Mixed Precision Dense Linear System Solvers
for High Performance Reconfigurable Computing

Tennessee Advanced Computing Laboratory
University of Tennessee

July 29th 2009

JunKyu Lee, Gregory D. Peterson, Robert J. Harrison, Robert J. Hinde

This work was partially supported by the National Science Foundation, grant NSF CHE-0625598.
I. Introduction
- What is a mixed precision solver?
- How has a mixed precision computations been developed? ⇔ What do we need to develop further?

II. Methodology
- What have we explored?
- What have we found?

III. Results
- What could be a possible RC architecture for this research?

IV. Future work

V. Conclusions
What is mixed precision solver? Why?

Mixed Precision Computation Method:
1. Employ more than one precision computation.
2. Lower precision (fast computation) for computationally intensive tasks and Higher precision (slow computation) for refining solution.

→ High performance (Lower precision computation)
→ Satisfactory numeric solution (Higher precision refinement)

Introduction
Methodology
Results
Future work
Conclusions
Mixed precision algorithm

To solve $Ax = b$;

Step 1: GEPP ($A$); $O(n^3)$ <= precision $P_I$;

Solve $LUx(1) = P \times b$; $O(n^2)$ <= precision $P_I$;

for $(i = 1$ to $x(i)$ accurate enough)
Step 2: $r(i) = b - A_h x(i)$; $O(n^2)$ <= precision $P_H$;
Step 3: $LUz(i) = P \times r(i)$; $O(n^2)$ <= precision $P_I$;
Step 4: $x(i) = x(i) + z(i)$; $O(n)$ <= precision $P_H$;
end

$P$ is a permutation matrix and $r$ is a residual vector.
To solve $Ax = b$;

**Step 1:** GEPP (A);

Solve $LUx(1) = P \times b$;

for $(i = 1$ to $x(i)$ accurate enough)$

**Step 2:** $r(i) = b - A_h x(i)$;

**Step 3:** $LUz(i) = P \times r(i)$;

**Step 4:** $x(i) = x(i) + z(i)$;

end
How have mixed precision methods been developed?

Previous Research

- Single precision for GEPP
- Double precision for GEPP
- Double precision refinement

Could we apply an arbitrary precision instead of just single or double precision?

Use FPGA!

(Arbitrary precision computation)


1. FPGA can employ arbitrary precision computation
   (Applying higher precision when a condition number of system is high)
2. Lower precision → Smaller, Faster ALUs → More ALUs
   (Significant performance difference for multiplication between lower precision and higher precision in FPGAs)

Mantissa: 16bits, Exponent: 7bits

<table>
<thead>
<tr>
<th>Problem size (n)</th>
<th>Average condition number</th>
<th>Average iterations</th>
<th>Failure (iterations&gt;30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>913</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>256</td>
<td>1818</td>
<td>5.1</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>4017</td>
<td>6.1</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>6196</td>
<td>6.3</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>9407</td>
<td>9.3</td>
<td>1/100</td>
</tr>
<tr>
<td>4096</td>
<td>22425</td>
<td>13.3</td>
<td>2/100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrices Size (J.Sun / Edelman)</th>
<th>Condition Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>913</td>
</tr>
<tr>
<td>256</td>
<td>1818</td>
</tr>
<tr>
<td>512</td>
<td>4017</td>
</tr>
<tr>
<td>1024</td>
<td>6196</td>
</tr>
<tr>
<td>2048</td>
<td>9407</td>
</tr>
<tr>
<td>4096</td>
<td>22425</td>
</tr>
</tbody>
</table>

Edelman’s analysis with random matrices: \( E(k(A)) = e^{\log(n) + 1.537} \)

<table>
<thead>
<tr>
<th>Data Formats</th>
<th>DSP48s</th>
<th>Frequency</th>
<th>Latency</th>
<th>GFLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>s52e11 (double)</td>
<td>16/96</td>
<td>237</td>
<td>21</td>
<td>1.42</td>
</tr>
<tr>
<td>s51e11</td>
<td>16/96</td>
<td>238</td>
<td>21</td>
<td>1.43</td>
</tr>
<tr>
<td>s50e11</td>
<td>9/96</td>
<td>245</td>
<td>19</td>
<td>2.61</td>
</tr>
<tr>
<td>s34e8</td>
<td>9/96</td>
<td>289</td>
<td>14</td>
<td>3.08</td>
</tr>
<tr>
<td>s33e8</td>
<td>4/96</td>
<td>292</td>
<td>9</td>
<td>7.01</td>
</tr>
<tr>
<td>s23e8 (single)</td>
<td>4/96</td>
<td>339</td>
<td>9</td>
<td>8.14</td>
</tr>
<tr>
<td>s17e8</td>
<td>4/96</td>
<td>370</td>
<td>9</td>
<td>8.88</td>
</tr>
<tr>
<td>s16e8</td>
<td>1/96</td>
<td>331</td>
<td>6</td>
<td>31.78</td>
</tr>
<tr>
<td>s16e7</td>
<td>1/96</td>
<td>352</td>
<td>6</td>
<td>33.79</td>
</tr>
<tr>
<td>s13e7</td>
<td>1/96</td>
<td>336</td>
<td>6</td>
<td>32.26</td>
</tr>
</tbody>
</table>

XC4LX160-10

Question (This is what the research is about)
If FPGA can employ arbitrary precision computation, what precision will be for GEPP? Could we predict success?

In this research, we address
1. Initial precision predictor: How to apply appropriate initial precision for first $O(n^3)$ computation for GEPP?
2. Failure manipulator: If the algorithm fails, how to predict the failure and how to fix it?
Exploit FPGA’s arbitrary precision computation

Fewer H/W resources in FPGAs
→ Lower Precision
→ Smaller, Faster ALUs
→ More ALUs

Initial precision predictor according to matrix characteristics

Failure manipulator

Mixed precision dense linear system solver for high performance reconfigurable computing
Guessing Condition Number of Matrix

System Matrix Characteristics

Decide Initial Precision

GEPP Generate L and U

Failure Manipulator

Compute Condition Number of Matrix

Work? Is Guess Right?

No

Failure

Yes

Done

Model
Initial Precision Predictor and Failure Manipulator

We note that

1. The condition number of a matrix is an important factor to predict success
2. The condition number of a random matrix increases with size
3. Computing a condition number from triangular matrices takes $O(n^2)$

I. Initial Precision Predictor
   ⇔ Note 1 and 2

II. Failure Manipulator
   ⇔ Note 1 and 3
Experiments with 1000 uniformly distributed random matrices (lagged Fibonacci generator) according to different matrix sizes

Different precision computations => Fix exponential bits, Change mantissa bits

What have we tried?

(Initial Precision Predictor)
1. Investigate success rate according to different size matrices with different precision computations for GEPP
2. Find out appropriate mantissa bit widths which guarantee 95% success rate according to different matrix sizes

(Failure Manipulator)
1. Investigate CDFs for condition numbers according to different size matrices
2. Find out the 95% condition numbers in CDFs
→ Relate the working precision (initial precision) to the condition numbers (failure manipulator)
I. Initial Working Precision

- Initial Precision (GEPP)
- 95% Success Rate

<table>
<thead>
<tr>
<th>Mantissa bit width</th>
<th>Matrix size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32x32</td>
</tr>
<tr>
<td>8 bits</td>
<td>19.1%</td>
</tr>
<tr>
<td>12 bits</td>
<td>85.5%</td>
</tr>
<tr>
<td>16 bits</td>
<td>98.6%</td>
</tr>
<tr>
<td>20 bits</td>
<td>99.8%</td>
</tr>
<tr>
<td>24 bits</td>
<td>100%</td>
</tr>
<tr>
<td>More than 28 bits</td>
<td>100%</td>
</tr>
</tbody>
</table>

- 95% $W_r$:
  - 14.90 bits
  - 16.09 bits
  - 19.31 bits
  - 20.95 bits

$$W = \frac{N}{33.6} + 14.3 \quad (N \geq 32) \quad (1)$$
## II. Failure Manipulator

Matrix Sizes

<table>
<thead>
<tr>
<th>Matrices Sizes</th>
<th>32×32</th>
<th>64×64</th>
<th>128×128</th>
<th>256×256</th>
<th>512×512</th>
</tr>
</thead>
<tbody>
<tr>
<td>95% CNs</td>
<td>5273</td>
<td>16590</td>
<td>60520</td>
<td>179000</td>
<td>475300</td>
</tr>
</tbody>
</table>

\[ Y = 197X \quad (Y: \text{ECN(Est. Condition Numbers)}/(2^X), \ X: \log_2(\text{matrix order})) \]

\[ CN = 197N \log_2(N), \ (N: \text{Matrix order}) \quad (2) \]

\[ CN = 197(33.6(W-14.3))\log_2(33.6(W-14.3)) \quad (3) \]
Provide appropriate initial precision and failure manipulator ➞ Optimize performance according to system attributes
Conclusions

1. Customized reconfigurable computing architecture for mixed precision linear system solver according to system matrix condition numbers.

2. First effort to find out the initial precision predictor for GEPP and failure manipulator to provide newly assigned appropriate precision according to the system matrix condition number.

3. High performance for linear solver along with reconfigurable computing multiple precision computation paradigm.

Thank you, Any Questions?