Advanced topics

Simon Scheidegger
simon.scheidegger@gmail.com
August 3rd, 2017
Open Source Macroeconomics Laboratory – BFI/UChicago

Including adapted teaching material from books, lectures and presentations by
C. Gheller, M. Martinasso, P. Sanan, M. Wetzstein

<u>Outline</u>

1.) Libraries

2.) Emerging hardware

3.) Trending topics

Libraries*

*The other commandments got lost in translation

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 Application Programming Interface (API)

- Typically versioned, documented, distributed, licensed.

Libraries for Scientific Computing: Pros

- **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

- Learning curves.

- Versioning, changing APIs.

- Bugs that someone else must fix.

- Syntax, design choices.

- Lack of documentation (or local experts).

- Oversold software.

- The scientific risks of using algorithms (or hardware) that you don’t understand.

Where libraries show up

There are many libraries for scientific computing. Some are specialized to **certain application** areas, others are **quite general**.

- Linear Algebra Libraries
- Sparse Linear Algebra
- Iterative Solvers
- Eigensolvers
- GPU-enabled Linear Algebra Libraries

$\Big\}$ Examples for general libs.

- TASMANIAN (Sparse grids)
- IPOPT

$\Big\}$ Specialized lib. examples

Dense Linear Algebra

- Dense linear algebra, that is – linear algebra on matrices that are stored as two-dimensional arrays has been standardized for a considerable time.

- You almost certainly use these operations already.

- You likely leverage (perhaps indirectly) libraries to do so.

→ Typical Operations include elementary **element-wise operations on matrices and vectors** : A + B,etc.

→ Norms, inner products, matrix-matrix multiplies, matrix-vector multiplies : $||x||_2, \langle x, y\rangle, AB, Ax,$

→ 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 triangle, R upper triangle

→ 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

- The basic operations are defined by the three levels of **Basic Linear Algebra Subprograms (BLAS)**:

→ **Level 1** defines **vector operations** that are characterized by a single loop.

→ **Level 2** defines **matrix vector** operations, both explicit such as the matrix-vector product, and implicit such as the solution of triangular systems.

→ **Level 3** defines **matrix-matrix operations**, most notably the matrix-matrix product.

- The name ‘BLAS’ suggests a **certain amount of generality**, but the original authors were clear that these sub-programs only covered dense linear algebra.

- Attempts to standardize sparse operations have never met with equal success.

BLAS & 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

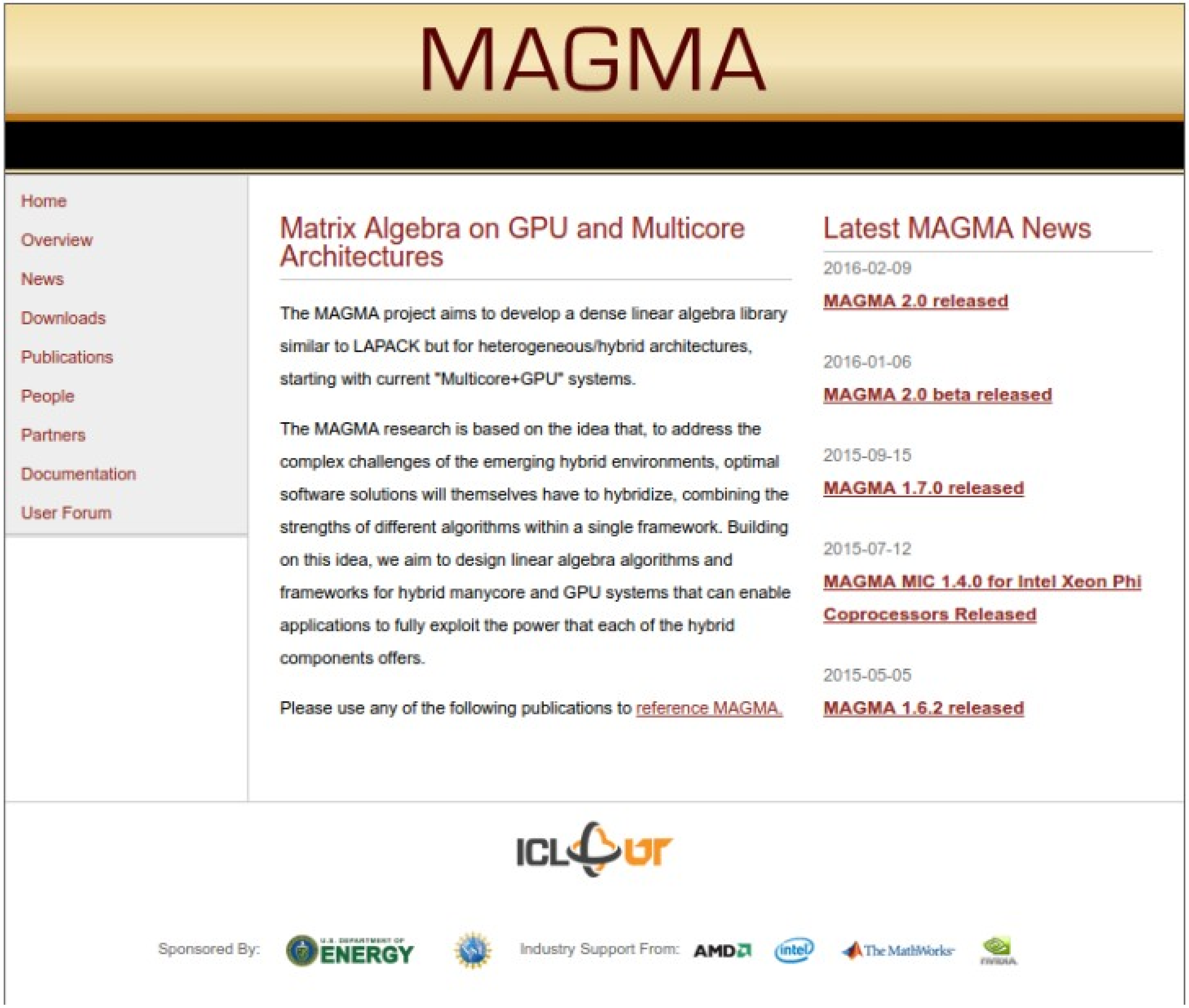
→ distributed memory LAPACK (includes BLACS as a communication layer)

- **Available implementations** on compute clusters often include the following:

→Intel’s math kernel library (MKL) includes BLAS and LAPACK,
Cray’s libsci : heavily optimized BLAS, LAPACK, SCALAPACK within the Cray, PGI, and GNU environments.

Example: MAGMA

http://icl.cs.utk.edu/magma/



Matrix Algebra on GPU and Multicore Architectures

The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.

The MAGMA research is based on the idea that, to address the complex challenges of the emerging hybrid environments, optimal software solutions will themselves have to hybridize, combining the strengths of different algorithms within a single framework. Building on this idea, we aim to design linear algebra algorithms and frameworks for hybrid manycore and GPU systems that can enable applications to fully exploit the power that each of the hybrid components offers.

Please use any of the following publications to reference MAGMA.

Latest MAGMA News

2016-02-09
MAGMA 2.0 released

2016-01-06
MAGMA 2.0 beta released

2015-09-15
MAGMA 1.7.0 released

2015-07-12
MAGMA MIC 1.4.0 for Intel Xeon Phi Coprocessors Released

2015-05-05
MAGMA 1.6.2 released

Sponsored By: U.S. DEPARTMENT OF ENERGY

Industry Support From: AMD intel The MathWorks NVIDIA

Sparse Linear Algebra

For a tutorial see http://www.users.csbsju.edu/~mheroux/ISC2016HerouxTutorial.pdf

- 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.

- Efficient for repeated solves suboptimal scaling and entry-dependent factorization time and storage.

- Challenging to parallelize.

- For large-enough systems, eventually beaten by optimally-scaling methods (iterative and/or multilevel algorithms).

Example: http://www.pardiso-project.org/

Download the package
(licence & binary) from there.

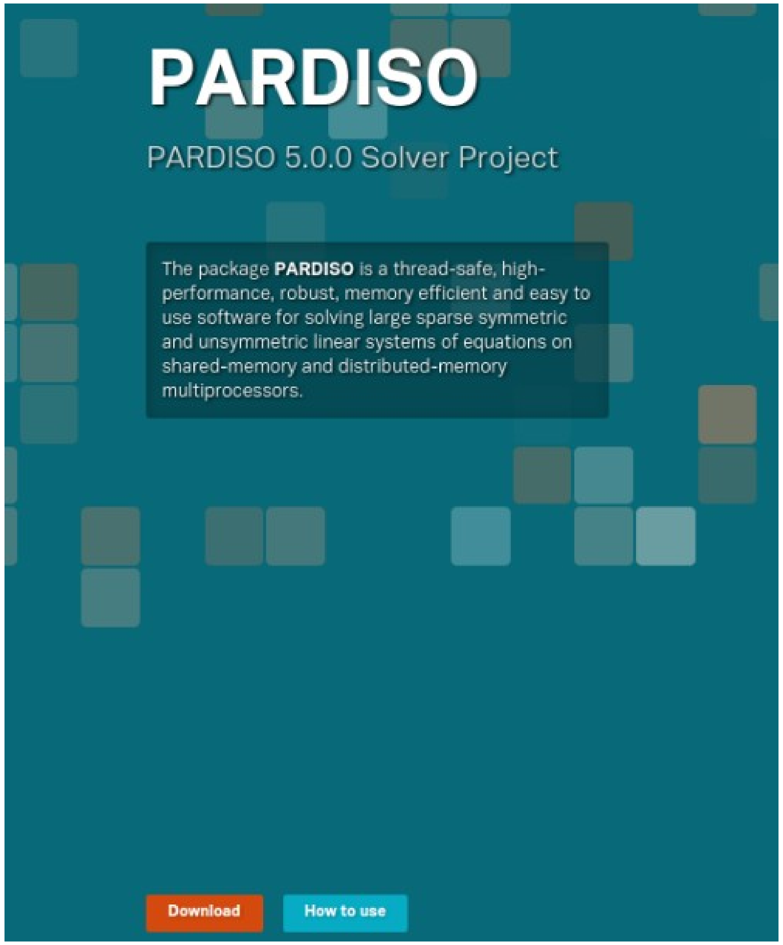
You want
libpardiso500-GNU481-X86-64.so

Add the binary here:
lectures/day5/code_day5/pardiso_example/lib

Copy the licence key to a
file named pardiso.lic

Make the licence visible by
a) adding it to your .bashrc
e.g. export PARDISO_LIC_PATH="~/licences/pardiso.lic"

b) putting pardiso.lic to your home directory (cp pardiso.lic ~)

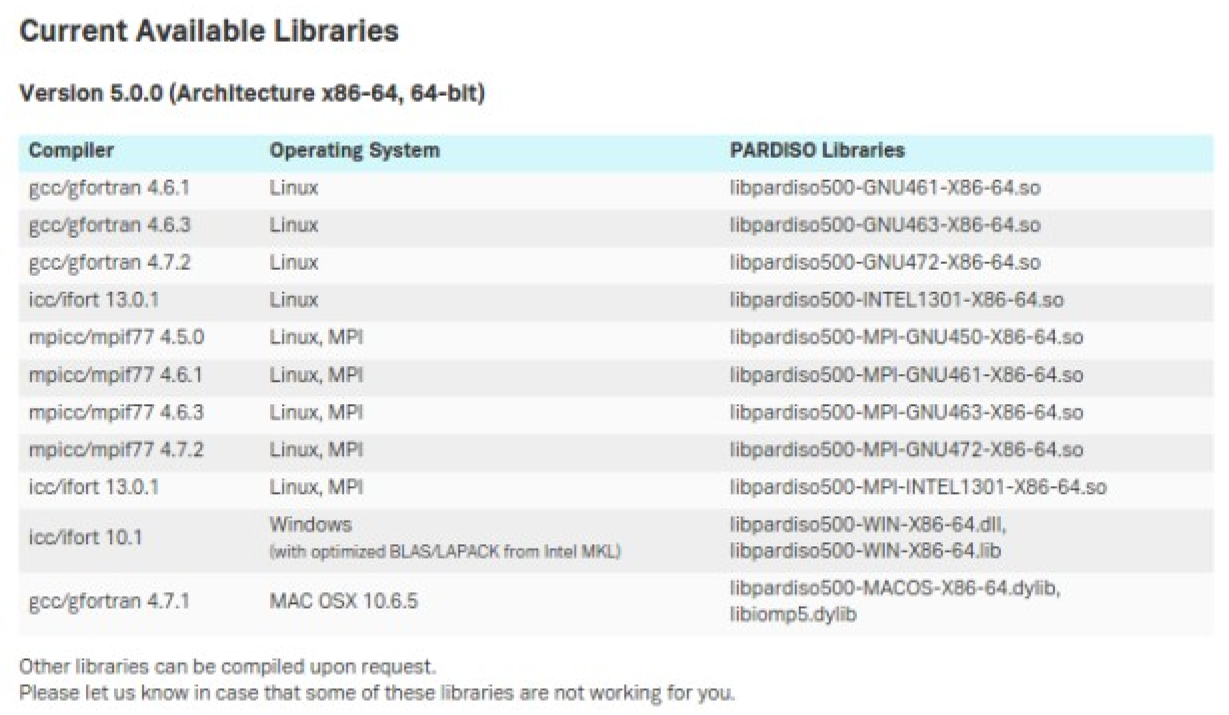


PARDISO

PARDISO 5.0.0 Solver Project

The package **PARDISO** is a thread-safe, high-performance, robust, memory efficient and easy to use software for solving large sparse symmetric and unsymmetric linear systems of equations on shared-memory and distributed-memory multiprocessors.

What hardware do you have?



Current Available Libraries

Version 5.0.0 (Architecture x86-64, 64-bit)

Compiler	Operating System	PARDISO Libraries
gcc/gfortran 4.6.1	Linux	libpardiso500-GNU461-X86-64.so
gcc/gfortran 4.6.3	Linux	libpardiso500-GNU463-X86-64.so
gcc/gfortran 4.7.2	Linux	libpardiso500-GNU472-X86-64.so
icc/ifort 13.0.1	Linux	libpardiso500-INTEL1301-X86-64.so
mpicc/mpif77 4.5.0	Linux, MPI	libpardiso500-MPI-GNU450-X86-64.so
mpicc/mpif77 4.6.1	Linux, MPI	libpardiso500-MPI-GNU461-X86-64.so
mpicc/mpif77 4.6.3	Linux, MPI	libpardiso500-MPI-GNU463-X86-64.so
mpicc/mpif77 4.7.2	Linux, MPI	libpardiso500-MPI-GNU472-X86-64.so
icc/ifort 13.0.1	Linux, MPI	libpardiso500-MPI-INTEL1301-X86-64.so
icc/ifort 10.1	Windows (with optimized BLAS/LAPACK from Intel MKL)	libpardiso500-WIN-X86-64.dll, libpardiso500-WIN-X86-64.lib
gcc/gfortran 4.7.1	MAC OSX 10.6.5	libpardiso500-MACOS-X86-64.dylib, libiomp5.dylib

Other libraries can be compiled upon request.
Please let us know in case that some of these libraries are not working for you.

1-dimensional Poisson equation

- Let us assume a discrete Poisson equation in 1D Cartesian coordinates

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f(x), \qquad x \in [0,1] \qquad u(0) = u_0 \quad \text{and} \quad u(1) = u_1,$$

- Let's discretize it, having *m* stencils (i ∈ [1, *m*]), and boundary values at i = 0 and i = m + 1.

- At i = 1, the Poisson equation then reads: $\Phi_0 - 2\Phi_1 + \Phi_2 = 4\pi G h^2 \rho_1$

The lower boundary condition Φ_0 is now shifted tothe other side of the equation:

$$-2\Phi_1 + \Phi_2 = 4\pi G h^2 \rho_1 - \Phi_0$$

The upper boundary at i = m is treated in analogy: $\Phi_{m-1} - 2\Phi_m = 4\pi G h^2 \rho_m - \Phi_{m+1}$

Hence, the Poisson equation can be casted into a matrix notation,
representing an m × m linear system of the general form

$$\boxed{A\vec{\Phi} = \vec{\rho}\,.}$$

Multi-dimensional Poisson equation

- In 3D Cartesian coordinates, the Poisson equation for the gravitational potential reads:

$$\Delta\Phi(x,y,z) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\Phi = 4\pi G\rho(x,y,z)$$

- The discretized version of this equation (assuming a uniform spatial discretization) on an **m × n × k grid** yields the following formula:

$$\Phi_{i-1,j,k} + \Phi_{i,j-1,k} + \Phi_{i,j,k-1} - 6\Phi_{i,j,k} + \Phi_{i+1,j,k} + \Phi_{i,j+1,k} + \Phi_{i,j,k+1} = 4\pi G h^2 \rho(x,y,z)\ ,$$

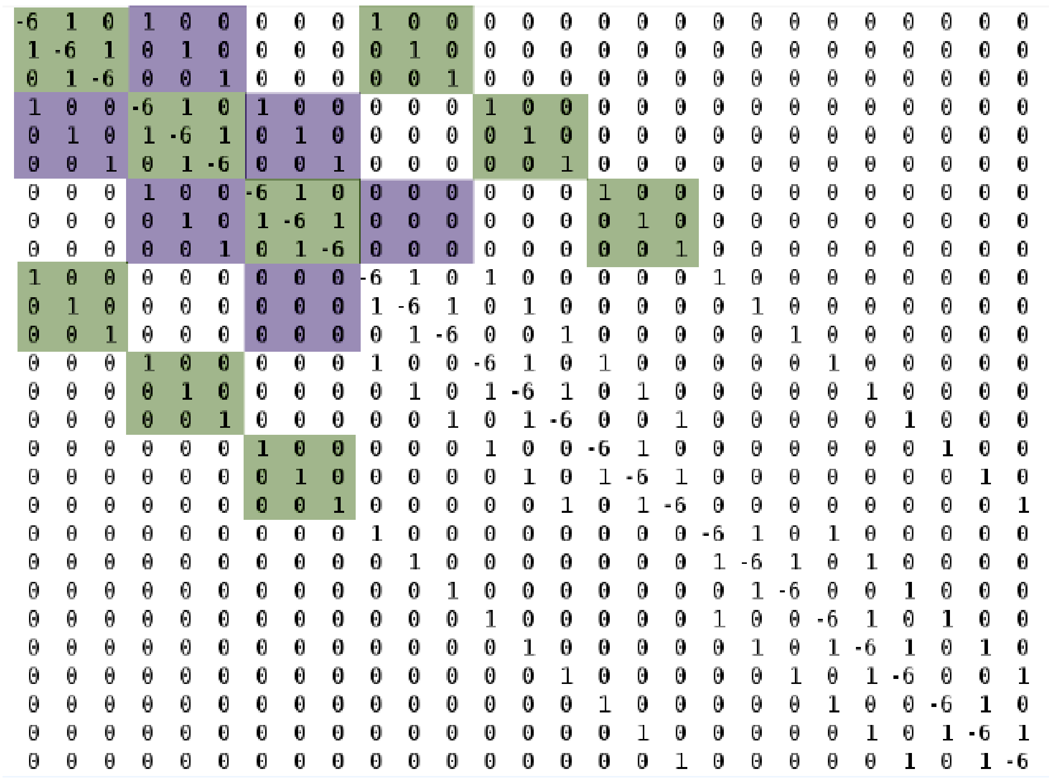
- ***h*** is the grid spacing.

- This can again be casted into a matrix notation, representing an **mnk × mnk linear system** of the general form

$$\boxed{A\vec{\Phi} = \vec{\rho}\,.}$$

Sparse matrix

For a 3 × 3 × 3 (m = 3, n = 3, k=3) grid with all the boundary nodes fixed (set to be zero), the matrix A of the system would look as displayed left below.
In our notation, the entries of the vectors **Φ** and **ρ** are defined as below right.



-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	-6	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	-6	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	-6	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	-6	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	-6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1	-6	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	-6	1	0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	1	-6	1	0	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	-6	1	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-6	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	-6	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	-6	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	-6	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6	1	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	-6	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	-6

$$\vec{\Phi} = \begin{pmatrix} \Phi_{111} \\ . \\ . \\ . \\ \Phi_{11k} \\ \Phi_{121} \\ . \\ . \\ . \\ \Phi_{1nk} \\ \Phi_{211} \\ . \\ . \\ . \\ \Phi_{mnk} \end{pmatrix} \quad ; \quad \vec{\rho} = 4\pi G h^2 \begin{pmatrix} \rho_{111} \\ . \\ . \\ . \\ \rho_{11k} \\ \rho_{121} \\ . \\ . \\ . \\ \rho_{1nk} \\ \rho_{211} \\ . \\ . \\ . \\ \rho_{mnk} \end{pmatrix}$$

This (and many other) PDEs can be solved by inverting the sparse matrix

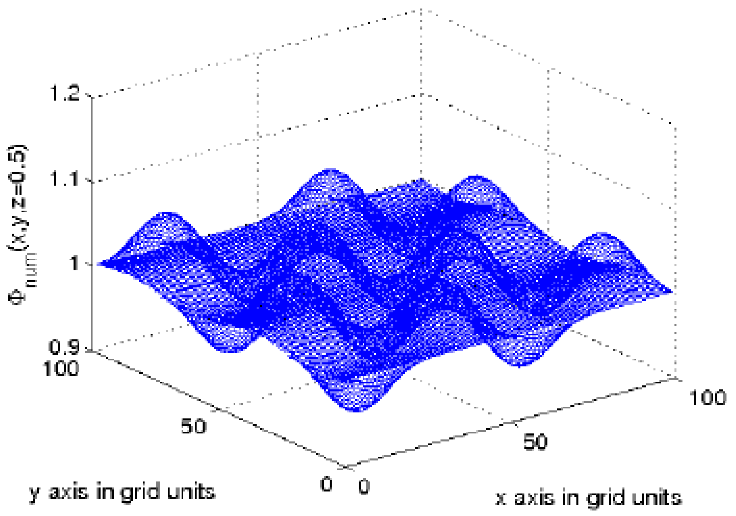
An analytical example

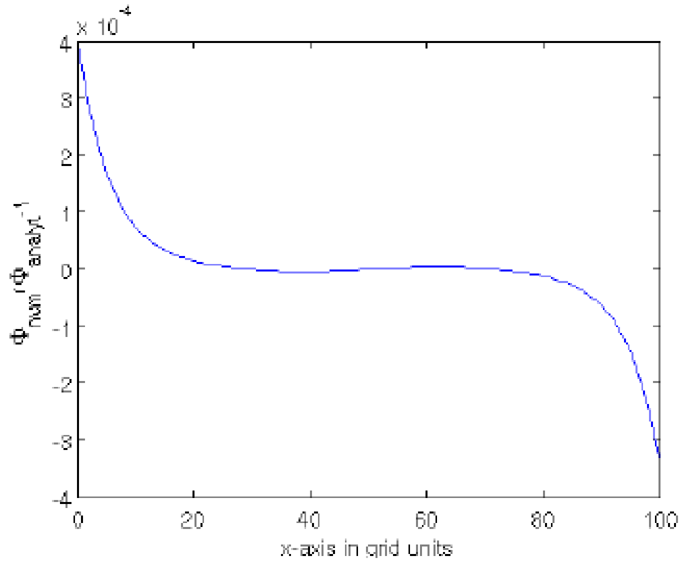
One exemplary analytical test: $\Delta\Phi(x,y,z) = f(x,y,z) = -48\pi^2 \sin(4\pi x)\sin(4\pi y)\sin(4\pi z)$.

Let's map the interval x, y, z ∈ 100^3 , mapped onto grid points

We impose Dirichlet boundary conditions ∂Ω = 1.

The analytical solution reads $\Phi(x,y,z) = \sin(4\pi x)\sin(4\pi y)\sin(4\pi z)$

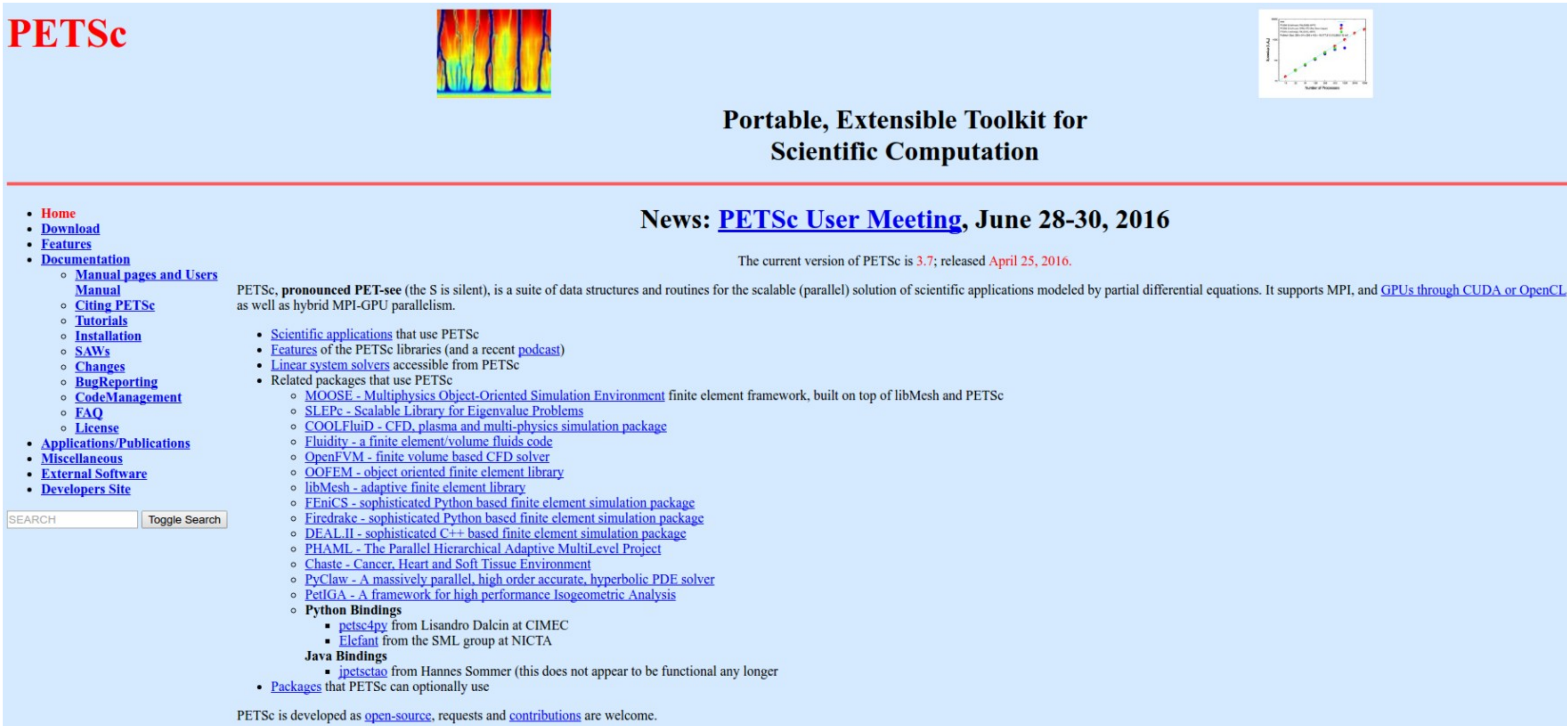




Left panel: Numerical solution of eq. 2.85 in the $z = 0.5$ plane. **Right panel:** Relative error $\Phi_{num}(x, 0.5, 0.5)/\Phi_{analyt}(x, 0.5, 0.5) - 1$ along the x-axis.

Example: PETSc

https://www.mcs.anl.gov/petsc/



PETSc

Portable, Extensible Toolkit for Scientific Computation

- Home
- Download
- Features
- Documentation
  - Manual pages and Users Manual
  - Citing PETSc
  - Tutorials
  - Installation
  - SAWs
  - Changes
  - BugReporting
  - CodeManagement
  - FAQ
  - License
- Applications/Publications
- Miscellaneous
- External Software
- Developers Site

SEARCH Toggle Search

News: PETSc User Meeting, June 28-30, 2016

The current version of PETSc is 3.7; released April 25, 2016.

PETSc, **pronounced PET-see** (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL as well as hybrid MPI-GPU parallelism.

- Scientific applications that use PETSc
- Features of the PETSc libraries (and a recent podcast)
- Linear system solvers accessible from PETSc
- Related packages that use PETSc
  - MOOSE - Multiphysics Object-Oriented Simulation Environment finite element framework, built on top of libMesh and PETSc
  - SLEPc - Scalable Library for Eigenvalue Problems
  - COOLFluiD - CFD, plasma and multi-physics simulation package
  - Fluidity - a finite element/volume fluids code
  - OpenFVM - finite volume based CFD solver
  - OOFEM - object oriented finite element library
  - libMesh - adaptive finite element library
  - FEniCS - sophisticated Python based finite element simulation package
  - Firedrake - sophisticated Python based finite element simulation package
  - DEAL.II - sophisticated C++ based finite element simulation package
  - PHAML - The Parallel Hierarchical Adaptive MultiLevel Project
  - Chaste - Cancer, Heart and Soft Tissue Environment
  - PyClaw - A massively parallel, high order accurate, hyperbolic PDE solver
  - PetIGA - A framework for high performance Isogeometric Analysis
  - **Python Bindings**
    - petsc4py from Lisandro Dalcin at CIMEC
    - Elefant from the SML group at NICTA

    Java Bindings
    - jpetsctao from Hannes Sommer (this does not appear to be functional any longer
- Packages that PETSc can optionally use

PETSc is developed as open-source, requests and contributions are welcome.

What is PETSc?

PETSc, the Portable Extendible Toolkit for Scientifc Computation, is a large powerful library, mostly concerned with linear and nonlinear system of equations that arise from discretized PDEs.

PETSc can be used as a **library in the traditional** sense, where you use some **high level functionality**, such as **solving a nonlinear system** of equations, in your program.

However, it **can also be used as a toolbox**, to compose your own numerical applications using low-level tools.
- Linear system solvers (sparse/dense, iterative/direct)
- Nonlinear system solvers
- Tools for distributed matrices
- Support for profiling, debugging, graphical output

The basic functionality of PETSc can be extended through external packages:
- Dense linear algebra: Scalapack, Plapack
- Grid partitioning software: ParMetis, Jostle, Chaco, Party
- ODE solvers: PVODE
- Eigenvalue solvers (including SVD): SLEPc
- Optimization: TAO

Why use PETSc?

- Write robust, scalable MPI codes to solve PDEs, without writing much MPI code yourself.

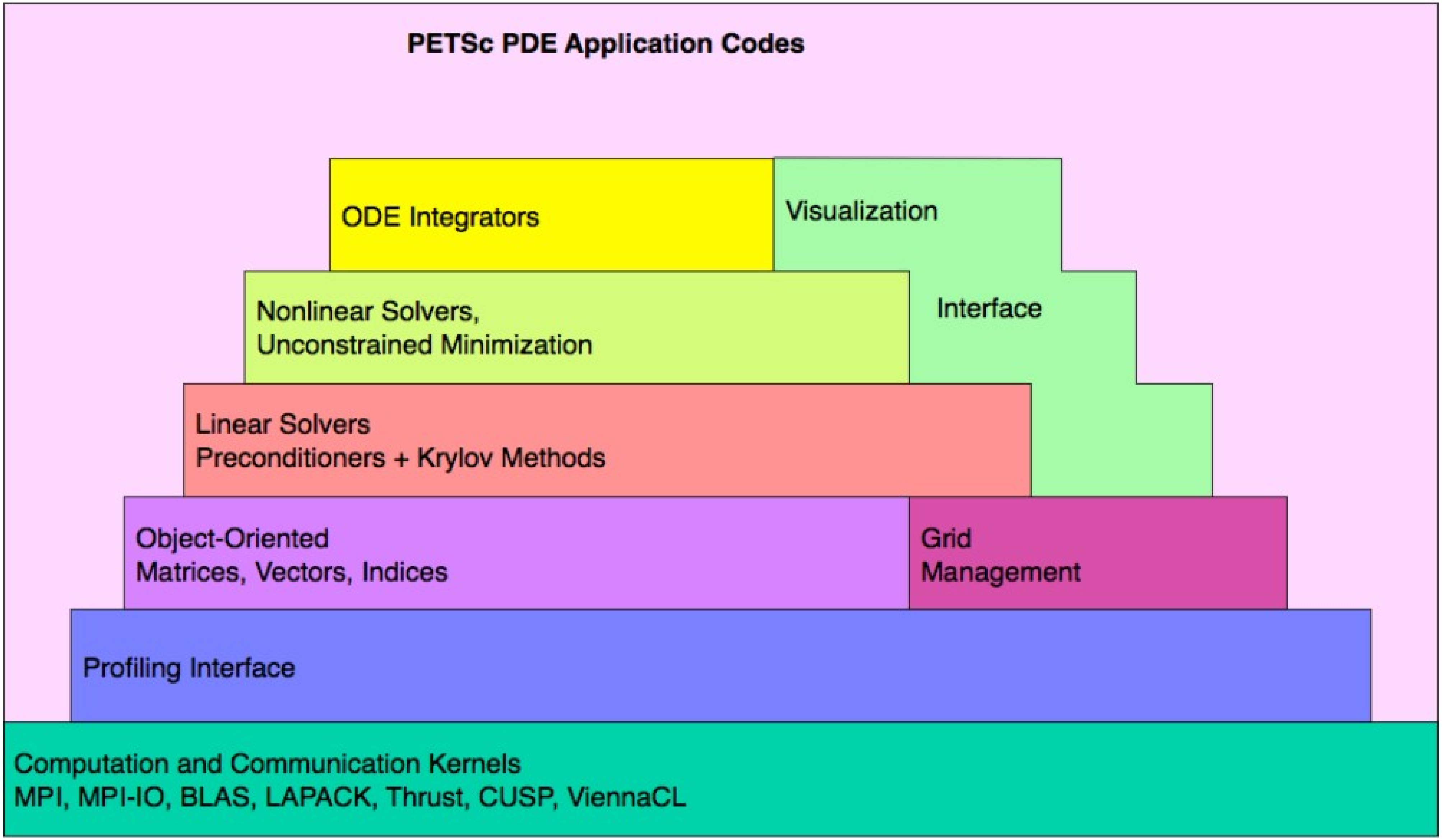
- Use a combinatorial explosion of solvers, configurable at runtime.

- Run your code essentially anywhere, from your laptop up to CSCS/Alphacruncher.

- Configure with a huge number of external packages (including external linear solvers).

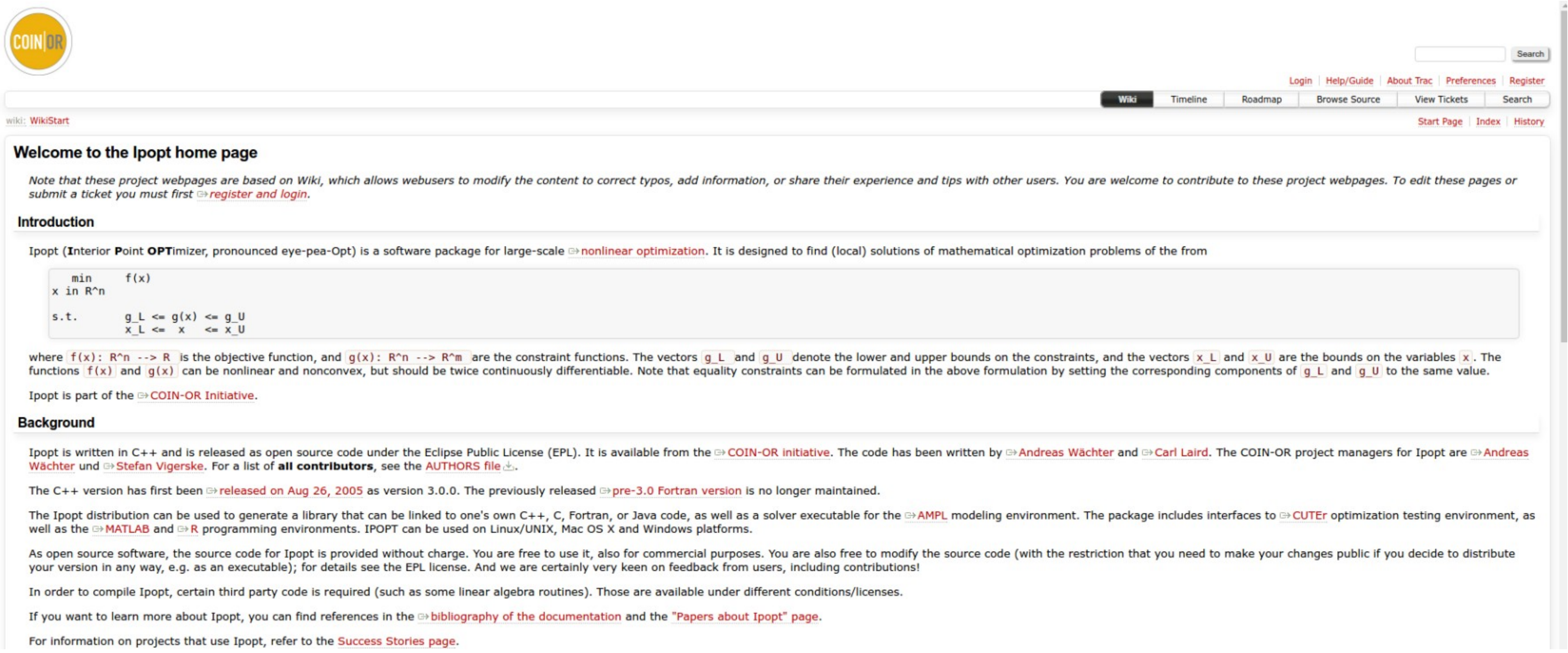
- Excellent support and community.

PETSc Components


PETSc PDE Application Codes
ODE Integrators
Visualization
Nonlinear Solvers,
Unconstrained Minimization
Interface
Linear Solvers
Preconditioners + Krylov Methods
Object-Oriented
Matrices, Vectors, Indices
Grid
Management
Profiling Interface
Computation and Communication Kernels
MPI, MPI-IO, BLAS, LAPACK, Thrust, CUSP, ViennaCL

Example: Optimization

https://projects.coin-or.org/Ipopt – we used it in one of the projects


COIN|OR
Search
Login | Help/Guide | About Trac | Preferences | Register
Wiki
Timeline
Roadmap
Browse Source
View Tickets
Search
wiki: WikiStart
Start Page | Index | History

Welcome to the Ipopt home page

Note that these project webpages are based on Wiki, which allows webusers to modify the content to correct typos, add information, or share their experience and tips with other users. You are welcome to contribute to these project webpages. To edit these pages or submit a ticket you must first register and login.

Introduction

Ipopt (**I**nterior **P**oint **OPT**imizer, pronounced eye-pea-Opt) is a software package for large-scale nonlinear optimization. It is designed to find (local) solutions of mathematical optimization problems of the from

```
   min     f(x)
x in R^n

s.t.       g_L <= g(x) <= g_U
           x_L <=  x   <= x_U
```

where `f(x): R^n --> R` is the objective function, and `g(x): R^n --> R^m` are the constraint functions. The vectors `g_L` and `g_U` denote the lower and upper bounds on the constraints, and the vectors `x_L` and `x_U` are the bounds on the variables `x`. The functions `f(x)` and `g(x)` can be nonlinear and nonconvex, but should be twice continuously differentiable. Note that equality constraints can be formulated in the above formulation by setting the corresponding components of `g_L` and `g_U` to the same value.

Ipopt is part of the COIN-OR Initiative.

Background

Ipopt is written in C++ and is released as open source code under the Eclipse Public License (EPL). It is available from the COIN-OR initiative. The code has been written by Andreas Wächter and Carl Laird. The COIN-OR project managers for Ipopt are Andreas Wächter und Stefan Vigerske. For a list of **all contributors**, see the AUTHORS file.

The C++ version has first been released on Aug 26, 2005 as version 3.0.0. The previously released pre-3.0 Fortran version is no longer maintained.

The Ipopt distribution can be used to generate a library that can be linked to one's own C++, C, Fortran, or Java code, as well as a solver executable for the AMPL modeling environment. The package includes interfaces to CUTEr optimization testing environment, as well as the MATLAB and R programming environments. IPOPT can be used on Linux/UNIX, Mac OS X and Windows platforms.

As open source software, the source code for Ipopt is provided without charge. You are free to use it, also for commercial purposes. You are also free to modify the source code (with the restriction that you need to make your changes public if you decide to distribute your version in any way, e.g. as an executable); for details see the EPL license. And we are certainly very keen on feedback from users, including contributions!

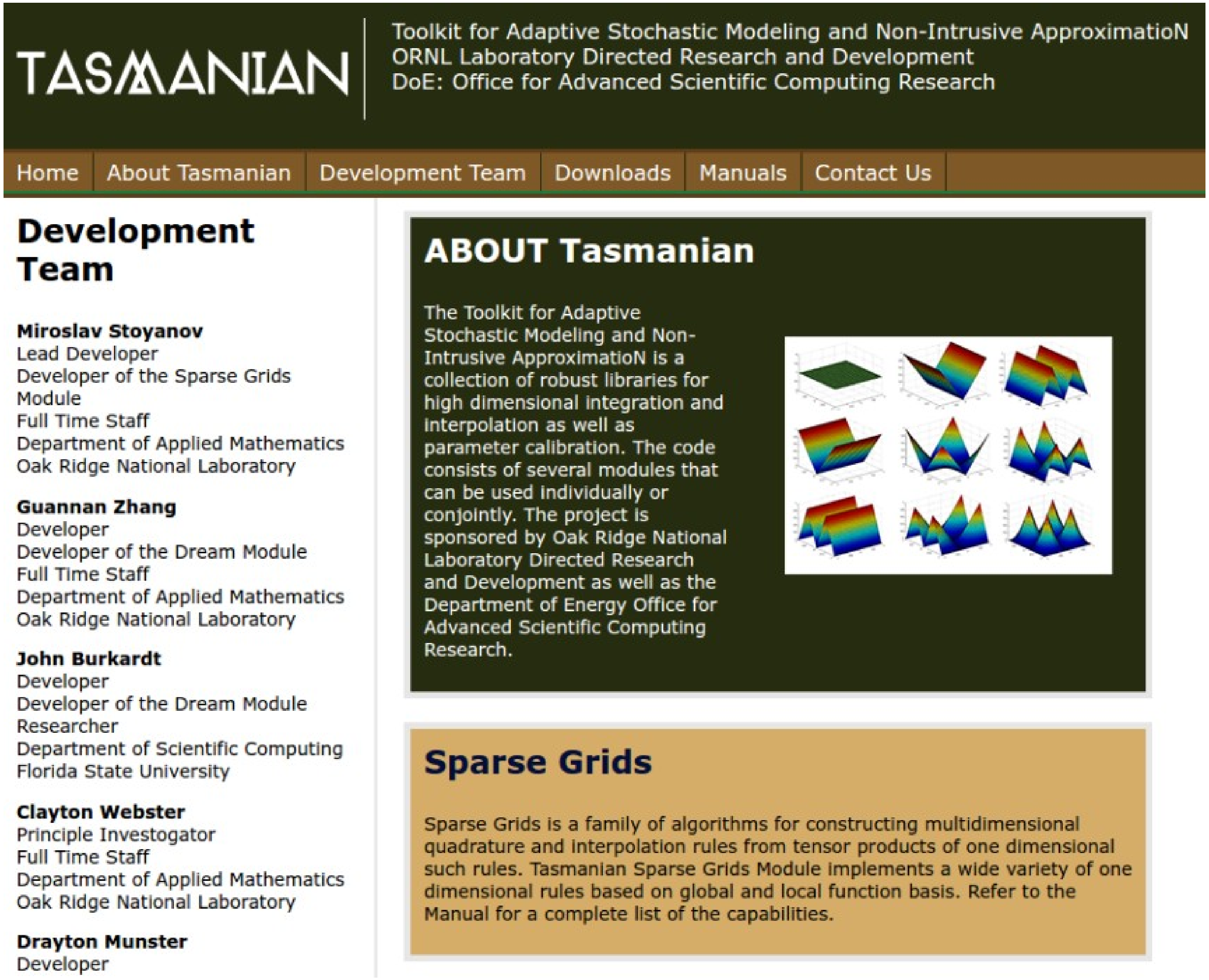
In order to compile Ipopt, certain third party code is required (such as some linear algebra routines). Those are available under different conditions/licenses.

If you want to learn more about Ipopt, you can find references in the bibliography of the documentation and the "Papers about Ipopt" page.

For information on projects that use Ipopt, refer to the Success Stories page.

Example: TASMANIAN

http://tasmanian.ornl.gov/



TASMANIAN

Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN
ORNL Laboratory Directed Research and Development
DoE: Office for Advanced Scientific Computing Research

Home | About Tasmanian | Development Team | Downloads | Manuals | Contact Us

Development Team

Miroslav Stoyanov
Lead Developer
Developer of the Sparse Grids Module
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

Guannan Zhang
Developer
Developer of the Dream Module
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

John Burkardt
Developer
Developer of the Dream Module
Researcher
Department of Scientific Computing
Florida State University

Clayton Webster
Principle Investogator
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

Drayton Munster
Developer

ABOUT Tasmanian

The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN is a collection of robust libraries for high dimensional integration and interpolation as well as parameter calibration. The code consists of several modules that can be used individually or conjointly. The project is sponsored by Oak Ridge National Laboratory Directed Research and Development as well as the Department of Energy Office for Advanced Scientific Computing Research.

Sparse Grids

Sparse Grids is a family of algorithms for constructing multidimensional quadrature and interpolation rules from tensor products of one dimensional such rules. Tasmanian Sparse Grids Module implements a wide variety of one dimensional rules based on global and local function basis. Refer to the Manual for a complete list of the capabilities.

Emerging hardware

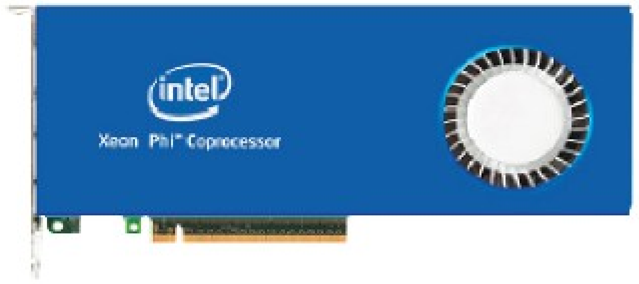
GPU: NVIDIA Tesla K20c

Kepler GK110, 28 nm

13 mp × 192 cores @ 0.71 GHz

5 GB GDDR5 @ 2.6 GHz

225W



MIC: Intel Xeon Phi 3120A

Knights Corner (KNC), 22 nm

57 cores @ 1.1 GHz

6GB GDDR5 @ 1.1 GHz

300W

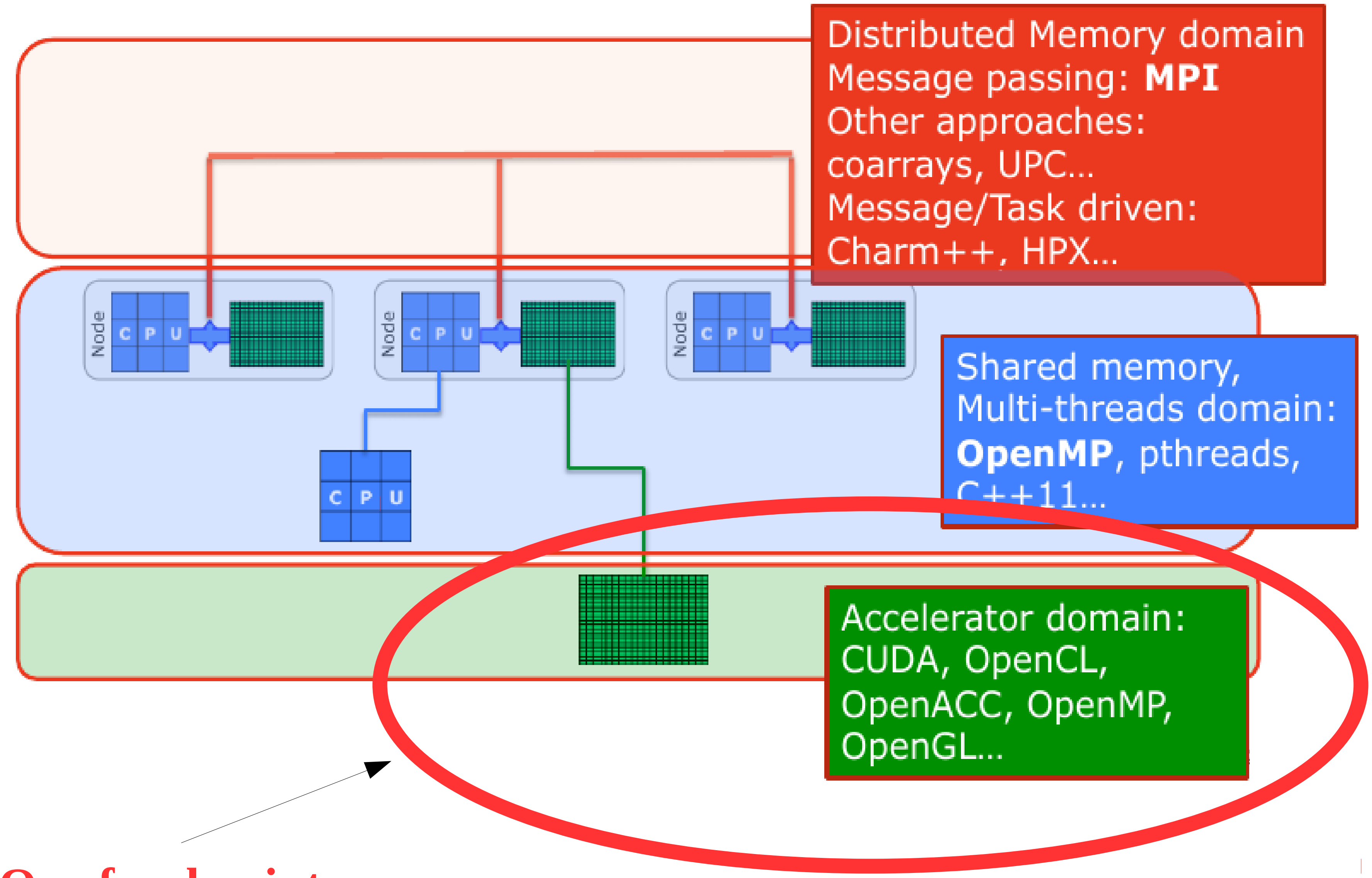
up to 4 threads per core

512-bit vectorization (AVX-512)

→ **Devices can have *O(Teraflops)***

Slide from D. Mikushin

Overall picture of programming models



Distributed Memory domain
Message passing: **MPI**
Other approaches:
coarrays, UPC...
Message/Task driven:
Charm++, HPX...

Shared memory,
Multi-threads domain:
OpenMP, pthreads,
C++11...

Accelerator domain:
CUDA, OpenCL,
OpenACC, OpenMP,
OpenGL...

Our focal point

(Slide from C. Gheller)

Why do we Need Coprocessors/or "Accelerators" on HPC?

- In the past, computers got faster by increasing the **clock frequency** of the core, but this has now reached its limit mainly due to **power requirements** and heat dissipation restrictions (unmanageable problem).

- **Today, processor cores are not getting any faster, but instead the number of cores per chip increases.**

- **On HPC, we need a chip that can provide higher computing performance at lower energy.**

General Purpose GPU

- Graphics Processing Unit (GPU):
  → Hardware designed for output to display.

- **General Purpose computing on GPUs (GPGPU)**:

  → Use GPUs for **non-graphics tasks,** e.g. physics simulation, signal processing, computational geometry, computer vision, database management, computational biology, computational finance

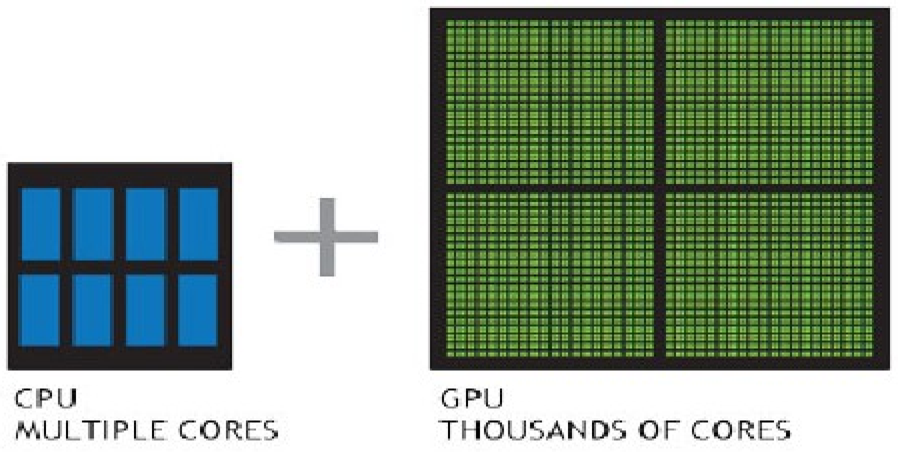
- GPUs evolved into a very flexible and powerful processor:
  → It’s programmable using high-level languages

Solution

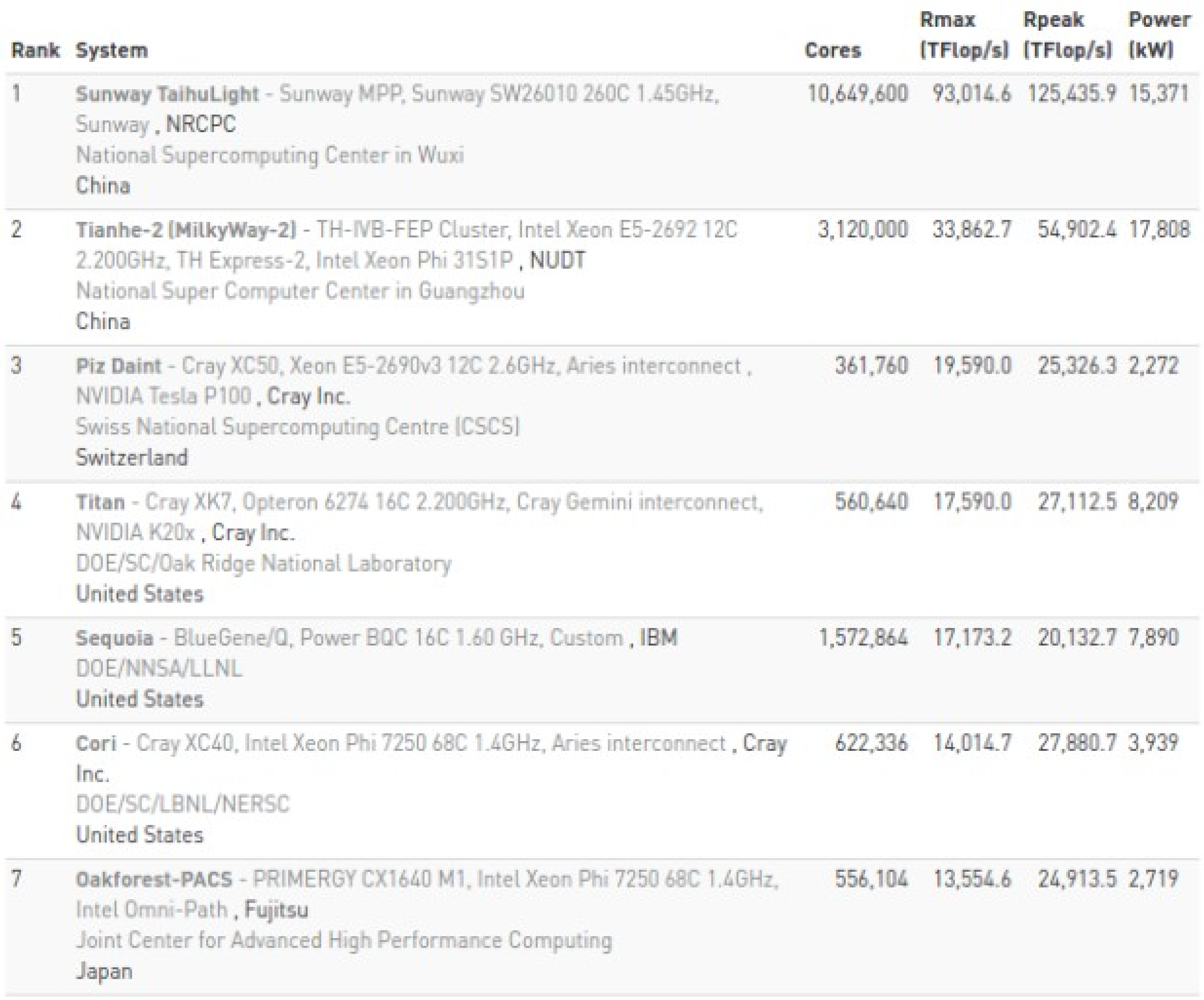
- The actual solution is a heterogeneous system containing both CPUs and “accelerators”, plus other forms of parallelism such as vector instruction support.

- Widely accepted that heterogeneous systems with accelerators deliver the highest performance and energy efficient computing in HPC.

- Today – the accelerated computing is revolutionising HPC.



Top 500 – June 2017



Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
6	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939
7	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu Joint Center for Advanced High Performance Computing Japan	556,104	13,554.6	24,913.5	2,719

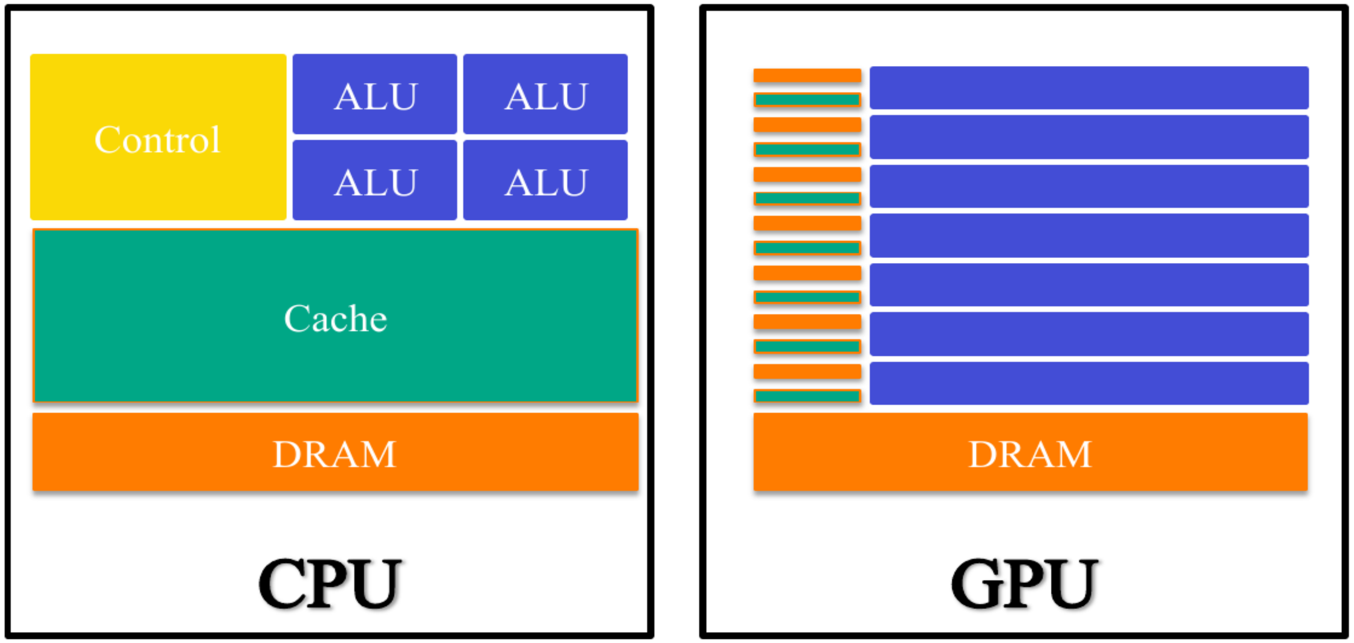
CPU vs. GPU

- Specialized for compute-intensive, highly-parallel computation, i.e. graphic output.

- Evolution pushed by gaming industry.

- CPU: large die area for control and caches.

- GPU: large die area for data processing.



Programming GPUs

- CUDA: Nvidia proprietary API, works only on Nvidia GPUs.

- OpenCL: open standard for heterogeneous computing.

- OpenACC: open standard based on compiler directives.

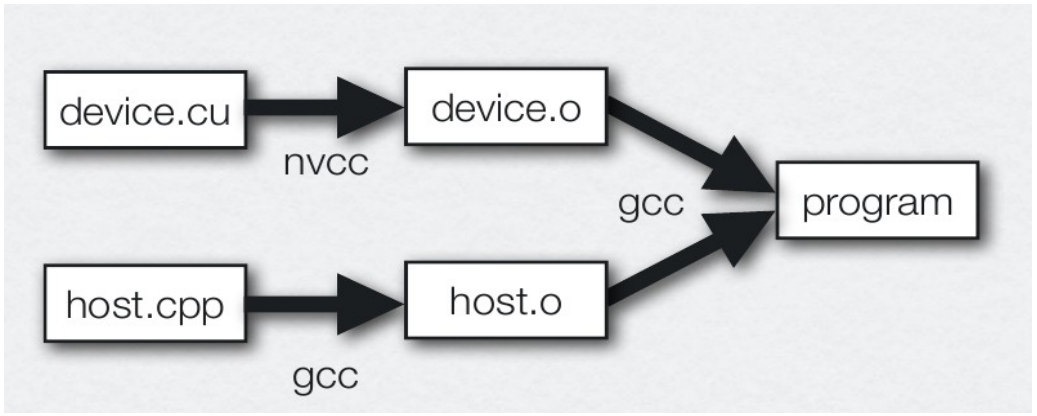
Nvidia CUDA

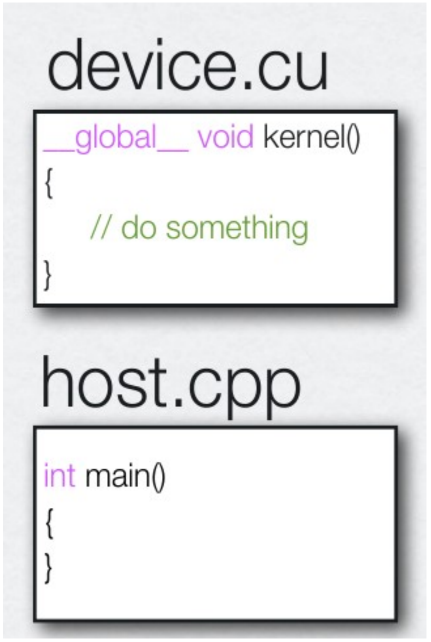
- Compute Unified Device Architecture (CUDA)

- C extension to write GPU code, support for C++

- Only supported by Nvidia GPUs

- Code compilation (nvcc) and linking:





device.cu

```
__global__ void kernel()
{
    // do something
}
```

host.cpp

```
int main()
{
}
```

OpenACC

- Programming with CUDA can be more difficult than writing SPMD (i.e., MPI) applications.

→ **OpenACC (Open Accelerators)**

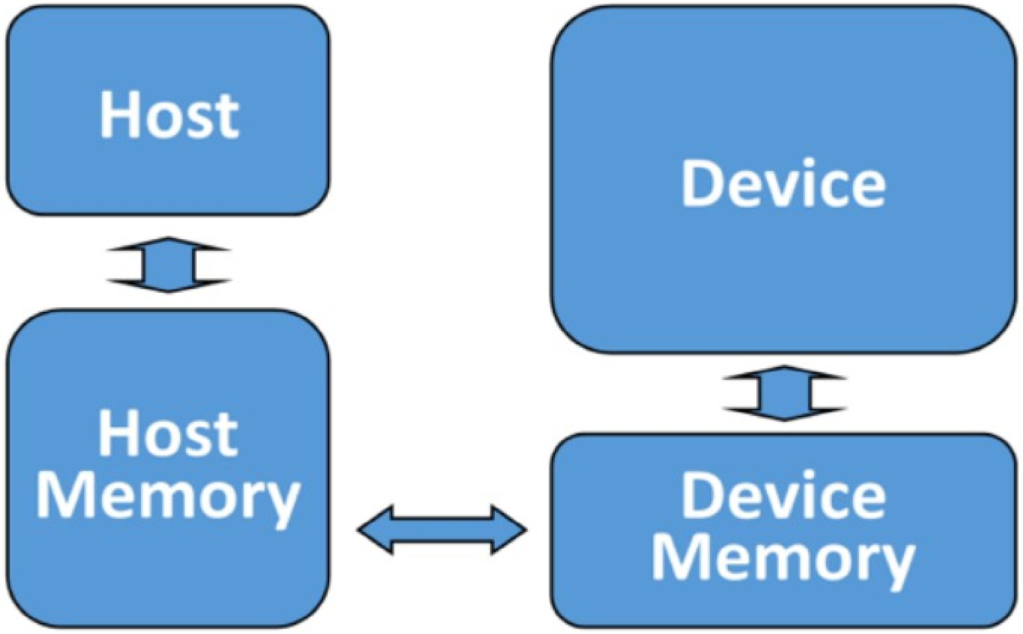
- Developed by Cray, CAPS, Nvidia, and PGI.
- Most recent specification: 2.5 (November 2015).
- Similar to OpenMP.
- High-level of abstraction.
- OpenACC members are also part of the OpenMP language committee.
- Compiler support from Cray, PGI, and CAPS.
- Experimental support for OpenACC in GCC/5.1.

OpenACC

- The **OpenACC API is a set of compiler directives** for offloading work to accelerators.

- For many systems, there will is a CPU host and GPU accelerator.

- **OpenACC will handle any accelerator memory management and the transfer of data**.



OpenACC Abstract Accelerator

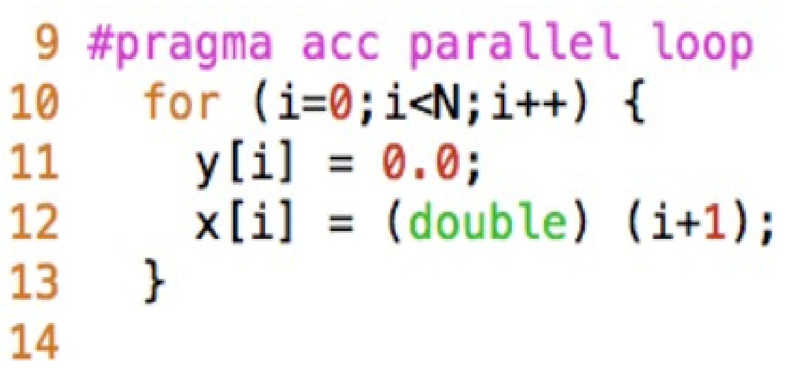
Directives

- Like OpenMP, OpenACC is primarily programmed using directives.
- Lower-level programming models like CUDA perform better for certain optimizations (i.e. abstraction penalty).

C/C++

#pragma acc directive-name [clause-list] new-line
Scope is the following block of code

Fortran
!$acc directive-name [clause-list] new-line
Scope is until **!$acc end** directive name



```
#pragma acc parallel loop
  for (i=0;i<N;i++) {
    y[i] = 0.0;
    x[i] = (double) (i+1);
  }

```

Intel Xeon Phi Products

- The first product was released in 2012 named **Knights Corner (KNC)** which is the first architecture supporting **512 bit vectors**.

- The 2nd generation released last week named **Knights Landing (KNL)** also support 512bit vectors with a new instruction set called Intel Advanced Vector Instructions 512 (Intel AVX-512).

- KNL has a peak performance of **6 TFLOP/s in single precision** ~ 3 times what KNC had, due to 2 **vector processing units (VPUs)** per core, doubled compared to the KNC.
  → Each VPU operates independently on 512-bit vector registers, which can hold 16 single precision or 8 double precession floating-point numbers.

Knights Corner (KNC)

- Add-on to CPU based systems (PCIe interface).

- High power efficiency ~ 1 TFLOP/s in DP.

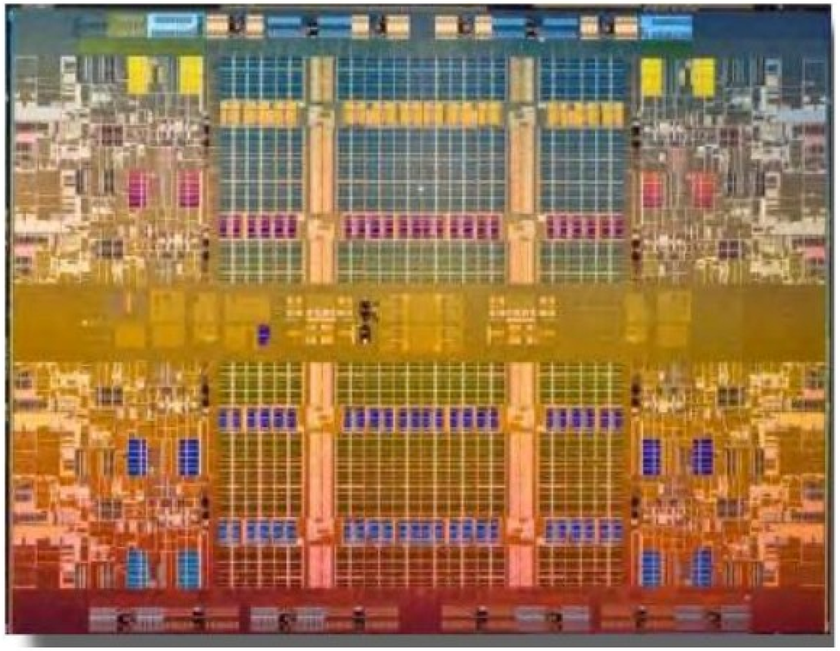
- Heterogeneous clustering.

- Currently it’s a major part of the world’s 2 fastest supercomputer: Tianhe-2 in Guangzhou, China.

- Today most of the major HPC vendors are supporting Xeon Phi.

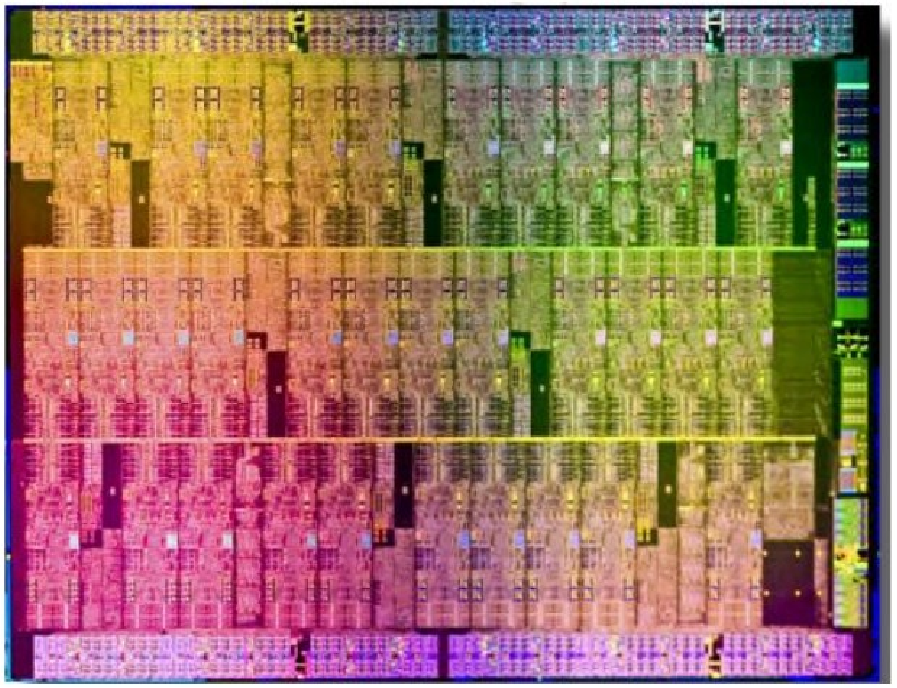
Multi-core vs. Many-core

Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Colfax 2013
http://www.colfaxintl.com/nd/xeonphi/book.aspx



Multi-core Intel Xeon processor

- ~ 16 physical cores ~ 3 GHz
- e.g Intel Sandy-Bridge 32 nm
- (AVX) 256-bit vector registers



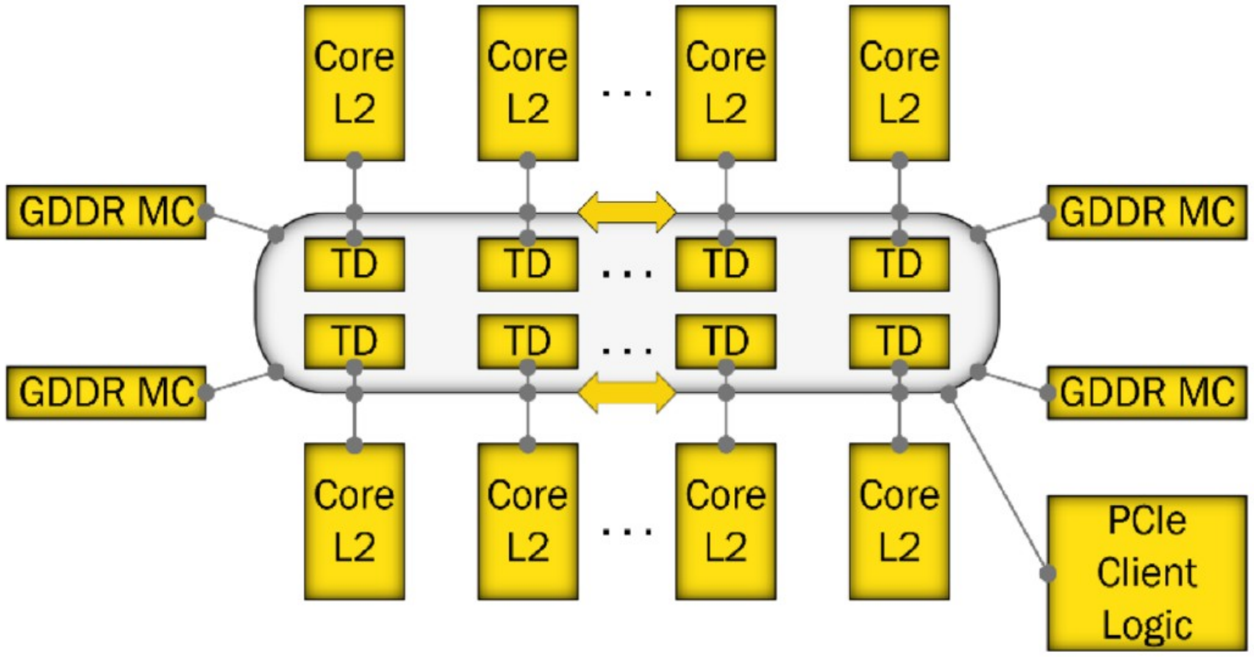
Many-core Intel Xeon Phi coprocessor

- ~ 61 cores (244) ~ 1 GHz
- 22 nm
- 512-bit vector registers

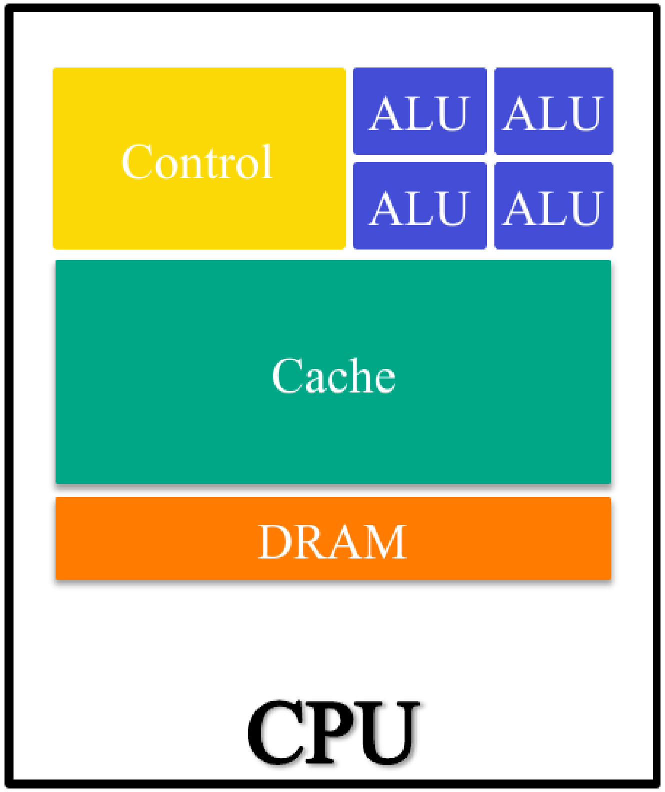
Many integrated core (MIC) Architecture

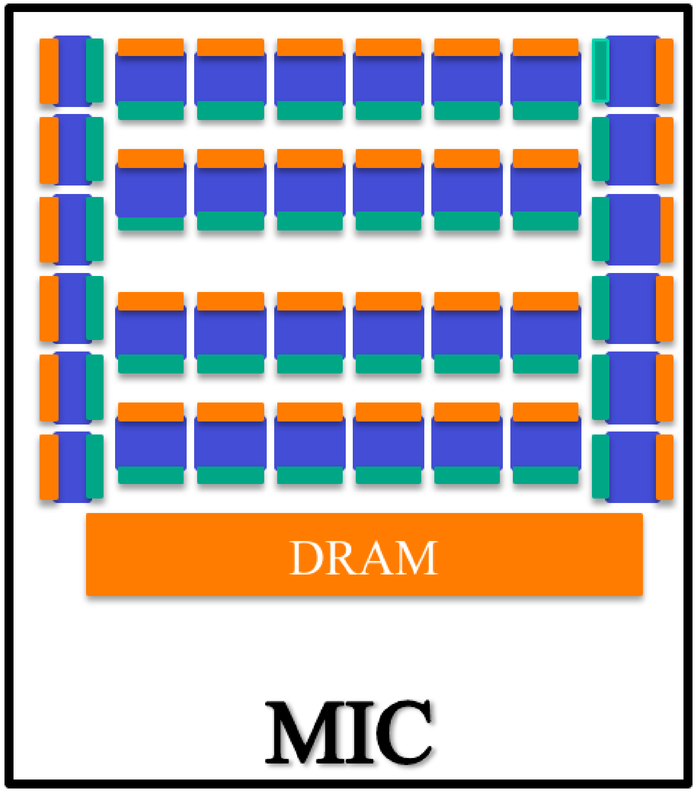
- High bandwidth network interconnect by bidirectional ring topology.

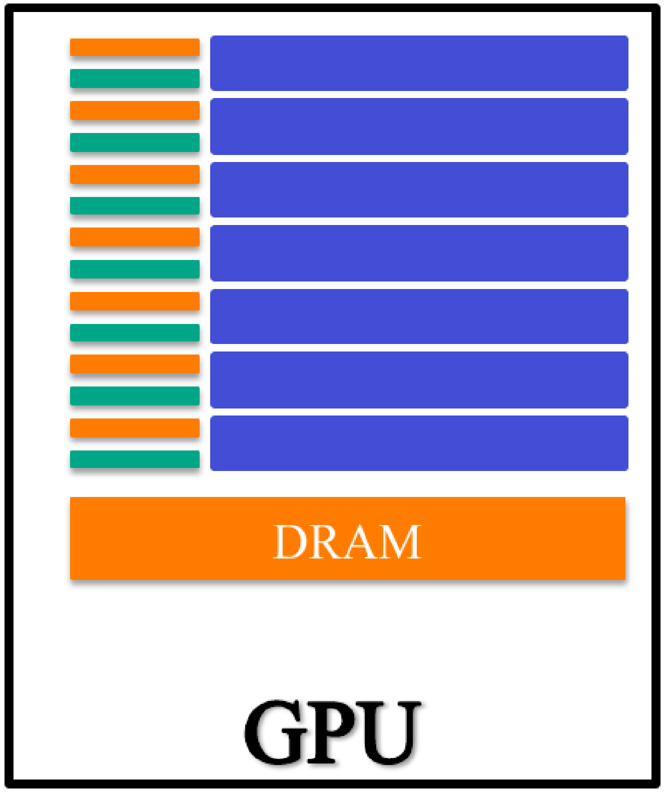
- The ring connects all the 61 cores, L2 caches through a distributed global tag directory (TD), PCIe client logic, GDDR5 memory controllers ...etc.



Architecture comparison







General-purpose architecture

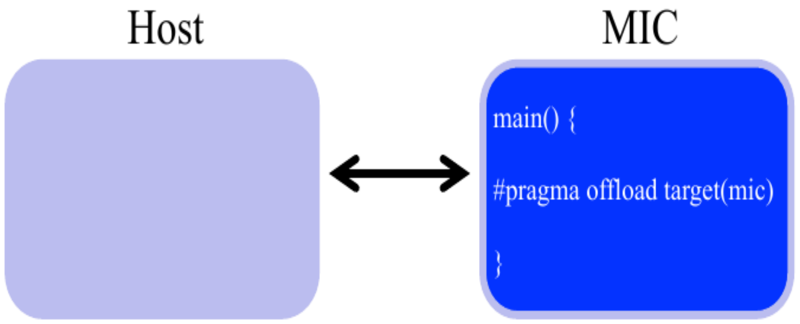
Power-efficient

Multiprocessor X86 design architecture

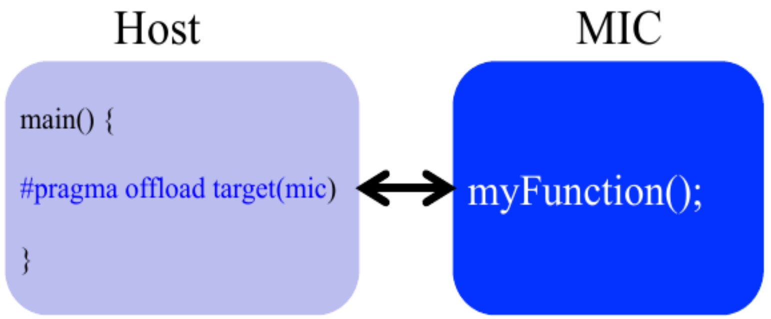
Massively data parallel

MIC programming models

Native Mode
→ Programs started on Xeon Phi.
→ Cross-compilation using –mmic.
→ User access to Xeon Phi is necessary.



Offload to MIC
→ Offload using OpenMP extensions.
→ Automatically offload some routines using MKL.
→ MKL Compiled assisted offload (CAO).
→ MKL automatic Offload (AO).



MPI tasks on Host and MIC
→ Treat the coprocessor like another host.
→ MPI only and MPI + X (X may be OpenMP, TBB, Cilk, OpenCL...etc.).

Advantages of MIC

- Retains programmability and flexibility of standard x86 architecture.

- No need to learn a new complicated language like CUDA or OpenCL.

- Offers possibilities we always missed on GPUs: Login onto the system, watching and controlling processes via top, kill etc. like on a Linux host.

- **Allows many different parallel programming models like OpenMP, MPI, and Intel Threading Building Blocks (TBB)**.

- Offers standard math-libraries like Intel MKL.

- **Supports whole Intel tool chain**, e.g. Intel C/C++ and Fortran Compiler, Debugger & Intel VTune Amplifier.

Offloading OpenMP computations

- C/C++ & OpenMP:

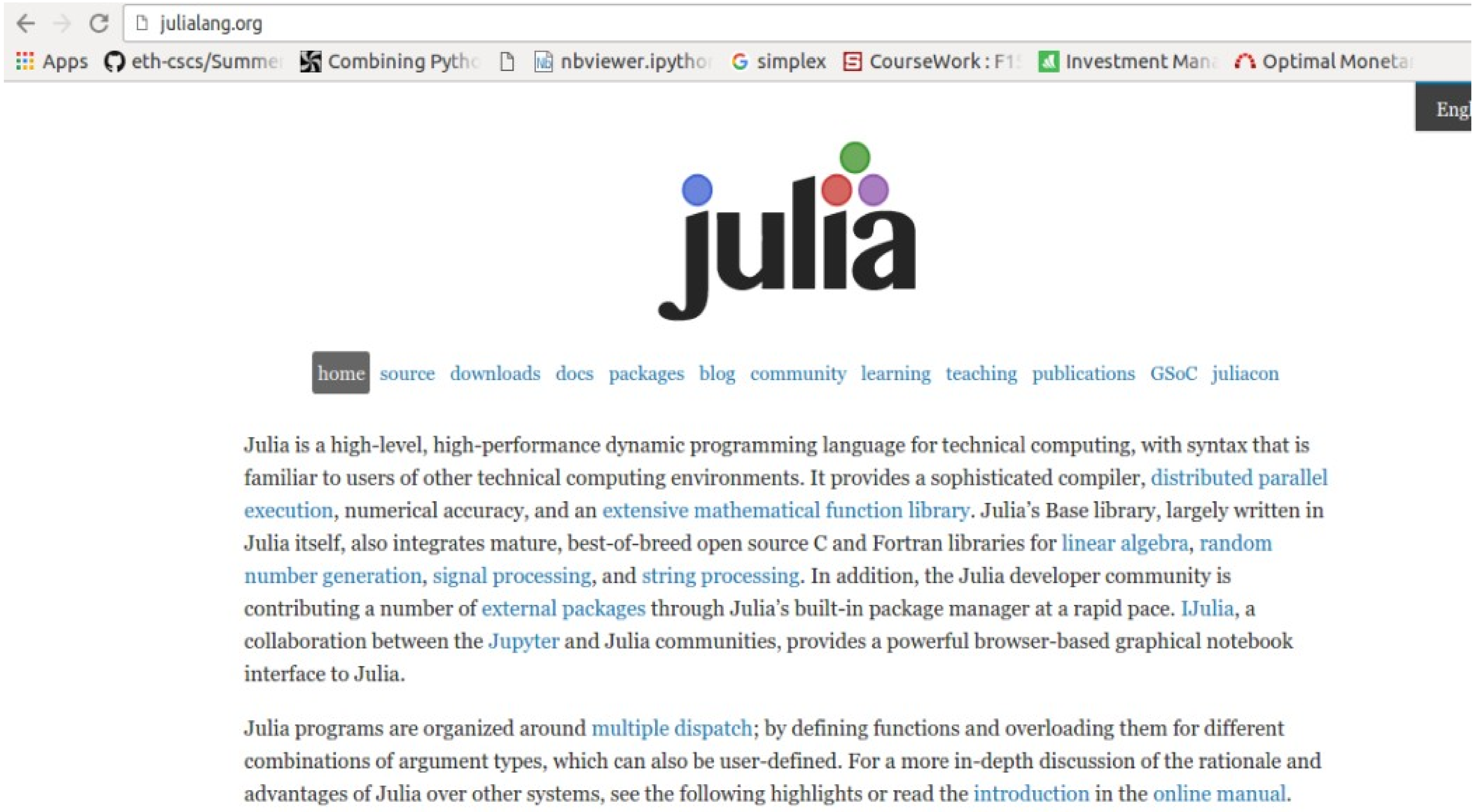
```
#pragma offload target(mic)
#pragma omp parallel for
for (int i=0;i<n;i++) {
        a[i]=c*b[i]+d;
}
```

- Fortran & OpenMP

```
!DIR$ offload target(mic)
!$OMP PARALLEL DO
   do i = 1, n
     a(i) = c*b(i) + d
   end do
!$omp END PARALLEL DO
```

Trending topics

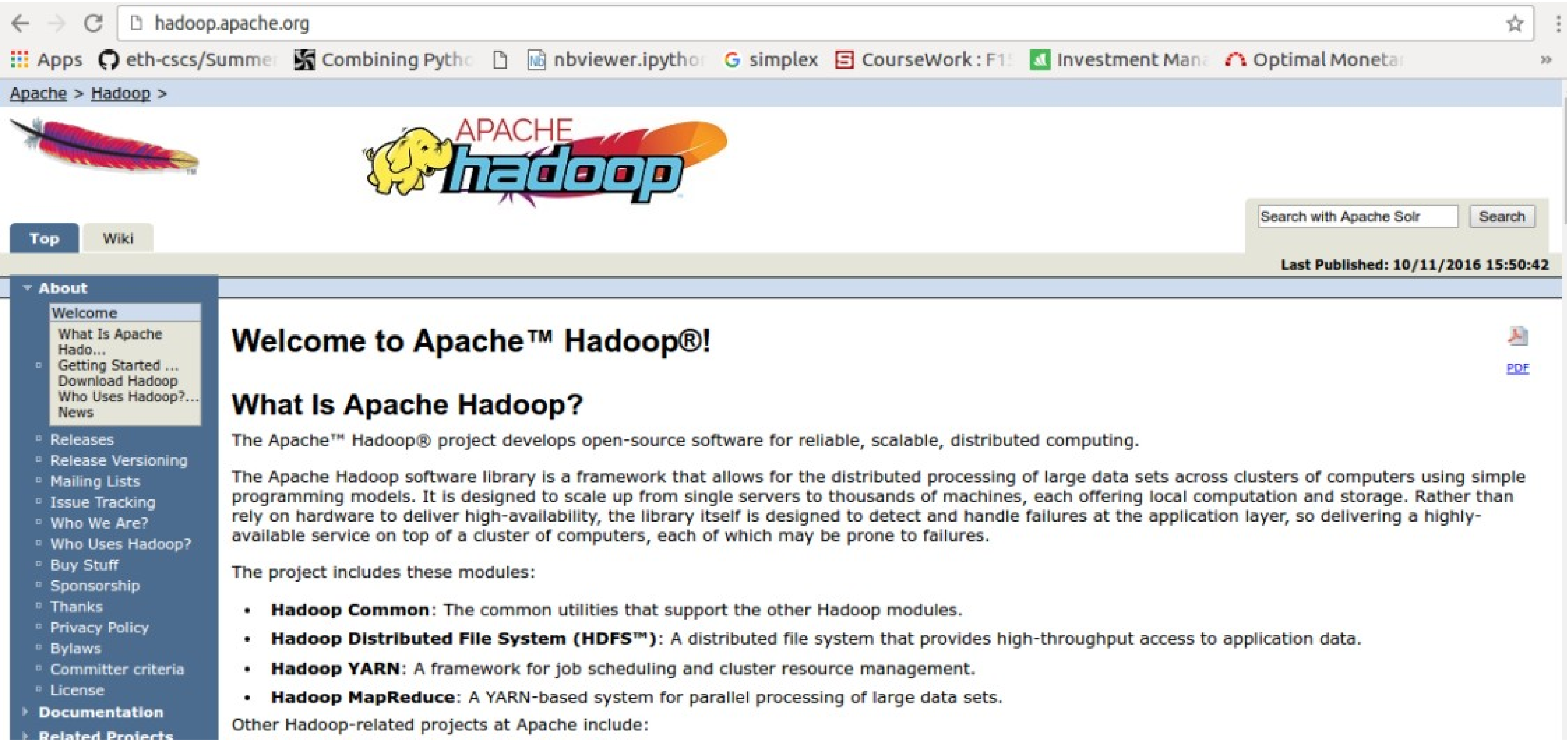
http://julialang.org/learning/
https://en.wikipedia.org/wiki/Julia_(programming_language)



Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Julia's Base library, largely written in Julia itself, also integrates mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing. In addition, the Julia developer community is contributing a number of external packages through Julia's built-in package manager at a rapid pace. IJulia, a collaboration between the Jupyter and Julia communities, provides a powerful browser-based graphical notebook interface to Julia.

Julia programs are organized around multiple dispatch; by defining functions and overloading them for different combinations of argument types, which can also be user-defined. For a more in-depth discussion of the rationale and advantages of Julia over other systems, see the following highlights or read the introduction in the online manual.

Apache Hadoop



Last Published: 10/11/2016 15:50:42

Welcome to Apache™ Hadoop®!

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

Apache Hadoop (2)

https://en.wikipedia.org/wiki/Apache_Hadoop
http://hadoop.apache.org/
https://www.tutorialspoint.com/hadoop/hadoop_introduction.htm

- Apache Hadoop is an open-source software framework for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware.

- All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework.

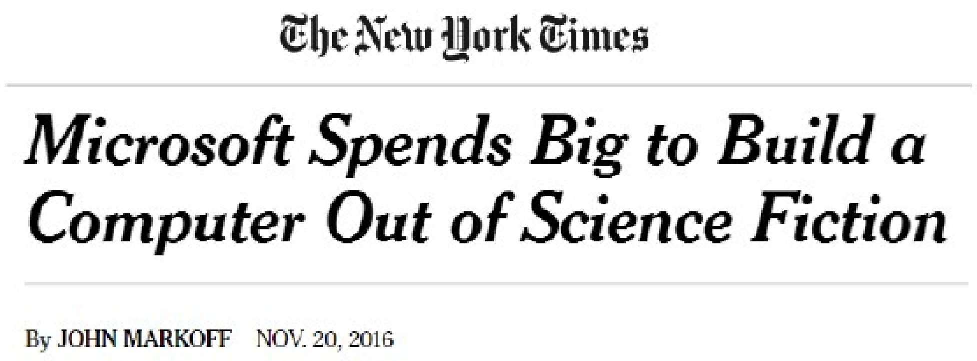
- The core of Apache Hadoop consists of a storage part, known as **Hadoop Distributed File System (HDFS)**, and a processing part called **MapReduce**.

- Hadoop splits files into large blocks and distributes them across nodes in a cluster.

- To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed.

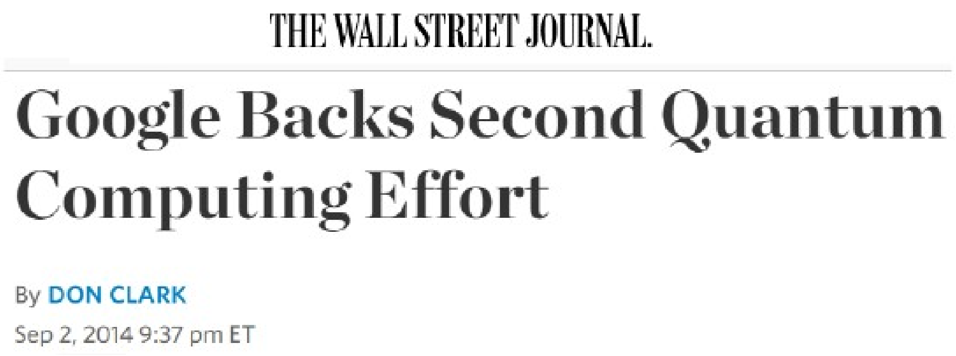
- This approach takes advantage of data locality – nodes manipulating the data they have access to – to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Quantum Computing – industry explosion


The New York Times

Microsoft Spends Big to Build a Computer Out of Science Fiction

By JOHN MARKOFF NOV. 20, 2016


THE WALL STREET JOURNAL.

Google Backs Second Quantum Computing Effort

By DON CLARK
Sep 2, 2014 9:37 pm ET

Companies building quantum computing hardware:

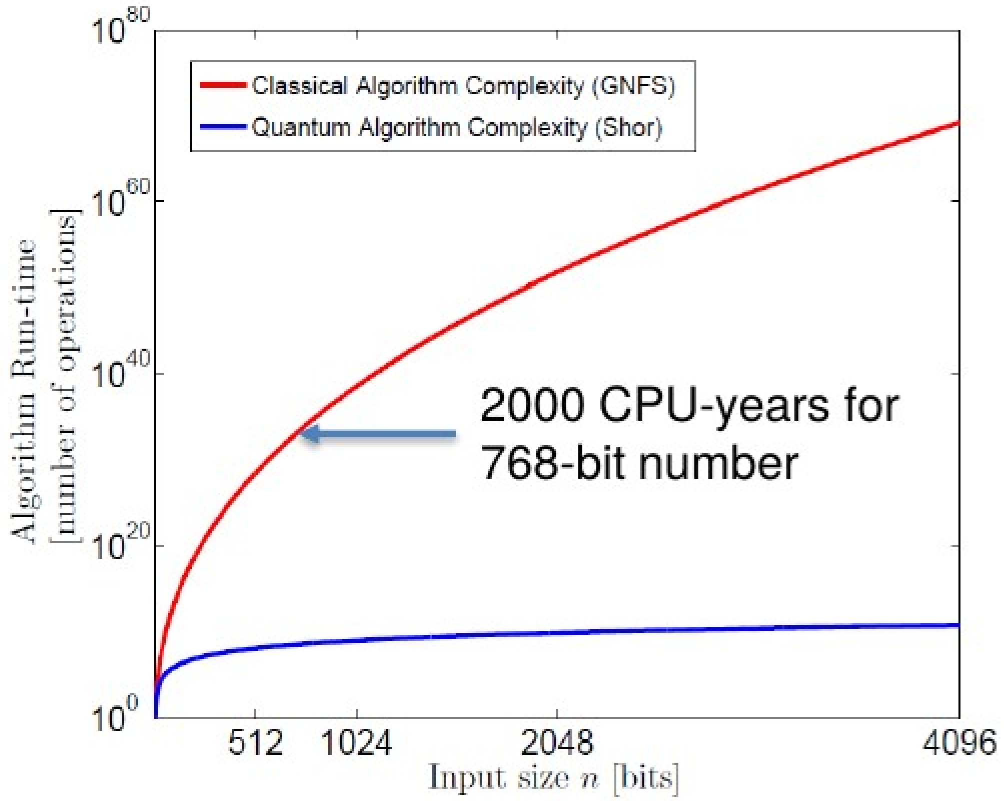
Microsoft IBM Google

rigetti HRL LABORATORIES ION D:WAVE

Slide: courtesy of P. McMahon

The promise of quantum computing

***Vastly* faster computation for some very specific problems.**



What is a quantum computer?

A computer that fundamentally takes advantage of the laws of quantum mechanics to solve problems.

(Quantum mechanics is the theory of how physical systems behave, and is typically relevant only when the system is very small.)

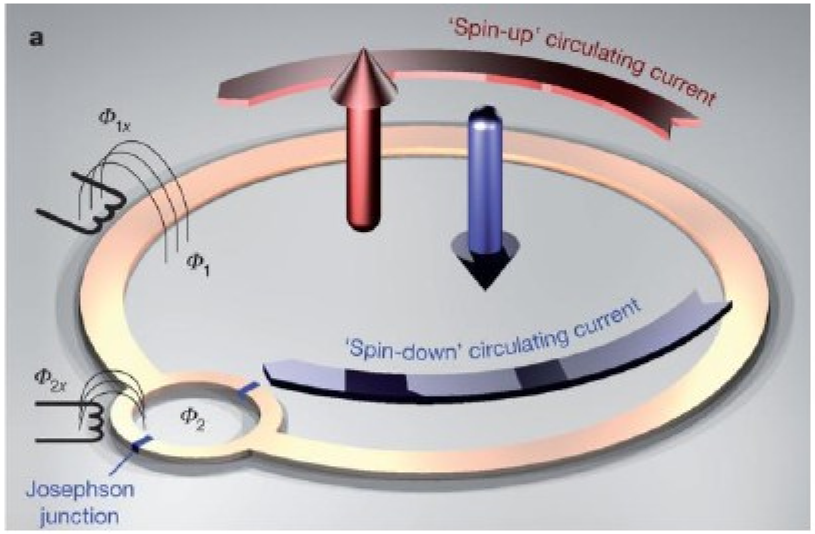

a
'Spin-up' circulating current
Φ_{1x}
Φ_1
'Spin-down' circulating current
Φ_{2x}
Φ_2
Josephson junction

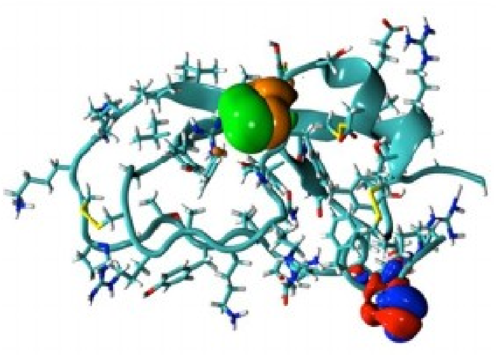
Image credit: M.W. Johnson, et al. *Nature* **473**, 194 (2011)

What is a quantum computer good for?

There are two major applications of quantum computing that the community is very convinced about. There are several others that are more speculative.

Quantum Chemistry

Calculation of chemical properties from first principles.



17% of Oak Ridge National Lab **supercomputer time** is spent on quantum chemistry (source: Juerg Hutter, 2011)

Prime Factoring

Breaking a product of two prime numbers into its prime factors (main use: breaking modern public-key cryptography systems)

15 = 3 x 5

642469 = 601 x 1069

Substantial resources are required for classical algorithms: **2000 CPU-years** for **768-bit** number

Slide: courtesy of P. McMahon

Moving forward

- Quantum computer hardware development now has strong industrial backing (Google, Microsoft, IBM, etc.)
- Google is expected to demonstrate "quantum supremacy" for the first time within the next year using a 49-qubit machine
- Google is aiming to reach 1 million physical qubits by 2027
- IBM will release a 17-qubit machine this year (and has 5-qubit machine available already)
- There are two well-established areas in which QC will give an advantage (chemistry and cryptography breaking)
- Discovering other areas is an active area of research

KEEP
CALM
AND
ACCEPT THE FACT
THAT IT'S OVER
KeepCalmAndPosters.com