FASTCF: FPGA-based Accelerator for Stochastic-gradient-descent-based Collaborative Filtering

Shijie Zhou, Rajgopal Kannan, Min Yu, Viktor K. Prasanna

FPGA 2018
Outline

• Introduction

• Optimizations and Accelerator Design

• Experimental Results

• Impact

• Conclusion
Recommender Systems

• Information filtering system to predict the interest of users

• Everywhere
  – Amazon
  – Netflix
  – YouTube
  – eBay
  – Facebook
Collaborative Filtering

• Apply machine learning algorithms to predict user’s rating for unrated items
  – Bayesian networks
  – Clustering
  – Matrix factorization

• **Stochastic gradient descent (SGD)**
  – Achieves high prediction accuracy
  – **Training** is computation-intensive
Problem Definition (1)

- **Input** is a partially filled rating matrix $R$
  - # of users = # of rows = $|U|$
  - # of items = # of columns = $|V|$
Problem Definition (2)

- **Output** contains two low-rank (i.e., $H$) matrices $P$ and $Q$
  - $P \times Q^T \approx R$
  - $p_i$ : the $i^{th}$ row of $P$ (latent feature vector of user $u_i$)
  - $q_j$ : the $j^{th}$ row of $Q$ (latent feature vector of item $v_j$)
  - $r_{ij}$: the rating of $v_j$ given by $u_i$
  - $r_{ij} \approx p_i \cdot q_j$
Problem Definition (3)

- **Training** process aims to find $P$ and $Q$ that **minimizes** overall squared prediction error

- SGD-based approach

Randomly initialize each $p_i$ and $q_j$

While not done do

For each known rating $r_{ij}$ do

$$err_{ij} = r_{ij} - p_i \cdot q_j$$

Update $p_i$ and $q_j$ based on $err_{ij}$

End for

End while

Return $P$ and $Q$
Challenges in Acceleration (1)

Challenge 1

- Feature vectors do not fit in on-chip memory of FPGA
- How to achieve efficient data reuse?

![Diagram showing External Memory and FPGA with On-chip RAM]

External Memory

\[ P \] \hspace{1cm} \[ Q \]

FPGA

On-chip RAM
Challenges in Acceleration (2)

Challenge 2

- Feature vector dependencies
  - Need to **incrementally** update feature vectors **once per rating**
Challenges in Acceleration (3)

Challenge 3

• Parallel processing units access on-chip RAMs
  – Concurrent accesses to the same RAM \( \rightarrow \) conflicts

• How to reduce access conflicts?
Contributions

• Novel optimizations to address the three challenges
  – Partitioning and communication hiding
    • Completely overlap communication with computation
  – Parallelism extraction to reduce data dependencies by $28x - 60x$
  – Scheduling to reduce bank conflicts by $2x - 4x$

• Sustain up to $217$ GFLOPS for training large real-life datasets

• Achieve $13.3x$ and $12.7x$ speedup compared with highly optimized multicore and GPU implementations

• Generic technique to accelerate SGD-based training
Outline

• Introduction

• Optimizations and Accelerator Design

• Experimental Results

• Impact

• Conclusion
Bipartite Graph Representation

\[
\begin{array}{|c|c|}
\hline
1 & 4 \\
2 & 5 \\
3.5 & 6 \\
\hline
\end{array}
\]

\( R \)

\( G \)

User vertices

\( u_0 \)

\( u_1 \)

\( u_2 \)

\( u_3 \)

Items vertices

\( v_0 \)

\( v_1 \)

1

2

5

3.5

4

6

1

2

5

3.5

4

6
Optimization 1 (1)

Graph Partitioning

Subgraph_0

Subgraph_1
Optimization 1 (2)
Communication Hiding

If \( \text{Density}(\text{Subgraph}_z) \geq \frac{\text{Proc}_\text{throughput} \times \text{FV}_\text{width}}{\text{Memory}_\text{bandwidth}} \) \( \forall z \),
the communication can be **completely overlapped** with the computation.
Optimization 1 (3)

Graph Partitioning and Communication Hiding

• Objective: each subgraph should have sufficient edges for the computation to hide the communication

• Fast **heuristic** partitioning approach
  – Subset degree of a vertex subset
    • number of edges that connect to the vertices in the subset
  – Balance subset degrees of distinct vertex subsets
    • Sort vertices based on vertex degree
    • Greedy assignment of each vertex into the non-full vertex set that has the minimum subset degree
Optimization 2

Parallelism Extraction

• Partition the edges of each subgraph into non-overlapping matchings
  – Edges in the same matching have **no common** vertices

![Diagram](https://via.placeholder.com/150)

Edge coloring
Optimization 3

Edge Scheduling

- Partition the edges in each matching into batches
  - Sort the edges based on the conflict index (i.e., the number of edges in the matching that have conflict with this edge)
  - Assign each edge to the batch where its addition does not increase the conflicts within the batch
Architecture of FASTCF

- Multi-ported RAM based on banking
- Parallel processing units to process distinct edges
- Resolve bank conflicts to feature vector buffer
- Check FV dependencies based on fine-grained locking
Outline

• Introduction

• Optimizations and Accelerator Design

• Experimental Results

• Impact

• Conclusion
Experimental Setup

- **Real-life datasets**

<table>
<thead>
<tr>
<th>Dataset</th>
<th># users</th>
<th># items</th>
<th># ratings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libim</td>
<td>135 K</td>
<td>168 K</td>
<td>17,359 K</td>
<td>Dating ratings</td>
</tr>
<tr>
<td>Netflix</td>
<td>480 K</td>
<td>17 K</td>
<td>100,480 K</td>
<td>Movie ratings</td>
</tr>
<tr>
<td>Yahoo</td>
<td>1,200 K</td>
<td>136 K</td>
<td>460,380 K</td>
<td>Music ratings</td>
</tr>
</tbody>
</table>

- **Virtex UltraScale+ xcvu9pflgb2104 FPGA (for training)**
  - 43 MB of on-chip RAM

- **Intel Xeon E5-2686 processor (for pre-processing)**
  - 8 cores @ 2.3 GHz
Pre-processing Overhead

\[ T_{\text{preprocess}} = T_{\text{Opt 1}} + T_{\text{Opt 2}} + T_{\text{Opt 3}} \]

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( T_{\text{Opt 1}} )</th>
<th>( T_{\text{Opt 2}} )</th>
<th>( T_{\text{Opt 3}} )</th>
<th>( T_{\text{preprocess}} )</th>
<th>( \frac{T_{\text{preprocess}}}{T_{\text{train}}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libim</td>
<td>0.4 sec</td>
<td>4.4 sec</td>
<td>2.7 sec</td>
<td>7.5 sec</td>
<td>2.1%</td>
</tr>
<tr>
<td>Netflix</td>
<td>1.0 sec</td>
<td>10.7 sec</td>
<td>7.0 sec</td>
<td>18.7 sec</td>
<td>2.1%</td>
</tr>
<tr>
<td>Yahoo</td>
<td>5.5 sec</td>
<td>42.3 sec</td>
<td>23.0 sec</td>
<td>70.8 sec</td>
<td>2.8%</td>
</tr>
</tbody>
</table>

Pre-processing overhead can be amortized since the training is iterative
Performance vs. Parallelism

Throughput (GFLOPS)

Number of parallel processing units ($M$)

Libim  Netflix  Yahoo

1: 47
2: 93
4: 163
8: 217
# Impact of Optimization 1

## Communication Hiding

- Optimized design: Opt 1 + Opt 2 + Opt 3
- Baseline design: Opt 2 + Opt 3

## Table

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$T_{\text{exec}}$ per iteration (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libim</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>Netflix</td>
<td>0.15</td>
<td>0.16</td>
</tr>
<tr>
<td>Yahoo</td>
<td>0.68</td>
<td>0.76</td>
</tr>
</tbody>
</table>
## Impact of Optimization 2

**Pipeline stall reduction**

- Optimized design: Opt 1 + Opt 2 + Opt 3
- Baseline design: Opt 1 + Opt 3

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Pipeline stalls due to dependencies</th>
<th>Reduction</th>
<th>$T_{exec}$ per iteration (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libim</td>
<td>2,005 K</td>
<td>57,524 K</td>
<td><strong>28.7x</strong></td>
<td>0.03</td>
</tr>
<tr>
<td>Netflix</td>
<td>6,151 K</td>
<td>314,884 K</td>
<td><strong>51.2x</strong></td>
<td>0.15</td>
</tr>
<tr>
<td>Yahoo</td>
<td>24,954 K</td>
<td>1,500,295 K</td>
<td><strong>60.1x</strong></td>
<td>0.68</td>
</tr>
</tbody>
</table>
Impact of Optimization 3

Bank conflict reduction

- Optimized design: Opt 1 + Opt 2 + Opt 3
- Baseline design: Opt 1 + Opt 2

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Bank conflicts</th>
<th>Reduction</th>
<th>$T_{\text{exec}}$ per iteration (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libim</td>
<td>1,165 K</td>
<td>2,798 K</td>
<td>2.4x</td>
<td>0.03</td>
</tr>
<tr>
<td>Netflix</td>
<td>3,960 K</td>
<td>16,686 K</td>
<td>4.2x</td>
<td>0.15</td>
</tr>
<tr>
<td>Yahoo</td>
<td>19,393 K</td>
<td>75,524 K</td>
<td>3.9x</td>
<td>0.68</td>
</tr>
</tbody>
</table>
Comparison with State-of-the-art

Up to 13.3x speedup for training the Netflix dataset

<table>
<thead>
<tr>
<th>Approach</th>
<th>Platform</th>
<th>Processor Power</th>
<th>Exec. Time per Iteration</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGMOD ’14</td>
<td>24-core Intel E5-2697</td>
<td>130 W</td>
<td>2.00 sec</td>
<td>13.3x</td>
</tr>
<tr>
<td>This paper</td>
<td>Virtex UltraScale+</td>
<td>14 W</td>
<td>0.15 sec</td>
<td></td>
</tr>
<tr>
<td>GPGPU ’15</td>
<td>2880-core Tesla K40C GPU</td>
<td>235 W</td>
<td>1.90 sec</td>
<td>12.7x</td>
</tr>
<tr>
<td>This paper</td>
<td>Virtex UltraScale+</td>
<td>14 W</td>
<td>0.15 sec</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Introduction

• Optimizations and Accelerator Design

• Experimental Results

• Impact

• Conclusion
Impact of This Work

• Accelerate other data science problems that derive latent features from observations
  – Topic modeling/extraction
  – Word embedding

• Techniques for accelerating SGD-based training algorithms

• Generic optimizations applicable to other platforms
Conclusion

• Developed FASTCF to accelerate SGD-based CF

• Designed three optimizations to
  – completely overlap communication with computation
  – reduce data dependencies to extract parallelism
  – reduce bank conflicts

• Achieved a high throughput of up to 217 GFLOPS for training large real-life datasets

• Achieved 13.3x and 12.7x speedup compared with state-of-art multicore and GPU implementations
Comments & Questions

Thank you

fpga.usc.edu