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

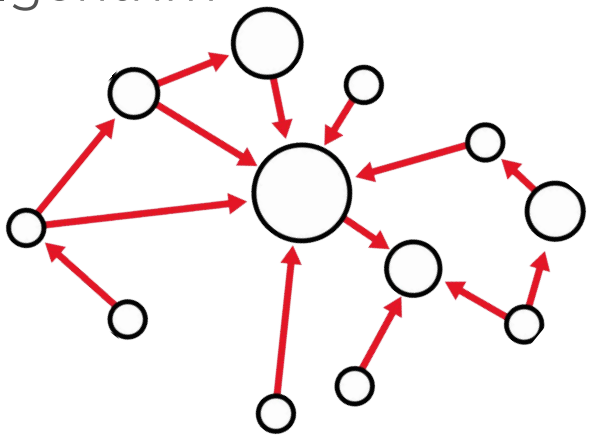
POLITECNICO
MILANO 1863

POLITECNICO MILANO 1863
NECST laboratory

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.

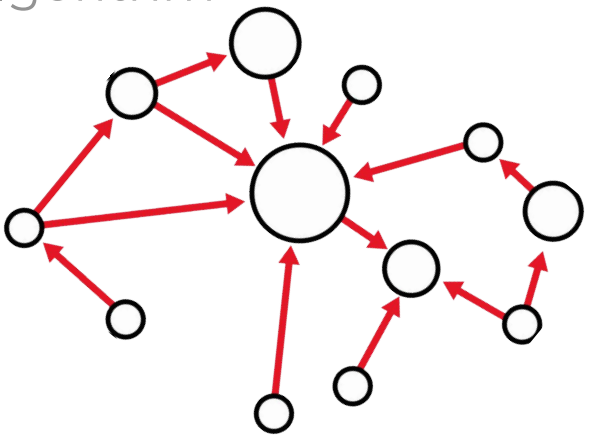


Alberto Parravicini
alberto.parravicini@polimi.it

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

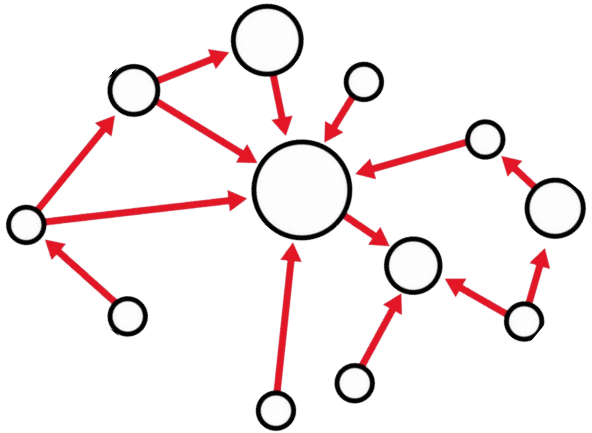
Alberto Parravicini
alberto.parravicini@polimi.it

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

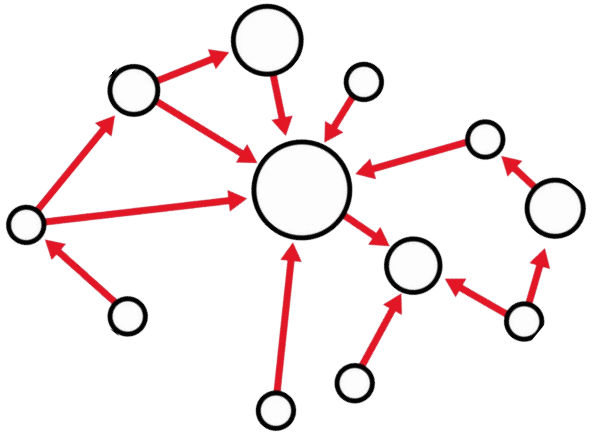
- The ranking is what matters!
- Reduced-precision fixed-point:
  better performance at no accuracy cost
- FPGAs provides high-performance
  and power efficiency



Alberto Parravicini
alberto.parravicini@polimi.it

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



Alberto Parravicini
alberto.parravicini@polimi.it

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

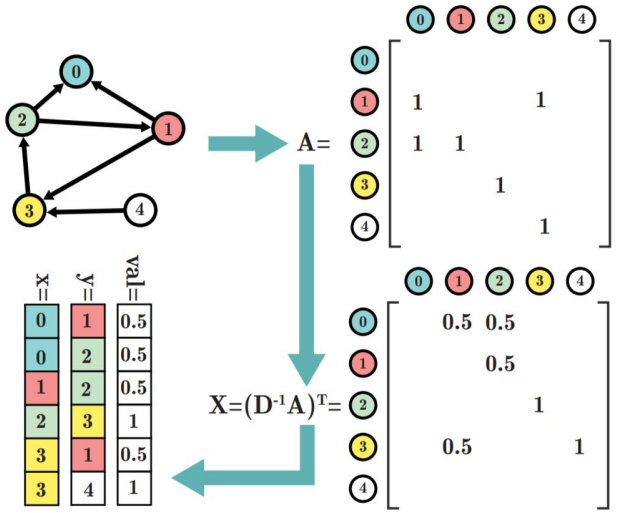
[1] Sungpack Hong et al. 2012. Green-Marl: a DSL for easy and efficient graph analysis
[2] Yunming Zhang et al. 2018. Graphit: A high-performance graph dsl
[3] Shijie Zhou et al. 2017. Design and implementation of parallel pagerank on multicore platforms
[4] Aydın Buluç et al. 2011. The Combinatorial BLAS: Design, implementation, and applications.
[5] ngGRAPH. docs.nvidia.com/cuda/nvgraph/index.html
[6] Carl Yang et al. 2019. GraphBLAST: A high-performance linear algebra-based graph framework on the GPU
[7] Paul Grigoras et al. 2015. Accelerating SpMV on FPGAs by compressing nonzero values
[8] Yaman Umuroglu and Magnus Jahre. 2015. A vector caching scheme for streaming fpga spmv accelerators
[9] Yi Shan et al. 2010. FPGA and GPU implementation of large scale SpMV.

Alberto Parravicini
alberto.parravicini@polimi.it

Problem Definition

Graphs can be seen as sparse matrices

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



Alberto Parravicini
alberto.parravicini@polimi.it

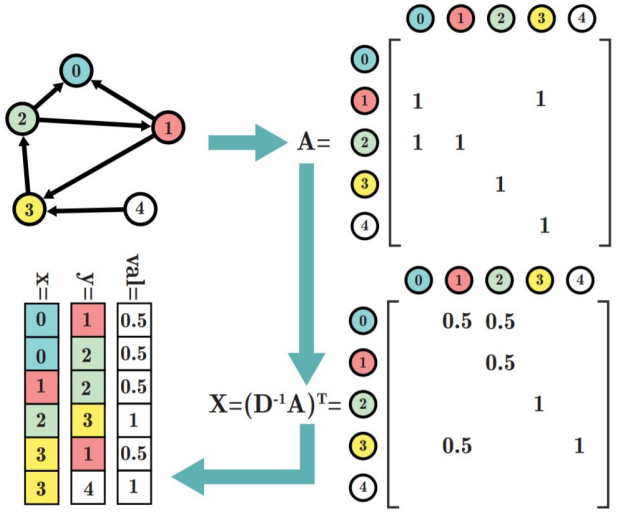
Problem Definition

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)

$$\mathbf{p_{t+1}} = \alpha \mathbf{X} \mathbf{p_t} + \frac{\alpha}{|V|}(\bar{\mathbf{d}}\mathbf{p_t})\mathbf{1} + (1-\alpha)\bar{\mathbf{v}}$$

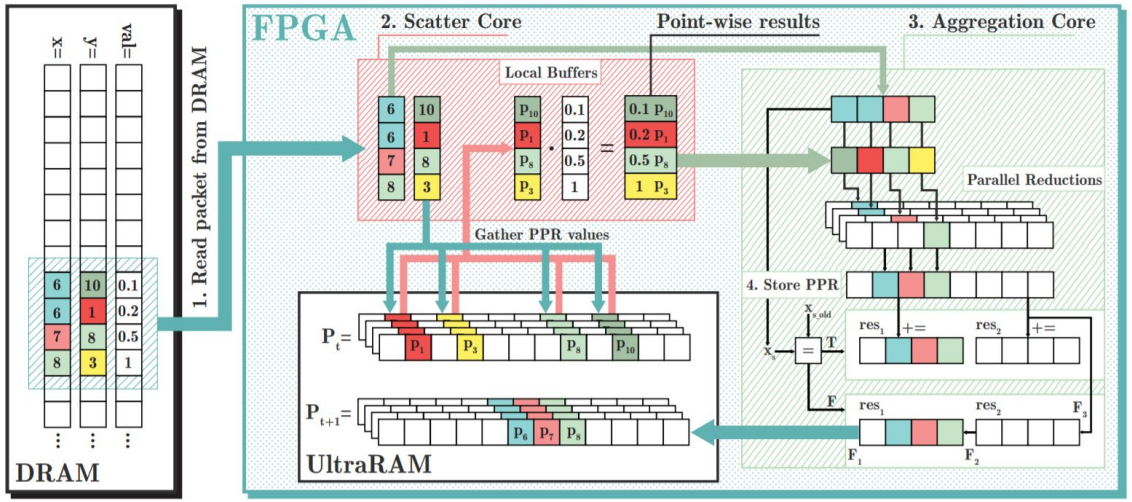
- The SpMV is the biggest bottleneck



Alberto Parravicini
alberto.parravicini@polimi.it

High-Level Architecture (1/4)

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

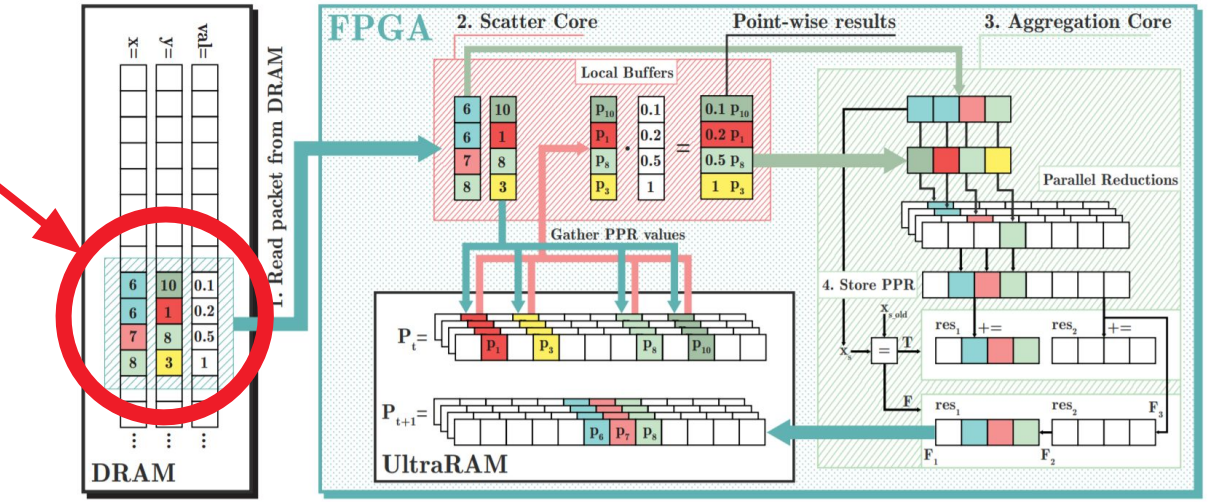


- Implemented on a Xilinx Alveo U200 Accelerator Card

Alberto Parravicini
alberto.parravicini@polimi.it

High-Level Architecture (2/4)

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

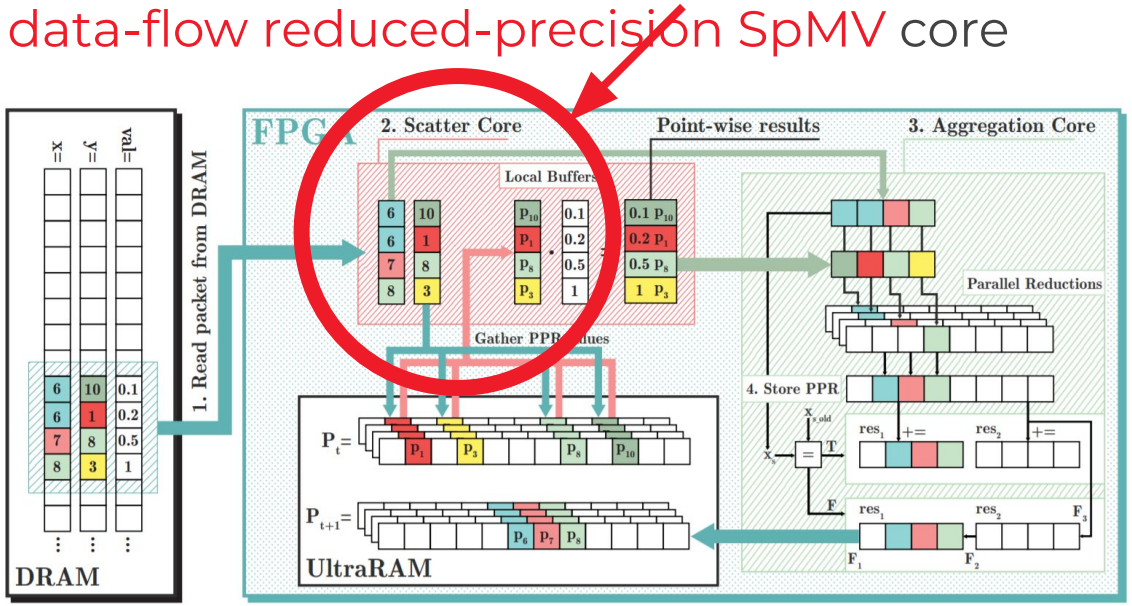


- Implemented on a Xilinx Alveo U200 Accelerator Card
- Use DRAM for burst sequential reads, at peak bandwidth

Alberto Parravicini
alberto.parravicini@polimi.it

High-Level Architecture (3/4)

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

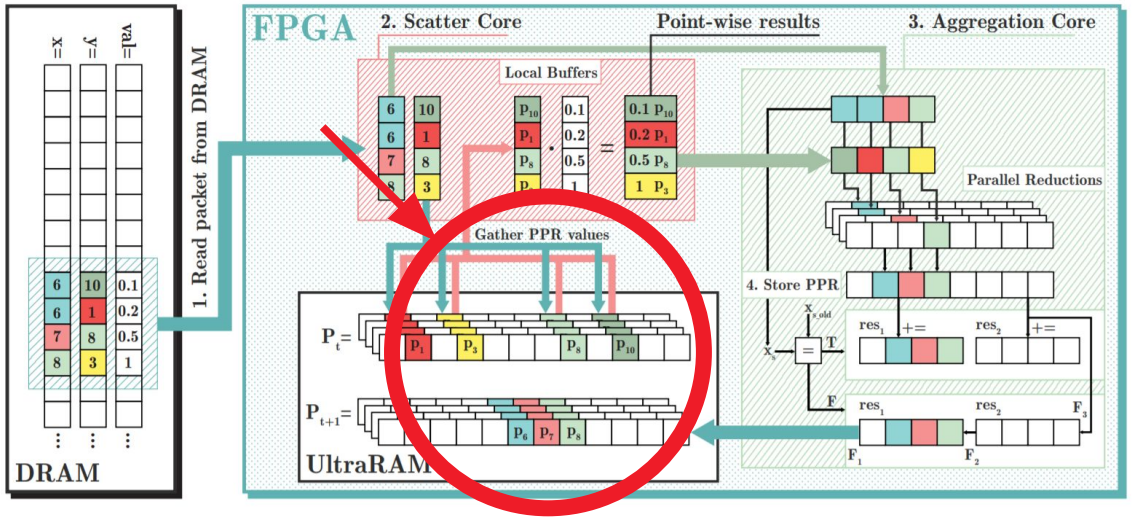


- Implemented on a Xilinx Alveo U200 Accelerator Card
- Use DRAM for burst sequential reads, at peak bandwidth
- Use UltraRAM for fast random accesses

Alberto Parravicini
alberto.parravicini@polimi.it

High-Level Architecture (4/4)

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



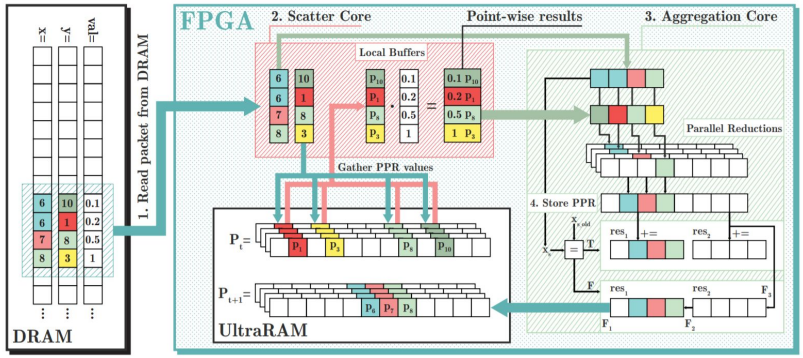
- We batch K=8 PPR computations at once
- Increase operational intensity by 8x
- Move bottleneck from memory access to computation

Alberto Parravicini
alberto.parravicini@polimi.it

Pseudo-code for PPR and SpMV

Algorithm 1 Personalized PageRank

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



Algorithm 2 COO SpMV

1: **function** SpMV($coo\_graph, \mathbf{P_t}, \mathbf{P_{t+1}}$)
2: **for** $i \leftarrow 0..|E|/B$ **do**
3: ▷ **1. Process COO in packets of size** B
4: $x \leftarrow coo\_graph.x[i]; y \leftarrow coo\_graph.y[i]$
5: $val \leftarrow coo\_graph.val[i]$
6:
7: **for** $k \leftarrow 0..\kappa$ **do** ▷ κ *personalization vertices*
8: ▷ **2. Update edge-wise PPR values**
9: **for** $j \leftarrow 0..B$ **do**
10: $dp\_buffer[k, j] = val[j] \cdot \mathbf{P_t}[k, y[j]]$
11: ▷ **3. Aggregate partial PPR values**
12: **for** $b1 \leftarrow 0..B$ **do**
13: **for** $b2 \leftarrow 0..B$ **do**
14: $agg\_res[k, x[0] \,\%\, B + b1] \mathrel{+}= dp\_buffer[k, b2] \cdot ((x[0] + b1) == x[b2])$
15: ▷ **4. Store PPR values on each vertex**
16: $x_s \leftarrow \lfloor x[0]/B \rfloor \cdot B$
17: **if** $x_s == x_{s\_old}$ **then**
18: **for** $j \leftarrow 0..B$ **do**
19: $res_1[k, j] \mathrel{+}= agg\_res[k, j]$
20: $res_2[k, j] \mathrel{+}= agg\_res[k, j + B]$
21: **else**
22: **for** $j \leftarrow 0..B$ **do**
23: $res[k, j + x_{s_{old}}] = res_1[k, j]$
24: $res_1[k, j] = res_2[k, j] + agg\_res[k, j]$
25: $res_2[k, j] = agg\_res[k, j + B]$
26: $reset(agg\_res)$; $x_{s\_old} \leftarrow x_s$

Alberto Parravicini
alberto.parravicini@polimi.it

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

Alberto Parravicini
alberto.parravicini@polimi.it

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

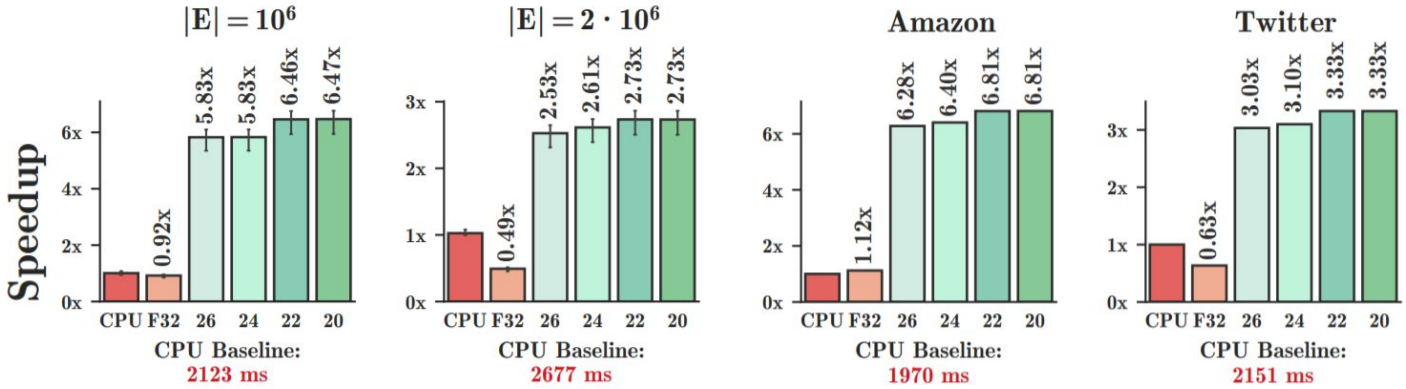
Bit-width	BRAM	DSP	FF	LUT	URAM	Clock (MHz)	Power Cons.
20 bits	14%	3%	4%	26%	20%	220	34 W
26 bits	14%	3%	4%	38%	20%	200	35 W
32 bits, **float**	14%	48%	35%	89%	26%	115	40 W
Available	4320	6840	2364480	1182240	960		

Alberto Parravicini
alberto.parravicini@polimi.it

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

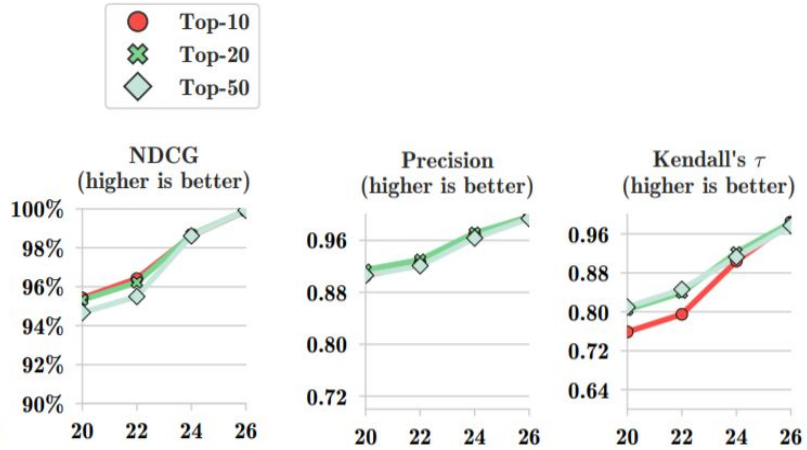


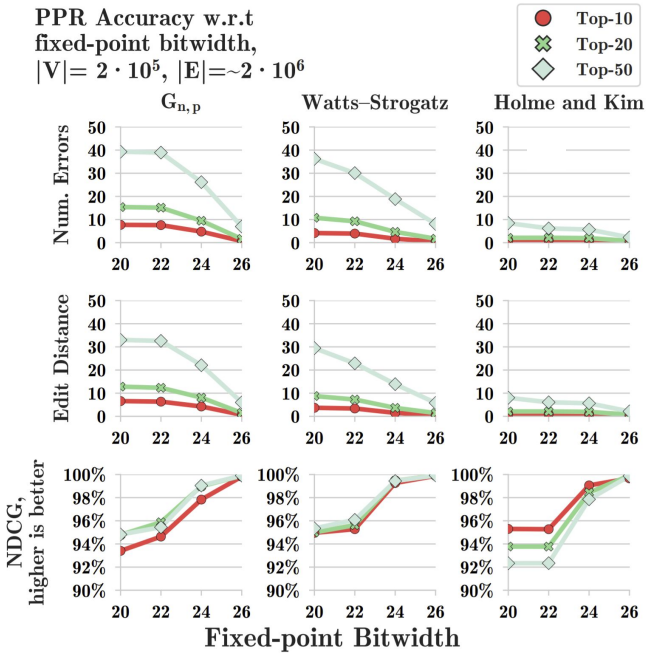
Alberto Parravicini
alberto.parravicini@polimi.it

Accuracy of reduced-precision

Accuracy of results tested with many metrics

- Num. of errors, NDCG, Precision, Kendall's τ, ...
- 26 bits provide no accuracy loss

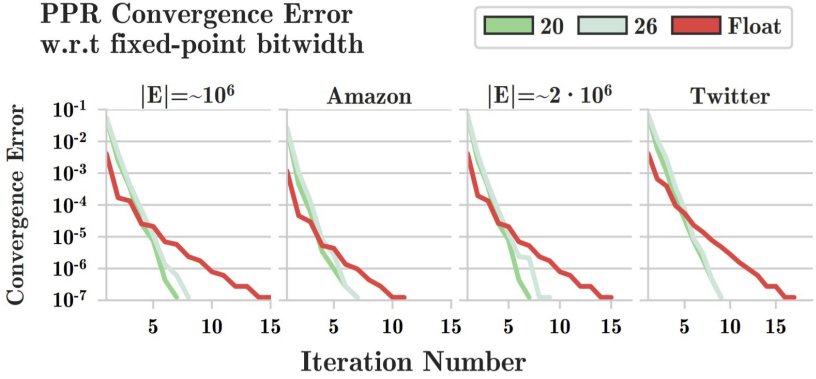


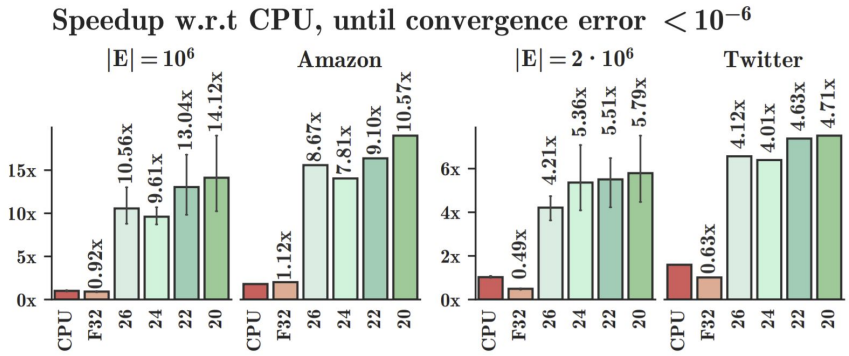


Alberto Parravicini
alberto.parravicini@polimi.it

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!





Alberto Parravicini
alberto.parravicini@polimi.it

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

Alberto Parravicini
alberto.parravicini@polimi.it

- 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

POLITECNICO MILANO 1863
NECST laboratory

POLITECNICO
MILANO 1863

ASP-DAC 2021, January 20th