Libraries for Scientific Computing

Patrick Sanan
patrick.sanan@[usi.ch,erdw.ethz.ch]

USI Lugano / ETH Zürich

Including and adapting material provided by
Dr. William Sawyer, Dr. Karl Rupp, Dr. Michael Heroux, Dr. Dimitar Lukarski, Prof. Stan Tomov, Dr. Peter Messmer

CSCS Summer School, July 27, 2017

This Tutorial

- Some high-level discussion of libraries for scientific computing
- Some more specific discussion of numerical libraries, focusing on GPU/MIC/accelerator/coprocessor-enabled libraries
- A quick tour of some available libraries
- We will aim to finish a bit early, and will begin the next tutorial. This will give some extra flexibility during the break if anyone has issues compiling the demo code.

What is a software library?

- A set of related functions to accomplish tasks required by more than one application
- Written and used by different people
- Relies on an application public interface (API)
- Typically versioned, documented, distributed, licensed

Libraries for Scientific Computing: Pros

```
#include <wheellib.h>
```

- Don't reinvent the wheel
- Don't reimplement the wheel
- Use the wheel even if you don't understand it or know how to optimize it!
- Leverage the work of experts
- Focus on your part of the "stack" to do science
- Experiment quickly
- Avoid "lock in" with respect to data structures and algorithms (maybe a wheel wasn't the right choice)
- Open source or community projects allow
  - Consolidation of efforts from many people
  - Continuity on time scales longer than projects/PhDs/grants/careers
  - Collaborative efforts good for science

Libraries for Scientific Computing: Cons

```
*** WHEELLIB ERROR ***
Fatal: Moving wheels only supported in Fortran 66.
To file a bug report, email wheellib4ever@aol.com
*********************
```

- Learning curves
- Versioning, changing APIs
- Bugs that someone else must fix
- Syntax, design choices
- Lack of documentation (or local experts)
- Oversold software, vaporware
- The scientific risks of using algorithms (or hardware) that you don't understand
- "Hell is other people['s code]"

Obtaining Libraries

In descending order of ease:

- System Installation (by default or by an admin)
- Cluster modules
- Package managers
- Binary Download
- Build system from another library
- Build from source

Disclaimers

- I will not cover all libraries available, even within the subfields I discuss here.
- The inclusion or exclusion of a library (including PETSc, used as the basis for the extended tutorial later) is as much a function of my familiarity with it than its absolute quality. I will have missed some important ones, so let me know for the next time this lecture is given!

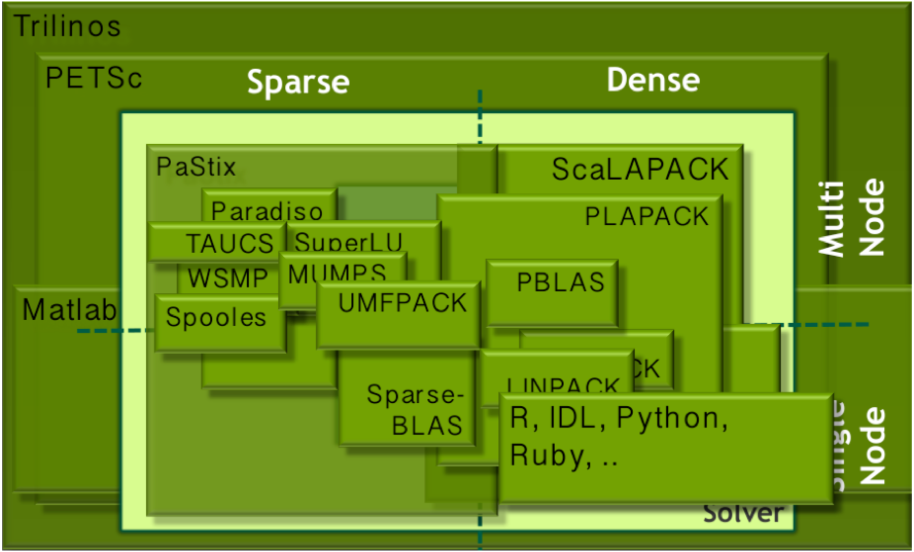
Section 1

Linear Algebra Libraries

Linear Algebra Libraries

- Operations involving vector spaces and linear operators are involved in most, if not all, scientific codes
- As such, for decades [1], libraries have existed to provide efficient, reusable abstractions and implementations to perform these operations
- Functionality includes
  - Vector operations
  - Dense matrix operations
  - Sparse matrix operations
  - Linear solvers: dense/sparse, exact/approximate, direct/iterative
  - Eigenanalysis
- As ubiquitous low-level operations that often form the majority of the computational effort, adaptation and optimization for different environments is important
  - Shared memory
  - Distributed memory
  - Accelerators / Coprocessors/ Throughput-oriented devices

[1] BLAS originated in 1979, for example


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

Dense Linear Algebra

- You almost certainly use these operations already
- You likely leverage (perhaps indirectly) libraries to do so
- Typical Operations include
  - Elementary elementwise operations on matrices and vectors : $A + B$,etc.
  - Norms, inner products, matrix-matrix multiplies, matrix-vector multiplies : $||x||_2, \langle x, y \rangle, AB, Ax$, etc.
  - Cholesky factorization: $A = LL^T$,L lower triangular
  - QR decomposition: $A = QR, Q^H Q = I$, R upper triangular
  - LU factorization: $A = P^T LU$, P permutation, L lower tri., R upper tri.
  - Triangular solves $y = L^{-1}x$
  - Eigenvalue decomposition : $Ax = \lambda x \iff A = Q \Lambda Q^T$, $Q^H Q = I$
  - Singular value decomposition $A = U \Sigma V^H$, $U^H H = I$, $V^H V = I$

BLAS and LAPACK

- Fundamental numerical libraries
- Many implementations, optimized for different architectures
- BLAS
  - vector operations (BLAS-1)
  - matrix-vector operations (BLAS-2)
  - matrix-matrix operations (BLAS-3)
- LAPACK
  - Matrix factorization and linear system solution
  - Least squares
- SCALAPACK : distributed memory LAPACK (includes BLACS as a communication layer)
- Available implementations at CSCS include the following.
  - Intel's math kernel library (MKL) includes BLAS and LAPACK, available with `PrgEnv-intel` [2]
  - Cray's `libsci` : heavily optimized BLAS, LAPACK, SCALAPACK within the Cray, PGI, and GNU environments.

[2] if you are an advanced MKL user and want the raw path, note that `MKLROOT` will be set in your environment

Elemental

- C++11 linear algebra library
- libelemental.org
- Depends on BLAS, LAPACK, and MPI
- Includes libFlame (Multithreaded dense linear algebra, independent of LAPACK)
- Competitive with other distributed-memory packages[3]

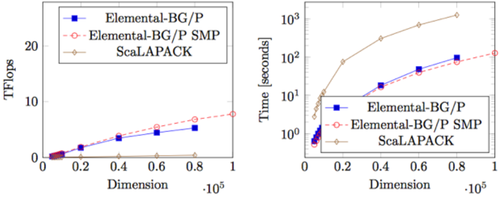


Figure 5: Real double-precision reduction of generalized eigenvalue problem to symmetric standard form on 8192 cores.

[3] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero. "Elemental: A New Framework for Distributed Memory Dense Matrix Computations". In: *ACM Trans. Math. Softw.* 39.2 (Feb. 2013), 13:1–13:24. DOI: 10.1145/2427023.2427030.

Eigen

- eigen.tuxfamily.org
- A template-only C++ dense and sparse linear algebra library (not distributed-memory)
  - Pro: efficent, natural syntax
  - Con: can be bewildering to debug, source code opaque
- Interfaces with ViennaCL, MKL, and other libraries discussed here

```
Matrix2d mat;
mat << 1, 2,
       3, 4;
Vector2d u(-1,1), v(2,0);
std::cout << "mat*mat:\n" << mat*mat << std::endl;
std::cout << "mat*u:\n" << mat*u << std::endl;
std::cout << "u^T*mat:\n" << u.transpose()*mat << std::endl;
std::cout << "u^T*v:\n" << u.transpose()*v << std::endl;
std::cout << "u*v^T:\n" << u*v.transpose() << std::endl;
```

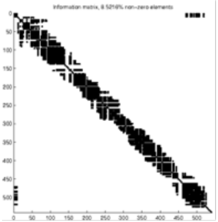
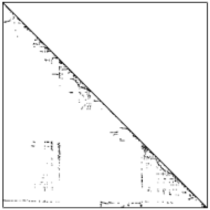
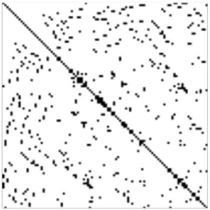
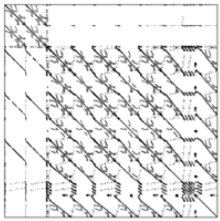
Taken from the matrix arithmetic tutorial on the Eigen website

Section 2

Sparse Linear Algebra

Sparse Linear Algebra

- Use cases: sparse PDE, big sparse data
- Fundamentally very different from dense linear algebra
  - Operations are difficult to vectorize
  - Typically limited by data movement (memory bandwidth), not floating-point performance
  - Any operator which can be applied (hence potentially inverted) in linear time must be sparse, and most sparse linear algebra libraries are aimed at large systems

http://math.nist.gov/MatrixMarket/

Sparse Direct Solvers

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

- Factor a matrix into a produce of easy-to-invert matrices
- Modern libraries apply to a wide range of matrices, approaching true "black box" status
- Efficient for repeated solves
- Suboptimal scaling and entry-dependent[4] factorization time and storage
- Challenging to parallelize
- For large-enough systems, eventually beaten by optimally-scaling methods (iterative and/or multilevel algorithms[5])

[4] and implementation-dependent

[5] Though those methods often involve sparse direct solvers, and multi-level sparse direct methods exist

Popular Distributed-Memory Sparse Direct Solver Packages

- MUMPS
- SuperLU
- PASTIX
- PARDISO (and MKL PARDISO)
- SuiteSparse UMFPACK
- SPOOLES
- WSMP

- Amongst optimized sparse direct solvers, there is typically a tradeoff between robustness (more elaborate pivoting and iterative refinement) and speed [6]

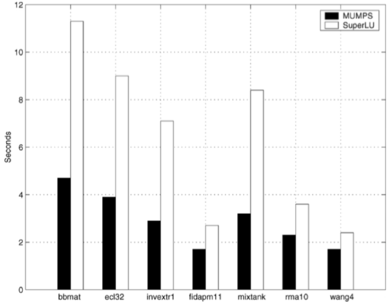
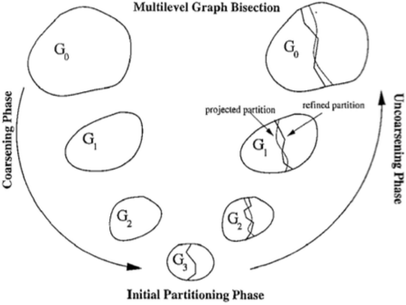


Fig. 5. Time comparison of the analysis phases of MUMPS and SuperLU. MC64 preprocessing is *not* used and AMD ordering *is* used.

[6] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'excellent, and Xiaoye S. Li. "Analysis and Comparison of Two General Sparse Solvers for Distributed Memory Computers". In: *ACM Trans. Math. Softw.* 27.4 (Dec. 2001), pp. 388–421. DOI: 10.1145/504210.504212

PARDISO and MKL PARDISO

- pardiso-project.org
- Fortran, using OpenMP and OpenMPI
- Sparse direct solvers and related algorithms such as efficient evaluation of Schur complements
- Developed locally by Olaf Schenk and his group at USI.



[7]

[7] Olaf Schenk, Klaus Gärtner, and Wolfgang Fichtner. "Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors". In: *BIT Numerical Mathematics* 40.1 (2000), pp. 158–176

Section 3

Iterative Solvers

Iterative Solvers

- Known for very long time (first methods with Gauss/Jacobi, CG invented in the 1950s)
- For an important class of discretized PDEs, Krylov methods and/or multilevel methods are the only know scalable ($O(N)$ time to solution) methods of solution

function PCG(A, M^{-1}, b, x_0)
  $r_0 \leftarrow b - Ax_0$
  $u_0 \leftarrow M^{-1} r_0$
  $p_0 \leftarrow u_0$
  $s_0 \leftarrow Ap_0$
  $\gamma_0 \leftarrow \langle u_0, r_0 \rangle$
  $\eta_0 \leftarrow \langle s_0, p_0 \rangle$
  $\alpha_0 \leftarrow \gamma_0 / \eta_0$
  for $i = 1, 2, \ldots$ **do**
    $x_i \leftarrow x_{i-1} + \alpha_{i-1} p_{i-1}$
    $r_i \leftarrow r_{i-1} - \alpha_{i-1} s_{i-1}$
    $u_i \leftarrow B(r_i)$
    $\gamma_i \leftarrow \langle u_i, r_i \rangle$
    $\beta_i \leftarrow \gamma_i / \gamma_{i-1}$
    $p_i \leftarrow u_i + \beta_i p_{i-1}$
    $s_i \leftarrow Ap_i$
    $\eta_i \leftarrow \langle s_i, p_i \rangle$
    $\alpha_i \leftarrow \gamma_i / \eta_i$

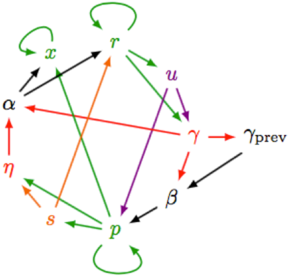


Figure 2: Schematic of the main loop of the preconditioned Conjugate Gradient (PCG) method, as described in Algorithm 3.

Iterative Solver Libraries

- Krylov methods can be written by hand with a lower-level library
- Some packages (Trilinos, PETSc,..) also include Krylov methods
- Multigrid methods [8], especially algebraic multigrid (AMG) methods, are rarely written by hand. Some AMG packages include
  - BoomerAMG (in Hypre)
  - ML
  - GAMG (in PETSc)
  - ILUPACK (multi-level ILU)

[8] To be pedantic, FMG can be described as a direct solver

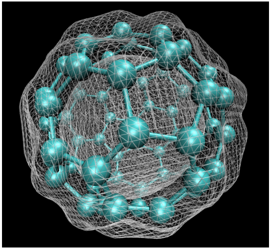
Section 4

Eigensolvers

Eigensolvers

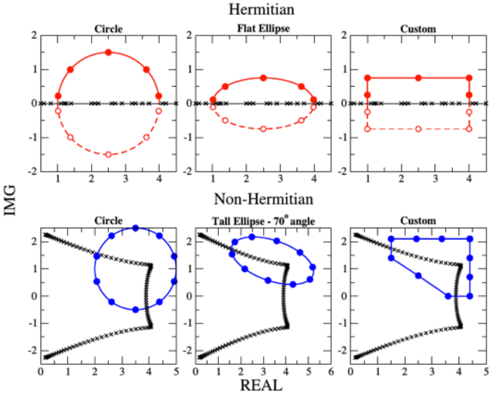
$$Ax = \lambda x$$

- The bottleneck in many physics computations is the computation of a large number of eigenvalues of a large system
- Specialized methods are required, beyond those available in the classic libraries discussed above



FEAST

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



From FEAST documentation

SLEPc

- slepc.upv.es
- Library for eigenanalysis of large, sparse, distributed linear systems
- 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

From SLEPc documentation

Section 5

GPU-enabled Linear Algebra Libraries

GPU-enabled Linear Algebra Libraries

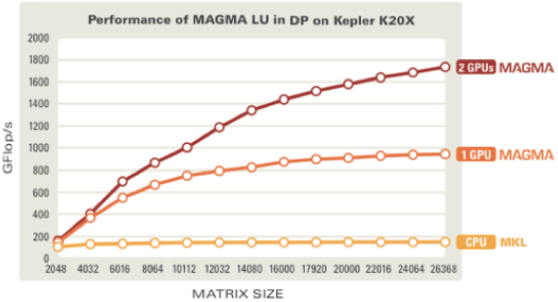
- Well-designed libraries are in high demand to remove the burden of porting common operations to GPU and other accelerators
- Transparent performance portability is very difficult
- For a overview of more then-current libraries, see the material from Will Sawyer from the 2014 Summer School.

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



GPU K20X
14 MP x192 @ 0.73 GHz

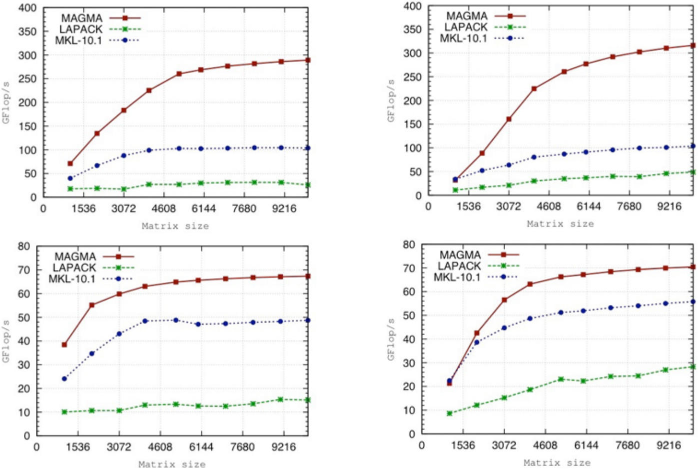
CPU Intel Xeon ES-2670
16 cores @ 2.60 GHz

FEATURES AND SUPPORT

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

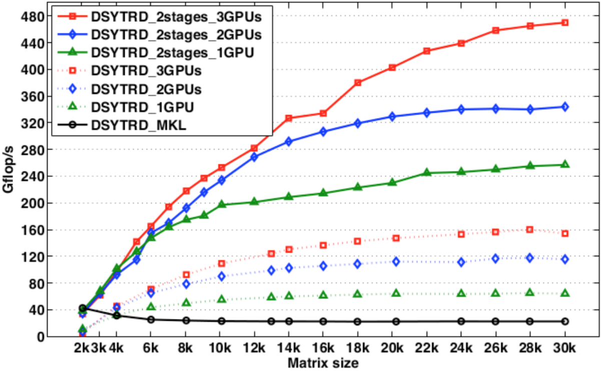
CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●		Eigenvalue problem solvers
●			MAGMA BLAS
●			CPU Interface
●	●	●	GPU Interface
●	●	●	Multiple precision support
●			Non-GPU-resident factorizations
●			Multicore and multi-GPU support
●			Tile factorizations with StarPU dynamic scheduling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



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

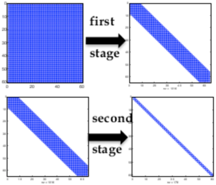
- Too many Blas-2 op,
- Relies on panel factorization,
- →Bulk sync phases,
- →Memory bound algorithm.

GPU 1-Stage

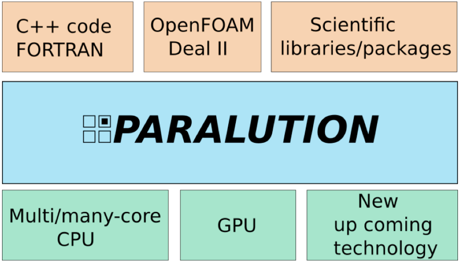
- Blas-2 GEMV moved to the GPU,
- Accelerate the algorithm by doing all BLAS-3 on GPU,
- →Bulk sync phases,
- →Memory bound algorithm.

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.

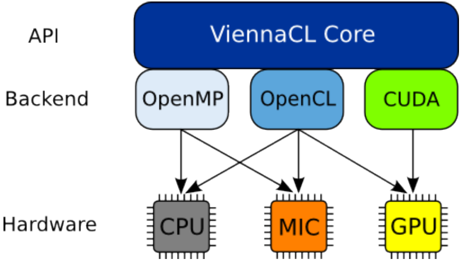


- paralution.com
- Sparse iterative solvers and (numerous) preconditioners
- Targeted: CPUs + accelerators
- Hardware abstraction
- OpenMP/CUDA/OpenMP opaque to user
- Code portable
- GPL v3

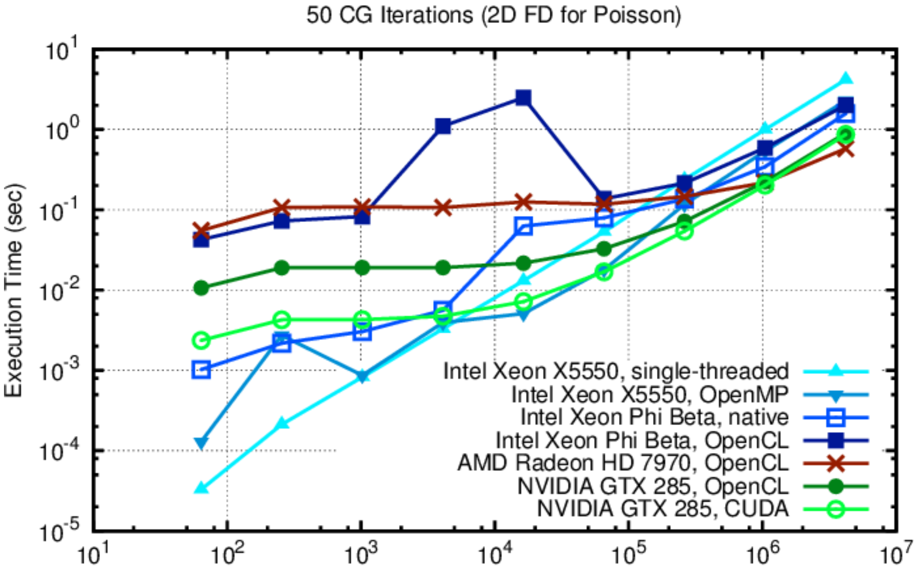


ViennaCL

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



ViennaCL Benchmark



GPU-enabled Linear Algebra Libraries: Words of Caution

- This is an exciting and dynamic field. Software and hardware are in flux: "Don't believe it unless you can run it"
- **Be aware of the fundamental limitations of the hardware and current software, especially with sparse linear algebra**
- If you expect to be memory bandwidth limited, look up the values for your CPU, your GPU, and your PCIe bus. This (not peak floating point performance) will likely bound speedup.
- For example, on Piz Daint, peak memory bandwidths are:
- The Pascal P100 16GB : 732 GB/s
- The Haswell ("gpu" XC50) : 68 GB/s
- Broadwell ("mc" XC40) dual-socket : $2 \times 77 = 154$ GB/s.
- Also consider these numbers weighted by power consumption, hardware cost, degree of parallelism required to reach peak performance, etc.
- Linear solvers: be aware that there are typically fewer GPU-enabled preconditioners available (though we are hard at work!).

Peak Memory Bandwidth over Time

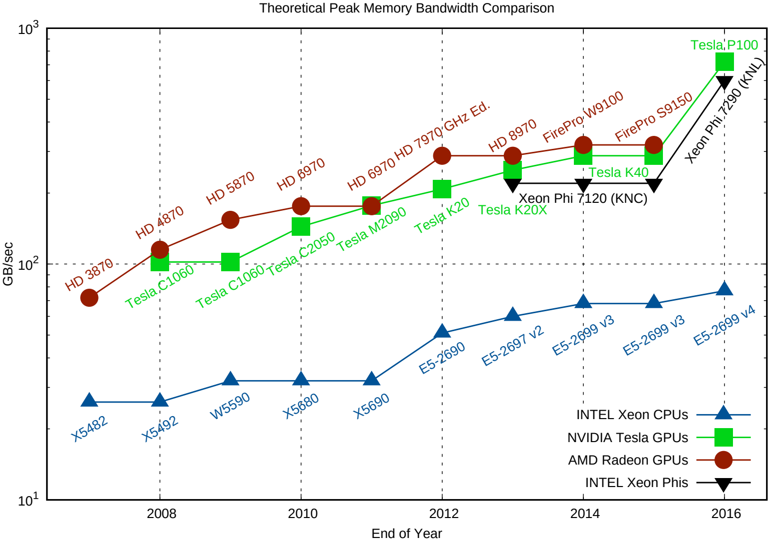


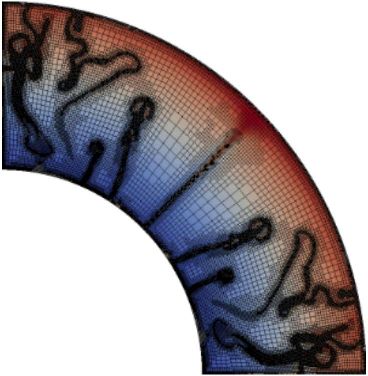
Chart by Karl Rupp from https://github.com/karlrupp/cpu-gpu-mic-comparison.

Section 6

Higher-level and Expansive Libraries/Frameworks

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)
- Libmesh (libmesh.github.io)
- These libraries all offer excellent documentation and high-level interfaces in the case of Fenics and Firedrake



aspect.dealii.org

HSL

- hsl.rl.ac.uk
- Formerly "The Harwell Subroutine Library", originally in Fortran with a long history
- Fortran, C, MATLAB interfaces
- Eigenanalysis, general linear algebra, ordering routines, and Krylov methods,
- Including some algorithms not commonly found elsewhere.

HSL_MC64 Permute and scale a sparse unsymmetric or rectangular matrix to put large entries on the diagonal

Given a sparse unsymmetric or rectangular matrix $\mathbf{A} = \{a_{ij}\}_{m\times n}$, $m \geq n$, this subroutine attempts to find a row and column permutation that makes the permuted matrix have n entries on its diagonal. If the matrix is structurally nonsingular, the subroutine optionally returns a permutation that maximizes the smallest element on the diagonal, maximizes the sum of the diagonal entries, or maximizes the product of the diagonal entries of the permuted matrix. For the latter option, the subroutine also finds scaling factors that may be used to scale the original matrix so that the nonzero diagonal entries of the permuted and scaled matrix are one in absolute value and all the off-diagonal entries are less than or equal to one in absolute value. The natural logarithms of the scaling factors u_i, $i = 1, \dots, n$, for the rows and v_j, $j = 1, \dots, n$, for the columns are returned so that the scaled matrix $\mathbf{B} = \{b_{ij}\}_{n\times n}$ has entries

$$b_{ij} = a_{ij} \exp\left(u_i + v_j\right).$$

In this Fortran 95 version, there are added facilities from the original MC64 code for working on rectangular and symmetric matrices. For the rectangular case, a **row and column permutation** are returned so that the user can permute the matching to the diagonal and identify the rows in the structurally nonsingular block. For the symmetric case, the user must only supply the lower triangle and, if a scaling is computed, it will be a symmetric scaling with the same property as in the unsymmetric case. Structually non-singular matrices are supported using the maximum product matching only.

Version 2.3.1

20th March 2013

User documentation

- Fortran
- C
- MATLAB

Recent Changes

Code Download

- Single
- Double
- Single Complex
- Double Complex

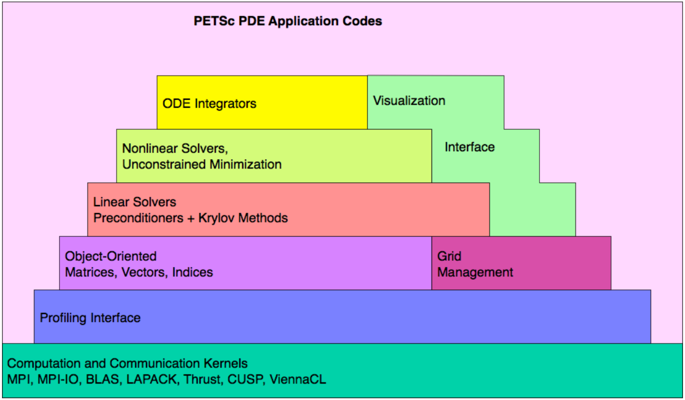
www.hsl.rl.ac.uk/catalogue/hsl_mc64.html

Trilinos

- trilinos.sandia.gov
- A collection of several somewhat-independent packages for scientific computation, covering essentially every topic discussed so far
  - 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

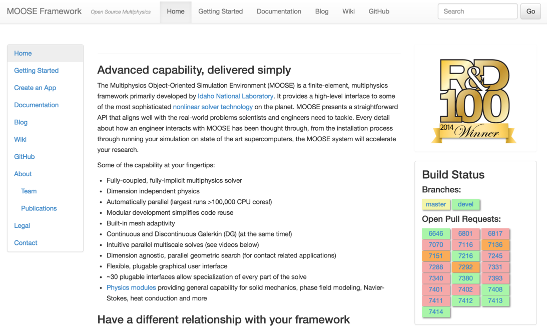
PETSc

- Toolkit centered around tools for large-scale discretized PDE
- Distributed sparse linear solvers, nonlinear solvers, ODE/DAE solvers, distributed data structures, ..
- More in the next lecture



MOOSE

- mooseframework.org
- Framework for multiphysics problems
- Developed for nuclear reactor simulation
- On top of PETSc and libmesh



Section 7

How To Ask Developers For Help

- Many of the Libraries discussed so far are
  - Open Source
  - Maintained partially or completely by small teams of volunteers
  - Not exhaustively documented
- Thus, if you decide to use one of them you may find it useful to ask a question.
- This is a very important skill, so let's attempt to quantify it a bit.
- **How?**

How to Pose a Question

Ask yourself questions about your question!

1. What *is* your question?
   - What are my assumptions?
2. What is your goal? (What is the *real* question?)
   - The “XY problem” : I have goal X, which I try to achieve with method Y. I fail, so I ask how to get Y to work, not mentioning X. (“Treating symptoms, not diseases”)
3. Are there terms you don't understand?
4. "Why can't I answer this myself?"

We will use a simple example to demonstrate these points more clearly

An example

- I have been tasked to prototype an application in C++. It is a FEM code, and fortunately my advisor has provided me with some reference code which accomplishes a similar task!
- You copy the code, set up a local git repository to track your progress, and happily begin modifying it to your purposes.
- At one point you want to prototype a local operation and, since you don't know Eigen, you write a little test script to understand how to add two small matrices together.

An example

```cpp
// Adapted from http://eigen.tuxfamily.org/dox/group__TutorialMatrixArithmetic.html

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

using namespace Eigen;

int main(int argc, char *argv[])
{

  Array2d a;
  a << 1, 2,
       3, 4;
  ArrayXd b(2,2);
  b << 2, 3,
       1, 4;
  std::cout << "a + b =\n" << a + b << std::endl;
  std::cout << "a - b =\n" << a - b << std::endl;

  return EXIT_SUCCESS;
}
```

An example

- Suddenly you run your code and are greeted with

```
$ make
clang++ -o main main.cpp
$ ./main
Assertion failed: (m_row<m_xpr.rows() && "Too many rows passed to comma initializer (
    operator<<)"), function operator,, file ./Eigen/src/Core/CommaInitializer.h, line
    66.
Abort trap: 6
```

Now What?!

Exercise: go over the standard questions

1. What *is* your question?
   - Terrible: "Why is Eigen so buggy!?"
   - Bad: "Why doesn't it work?"
   - Not that bad: "Why is this frustrating me so much?"
   - OK: "Why can't I add these two matrices?"
   - Good: "What's wrong with my assumption that I should be able to initialize arrays with the comma initializer and then add them?"
   - What are my assumptions?
     - The error message is incomprehensible
     - I can define matrices by comma-initializing Array objects
     - I can add these matrices together
     - I can output these matrices, ...
2. What is your goal? (What is the *real* question?)
   - How can I write a simple test of matrix addition with Eigen?
3. Are there terms you don't understand?
   - "Array"? "Comma initialization?","operator<<",...
4. Why can't I answer this myself?

How to ask for help

Now that I have posed my question and decided that I can't answer it, it's time to look for outside help.

- Have I read the error message?
- Is there documentation that I can understand? [If I can't understand it, why not?]
- Search engines
  - This is fairly obvious, but it's helpful to have understood your problem enough to provide good search terms.
- Colleagues
  - Often the most efficient way - no general advice here, as this depends strongly on the nature of the interpersonal relationship.
- Mailing Lists
- Q+A sites / message boards

Mailing Lists

When asking for help (often for free) from a stranger, it's important to establish immediately that you are taking the interaction seriously, and that you value their time.

- Read the error message, and understand the terms.
- Know the question you are asking. Refine it.
- Due Diligence. Error Messages. Documentation. FAQ.
- Write your email, and then edit it.
  - You are often writing because you are frustrated. Pretend to be calm.
- Describe the environment. **Allow reproducibility**.
  - Versions of software, compilers, and full error messages.
  - Code to reproduce, if at all possible
  - Question your assumptions.

Mailing Lists

- **Don't draw premature conclusions**. The most frustrating problems are usually the ones relying on your own faulty assumption.
  - Be very wary of claiming that you have found a bug.
  - It is better to simply report your problem in a reproducible way;
  - It's a good idea to ask one question at a time.
- **Don't be afraid!** Software developers can seem unfriendly, but this is often because someone hasn't done their due diligence.
  - **"Beginner's question"** $\neq$ **"Stupid Question"**
  - "Question with an easy-to-find answer" $=$ "Stupid Question"
  - These lists are beneficial to the developers as well, and they are meant to be used - bugs are reported, user experience is reported.
- Don't be offended by short answers, lack of salutations and other formalities, etc.
- Empathize with the people on the other end (if someone were to ask you a question about code problems, what would you want?).

Two Ways To Answer a Question

1. Experience ("I've seen this before")
2. Experiment ("I have sufficient information to test hypotheses")

You want your email to allow at least the former, and if at all possible the latter (send code)!

Q+A sites and message boards

Most of the same points apply as with email, but a few more:

- You should already have identified unknown terminology, but now do your best to define the relevant terms to your problem. For example, "Eigen", "Array", "C++", "comma initializer", "runtime error", might be helpful for our example.
- Due diligence. Search the site. Read the site's rules and FAQ.
- Allow reproducibility. Give details of your environment, and code if at all possible.
- Value the reader's time. Write clearly, and format nicely.
- Post what worked for future readers.
- Put some **words in bold** [9].

[9] (This is partially a joke)

Our example, continued

- Say that I am still having problems with my code to add two matrices together.
- Mental Exercise: How would you draft an email to ask about this? (In reality, this problem can probably be resolved before you would need to ask the developers)

Post-mortem

Learn from your mistakes

- What was the actual solution my problem? (You may have tried many things)
- What concept, if any, led to my confusion?
  - In our example, it was our confusion (perhaps born of `MATLAB`), that an `Array` is the same thing as a `Matrix`.
  - This is especially important if you "solved" your problem by googling the error message and copying whatever you found. Can you understand better, while the issue is fresh in your mind?
- If this was difficult or time-consuming in any way, where can I write this down so that when I see this same error, I won't have to repeat all of this?
  - This is especially useful when dealing with large and complex systems which are being changed and updated

Section 8

Other Topics and Final Thoughts

Other Topics

We've focused on solver libraries here, due to time and my expertise, but there are entire classes of libraries we've left out:

- Optimization Libraries: TAO, IPOPT, Dolphin-adjoint, ...
- ODE/Timestepper libraries : SUNDIALS, ...
- Meshing and Partitioning Libraries: METIS/PARMETIS, TetGen ...
- I/O and database libraries (see other Summer School lectures)
- Communication Libraries (see other Summer School lectures)
- Domain/physics-specific libraries
- Many more... so many more that it can be daunting!

Final Thoughts

- Libraries are, ultimately, supposed to save time
- How do you know if a library is worth your time? "Measure twice; cut once"
  - Make sure you understand what the needs of your code actually are (Not as easy as it sounds, as you may want your library to keep up as your code scales and changes)
  - Make sure that you understand what each library actually does (Read documentation and publications. If you can't tell what the library does, that's a bad sign)
  - Ask everyone that you can
  - Look for real, working examples
  - Look for active communities and help streams
  - Is the library depended upon by other libraries?
- Other time-saving tips
  - Do what you can to use the most recent version of a library. Practically, that is the one which will be supported.
  - Practice your question-asking skills (mailing lists, StackExchange, etc.)