ADAPTIVE SCALABLE SVD UNIT FOR FAST PROCESSING OF LARGE LSE PROBLEMS

IÑAKI BILDOSOLA & UNAI MARTINEZ-CORRAL KOLDO BASTERRETXEA
MOTIVATION

• Previous Project
  - Computational Intelligence applications
  - Real Time Computation
  - LSE problems
• Resulting Matrices:
  - Large-scale
  - Rank-deficient
  - Ill-conditioned matrices
• Implementation in MicroBlaze → Too Much Delay
• Need for acceleration → parallel processing & optimized faster implementation → FPGA
SELECTING THE ALGORITHM WHY SVD?

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

• We needed an algorithm numerically robust
• Struggling with deficient matrices
• Struggling with non-square matrices
• Avoid the Inverse calculation
• Obtain the Pseudoinverse
• Good Base for Problem Reduction (future Work)
SELECTING SVD METHOD
WHY ONE-SIDED JACOBI?

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

• Easily Parallelizable $\Rightarrow$ Jacobi
• What more?
  • Purely non-conflicting $\Rightarrow$ one-sided
  • Optimizing the managed unit size $\Rightarrow$ one-sided
• Main features
  • Based on Column Pairs Orthogonalization
  • Given’s rotations by Rutishauser formulas

\[
\tan(2\theta_{ij}^k) = \frac{2 \cdot (A_{ij}^k \cdot A_{ij}^k)}{||A_{ij}^k||^2 - ||A_{ij}^i||^2} \quad AV = W \quad A = USV^T
\]
• Computing Precisions $\rightarrow$ Sets The Maximum
• Matrix Conditioning $\rightarrow$ Impacts on the Accuracy
  • $K(A) = 10^k$ ; $CP = 10^m$ ; Solution $= 10^{m-k}$.
  • Matrix Size $\rightarrow$ Possible Accumulated error.
  • $K(A)$ & Matrix Size Impact close to $CP$ $\rightarrow$ No Solution or very degraded
SPEAKING ABOUT THE ACCURACY: IMPOSED CONDITIONS

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

- Randomly generated matrices \( f(\text{Size}, k(A)) \)
- Errors: Our Algorithm (’) in Single Vs Matlab(”’) in double
  - Singulars = Maximum Normalized Error = \( \frac{\sigma' - \sigma''}{\sigma''} \)
  - Inverse = \( \| A_{inv}' - A_{inv}'' \|^2 \)
  - Remainder = \( \| A - SVD \|^2 \)
SPEAKING ABOUT THE ACCURACY DECIDED CONDITIONS

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

- **Threshold Value**
  - Iterative Algorithm Finisher → Orthogonalization
  - User Defined Parameter → Time & Accuracy Trade-off
  - Error Saturation Phenomenon → (Imposed Conditions)
IMPROVING PREVIOUS WORK LEARNING FROM OTHERS

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

• **Brent and Luck**
  - Highlighted the column norm importance
  - Normalized the Threshold $\rightarrow$ Adapting to the columns’ norm $\rightarrow$ Actually calculating the cosine: $\frac{A_i \cdot A_j^T}{||A_i|| ||A_j||} < \text{Threshold}$

• **Hestenes**
  - Swap the columns $\rightarrow$ Active Sorting $\rightarrow f(\text{column norm})$
- Motivation
  - Selecting the Algorithm
  - Selecting SVD Method
  - Speaking About the Accuracy
  - Improving Previous Work
  - Results
  - HW FPGA Implementation
  - Why and How Scalable?
  - Conclusion
  - Future Work

**IMPROVING PREVIOUS WORK ADDING OUR TOUCH**

- Increased Adaptability
  - Realizing that the “Inverse Error” lies on small columns
  - Being Fussier with them → Harder Threshold
  - With Easier Threshold → Same Solution Accuracy
  - Not rotating in vain the big columns
  - AMN:
    \[
    \frac{A_i \cdot A_j^T}{\|A_i\| \cdot \|A_j\|} = \cos(A_i, A_j) < \text{Threshold} \cdot \min(\|A_i\|, \|A_j\|)
    \]
  - AAMN:
    \[
    \frac{A_i \cdot A_j^T}{\|A_i\| \cdot \|A_j\|} = \cos(A_i, A_j) < \text{Threshold} \cdot \|A_j\|
    \]
Initially Two Angles Calculation:
- The Decision → f(cosine)
- The Rotation → f(Rutishauser)
- Cos & Rutishauser → both f(columns and its norms)

Killing two bird with one stone
Decision and rotation → f(Rutishauser)

Readaptation → More sensitive

AARH: \( \theta_{ij} < \text{Threshold} \cdot \|A_j\|^2 \)

Avoiding root squares
RESULTS
THE MODIFIED ONE-SIDED JACOBI

Initialization: Problem Size, Flag & Counters

Iteration: Swap & Null Columns Management

Iteration: Decision, Rotation & Actualization

Finish: Matrix Factorization

\[ \sigma_i = \|W(:,i)\| \quad U(:,i) = \frac{W(:,i)}{\sigma_i} \]
COMPARATION ANALYSIS TOOLBOX

Column Arrangement
Selection of the Comparation

Error Evolution
Results
RESULTS

COMPARING WITH THE REST

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results

- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

Obtaining Same Accuracy → Easier Threshold
Obtaining Same Accuracy → Less Rotations
Obtaining a Better Result → The Higher the K(A)

Testing With Real Matrices & Obtaining Expected Results

### TABLE I: Comparative of algorithm performance

<table>
<thead>
<tr>
<th></th>
<th>Inverse Error</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>4.19E-6</td>
<td>2.11E-5</td>
</tr>
<tr>
<td>B&amp;L</td>
<td>3.98E-6</td>
<td>1.93E-5</td>
</tr>
<tr>
<td>ABL</td>
<td>3.65E-6</td>
<td>1.83E-5</td>
</tr>
<tr>
<td>AAMN</td>
<td>5.60E-6</td>
<td>1.76E-5</td>
</tr>
<tr>
<td>AARH</td>
<td>6.77E-6</td>
<td>1.87E-5</td>
</tr>
<tr>
<td>K(A)</td>
<td>1.00E+01</td>
<td>1.00E+02</td>
</tr>
<tr>
<td>Rotations</td>
<td>33,606</td>
<td>33,530</td>
</tr>
<tr>
<td></td>
<td>33,859</td>
<td>10.50</td>
</tr>
<tr>
<td>B&amp;L</td>
<td>32,303</td>
<td>9.70</td>
</tr>
<tr>
<td>ABL</td>
<td>26,096</td>
<td>8.20</td>
</tr>
<tr>
<td>AAMN</td>
<td>25,275</td>
<td>8.18</td>
</tr>
<tr>
<td>AARH</td>
<td>25,312</td>
<td>8.40</td>
</tr>
</tbody>
</table>

### TABLE II: TEST MATRICES RESULTS

<table>
<thead>
<tr>
<th>Size</th>
<th>Name</th>
<th>k</th>
<th>JGD_Kocay/Trec9</th>
<th>JGD_Forest/TF11</th>
<th>Bai/bwm200</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABL</td>
<td>1.99E6</td>
<td>5,004</td>
<td>7</td>
<td>1.97E3</td>
</tr>
<tr>
<td></td>
<td>AAMN</td>
<td>1.96E6</td>
<td>4,903</td>
<td>8</td>
<td>5.3E4</td>
</tr>
<tr>
<td></td>
<td>AARH</td>
<td>2.21E6</td>
<td>4,908</td>
<td>9</td>
<td>8.03E4</td>
</tr>
</tbody>
</table>

- Testing With Real Matrices & Obtaining Expected Results
- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

RESULTS
COMPARING WITH THE REST

• Savings in Number of Rotations

<table>
<thead>
<tr>
<th>CN</th>
<th>Fixed to AARH</th>
<th>B&amp;L to AARH</th>
<th>ABL to AARH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,00E+01</td>
<td>24.29%</td>
<td>21.02%</td>
<td>2.24%</td>
</tr>
<tr>
<td>1,00E+02</td>
<td>30.38%</td>
<td>24.48%</td>
<td>4.15%</td>
</tr>
<tr>
<td>1,00E+03</td>
<td>44.08%</td>
<td>36.50%</td>
<td>18.00%</td>
</tr>
<tr>
<td>1,00E+04</td>
<td>52.72%</td>
<td>39.29%</td>
<td>19.32%</td>
</tr>
</tbody>
</table>
Double Data-Flow:

- Primary: Linear array to manage Ai/Aj
- Secondary: Asynchronous full-duplex shared bus to manage Vi/Vj
- FIFO between PUs
HW FPGA IMPLEMENTATION
PROCESSING UNIT

PU Design:
- Evaluation: Computing square Euclidean norms and vector multiplication, swapping and deciding
- Cordic: Theta calculation and rotations’ performing
- Cache: L(m+n) and R(max(m,2n))
WHY AND HOW SCALABLE?

- Motivation
- Selecting the Algorithm
- Selecting SVD Method
- Speaking About the Accuracy
- Improving Previous Work
- Results
- HW FPGA Implementation
- Why and How Scalable?
- Conclusion
- Future Work

**No limited to specific matrices and HW:**

- **Generic Solution**
  - Different sizes
  - Different Shapes
  - Different budgets

**Architecture**

- Based on basics processing units PUs
- PUs variable on quantity
- From 2 to \( \frac{n}{2} \)
# IMPLEMENTATION RESULTS

<table>
<thead>
<tr>
<th></th>
<th>xc6slx45-3fgg484</th>
<th>xc7k160t-3fbg484</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area</strong></td>
<td>56 %</td>
<td>86 %</td>
</tr>
<tr>
<td><strong>DSPs</strong></td>
<td>93 %</td>
<td>60 %</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>62 %</td>
<td>44 %</td>
</tr>
<tr>
<td><strong>Matrix Size ; K(A)</strong></td>
<td>300x100 ; 10^2</td>
<td>750 x 250</td>
</tr>
<tr>
<td><strong>PUs</strong></td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>55 MHz</td>
<td>90 MHz</td>
</tr>
<tr>
<td><strong>Processing Time</strong></td>
<td>60 ms (5-6 sweeps, 8-16 ms/sweep)</td>
<td></td>
</tr>
</tbody>
</table>

Word-Length :18 bits
CONCLUSION

- AAMN and AARH proposed outperforming previous proposals.
  - Small Columns Important Columns
  - Same Accuracy Less Rotations
  - User-defined Accuracy -> Threshold
  - HW Friendly
- An implemented parallel processing scheme proposed:
  - Linear Array of PUs
  - Scalable
  - Double Data-Flow
FUTURE WORK

• Online reduction of problem size
• Improve sorting
• Optimize PU design
  • Improved CORDIC realization (Redundant arithmetic (Ercegovac) or Square root and division free (Gotze))
  • Ad-hoc online estimators
    • Atan
    • Square norm
THANK YOU VERY MUCH

• Questions or Details?