# PETSc

## Portable, Extensible Toolkit for Scientific Computation

Karl Rupp
rupp@mcs.anl.gov

Mathematics and Computer Science Division
Argonne National Laboratory

Tutorial at the HPC Symposium 2013

April 10th, 2013

Argonne NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

# Before we start...

## Ask Questions

Tell me if you do not understand

Ask for further details

# Table of Contents

# About PETSc

# PETSc Origins

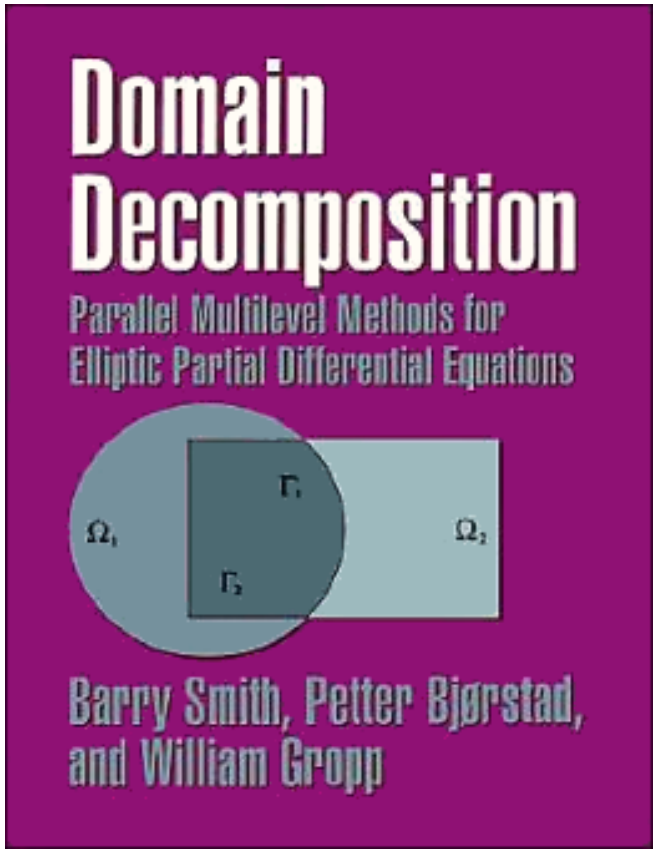
PETSc was developed as a Platform for

**Experimentation**

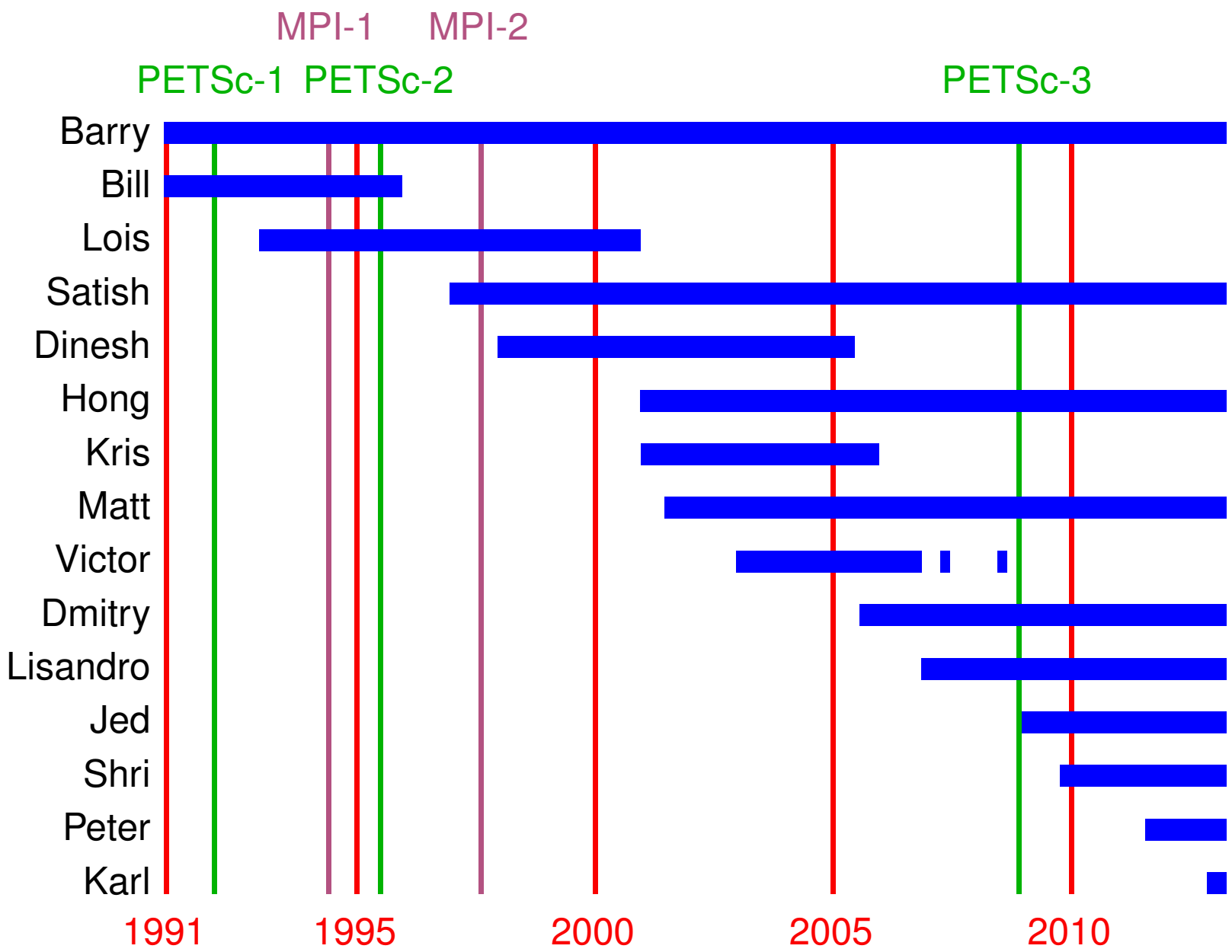
We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms

These boundaries are often blurred...



# Timeline


MPI-1
MPI-2
PETSc-1
PETSc-2
PETSc-3
Barry
Bill
Lois
Satish
Dinesh
Hong
Kris
Matt
Victor
Dmitry
Lisandro
Jed
Shri
Peter
Karl
1991
1995
2000
2005
2010

# PETSc

**Portable** Extensible Toolkit for Scientific Computing

## Architecture

- tightly coupled (e.g. XT5, BG/P, Earth Simulator)
- loosely coupled such as network of workstations
- GPU clusters (many vector and sparse matrix kernels)

## Software Environment

- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Real/complex, single/double/quad precision, 32/64-bit int

## System Size

- 500B unknowns, 75% weak scalability on Jaguar (225k cores) and Jugene (295k cores)
- Same code runs performantly on a laptop

## Free to everyone (BSD-style license), open development

Portable **Extensible** Toolkit for Scientific Computing

## Philosophy: Everything has a plugin architecture

Vectors, Matrices, Coloring/ordering/partitioning algorithms

Preconditioners, Krylov accelerators

Nonlinear solvers, Time integrators

Spatial discretizations/topology*

## Example

Vendor supplies matrix format and associated preconditioner, distributes compiled shared library.

Application user loads plugin at runtime, no source code in sight.

# PETSc

Portable Extensible **Toolkit** for Scientific Computing

## Toolset

- algorithms
- (parallel) debugging aids
- low-overhead profiling

## Composability

- try new algorithms by choosing from product space
- composing existing algorithms (multilevel, domain decomposition, splitting)

## Experimentation

- Impossible to pick the solver *a priori*
- PETSc's response: expose an algebra of composition
- keep solvers decoupled from physics and discretization

# PETSc

Portable Extensible Toolkit for **Scientific Computing**

## Computational Scientists

PyLith (CIG), Underworld (Monash), Magma Dynamics (LDEO, Columbia), PFLOTRAN (DOE), SHARP/UNIC (DOE)

## Algorithm Developers (iterative methods and preconditioning)

## Package Developers

SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM

## Funding


Department of Energy

SciDAC, ASCR ISICLES, MICS Program, INL Reactor Program

National Science Foundation

CIG, CISE, Multidisciplinary Challenge Program


## Documentation and Support

Hundreds of tutorial-style examples

Hyperlinked manual, examples, and manual pages for all routines

Support from `petsc-maint@mcs.anl.gov`

# The Role of PETSc

*Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.*

*PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.*

— Barry Smith

# The Role of PETSc

*You want to think about how you decompose your data structures, how you think about them globally. [...]*

*If you were building a house, you'd start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn't start with a bunch of tiles and say. "Well I'll put this tile down on the ground, and then I'll find a tile to go next to it."*

*But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you're going to survive making your code parallel.*

— Bill Gropp
— http://www.rce-cast.com/Podcast/rce-28-mpich2.html

# First Steps

## Obtaining PETSc

http://mcs.anl.gov/petsc, download tarball

Linux Package Managers

Git: https://bitbucket.org/petsc/petsc

Mercurial: https://bitbucket.org/petsc/petsc-hg

## Installing PETSc

```
$> export PETSC_DIR=$PWD PETSC_ARCH=mpich-gcc-dbg
```

```
$> ./configure --with-shared-libraries
               --with-blas-lapack-dir=/usr
               --download-{mpich,ml,hypre}
```

```
$> make all test
```

# PETSc External Packages

## Most packages can be automatically

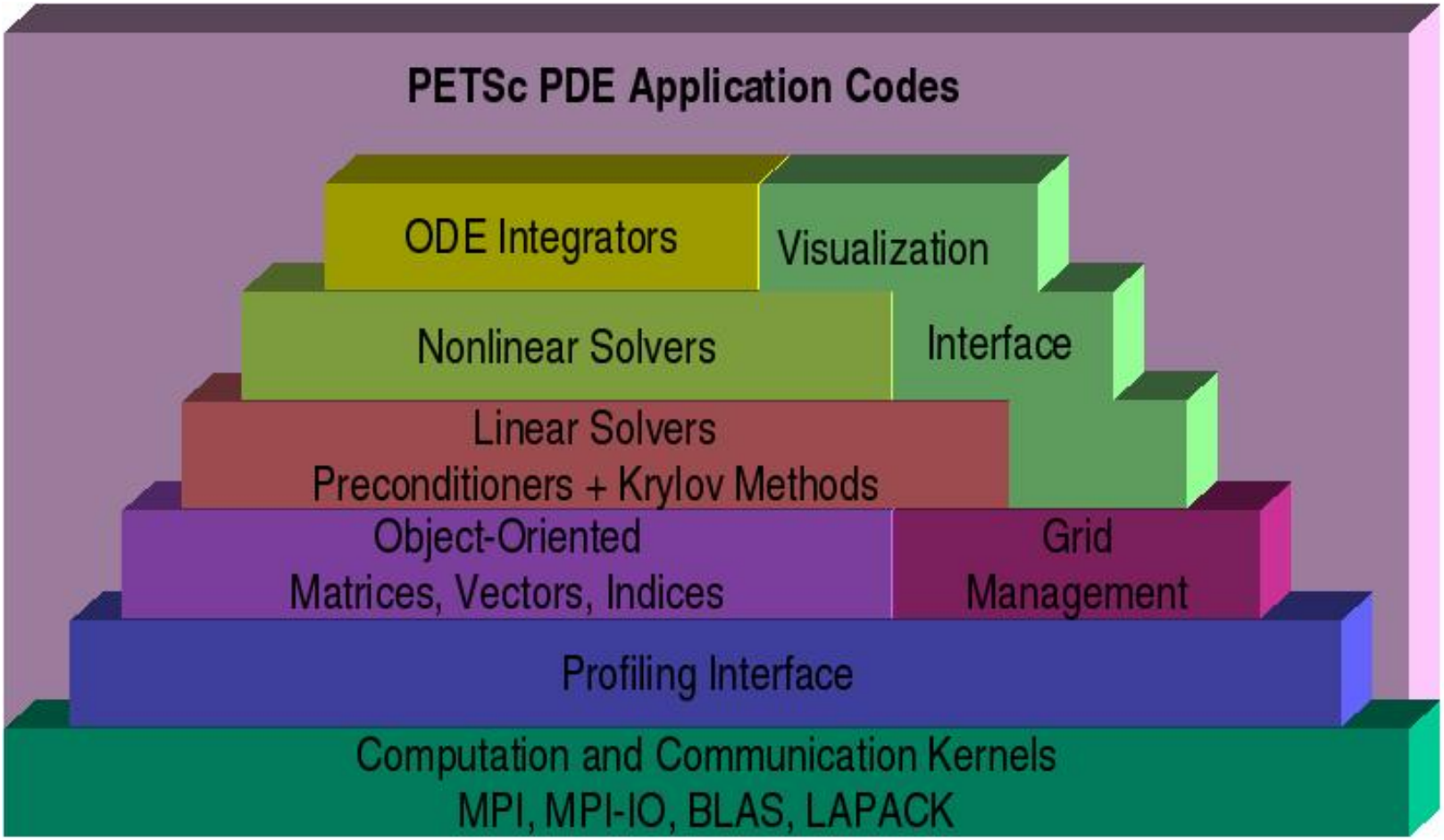
- Downloaded
- Configured and Built (in `$PETSC_DIR/externalpackages`)
- Installed with PETSc

## Currently works for

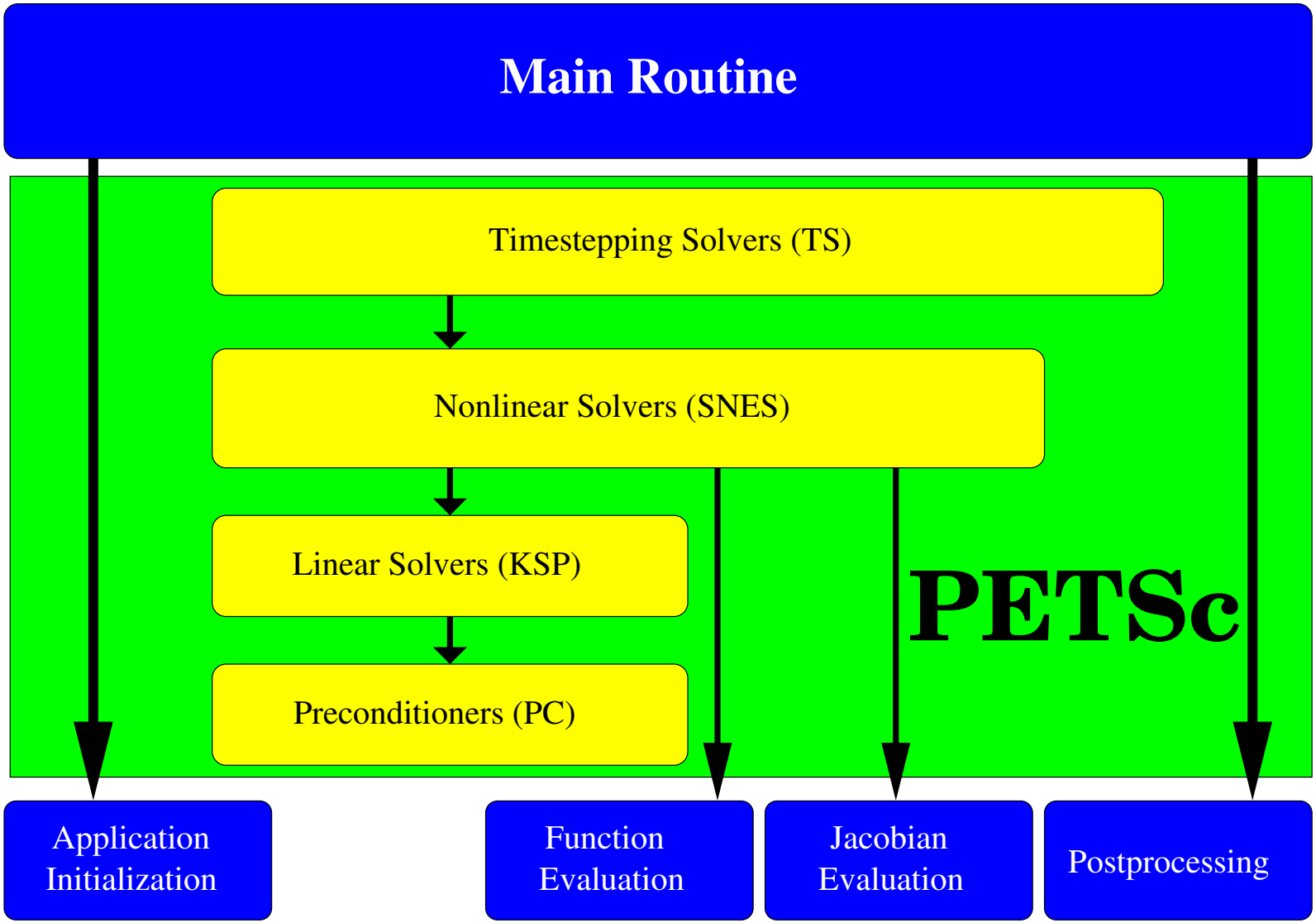
- petsc4py
- PETSc documentation utilities (Sowing, lgrind, c2html)
- BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
- MPICH, MPE, OpenMPI
- ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
- MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
- PaStiX, BLOPEX, FFTW, SPRNG
- Prometheus, HYPRE, ML, SPAI
- Sundials
- Triangle, TetGen, FIAT, FFC, Generator
- HDF5, Boost

# PETSc Pyramid

## PETSc Structure



# Flow Control for a PETSc Application



# PETSc Objects

## Sample Code

```
Mat A;
PetscInt m,n,M,N;
MatCreate(comm,&A);
MatSetSizes(A,m,n,M,N);        /* or PETSC_DECIDE */
MatSetOptionsPrefix(A,"foo_");
MatSetFromOptions(A);
/* Use A */
MatView(A,PETSC_VIEWER_DRAW_WORLD);
MatDestroy(A);
```

## Remarks

`Mat` is an opaque object (pointer to incomplete type)

- Assignment, comparison, etc, are cheap

What's up with this "Options" stuff?

- We will discuss this later...

# Basic PetscObject Usage

Every object in PETSc supports a basic interface

| Function | Operation |
|---|---|
| `Create()` | create the object |
| `Get/SetName()` | name the object |
| `Get/SetType()` | set the implementation type |
| `Get/SetOptionsPrefix()` | set the prefix for all options |
| `SetFromOptions()` | customize object from command line |
| `SetUp()` | perform other initialization |
| `View()` | view the object |
| `Destroy()` | cleanup object allocation |

Also, all objects support the `-help` option.

# PETSc Options

## Ways to set options

- Command line
- Filename in the third argument of `PetscInitialize()`
- `~/.petscrc`
- `$PWD/.petscrc`
- `$PWD/petscrc`
- `PetscOptionsInsertFile()`
- `PetscOptionsInsertString()`
- `PETSC_OPTIONS` environment variable
- command line option `-options_file [file]`

# PETSc Options

## Example of Command Line Control

```
$> ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7
     -snes_monitor -{ksp,snes}_converged_reason
     -snes_view
$> ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7
     -snes_monitor -{ksp,snes}_converged_reason
     -snes_view -mat_view_draw -draw_pause 0.5
$> ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7
     -snes_monitor -{ksp,snes}_converged_reason
     -snes_view -mat_view_draw -draw_pause 0.5
     -pc_type lu -pc_factor_mat_ordering_type natural
```

Use `-help` to find other ordering types

# Application Integration

# Application Integration

## Be willing to experiment with algorithms

No optimality without interplay between physics and algorithmics

## Adopt flexible, extensible programming

Algorithms and data structures not hardwired

## Be willing to play with the real code

Toy models have limited usefulness

But make test cases that run quickly

## If possible, profile before integration

Automatic in PETSc

# Incorporating PETSc into Existing Codes

PETSc does not seize `main()`, does not control output

Propogates errors from underlying packages, flexible

Nothing special about `MPI_COMM_WORLD`

Can wrap existing data structures/algorithms

- `MatShell`, `PCShell`, full implementations
- `VecCreateMPIWithArray()`
- `MatCreateSeqAIJWithArrays()`
- Use an existing semi-implicit solver as a preconditioner
- Usually worthwhile to use native PETSc data structures unless you have a good reason not to

Uniform interfaces across languages

- C, C++, Fortran 77/90, Python, MATLAB

Do not have to use high level interfaces (e.g. SNES, TS, DM)

- but PETSc can offer more if you do, like MFFD and SNES Test

# Integration Stages

## Version Control

It is impossible to overemphasize

## Initialization

Linking to PETSc

## Profiling

Profile before changing

Also incorporate command line processing

## Linear Algebra

First PETSc data structures

## Solvers

Very easy after linear algebra is integrated

# Initialization

## Call `PetscInitialize()`

- Setup static data and services
- Setup MPI if it is not already
- Can set `PETSC_COMM_WORLD` to use your communicator
  (can always use subcommunicators for each object)

## Call `PetscFinalize()`

- Calculates logging summary
- Can check for leaks/unused options
- Shutdown and release resources

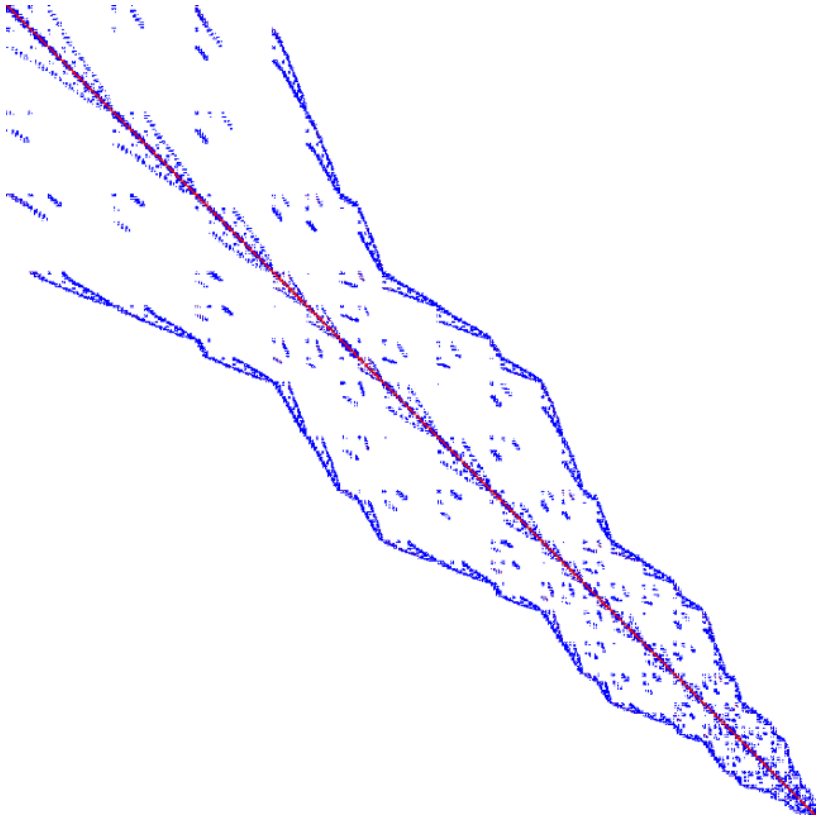
## Can only initialize PETSc once

## Sparse Matrices

**The** important data type when solving PDEs

Two main phases:

- Filling with entries (assembly)
- Application of its action (e.g. SpMV)



# Matrix Memory Preallocation

## PETSc sparse matrices are dynamic data structures

- can add additional nonzeros freely

## Dynamically adding many nonzeros

- requires additional memory allocations
- requires copies
- can kill performance

## Memory preallocation provides

- the freedom of dynamic data structures
- good performance

## Easiest solution is to replicate the assembly code

- Remove computation, but preserve the indexing code
- Store set of columns for each row

## Call preallocation routines for all datatypes

- `MatSeqAIJSetPreallocation()`
- `MatMPIBAIJSetPreallocation()`
- Only the relevant data will be used

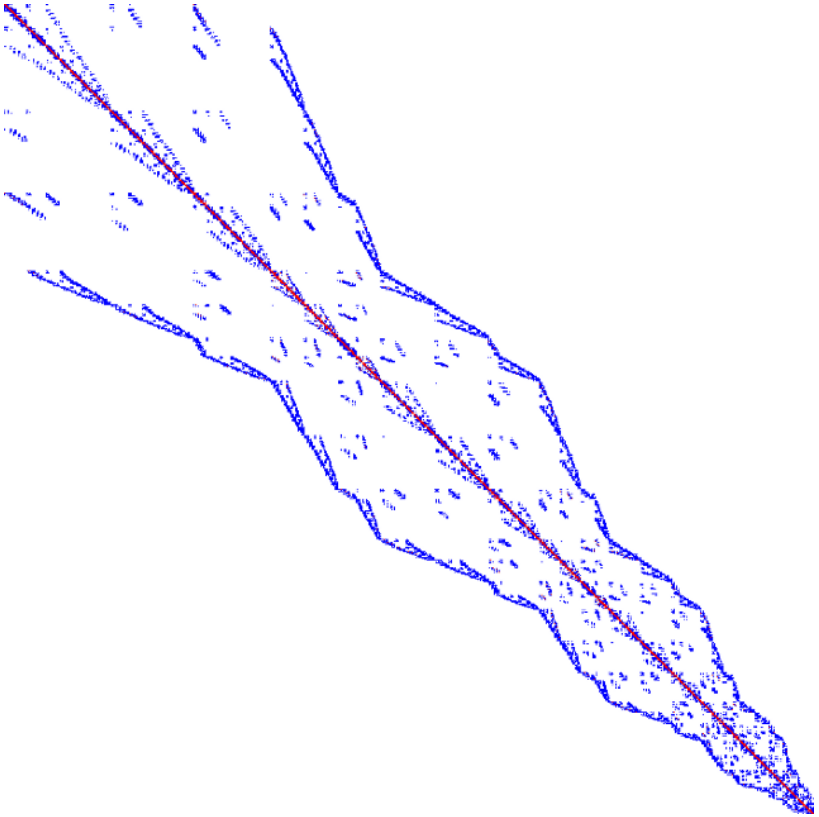
# PETSc Application Integration

## Sequential Sparse Matrices

```
MatSeqAIJSetPreallocation(Mat A, int nz, int nnz[])
```

nz: expected number of nonzeros in any row

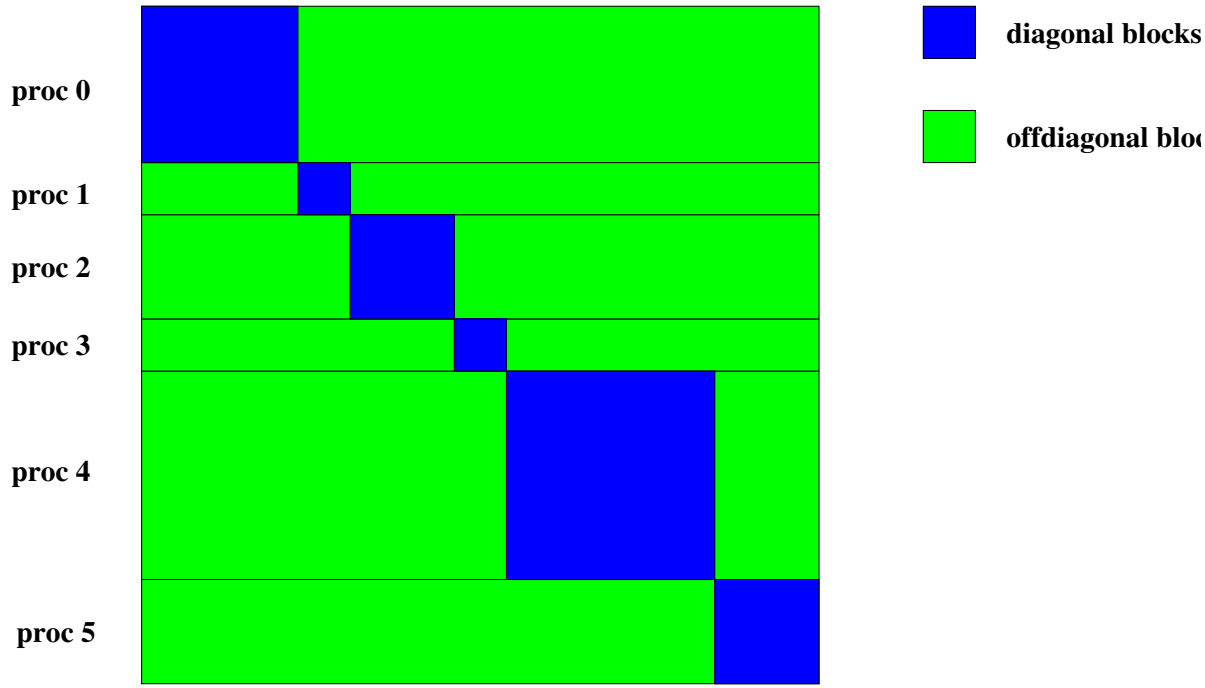
nnz(i): expected number of nonzeros in row i



## Parallel Sparse Matrix

Each process locally owns a submatrix of contiguous global rows

Each submatrix consists of diagonal and off-diagonal parts




```
MatGetOwnershipRange(Mat A,int *start,int *end)
```

`start`: first locally owned row of global matrix
`end-1`: last locally owned row of global matrix

# PETSc Application Integration

## Parallel Sparse Matrix

```
MatMPIAIJSetPreallocation(Mat A, int dnz, int dnnz[],
                                 int onz, int onnz[]
```

dnz: expected number of nonzeros in any row in the diagonal block

dnnz(i): expected number of nonzeros in row i in the diagonal block

onz: expected number of nonzeros in any row in the offdiagonal portion

onnz(i): expected number of nonzeros in row i in the offdiagonal portion

## Verifying Preallocation

Use runtime options

```
-mat_new_nonzero_location_err
-mat_new_nonzero_allocation_err
```

Use runtime option

```
-info
```

Output:

```
[proc #] Matrix size: %d X %d; storage space: %d unneeded, %d used
[proc #] Number of mallocs during MatSetValues( )  is %d
```

```
[merlin] mpirun ex2 -log_info
[0]MatAssemblyEnd_SeqAIJ:Matrix size: 56 X 56; storage space:
[0]   310 unneeded, 250 used
[0]MatAssemblyEnd_SeqAIJ:Number of mallocs during MatSetValues() is 0
[0]MatAssemblyEnd_SeqAIJ:Most nonzeros in any row is 5
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routines
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routines
Norm of error 0.000156044 iterations 6
[0]PetscFinalize:PETSc successfully ended!
```

# Block and Symmetric Formats

## BAIJ

Like AIJ, but uses static block size

Preallocation is like AIJ, but just one index per block

## SBAIJ

Only stores upper triangular part

Preallocation needs number of nonzeros in upper triangular parts of on- and off-diagonal blocks

## MatSetValuesBlocked()

Better performance with blocked formats

Also works with scalar formats, if `MatSetBlockSize()` was called

Variants `MatSetValuesBlockedLocal()`, `MatSetValuesBlockedStencil()`

Change matrix format at runtime, don't need to touch assembly code

# Matrices

## Definition (Matrix)

A matrix is a linear transformation between finite dimensional vector spaces.

## Definition (Forming a matrix)

Forming or assembling a matrix means defining it's action in terms of entries (usually stored in a sparse format).

## Important Matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
4. Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

## Important Matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
4. Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

These matrices are dense. Never form them.

## Important Matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
4. Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

These are not very sparse. Don't form them.

## Important Matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
4. Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

None of these matrices “have entries”

# Matrices

*What can we do with a matrix that doesn't have entries?*

## Krylov solvers for $Ax = b$

Krylov subspace: $\{b, Ab, A^2b, A^3b, \dots\}$

Convergence rate depends on the spectral properties of the matrix

For any popular Krylov method $\mathcal{K}$, there is a matrix of size $m$, such that $\mathcal{K}$ outperforms all other methods by a factor at least $\mathcal{O}(\sqrt{m})$ [Nachtigal et. al., 1992]

## Typically...

The action $y \leftarrow Ax$ can be computed in $\mathcal{O}(m)$

Aside from matrix multiply, the $n^{\text{th}}$ iteration requires at most $\mathcal{O}(mn)$

# GMRES

## Brute force minimization of residual in $\{b, Ab, A^2b, \dots\}$

1. Use Arnoldi to orthogonalize the $n$th subspace, producing

$$AQ_n = Q_{n+1}H_n$$

2. Minimize residual in this space by solving the overdetermined system

$$H_n y_n = e_1^{(n+1)}$$

using $QR$-decomposition, updated cheaply at each iteration.

## Properties

Converges in $n$ steps for all right hand sides if there exists a polynomial of degree $n$ such that $\|p_n(A)\| < \mathit{tol}$ and $p_n(0) = 1$.

Residual is monotonically decreasing, robust in practice

Restarted variants are used to bound memory requirements

## Linear Solvers - Krylov Methods

Using PETSc linear algebra, just add:

```
KSPSetOperators(KSP ksp, Mat A, Mat M, MatStructure flag)
KSPSolve(KSP ksp, Vec b, Vec x)
```

Can access subobjects

```
KSPGetPC(KSP ksp, PC *pc)
```

Preconditioners must obey PETSc interface

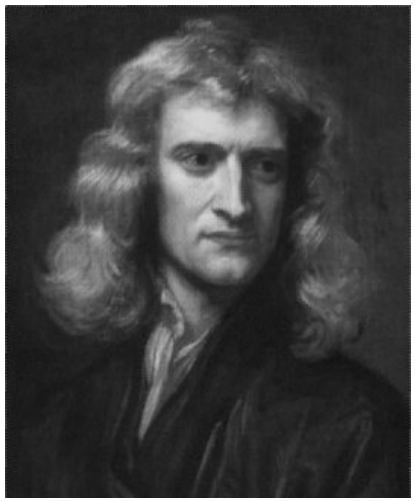
- Basically just the KSP interface

Can change solver dynamically from the command line, `-ksp_type`

# Newton iteration: workhorse of SNES

## Standard form of a nonlinear system

$$F(u) = 0$$



## Iteration

Solve: $J(u)w = -F(u)$

Update: $u^+ \leftarrow u + w$

Quadratically convergent near a root: $|u^{n+1} - u^*| \in \mathcal{O}\left(|u^n - u^*|^2\right)$

Picard is the same operation with a different $J(u)$

## Nonlinear Solvers - Newton and Picard Methods

Using PETSc linear algebra, just add:

```
SNESSetFunction(SNES snes, Vec r, residualFunc, void *ctx)
SNESSetJacobian(SNES snes, Mat A, Mat M, jacFunc, void *ctx)
SNESSolve(SNES snes, Vec b, Vec x)
```

Can access subobjects

```
SNESGetKSP(SNES snes, KSP *ksp)
```

Can customize subobjects from the cmd line

Set the subdomain preconditioner to ILU with `-sub_pc_type ilu`

# Profiling

## Profiling

- Use `-log_summary` for a performance profile
  - Event timing
  - Event flops
  - Memory usage
  - MPI messages
- Call `PetscLogStagePush()` and `PetscLogStagePop()`
  - User can add new stages
- Call `PetscLogEventBegin()` and `PetscLogEventEnd()`
  - User can add new events
- Call `PetscLogFlops()` to include your flops

## Reading -log_summary

```
                         Max       Max/Min        Avg      Total
Time (sec):           1.548e+02     1.00122   1.547e+02
Objects:              1.028e+03     1.00000   1.028e+03
Flops:                1.519e+10     1.01953   1.505e+10  1.204e+11
Flops/sec:            9.814e+07     1.01829   9.727e+07  7.782e+08
MPI Messages:         8.854e+03     1.00556   8.819e+03  7.055e+04
MPI Message Lengths:  1.936e+08     1.00950   2.185e+04  1.541e+09
MPI Reductions:       2.799e+03     1.00000
```

- Also a summary per stage
- Memory usage per stage (based on when it was allocated)
- Time, messages, reductions, balance, flops per event per stage
- Always send `-log_summary` when asking performance questions on mailing list

# PETSc Profiling

```
Event                Count      Time (sec)     Flops                             --- Global ---  --- S
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %T %l
------------------------------------------------------------------------------------------------------
--- Event Stage 1: Full solve
VecDot                43 1.0 4.8879e-02 8.3 1.77e+06 1.0 0.0e+00 0.0e+00 4.3e+01  0  0  0  0  0   0  0
VecMDot             1747 1.0 1.3021e+00 4.6 8.16e+07 1.0 0.0e+00 0.0e+00 1.7e+03  0  1  0  0 14   1  1
VecNorm             3972 1.0 1.5460e+00 2.5 8.48e+07 1.0 0.0e+00 0.0e+00 4.0e+03  0  1  0  0 31   1  1
VecScale            3261 1.0 1.6703e-01 1.0 3.38e+07 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0
VecScatterBegin     4503 1.0 4.0440e-01 1.0 0.00e+00 0.0 6.1e+07 2.0e+03 0.0e+00  0  0 50 26  0   0  0
VecScatterEnd       4503 1.0 2.8207e+00 6.4 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0
MatMult             3001 1.0 3.2634e+01 1.1 3.68e+09 1.1 4.9e+07 2.3e+03 0.0e+00 11 22 40 24  0  22 44
MatMultAdd           604 1.0 6.0195e-01 1.0 5.66e+07 1.0 3.7e+06 1.3e+02 0.0e+00  0  0  3  0  0   0  1
MatMultTranspose     676 1.0 1.3220e+00 1.6 6.50e+07 1.0 4.2e+06 1.4e+02 0.0e+00  0  0  3  0  0   1  1
MatSolve            3020 1.0 2.5957e+01 1.0 3.25e+09 1.0 0.0e+00 0.0e+00 0.0e+00  9 21  0  0  0  18 41
MatCholFctrSym         3 1.0 2.8324e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0
MatCholFctrNum        69 1.0 5.7241e+00 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 0.0e+00  2  4  0  0  0   4  5
MatAssemblyBegin     119 1.0 2.8250e+00 1.5 0.00e+00 0.0 2.1e+06 5.4e+04 3.1e+02  1  0  2 24  2   2  0
MatAssemblyEnd       119 1.0 1.9689e+00 1.4 0.00e+00 0.0 2.8e+05 1.3e+03 6.8e+01  1  0  0  0  1   1  0
SNESSolve              4 1.0 1.4302e+02 1.0 8.11e+09 1.0 6.3e+07 3.8e+03 6.3e+03 51 50 52 50 50  99100
SNESLineSearch        43 1.0 1.5116e+01 1.0 1.05e+08 1.1 2.4e+06 3.6e+03 1.8e+02  5  1  2  2  1  10  1
SNESFunctionEval      55 1.0 1.4930e+01 1.0 0.00e+00 0.0 1.8e+06 3.3e+03 8.0e+00  5  0  1  1  0  10  0
SNESJacobianEval      43 1.0 3.7077e+01 1.0 7.77e+06 1.0 4.3e+06 2.6e+04 3.0e+02 13  0  4 24  2  26  0
KSPGMRESOrthog      1747 1.0 1.5737e+00 2.9 1.63e+08 1.0 0.0e+00 0.0e+00 1.7e+03  1  1  0  0 14   1  2
KSPSetup             224 1.0 2.1040e-02 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 3.0e+01  0  0  0  0  0   0  0
KSPSolve              43 1.0 8.9988e+01 1.0 7.99e+09 1.0 5.6e+07 2.0e+03 5.8e+03 32 49 46 24 46  62 9
PCSetUp              112 1.0 1.7354e+01 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 8.7e+01  6  4  0  0  1  12  1
PCSetUpOnBlocks     1208 1.0 5.8182e+00 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 8.7e+01  2  4  0  0  1   4  1
PCApply              276 1.0 7.1497e+01 1.0 7.14e+09 1.0 5.2e+07 1.8e+03 5.1e+03 25 44 42 20 41  49 88
```

## Communication Costs

Reductions: usually part of Krylov method, latency limited

- `VecDot`
- `VecMDot`
- `VecNorm`
- `MatAssemblyBegin`

Change algorithm (e.g. IBCGS)

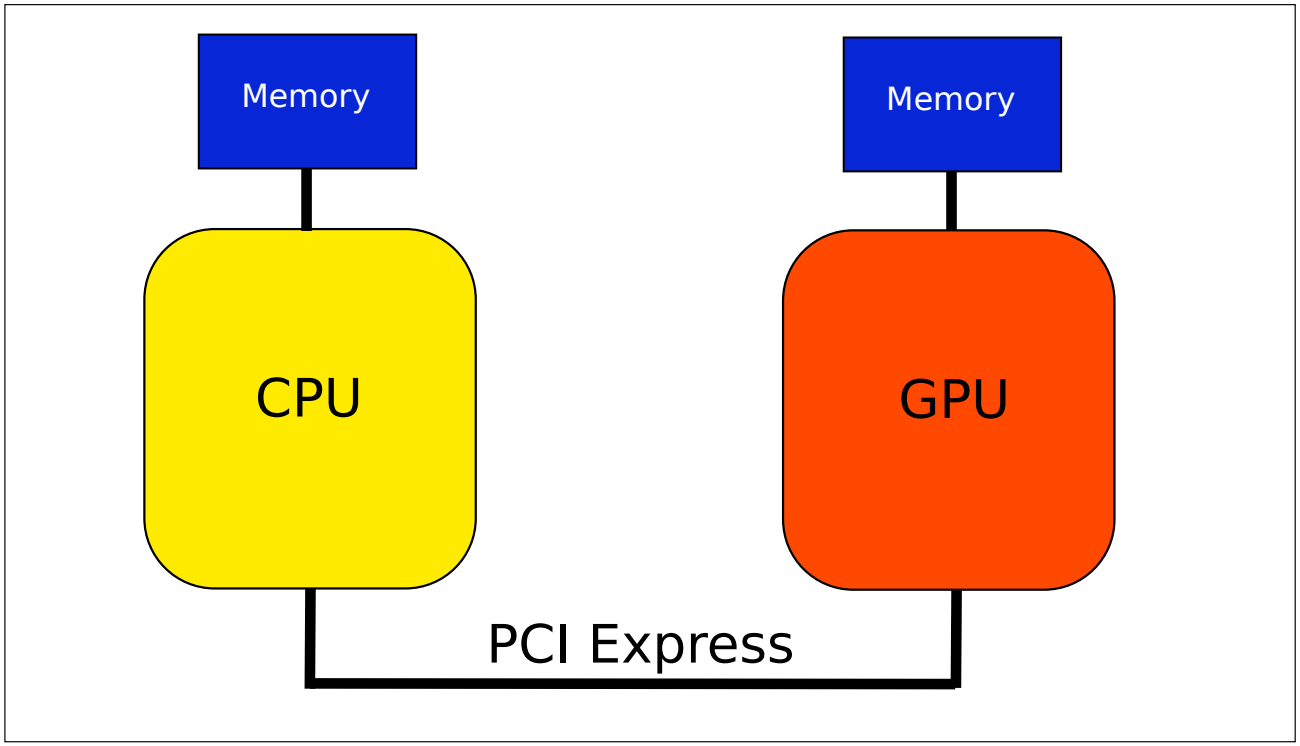
Point-to-point (nearest neighbor), latency or bandwidth

- `VecScatter`
- `MatMult`
- `PCApply`
- `MatAssembly`
- `SNESFunctionEval`
- `SNESJacobianEval`

Compute subdomain boundary fluxes redundantly
Ghost exchange for all fields at once
Better partition

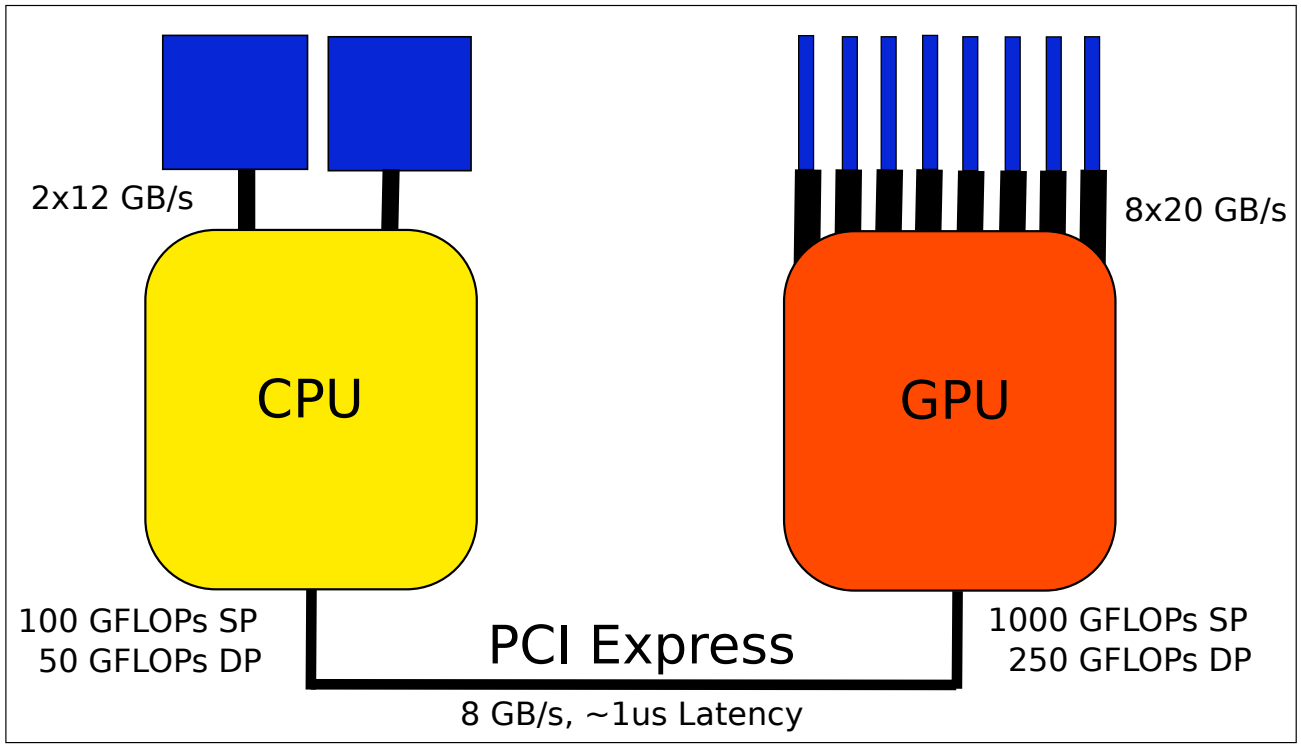
# PETSc and GPUs

## Computing Architecture Schematic



# GPUs: Disillusion

## Computing Architecture Schematic




Good for large FLOP-intensive tasks, high memory bandwidth

PCI-Express can be a bottleneck

$\gg$ 10-fold speedups (usually) not backed by hardware

# GPU Programming Approaches

## CUDA

- Almost no additional code required
- Vendor-lock
- Relies on `nvcc` being available

## OpenCL

- Additional boilerplate code required (low-level API)
- Broad hardware support (separate SDKs)
- No more development effort from NVIDIA

## Directives

- Annotate existing code with OpenMP-style Pragmas
- OpenACC and others

# PETSc GPU Support

## NVIDIA Cusp/Thrust/CUSPARSE

Compile PETSc with CUDA support

Use command line options to enable types, e.g.

```
-vec_type cusp -mat_type aijcusp
```

## ViennaCL (OpenCL)

Compile PETSc with OpenCL support

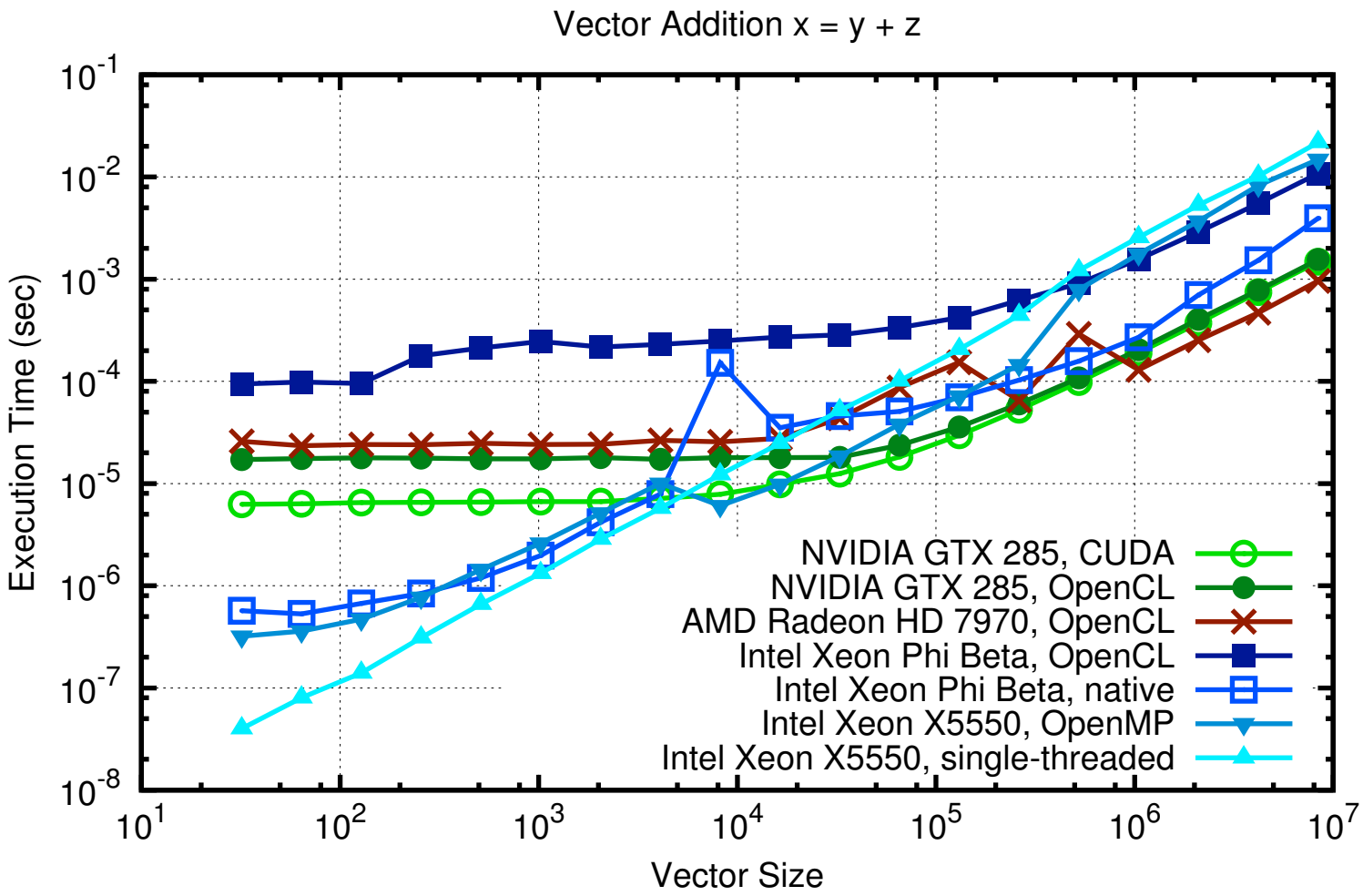
Use command line options to enable types, e.g.

```
-vec_type viennacl -mat_type aijviennacl
```

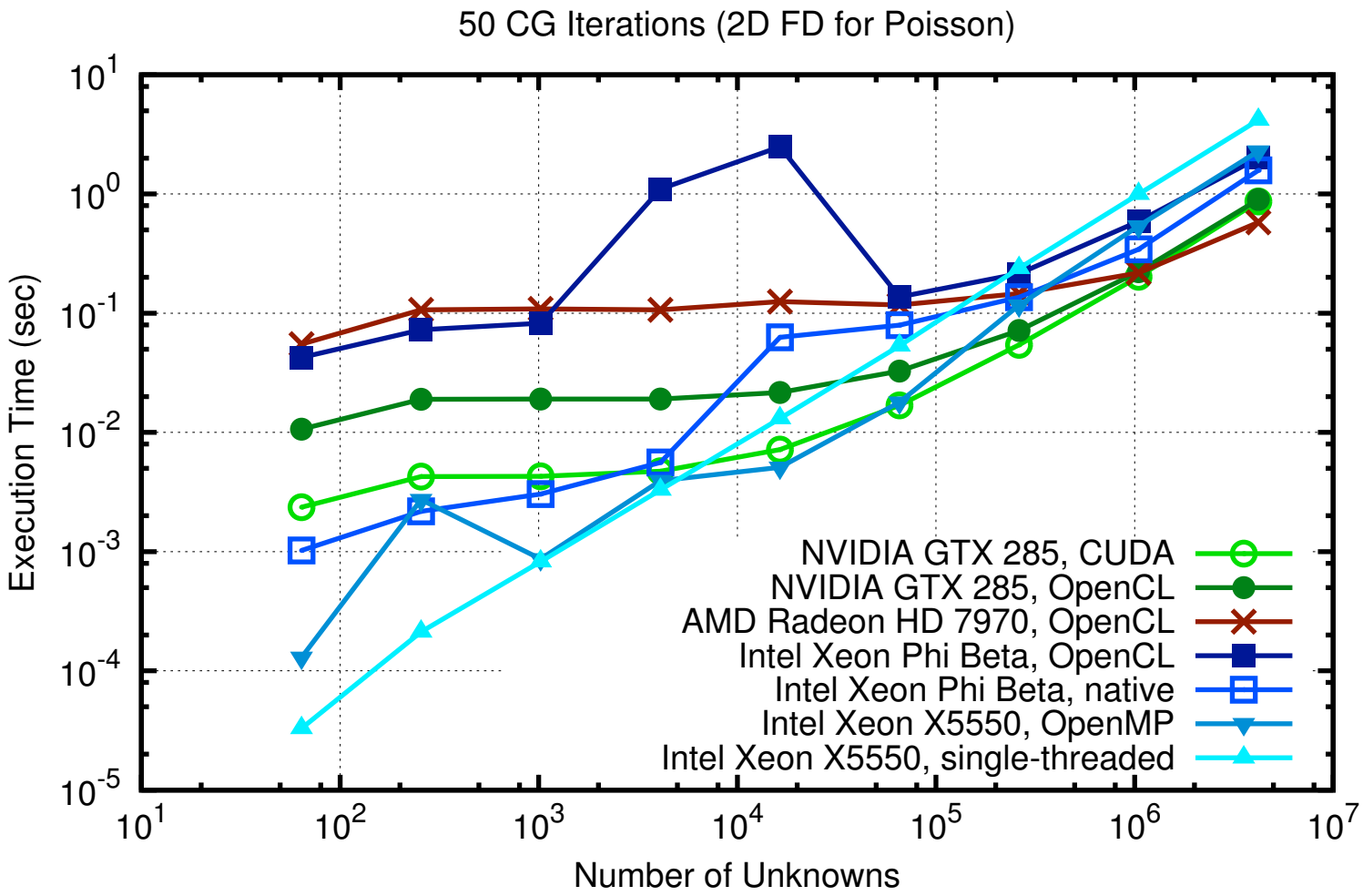
Used for subsequent benchmarks

No change in application code required!

# Benchmarks


Vector Addition x = y + z
Execution Time (sec)
Vector Size
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
$10^{-5}$
$10^{-6}$
$10^{-7}$
$10^{-8}$
$10^{1}$
$10^{2}$
$10^{3}$
$10^{4}$
$10^{5}$
$10^{6}$
$10^{7}$
NVIDIA GTX 285, CUDA
NVIDIA GTX 285, OpenCL
AMD Radeon HD 7970, OpenCL
Intel Xeon Phi Beta, OpenCL
Intel Xeon Phi Beta, native
Intel Xeon X5550, OpenMP
Intel Xeon X5550, single-threaded

# Benchmarks


50 CG Iterations (2D FD for Poisson)
Execution Time (sec)
Number of Unknowns
$10^{1}$
$10^{0}$
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
$10^{-5}$
$10^{1}$
$10^{2}$
$10^{3}$
$10^{4}$
$10^{5}$
$10^{6}$
$10^{7}$
NVIDIA GTX 285, CUDA
NVIDIA GTX 285, OpenCL
AMD Radeon HD 7970, OpenCL
Intel Xeon Phi Beta, OpenCL
Intel Xeon Phi Beta, native
Intel Xeon X5550, OpenMP
Intel Xeon X5550, single-threaded

# Conclusions

## PETSc can help You

solve algebraic and DAE problems in your application area

rapidly develop efficient parallel code, can start from examples

develop new solution methods and data structures

debug and analyze performance

advice on software design, solution algorithms, and performance

`petsc-{users,dev,maint}@mcs.anl.gov`

## You can help PETSc

report bugs and inconsistencies, or if you think there is a better way

tell us if the documentation is inconsistent or unclear

consider developing new algebraic methods as plugins, contribute if your idea works