Offloading Embedding Lookups to Processing-In-Memory for Deep Learning Recommender Models

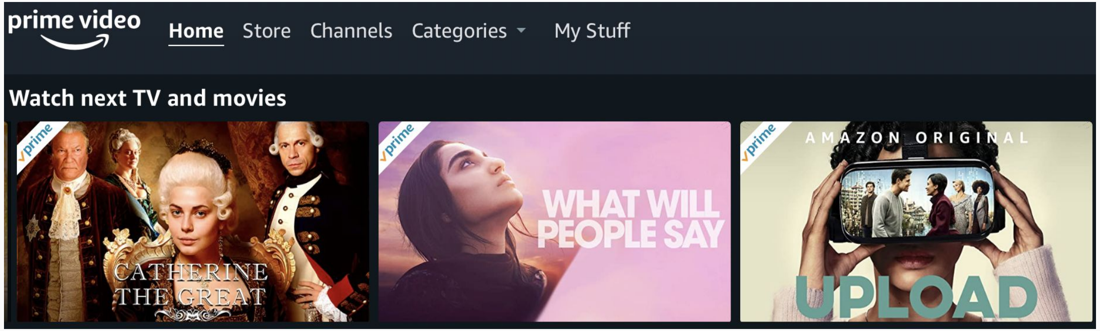
EuroPar 2024- ABUMPIMP
Niloofar Zarif, Justin Wong, Alexandra Fedorova
Aug 2024

a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

UBC Systopia
Research Group

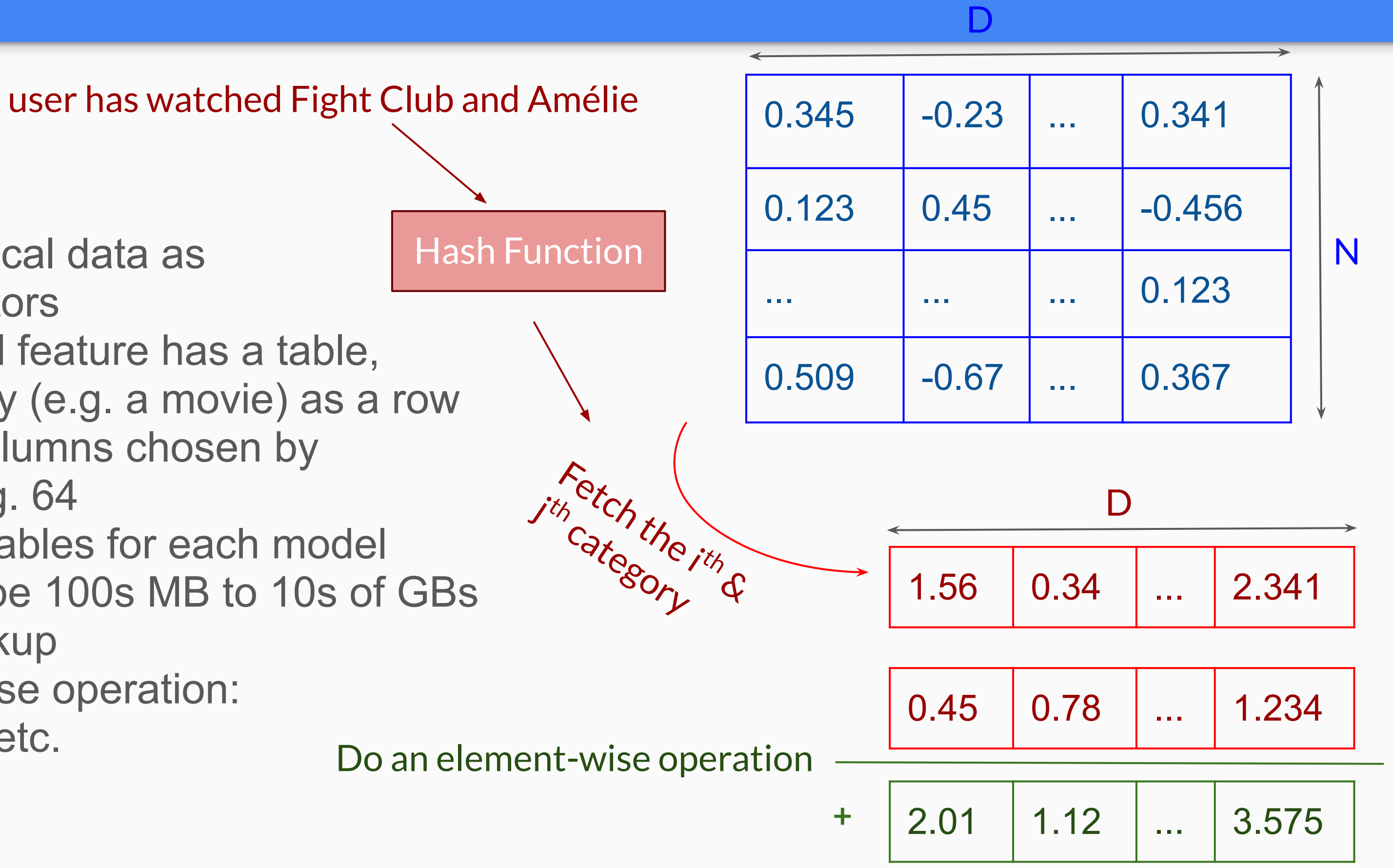
Recommender Models

- Recommender Systems in our everyday life:
  Facebook Marketplace, Google Ads, Netflix
- Deep Learning for Recommender Models
- Different from DNN or RNN
- Features:
  - Numerical
  - Categorical
- Embedding Layers



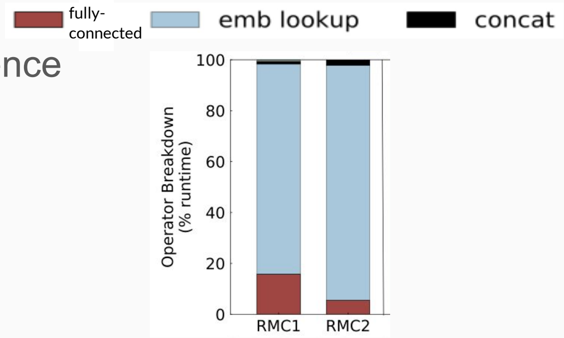
Embedding Layer

- Present categorical data as normalized vectors
- Each categorical feature has a table,
- N: Each category (e.g. a movie) as a row
- D: Number of columns chosen by the engineer, e.g. 64
- Usually tens of tables for each model
- Each table can be 100s MB to 10s of GBs
- Embedding Lookup
  - Element-wise operation: max, sum, etc.



DLRM Inference Workload

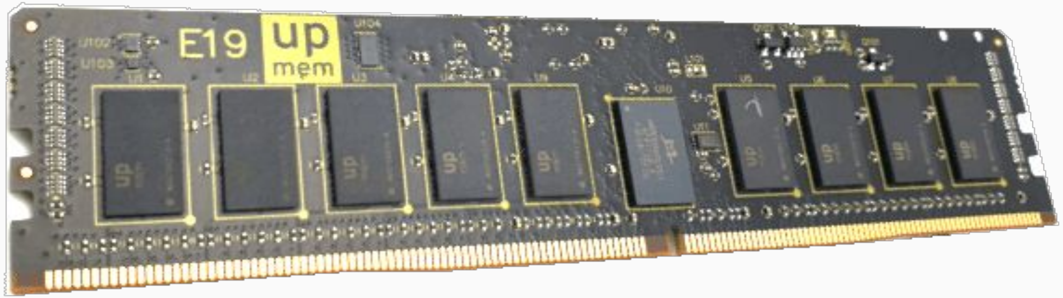
- DLRM: Meta’s recommender system
  - MLP
  - Embedding Layer
- Low Inference Latency important -> CPU prefered
- There are models with more than 80% of execution time of each inference cycle spend on embedding lookup[1]
- Embedding lookups:
  - Very Irregular memory accesses -> higher MPKI and lower IPC
  - Low computational intensity -> lower FLOPS
- PIM-Rec:
  - Use Processing-In-Memory for Embedding Lookups



[1] Gupta, Udit, et al. "The architectural implications of facebook's dnn-based personalized recommendation." *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020.

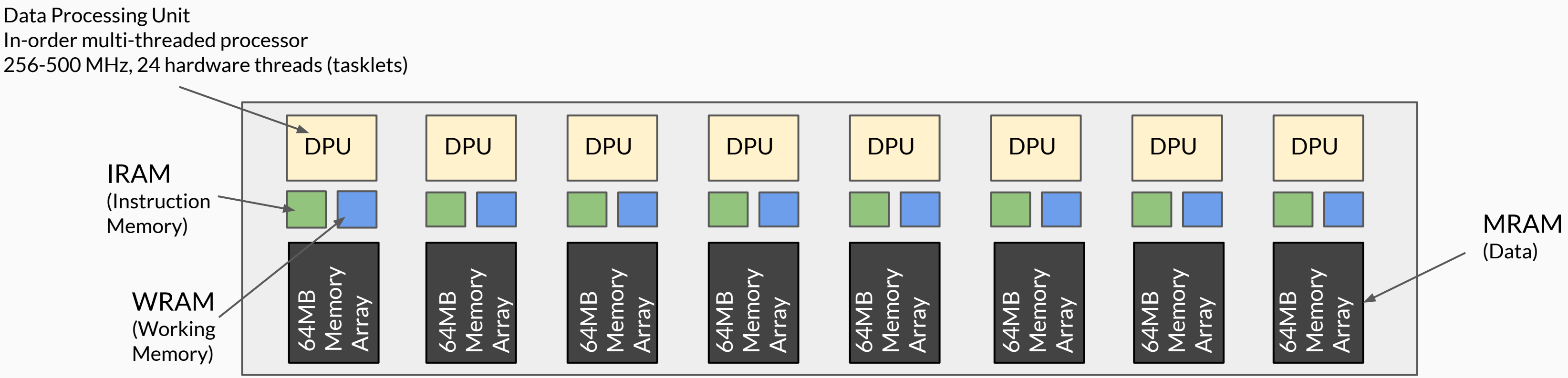
UPMEM PIM Solution

- Perform computations right where the data lives and avoid memory wall (limited memory bandwidth)
- This approach has been used before but with specialized hardware:
  - Do this with the first commercially available PIM solution, that is a drop-in replacement for existing DRAM
- UPMEM DRAM: Delivered as standard DDR4 DIMM modules



UPMEM PIM Architecture

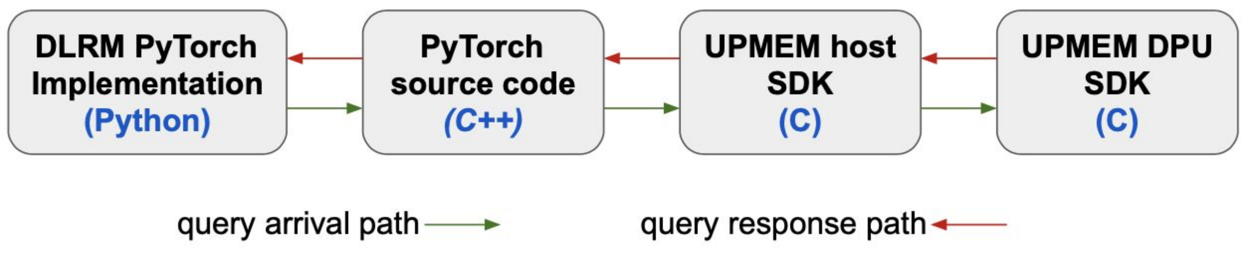
- Constraints:
  - No cross-dpu memory sharing
  - Cannot process floating point
- Huge bandwidth potential
- Each DIMM, 2 ranks and each rank 64 DPUs



UPMEM PIM DRAM Chip

Design Challenges

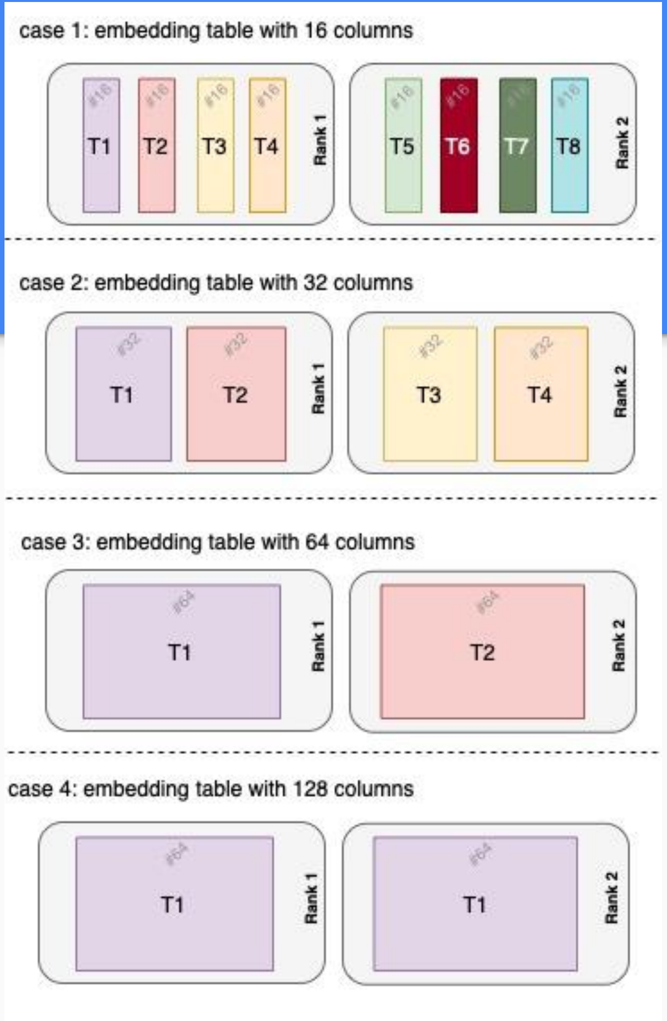
- Minimal implementation overhead
  - Python vs. C memory management
- No inter-DPU communication
- No floating point operation



PIM-Rec Design

- Loading embedding tables to UPMEM memory
  - Break tables into columns (16,32 or 64)
  - Each column copied to 1 DPU
  - Turn 32-FP values into 32-int
  - Pre-processing done just once

- Receiving lookup query
  - Break down for each table
  - Copy to corresponding DPUs
  - Aggregate on host-side
  - Turn 32-int back into 32-FP

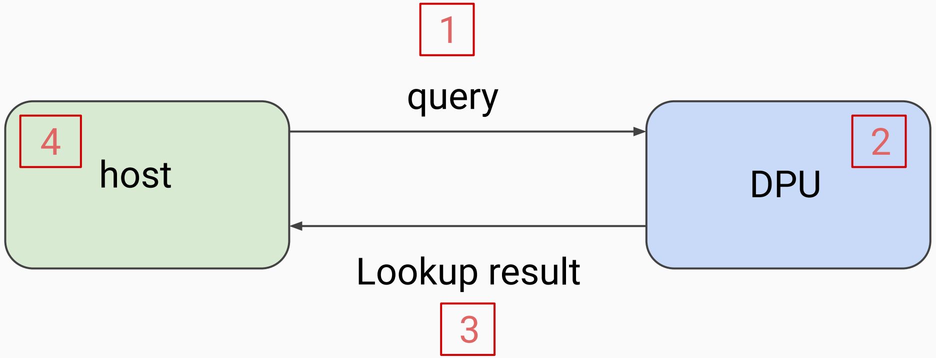


PIM-Rec Design(cont.)

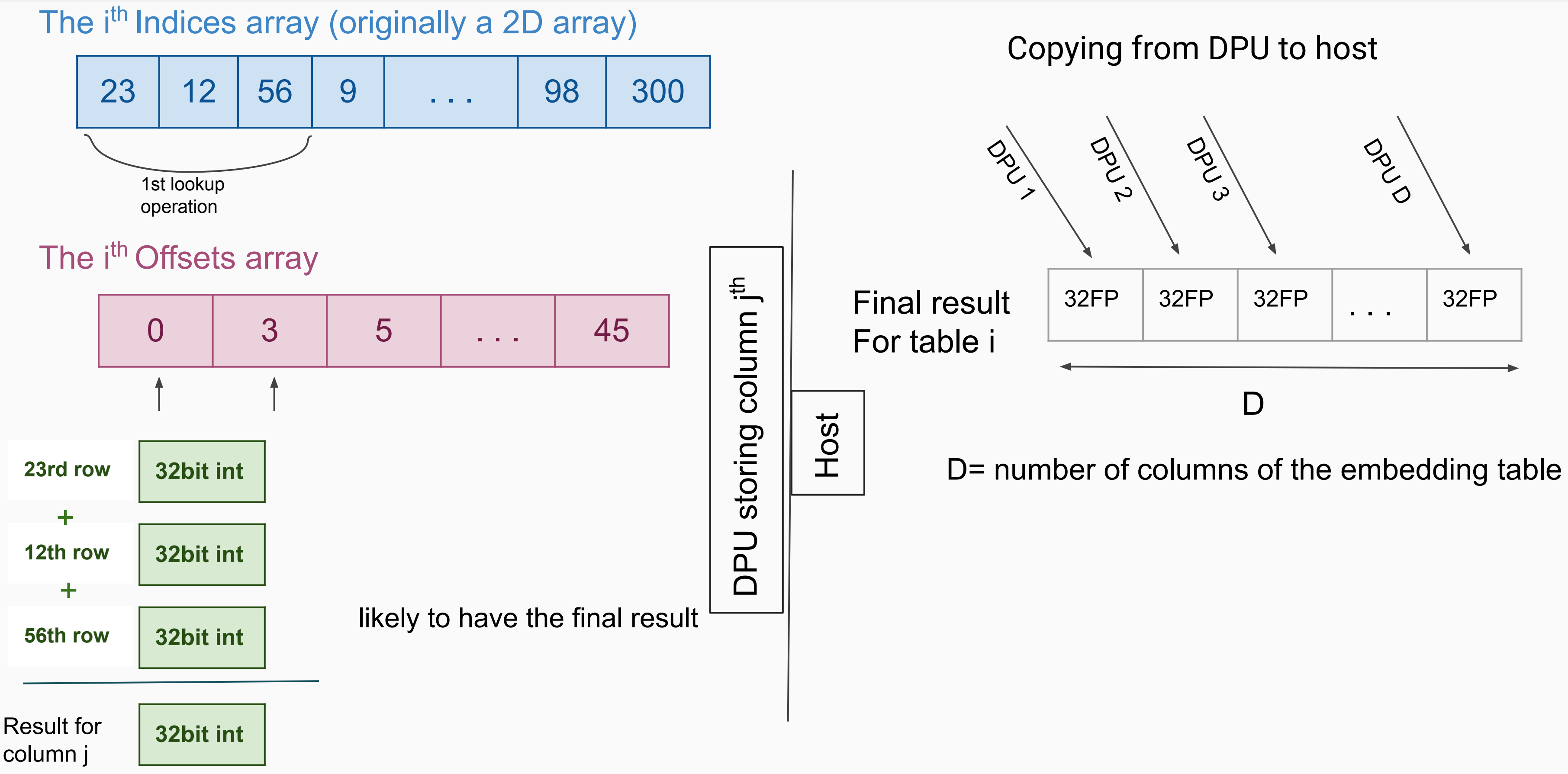
- Loading embedding tables to UPMEM memory
  - Break tables into columns (16,32 or 64)
  - Each column copied to 1 DPU
  - Each table copied to at least 1 rank
  - Turn 32-FP values into 32-int

- Receiving lookup query

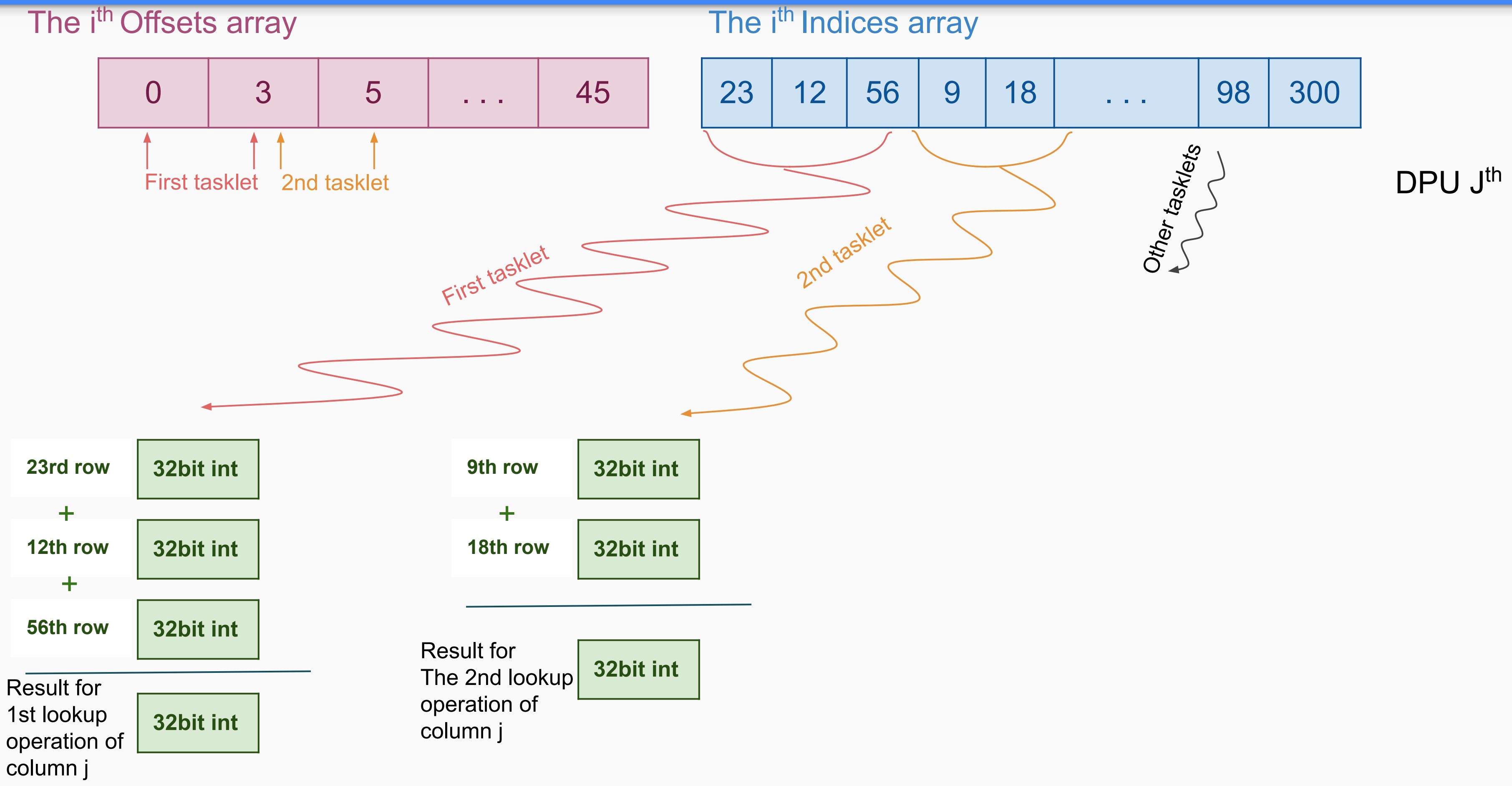
1. Break query and copy to DPUs
   a. Parallel transfers
2. Process in DPU and store in mram
3. Copy from MRAM (DPU) to host
4. Turn 32-int back into 32-FP



Lookup Processing in DPUs



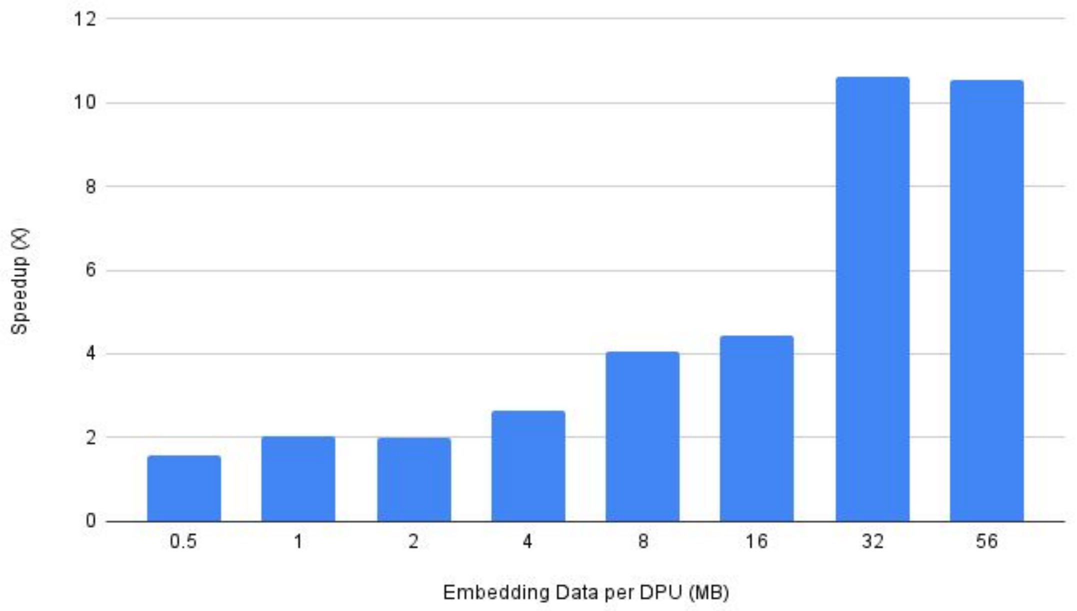
Parallelism in DPUs: Tasklets



Experimental Results

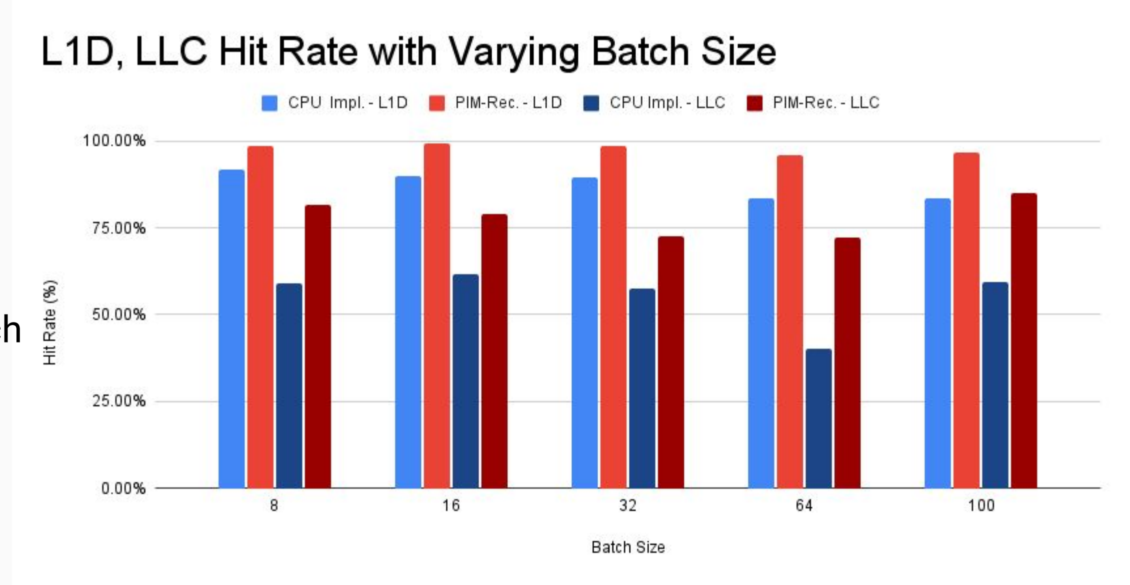
Speedup

- 2048 DPUs
  - 32 embedding tables
  - 64 columns per table
- 0.5 to 56 MB data per DPU
  - 125K to 13.9M 32bit integers
- 30 KB queries
  - Batch size of 64
  - ~120 lookup operation per batch
- 1 to 114 GB total embedding data
  - 32 tables
  - 0.5 to 56 MB per table



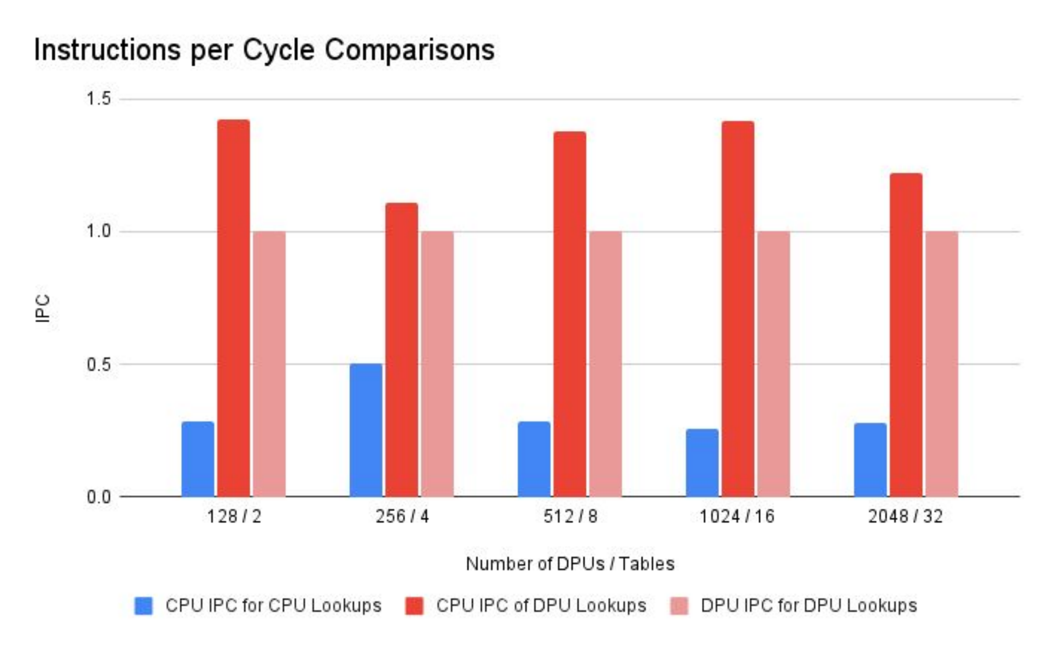
Cache Hit Rate

- 2048 DPUs
  - 32 embedding tables
  - 64 columns per table
- 2 MB data per DPU
  - 500K 32bit integers
- 3.8 to 48 KB queries
  - Batch size of 8 to 100
  - ~120 lookup operation per batch
- 4 GB total embedding data
  - 32 tables
  - 2 MB per table



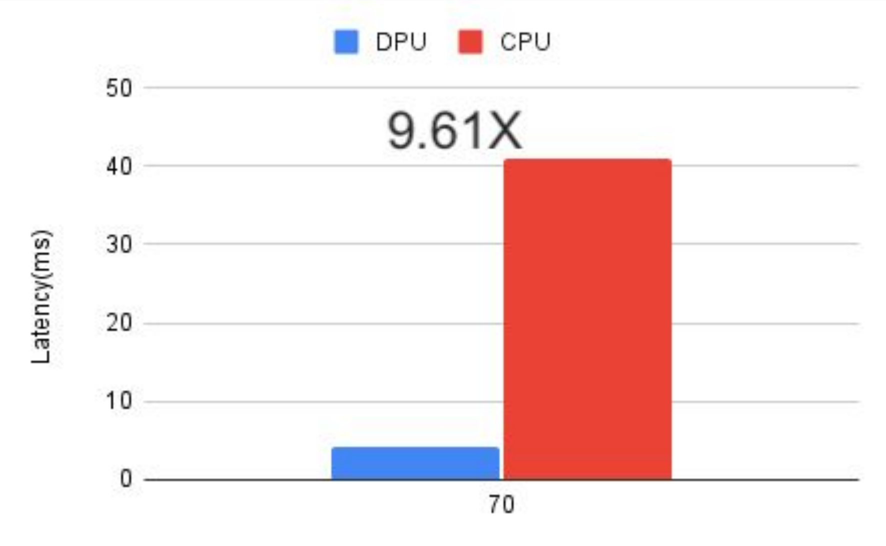
Processor Performance

- 128 to 2048 DPUs
  - 2 to 32 embedding tables
  - 64 columns per table
- 2 MB data per DPU
  - 500K 32bit integers
- 30 KB queries
  - Batch size of 64
  - ~120 lookup operation per batch
- 256 MB to 4 GB total embedding data
  - 2 to 32 tables
  - 2 MB per table



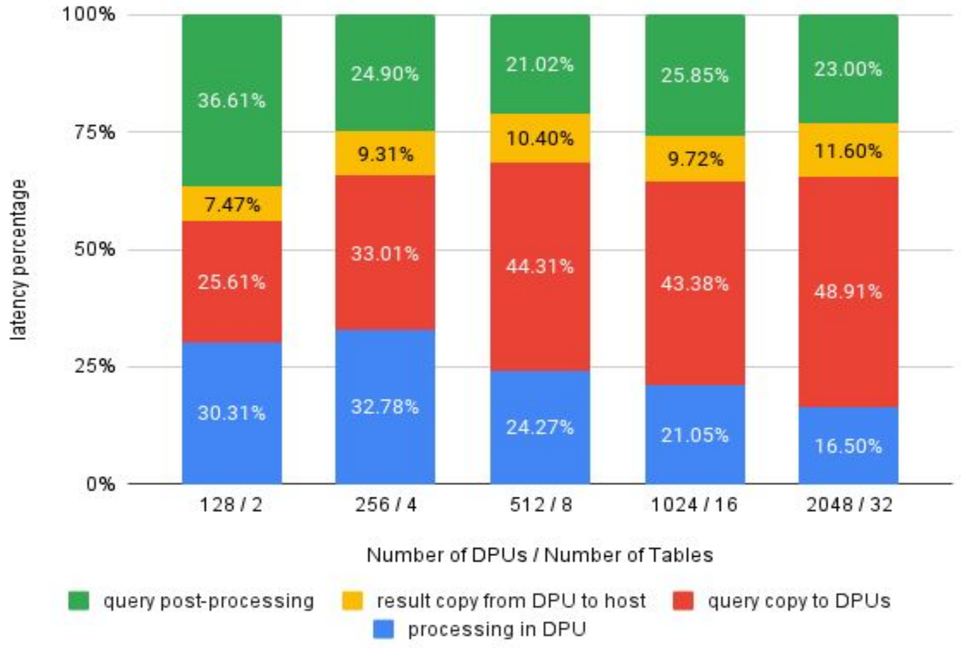
Favourable Workload

- 4480 DPUs
  - 70 embedding tables
  - 64 columns per table
- 400 KB data per DPU
  - 100K 32bit integers
- 4 KB queries(32bit int)
  - Batch size of 16
  - ~ 64 lookup operation per batch
- 448 MB total embedding data
  - 64 tables
  - 6.6 MB per table



Latency Breakdown

- 128 to 2048 DPUs
  - 2 to 32 embedding tables
  - 64 columns per table
- 2 MB data per DPU
  - 500K 32bit integers
- 30 KB queries
  - Batch size of 64
  - ~120 lookup operation per batch
- 256 MB to 4 GB total embedding data
  - 2 to 32 tables
  - 2 MB per table



Conclusion

- PIM-Rec offers up to 10.5X speedup
- CPU used more efficiently, higher IPC
- Cache used more efficiently, higher LLC and L1D hit rate
- UPMEM PIM lookups exhibit promising scalability

Further experimental results:

MSc Thesis on UBC library

Thank you!

Special Thanks to Prof. Alexandra(Sasha) Fedorova and Justin Wong!

Questions?