Development of Sparse Direct Solvers and Eigensolvers in TOPS

Sherry Li
Lawrence Berkeley National Laboratory

CScADS Workshop on Libraries and Algorithms for Petascale Applications
Snowbird, Utah
July 30th – August 2nd, 2007

People

- TOPS participants:
  - **Jim Demmel, UC Berkeley**
  - **Laura Grigori, INRIA, France**
  - **Parry Husbands, LBNL**
  - **Sherry Li, LBNL**
  - **Esmond Ng, LBNL**
  - **Jason Riedy, UC Berkeley**
  - **Chao Yang, LBNL**
- Collaborators:
  - **Zhaojun Bai, UC Davis**
  - **Weiguo Gao, Fudan Univ. China**
  - **Ming Gu, UC Berkeley**
  - **Osni Marques, LBNL**
  - **Panayot Vassilevski, LLNL**
  - **Jianlin Xia, UCLA**

Outline of This Talk

- Sparse direct linear solvers
  - **Parallelizing symbolic analysis for SuperLU**
  - **Linear-complexity factorizations**
- Sparse eigensolvers
  - **Algebraic substructuring method (ASEIG)**
  - **An Expert Eigensolver Toolbox (EigAdept)**

3

Available sparse codes

- Survey of different types of factorization codes

  http://crd.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf
  - **LL^T (s.p.d.), LDL^T (symmetric indefinite), LU (nonsymmetric), QR (least squares)**
  - **Sequential, shared-memory, distributed-memory, out-of-core**
- Distributed-memory solvers: usually MPI-based
  - **SuperLU_DIST [Li, Demmel, Grigori]**
    - **Accessible from PETSc, Trilinos**
  - **MUMPS, PasTiX, WSMP, . . .**

4

SuperLU_DIST major steps:

(parallelization perspectives)

- Static numerical pivoting: improve diagonal dominance
  - **Currently use MC64 (HSL);**
  - **Being parallelized [J. Riedy]**
- Sparsity-preserving ordering
  - **Can use ParMeTis**
- Symbolic factorization: determine pattern of {L\U}
  - **Being parallelized [L. Grigori]**
- Numerics: factorization, triangular solves, iterative refinement (usually dominate total time)
  - **Parallelized a while ago; Need to improve load balance, latency-hiding**

5

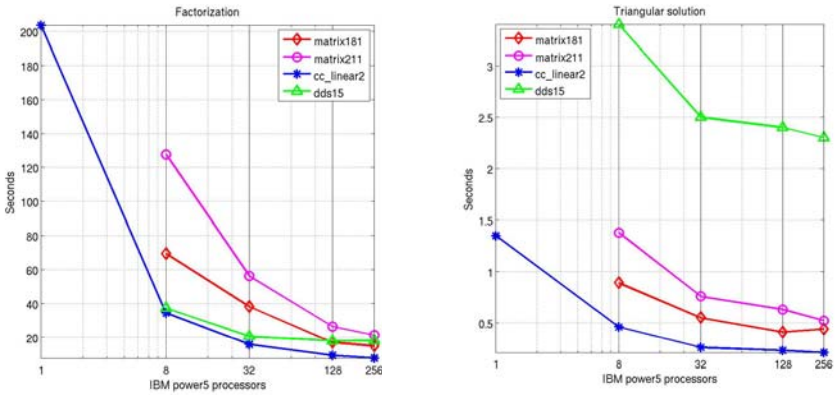
Examples

Name	Codes	Type	N	|A| / N	Fill-ratio
matrix181	M3D-C1 (Fusion)	Real	589,698	161	9.3
matrix211	M3D-C1 (Fusion)	Real	801,378	161	9.8
cc_linear2	NIMROD (Fusion)	Complex	259,203	109	7.5
dds15	Omega3P (Accelerator)	Real	834,575	16	40.2

- Sparsity-preserving ordering: MeTis applied to structure of A'+A

6

Performance on IBM Power5 (1.9 GHz)

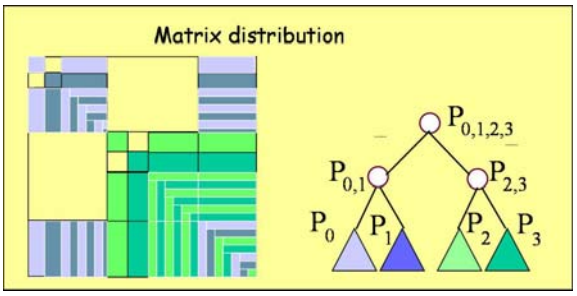


- 161 Gflops factorization rate for matrix121

7

Parallelizing symbolic factorization

- Serial algorithm is fast (usually < 10% total time) but requires entire structure of A, limiting memory scalability
- Parallel approach
  - **Use graph partitioning to reorder/partition matrix.**
    - **ParMetis on structure of A + A'**
  - **Exploit parallelism given by this partition (coarse level) and by a block cyclic distribution (fine level)**



8

Max. memory (MB) of parallel symbolic

Matrix181		P = 8	P = 256
	LU fill (millions)	1094.2	1445.3
symbolic	Sequential Parallel Ratio Seq./Par.	365.5 18.0 20	365.5 8.1 45
Entire solver	Old New	1445.1 1262.8	377.2 84.3

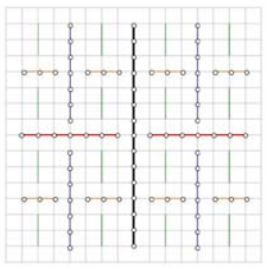
dds15		P = 8	P = 256
	LU fill (millions)	528.9	583.7
symbolic	Sequntial Parallel Ratio Seq./Par.	295.8 27.2 11	295.8 10.8 27
Entire solver	Old New	1061.9 817.0	341.3 113.1

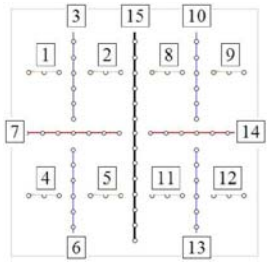
Open problem: improve quality of parallel orderings.

9

Linear-complexity sparse factorization

- In the spirit of fast multipole, but for matrix inversion
- Model problem: discretized system Ax = b from certain PDEs, e.g., 5-point stencil on k x k grid, $n = k^2$
- Nested dissection ordering gave optimal complexity using exact elimination [Hoffman/Martin/Ross]
  - **Factorization cost: $O(n^{3/2})$ (3D: $O(n^2)$)**





10

Linear-complexity sparse factorization

- Exploit low-rank structure -- represented by semi-separable matrices (… think about SVD)
  Example: semi-separable matrix with 4 x 4 blocks

$$A \cong \begin{pmatrix} D_1 & U_1V_2^T & U_1W_2V_3^T & U_1W_2W_3V_4^T \\ V_2U_1^T & D_2 & U_2V_3^T & U_2W_3V_4^T \\ V_3W_2^TU_1^T & V_3U_2^T & D_3 & U_3V_4^T \\ V_4W_3^TW_2^TU_1 & V_4W_3U_2^T & V_4U_3^T & D_4 \end{pmatrix}$$

First and second off-diagonal blocks of A are

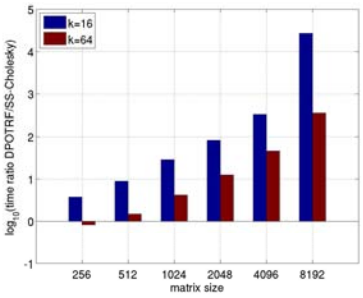
$$U_1(V_2^T \;\; W_2V_3^T \;\; W_2W_3V_4^T) \quad \text{and} \quad \begin{pmatrix} U_1W_2 \\ U_2 \end{pmatrix}(V_3^T \;\; W_3V_4^T)$$

(i,j) block entry is $U_iW_{i+1}W_{i+2}\cdots W_{j-1}V_j^T$
D, U, W, and V are matrices of dimension k
A is n x n, uses O(n k) memory, good for k<< n

Examples: banded matrices and their inverses

Results of fast solvers

- **Fast dense SS-Cholesky**
  - **Speedup over LAPACK DPOTRF**



- **Fast sparse multifrontal solver:**
  Represent separator submatrices as SS structures, and perform SS-Cholesky for the separators.
- **Complexity for model problems**
  2D: O(p k²), p is related to tolerance (i.e., numerical rank)
  3D: O(c(p) k³), c(p) is a polynomial

Multifrontal solvers on SGI Altix

2D Mesh	255	1023	4095
MF	0.22	8.7	383.6
Fast MF	0.24	6.1	113.4

Great potential as preconditioners !

Outline of This Talk

- Sparse eigensolvers
  - **Algebraic sub-structuring method (ASEIG)**
  - **An Expert Eigensolver Toolbox (EigAdept)**

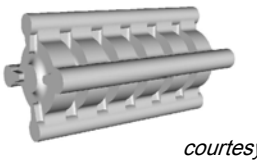
13

Eigenvalue problem

$$K x = \lambda M x \quad \xrightarrow{\sigma} \quad K^{\sigma} x = \theta^{\sigma} M x$$

where, $K^{\sigma} = K - \sigma M$

- Motivated by the EVP from accelerator SciDAC
  - **Small eigenvalues (tightly clustered) out of a large-eigenvalue dominated eigenspetrum: many small nonzero eigenvalues desired**
  - **Large null space in the stiffness matrix K**
  - **Requires high accuracy for eigenpairs**
  - **Need to be solved many times in shape optimization loop**



courtesy of SLAC

14

Substructuring methods

- Substructuring dates back to the 1960s, e.g., CMS
- Holds great promoise for solving extremely large scale problems, e.g., AMLS
  - **Industrial strength code in structural engineering (Bennighof, Kaplan, Lehoucq)**
    - **Compute vibration modes**
    - **Perform frequency response analysis**
- Open questions as the techniques extended for broad applications:
  - **Arbitrary eigenmodes**
  - **High frequency response analysis**
  - **Accuracy**
  - **Performance**

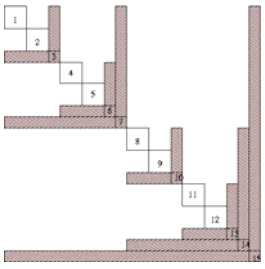
15

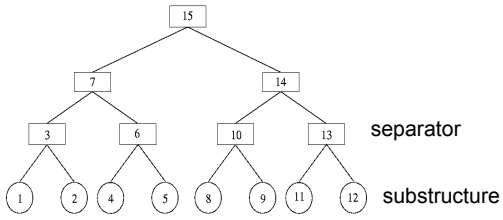
Substructure partition

- Single-level

$$K^{\sigma} = \begin{pmatrix} K_{11}^{\sigma} & & K_{13}^{\sigma} \\ & K_{22}^{\sigma} & K_{23}^{\sigma} \\ K_{31}^{\sigma} & K_{32}^{\sigma} & K_{33}^{\sigma} \end{pmatrix}, \quad M = \begin{pmatrix} M_{11} & & M_{13} \\ & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix}$$

- Multi-level (nested dissection)





16

Substructure reduction

- Block elimination matrix L (Craig-Bampton form)

$$\hat{K}^{\sigma} = L^{-T} K^{\sigma} L^{-1} = \begin{pmatrix} K_{11}^{\sigma} & & \\ & K_{22}^{\sigma} & \\ & & \hat{K}_{33}^{\sigma} \end{pmatrix}, \quad \hat{M} = L^{-T} M L^{-1} = \begin{pmatrix} M_{11} & & \hat{M}_{13} \\ & M_{22} & \hat{M}_{23} \\ \hat{M}_{31} & \hat{M}_{32} & \hat{M}_{33} \end{pmatrix}$$

- Local modes

$$S_m = \begin{pmatrix} S_1 & & \\ & S_2 & \\ & & S_3 \end{pmatrix} \begin{matrix} \leftarrow (K_{11}^{\sigma}, M_{11}) \\ \leftarrow (K_{22}^{\sigma}, M_{22}) \\ \leftarrow (\hat{K}_{33}^{\sigma}, \hat{M}_{33}) \end{matrix}$$

- $(\hat{K}^{\sigma}, \hat{M})$ – projection on span $\{S_m\}$

$$K_m^{\sigma} = S_m^T \hat{K}^{\sigma} S_m = \begin{pmatrix} k_{11}^{\sigma} & & \\ & k_{22}^{\sigma} & \\ & & \hat{k}_{33}^{\sigma} \end{pmatrix}, \quad M_m = S_m^T \hat{M} S_m = \begin{pmatrix} m_{11} & & \hat{m}_{13} \\ & m_{22} & \hat{m}_{23} \\ \hat{m}_{31} & \hat{m}_{32} & \hat{m}_{33} \end{pmatrix}$$

- Projected eigenvalue problem

$$K_m^{\sigma} \Phi = M_m \Phi \Theta^{\sigma}$$

17

Implementation and performance evaluation

- Major operations:
  1. **transformations and projection**
  2. **projected eigenvalue problem**
- Cost:
  1. **flops: more than a single sparse Cholesky factorization**
  2. **storage: Block Cholesky factors + projected matrices + ...**

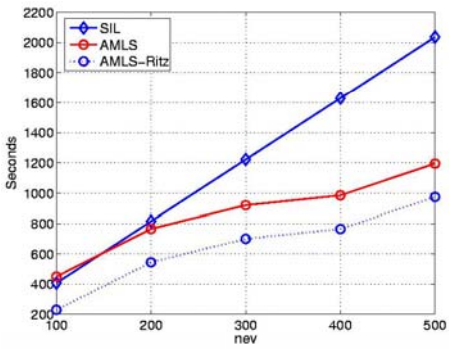
  But no triangular solvers, no (re)-orthogonalization
- ASEIG
  1. **interleave steps 1-4 computations**
  2. **develop an in-core implementation**
  3. **recompute some intermediate matrix blocks instead of storing (semi-implicit representation)**

     50% of memory saving with ≈ 15% recompute time

18

Case study: accelerator structure

- A 6-cell DDS realistic structure
  - **N = 65K, nnz = 1.5M**
- 4-level AS, $n_{proj} \approx 3K$



- Many eigenvalues are wanted, $O(10^3)$
- SIL reqiures multiple shifts (factorizations)

19

EigAdept: an expert eigensolver toolbox

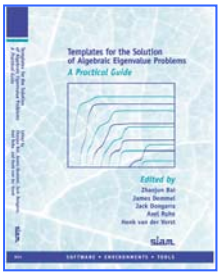
- Motivated by nano-science, accelerator SciDACs ...
- Many eigenvalue libraries have been developed
  - **Serial**
  - **Parallel**
- No unified software framework
- A large number of parameters associated with each algorithm
- Different architectures/applications need different configurations
- Difficult to choose the “optimal” algorithm for a particular application

20

EigAdept: *goals*

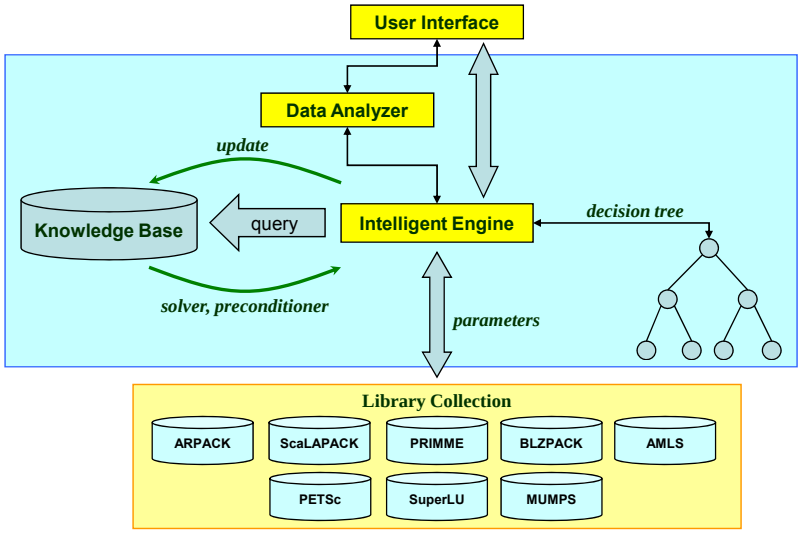
1. Uniform interface
   - **Easy to use**
   - **Hide unnecessary information**
2. Intelligent engine
   - **Guide user through the various numerical libraries**
   - **Help user to achieve the best performance for the target application**

Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, Eds. Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia, 2000.



21

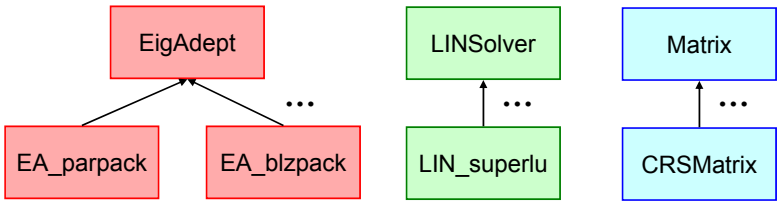
EigAdept: *Architecture*



22

EigAdept: *infrastructure*

C++ class hierarchy



- New eigensolver can be added in EigAdept base class
- New linear solver can be added in LINSolver base class
- New matrix format can be added in Matrix base class

23

How to Use EigAdept?

- Simplest: intelligent engine selects solver

```
EigAdept myeig(A,B);
/* "set" methods to set system properties */
myeig.solve();
```

- User selects eigensolver: bypass intelligent engine

```
EA_parpack myeig(A, B);
myeig.setNev(nev);  /* "set" methods to modify parameters */
myeig.solve();
```

- User selects both eigensolver and linear solver

```
EA_parpack myeig(A,B);
LIN_superlu mylin(A-σB);
myeig.solve(mylin);
```

24

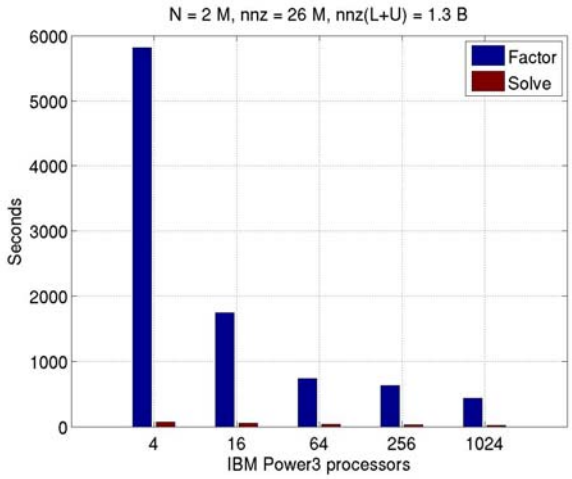
Summary

- Direct linear solvers
  - **Parallel analysis code will be released soon**
  - **Will look into triangular solver**
  - **Superfast solver: needs to generalize to more general geometry, and parallelization**
- Eigensolvers
  - **Parallelize ASEIG**
  - **EigAdept: connect more eigensolvers, and build knowledge base.**

25

26

Performance on IBM Power3 (375 MHz)



- Quantum mechanics, complex: N = 2 million

27