Design, Implementation and Applications of PETSc-MUMPS Inteface

Hong Zhang

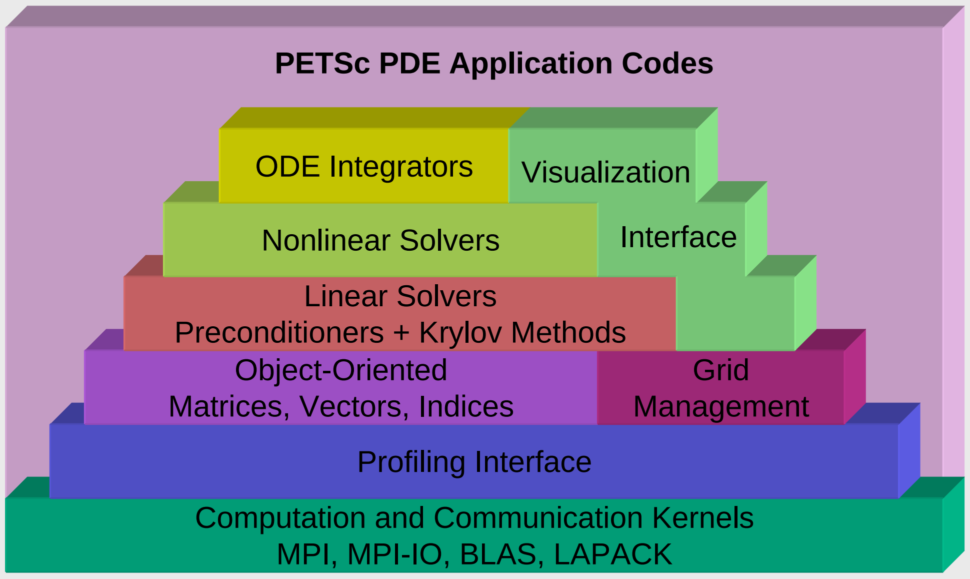
Computer Science, Illinois Institute of Technology
Mathematics and Computer Science, Argonne National Laboratory

What is PETSc?

***P**ortable, **E**xtensible **T**oolkit for **S**cientific **c**omputation*

- Sequential and parallel data structures
- Sequential and parallel algebraic solvers
- API for advanced methods
- Portable(?) to virtually all systems
- Funded largely by the US Dept. of Energy
- www.mcs.anl.gov/petsc (free)

Structure of PETSc


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

PETSc Structure

PETSc Numerical Components

Nonlinear Solvers

Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers

Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods

GMRES	CG	CGS	Bi-CG-STAB	TFQMR	Richardson	Chebychev	Other

Preconditioners

Additive Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

Matrices

Compressed Sparse Row (AIJ)	Blocked Compressed Sparse Row (BAIJ)	Block Diagonal (BDIAG)	Dense	Matrix-free	Other

Distributed Arrays

Index Sets

Indices	Block Indices	Stride	Other

Vectors

What is MUMPS?

***MU**ltifrontal **M**assively **P**arallel sparse direct **S**olver*

- Solution of large linear systems with spd and general matrices
- Iterative refinement and backward error analysis
- Partial factorization and Schur complement matrix
- Several orderings interfaced: AMD, AMF, PORD,METIS
- Written in F90 with C interface
- Parallel version requires BLACS and ScaLAPACK

What is MUMPS?

- Expoits both parallelism arising from sparsity in the matrix and from dense factorizations kernels.
- Partially funded by CEC ESPRIT IV long term research project
- www.enseeiht.fr/irit/apo/MUMPS/

MUMPS solves *Ax=b* in three main steps:

1. Analysis (Job=1):
- the host performs an ordering
- the host carries out symbolic factorization

2. Factorization A=LU or A=LDL^T (Job=2)*:*
- *A* is distributed to processors
- the numerical factorization on each frontal matrix is conducted by a *master* and one or more *slave* processors

3. Solution (Job=3)*:*
- *b* is broadcast from the host
- *x* is computed using the distributed factors
- *x* is either assembled on the host or kept distributed on the processors

MUMPS:

- Each of the phases can be called separately
- Asynchronous communication

  Enable overlapping between communication and computation
- Dynamic scheduling

  Algorithm can adapt itself at execution time to remap work and data to appropriate processors

PETSc-MUMPS Interface

Enable an easy use of

the MUMPS’ parallel sparse direct solvers

under the PETSc environment for

- algorithmic study
- solving computational-intensive problems

Installation of PETSc and MUMPS

1. Download PETSc
2. Configure PETSc with

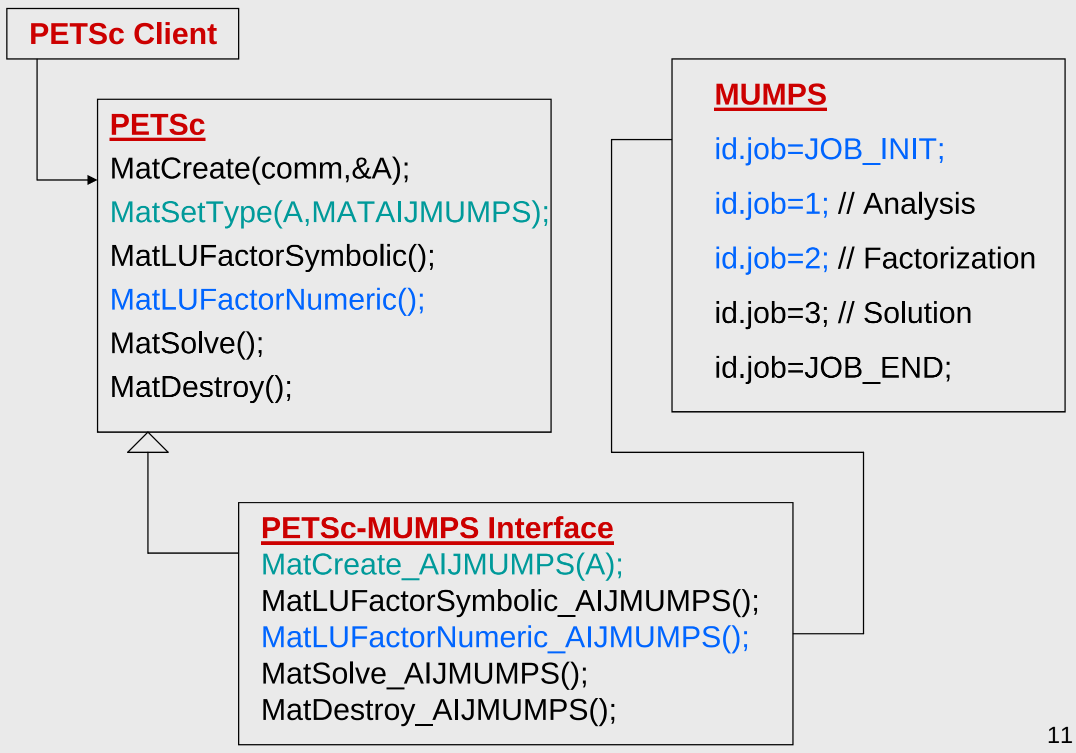
```
./configure.py <petsc_config_opts>
  --download-mumps=yes
  --download-scalapack=yes
  --download-blacs=yes
```

3. Build libraries:

```
./make all
```

Reference: ~petsc/python/PETSc/packages/MUMPS.py

Design of PETSc-MUMPS Interface

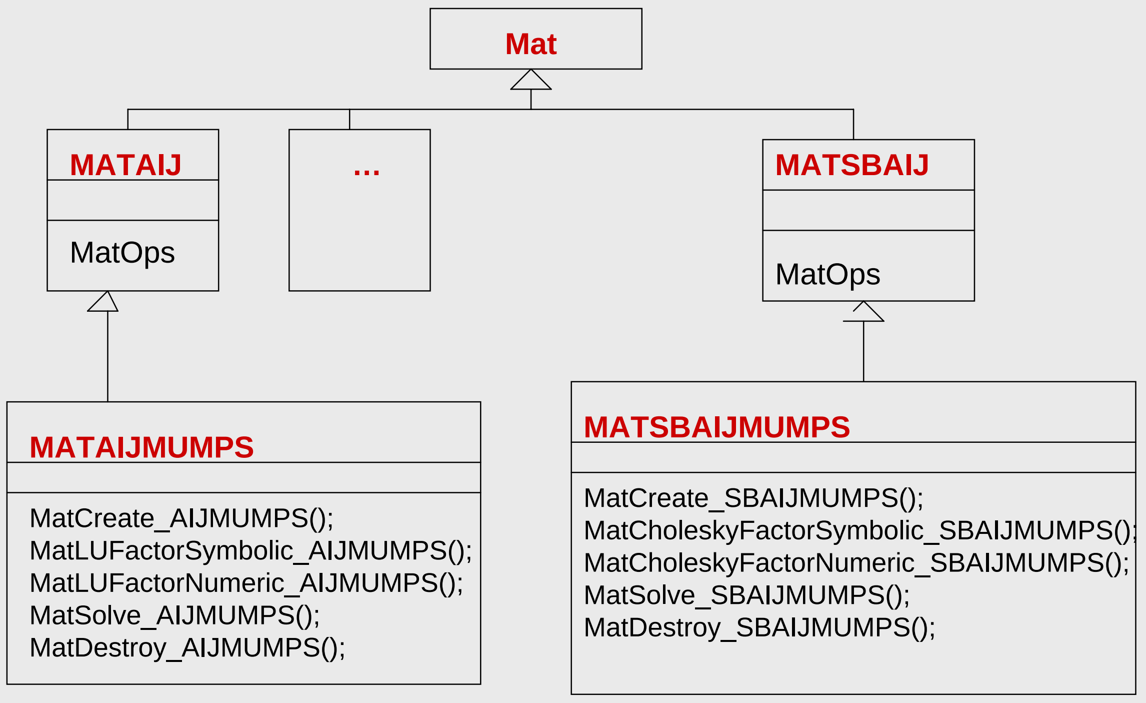


```
MatCreate(comm,&A);
MatSetType(A,MATAIJMUMPS);
MatLUFactorSymbolic();
MatLUFactorNumeric();
MatSolve();
MatDestroy();
```

```
id.job=JOB_INIT;
id.job=1; // Analysis
id.job=2; // Factorization
id.job=3; // Solution
id.job=JOB_END;
```

```
MatCreate_AIJMUMPS(A);
MatLUFactorSymbolic_AIJMUMPS();
MatLUFactorNumeric_AIJMUMPS();
MatSolve_AIJMUMPS();
MatDestroy_AIJMUMPS();
```

Design of PETSc-MUMPS Interface

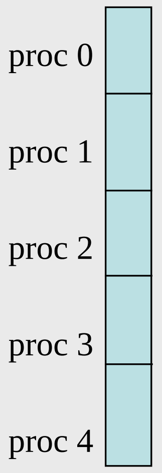


```
MatCreate_AIJMUMPS();
MatLUFactorSymbolic_AIJMUMPS();
MatLUFactorNumeric_AIJMUMPS();
MatSolve_AIJMUMPS();
MatDestroy_AIJMUMPS();
```

```
MatCreate_SBAIJMUMPS();
MatCholeskyFactorSymbolic_SBAIJMUMPS();
MatCholeskyFactorNumeric_SBAIJMUMPS();
MatSolve_SBAIJMUMPS();
MatDestroy_SBAIJMUMPS();
```

PETSc Vector

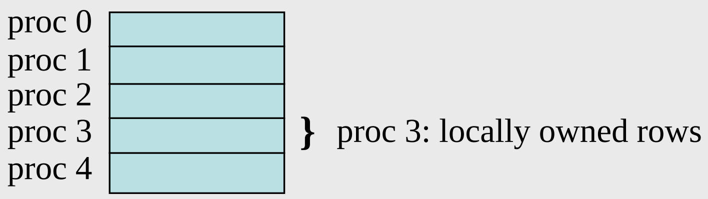
- What are PETSc vectors?
  - Fundamental objects for storing field solutions, right-hand sides, etc.
  - Each process locally owns a subvector of contiguously numbered global indices
- Create vectors via
  - VecCreate(MPI_Comm,Vec *)
    - MPI_Comm - processes that share the vector
  - VecSetSizes(Vec, int, int)
    - number of elements local to this process
    - or total number of elements
  - VecSetType(Vec,VecType)
    - Where VecType is
      - VEC_SEQ, VEC_MPI, or VEC_SHARED
    - VecSetFromOptions(Vec) lets you set the type at *runtime*



data objects:
vectors

PETSc Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global rows.



MatGetOwnershipRange(Mat A, int *rstart, int *rend)

- rstart: first locally owned row of global matrix
- rend -1: last locally owned row of global matrix

data objects:
matrices

MUMPS Matrix Input Structure

- Elemental format and input centrally on the host
- Assembled format:
  1. Input centrally on the host processor
  2. Structure is provided on the host (analysis), entries are distributed across the processors (numeric factorization)
  3. Both structure and entries are provided as local triplets (ICNTL(18)=3)

Matrix Conversion - MatLUFactorNumeric_AIJMUMPS():

PETSc

A_i, B_i:
local diagonal and off-diagonal matrices in row compressed format

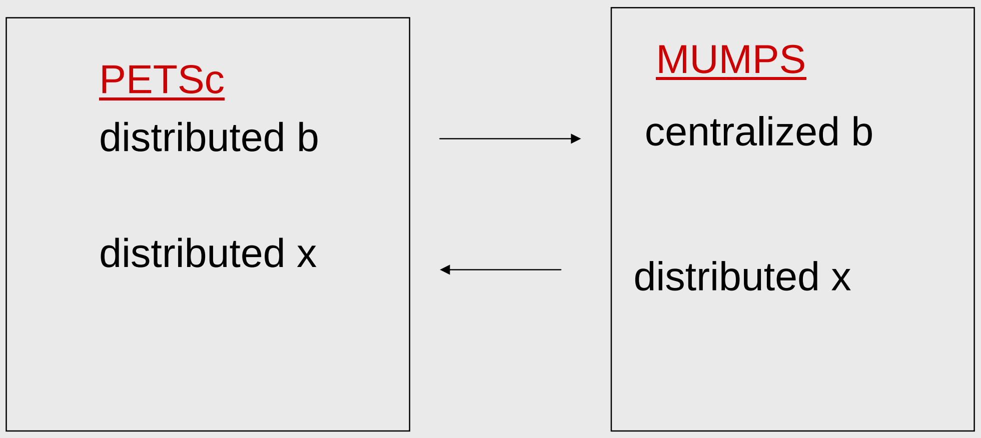
→

MUMPS

C_i:

local matrices in triples

Vector Conversion - MatSolve_AIJMUMPS():



Using PETSc-MUMPS Interface

```
mpirun –np <np> petsc-prog                    \
      –ksp_type preonly –pc_type lu           \
      –mat_type aijmumps <mumps_opts>

mpirun –np <np> petsc-prog                    \
      –ksp_type preonly –pc_type cholesky \
      –mat_type sbaijmumps <mumps_opts>
```

An application:
Modeling of Nanostructured Materials

- **Goal:** Characterisation/prediction of various nanoscale properties
- **Approach:** Determination and analysis of most stable atomic structure
  → Minimisation of many-particle interaction energy

$$E_{\text{tot}}(\{\vec{R}_{\text{at}}\}) = \underbrace{E_{\text{el}}(\{\vec{r}_{\text{el}}\};\{\vec{R}_{\text{at}}\})}_{\textit{hard}} + \underbrace{E_{\text{nuc}}(\{\vec{R}_{\text{at}}\})}_{\textit{“easy”}}$$

- **Methods:**

  1. molecular orbital theory (Schrodinger equation)
  2. density functional theory (DFT)
  3. $\ast$ tight-binding (TB, **DFTB**); semi-empirical
  4. classical potentials (Lennard-Jones, Brenner, . . .)

System size ↓ Accuracy ↑

Density-Functional based Tight-Binding (DFTB)

- **Physics:** *approximate solution of Schrdinger-like equation*
  - *input:* atomic positions $\vec{R}_a$
  - *aux. data:* pairwise interaction functions from higher-level theory

$$\{h_{\mu\nu}, s_{\mu\nu}, \gamma_{\mu\nu}\} = f(\vec{R}_a, \vec{R}_b)$$

  - *output:* Energy E, atomic forces ($\vec{F}_a = \partial E/\partial \vec{R}_a$), wave functions Φ_i, atomic charges q_a, . . .

- **Mathematical core:** *real symmetric definite generalised eigenproblem*

$$\underline{\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{B}\mathbf{x}_i\,,} \quad i = 1 \ldots N$$

λ_i	eigenvalues (electronic energy levels ε_i) − *need lower* *60 %*
$\mathbf{x}_i$	eigenvectors (wave function coefficients $\Phi_{i\mu}$)
$\mathbf{A}, \mathbf{B}$	interaction matrices: $f(\mathbf{h}, \mathbf{s}, \gamma, \mathbf{x}) \to$ *self-consistent problem*

Matrices are

- large: ultimate goal
  50,000 atoms with electronic structure
  ~ N=200,000
- sparse:
  non-zero density -> 0 as N increases
- dense solutions are requested:
  60% eigenvalues and eigenvectors

Dense solutions of large sparse problems!

DFTB-eigenvalue problem is distinguished by

- (A, B) is large and sparse
  Iterative method
- A large number of eigensolutions (60%) are requested
  Iterative method + multiple shift-and-invert
- The spectrum has
  - poor average eigenvalue separation O(1/N),
  - cluster with hundreds of tightly packed eigenvalues
  - gap >> O(1/N)
  Iterative method + multiple shift-and-invert + robusness
- The matrix factorization of $(A-\sigma B)=LDL^T$:
  not-very-sparse(7%) <= nonzero density <= dense(50%)
  Iterative method + multiple shift-and-invert + robusness + efficiency
- $Ax=\lambda Bx$ is solved many times (possibly 1000's)
  Iterative method + multiple shift-and-invert + robusness + efficiency
  + initial approximation of eigensolutions

Lanczos shift-and-invert method for Ax = λBx:

$$Ax = \lambda Bx$$

$$\iff (A - \sigma B)x = (\lambda - \sigma)Bx, \text{ shift } \sigma \neq \lambda$$

$$\iff \frac{1}{\lambda-\sigma} y = \underbrace{B(A - \sigma B)^{-1}}_{C} y, \; y = Bx$$

$$\iff \tilde{\lambda} y = Cy, \; \tilde{\lambda} = \frac{1}{\lambda-\sigma}$$

K(C, v) = span{ v, Cv, C^2v, …., C^{k-1}v }

Eigensolutions of T_k $\longrightarrow$ Eigenvalues of (A,B) close to σ and their eigenvectors

Lanczos shift-and-invert method for Ax = λBx:

- Cost:
  - one matrix factorization:

$$A - \sigma B = LDL^T$$

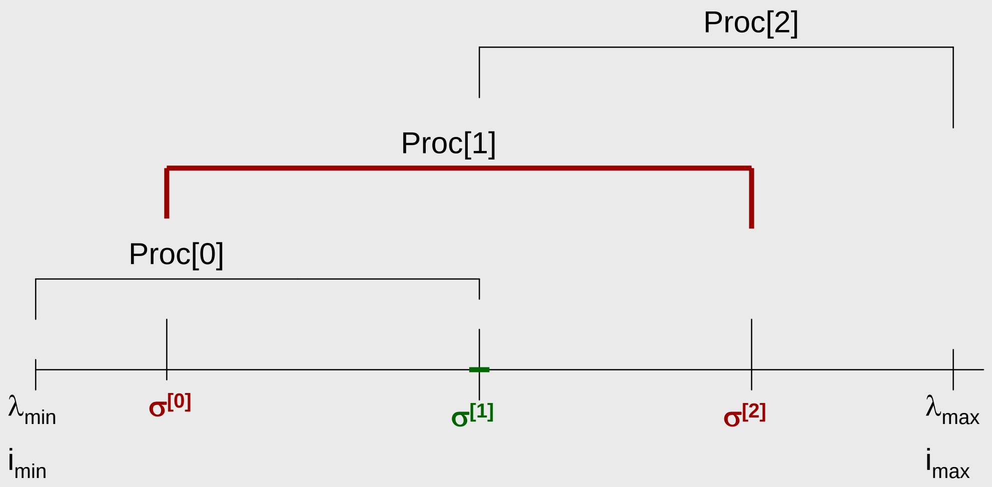
  - many triangular matrix solves:

$$Cv = L^{-T}D^{-1}L^{-1}v, \; C = B(A - \sigma B)^{-1}$$

- Gain:
  - fast convergence
  - clustering eigenvalues are transformed to well-separated eigenvalues
  - preferred in most practical cases

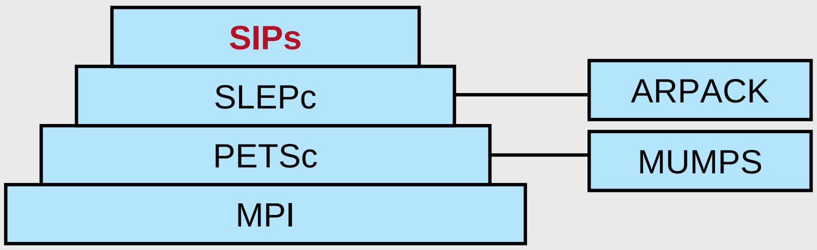
Multiple Shift-and-Invert Parallel Eigenvalue Algorithm

Idea: distributed spectral slicing
compute eigensolutions in distributed subintervals



Software Structure — SIPs

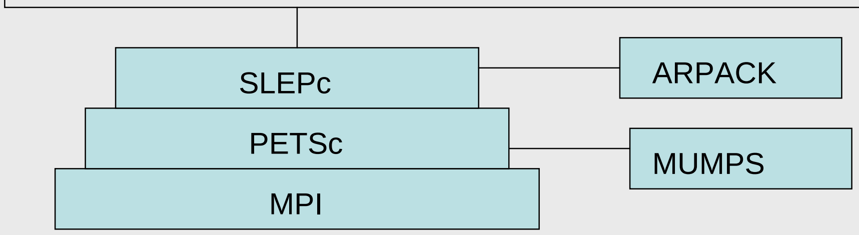
- **S**hift-and-**I**nvert **P**arallel Spectral Transform**s**
  - *Parallelize by spectrum intervals (multiple shifts)*
  - *Balance parallel jobs*
  - *Ensure global orthogonality of eigenvectors*
  - *Manage matrix storage*
  - Builds on existing packages for data and solvers



Software Structure

Shift-and-**I**nvert **P**arallel Spectral Transform**s** (**SIPs**)

- Select shifts
- Bookkeep and validate eigensolutions
- Balance parallel jobs
- Ensure global orthogonality of eigenvectors
- Subgroup of communicators

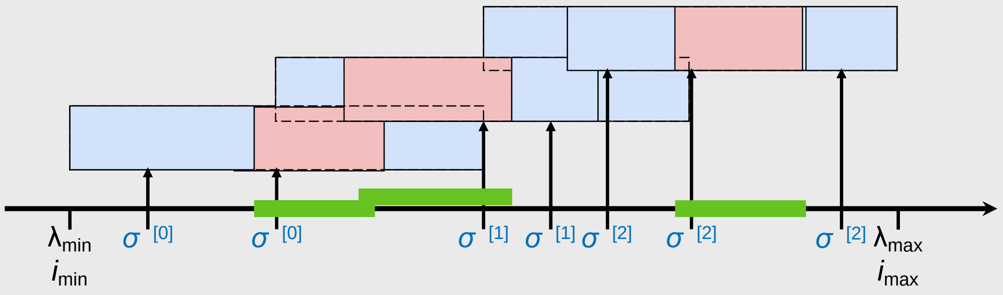


Software Structure — Algebra packages

- SLEPc
  - ***Scalable Library for Eigenvalue Problem Computations***
    - www.grycap.upv.es/slepc/
- ARPACK
  - ***ARnoldi PACKage***
    - www.caam.rice.edu/software/ARPACK/
- MUMPS
  - ***MUltifrontal Massively Parallel sparse direct Solver***
    - www.enseeiht.fr/lima/apo/MUMPS/

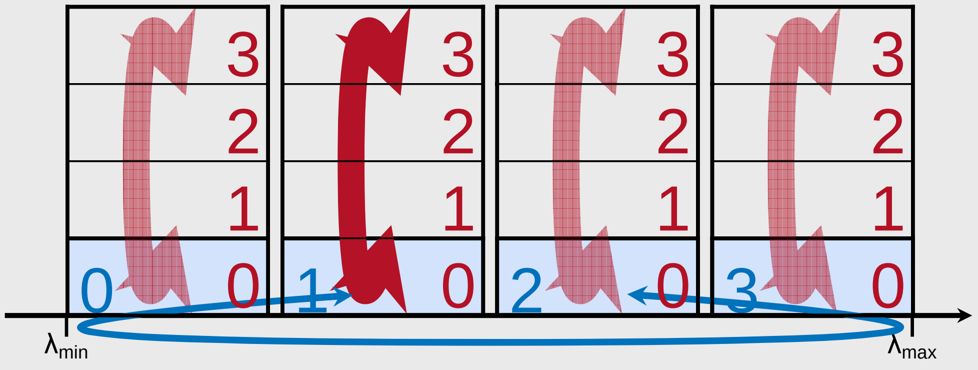
Distributed spectral slicing

- assign intervals to processors
  - compute eigensolutions near shifts (the hard part)
  - validate and tally (handle overlaps later)
  - pick new shifts
  - shrink assigned spectrum (communication)
- iterate



Domain decomposition: “Frequency and Space”

- When a single process cannot store replicated matrices
  - Use more processes, distribute matrix storage
  - introduce sub-communicators
- comb-like communication pattern

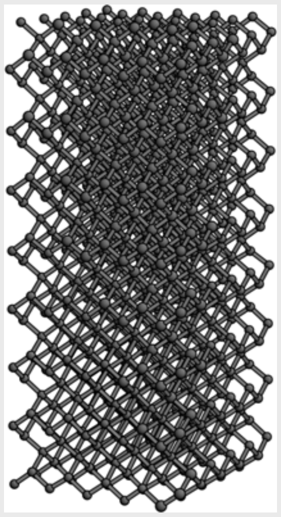


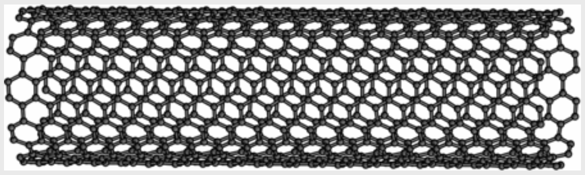
Numerical Experiments — Jazz

- Linux cluster at Argonne
- Compute:
  - 350 nodes with 2.4 GHz Pentium Xeon
- Memory:
  - 175 nodes with 2 GB of RAM
  - 175 nodes with 1 GB of RAM
- Network:
  - Myrinet 2000 (fast)
  - 1 Gb Ethernet (slow)

Physical test systems

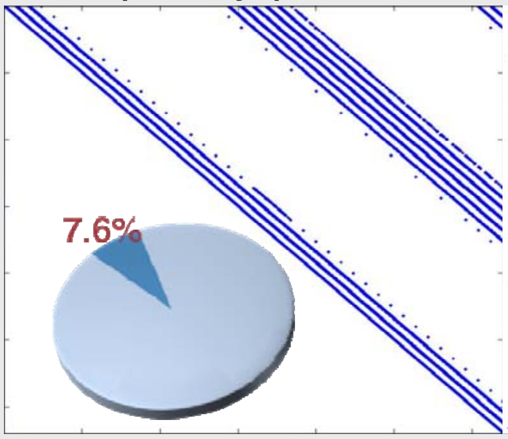
- Single-wall carbon nanotube (10,10)
- Diamond nanowire (25 at. cross sec.)
- Diamond (3D bulk)
- Σ13, Σ29 Grainboundaries
- Graphene
- Si, SiO_2
- … all usually randomized



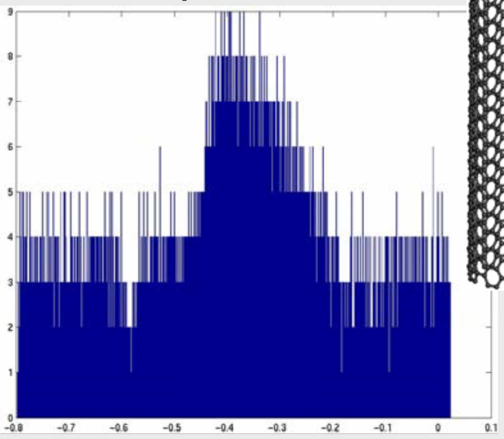


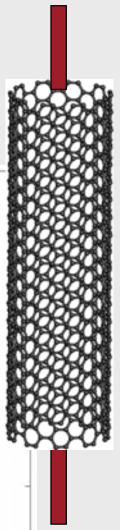
Test system 1 − single-wall carbon nanotube

Sparsity pattern



Spectrum



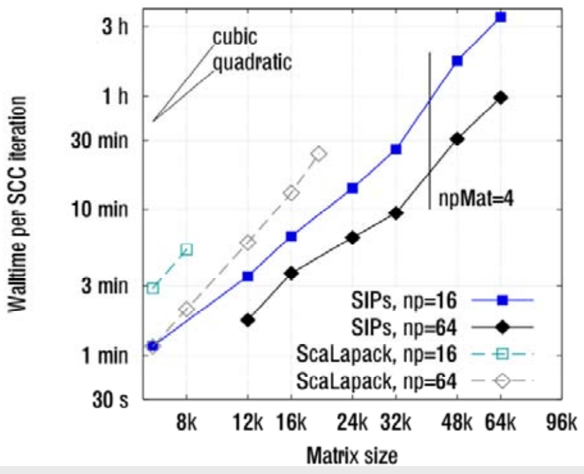


N = 16 000

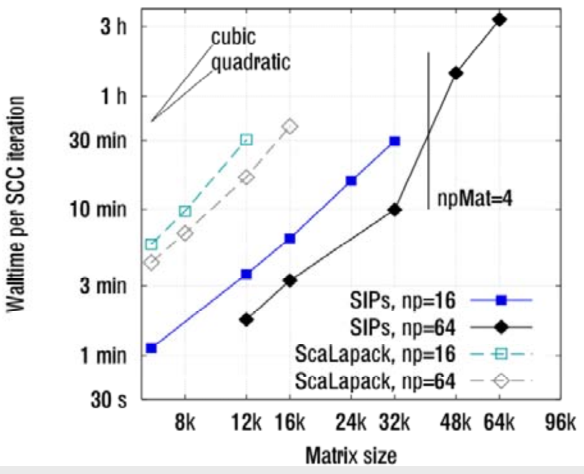
- sparse 1D-system
- randomized positions — limited degeneracies

Numerical results − single-wall carbon nanotube

Myrinet 2000



1 Gb Ethernet

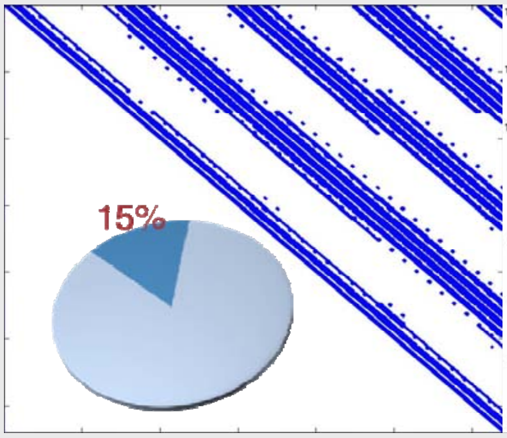


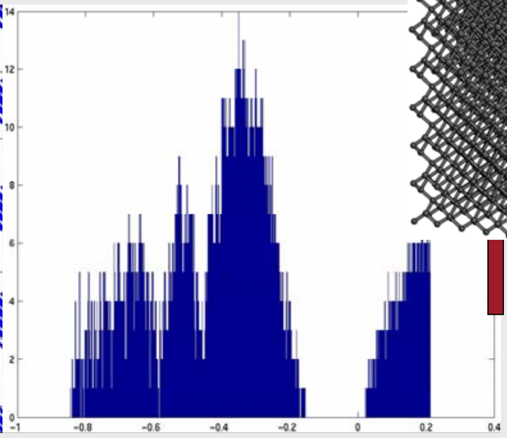
- SIPs faster than ScaLAPACK
- better scaling — SIPs: $\mathcal{O}(N^2)$; ScaLAPACK: $\mathcal{O}(N^3)$

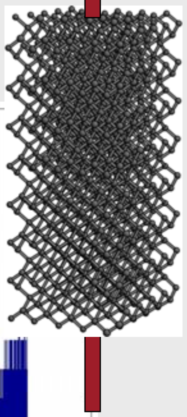
Test system 2 − diamond nanowire

Sparsity pattern

Spectrum







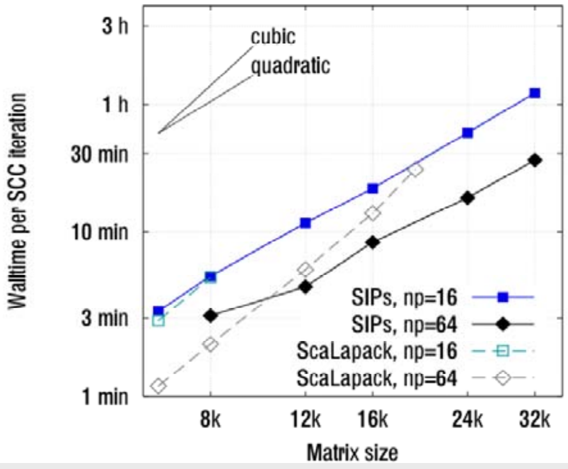
N = 16 000

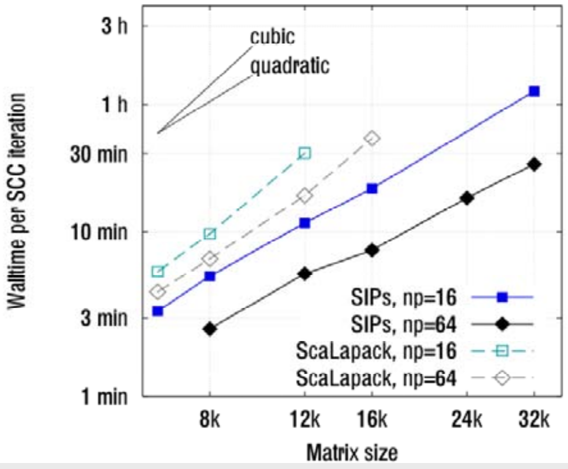
- medium sparse 1D-system
- randomized positions

Numerical results − diamond nanowire

Myrinet 2000

1 Gb Ethernet



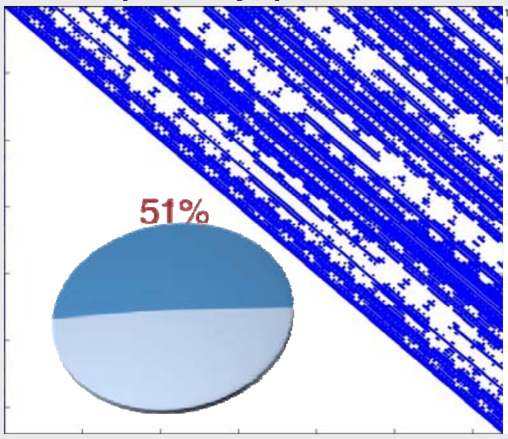


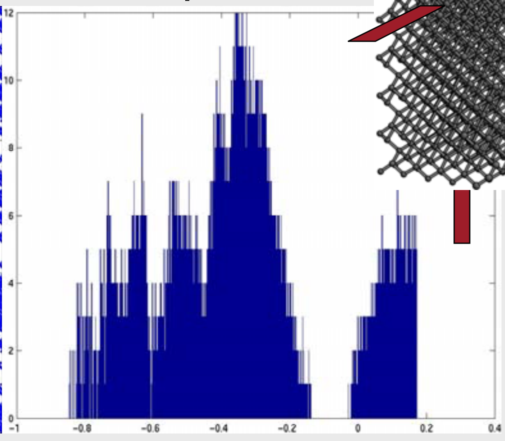
- SIPs still competitive (time, scaling)
- better memory usage — larger systems accessible

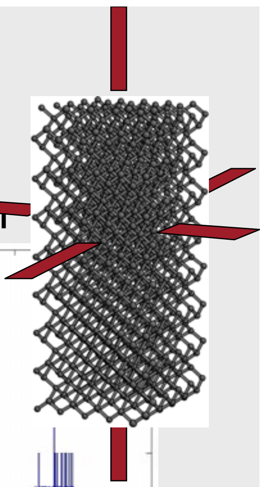
Test system 3 − diamond crystal

Sparsity pattern

Spectrum







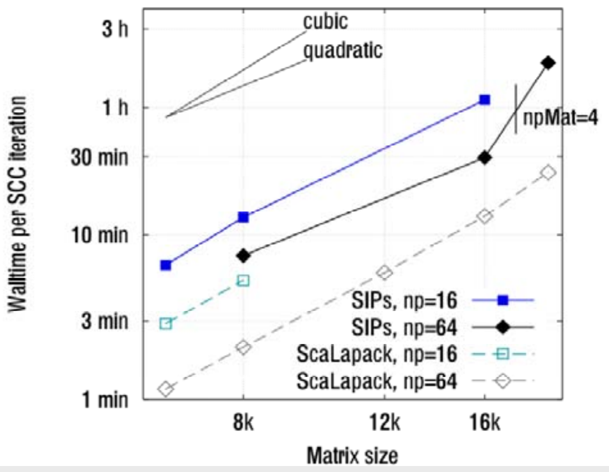
N = 16 000

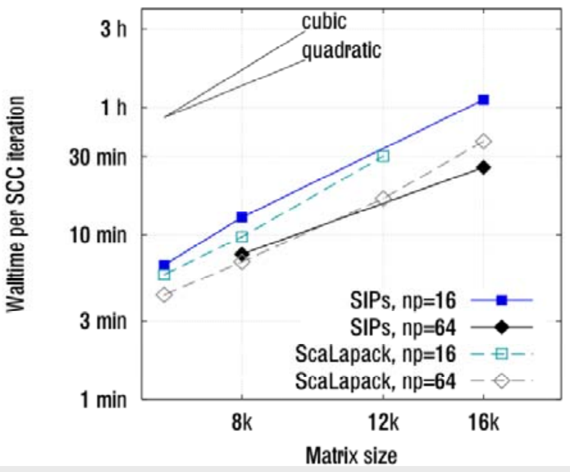
- dense 3D-system

Numerical results − diamond crystal

Myrinet 2000

1 Gb Ethernet





- SIPs can’t compete on fast network
- Good on *commodity network* (GbE) — $\mathcal{O}(N^{3-x})$

Summary

- **SIPs: a new multiple Shift-and-Invert Parallel eigensolver**.
- **Competitive computational speed:**
  - matrices with sparse factorization:
    SIPs: $(O(N^2))$; ScaLAPACK: $(O(N^3))$
  - matrices with dense factorization:
    SIPs outperforms ScaLAPCK on slower network (fast Ethernet) as the number of processors increases
- **Efficient memory usage**:
  SIPs solves much larger eigenvalue problems than ScaLAPACK,
  e.g., nproc=64, SIPs: N>64k; ScaLAPACK: N=19k
- **Object-oriented design:**
  - developed on top of PETSc and SLEPc.
    PETSc provides sequential and parallel data structure;
    SLEPc offers built-in support for eigensolver and spectral transformation.

  - through the interfaces of PETSc and SLEPc, SIPs easily uses external eigenvalue package ARPACK and parallel sparse direct solver MUMPS.
    The packages can be upgraded or replaced without extra programming effort.

Request for Improvements:

- Distributed right-hand-side vector b?
- Efficient matrix conversion?
- Large number of processes,
  e.g., np = 1k,…, 10k?
- Almost exact direct solver with reduced communications?
- Take advantage of a distribution of an initial problem into subdomains?
- …