Using Mixed Precision in Numerical Computations to Speedup Linear Algebra Solvers

Jack Dongarra, UTK/ORNL/U Manchester  
Azzam Haidar, Nvidia  
Nick Higham, U of Manchester  
Stan Tomov, UTK

Background

• My interest in mixed precision began with my dissertation ...
  
  ▪ Improving the Accuracy of Computed Matrix Eigenvalues
    • Compute the eigenvalues and eigenvectors in low precision then improve selected values/vectors to higher precision for $O(n^2)$ ops using the matrix decomposition
  
  ▪ Extended to singular values, 1983
  ▪ Algorithm in TOMS 710, 1992
IBM’s Cell Processor - 2004

- **9 Cores**
  - Power PC at 3.2 GHz
  - 8 SPEs

- **204.8 Gflop/s peak!**
  - The catch is that this is for 32 bit fl pt; (Single Precision SP)
  - 64 bit fl pt peak at 14.6 Gflop/s
    - 14 times slower than SP; factor of 2 because of DP and 7 because of latency issues

The SPEs were fully IEEE-754 compliant in double precision.
In single precision, they only implement round-towards-zero, denormalized numbers are flushed to zero and NaNs are treated like normal numbers.
Mixed Precision Idea Goes Something Like This...

- Exploit 32 bit floating point as much as possible.
  - Especially for the bulk of the computation
- 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.
Iterative refinement for dense systems, \( Ax = b \), can work this way.

\[
L U = \text{lu}(A) \\
x = U \backslash (L \backslash b) \\
r = b - Ax \text{ (with original } A) 
\]

\[
\text{WHILE } ||r|| \text{ not small enough} \\
\quad 1. \text{ find a correction } "z" \text{ to adjust } x \text{ that satisfy } Az = r \\
\quad 2. \quad x = x + z \\
\quad 3. \quad r = b - Ax \text{ (with original } A) \\
\text{END}
\]

**FP32 precision**
- \( O(n^3) \)
- \( O(n^2) \)

**FP64 precision**
- \( O(n^2) \)

**Idea:** use low precision to compute the expensive flops (\( LU \ O(n^3) \)) and then iteratively refine (\( O(n^2) \)) the solution in order to achieve the FP64 arithmetic.

- 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.
- Need a copy of the original matrix to compute residual (r) and matrix cannot be too badly conditioned.

IBM Cell 3.2 GHz, Ax = \( b \)

**SP Peak (204 Gflop/s)**

**DP Peak (15 Gflop/s)**

- **SP Theoretical Peak**
- **DP Theoretical Peak**

- **8 SGEMM (Embarrassingly Parallel)**

Matrix Size vs. GFlop/s graph.
IBM Cell 3.2 GHz, Ax = b

Matrix Size

GFlop/s

SP Peak (204 Gflop/s)
SP Ax=b IBM
DP Peak (15 Gflop/s)
DP Ax=b IBM

8 SGEMM (Embarrassingly Parallel)

SP Ax=b Performance

DP Ax=b Performance

SP Ax=b

.30 secs

3.9 secs
IBM Cell 3.2 GHz, Ax = b

8 SGEMM (Embarrassingly Parallel)

Matrix Size

GFlop/s

8.3X Speedup

Mixed Precision Performance

3.9 secs

.47 secs

.30 secs

SP Peak (204 Gflop/s)

SP Ax=b IBM

DSGESV

DP Peak (15 Gflop/s)

DP Ax=b IBM

8 SGEMM (Embarrassingly Parallel)
Intriguing Potential

- Exploit lower precision as much as possible
  - Payoff in performance
    - Faster floating point
    - Less data to move
- Automatically switch between SP and DP to match the desired accuracy
  - Compute solution in SP and then a correction to the solution in DP
- Potential for GPU, FPGA, special purpose processors
  - Use as little precision as you can get away with and improve the accuracy
- Linear systems and Eigenvalue, optimization problems, where Newton’s method is used.

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

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

\[
z = - A \backslash (b - Ax)
\]

\[
z \text{ (correction, } x_{i+1} - x_i \text{ )}
\]

\[
x_{i+1}
\]
Machine Learning in Computational Science

Many fields are beginning to adopt machine learning to augment modeling and simulation methods

- Climate
- Biology
- Drug Design
- Epidemiology
- Materials
- Cosmology
- High-Energy Physics
Deep Learning Needs Small Matrix Operations

Matrix Multiply is the time consuming part.

Convolution Layers and Fully Connected Layers require matrix multiply

There are many GEMM’s of small matrices, perfectly parallel, can get by with 16-bit floating point

Convolution Step
In this case 3x3 GEMM

Fully Connected Classification
Nvidia Volta Peak Rates

- Four Performance levels for the different precision
  - 64 bit floating point (FMA): 7.5 Tflop/s peak
  - 32 bit floating point (FMA): 15 Tflop/s peak
  - 16 bit floating point (FMA): 30 Tflop/s peak
  - 16 bit floating point w/Tensor core: 120 Tflop/s peak

Tensor Core, special hardware for:
  Mixed Precision Matrix Multiply
  4x4 Matrices

\[ D = AB + C \]
VOLTA TENSOR OPERATION

FP16 storage/input  Full precision product  Sum with FP32 accumulator  Convert to FP32 result

Also supports FP16 accumulator mode for inferencing
Mixed Precision

- Today many precisions to deal with (IEEE Standard)

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
<th>$u = 2^{-t}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>half</td>
<td>16 bits</td>
<td>$10^{±5}$</td>
<td>$2^{-11} \approx 4.9 \times 10^{-4}$</td>
</tr>
<tr>
<td>single</td>
<td>32 bits</td>
<td>$10^{±38}$</td>
<td>$2^{-24} \approx 6.0 \times 10^{-8}$</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>$10^{±308}$</td>
<td>$2^{-53} \approx 1.1 \times 10^{-16}$</td>
</tr>
<tr>
<td>quadruple</td>
<td>128 bits</td>
<td>$10^{±4932}$</td>
<td>$2^{-113} \approx 9.6 \times 10^{-35}$</td>
</tr>
</tbody>
</table>

- Note the number range with half precision (16 bit fl.pt.)

largest fl pt number

65,504

largest fl pt number

$O(10^{38})$
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100

- `dgemm` achieve about 6.4 Tflop/s
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100

- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100

- `dgemm` achieve about 6.4 Tflop/s
- `sgemm` achieve about 14 Tflop/s
- `hgemm` achieve about 27 Tflop/s

Matrix matrix multiplication GEMM

\[ C = \alpha A + \beta B \]
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100

- dgemm achieve about 6.4 Tflop/s
- sgemm achieve about 14 Tflop/s
- hgemm achieve about 27 Tflop/s
- Tensor cores gemm reach about 85 Tflop/s

Matrix matrix multiplication GEMM

\[ C = \alpha A B + \beta C \]
Leveraging Half Precision in HPC on V100

Study of the Matrix Matrix multiplication kernel on Nvidia V100

- `dgemm` achieve about 6.4 Tflop/s
- `sgemm` achieve about 14 Tflop/s
- `hgemm` achieve about 27 Tflop/s
- Tensor cores `gemm` reach about 85 Tflop/s
Leveraging Half Precision in HPC on V100

Study of the rank k update used by the LU factorization algorithm on Nvidia V100

- In LU factorization need matrix multiple but operations is a rank-k update computing the Schur complement

```
[ ] = [ ] + [ ]
```

Rank-k GEMM needed by LU does not perform as well as square but still OK
Leveraging Half Precision in HPC on V100

Study of the LU factorization algorithm on Nvidia V100

Performance of the LU factorization with different precision

- LU factorization is used to solve a linear system $Ax=b$
  
  $A \times x = b$

  
  $LUx = b$

  
  $Ly = b$

  
  then

  
  $Ux = y$

For the LU, half precision used only in GEMM, Panel and TRSM in SP.
Leveraging Half Precision in HPC on V100

Use Mixed Precision algorithms

- Achieve higher performance
  - faster time to solution (benefit from operations and data movement)
- Reduce power consumption by decreasing the execution time
  - Energy Savings !!!

- Reformulate to find correction to solution, rather than solution; \( \Delta x \) rather than \( x \).

A. Haidar, P. Wu, S. Tomov, J. Dongarra,
Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers,

A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham,
Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers, SC-18, Dallas, IEEE.
Iterative refinement for dense systems, $Ax = b$, can work this way.

\begin{align*}
L U &= \text{lu}(A) \\
x &= L \backslash (L \backslash b) \\
r &= b - Ax \text{ (with original A)}
\end{align*}

**WHILE** $\|r\|$ not small enough

1. find a correction “z” to adjust $x$ that satisfy $Az=r$
   - Classical Iterative Refinement
   - GMRes
   - GMRes preconditioned by the LU to solve $Az=r$

2. $x = x + z$
3. $r = b - Ax \text{ (with original A)}$

**END**

Higham and Carson showed can solve the inner problem with iterative method and not infect the solution.

- 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.
- Need the original matrix to compute residual ($r$) and matrix cannot be too badly conditioned

Iterative refinement for dense systems, \( Ax = b \), can work this way. 

\[ \begin{align*} 
L U &= \text{lu}(A) \\
x &= U \setminus (L \setminus b) \\
\text{GMRes} \text{ preconditioned by the LU to solve } Ax = b 
\end{align*} \]

\( LU \) \( O(n^3) \) 
\( x \) \( O(n^2) \) 
\( \text{GMRes} \) \( O(n^2) \) 
\( \text{FP64 precision} \) \( O(n^2) \)

**Idea:** use low precision to compute the expensive flops \( LU \) \( O(n^3) \) and then iteratively refine \( O(n^2) \) the solution in order to achieve the FP64 arithmetic.

Ø Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.

Ø It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

Ø Need the original matrix to compute residual \( r \) and matrix cannot be too badly conditioned.
Leveraging Half Precision in HPC on V100 solving linear system $Ax = b$

For $s = 0, nb, .. N$

1. panel factorize
2. update trailing matrix

- Panel Factorization performed with 32 bit fl pt
  - Done using MAGMA on the front-end system
- TRSM - Triangular solve performed with 32 bit fl pt
  - Done using V100 (no Tensor core)
- GEMM - Matrix Multiply performed with 16 bit fl pt
  - Done on V100 with Tensor cores

Most of the performance comes from GEMM using 16 bit fl pt
Tensor Core Accelerated IRS solving linear system $Ax = b$ Performance

Behavior

- solving $Ax = b$ using FP64 LU
- solving $Ax = b$ using FP32 LU and iterative refinement to achieve FP64 accuracy
- solving $Ax = b$ using FP16 Tensor Cores LU and iterative refinement to achieve FP64 accuracy

Results obtained using CUDA 10.2 and GV100 GPU.

Problem generated with an arithmetic distribution of the singular values $\sigma_i = 1 - \left( \frac{i-1}{n-1} \right) (1 - \frac{1}{\text{cond}})$ and positive eigenvalues.
Tensor Core Accelerated IRS solving linear system $Ax = b$ Behavior

![Graph showing performance of solving $Ax=b$ to the FP64 accuracy](image)

- **Performance**
  - Harder case
    - Solving $Ax = b$ using **FP64 LU**
    - Solving $Ax = b$ using **FP32 LU** and iterative refinement to achieve FP64 accuracy
    - Solving $Ax = b$ using **FP16 Tensor Cores LU** and iterative refinement to achieve FP64 accuracy

Results obtained using CUDA 10.2 and GV100 GPU.

Problem generated with an clustered distribution of the singular values $\sigma = [1, \cdots, 1, \frac{1}{cond}]$.

Flops $= 2n^3/(3 \text{ time})$, meaning twice higher is twice faster.
Convergence history of the iterative refinement solvers to achieve FP64 solution accuracy.

Interestingly, the **FP16→64 (Tensor Cores Accelerated Iterative Refinement Solver)** converge to the FP64 accuracy with only slightly more iterations than FP32→64 and also **outperforms** both the FP32→64 and the basic FP64 in term of time to solution.

Scaling help the **FP16→64 (Tensor Cores)** convergence.

Results obtained using CUDA 10.2 and GV100 GPU.
Leveraging Half Precision in HPC on V100

Use Mixed Precision algorithms

Idea: use lower precision to compute the expensive flops ($LU \ O(n^3)$) and then iteratively refine the solution in order to achieve the FP64 arithmetic

- Achieve higher performance $\rightarrow$ faster time to solution
- Reduce power consumption by decreasing the execution time $\rightarrow$ Energy Savings !!!
Tensor Core Accelerated IRS solving linear system \( Ax = b \)

Problem generated with an arithmetic distribution of the singular values \( \sigma_i = 1 - \left( \frac{i-1}{n-1} \right) \left( 1 - \frac{1}{\text{cond}} \right) \) and positive eigenvalues.

Energy Efficiency

Mixed precision techniques can provide a large gain in energy efficiency:

- Power consumption for a matrix of size 40K
- The FP64 algorithm achieve 5.3 Tflop/s providing about 21 Gflops/Watts.
- The FP32\( \rightarrow \)64 algorithm achieve 10 Tflop/s providing about 40 Gflops/Watts.
- The FP16\( \rightarrow \)64 TC algorithm using Tensor Cores achieve 22 Tflop/s providing about 94 Gflops/Watts.

Results obtained using CUDA 10.2 and GV100 GPU.
Tensor Core Accelerated IRS
solving linear system $Ax = b$

Performance on a wider range of real-life problems

<table>
<thead>
<tr>
<th>name</th>
<th>Description</th>
<th>size</th>
<th>$\kappa_s(A)$</th>
<th>$\text{dgesv FP64}$ time(s)</th>
<th># iter</th>
<th>$\text{dgesv FP32}$ time(s)</th>
<th>speedup</th>
<th># iter</th>
<th>time (s)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>em192</td>
<td>radar design</td>
<td>26896</td>
<td>$10^6$</td>
<td>5.70</td>
<td>3</td>
<td>3.11</td>
<td>1.8328</td>
<td>10</td>
<td>2.05</td>
<td>2.7805</td>
</tr>
<tr>
<td>appu</td>
<td>NASA app benchmark</td>
<td>14000</td>
<td>$10^4$</td>
<td>0.43</td>
<td>2</td>
<td>0.27</td>
<td>1.5926</td>
<td>4</td>
<td>0.19</td>
<td>2.2632</td>
</tr>
<tr>
<td>ns3Da</td>
<td>3D Navier Stokes</td>
<td>20414</td>
<td>$7.6 \times 10^3$</td>
<td>1.12</td>
<td>2</td>
<td>0.69</td>
<td>1.6232</td>
<td>4</td>
<td>0.43</td>
<td>2.6047</td>
</tr>
<tr>
<td>nd6k</td>
<td>ND problem set</td>
<td>18000</td>
<td>$3.5 \times 10^2$</td>
<td>0.81</td>
<td>2</td>
<td>0.45</td>
<td>1.8000</td>
<td>3</td>
<td>0.30</td>
<td>2.7000</td>
</tr>
<tr>
<td>nd12k</td>
<td>ND problem set</td>
<td>36000</td>
<td>$4.3 \times 10^2$</td>
<td>5.36</td>
<td>2</td>
<td>2.75</td>
<td>1.9491</td>
<td>3</td>
<td>1.31</td>
<td>4.0916</td>
</tr>
<tr>
<td>Poisson</td>
<td>2D Poisson problem</td>
<td>32000</td>
<td>$2.1 \times 10^6$</td>
<td>3.81</td>
<td>2</td>
<td>2.15</td>
<td>1.7721</td>
<td>10</td>
<td>1.13</td>
<td>3.3717</td>
</tr>
<tr>
<td>Vlasov</td>
<td>2D Vlasov problem</td>
<td>22000</td>
<td>$8.3 \times 10^3$</td>
<td>1.65</td>
<td>2</td>
<td>0.95</td>
<td>1.7368</td>
<td>3</td>
<td>0.48</td>
<td>3.4375</td>
</tr>
</tbody>
</table>

Performance for real-life matrices from the SuiteSparse Collection and from dense matrix arising from radar design.
### System Performance
- Peak performance of 200 Pflop/s for modeling & simulation
- Peak performance of 3.3 Eflop/s for 16 bit floating point used in for data analytics, ML, and artificial intelligence

### Each node has
- 2 IBM POWER9 processors
  - Each w/22 cores
  - 2.3% performance of system
- 6 NVIDIA Tesla V100 GPUs
  - Each w/80 SMs
  - 97.7% performance of system
- 608 GB of fast memory
- 1.6 TB of NVMe memory

### The system includes
- 4608 nodes
  - 27,648 GPUs
  - Street value $8K each
- Dual-rail Mellanox EDR InfiniBand network
- 250 PB IBM Spectrum Scale file system transferring data at 2.5 TB/s
HPL-AI (A MIXED PRECISION BENCHMARK)

The HPL-AI benchmark seeks to highlight the emerging convergence of high-performance computing (HPC) and artificial intelligence (AI) workloads and highlight the advantages of mixed precision.

The benchmark is a combination of LU factorization (at lower precision) and iterative refinement method (like GMRes) to bring the solution back to 64-bit accuracy.

Iterative refinement for dense systems, \(Ax = b\), can work this way.

- \(L U = lu(A)\)
- \(x = U\backslash (L\backslash b)\)
- GMRes preconditioned by the LU to solve \(Ax=b\)

Recent Results Run at Scale...

• Mixed precision iterative refinement approach solved a matrix of order 10,091,520 on ORNL’s Summit system.
  – Composed of nodes made up of 2 IBM Power-9 processors (22 cores each) plus 6 Nvidia V100 GPUs (84 SMs each)
  – The run used 4500 nodes of Summit, 2,466,000 cores = 4500*(22*2 + 84*6)
  – Used a random matrix with large diagonal elements to insure convergence of the method.

• Mixed precision HPL achieved 550 PFLOPS or 3.7 X over DP precision HPL result on the Top500 (148 PFLOPS).
  – 53 Gflops/Watt

• Same accuracy compared to full 64 bit precision
Conclusion:

- We accelerated the solution of linear system $Ax = b$ solver using hardware-accelerated FP16 arithmetic on GPUs;

- We introduced a framework for exploiting mixed-precision FP16-FP32/FP64 iterative refinement solvers and describe the path to draw high-performance and energy-aware GPU implementations;
  - Ideas can be applied to other 1 sided reductions (LU, $LL^T$, $LDL^T$, QR) and also for 2 sided in the case of eigen (singular) values/vectors, (where are few are required).

- Our technique shows that a number of problems can be accelerated up to 4X by the usage of the FP16-TC or 2X using the FP32 arithmetic.

- We studied the energy-efficiency of our approach that showed significant energy savings, 5X energy savings using the FP16-TC compared to the FP64 implementation.

- We illustrated a technique to use V100 Tensor Cores FP16-TC that achieves FP64 accuracy at a highly efficient/accelerated performance equating to 74 Gflops/Watt and 24 Tflops/s.

- There is a rigorous error analysis to support everything


