Honoring: Yousuff Hussaini

Scalable Parallel Sparse Matrix Computations

Ahmed Sameh
Computer Science, Purdue University
September 28, 2012

Joint work with:
M. Manguoglu, F. Saied, O. Schenk

Support: ARO, Intel, NSF.
Sparse Matrix Computations

• **Importance**
  
  – They arise in:
    
    • *computational engineering applications*
    
    • *network analysis*
    
    • *analysis of large data sets*
  
  – They give rise to indirect addressing which often leads to significant performance degradation on various parallel architectures.
  
  – Performance of sparse matrix primitives and algorithms on parallel architectures often governs the overall performance of many applications.
Sparse Matrix Computations...

- Fresh ideas for designing parallel sparse matrix algorithms are needed:
  - the availability of various parallel programming tools proved to be insufficient to assure high performance in implementing familiar sequential sparse matrix kernels and algorithms.
The focus here is on the design of sparse matrix computation schemes that:

- exhibit ample concurrency,
- address memory management bottlenecks within a node, and
- minimize internode communications.
Outline

• Parallel sparse matrix primitives:
  – matrix reordering
  – sparse matrix-vector (multivector) multiplication

• Parallel sparse matrix algorithms for two fundamental linear algebra problems with wide applications:
  – linear systems of equations
  – symmetric algebraic eigenvalue problems
Computing Platform

- Endeavor Intel cluster with infiniband interconnect
- Each node contains 12 to 80 cores
- Local memory per node $\leq$ 48 GB
- Architectures ranging from Nehalem to Sandy Bridge.
- Most recent version of MKL and Olaf Schenk’s direct sparse system solver -- PARDISO.
Two important sparse matrix primitives
Primitive 1: Reordering

- Parallel sparse matrix reordering enables:
  - Faster sparse matrix-vector multiplications.
  - Extracting more effective parallel preconditioners for iterative sparse linear system solvers.
**UFL: smt -- structural mechanics**

N: 25,710  NNZ: 3,749,582

obtaining the Fiedler vector via the eigensolver: TraceMIN

*(Wisniewski and A.S. -- SINUM, ’82)*
Relative bandwidth (RBW):
\[ \frac{\sum |a(i,j)| \text{ within band}}{\sum |a(i,j)|} \]

Reordering:
A := BCSSTK22

Blue: no reordering
Green: HSL-MC73
Red: TraceMIN-Fiedler

half-bandwidth: k
Parallel Scalability of our weighted spectral reordering scheme
TraceMIN-Fiedler vs. HSL-MC73 *(Pothen & Simon)*

<table>
<thead>
<tr>
<th>Matrix Group/Name</th>
<th>$n$</th>
<th>nnz</th>
<th>symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Rajat/rajat31</td>
<td>4,690,002</td>
<td>20,316,253</td>
<td>no</td>
</tr>
<tr>
<td>2. Schenk/nlpkkt120</td>
<td>3,542,400</td>
<td>95,117,792</td>
<td>yes</td>
</tr>
<tr>
<td>3. Freescale/Freescale1</td>
<td>3,428,755</td>
<td>17,052,626</td>
<td>no</td>
</tr>
<tr>
<td>4. Zaoui/kkt_power</td>
<td>2,063,494</td>
<td>12,771,361</td>
<td>yes</td>
</tr>
</tbody>
</table>
$T(\text{HSL-MC73}) ÷ T(\text{TraceMIN-Fiedler})$
Weighted spectral reordering of MEMS benchmark 1

System size:

\( N = 11,333,520 \)

# of nonzeros:

61,026,416

bandwidth:

334,613
Scalability of TraceMin-Fiedler

~ Time in seconds

Nodes: 1 to 32

Speed improvement ~ 22
**Primitive 2:**

Matrix-vector multiplication  

*MATVEC*

- $P A P' = B + E$ (symmetric reordering)
  - $A$: sparse
  - $B$: banded, $E$: sparse of low rank
- $y = A \times x$
  1. $u = P \times x$
  2. $v = B \times u; w = E \times u$
  3. $z = (v + w)$
  4. $y = P' \times z$

**High performance:** $B \times u$

**Low cost:** $u = P \times x$ & $y = P' \times z$
Target Computational Loop

Integration

Newton iteration

Linear system solvers

require relative residuals of $O(10^{-5}$ or $10^{-6}$)

$\eta_k$

$\epsilon_k$

$\Delta t$

$\in k$
How expensive is spectral reordering?

Integration

Newton Iteration

Linear system solvers

Cost of WSO is amortized across several time steps.

\( \eta_k \)

\( \varepsilon_k \)

\( \Delta t \)

will return to this issue later
Impact of a faster MATVEC on a time-dependent problem:

Animation

Solving s.p.d. systems via a preconditioned C.G. scheme at each time step
Permutation of time-step #1 applied to time-step #2

\[ C_2 = P_1 A_2 P_1^T \]
Permutation of time-step#1 applied to time-step#16

\[ A_{16} \]

\[ C_{16} = P_1 A_{16} P_1^T \]
## Time in seconds to process one frame (16 time steps)

<table>
<thead>
<tr>
<th></th>
<th>ISV</th>
<th>MKL Matvec</th>
<th>our Matvec after Reorder</th>
<th>our Matvec after Reorder</th>
<th>our Matvec after Reorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-core Nehalem</td>
<td>3.04</td>
<td>1.32</td>
<td>0.84</td>
<td>0.30</td>
<td>0.14</td>
</tr>
<tr>
<td>12-core Westmere</td>
<td>1.32</td>
<td>0.84</td>
<td>0.30</td>
<td>0.14</td>
<td>22</td>
</tr>
<tr>
<td>12-core Westmere</td>
<td>0.84</td>
<td>0.30</td>
<td>0.14</td>
<td>22</td>
<td>0.14</td>
</tr>
<tr>
<td>40-core Westmere</td>
<td>0.30</td>
<td>0.14</td>
<td>0.03</td>
<td>22</td>
<td>0.14</td>
</tr>
<tr>
<td>16 12-core nodes Westmere</td>
<td>0.14</td>
<td>0.03</td>
<td>0.03</td>
<td>22</td>
<td>0.14</td>
</tr>
</tbody>
</table>
A Hybrid Sparse Linear System Solver: PSPIKE
Target Computational Loop

Integration

Newton iteration

Linear system solvers
require relative residuals of $O(10^{-5} \text{ or } 10^{-6})$

$\eta_k$

$\epsilon_k$

$\Delta t$
PSPIKE

• Systematic approach for solving sparse linear systems:
  – Apply our parallel spectral reordering scheme via our eigensolver TraceMIN_Fiedler.
  – Extract preconditioner
  – Use the nested iterative scheme:
    • Outer Krylov subspace method
    • Inner modified Richardson splitting**

** the multicore sparse direct solver PARDISO is applied simultaneously to handle several smaller systems one per node
UFL: f2 -- structural mechanics

N: 71,505  NNZ: 5,294,285

Original matrix  After MC73  After TraceMin-Fiedler

**TraceMIN-Fiedler**: Murat Manguoglu et. al.
Before reordering

After reordering via TraceMIN-Fiedler

UFL – f2
\[ M z = r \quad (M \text{ is } \text{“banded”}) \]

\[ P = M + \delta(M) = D' \ast S' \]

(i) Solve \( D'y = r \)

(ii) Solve \( S'z = y \)

Solving systems involving

The preconditioner \( Pz = r \)
Spike Matrix $S$ for 3 partitions

\[ \hat{S} \hat{x} = \hat{g} \]

reduced system
Generating tips of the spikes

\[ M_{kk} \]

\[ ? = 0 \]

Obtain the upper and lower tips of the solution block via the modified direct sparse system solver “Pardiso”.
$A \mathbf{x} = \mathbf{f}$

$n = 600,000$

$bw = 99$

Time (sec) on a 4-core Intel Clovertown

MKL uses ScaLapack (LU factorization)
off-chip data accessed (in bytes)

"Analyzing memory access intensity in parallel programs for multicore architectures"

L. Liu, Z. Li, and A. S.
Parallel Scalability of PSPIKE vs. direct solvers
UFL – Rajat31 (circuit simulation)

\[ N \sim 4.7 \text{ M} \]
\[ \text{nnz} \sim 20 \text{ M} \]

nonsymmetric
PSPIKE – WSMP – MUMPS

Time in seconds

# of cores (8 cores per node)

PSPIKE: \[\beta = 5\]

PSPIKE: 1 MPI proc./node
8 OMP threads/MPI proc.

Total time for WSMP & MUMPS

PSPIKE:

\[\text{rel. res.} = O(10^{-5})\]

(128 nodes)
Pardiso vs. PSPIKE on a single node

- PSPIKE is used on an 80 core single node (Intel Xeon E7-8870 server, 2.4 GHz) with a number of different choices of:
  - Number of MPI processes
  - Number of OpenMP threads per MPI process
  - Number of cores used

The total number of cores used is the product of the # of MPI processes and the number of threads per process.
System 1: Matrix -- Dziekonski/dielFilterV2real

*(High-order finite element method in EM)*

http://www.cise.ufl.edu/research/sparse/matrices/Dziekonski/dielFilterV2real.html

<table>
<thead>
<tr>
<th>Matrix properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rows</td>
<td>1,157,456</td>
</tr>
<tr>
<td>number of columns</td>
<td>1,157,456</td>
</tr>
<tr>
<td>nonzeros</td>
<td>48,538,952</td>
</tr>
<tr>
<td>structural full rank?</td>
<td>yes</td>
</tr>
<tr>
<td>structural rank</td>
<td>1,157,456</td>
</tr>
<tr>
<td># of blocks from dmperm</td>
<td>1</td>
</tr>
<tr>
<td># strongly connected comp.</td>
<td>1</td>
</tr>
<tr>
<td>explicit zero entries</td>
<td>0</td>
</tr>
<tr>
<td>nonzero pattern symmetry</td>
<td>symmetric</td>
</tr>
<tr>
<td>numeric value symmetry</td>
<td>symmetric</td>
</tr>
<tr>
<td>type</td>
<td>real</td>
</tr>
<tr>
<td>structure</td>
<td>symmetric</td>
</tr>
<tr>
<td>Cholesky candidate?</td>
<td>no</td>
</tr>
<tr>
<td>positive definite?</td>
<td>no</td>
</tr>
</tbody>
</table>

**PSPIKE:**

rel. residual $\leq 10^{-8}$
### PSPIKE vs. Pardiso

*(rel. res. \( \leq 10^{-8} \))

<table>
<thead>
<tr>
<th>MPI proc.</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>16</th>
<th>( T(\text{Pardiso}) ÷ T(\text{PSPIKE}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores: 1</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
<td>.95</td>
</tr>
<tr>
<td>2</td>
<td>228</td>
<td>163</td>
<td></td>
<td></td>
<td>1.23</td>
</tr>
<tr>
<td>4</td>
<td>141</td>
<td>102</td>
<td>43</td>
<td></td>
<td>2.46</td>
</tr>
<tr>
<td>64</td>
<td>62</td>
<td>39</td>
<td>16</td>
<td>8</td>
<td>3.88</td>
</tr>
</tbody>
</table>
System 2: Matrix – vanHeukelum/cage13
(DNA electrophoresis, polymer. A. van Heukelum, Utrecht U)

<table>
<thead>
<tr>
<th>Matrix properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rows</td>
<td>445,315</td>
</tr>
<tr>
<td>number of columns</td>
<td>445,315</td>
</tr>
<tr>
<td>nonzeros</td>
<td>7,479,343</td>
</tr>
<tr>
<td># strongly connected comp.</td>
<td>1</td>
</tr>
<tr>
<td>explicit zero entries</td>
<td>0</td>
</tr>
<tr>
<td>nonzero pattern symmetry</td>
<td>symmetric</td>
</tr>
<tr>
<td>numeric value symmetry</td>
<td>20%</td>
</tr>
<tr>
<td>type</td>
<td>real</td>
</tr>
<tr>
<td>structure</td>
<td>unsymmetric</td>
</tr>
<tr>
<td>Cholesky candidate?</td>
<td>no</td>
</tr>
<tr>
<td>positive definite?</td>
<td>no</td>
</tr>
</tbody>
</table>

**PSPIKE:**
rel. residual $\leq 10^{-8}$
**PSPIKE vs. Pardiso**

*(rel. res. ≤ 10^{-8})*

<table>
<thead>
<tr>
<th>MPI proc.</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>16</th>
<th>$\frac{T(\text{Pardiso})}{T(\text{PSPIKE})}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cores = 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~ 1</td>
</tr>
<tr>
<td>2</td>
<td>21,266</td>
<td></td>
<td></td>
<td></td>
<td>~ 1</td>
</tr>
<tr>
<td>4</td>
<td>12,034</td>
<td>5,309</td>
<td></td>
<td></td>
<td>~ 2</td>
</tr>
<tr>
<td>64</td>
<td>6,223</td>
<td>3,033</td>
<td>851</td>
<td></td>
<td>~ 6</td>
</tr>
<tr>
<td>1,055</td>
<td>584</td>
<td>165</td>
<td>12</td>
<td></td>
<td>~ 182</td>
</tr>
</tbody>
</table>
Robustness & Parallel Scalability of PSPIKE vs. preconditioned iterative solvers
Computational Electromagnetics

UFL: DW8192

Discretization of the Helmholtz equation (2D):

\[ \nabla^2 H_x + k^2 n^2(x, y) H_x = \beta^2 H_x, \]
\[ \nabla^2 H_y + k^2 n^2(x, y) H_y = \beta^2 H_y. \]
System based on sparse matrix DW8192:
- $n = 8192$
- $nnz = 41,746$
- $\kappa = O(10^7)$

$\text{Spectrum of } P^{-1}A$

$\text{MC64 + ILUT Preconditioner: } P$
- 20% fill-in per row
- rel. drop tol = $10^{-1}$
System based on sparse matrix DW8192:
- $n = 8192$
- $nnz = 41,746$
- $\kappa = O(10^7)$

WSO + narrow-banded preconditioner: $M$
- $\varepsilon = 10^{-4}$
- half-bandwidth $\beta \leq 50$
MEMS simulation benchmark 1

System size:
N = 11,333,520

# of nonzeros:
61,026,416

bandwidth:
334,613

stopping criterion:
rel. res. = O(10^{-2})
Scalability of PSPIKE vs. Trilinos

Intel Harpertown

• Strong scalability of PSPIKE
  Fixed problem size – 1 to 64 nodes (or 8 to 512 cores)
• Comparison with AMG-preconditioned Krylov subspace solvers in:
  • Hypre (LLNL)
  • Trilinos-ML (Sandia)
    • Smoother –
      • Chebyshev fastest
      • Jacobi
      • Gauss-Seidel
Speed Improvement over Trilinos-ML

Time (Trilinos-ML) ÷ Time (PSPIKE)

PSPIKE: k threads per MPI process

break-even @ 4 nodes

Preconditioner bw: β = 5

# of nodes k
1 to 4 1
8 to 16 4
> 16 8

Intel Harpertown

MEMS benchmark 1
Strong Scalability on Intel Nehalem for a MEMS system of order ~ 23M (benchmark 2)

Speed improvement: ~ 7.7
Efficiency: ~ 48%

Time (sec)

Number of cores

64 128 256 512 1024

(128 nodes)
A Parallel Symmetric Eigenvalue Problem Solver: TraceMIN
The Trace minimization scheme:

\[ Ax = \lambda Bx; \quad \text{obtain the } p \text{ smallest eigenpairs} \]

\[ A = A^T; \quad B: \text{s.p.d} \]

\[
\begin{align*}
\min_y & \quad tr(Y^TAY) = \sum_{i=1}^{p} \lambda_i \\
\text{s.t.} & \quad Y^TBY = I_p \\
& \quad \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_p < \lambda_{p+1} \leq \cdots \leq \lambda_n \\
& \quad Y \in \mathbb{R}^{n \times p}; \quad p \ll n.
\end{align*}
\]

A.S. & J. Wisniewski: SINUM, 1982
\[ Y_k^T A Y_k = \Sigma_k = \text{diag}(\sigma_1^{(k)}, \ldots, \sigma_p^{(k)}) \]

\[ Y_k^T B Y_k = I_p \]

\[ Y_{k+1} = (Y_k - \Delta_k)S_k \]

\[
\begin{align*}
\text{min} & \quad \text{tr}[(Y_k - \Delta_k)^T A(Y_k - \Delta_k)] \\
\text{s.t.} & \quad Y_k^T B \Delta_k = 0
\end{align*}
\]

**Note:** if \( A \) were s.p.d. we have \( p \) indep. problems of the form:

\[
\begin{align*}
\text{min} & \quad (y_j^{(k)} - d_j^{(k)})^T A(y_j^{(k)} - d_j^{(k)}) \\
\text{s.t.} & \quad Y_k^T B d_j^{(k)} = 0 \quad j = 1, 2, \ldots, p
\end{align*}
\]
TraceMin (Outer iterations)

• relative residual $\leq \varepsilon_{\text{out}}$

• **form a section**

\[
Y^T AY = \Sigma; Y^T BY = I_p
\]

• **solve**

\[
\begin{pmatrix}
A & BY \\
Y^T B & O
\end{pmatrix}
\begin{pmatrix}
Y - \Delta \\
- L
\end{pmatrix} =
\begin{pmatrix}
O \\
I_p
\end{pmatrix}
\]
solve

\[
\begin{pmatrix}
A & BY_k \\
Y_k^T B & O
\end{pmatrix}
\begin{pmatrix}
\Delta_k \\
L_k
\end{pmatrix} =
\begin{pmatrix}
AY_k \\
O
\end{pmatrix}
\]

or

\[
\begin{pmatrix}
A & BY_k \\
Y_k^T B & O
\end{pmatrix}
\begin{pmatrix}
Y_k - \Delta_k \\
-L_k
\end{pmatrix} =
\begin{pmatrix}
O \\
I_p
\end{pmatrix}
\]

• different schemes & preconditioners.

• TraceMin does not require obtaining solutions with low relative residuals.
with shifts chosen from

$$\Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_p)$$

$$(A - \nu_j B)x_j = (\lambda - \nu_j)Bx_j$$

- convergence rate is ultimately cubic.

- $\nu_j$'s can be chosen to maintain global convergence.
TraceMIN vs. Trilinos

• We compare our TraceMIN parallel eigensolver against two counterparts in Sandia’s parallel Trilinos library:
  
  LOBPCG & Block Krylov-Schur

For two problems:

• Generic 3-D discretization of the Poisson operator on a cube (need lowest 4 eigenpairs),
• Predicting car body dynamics at high frequencies (an MSC/NASTRAN benchmark) (need lowest 1000 eigenpairs)
$A x = \lambda x$; lowest 4 eigenpairs

$A$: Poisson operator

Matrix size: 64 million

at 512 cores (64 nodes):

- Block Krylov-Schur.....681 sec.
- LOBPCG..................508
- TraceMIN..................57

ratios $\sim 12 : 9 : 1$

achieved rel. res. $= 10^{-5}$
Obtaining selected eigenpairs

• A generalized symmetric eigenvalue problem resulting from studying car body dynamics at higher frequencies:
  – $A \mathbf{x} = \lambda B \mathbf{x}$
  – $A$, $B$ are ill-conditioned ($\kappa \sim O(10^{12})$)
  – sizes: 1.5 M and 7.2 M
Sparsity structure of $A$ and $B$

$n \sim 1.5$ Million
$A x = \lambda B x$

Matrix size: 1.5 million

$B$ is computationally singular

**Car Body Problem**

*TraceMIN on one node: 8070 sec*

*TraceMIN on 64 nodes: 248 sec*

*Speed improv. = 32.5*

Both LOBPCG & BKS failed for this problem!
Sampling the spectrum via TraceMIN
4 eigenpairs closest to $\alpha_j$, $j = 1, 2, \ldots, 100$

(1.5 M problem)

- 100 nodes – 1 MPI process/node (12 cores)
- 12 threads/MPI process
- One Pardiso factorization per MPI task
- Total # of eigenpairs computed: 317

<table>
<thead>
<tr>
<th>Time in seconds</th>
<th>Relative Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td>21</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>22</td>
<td>$10^{-9}$</td>
</tr>
</tbody>
</table>
7.2 million Car Body Problem

• \( A \mathbf{x} = \lambda B \mathbf{x} \)

• Both LOBPCG and BKS in Trilinos failed to solve this generalized eigenvalue problem

• TraceMIN time on 2 nodes: 632 seconds

• TraceMIN time on 64 nodes: 38 seconds

• Speed improvement: \( \sim 17 \)

• Efficiency: \( \sim 53\% \)
Thank you!
Generating the weighted graph Laplacian

• **Case 1:**
  – A is a symmetric matrix of order n
  – B = A
  – The weighted Laplacian matrix L is given by:
    • $L(i,i) = \sum |B(i,k)|$; for $k = 1,2,...,n; k \neq i$
    • $L(i,j) = -|B(i,j)|$; for $i \neq j$

• **Case 2:**
  – A is nonsymmetric
  – $B = (|A| + |A^T|)/2$
  – L is obtained as in Case 1.
The Fiedler vector

• Obtain the eigenvector of the second smallest eigenvalue of $L x = \lambda x$:

$$\lambda := \{0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_n\}$$

• The sorting process of the Fiedler vector, based on the values of its entries, provides the permutation needed for weighted spectral reordering
A Parallel Weighted Spectral Reordering Scheme:

TraceMIN-Fiedler*

* Murat Manguoglu et. al.
TraceMIN-Fiedler

• \( Lx = \lambda x \); \( L \) is s.p.s.d.
• Minimize \( \text{tr}(Y^T L Y) \) s.t. \( (Y^T Y) = I_p \)

\[
\text{solution: } \min \text{tr}(Y^TLY) = \sum \lambda_j \quad (j=1,2,\ldots,p)
\]

\[
0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_p < \lambda_{p+1} \leq \ldots \leq \lambda_n
\]

Most time consuming kernel in each TraceMIN-Fiedler iteration is solving: \( LW = Y \) via PCG