Mixed Precision Methods

- Mixed precision, use the lowest precision required to achieve a given accuracy outcome
  - Improves runtime, reduce power consumption, lower data movement
  - Reformulate to find correction to solution, rather than solution; $\Delta x$ rather than $x$.

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

\[ x_{i+1} - x_i = -\frac{f(x_i)}{f'(x_i)} \]
Exploit 32 bit floating point as much as possible

- Correct or update the solution with selective use of 64 bit floating point to provide a refined results

- Intuitively:
  - Compute a 32 bit result,
  - Calculate a correction to 32 bit result using selected higher precision and,
  - Perform the update of the 32 bit results with the correction using high precision.
Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, $Ax = b$, can work this way.

  $LU = lu(A)$  \hspace{1cm} $O(n^3)$
  $x = L\backslash(U\backslash b)$  \hspace{1cm} $O(n^2)$
  $r = b - Ax$  \hspace{1cm} $O(n^2)$

  WHILE $|| r ||$ not small enough

  $z = L\backslash(U\backslash r)$  \hspace{1cm} $O(n^2)$
  $x = x + z$  \hspace{1cm} $O(n^1)$
  $r = b - Ax$  \hspace{1cm} $O(n^2)$

END

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, $Ax = b$, can work this way.

$L U = lu(A)$ \hspace{2cm} SINGLE $O(n^3)$

$x = L\backslash(U\backslash b)$ \hspace{2cm} SINGLE $O(n^2)$

$r = b - Ax$ \hspace{2cm} DOUBLE $O(n^2)$

WHILE $||r||$ not small enough

$z = L\backslash(U\backslash r)$ \hspace{2cm} SINGLE $O(n^2)$

$x = x + z$ \hspace{2cm} DOUBLE $O(n^1)$

$r = b - Ax$ \hspace{2cm} DOUBLE $O(n^2)$

END

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

- Requires extra storage, total is 1.5 times normal;
- $O(n^3)$ work is done in lower precision
- $O(n^2)$ work is done in high precision
- Problems if the matrix is ill-conditioned in sp; $O(10^8)$
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

**GPU**
- K20c: 13 MP @0.7 GHz, peak 1165 GFlop/s

**CPU**
- Genuine Intel: (2x8 @2.60GHz, peak 333 GFlop/s)

Matrix size vs. GFlop/s graph is shown with two lines:
- SP Solve (blue line)
- DP Solve (red line)

Graph data points:
- 2048: SP Solve ~ 600 GFlop/s, DP Solve ~ 400 GFlop/s
- 4032: SP Solve ~ 900 GFlop/s, DP Solve ~ 600 GFlop/s
- 6016: SP Solve ~ 1200 GFlop/s, DP Solve ~ 800 GFlop/s
- 8192: SP Solve ~ 1500 GFlop/s, DP Solve ~ 1000 GFlop/s
- 10000: SP Solve ~ 1800 GFlop/s, DP Solve ~ 1200 GFlop/s
- 12000: SP Solve ~ 2100 GFlop/s, DP Solve ~ 1400 GFlop/s
- 14000: SP Solve ~ 2400 GFlop/s, DP Solve ~ 1600 GFlop/s
- 16000: SP Solve ~ 2700 GFlop/s, DP Solve ~ 1800 GFlop/s
- 17984: SP Solve ~ 3000 GFlop/s, DP Solve ~ 2000 GFlop/s
- 20000: SP Solve ~ 3300 GFlop/s, DP Solve ~ 2200 GFlop/s
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

- SP Solve
- DP Solve (MP Iter.Ref.)
- DP Solve

Matrix size

GPU K20c  
(13 MP @0.7 GHz,  peak 1165 GFlop/s)

CPU Genuine Intel (2x8 @2.60GHz, peak 333 GFlop/s)
Sparse Direct Solver and Iterative Refinement

MUMPS package based on multifrontal approach which generates small dense matrix multiplies

Tim Davis’s Collection, n=100K - 3M
Sparse Iterative Methods (PCG)

- **Outer/Inner Iteration**

  Outer iterations using 64 bit floating point

  Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

  for $i = 1, 2, \ldots$

  solve $Mz^{(i-1)} = r^{(i-1)}$

  $\rho_{i-1} = r^{(i-1)T}z^{(i-1)}$

  if $i = 1$

  $p^{(1)} = z^{(0)}$

  else

  $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$

  $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$

  endif

  $q^{(i)} = Ap^{(i)}$

  $\alpha_i = \rho_{i-1}/p^{(i)T}q^{(i)}$

  $x^{(i)} = x^{(i-1)} + \alpha_ip^{(i)}$

  $r^{(i)} = r^{(i-1)} - \alpha_iq^{(i)}$

  check convergence; continue if necessary

  end

  Inner iteration:

  In 32 bit floating point

  Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

  for $i = 1, 2, \ldots$

  solve $Mz^{(i-1)} = r^{(i-1)}$

  $\rho_{i-1} = r^{(i-1)T}z^{(i-1)}$

  if $i = 1$

  $p^{(1)} = z^{(0)}$

  else

  $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$

  $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$

  endif

  $q^{(i)} = Ap^{(i)}$

  $\alpha_i = \rho_{i-1}/p^{(i)T}q^{(i)}$

  $x^{(i)} = x^{(i-1)} + \alpha_ip^{(i)}$

  $r^{(i)} = r^{(i-1)} - \alpha_iq^{(i)}$

  check convergence; continue if necessary

  end

- **Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**
Mixed Precision Computations for Sparse Inner/Outer-type Iterative Solvers

**Speedups** for mixed precision
Inner SP/Outer DP (SP/DP) iter. methods vs DP/DP (CG^2, GMRES^2, PCG^2, and PGMRES^2 with diagonal prec.)
*(Higher is better)*

**Iterations** for mixed precision
SP/DP iterative methods vs DP/DP *(Lower is better)*

**Machine:**
Intel Woodcrest (3GHz, 1333MHz bus)

**Stopping criteria:**
Relative to \( r_0 \) residual reduction \( (10^{-12}) \)
Reduce Communication

- Some factorization methods require pivoting to maintain stability
  - LU (general matrix) and LDLᵀ (symmetric indefinite matrix)
- Cost of pivoting can be high

Cost of partial pivoting in LU factorization (MAGMA)
1. Quad-Core Intel Core2 Q9300 @ 2.50 GHz - GPU C2050 @ 1.15 GHz
Techniques to Reduce Communication

• Communication in pivoting can be reduced by using tournament pivoting
  ▪ [Grigori, Demmel, Xiang, SIMAX 2011]

• We can remove completely the pivoting by preprocessing the system by randomization (O(n^2) flops)
  ▪ Transform the original matrix into a matrix “sufficiently random” so that, with a probability close to 1, pivoting is not needed
  ▪ [Baboulin, JD, Herrmann, Tomov, TOMS 2012]
Randomization

• To Solve $Ax = b$
  ▪ Compute $A_r = U^TAV$, with $U$ and $V$ random matrices
  ▪ Factor $A_r$ (without pivoting GENP)
  ▪ Solve $A_r y = U^Tb$ and then solve $x =Vy$
  ▪ Apply Iterative Refinement to correct & verify accuracy

• $U$ and $V$ are Recursive Butterfly Matrices

• Properties
  ▪ Randomization is cheap ($O(n^2)$ operations)
  ▪ GENP is fast (Communication is reduced)
  ▪ Accuracy is in practice similar to GE w/PP
    ▪ When doing iterative refinement
Butterfly Matrix

A **butterfly matrix** is defined as any $n$-by-$n$ matrix of the form:

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix}$$

where $R$ and $S$ are random diagonal matrices.

![Butterfly Diagram](image)

Remark:

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} I_{n/2} & I_{n/2} \\ I_{n/2} & -I_{n/2} \end{pmatrix} \begin{pmatrix} R_0 & 0 \\ 0 & R_1 \end{pmatrix}$$
Comparison of componentwise backward error for PRBT and other solvers

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Cond</th>
<th>GENP</th>
<th>GEPP</th>
<th>QR</th>
<th>PRBT</th>
<th>REC</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment</td>
<td>$4 \cdot 10^4$</td>
<td>$1.28 \cdot 10^{-14}$</td>
<td>$2.28 \cdot 10^{-15}$</td>
<td>$2.99 \cdot 10^{-16}$</td>
<td>$2.81 \cdot 10^{-16}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gfpp</td>
<td>$5 \cdot 10^2$</td>
<td>$9.01 \cdot 10^{-01}$</td>
<td>$6.88 \cdot 10^{-01}$</td>
<td>$1.06 \cdot 10^{-16}$</td>
<td>$1.27 \cdot 10^{-16}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>chebspec</td>
<td>$2 \cdot 10^{14}$</td>
<td>$1.19 \cdot 10^{-15}$</td>
<td>$3.29 \cdot 10^{-16}$</td>
<td>$5.22 \cdot 10^{-15}$</td>
<td>$3.23 \cdot 10^{-14}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>circul</td>
<td>$1 \cdot 10^3$</td>
<td>$1.74 \cdot 10^{-13}$</td>
<td>$1.66 \cdot 10^{-15}$</td>
<td>$2.66 \cdot 10^{-15}$</td>
<td>$2.66 \cdot 10^{-15}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>condex</td>
<td>$1 \cdot 10^2$</td>
<td>$7.32 \cdot 10^{-15}$</td>
<td>$5.98 \cdot 10^{-15}$</td>
<td>$8.34 \cdot 10^{-15}$</td>
<td>$6.50 \cdot 10^{-15}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fiedler</td>
<td>$7 \cdot 10^5$</td>
<td>Fail</td>
<td>$2.11 \cdot 10^{-15}$</td>
<td>$1.54 \cdot 10^{-14}$</td>
<td>$7.90 \cdot 10^{-15}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Hadamard</td>
<td>$1 \cdot 10^0$</td>
<td>$0 \cdot 10^0$</td>
<td>$0 \cdot 10^0$</td>
<td>$7.58 \cdot 10^{-16}$</td>
<td>$8.33 \cdot 10^{-15}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>normaldata</td>
<td>$3 \cdot 10^4$</td>
<td>$2.03 \cdot 10^{-12}$</td>
<td>$6.30 \cdot 10^{-15}$</td>
<td>$2.38 \cdot 10^{-16}$</td>
<td>$3.30 \cdot 10^{-16}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>orthog</td>
<td>$1 \cdot 10^0$</td>
<td>$5.64 \cdot 10^{-01}$</td>
<td>$4.33 \cdot 10^{-15}$</td>
<td>$3.70 \cdot 10^{-16}$</td>
<td>$4.31 \cdot 10^{-16}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>randcorr</td>
<td>$3 \cdot 10^3$</td>
<td>$5.12 \cdot 10^{-16}$</td>
<td>$4.04 \cdot 10^{-16}$</td>
<td>$5.73 \cdot 10^{-16}$</td>
<td>$5.92 \cdot 10^{-16}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>toeppd</td>
<td>$7 \cdot 10^5$</td>
<td>$2.53 \cdot 10^{-13}$</td>
<td>$2.60 \cdot 10^{-15}$</td>
<td>$8.39 \cdot 10^{-15}$</td>
<td>$5.71 \cdot 10^{-15}$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Foster</td>
<td>$5 \cdot 10^2$</td>
<td>$1 \cdot 10^0$</td>
<td>$1 \cdot 10^0$</td>
<td>$1.90 \cdot 10^{-16}$</td>
<td>$3.30 \cdot 10^{-16}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$[-1, 1]$</td>
<td>$2 \cdot 10^3$</td>
<td>$2.19 \cdot 10^{-11}$</td>
<td>$5.19 \cdot 10^{-15}$</td>
<td>$2.33 \cdot 10^{-16}$</td>
<td>$2.35 \cdot 10^{-16}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$[0, 1]$</td>
<td>$4 \cdot 10^4$</td>
<td>$1.97 \cdot 10^{-12}$</td>
<td>$2.85 \cdot 10^{-15}$</td>
<td>$2.15 \cdot 10^{-15}$</td>
<td>$1.79 \cdot 10^{-15}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$[-1, 1]$</td>
<td>$4 \cdot 10^3$</td>
<td>Fail</td>
<td>$3.96 \cdot 10^{-15}$</td>
<td>$2.38 \cdot 10^{-16}$</td>
<td>$2.70 \cdot 10^{-16}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>${0, 1}$</td>
<td>$5 \cdot 10^4$</td>
<td>Fail</td>
<td>$4.39 \cdot 10^{-15}$</td>
<td>$2.19 \cdot 10^{-15}$</td>
<td>$1.09 \cdot 10^{-15}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Turing</td>
<td>$5 \cdot 10^{19}$</td>
<td>$0 \cdot 10^0$</td>
<td>$0 \cdot 10^0$</td>
<td>$7.16 \cdot 10^{-13}$</td>
<td>$1.05 \cdot 10^{-14}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>${</td>
<td>i-j</td>
<td>}$</td>
<td>$7 \cdot 10^5$</td>
<td>Fail</td>
<td>$3.33 \cdot 10^{-16}$</td>
<td>$1.54 \cdot 10^{-14}$</td>
<td>$6.05 \cdot 10^{-15}$</td>
</tr>
<tr>
<td>max(i,j)</td>
<td>$3 \cdot 10^6$</td>
<td>$2.16 \cdot 10^{-14}$</td>
<td>$1.21 \cdot 10^{-15}$</td>
<td>$1.46 \cdot 10^{-14}$</td>
<td>$2.27 \cdot 10^{-15}$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Performance of Parallel Randomize Butterfly Transformation

[ Baboulin, JD, Herrmann, Tomov, TOMS 2012 ]
**Eigenproblem Solvers in on GPUs**

- $\mathbf{A} \mathbf{x} = \lambda \mathbf{x}$
  - Quantum mechanics (Schrödinger equation)
  - Quantum chemistry
  - Principal component analysis (in data mining)
  - Vibration analysis (of mechanical structures)
  - Image processing, compression, face recognition
  - Eigenvalues of graph, e.g., in Google’s page rank

- **Need to solve it fast**

  **Current MAGMA results:**
  MAGMA with 1 GPU can be 12x faster vs vendor libraries on state-of-art multicore systems


Approaches to Two-sided fact.

• One stage
  - Directly factor to bidiagonal/tridiagonal/Hessenberg forms
  - Regular computation (good for GPUs)
  - But requires Level 2 BLAS
Approaches to Two-sided fact.

- **One stage**
  - Directly factor to bidiagonal/tridiagonal/Hessenberg forms
  - Regular computation (good for GPUs)
  - But requires Level 2 BLAS

- **Two stage**
  - 1\(^{st}\) go to band reduction and 2\(^{nd}\) do bulge chasing to the form desired
  - Leads to irregular computation (bulge chasing)
  - But avoids Level 2 BLAS
## Toward fast Eigensolver

### Characteristics
- Too many Blas-2 op,
- Relies on panel factorization,
- Bulk sync phases,
- Memory bound algorithm.

### Chart
- Flops formula: $n^3/3 \times \text{time}$
- Higher is faster

Keeneland system, using one node
- 3 NVIDIA GPUs (M2090 @ 1.1 GHz, 5.4 GB)
- 2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)

### Reference
Toward fast Eigensolver

 Characteristics

• Blas-2 GEMV moved to the GPU,
• Accelerate the algorithm by doing all BLAS-3 on GPU,
• $B_{ulk}$ sync phases,
• $M_{emory}$ bound algorithm.

Toward fast Eigensolver

**Characteristics**

- **Stage 1:** BLAS-3, increasing computational intensity,
- **Stage 2:** BLAS-1.5, new cache friendly kernel,
- 4X/12X faster than standard approach,
- Bottleneck: if all Eigenvectors are required, it has 1 back transformation extra cost.


Flops formula: \( n^3/3 \times \text{time} \)

Higher is faster

Keeneland system, using one node
3 NVIDIA GPUs (M2090@ 1.1 GHz, 5.4 GB)
2 x 6 Intel Cores (X5660 @ 2.8 GHz, 23 GB)
Summary

- These are old ideas
- **Major Challenges are ahead for extreme computing**
  - Power
  - Levels of Parallelism
  - Communication
  - Hybrid
  - Fault Tolerance
  - ... and many others not discussed here

- **Not just a programming assignment or implementation detail.**

- **This opens up many new opportunities for applied mathematicians and computer scientists**
Collaborators / Software / Support

- **PLASMA**
  http://icl.cs.utk.edu/plasma/

- **MAGMA**
  http://icl.cs.utk.edu/magma/

- **Quark (RT for Shared Memory)**
  http://icl.cs.utk.edu/quark/

- **DAGue (RT for Distributed Memory)**
  http://icl.cs.utk.edu/dague/

- Collaborating partners
  University of Tennessee, Knoxville
  University of California, Berkeley
  University of Colorado, Denver
  INRIA, France
  KAUST, Saudi Arabia

These tools are being applied to a range of applications beyond dense LA:
Sparse direct, Sparse iterations methods and Fast Multipole Methods