UNIVERSITÀ
DI SIENA 1240

Imperial College
London

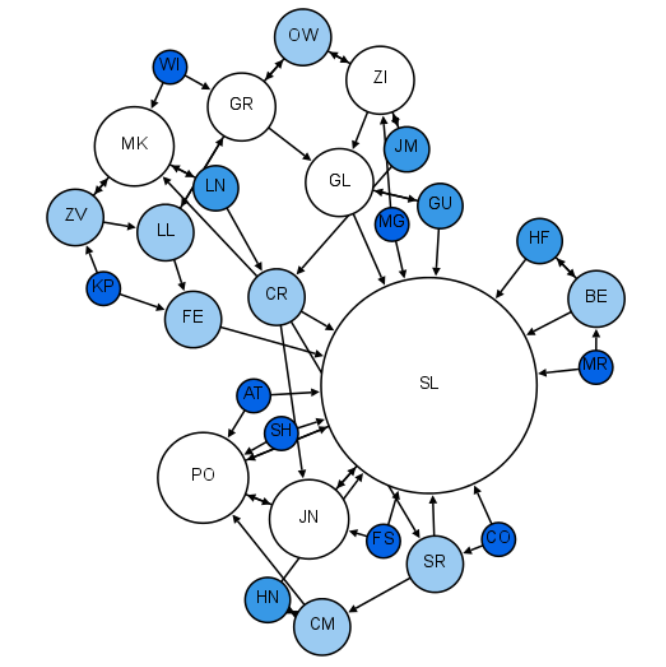
Accelerating Large-Scale Graph-Processing with FPGAs: Lesson Learned and Future Directions

Marco Procaccini*, Amin Sahebi*, Marco Barbone§, Georgi Gaydadjiev§, Wayne Luk§, Roberto Giorgi*

*University of Siena, Italy - §Imperial College London, UK

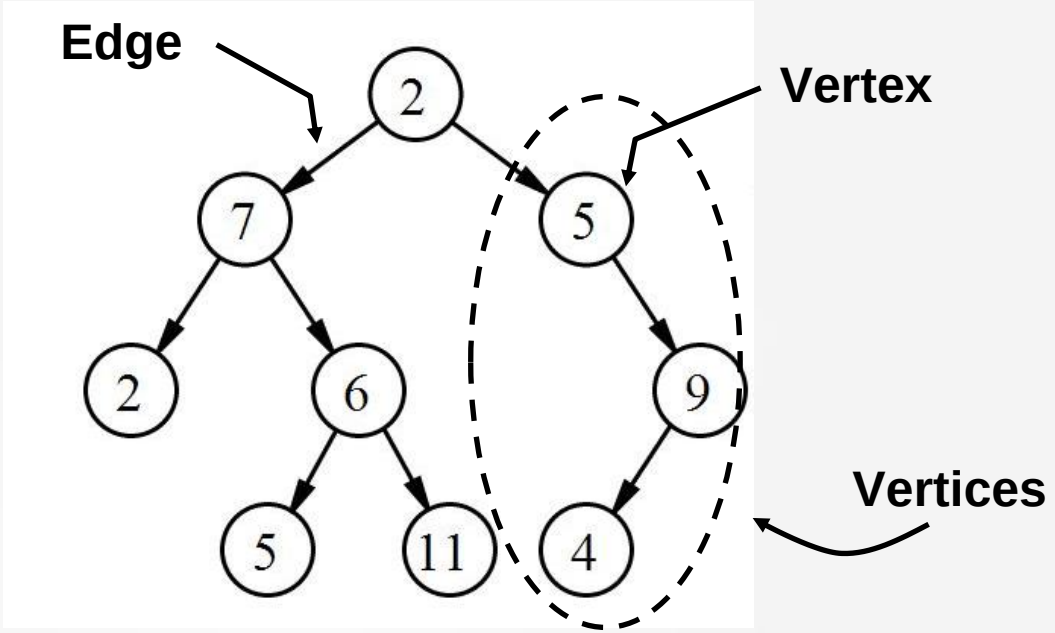
PARMA-DITAM Workshop
HiPEAC 2024 Conference
17-19 January. 2024, Munich - DE

Graphs: a common way to represent information



https://commons.wikimedia.org/wiki/Category:Sociology

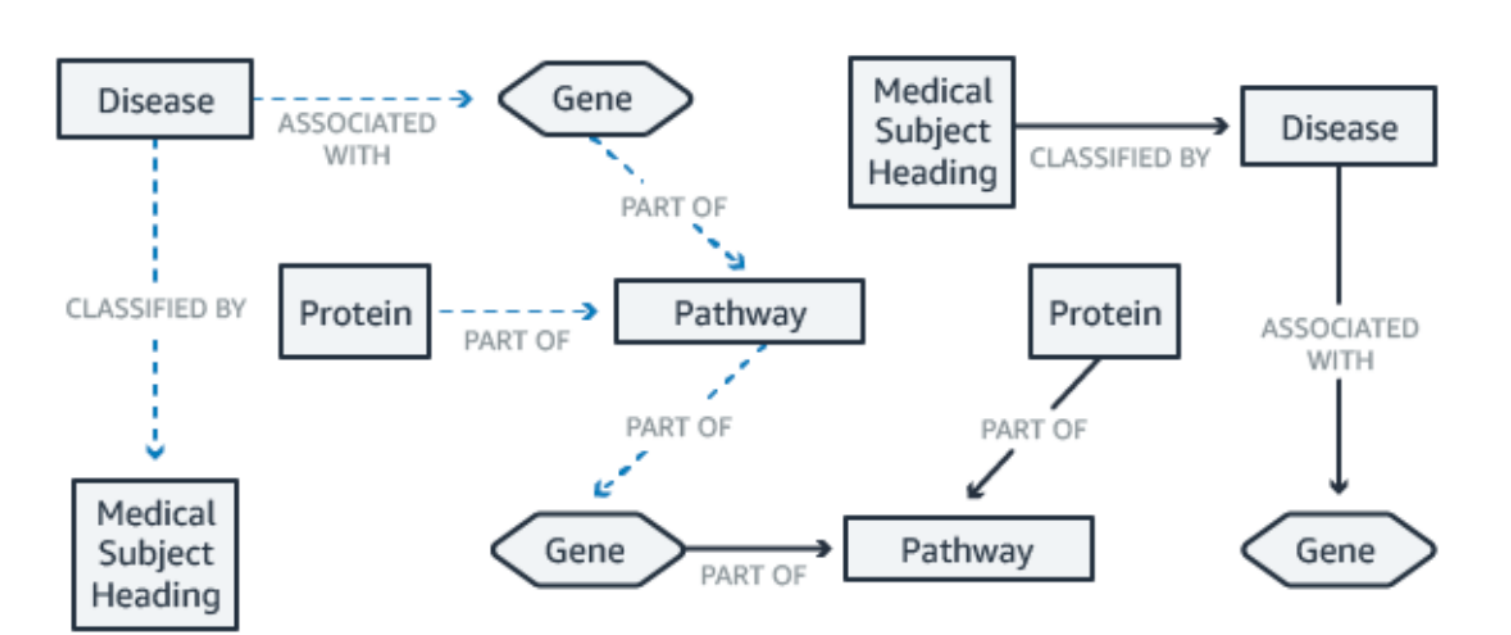
G(V,E)



All the ecommerce space on the internet is driven by graphs.
Such as **web** or **social networks**, **computational problems**, **analytics**, **machine learning, etc.**

Why processing graphs

https://www.infoq.com/news/2018/06/aws-neptune-graph-ga/

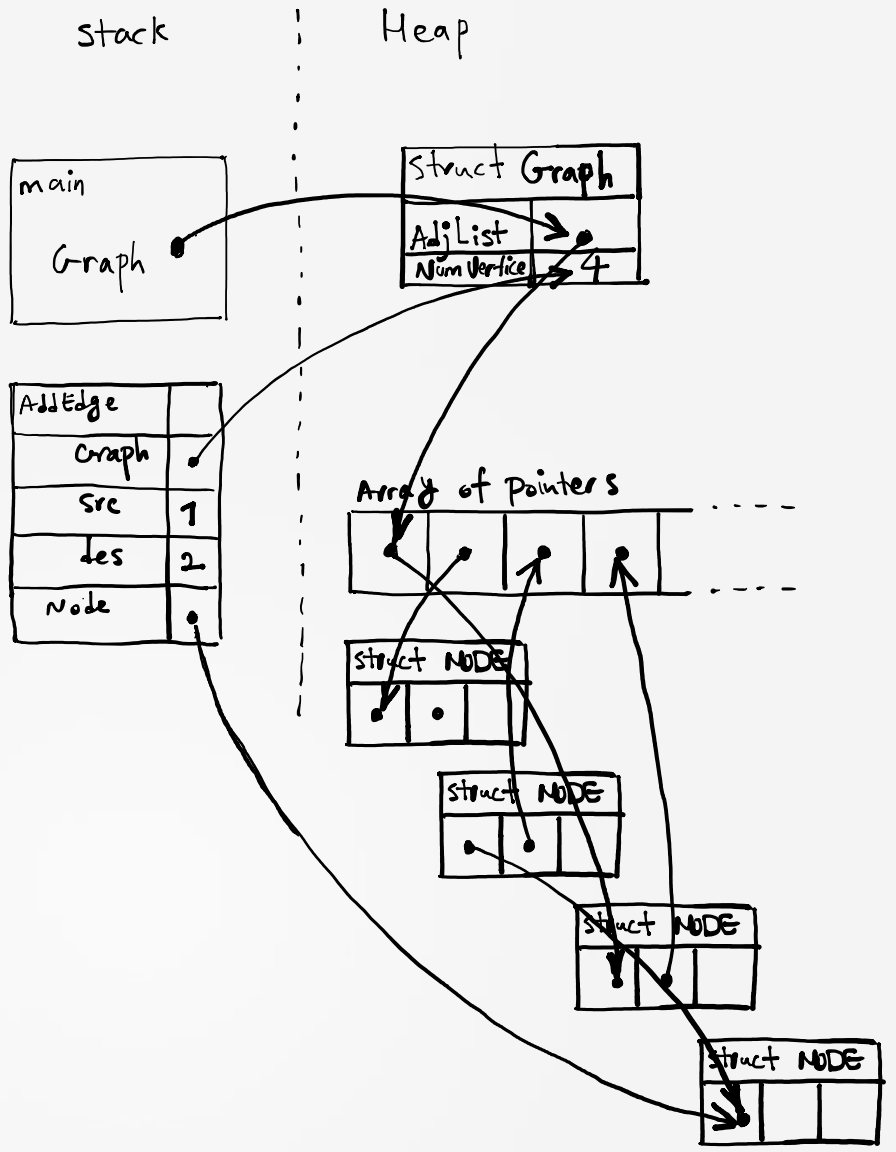


- Analyzing data to get more knowledge
- Improve the process efficiency
- Get insights/recommendations with machine learning and reducing the overall cost

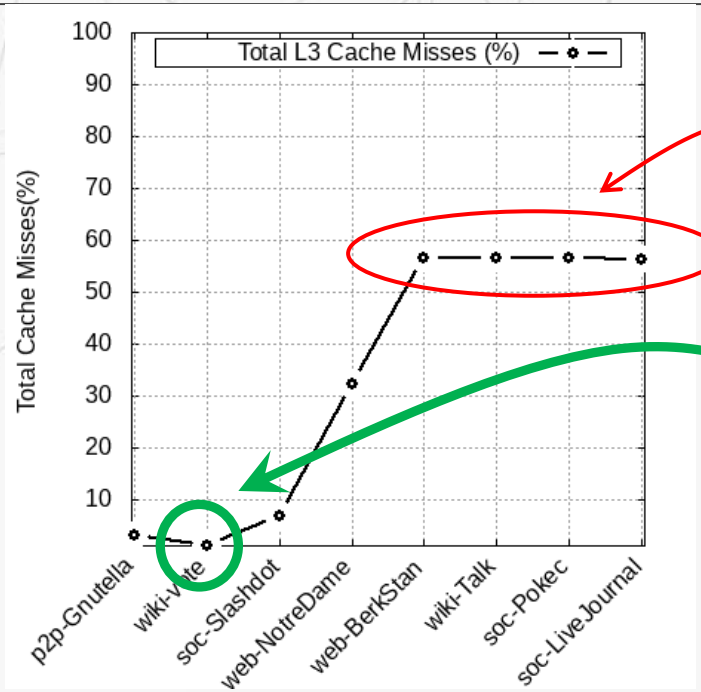
What are the Challenges?

- Graphs are getting bigger and bigger every day!
  - Large-scale graphs size is in the order of GB/TB
  - Scalability:
    ability to adapt to larger and larger graphs

- Computing graphs in an efficient way
  - The irregular structure of the graph usually leads to a highly random memory access which typically impact negatively CPU and GPU memory hierarchy performance
  - Scarce locality:
    data accessed randomly throughout the memory

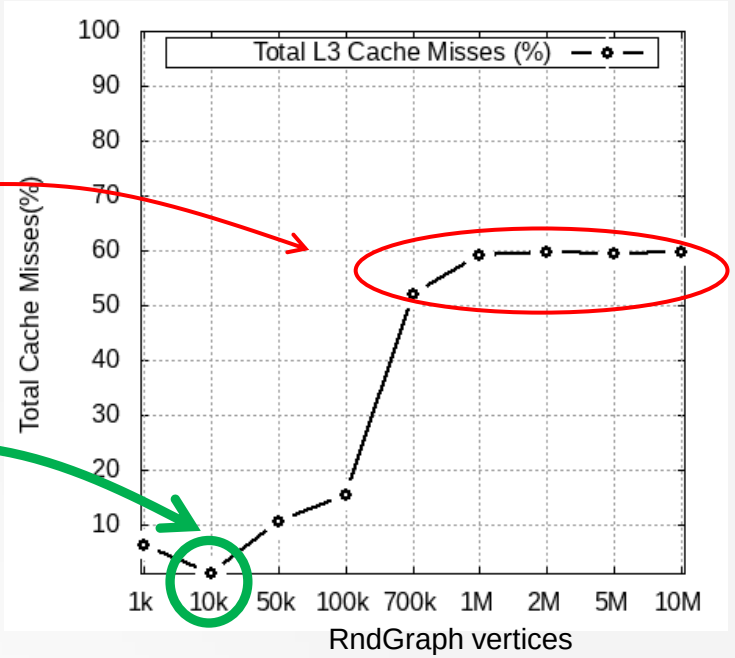


Cache performance for large and sparse graphs



When the graphs size exceeds L3 cache size,
the L3 cache misses jump to 60%

When the graph size fits the cache, the L3 cache misses are low



Real world database	V	E	Size on Disk
P2p-Gnutella	6301	20777	211 K
Wiki-vote	7,115	103,689	1 MB
Soc-Slashdot	77,360	905,468	11 MB
web-NotreDame	325,729	1,497,134	21 MB
web-BerkStan	685,230	7,600,595	106 MB
wiki-Talk	2,394,385	5,021,410	64 MB
soc-Pokec	1,632,803	30,622,564	405 MB
soc-LiveJournal	4,847,571	68,993,773	1.1 GB

Intel Core i7-8700K
– 6 Cores (12 hw-threads), Freq 3.60 GHz
L1i Cache 32K
L1d Cache 32K
L2 Cache 256 K
L3 Cache 12288K (each hw-thread 1MB)

PAPI Version: 6.0.0.1

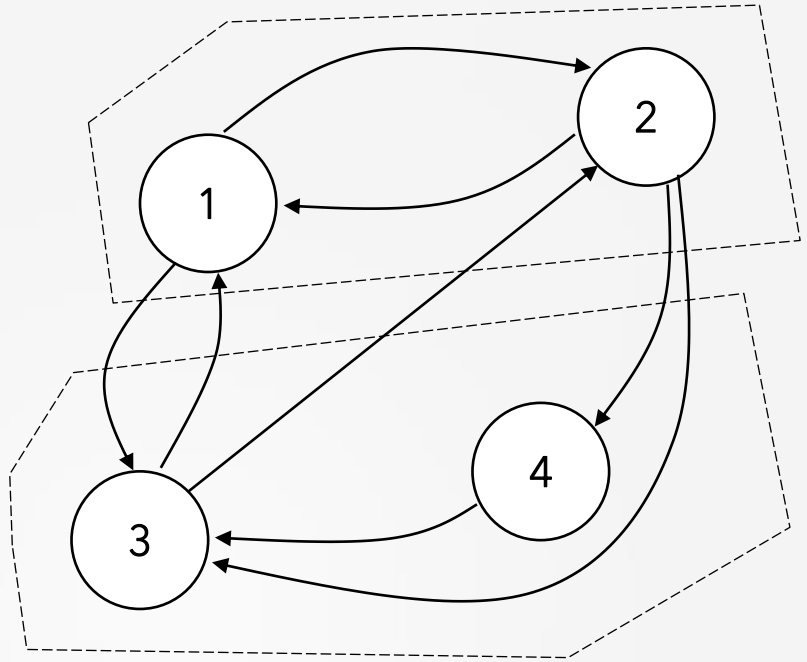
Synthetic database	V	E	Size on Disk
RndG-1K	1000	10,000	76 KB
RndG-10k	10K	100K	1 MB
RndG-500k	50K	500K	5.6 MB
RndG-100k	100K	1M	12 MB
RndG-700k	700K	7M	92M
RndG-1M	1M	10M	132 MB
RndG-2M	2M	20M	284 MB
RndG-5M	5M	50M	760 MB
RndG-10M	10M	100M	1.5 GB

Why Partitioning?

1. If the graph is big, it does not fit in memory
2. It is unlikely that the graph fits in cache

- Problem: most algorithms perform random memory accesses

- A possible solution is to partition the graph into chunks that fit in cache and/or memory



Software Baseline

GridGraph

https://github.com/thu-pacman/GridGraph.git

Multicore Implementation using Grid Preprocessing and OpenMP

EverythingGraph

https://github.com/jmalicevic/EverythingGraph.git

Comparison the existing algorithm and implementations and discuss which approach is suitable for which problem

Ligra

https://github.com/jshun/ligra.git

Multicore OpenMP and Cilk implementation

1) Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2015. GridGraph: large-scale graph processing on a single machine using 2-level hierarchical partitioning. In Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15). USENIX Association, USA, 375–386.

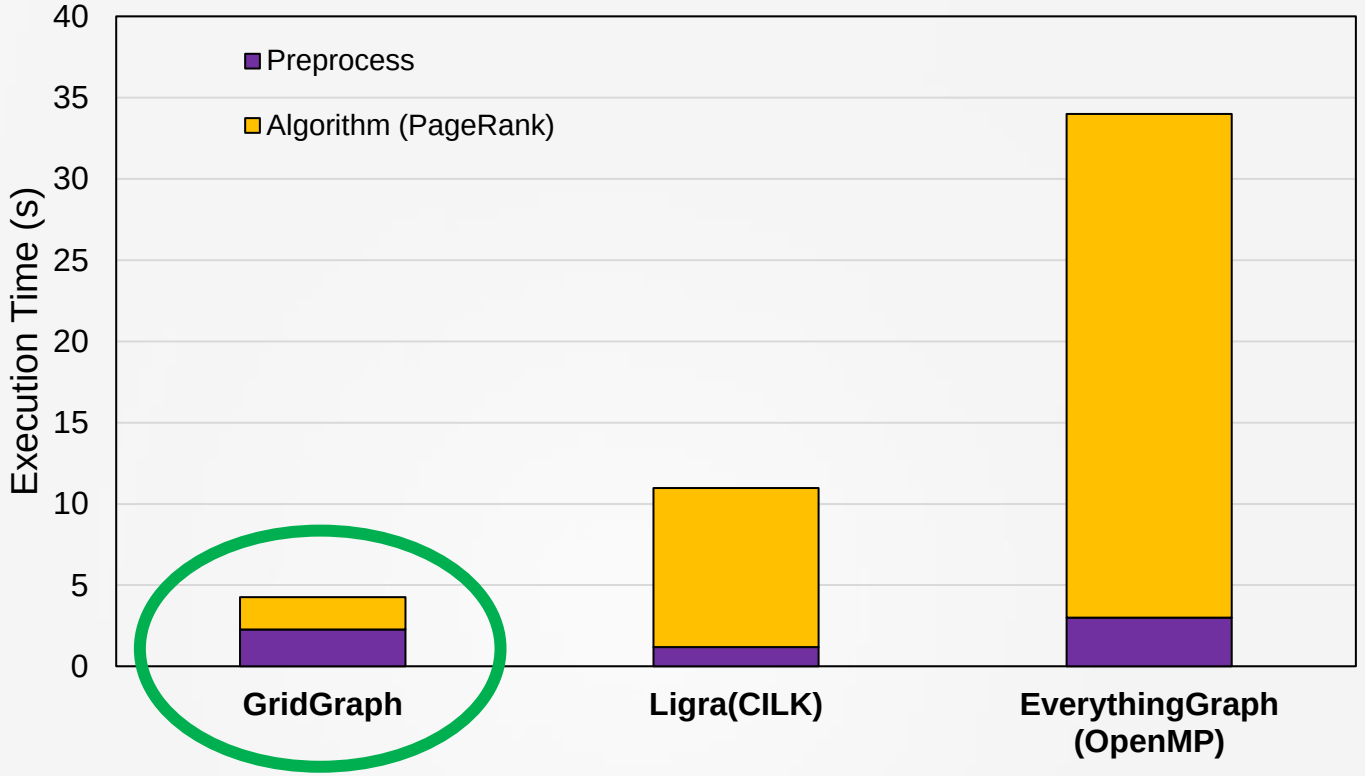
2) Jasmina Malicevic, Baptiste Lepers, and Willy Zwaenepoel. 2017. Everything you always wanted to know about multicore graph processing but were afraid to ask. In Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17). USENIX Association, USA, 631–643.

3) Julian Shun and Guy E. Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. SIGPLAN Not. 48, 8 (August 2013), 135–146.

Baseline (execution time metric)

The baseline for above mentioned applications is the parallel (multicore) implementation on Conventional Machines. C/C++ implementation using parallelizing Multithread libraries such as OpenMP or opencilk!

Social Network Dataset: soc-LiveJournal
Nodes 4,847,571
Edges 68,993,773
10 Iteration over 12 Cores
(experiment re-done locally from
the available source code –see previous slide)

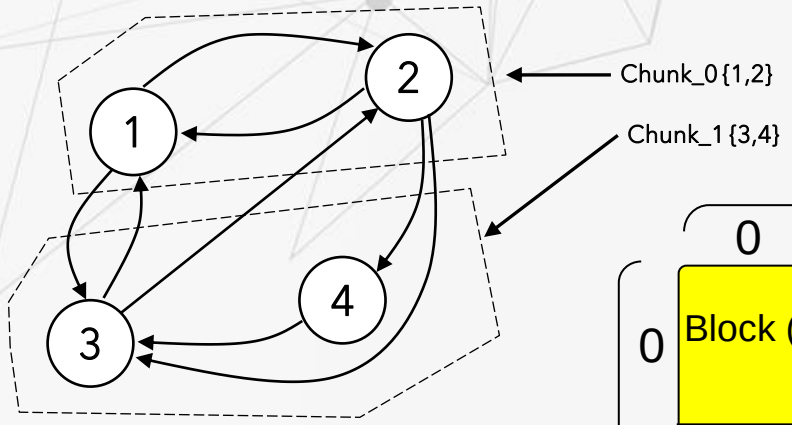


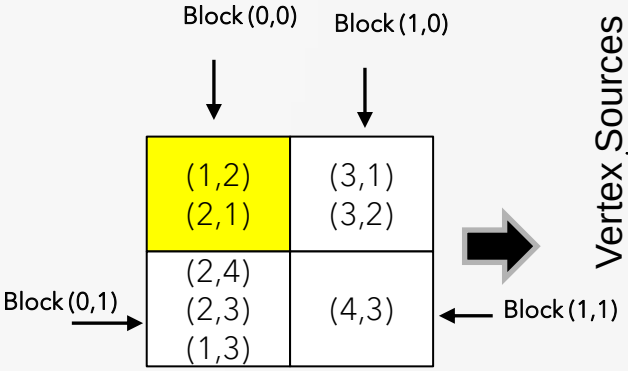
We have to measure end to end time.

End-to-end time = Pre-processing + Algorithm time

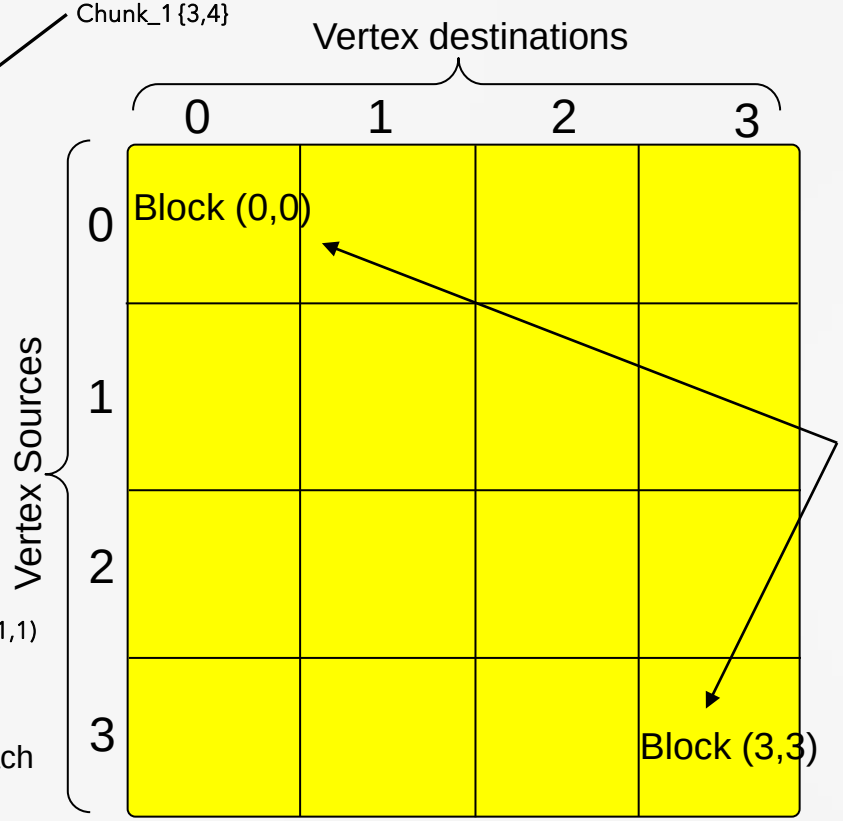
Preprocessing Step: GridGraph Partitioning Method

A given sample graph



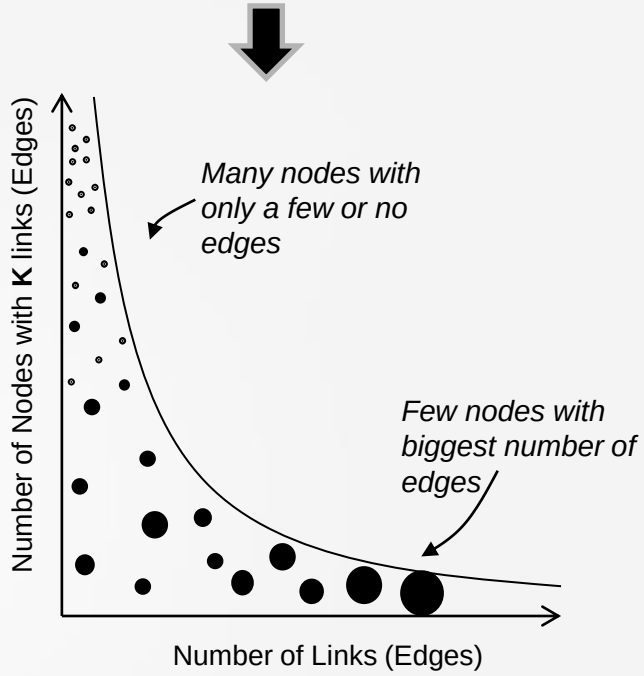


Partitioning based on GridGraph* approach



Data chunks are almost with same size all over the graph dataset

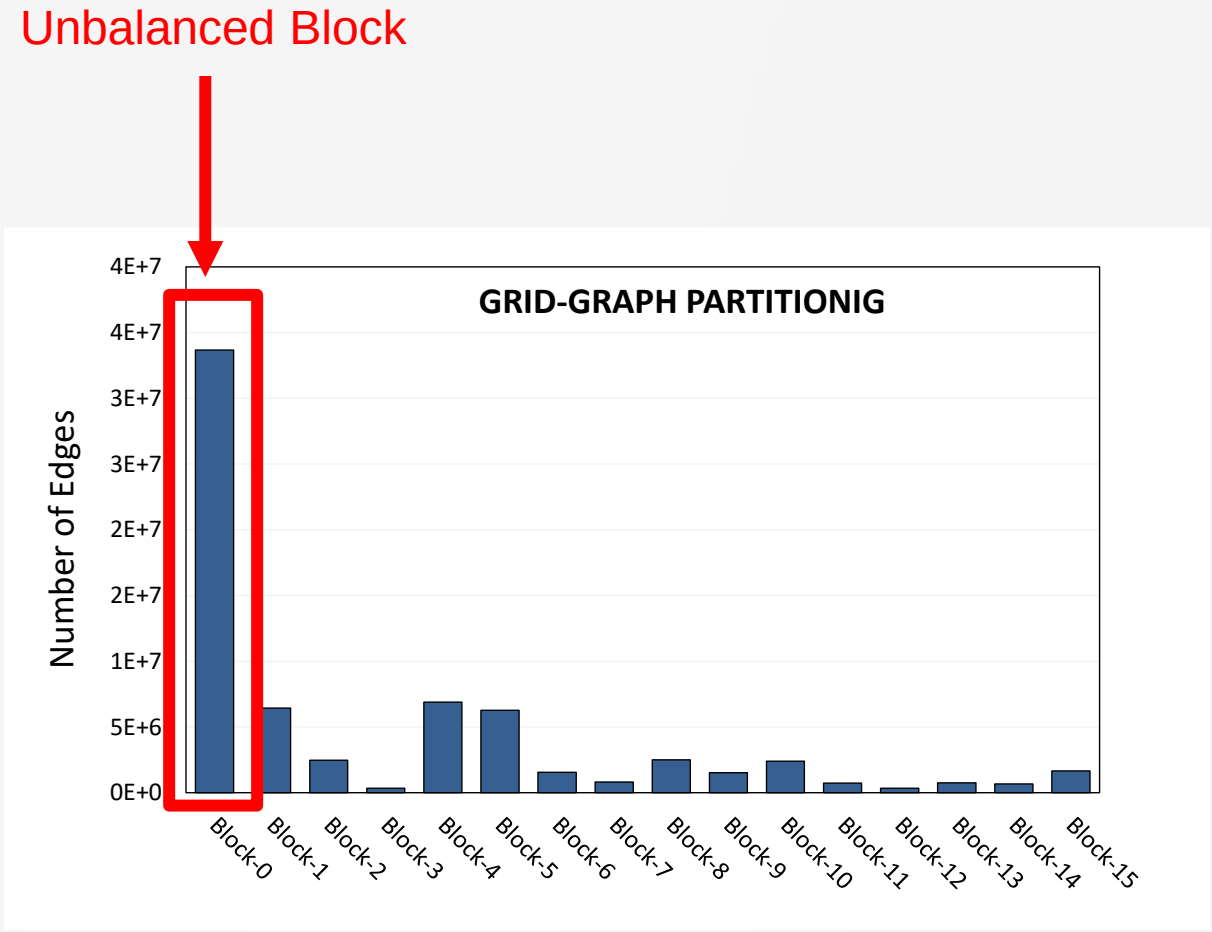
The challenge of partitioning is Real-world Powerlaw graphs which produces uneven block sizes of edge data



* Xiaowei Zhu, Wentao Han, and Wenguang Chen. “GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning”. USENIX ATC ’15. Santa Clara,

Lesson Learned: GridGraph partitioning technique can be further optimized

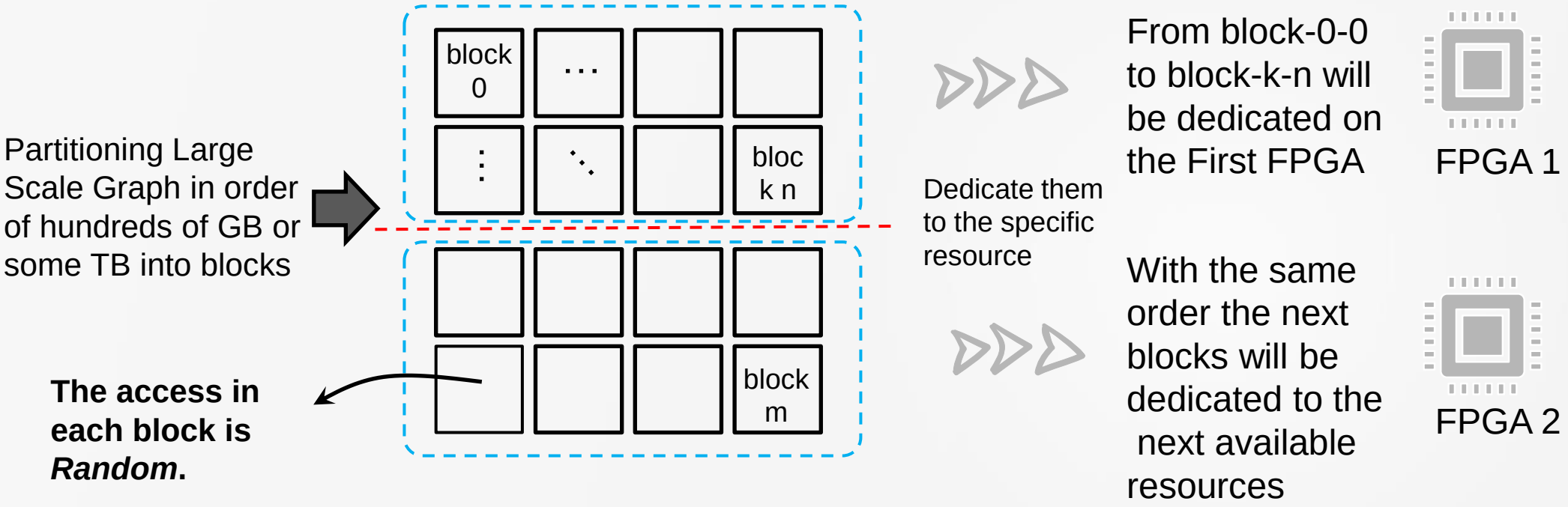
- GridGraph produces unbalanced edge-blocks
- For example, in the the LiveJournal dataset, most of the edges are placed in the block-0
- Problem: this creates unbalanced workload



GridGraph partitioning analysis of the LiveJournal dataset

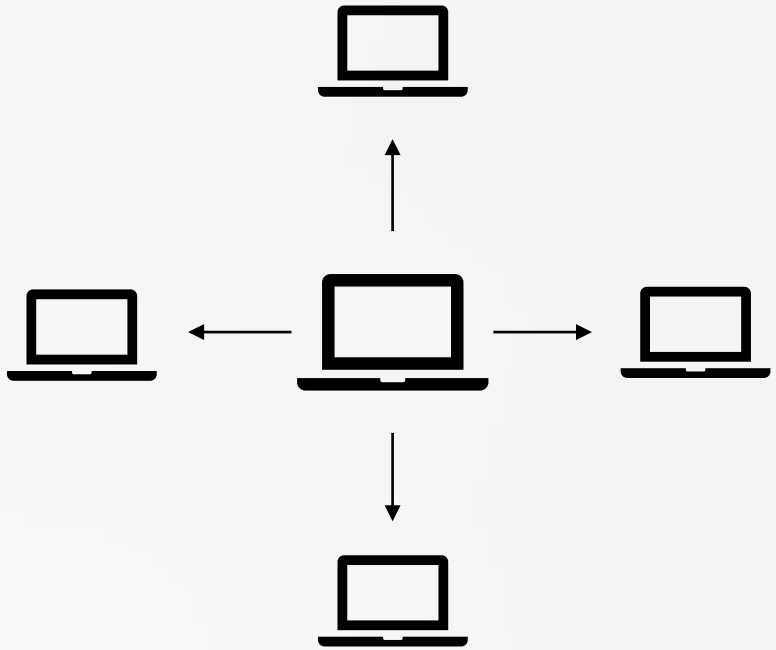
Sub-blocks partitioning (hierarchical partitioning)

- Graphs are large-scale and each partitions will be in order of some GB
- A sub-block partitioning is needed to assign each sub-block to a specific resource
- Blocks are streamed-in sequentially and treated independently
- Memory access in each block is random



Why distributed?

- Both the “data size” and “computation” are expensive and time consuming on one single node
- Data intensive problems need a combination of nodes in a network to solve the problem



- Advantages
  - Resources sharing
  - Economics: distributed is less expensive and can be easily be on cloud
  - Scalability is achieved by using the appropriate number of nodes

- Disadvantages
  - Communication issues
  - Faults are more frequent

Choosing an appropriate hardware

Running on CPU and GPU?

- Poor locality
- Lack of scalability
- Random data access pattern
- Heavy data conflicts

	Advantages	Disadvantages
General Purpose CPUs	Development time, less effort to program	Fixed memory access granularity based on cache line size
		Do not have flexible high-degree parallelism
		Cache problem with irregular graph processing with Little or No temporal and Spatial Locality

	Advantages	Disadvantages
GPUs	Easy to develop, high-degree of data parallelism	Poorly managing irregular accesses

	Advantages	Disadvantages
FPGAs	Easy to have rapid prototype application specific hardware	Hard to develop, not easy and not flexible programming environment, Is not flexible with MATH operations like floating point
	Do not need to decode instructions, it's Data-Driven	
	Massive parallelism scale	
	In the case of Random Memory accesses and Branch-Prediction problem in CPU or GPU, FPGA could easily alleviate these problem by its structure	
	Large cumulated bandwidth on-chip memory (BRAMS), store reusable data and take advantage of temporal locality	
	Power Efficient	

FPGA can be the optimal candidate because of many good features that can overcome CPU and GPU

To achieve scaling: a distributed Graph Processing Framework on FPGA clusters is a challenge

A Taxonomy of the more similar related studies

Work	Distributed	Language	Implementation	Access to Host Memory	Evaluation Size	Public repository?	FPGA Platform	Year
ForeGraph[1]	Yes	HDL	Simulation	No	Medium	No	Xilinx VCU110	2017
FabGraph[2]	No	HLS	Simulation	No	Medium	No	Xilinx VCU110 and VCU118	2019
HitGraph[3]	No	HDL	Hardware	No	Small	Yes	Xilinx Virtex US+	2019
ThunderGP[4]	No	HLS	Hardware	No	Medium	Yes	Alveo Family	2021
GraVF-M[5]	Yes	Python		No	Medium	Yes	Micron Pico se-6 platform	2019
GridGas[6]	Yes	HDL	Hardware	Yes	Medium	No	Xilinx Kintex	2018
FPGP[7]	No	HDL	Hardware	No	Medium	No	Xilinx Virtex-7	2016
Ref[8]	No	Chisel	Hardware	No	Large	No	Xilinx Virtex US+ (AWS Platform)	2021
GraphOps[9]	No	MAXJ	Hardware	No	Small	Yes	Maxeler Boards	2016
This work	Yes	HLS	Hardware	Yes	Very Large	Yes	Alveo Family	2023

*References in the next page

The repository is for public access: https://github.com/AminSahebi/distributed-graph-fpga

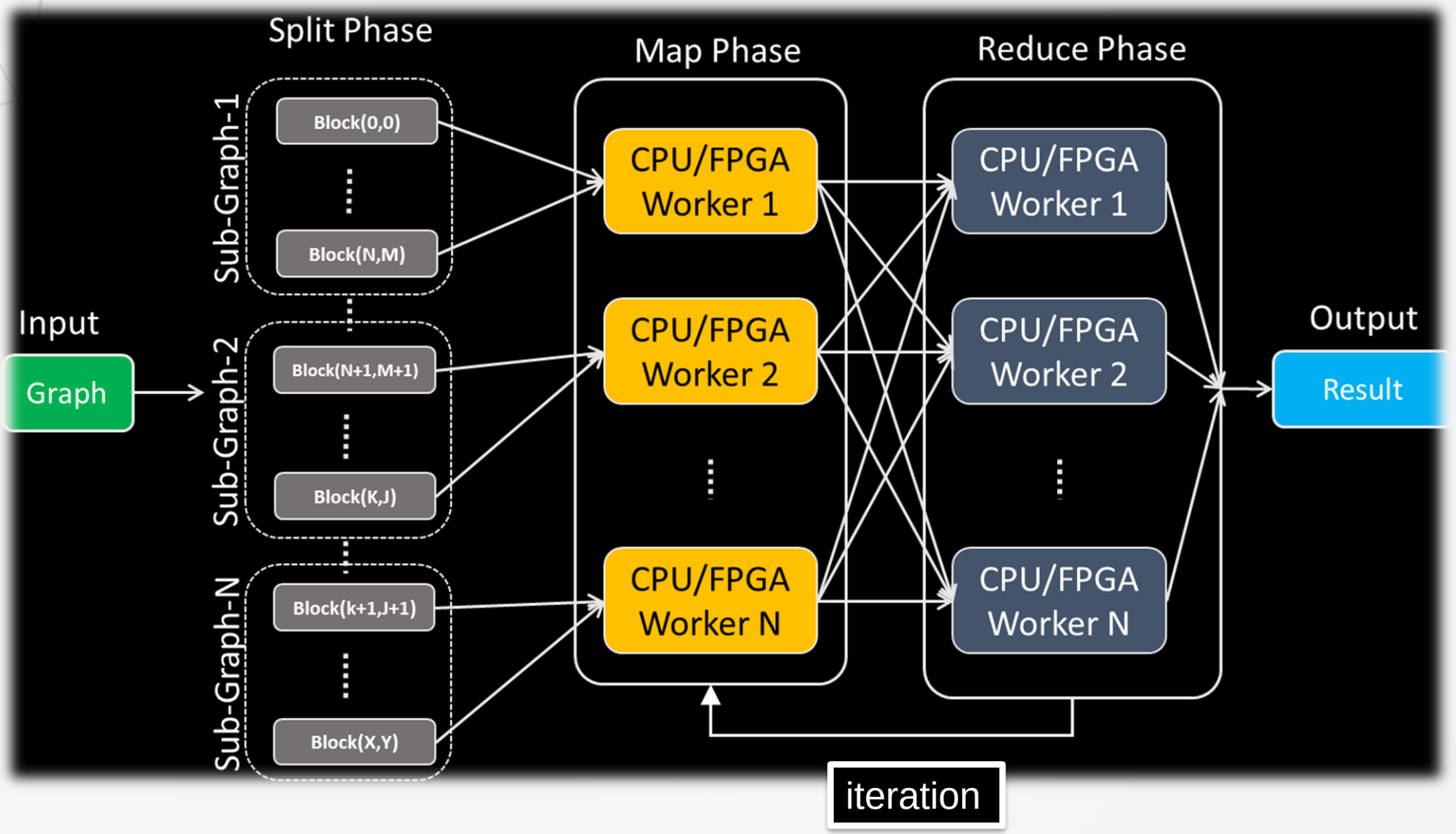
Related studies

[1] Guohao Dai, Tianhao Huang, Yuze Chi, et al. “ForeGraph: Exploring Large-Scale Graph Processing on Multi-FPGA Architecture”. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.
[2] Zhiyuan Shao, Ruoshi Li, Diqing Hu, Xiaofei Liao, and Hai Jin. “Improving Performance of Graph Processing on FPGA-DRAM Platform by Two-level Vertex Caching”, 2019.
[3] Shijie Zhou, Rajgopal Kannan, Viktor K. Prasanna, Guna Seetharaman, and Qing Wu. “HitGraph: High-throughput Graph Processing Framework on FPGA”. In: IEEE Transactions on Parallel and Distributed Systems(2019)
[4] Xinyu Chen, Hongshi Tan, Yao Chen, et al. “ThunderGP: HLS-Based Graph Processing Framework on FPGAs”. In: The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. FPGA ’21
[5] Nina Engelhardt and Hayden K.-H. So. “GraVF-M: Graph Processing System Generation for Multi-FPGA Platforms”. In: ACM Trans. 2019
[6] Yu Zou and Mingjie Lin. “GridGAS: An I/O-Efficient Heterogeneous FPGA+CPU Computing Platform for Very Large-Scale Graph Analytics”. In: 2018 International Conference on Field-Programmable Technology (FPT). 2018
[7] Guohao Dai, Yuze Chi, Yu Wang, and Huazhong Yang. “FPGP: Graph Processing Framework on FPGA A Case Study of Breadth-First Search”. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. FPGA ’16
[8] Mikhail Asiatici and Paolo Ienne. “Large-Scale Graph Processing on FPGAs with Caches for Thousands of Simultaneous Misses”. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)
[9] Tayo Oguntebi and Kunle Olukotun. “GraphOps: A Dataflow Library for Graph Analytics Acceleration”. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field Programmable Gate Arrays. FPGA ’16

The Architecture from high-level perspective

We consider **Hadoop** as the High-level Data Management Layer

We aim at using both CPUs and FPGAs as workers, for now we have achieved one CPU as host and multiple FPGAs as workers

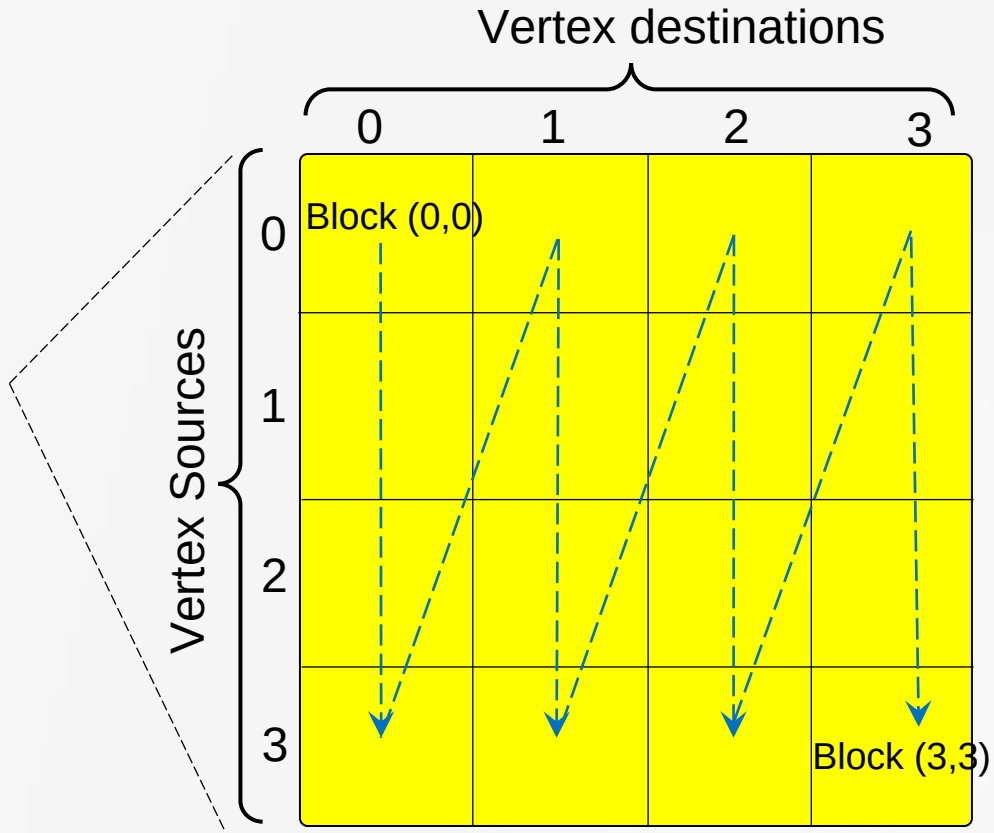


Block streaming to FPGA

Input

Graph

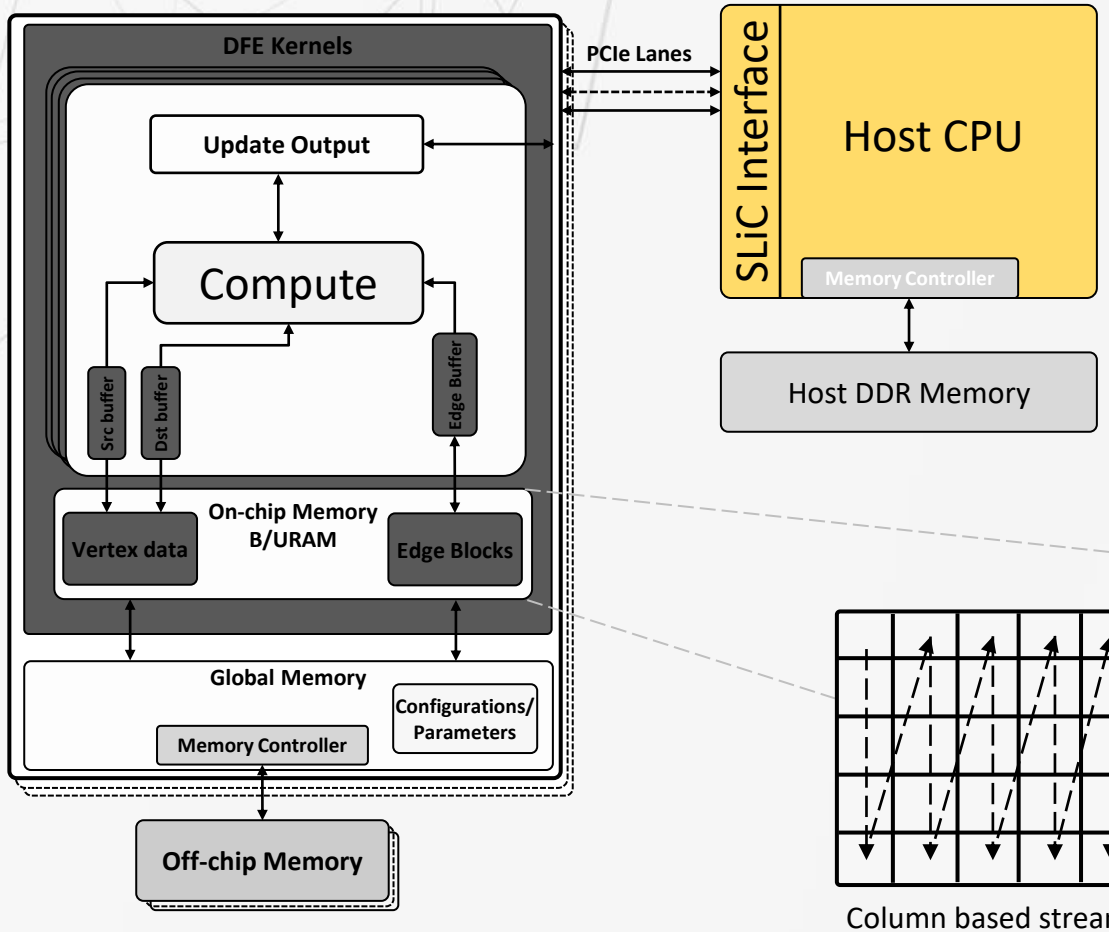
Dataset edge array turn into Grid structure using offline partitioning



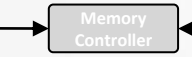
Blocks will be streamed sequentially from CPU to FPGA via PCIe (for the first scenario we decided to skip board-DRAM, and directly read and write from PCIe to FPGA)

Blocks are queued in RAM and transferred sequentially to the FPGA via PCIe. On-Board Memory unused since there is not partition reuse between iterations

The Hardware Design



Column based stream
order of block edges

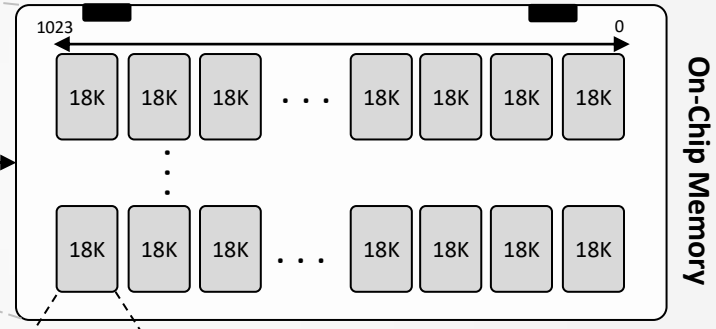


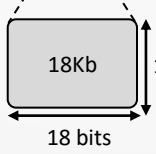
Edge blocks Stream
from PCIe(sequentially)

Multi-port configuration
needs more resources but provides Parallel Rd/Wr
access in one clock cycle.

Read/Write
from/to
Kernels

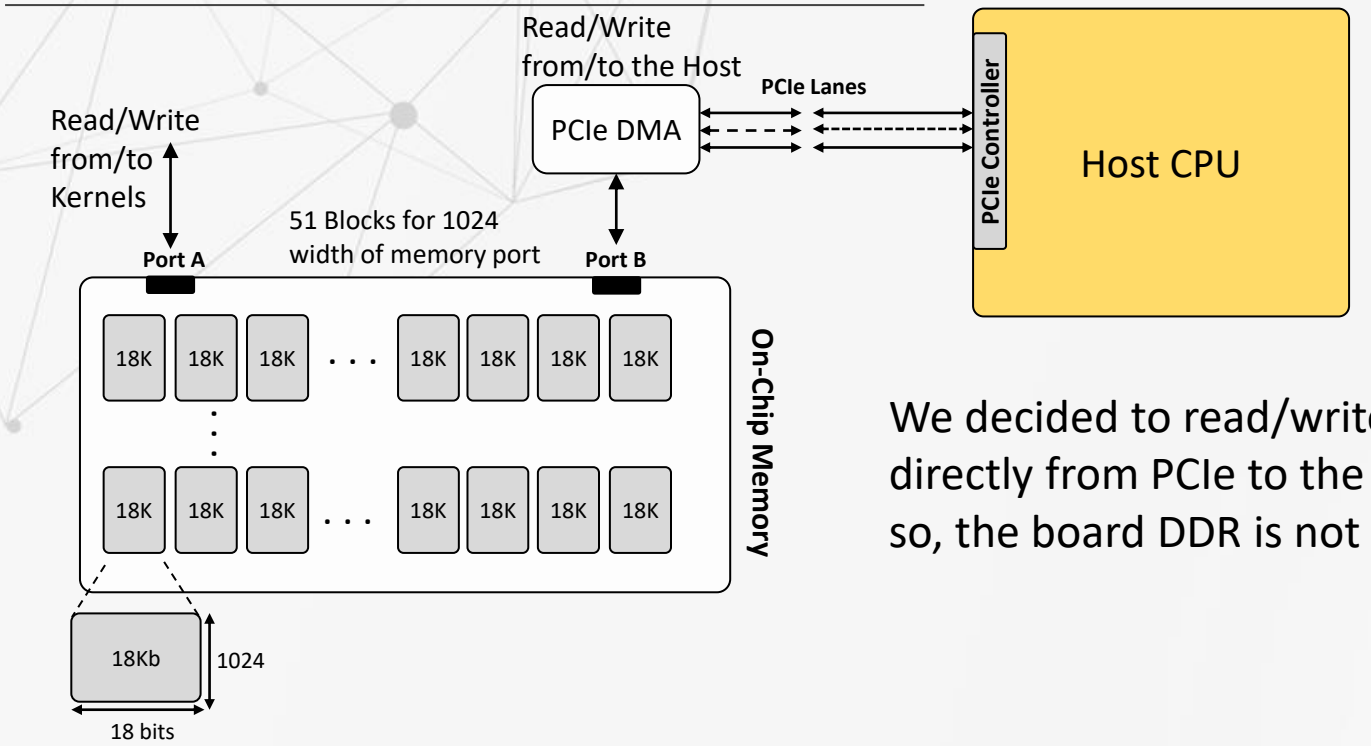
51 Blocks for 1024
width of memory port



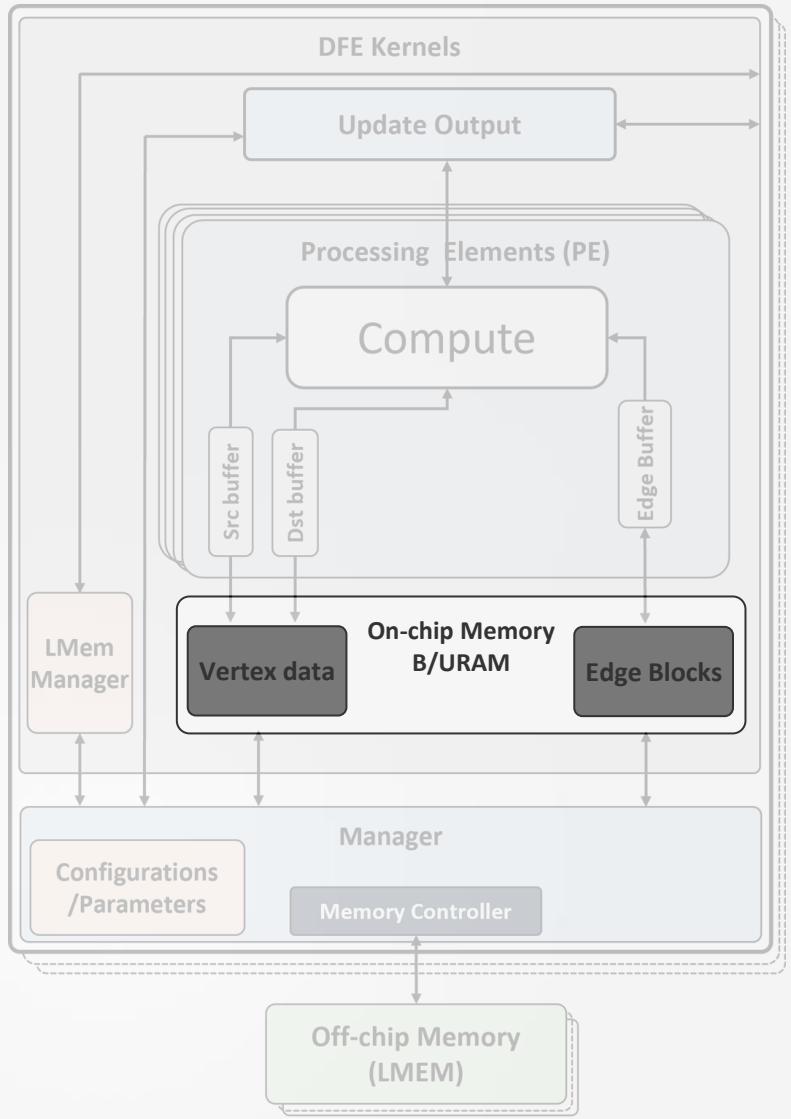


Based on Xilinx Family FPGA BRAM configurations
https://docs.xilinx.com/v/u/en-US/pg078-axi-bram-ctrl

BRAM mapping (Block-RAM)



We decided to read/write directly from PCIe to the FPGA, so, the board DDR is not used



Lesson Learned: finding the optimal configuration to allocate memory on BRAM is a key

- With smaller chunks we need to allocate many memories and therefore more ports will be created, and eventually more hardware will be utilized
- With wider bit width and bigger chunk or allocated memory we have less utilization of resources but harder to meet the timing constraints

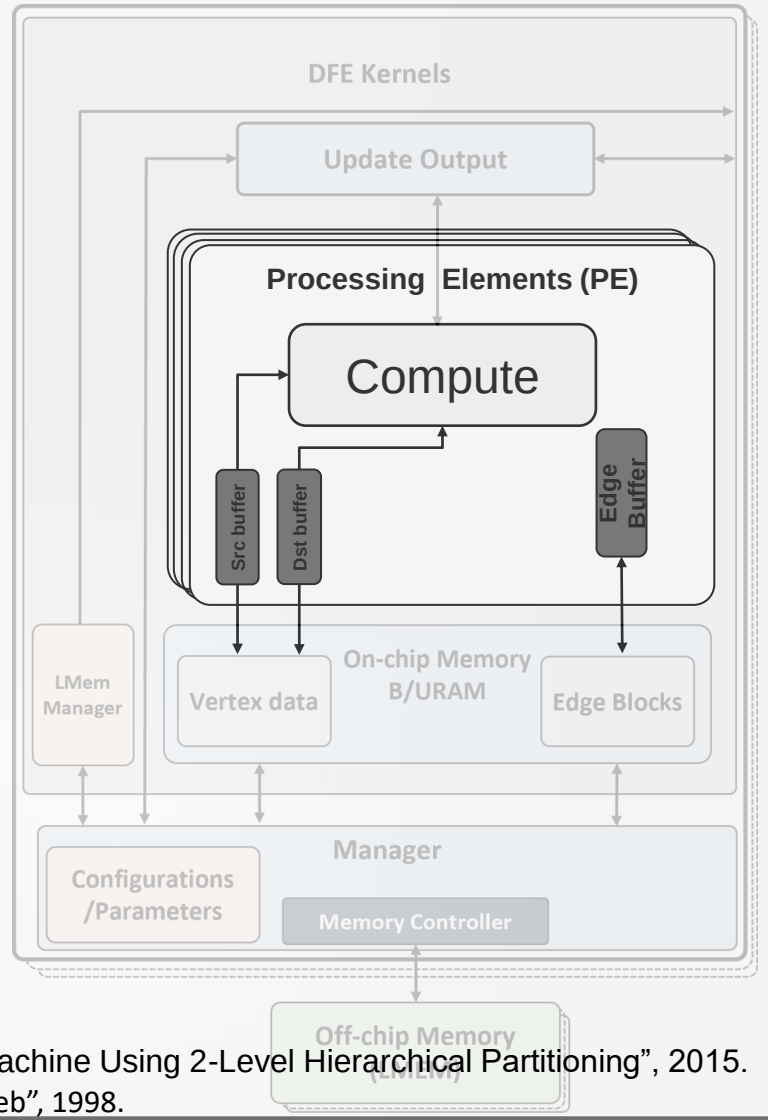
Mapping PageRank onto FPGA Processing Elements

Algorithm 2. PageRank algorithm with streaming vertexes and Grid Edge blocks

1 $G(V, E)$ ▷ Given Input Graph
2 $Pr = [1, \dots 1]$ ▷ PageRank array with size $|V|$
3 $d = 0.85$ ▷ Damping factor often consider 0.85
4 $converged = 0$

5 **While** $\neg Converged$ **do**
6 $NewPr = [0, \dots 0]$
7 **for each** $edge \in block$ **do**
8 $ADD\ (\&NewPr[edge.dest], \frac{Pr[edge.source]}{Deg[edge.source]})$
9 **end for**
10 **for each** $Vertex \in V$ **do**
11 $NewPr[v] \leftarrow 0.15 + 0.85 \times NewPr[v]$
21 **diff** = $|NewPr[v] - Pr[v]|$
13 **end for**
14 *Swap* ***(PR, NewPR)***
15 $Converged = \frac{Diff}{|v|} \leq \varepsilon$
16 **End While**

HLS Kernels and Functions



Xiaowei Zhu, Wentao Han, and Wenguang Chen. "GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning", 2015.

Larry Page and Sergey Brin and R. Motwani and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web", 1998.

The Evaluation Setup and the datasets

The XACC Xilinx server used to evaluate the real implementation

Instance Name	CPU	Freq	No. of Cores	Memory	Hardware Accl.
alveo2a.ethz.ch	Intel® Xeon® Gold 6234	2.50 GHz	16	128 GiB	Alveo U250

The Alveo U250 resource utilization in this experiment

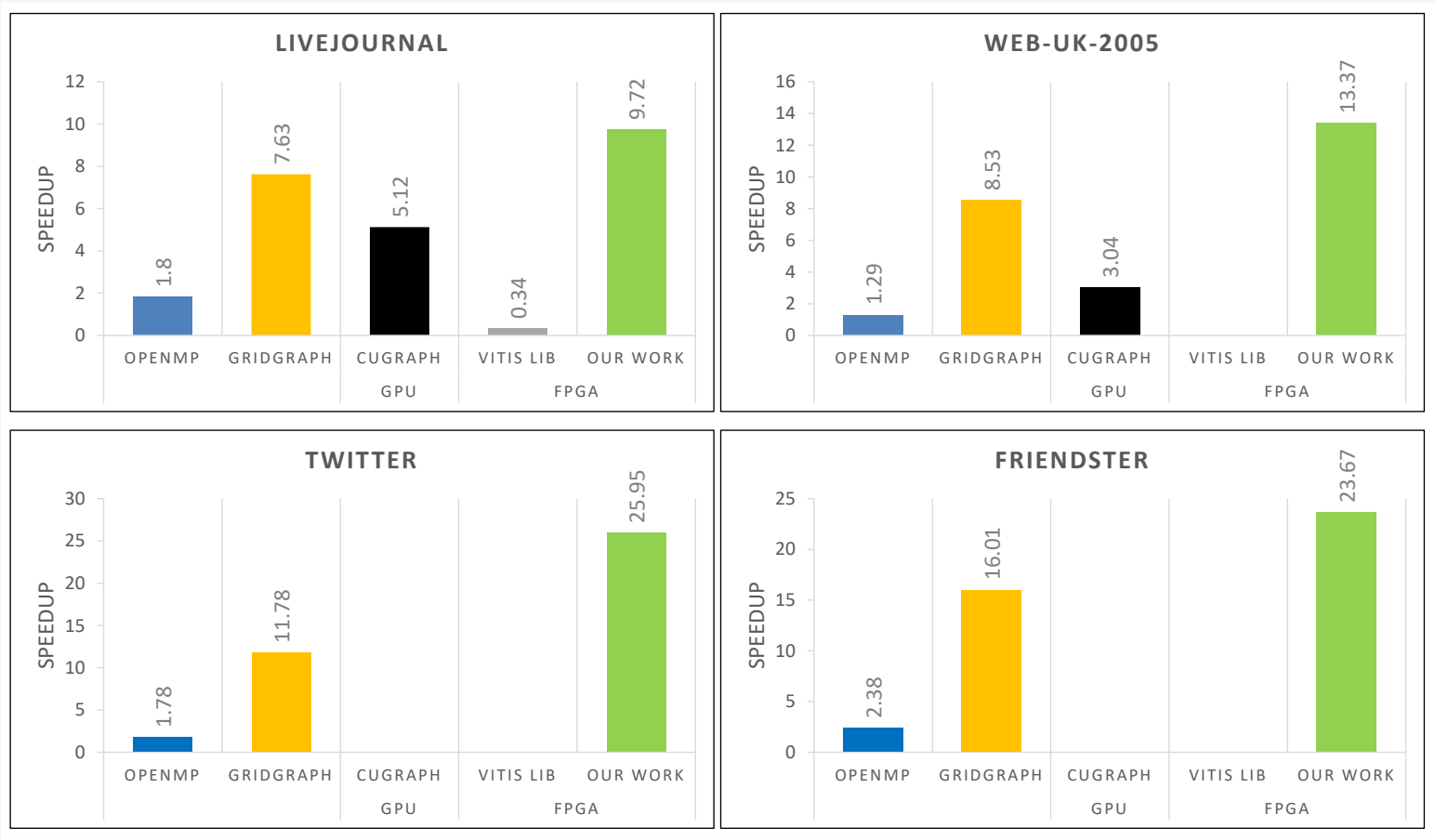
Resources	CLBs	BRAM	URAM	DSP	Power
Available	215777	2688	1280	12280	-
Used	55771 (25.85%)	580 (21.60%)	0	4	23.47 (w)

Dataset of the experiments

Graph dataset	Vertices	Edges	Size (GB)	Type
LiveJournal [12]	4.8M	0.069 B	1.1	Social Web
Web-UK-2005 [6]	39M	0.994 B	16	Web Graph
Twitter [11]	41.6 M	1.47 B	23	Web Graph
Friendster [12]	68.3M	2.58 B	43	Social Web

The Evaluation Results

- Speedup evaluation of a **single FPGA** implementation against sequential CPU execution
- **CPU comparison** with OpenMP and GridGraph
- **GPU comparison** with cuGraph
- **FPGA comparison** with the PageRank algorithm available in the AMD/Xilinx Vitis Library



[1] Sahebi, A., Barbone, M., Procaccini, M., Luk, W., Gaydadjiev, G., & Giorgi, R. (2023). Distributed large-scale graph processing on FPGAs. *Journal of Big Data*, *10*(1), 1-28.

Hadoop Performance Model

- The Hadoop framework has been extensively studied in the literature, and its performance is modelled and predictable [1,2]

- We forecast the performance by leveraging performance model already available [1,2] due to the lack of an Hadoop cluster equipped with FPGAs

$$T = T_{split-input} + T_{overhead} + N(T_{scatter} + max(T_{map}) + T_{transfer} + T_{reduce}) \quad (1)$$

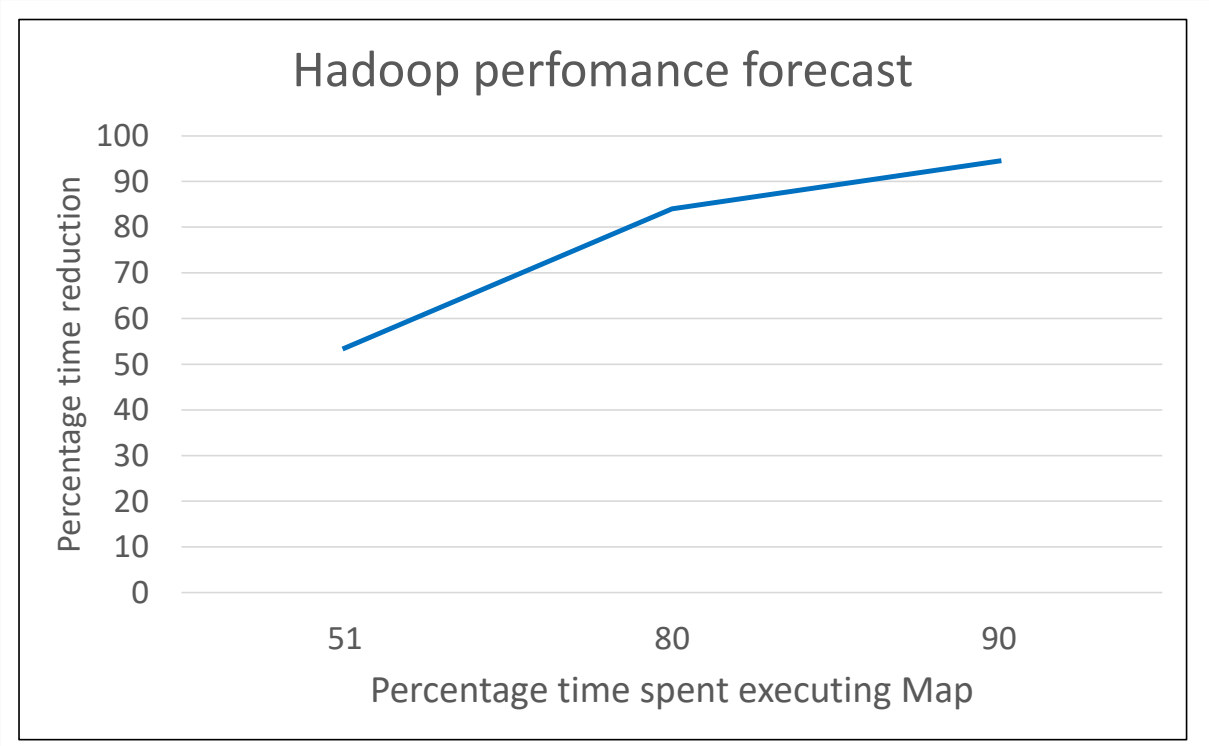
where:

- $T_{split-input}$ time needed to partition the graph in sub-graphs;
- N is the number of iterations of the iterative algorithm;
- $T_{overhead}$ is the overhead introduced by map-reduce;
- $T_{scatter}$ is the time needed to distribute the state vector to all the workers;
- $max(T_{map})$ is the time needed by the slowest mapper;
- $T_{transfer}$ is the time needed to transfer the state vector to the reducers (account for shuffle and sort);
- T_{reduce} is the time needed to aggregate the partial state vectors.

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
[2] Karloff H, Suri S, Vassilvitskii S. A model of computation for mapreduce. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms. SIAM, ACM: Austin; 2010. p. 938–948.

Hadoop Performance Model Evaluation

- Assumption that most of the time is spent in the mapping phase
- The performance model has been tuned with the results obtained in the single FPGA evaluation



Lesson Learned and Future Direction

1) Distributed computation across multicore+multifpga clusters is a promising platform to process large-scale graphs
2) Appropriate partitioning and block-scheduling may be needed to achieve a better load balance, mostly when unbalanced graphs are used (i.e., “natural graphs”)
3) Memory access patterns to process large-graphs may easily downgrade the performance (i.e., cache pollution)
4) The Host-FPGA communication overhead must be deeply evaluated to find further improvements.
5) The parallelism of the Hardware Model can be further improved
6) The model must be evaluated with a more comprehensive benchmark set

Please refer to this paper regarding this work: https://doi.org/10.1186/s40537-023-00756-x

Sahebi, A., Barbone, M., Procaccini, M., Luk, W., Gaydadjiev, G., & Giorgi, R. (2023). Distributed large-scale graph processing on FPGAs. *Journal of Big Data*, *10*(1), 1-28.

Thank you for you attention! ☺