

UNIVERSITÀ
DI SIENA 1240

HashGrid: Optimized FPGA Architecture for Accelerating Graph Processing in Large-Scale Computing

Amin Sahebi, Marco Procaccini, Roberto Giorgi

University of Siena, Italy

HiPEAC Workshop on Reconfigurable Computing (WRC'2025)
January 21st, 2025, Barcelona, Spain

TABLE OF CONTENTS

01 **Brief Introduction to Graphs**
Here we intoruduce some facts about Graph computing

02 **Motivation**
Here we motivate or work and our intention to our work

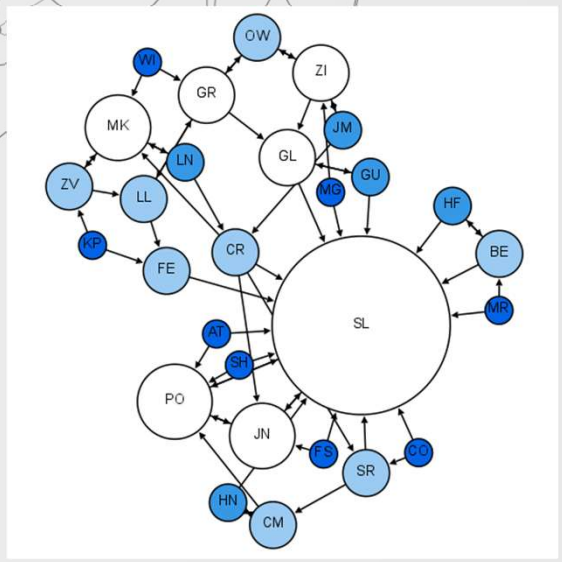
03 **HashGrid Partitioning**
Here we discuss the chosen partitioning method

04 **HashGrid Architecture**
inegration

05 **Experiments**

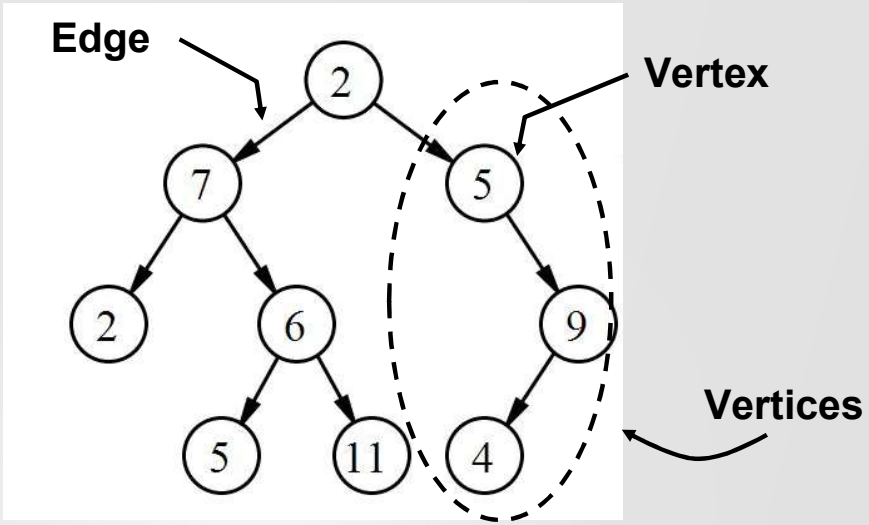
06 **GPP/GPU/FPGA comparison**
Here we show Where are we and what is the progress status

Graphs: a common way to represent information



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

G(V,E)



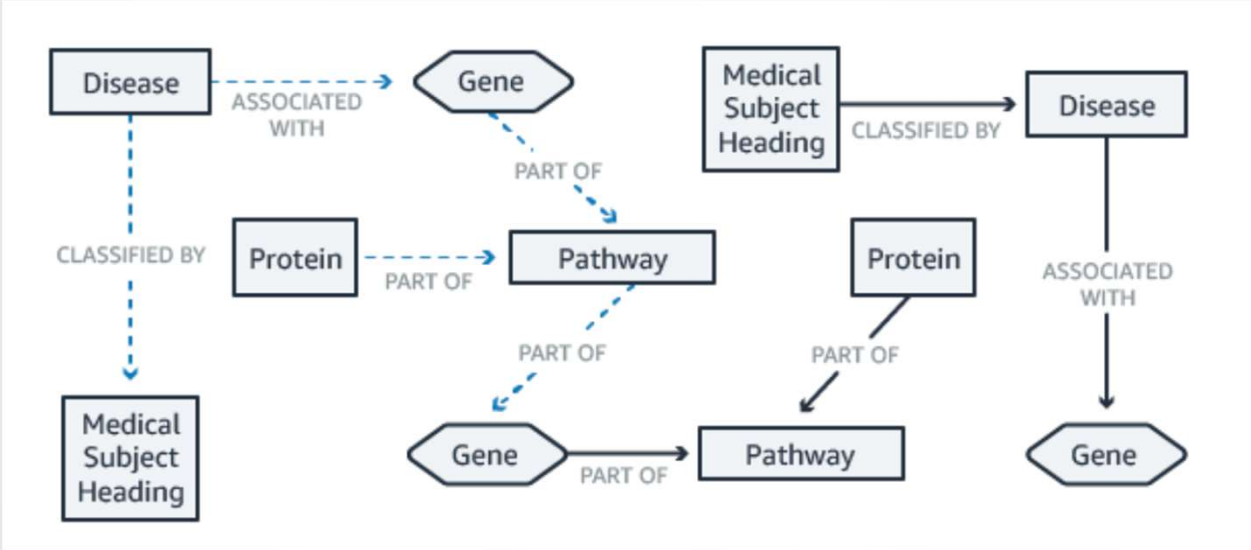
All the ecommerce space on the internet is driven by graphs.
Such as **web** or **social networks**, **computational problems**, **analytics**, **machine learning, etc.**

Why processing graphs



UNIVERSITÀ DI SIENA 1240

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

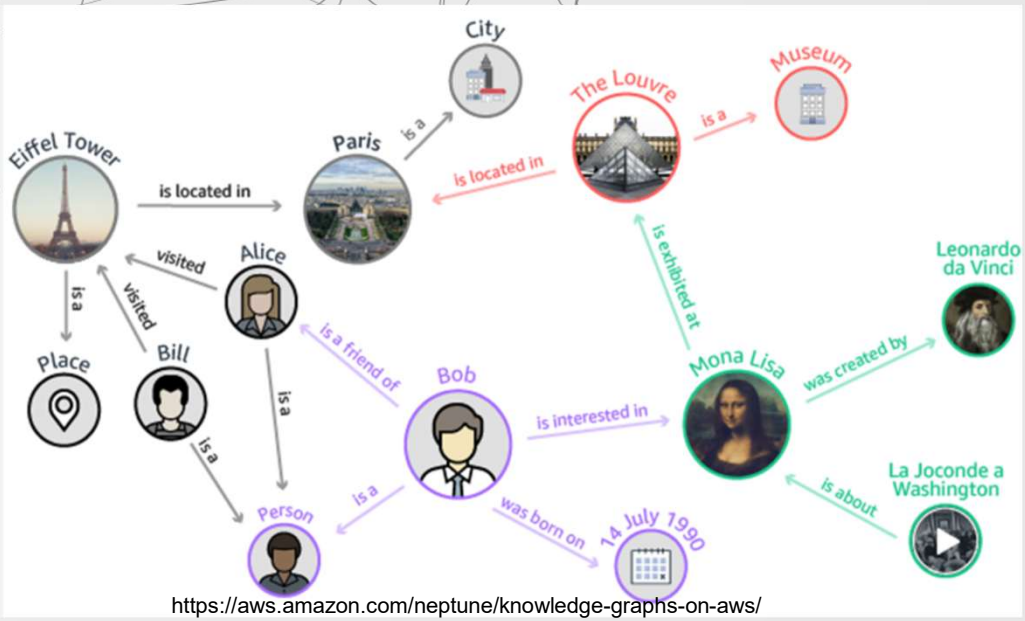


- Analyzing data to get more knowledge
- Improve the process efficiency
- Get insights/recommendations with machine learning and reducing the overall cost

Examples of large graphs

UNIVERSITÀ DI SIENA 1240

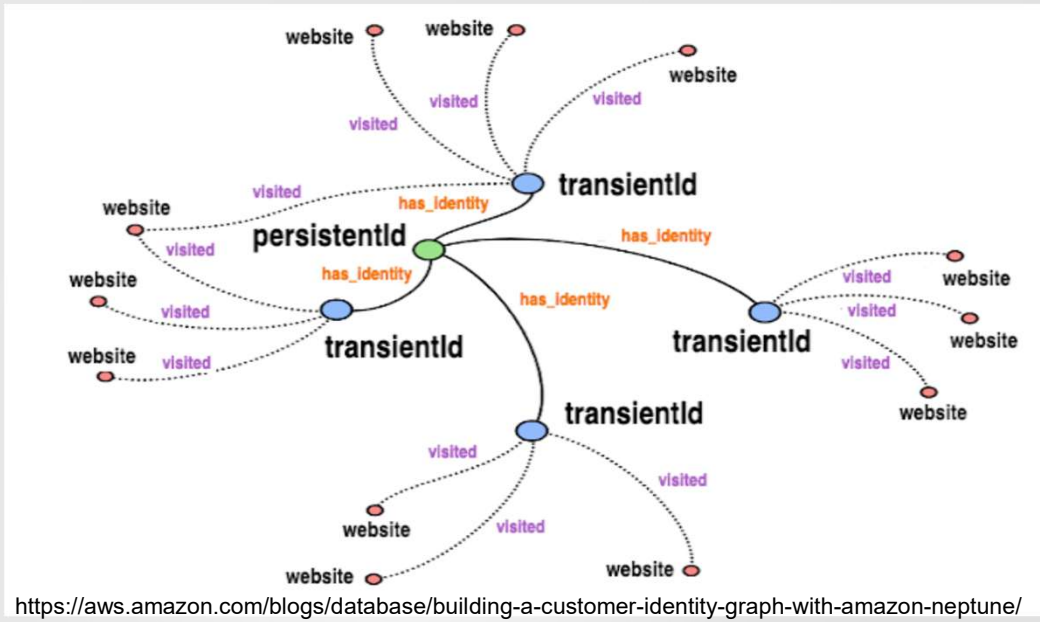
Knowledge Graphs



https://aws.amazon.com/neptune/knowledge-graphs-on-aws/

A knowledge graph is a knowledge base that uses a graph-structured data model or topology to integrate data, mostly used with search engines, and question-answer engines like Amazon Alexa

Identity Graphs



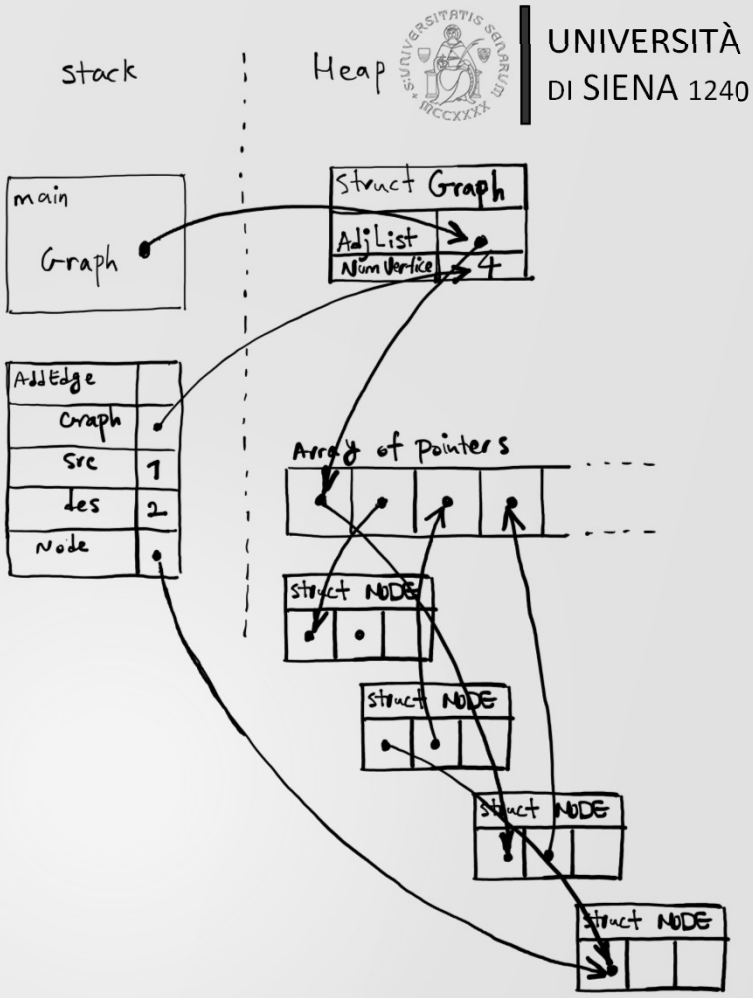
https://aws.amazon.com/blogs/database/building-a-customer-identity-graph-with-amazon-neptune/

An identity graph, or ID graph, is a collection of known customer identifiers that can be associated with one another.

What are the Challenges?

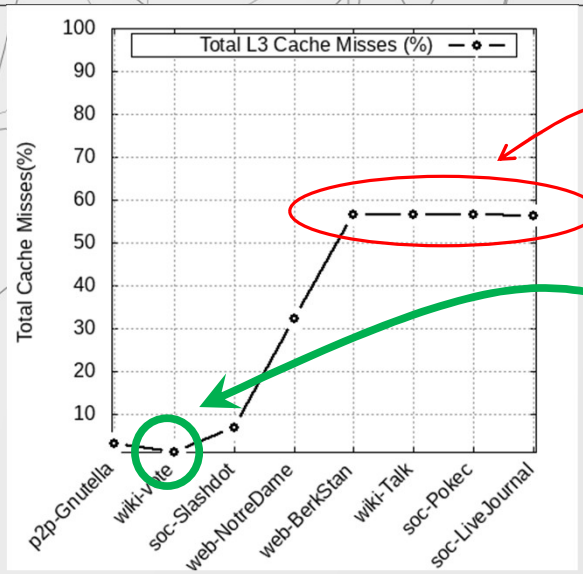
- Graphs are getting bigger and bigger every day!
  - Large-scale graphs size is in the order of GB/TB
  - Scalability:
    ability to adapt to larger and larger graphs

- Computing graphs in an efficient way
  - The irregular structure of the graph usually leads
    to a high random memory access
    which is the weak point
    of the CPU and GPU memory hierarchy
  - Scarce locality:
    data accessed randomly throughout the memory



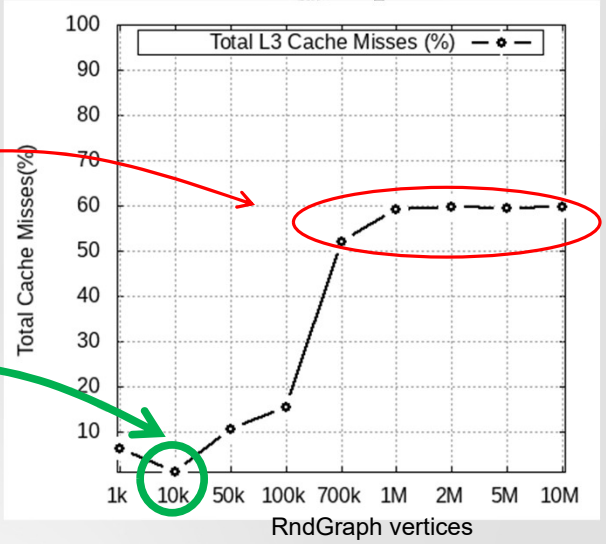
Cache performance for large and sparse graphs

UNIVERSITÀ DI SIENA 1240



When the graphs size exceeds L2 size (for sparse graphs) or L3 cache for real world graphs
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

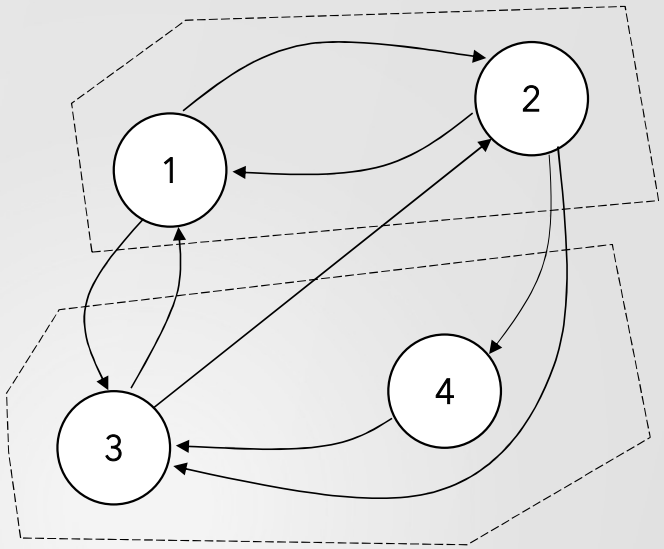
Synthetic database	V	E	Size on Disk
RndG-1K	1000	10,000	76 KB
RndG-10k	10K	100K	1 MB
RndG-500k	50K	500K	5.6 MB
RndG-100k	100K	1M	12 MB
RndG-700k	700K	7M	92M
RndG-1M	1M	10M	132 MB
RndG-2M	2M	20M	284 MB
RndG-5M	5M	50M	760 MB
RndG-10M	10M	100M	1.5 GB

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



Partitioning the graph

UNIVERSITÀ DI SIENA 1240

Different stages for General Graph Computing Experiment
And the best candidates for some applications

Pre-processing

- ✓How to represent the graph? → ○ Adjacency lists ○ Edge arrays
- ✓Cost of creating the representation? → ○ Radix sort wins for adjacency lists
- ✓What data layout is best? → ○ BFS: Adj. list ○ PR: Grid ○ SpMV: Edge array

Algorithms

- ✓Can we improve cache locality? → ○ Yes. By laying out the edges in a grid format ○ BFS: Adj. list ○ PR: Grid ○ SpMV: Edge array
- ✓Should we optimize for NUMA? → ○ Can pay off only on big machines ○ BFS & SpMV: No gain ○ PR: NUMA-optimize
- ✓Information flow: **push**, **pull** or a **both**? → ○ Less synchronization not always a win ○ BFS: Push (locks) ○ PR: Pull (no locks) ○ Push/Pull no win in end-to-end (directed graphs)

70

Turns out that "**Grid**" is the best solution for partition the Graph (at least for PageRank)

https://github.com/epfllabos/EverythingGraph

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.

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.

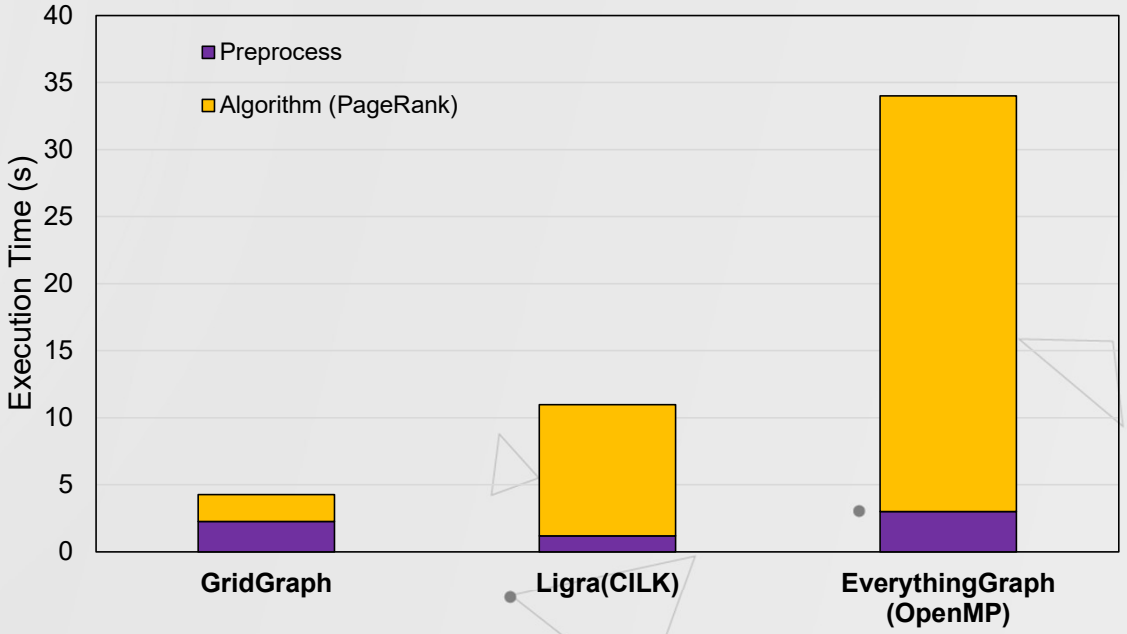
2) Jasmina Malicevic, Baptiste Lepers, and Willy Zwaenepoel. 2017. Everything you always wanted to know about multicore graph processing but were afraid to ask. In Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17). USENIX Association, USA, 631–643.

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)

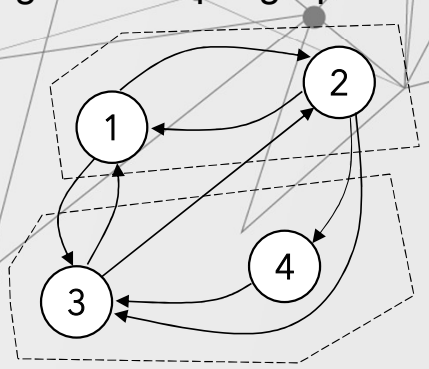


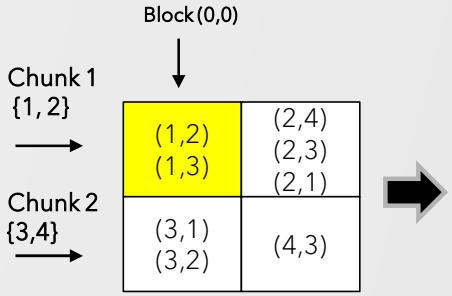
We have to measure end to end time.

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

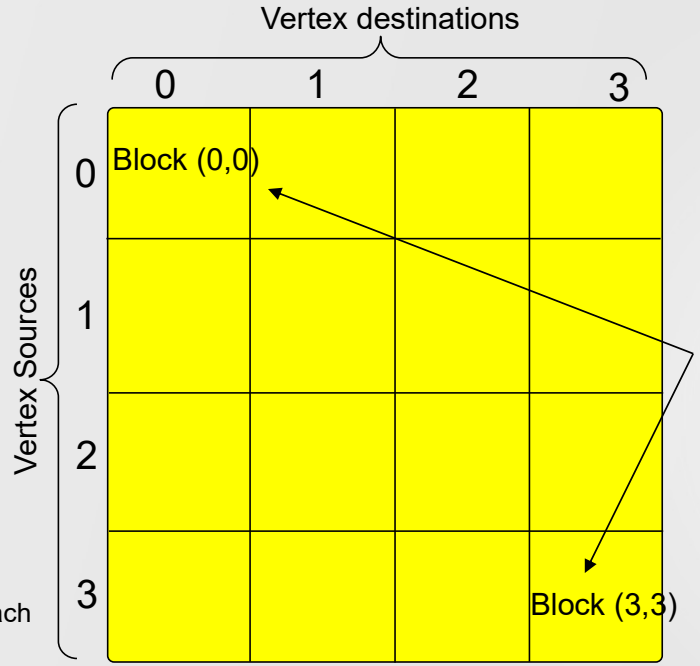
Preprocessing Step: GridGraph Partitioning Method

A given sample graph



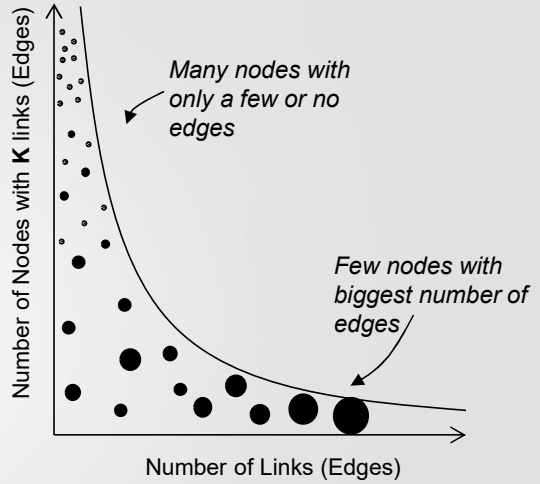


Partitioning based on GridGraph* approach



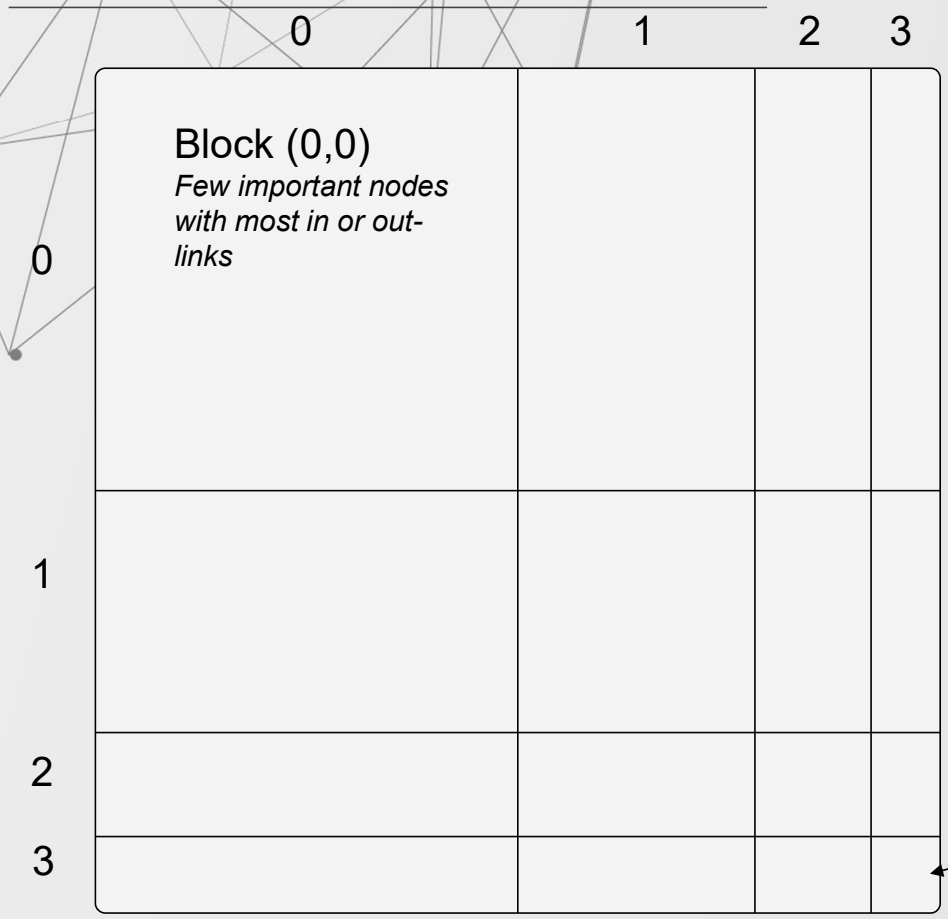
Data chuncks are almost with same size all over the graph dataset

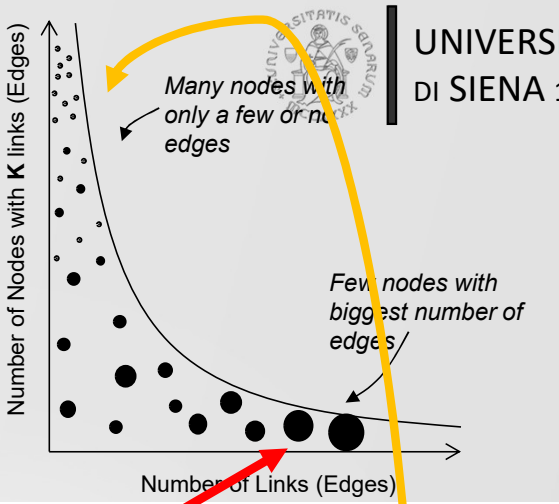
The challenge of partitioning is Real-world Powerlaw graphs which produces uneven block sizes of edge data

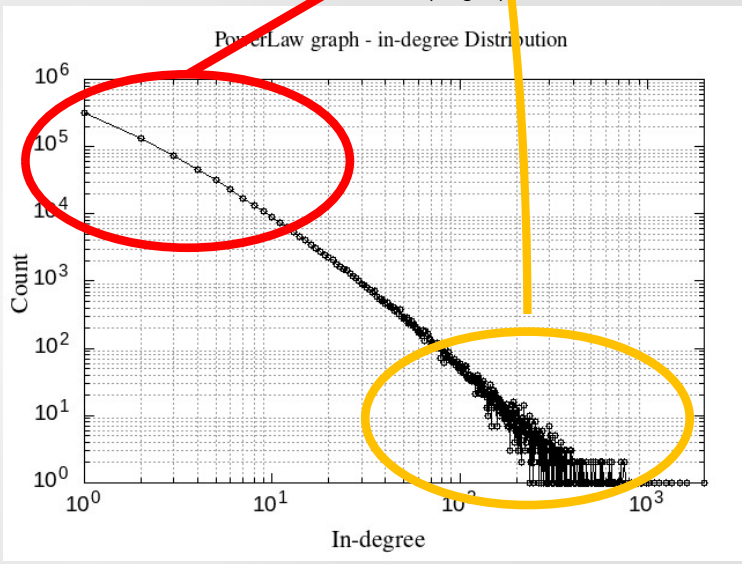


* Xiaowei Zhu, Wentao Han, and Wenguang Chen. "GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning". USENIX ATC '15. Santa Clara,

Unbalanced nodes in real-world graphs







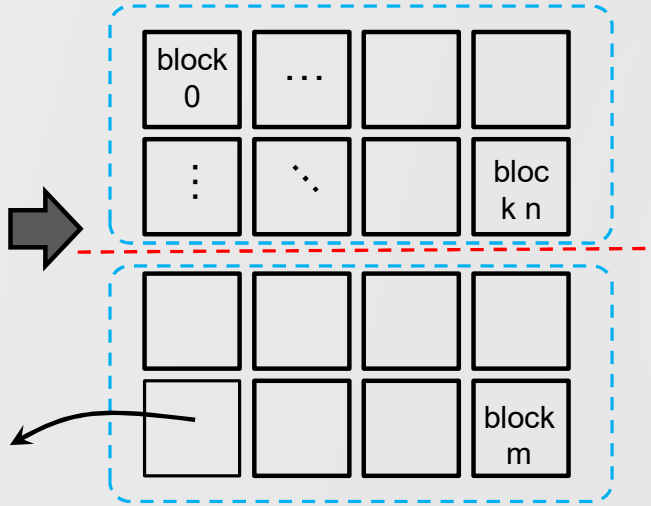
UNIVERSITÀ DI SIENA 1240

Sub-blocks partitioning (hierarchical partitioning)

UNIVERSITÀ DI SIENA 1240

- 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

Partitioning Large Scale Graph in order of hundreds of GB or some TB into blocks



The access in each block is *Random*.

Dedicate them to the specific resource

From block-0-0 to block-k-n will be dedicated on the First FPGA

FPGA 1

With the same order the next blocks will be dedicated to the next available resources

FPGA 2

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	Pooerly 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	

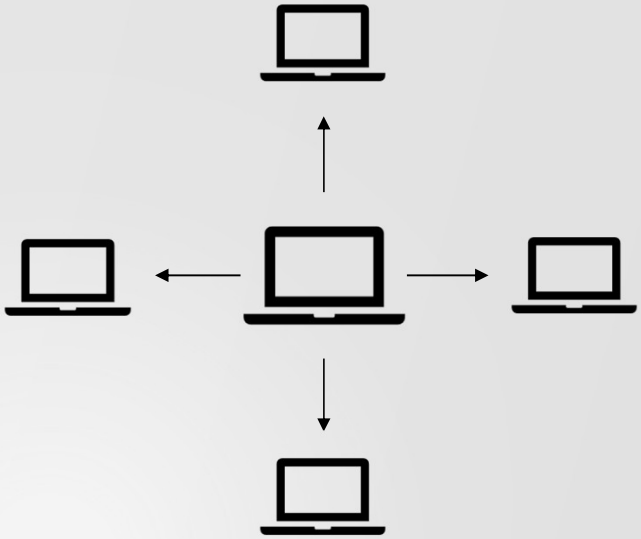
FPGA can be the optimal candidate because of many good features that can overcome CPU and GPU

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

Distributed solutions?

UNIVERSITÀ DI SIENA 1240

- 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

Why distributed FPGA?

UNIVERSITÀ DI SIENA 1240

Advantages of a distributed FPGA architecture:

- Most efficient high-performance computing platform
- Cloud-computing support
- Pricing for instances (CPU and FPGA)

Disadvantage of using FPGA on distributed platform:

- Development and implementation may take more time than others

For instance, targeting amazon AWS platform:
Assuming, FPGA 10x faster than CPU

→ **10x instance of m6gd.16xlarge it is ~29$/hr**
→ **1 instance of f1.16xlarge is ~13$/hr**

Name	FPGAs	vCPUs	Instance Memory (GiB)	SSD Storage (GB)	On-Demand (Price/hr)
f1.2xlarge	1	8	122	470	$1.65
f1.4xlarge	2	16	244	940	$3.30
f1.16xlarge	8	64	976	4 x 940	$13.20

https://aws.amazon.com/ec2/instance-types/f1/ - June 2022

	vCPU	Memory	On-Demand Price/hr
m6gd.xlarge	4	16 GiB	$0.1808
m6gd.2xlarge	8	32 GiB	$0.3616
m6gd.4xlarge	16	64 GiB	$0.7232
m6gd.8xlarge	32	128 GiB	$1.4464
m6gd.12xlarge	48	192 GiB	$2.1696
m6gd.16xlarge	64	256 GiB	$2.8928

https://aws.amazon.com/ec2/pricing/on-demand/ - June 2022

State-of-the-art

	HitGraph [1]	GridGraph [2]	GraphOps [3]	ForeGraph [4]	Xstream [5]
Locality	High	High	Low	High	Low
Scalability	Poor	Medium	Poor	Medium	Poor
Data Access Pattern	Edge Centric	Grid	Reading Neighbor's Vertex	Interval Shards	Edge Centric
Data Conflict	Low	Low	High	High	Low
Implementation Language	HDL	C/C++	MAXJ	HDL	C/C++

[1] S. Zhou, et al, "**HitGraph: High-throughput Graph Processing Framework on FPGA**," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249-2264, 1 Oct. 2019.

[2] 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.

[3] 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). Association for Computing Machinery, New York, NY, USA, 111–117.

[4] Guohao Dai, Tianhao Huang, Yuze Chi, Ningyi Xu, Yu Wang, and Huazhong Yang. 2017. **ForeGraph: Exploring Large-scale Graph Processing on Multi-FPGA Architecture.** In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17).

[5] Amitabha Roy et al**. X-Stream: edge-centric graph processing using streaming partitions**. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). Association for Computing Machinery, New York, NY, USA, 472–488.

Graph processing evaluation algorithms

Which Application/Algorithm can be used to evaluate fairly FPGA, CPU or GPU?

Graph processing algorithms and their application usage.

Algorithm	Purpose	Key Idea	Example Usage
PageRank	Ranking Importance	Determine the importance of nodes in a graph based on their connections to other nodes.	Web search ranking, social influence analysis
SSSP (Single-Source Shortest Path)	Finding Shortest Paths	Find the shortest path from a single-source node to all other nodes in the graph.	Social network distance analysis, Navigation, Network routing
WCC (Weakly Connected Components)	Identifying Components	Identify clusters of nodes where each node is reachable from any other node.	Social network analysis, Component identification in large systems

S. Zhou, et al, "**HitGraph: High-throughput Graph Processing Framework on FPGA**," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249-2264, 1 Oct. 2019.

A Taxonomy of the more similar related studies



UNIVERSITÀ DI SIENA 1240

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	Hardware	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	
Our prev. work [10]	Yes	HLS	Hardware	Yes	Very Large	Yes	Alveo Family	2022	GITHUB public repo
This work [11]	No	HLS	Hardware	Yes	Very large	Yes	Alveo Family	2025	GITHUB public repo

*References in the next page

Related studies



UNIVERSITÀ
DI SIENA 1240

[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

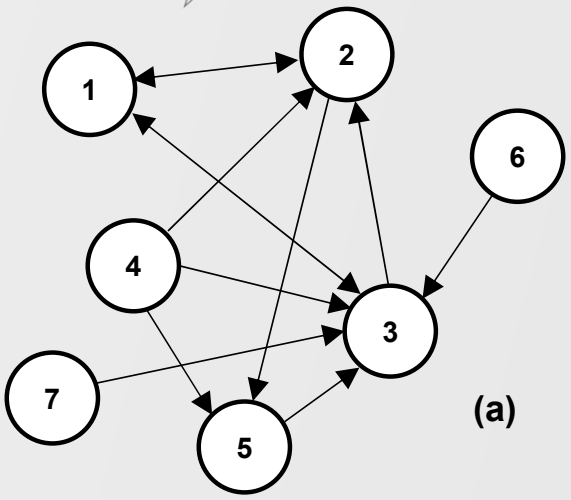
[10] A. Sahebi, M. Barbone, M. Procaccini, W. Luk, G. Gaydadjiev, R. Giorgi, "Distributed large-scale graph processing on FPGAs", Springer Journal of Big Data, vol. 10, no. 1, Jun 2023, pp. 28. doi:10.1186/s40537-023-00756-x

[11] Amin Sahebi and Marco Procaccini and Roberto Giorgi, "HashGrid: An optimized architecture for accelerating graph computing on FPGAs", Future Generation Computer Systems, vol. 162, 2025, pp. 107497, doi:10.1016/j.future.2024.107497

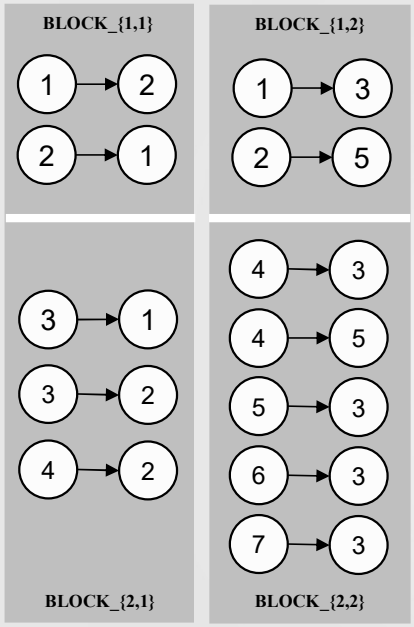
Proposed Solution: HashGrid partitioning

UNIVERSITÀ DI SIENA 1240

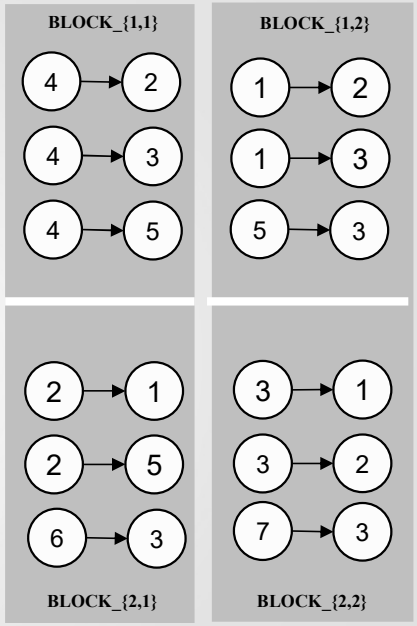
- - Enhanced GridGraph partitioning with a hash function
- - Uniform distribution of graph edges to minimize bottlenecks



(a)



(b)



(c)

Mathematically…

$$e_{s,d} \in B_{i,j} \Leftrightarrow \quad h_{i,j}(s,d) = \begin{cases} i = h_i(s,d) = (s \bmod P^2) \;/\; P \\ j = h_j(s,d) = (s \bmod P^2) \bmod P \end{cases} \tag{1}$$

Where,

- $e_{s,d}$ is the edge having source vertex s and destination vertex d.
- $B_{i,j}$ is the block at the ith row and jth column of the grid.
- $h_{i,j}$ is the hash function to generate i and j.
- P is the number of partitions.

Datasets

The Real-World graph datasets used to evaluate our proposed framework. The datasets are represented by the number of edges (expressed in millions (M)) and vertices (expressed in billions (B)). Additionally, the average degree (d_{avg}) and the maximum degree (d_{max}) are provided. The size of the graphs is expressed in Gigabytes (GB).

Dataset	Vertices	Edges	d_{avg}	d_{max}	Size (GB)	Type
LiveJournal [38]	4.8 M	0.069 B	17.89	20.3k	1.1	Social Web
UK-2005 [39]	39.4 M	0.936 B	23.72	1,776k	16	Web Graph
TW-2010 [40]	41.6 M	1.47 B	70.5	3,081k	23	Web Graph
Friendster [40]	68.3 M	1.81 B	75.6	7.3k	43	Social Web

Edge distribution across partition blocks

UNIVERSITÀ DI SIENA 1240

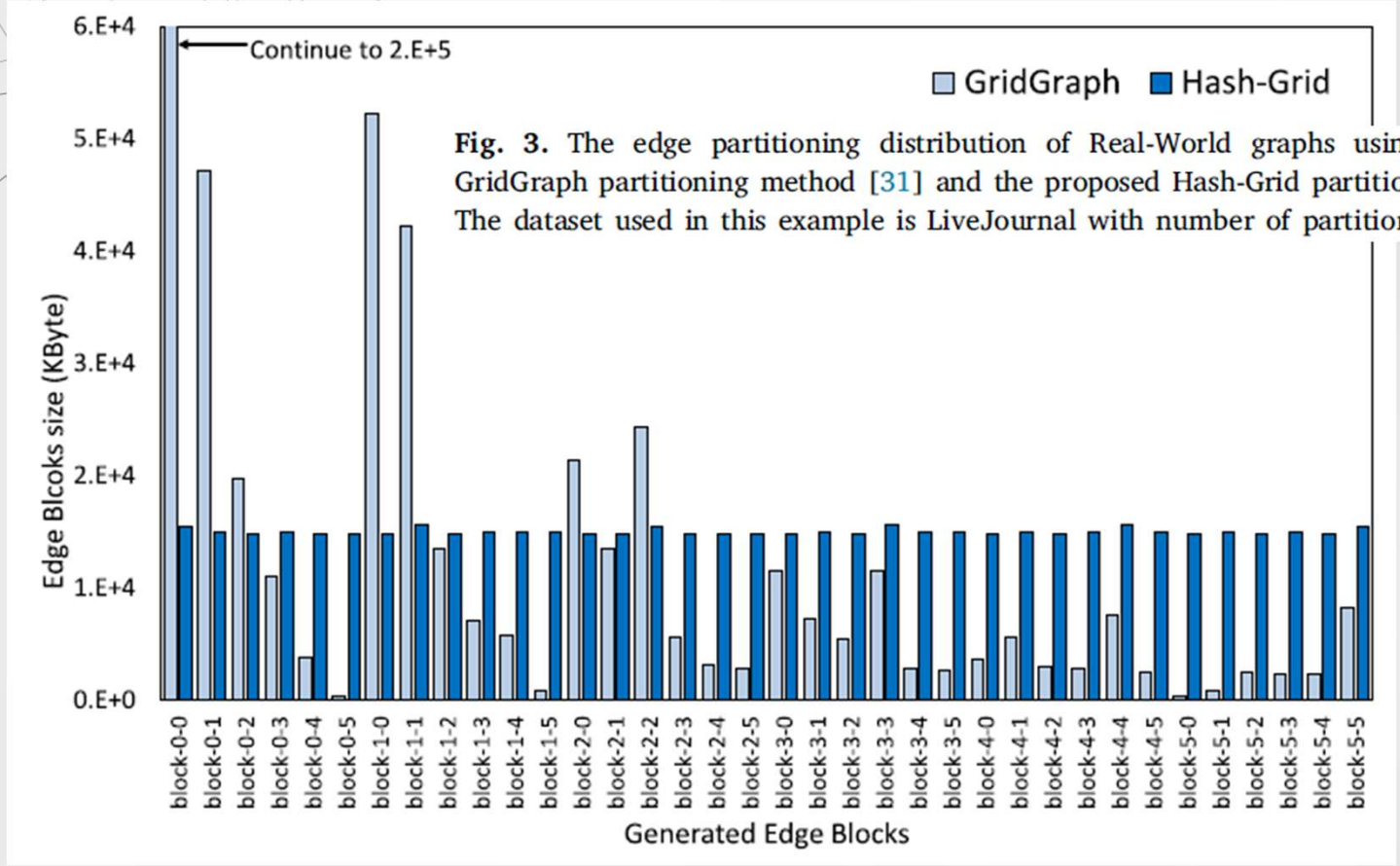


Fig. 3. The edge partitioning distribution of Real-World graphs using the native GridGraph partitioning method [31] and the proposed Hash-Grid partitioning method. The dataset used in this example is LiveJournal with number of partitions equal to 6.

Partitioning time comparison

Comparison of the execution time (measured in seconds) for the partitioning step between the GridGraph and Hash-Grid methods.

	GridGraph			Hash-Grid		
	$P = 20$	$P = 40$	$P = 60$	$P = 20$	$P = 40$	$P = 60$
LJ	1.06	**1.23**	**1.58**	**0.9**	1.33	2.02
UK-2005	12.11	14.69	15.5	**11.23**	**11.26**	**11.8**
TW-2010	10.18	11.68	11.82	**7.67**	**8.9**	**9.36**
FS	28.08	30.48	31.2	**25.58**	**29.88**	**30.1**

Changing the number of partitions: Throughput

UNIVERSITÀ DI SIENA 1240

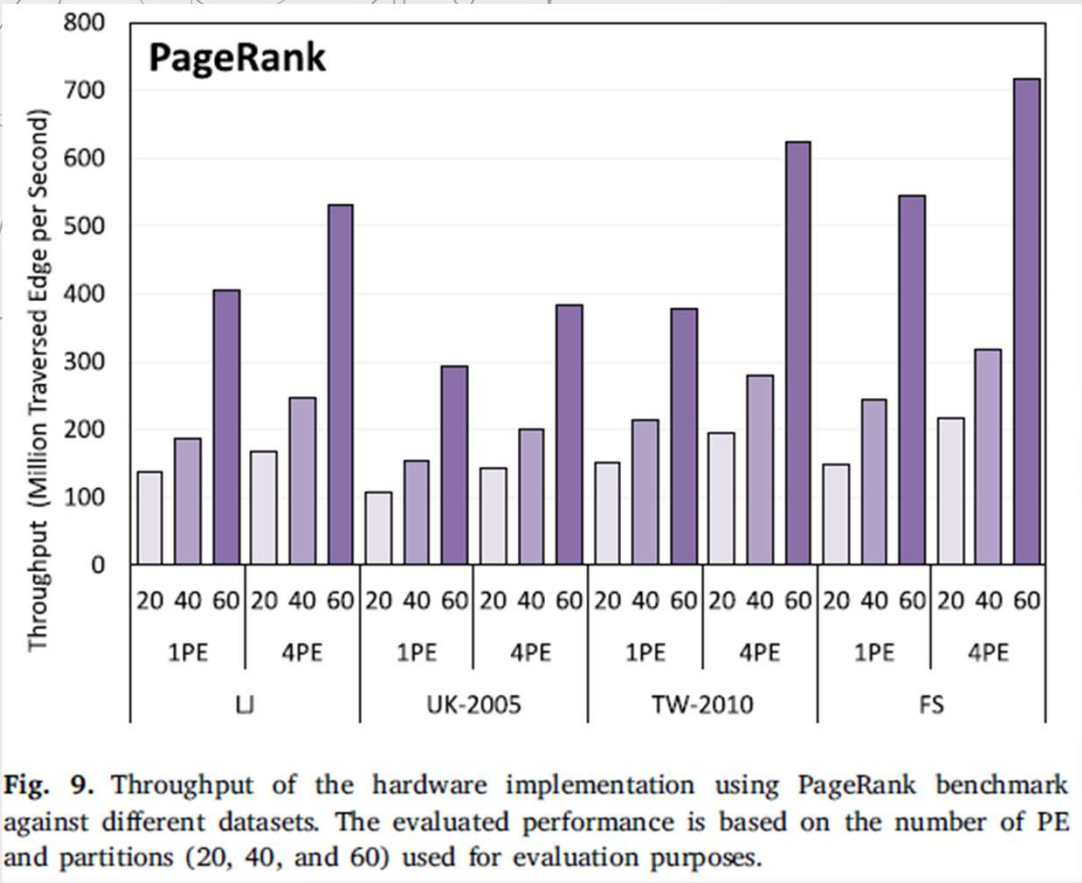
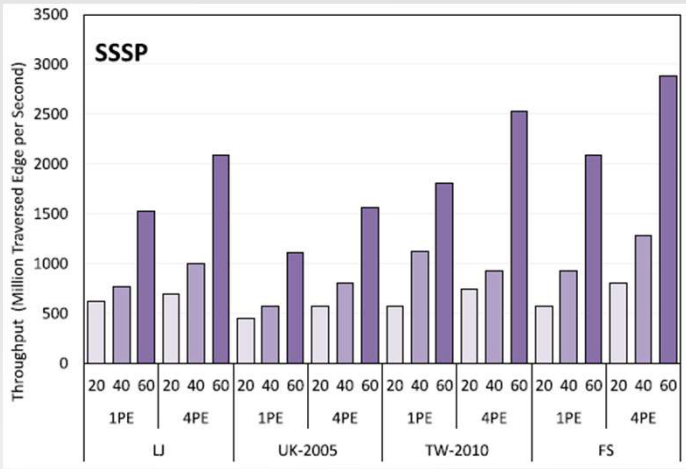
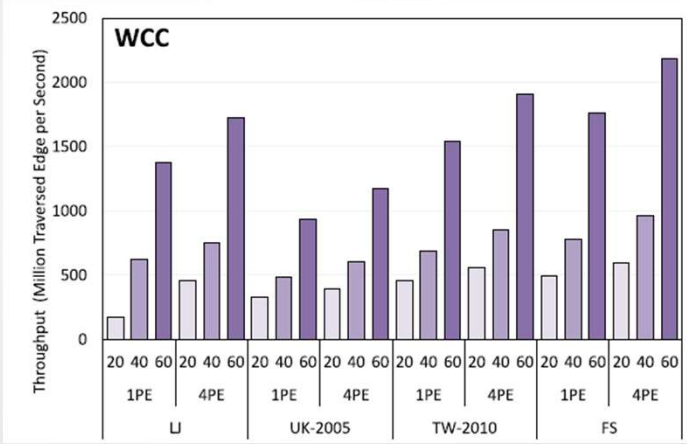


Fig. 9. Throughput of the hardware implementation using PageRank benchmark against different datasets. The evaluated performance is based on the number of PE and partitions (20, 40, and 60) used for evaluation purposes.





Changing the number of partitions: Execution Time

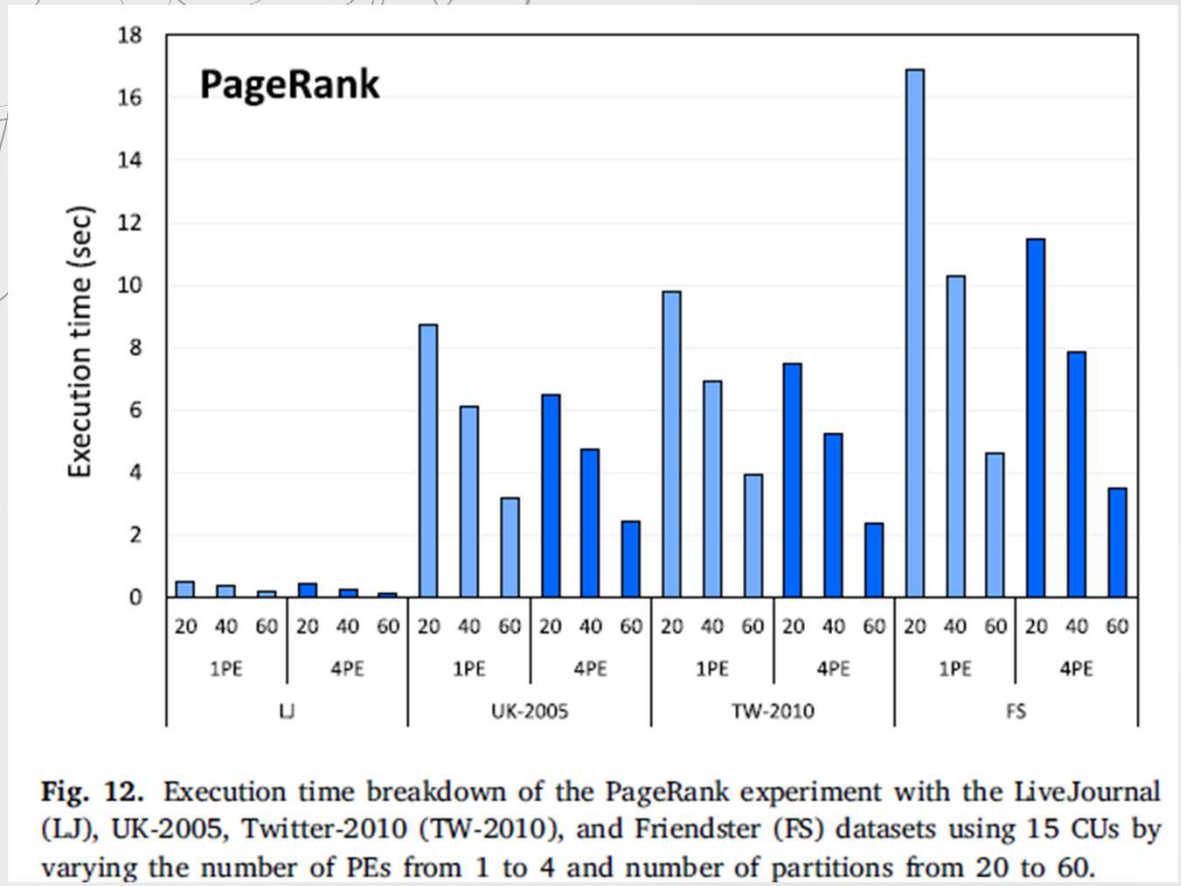
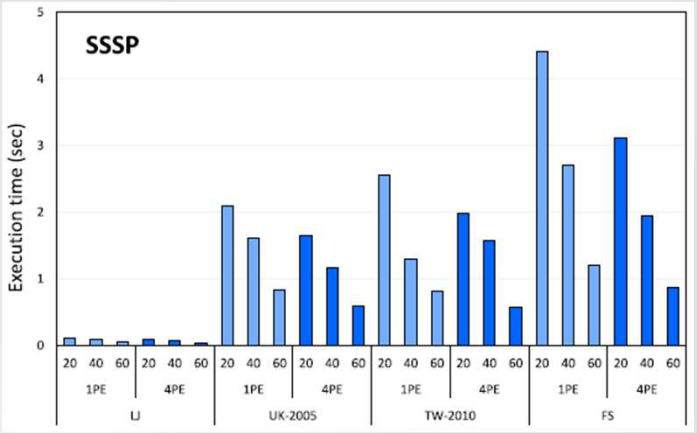
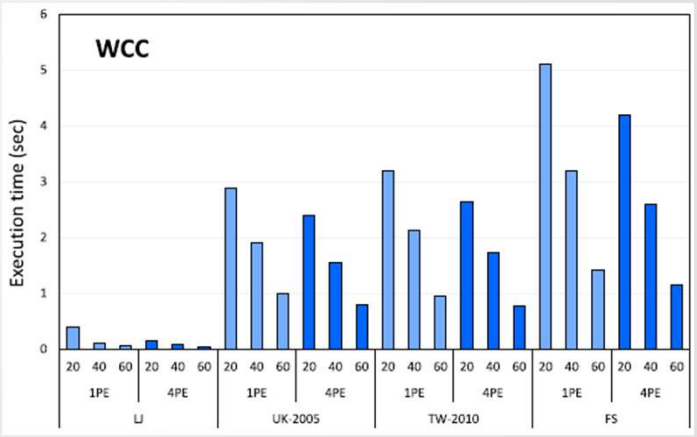


Fig. 12. Execution time breakdown of the PageRank experiment with the LiveJournal (LJ), UK-2005, Twitter-2010 (TW-2010), and Friendster (FS) datasets using 15 CUs by varying the number of PEs from 1 to 4 and number of partitions from 20 to 60.





CPU/GPU/FPGA platforms

Features comparison between CPU, GPU, and FPGA used in the evaluation.

Features	CPU Xeon Gold 6234	GPU RTX A5000	FPGA Alveo U250
Frequency	3.30 GHz	1695 MHz	300 MHz
Cores	8C/16T[b]	128SM[c]	8PE[d]
Peak Performance	0.11 TFLOPS	27.7 TFLOPS	0.6 TFLOPS
On-chip Memory	25 MB L3 Cache	6 MB L2 Cache	54 MB
Memory Bandwidth	140 GB/s	768 GB/s	77 GB/s
Power[a]	200 W	230 W	150 W

[a] More details about power consumption appears in Section 5.5.

[b] C = Cores; T = Threads

[c] SM = Streaming Multiprocessors

[d] PE = Processing Elements used in this work

CPU/GPU/FPGA: execution time

	PageRank				
	GridGraph (CPU)	cuGraph (GPU)	FPGA		
			HitGraph	ThunderGP	This Work
LJ	7.33	0.29	0.327	0.293	**0.13**
UK-2005	57.7	6.55	N/A[a]	N/A[d]	**2.45**
TW-2010	63.81	5.68	**0.590**	N/A[c]	2.35
FS	130.33	13.51	N/A[a]	N/A[c]	**3.5**

SSSP				
GridGraph (CPU)	cuGraph (GPU)	FPGA		
		HitGraph	ThunderGP	This Work
2.16	0.19	0.592	0.13	**0.033**
18.31	2.26	N/A[a]	N/A[d]	**0.6**
19.31	8.2	5.58	N/A[c]	**0.58**
39.45	22.95	N/A[a]	N/A[c]	**0.87**

WCC				
GridGraph (CPU)	cuGraph (GPU)	FPGA		
		HitGraph	ThunderGP	This work
2.75	0.296	0.412	0.31	**0.04**
322.41	N/A[b]	N/A[a]	N/A[d]	**0.8**
103.9	N/A[b]	6.61	N/A[c]	**0.77**
269.9	N/A[b]	N/A[a]	N/A[c]	**1.15**

HashGrid: the whole thing

UNIVERSITÀ DI SIENA 1240

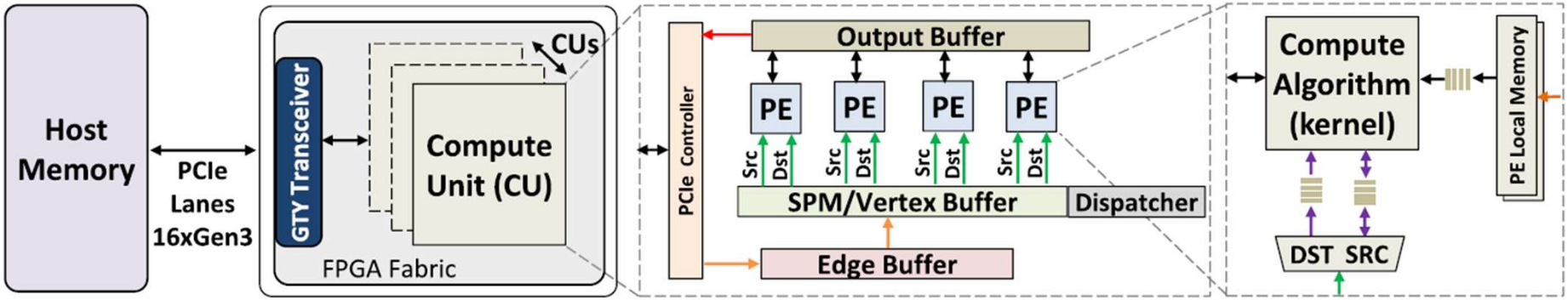
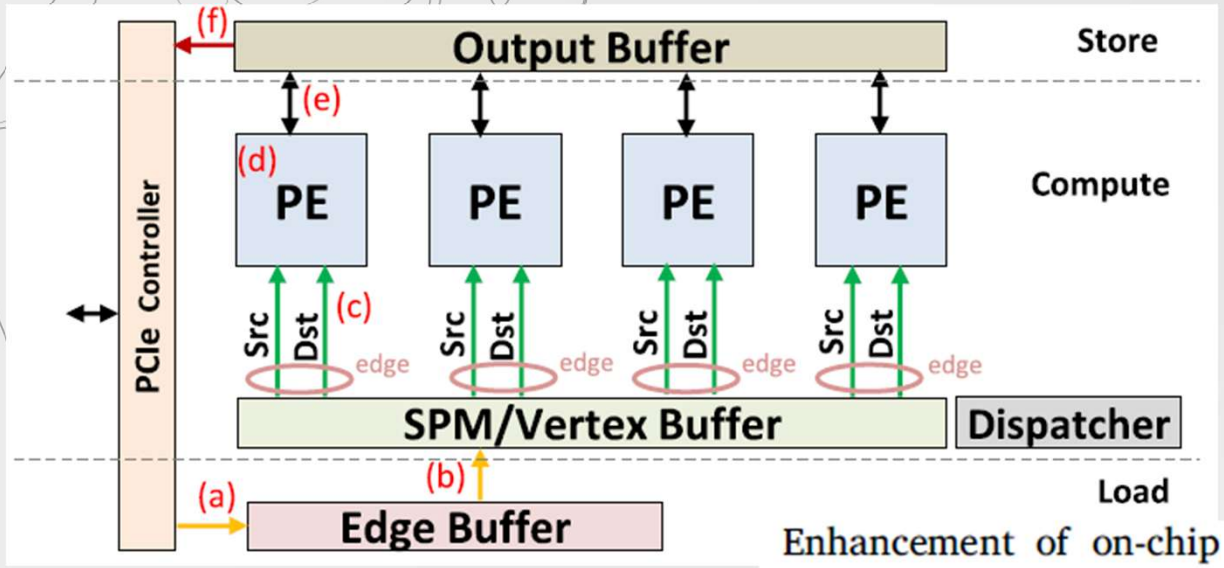


Fig. 4. An overview block diagram of the proposed hardware design for large-scale graph computing from the Host Memory to the FPGA computing. The edge blocks are distributed across Compute Units (CU) , which they communicate directly with the Host Memory through Gigabit Transceivers (GTY). The CUs utilize array of Processing Element (PE) to compute the algorithm (e.g., PageRank). Each PE can access the scratchpad memory to load the edges (Edge Buffer), the Source (SRC), Destination (DST) and the vertex (Vertex Buffer).

HashGrid: adding ScratchPad Memory (SPM) support



Enhancement of on-chip memory bandwidth utilizing the ScratchPad Memory (SPM) module, according to the data provided in Table 3 and Eq. (2), with a clock speed of 250MHz and a data width of 32 bits.

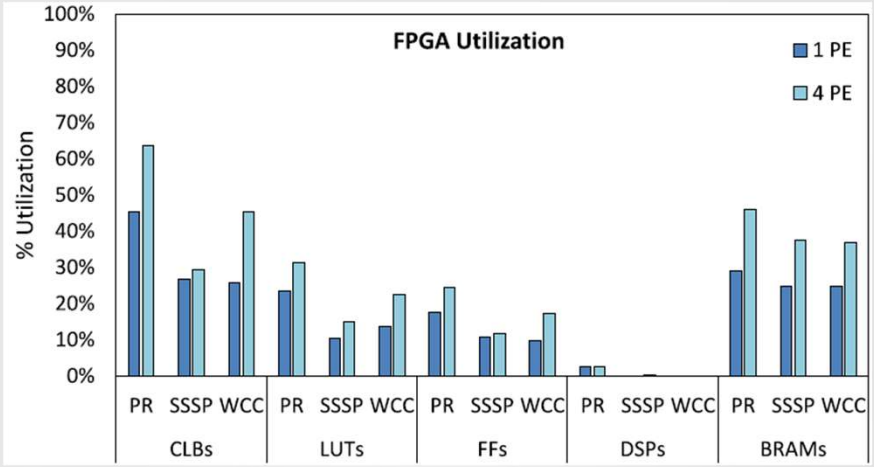
Configuration	Bandwidth (MBytes/s)
Without SPM	~ 2.35
With SPM	~ 3.12 (+32%)

FPGA resource utilization (U250) and Energy

UNIVERSITÀ DI SIENA 1240

GITHUB:

https://github.com/AminSahebi/distributed-graphfpga



The power utilization breakdown of the FPGA device based on benchmarks used in the evaluation.

	Power (watt)					
	1 PE			4 PE		
	Static	Dynamic	On-Chip	Static	Dynamic	On-Chip
PR	10.2	27.71	37.97	10.89	29.87	40.6
SSSP	8.04	18.27	26.28	8.12	18.69	26.8
WCC	8.4	20.06	28.4	8.65	21.29	29.93

Summary of Results



UNIVERSITÀ
DI SIENA 1240

- Evaluation Results (PageRank):
- - Up to 56x faster than CPU
- - 4x faster than GPU
- - 2.5x faster than FPGA alternatives
- Evaluation Results (SSSP):
- - Up to 65x faster than CPU
- - 26x faster than GPU
- - 18x faster than FPGA alternatives
- Evaluation Results (WCC):
- - Up to 403x faster than CPU
- - 7.4x faster than GPU
- - 10.3x faster than FPGA alternatives

Conclusions



UNIVERSITÀ
DI SIENA 1240

- **HashGrid achieves state-of-the-art performance for large-scale graph processing**
- HashGrid combines optimized partitioning with FPGA hardware
  - Features a custom memory layer for efficient data handling
  - Scratchpad memory and processing elements (PEs)
  - Load-compute-store pipeline for high throughput
- Benefits:
  - Improved workload balance
  - Reduced data transfer overhead
  - Scalable for large graphs
  - Lower power consumption despite higher throughput

A. Sahebi, M. Procaccini, R. Roberto Giorgi,
"HashGrid: An optimized architecture for accelerating graph computing on FPGAs",
Future Generation Computer Systems, vol. 162, 2025, doi:10.1016/j.future.2024.107497

- Future work: Expanding to multi-FPGA systems and new graph algorithms

BACKUP SLIDES (1)

the effective data transfer rate. With the SPM, we can actually overlap the load and compute phase, so $L = Load\,Latency$, while without the SPM, $L = Load\,Latency + Compute\,Latency$.

$$BW = F/(S + L + O) * (L * D) \quad (2)$$

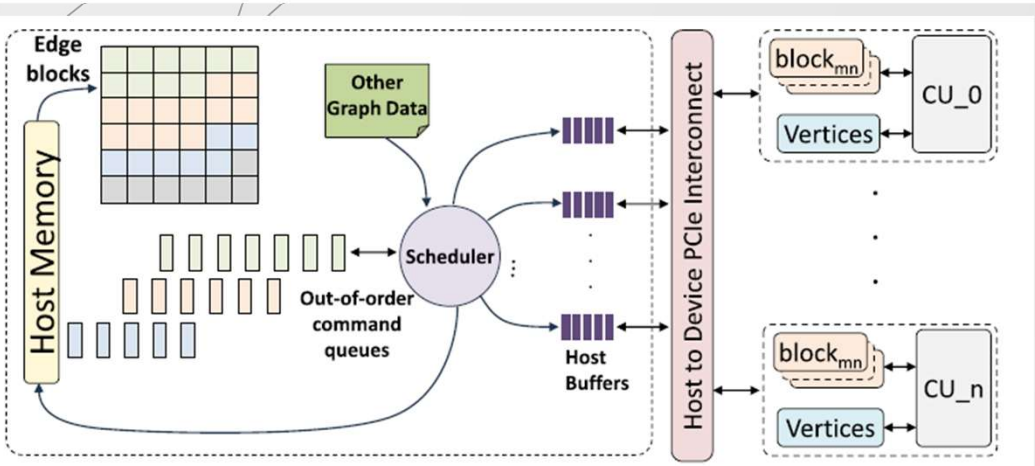


Fig. 5. The overview of the host's execution management, which is responsible for loading edge blocks from main memory, organizing the scheduler based on the out-of-order command queue, and finally, transmitting the data to the FPGA device, where the edge blocks are assigned to the available Computational Units (CUs).

P=60 is a sweet spot in out experiments/platforms



UNIVERSITÀ DI SIENA 1240

- Decreasing the buffer size of the transfer unit between host memory to FPGA BRAM, the throughput is deteriorating
- This is also the reason why it is not worthwhile to increase the number of partitions too much, or in other terms, the number we chose in our design (around $P = 60$) is a sweet spot that corresponds to a buffer size of around 1 MiB, i.e., the middle of the horizontal axis of Fig. 8.

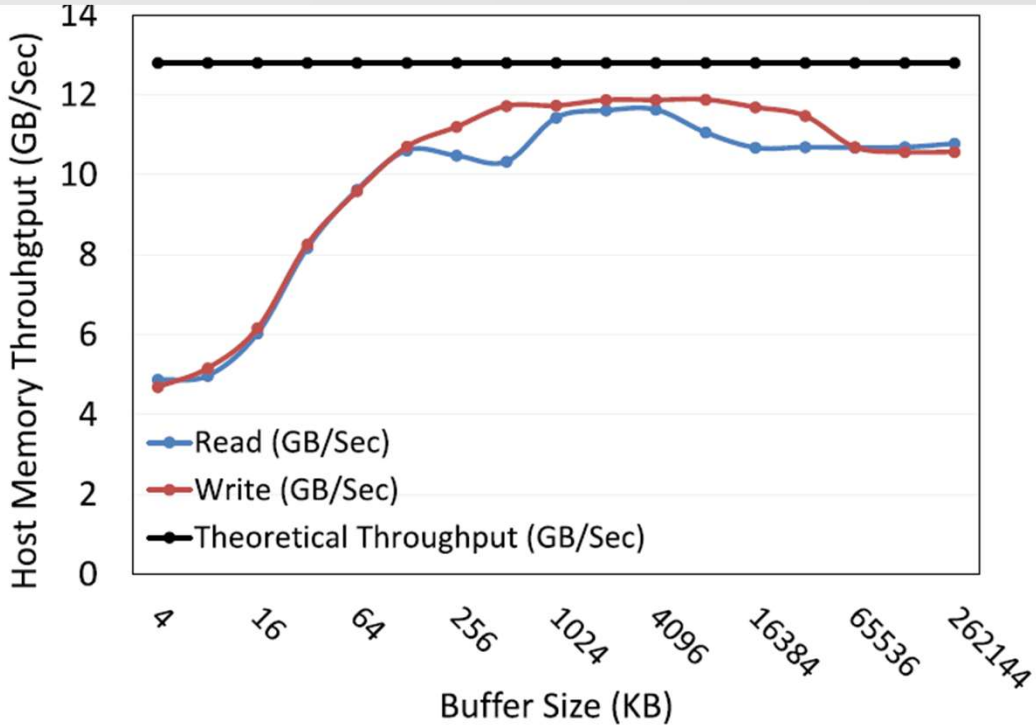


Fig. 8. Throughput of the host memory to the on-chip memory with different buffer data sizes. The throughput is measured in Gigabyte per second (GB/Sec), and the buffer size is in Kilobyte (KB).