A reduced-precision streaming SpMV architecture for Personalized PageRank on FPGA

Alberto Parravicini (alberto.parravicini@polimi.it)
Francesco Sgherzi
Prof. Marco Santambrogio

ASP-DAC 2021, January 20th
Approximating Personalized PageRank (PPR) on FPGA

PPR is a variation of the famous PageRank (PR) algorithm
- PR is a *graph ranking* algorithm
- Find the *most important* vertices in a graph
- E.g. web-sites, cities, etc.
Approximating Personalized PageRank (PPR) on FPGA

PPR is a variation of the famous PageRank (PR) algorithm

- PR is a graph ranking algorithm
- Find the most important vertices in a graph
- E.g. web-sites, cities, etc.

PPR is a building block of recommender systems in e-commerce and social networks

- For an input node, find the most similar nodes
- E.g. find the most relevant products/communities
- We need real-time results, with high power efficiency
Approximating Personalized PageRank (PPR) on FPGA

The goal in a recommender system is to provide a good set of recommendations

No need for 100% numerical accuracy
- The ranking is what matters!
- Reduced-precision fixed-point: better performance at no accuracy cost
- FPGAs provides high-performance and power efficiency
Our contributions

- Novel FPGA design for PPR that exploits streaming COO SpMV and reduced-precision arithmetic
- We provide 6.8x better performance and 42x better power efficiency vs. a SoA CPU implementation, on community and co-purchasing graphs
- We characterize how reduced-precision arithmetic leads to better performance, negligible accuracy loss, and 2x faster convergence
Related work

- On CPUs and GPUs, the focus has mostly been on Domain Specific Languages, and web-scale graphs
  - For CPUs, Green-Marl [1], GraphIt [2], Zhou et al. [3], Zhu et al. [4]
  - On GPUs, nvGRAPH [5] and GraphBLAST [6]
- On FPGA, no previous work on PPR
  - There is work on SpMV (Grigoras [7], Umuroglu [8], Shan [9], etc.)
  - Reduced-precision arithmetic unexplored for graph ranking
  - We focus on community graphs, not web-scale graphs

[7] Paul Grigoras et al. 2015. Accelerating SpMV on FPGAs by compressing nonzero values
[9] Yi Shan et al. 2010. FPGA and GPU implementation of large scale SpMV.
Problem Definition

Graphs can be seen as sparse matrices

- E.g. COO format, list of edges/non-zero matrix entries
Problem Definition

Graphs can be seen as sparse matrices

- E.g. COO format, list of edges/non-zero matrix entries
- The PPR equation becomes a repeated SpMV (Sparse matrix-vector multiplication)

\[ p_{t+1} = \alpha Xp_t + \frac{\alpha}{|V|}(\tilde{d}p_t)1 + (1 - \alpha)\tilde{v} \]

- The SpMV is the biggest bottleneck
High-Level Architecture (1/4)

The heart of our implementation is a data-flow reduced-precision SpMV core

- Implemented on a Xilinx Alveo U200 Accelerator Card
The heart of our implementation is a data-flow reduced-precision SpMV core

- Implemented on a Xilinx Alveo U200 Accelerator Card
- Use DRAM for burst sequential reads, at peak bandwidth
The heart of our implementation is a data-flow reduced-precision SpMV core

- Implemented on a Xilinx Alveo U200 Accelerator Card
- Use DRAM for burst sequential reads, at peak bandwidth
- Use UltraRAM for fast random accesses
The heart of our implementation is a data-flow reduced-precision SpMV core

- We batch $K=8$ PPR computations at once
- Increase operational intensity by 8x
- Move bottleneck from memory access to computation
Algorithm 1 Personalized PageRank

1: function PPR(\text{coo\_graph}, \tilde{V}, \tilde{d}, \alpha, \text{max\_iter})
2: Initialize local buffers to 0
3: for \(k \leftarrow 0, \kappa\) do \(
\quad \text{Set PR=1 on pers. vertices}
\)
4: \(P_1[k] = \tilde{V}[k]\)
5: for \(i \leftarrow 0, \text{max\_iter}\) do \(
\quad \text{i.e. } \frac{\alpha}{|V|}P_1\tilde{d}\)
6: \(\text{scaling\_vec} \leftarrow \text{scaling}(P_1, \tilde{d})\)
7: \(\text{SpMV}(\text{coo\_graph}, P_1, P_2)\) \(
\quad \text{Xp}_i \text{ in eq. (1)}\)
8: \(P_1 = \alpha P_2 + \text{scaling\_vec} + (1 - \alpha)\tilde{V}\)
9: Write \(P_1\) to output

Algorithm 2 COO SpMV

1: function SpMV(\text{coo\_graph}, P_t, P_{t+1})
2: for \(i \leftarrow 0, \lfloor |E|/B \rfloor\) do
3: \(\quad \text{1. Process COO in packets of size } B\)
4: \(\quad x \leftarrow \text{coo\_graph}.x[i]; y \leftarrow \text{coo\_graph}.y[i]\)
5: \(\quad \text{val} \leftarrow \text{coo\_graph}.val[i]\)
6: \(\quad \text{for } k \leftarrow 0, \kappa \text{ do} \quad \kappa \text{ personalization vertices}\)
7: \(\quad \quad \text{2. Update edge-wise PPR values}\)
8: \(\quad \quad \text{for } j \leftarrow 0, \ldots, B\) do
9: \(\quad \quad \text{\(dp\_buffer[k, j] = \text{val}[j] \cdot P_t[k, y[j]]\)}\)
10: \(\quad \quad \text{3. Aggregate partial PPR values}\)
11: \(\quad \quad \text{for } b1 \leftarrow 0, \ldots, B\) do
12: \(\quad \quad \quad \text{for } b2 \leftarrow 0, \ldots, B\) do
13: \(\quad \quad \quad \quad \text{agg\_res}[k, x[0] \% B + b1] += \quad \text{\(dp\_buffer[k, b2] \cdot ((x[0] + b1) \equiv x[b2])\)}\)
14: \(\quad \quad \quad \text{4. Store PPR values on each vertex}\)
15: \(\quad \quad \quad x_{s} \leftarrow [x[0]/B] \cdot B\)
16: \(\quad \quad \quad \text{if } x_{s} = x_{s, \text{old}} \text{ then}\)
17: \(\quad \quad \quad \quad \text{for } j \leftarrow 0, \ldots, B\) do
18: \(\quad \quad \quad \quad \quad \text{res}_1[k, j] += \text{agg\_res}[k, j]\)
19: \(\quad \quad \quad \quad \quad \text{res}_2[k, j] += \text{agg\_res}[k, j + B]\)
20: \(\quad \quad \quad \quad \text{else}\)
21: \(\quad \quad \quad \quad \quad \text{for } j \leftarrow 0, \ldots, B\) do
22: \(\quad \quad \quad \quad \quad \text{res}[k, j + x_{s, \text{old}}] = \text{res}_1[k, j]\)
23: \(\quad \quad \quad \quad \quad \text{res}_1[k, j] = \text{res}_2[k, j] + \text{agg\_res}[k, j]\)
24: \(\quad \quad \quad \quad \quad \text{res}_2[k, j] = \text{agg\_res}[k, j + B]\)
25: \(\quad \quad \quad \quad \quad \text{reset(agg\_res); } x_{s, \text{old}} \leftarrow x_{s}\)
26: \)
Experimental Setup

We compare against a *multi-threaded SoA CPU* (PGX 19.3.1) and a *floating-point FPGA* implementations

Test with 8 different real and synthetic graphs

- Between **500K and 2M edges**
- Size similar to real community and co-purchasing graphs

Different fixed-point sizes, from 20 to 26 bits

- Lower bit-width gives better clock speed and lower resource usage

**CPU:** 2x Intel Xeon E5-2680 v2, 384GB RAM
**FPGA:** Xilinx Alveo U200
Summary of graphs and designs

Graphs in the evaluation

| Graph Distribution               | |V| | |E| | Sparsity |
|----------------------------------|---|---|---|---|
| $G_{n,p}$ (Erdős-Renyi)          | $10^5$ | 1002178 | $10^{-4}$ |
|                                  | $2 \cdot 10^5$ | 1999249 | $4.9 \cdot 10^{-5}$ |
| Watts–Strogatz small-world       | $10^5$ | 1000000 | $10^{-4}$ |
|                                  | $2 \cdot 10^5$ | 2000000 | $5 \cdot 10^{-5}$ |
| Holme and Kim powerlaw           | $10^5$ | 999845 | $0.99 \cdot 10^{-4}$ |
|                                  | $2 \cdot 10^5$ | 1999825 | $4.9 \cdot 10^{-5}$ |
| Amazon co-purchasing network     | 128000 | 443378 | $2.7 \cdot 10^{-5}$ |
| Twitter social circles           | 81306 | 1572670 | $2.3 \cdot 10^{-4}$ |

FPGA designs

<table>
<thead>
<tr>
<th>Bit-width</th>
<th>BRAM</th>
<th>DSP</th>
<th>FF</th>
<th>LUT</th>
<th>URAM</th>
<th>Clock (MHz)</th>
<th>Power Cons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bits</td>
<td>14%</td>
<td>3%</td>
<td>4%</td>
<td>26%</td>
<td>20%</td>
<td>220</td>
<td>34 W</td>
</tr>
<tr>
<td>26 bits</td>
<td>14%</td>
<td>3%</td>
<td>4%</td>
<td>38%</td>
<td>20%</td>
<td>200</td>
<td>35 W</td>
</tr>
<tr>
<td>32 bits, float</td>
<td>14%</td>
<td>48%</td>
<td>35%</td>
<td>89%</td>
<td>26%</td>
<td>115</td>
<td>40 W</td>
</tr>
<tr>
<td>Available</td>
<td>4320</td>
<td>6840</td>
<td>2364480</td>
<td>1182240</td>
<td>960</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results - Speedup

Up to 6x speedup w.r.t. CPU and floating-point FPGA
- Time to compute 100 personalization vertices
- 42x higher power efficiency w.r.t. CPU and up to 6x w.r.t. floating point FPGA
Accuracy of reduced-precision

Accuracy of results tested with many metrics
- Num. of errors, NDCG, Precision, Kendall's $\tau$, ...
- **26 bits** provide no accuracy loss

![Graphs showing NDCG, Precision, and Kendall's $\tau$ for different fixed-point bitwidths (20, 22, 24, 26 bits). The graphs indicate that 26 bits provide no accuracy loss.](image-url)
Convergence speed

- Reduced precision provides $2x$ faster convergence
  - With no accuracy loss
  - We can run PPR for half the iterations
  - In practice, $2x$ additional speedup!
Future works

Improve performance using HBM on the Alveo U280
- 6x peak bandwidth w.r.t DDR4 (460GB/s vs 77GB/s)
- Remove size constraints of URAM

Leverage our SpMV in other applications:
- Eigenvalue computation
- Graph Embedding
- Top-K Similarity
High-Performance FPGA implementation of Approximate PPR

Use **fixed-point data-flow SpMV** for maximum throughput

- Up to **6.8x** faster than CPU and **42x** higher power efficiency
- No accuracy loss and **2x** faster convergence

Thank you!

Alberto Parravicini - alberto.parravicini@polimi.it
Francesco Sgherzi - francesco1.sgherzi@mail.polimi.it
Marco Santambrogio - marco.santambrogio@polimi.it

ASP-DAC 2021, January 20th