factorizations
●			Multicore and multi-GPU support
●			Tile factorizations with StarPU dynamic scheduling
●	●	●	LAPACK testing
●	●	●	Linux
●			Windows
●			Mac OS

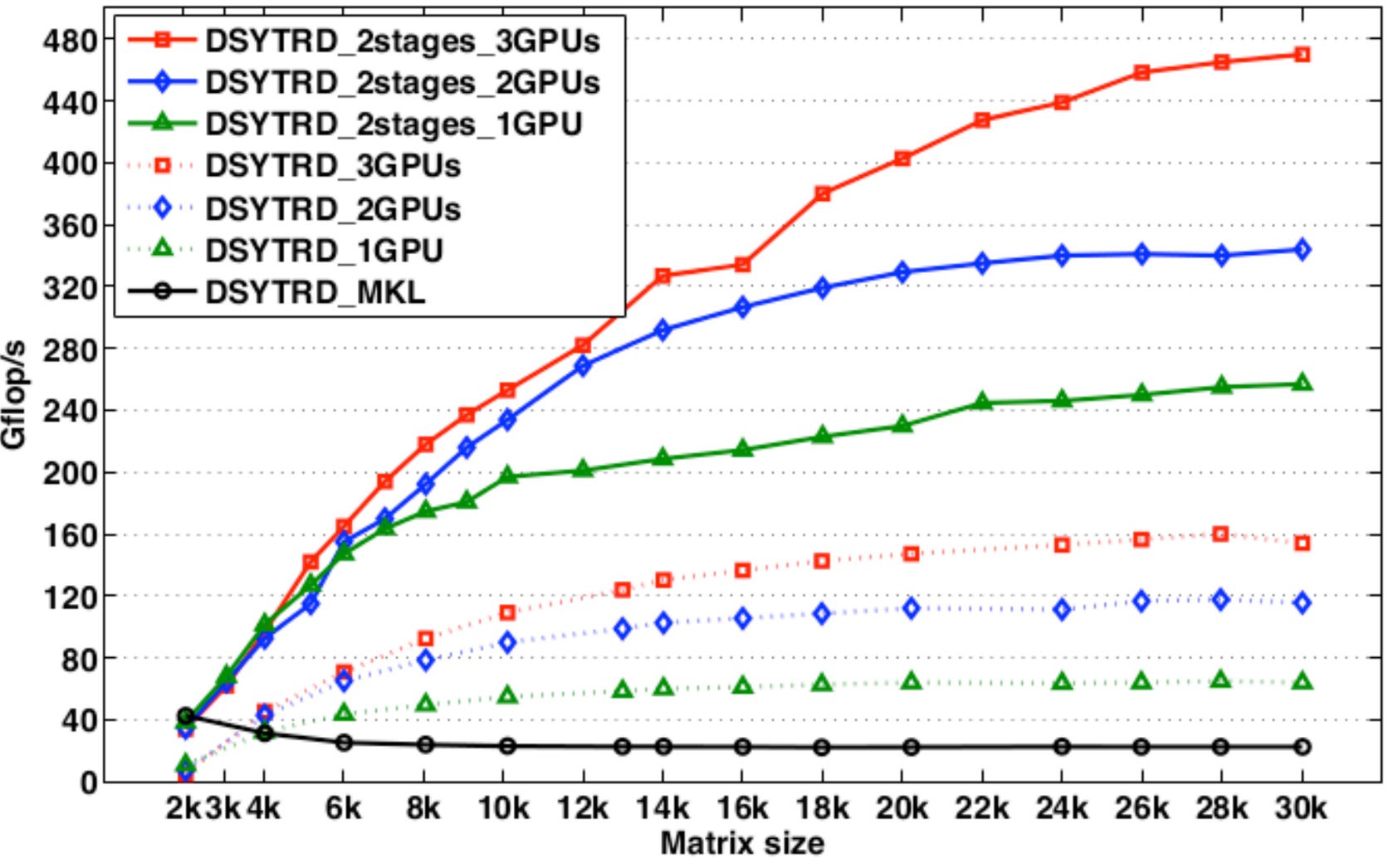
MAGMA Performance





MAGMA on GTX280 vs. Xeon quad core Left: QR decomp. SP/DP Right: LU decomp. SP/DP

MAGMA Gen EVP $Ax = \lambda Bx$



A. Haidar, S. Tomov, J. Dongarra, T. Schulthess, and R. Solca, *A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks*, ICL Technical report, 03/2012.

MKL Implementation

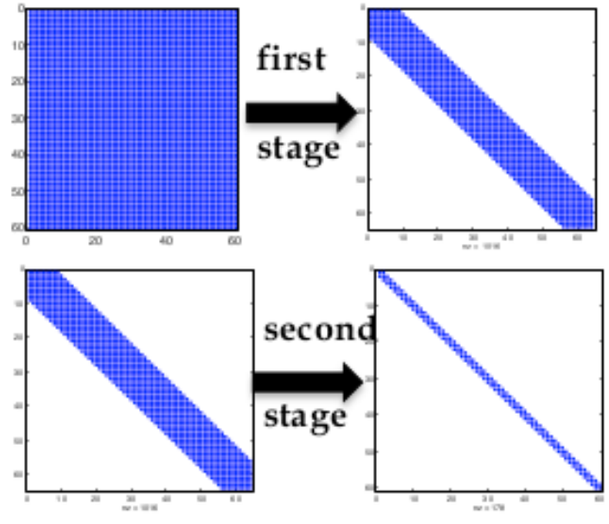
- Too many Blas-2 op,
- Relies on panel factorization,
- →Bulk sync phases,
- →Memory bound algorithm.

GPU 1-Stage

- Blas-2 GEMV moved to the GPU,
- Accelerate the algorithm by doing all BLAS-3 on GPU,
- →Bulk sync phases,
- →Memory bound algorithm.

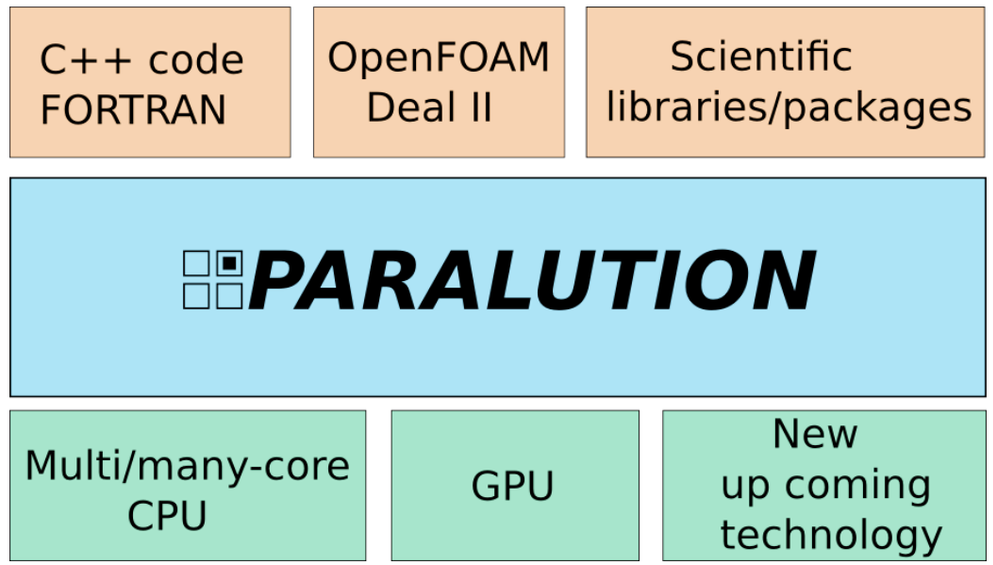
GPU 2-Stage

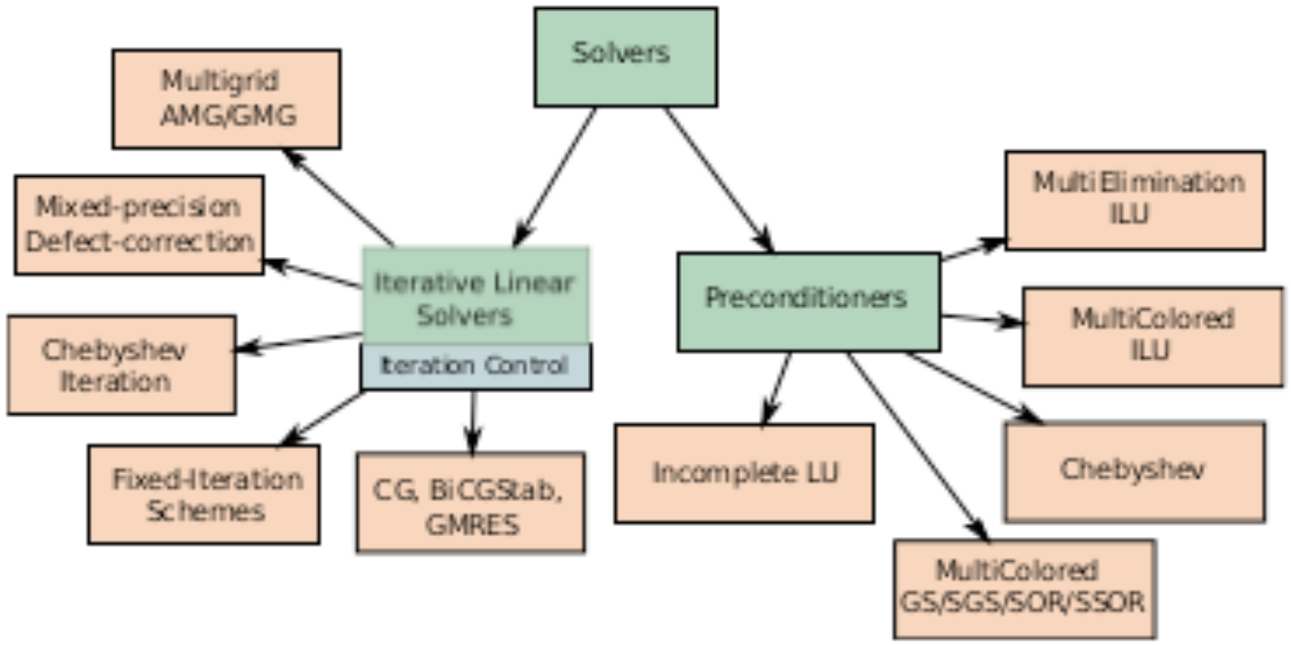
- **Stage 1:** BLAS-3, increasing computational intensity,
- **Stage 2:** BLAS-1.5, new cache friendly kernel,
- 4X/12X faster than standard approach,
- Bottelneck: if all Eigenvectors are required, it has 1 back transformation extra cost.



Paralution

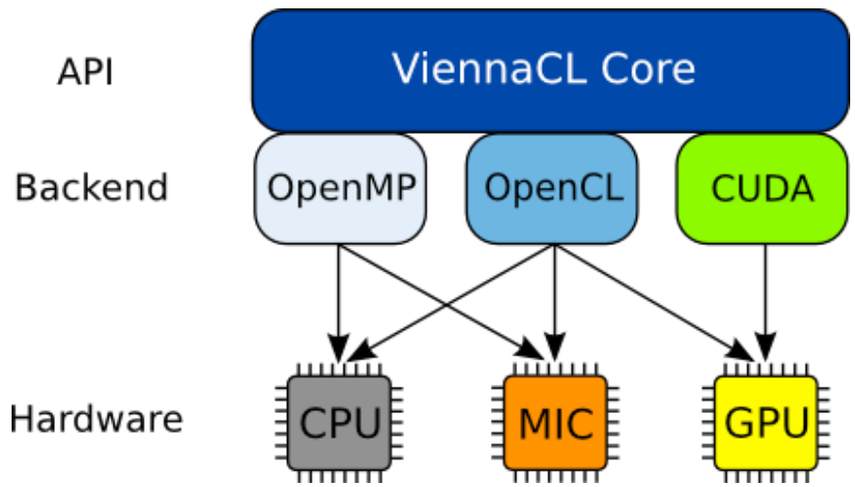
- *Sparse Iterative solvers & preconditioners*
- *Targeted: CPUs + accelerators*
- *Hardware abstraction*
- *OpenMP/CUDA/OpenMP opaque to user*
- *Code portable*
- *GPL v3*
- *http://www.paralution.com*



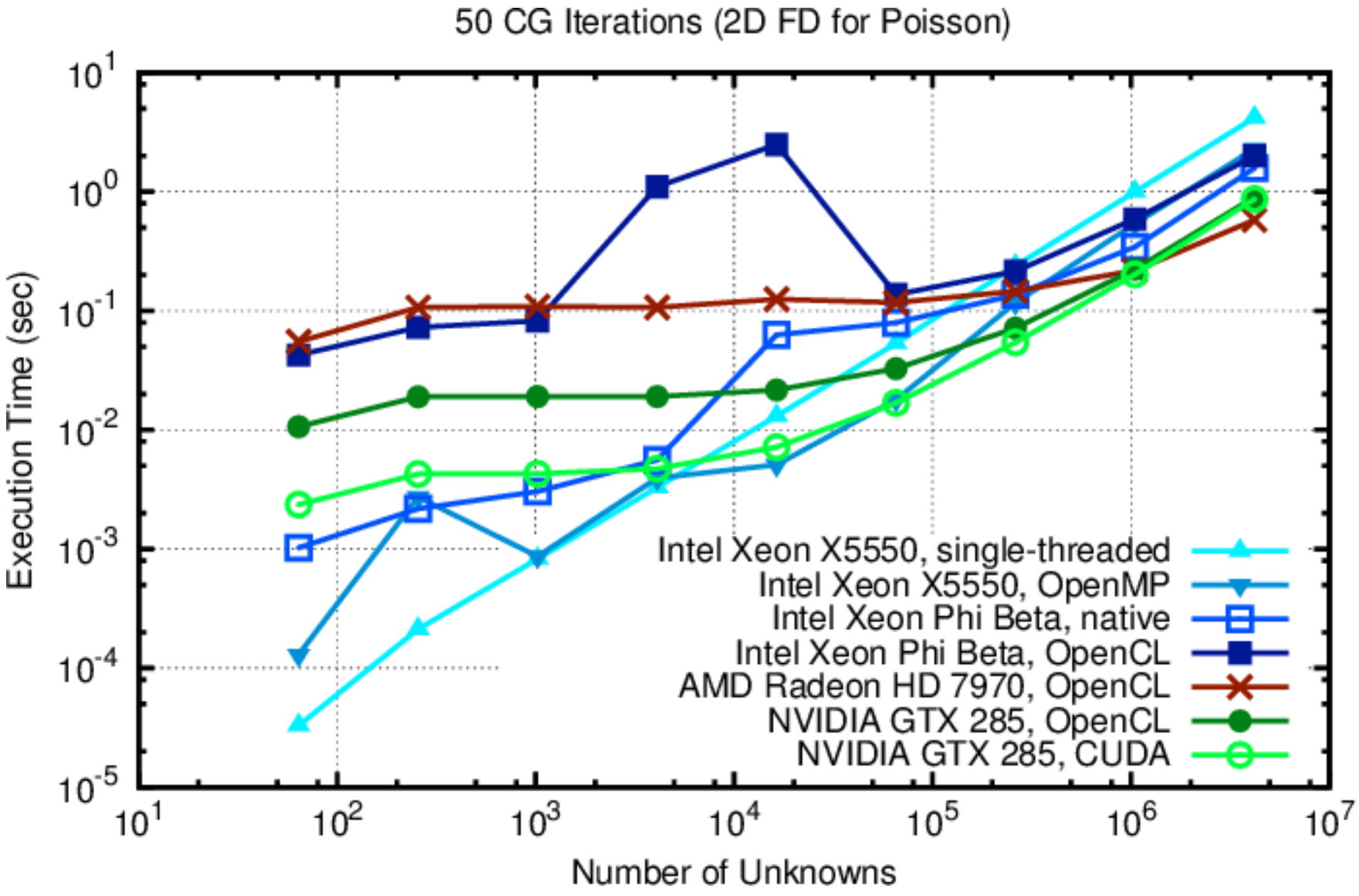


ViennaCL

- C++ Linear algebra library for many core architectures (GPUs, CPUs, Intel Xeon Phi)
- Supports BLAS 1-3
- Iterative solvers
- Sparse row matrix-vector multiplication, solvers
- Goals:
  - ➡ Simplicity, minimal dependencies
  - ➡ Compatible with Boost.uBLAS, Eigen,...
  - ➡ Open source, header-only library



viennacl.sourceforge.net

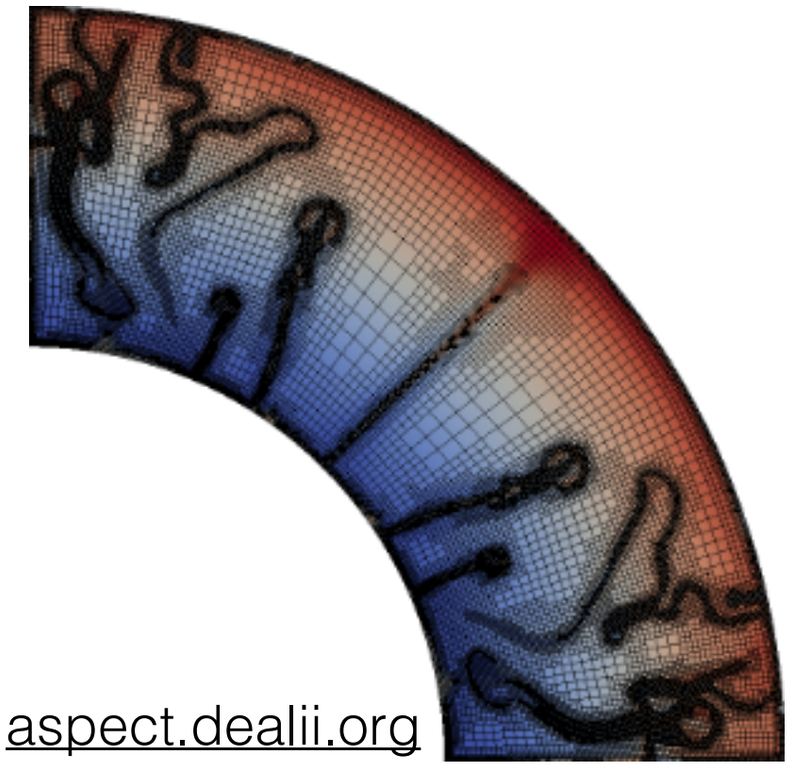


http://viennacl.sourceforge.net/viennacl-benchmarks.html

Finite Element Libraries

Libraries are increasingly allowing for flexibility at higher and higher levels of abstraction, and can benefit from tight coupling between discretizations and linear algebra / solver software

- Fenics (fenicsproject.org)
- Firedrake (firedrakeproject.org)
- Deal.II (dealii.org)

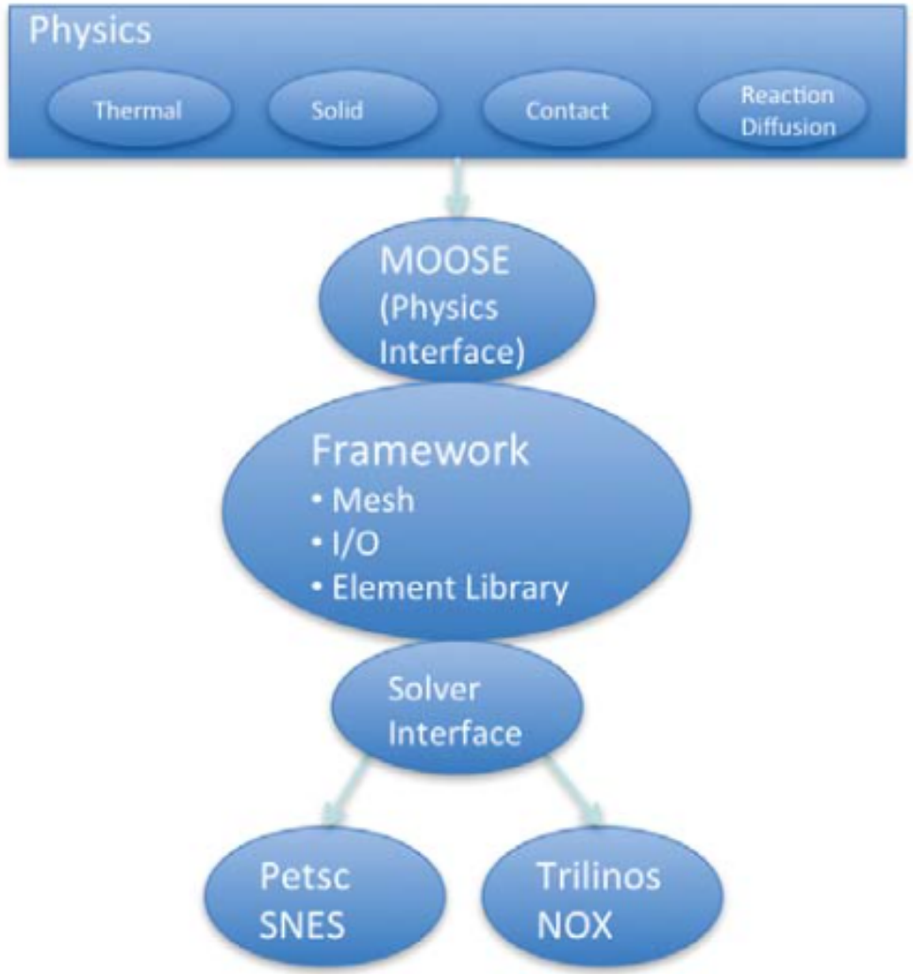


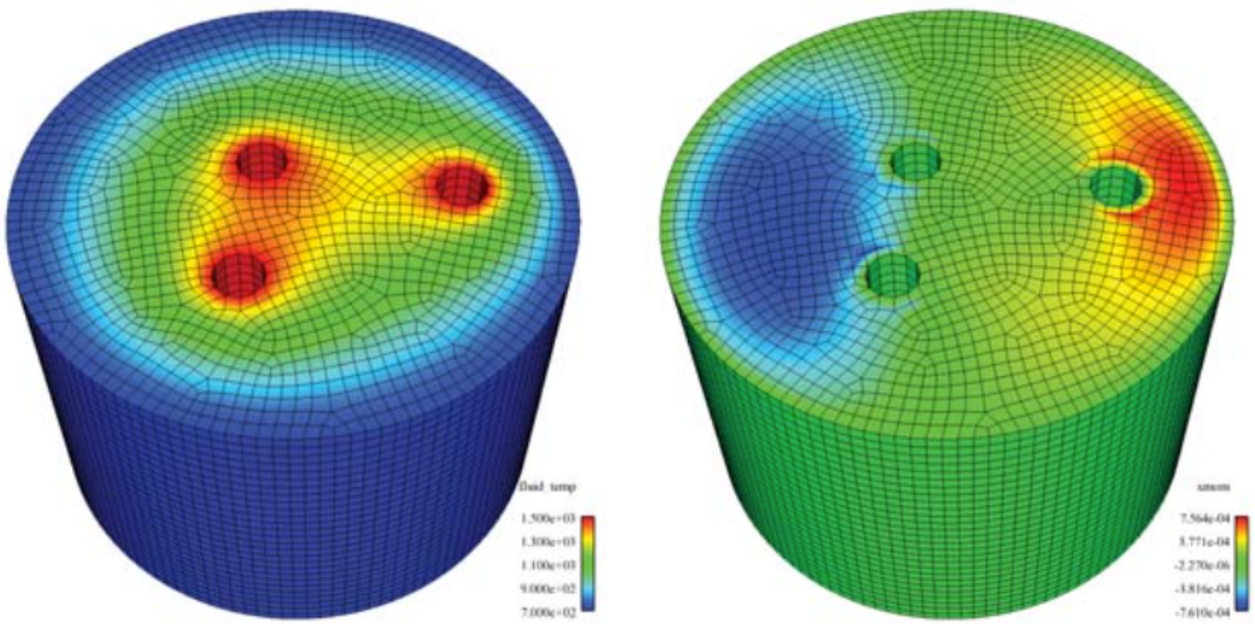
aspect.dealii.org

These libraries all offer excellent documentation and high-level interfaces in the case of Fenics and Firedrake

MOOSE

- mooseframework.org
- Framework for **multiphysics** problems
  - Developed for reactor simulation
- On top of PETSc





(a) Fluid Temperature

(b) x Momentum

MOOSE: A parallel computational framework for coupled systems of nonlinear equations. Nuclear Engineering and Design, 239 (10)(October 2009), pp. 1768-1778.
D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandie

Trilinos

- A suite of libraries:
  - Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
  - Preconditioners: *AztecOO, Ifpack2, ML, Meros*
  - Iterative linear solvers: *AztecOO, Belos*
  - Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*
  - Non-linear / optimization solvers: *NOX, MOOCHO*
  - Eigensolvers: *Anasazi*
  - Mesh generation / adaptivity: *Mesquite, PAMGEN*
  - Domain decomposition: *Claps*
  - Partitioning / load balance: *Isorropia, Zoltan2*
- trilinos.sandia.gov
- Offers a superset of the functionality of PETSc, but with less integration and uniformity between the packages

PETSc

Includes several of the functionalities discussed so far, and more:

- Sparse linear algebra
- Linear and nonlinear solves
- Timesteppers
- Domain management
- Optimization

The focus of the next tutorial!