High-Performance Graph Analytics Accelerator

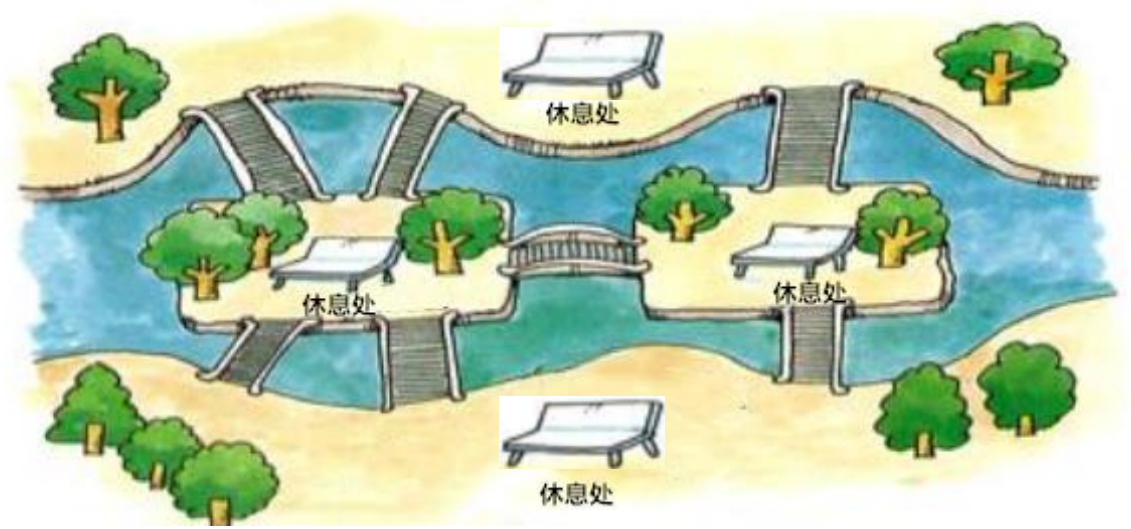
Long Zheng, **Pengcheng Yao**

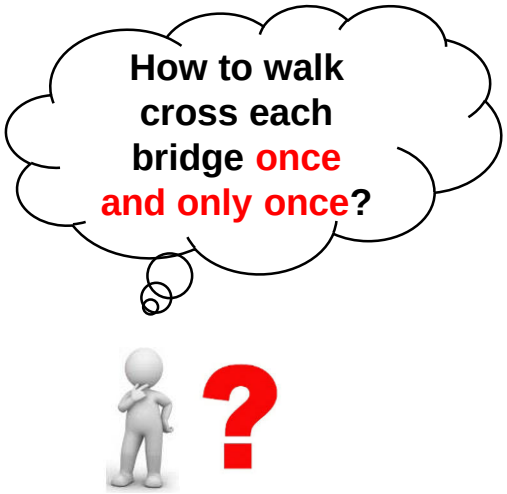
Huazhong University of Science & Technology

Outline

- **Introduction of Graph Analytics**
  - Background of Graph Analytics
  - The importance of Graph Analytics
  - Why we need Graph accelerator
- Introduction of our work
  - Performance analysis
  - Computation Architecture Designs
  - Memory System Designs
- Conclusion

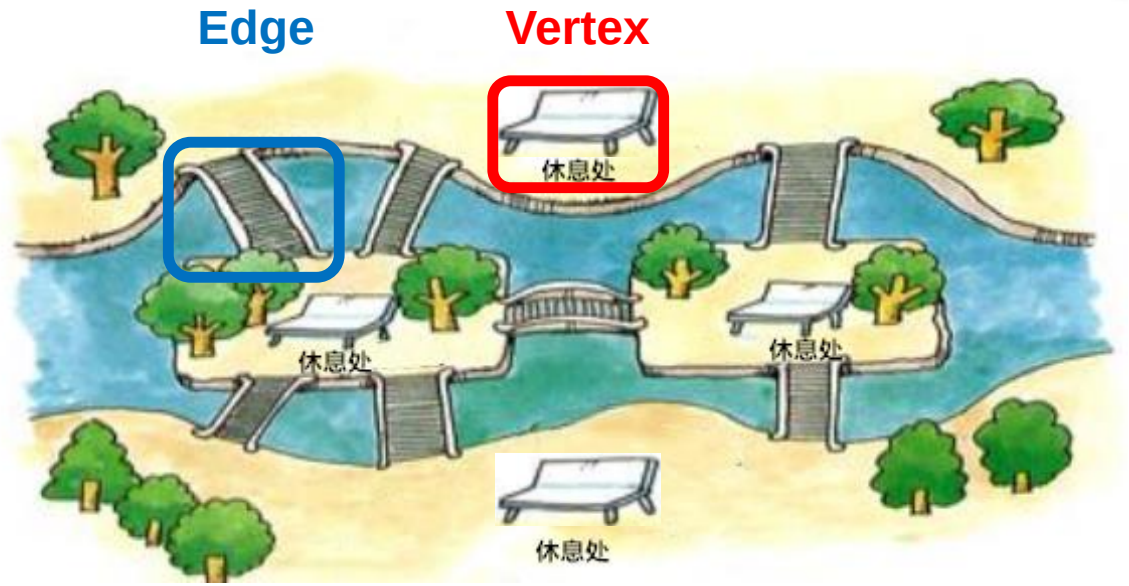
What is Graph?

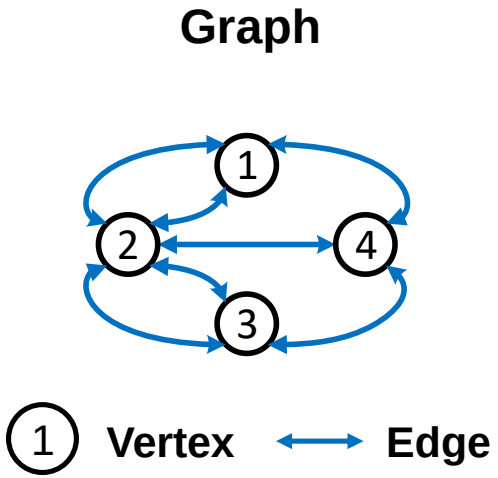




- **Origin**: Seven Bridges of Königsberg

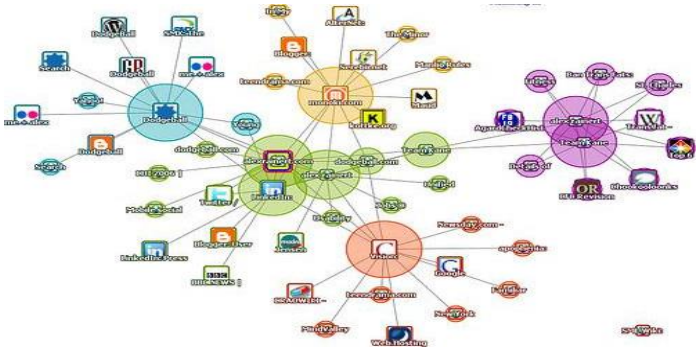
What is Graph?





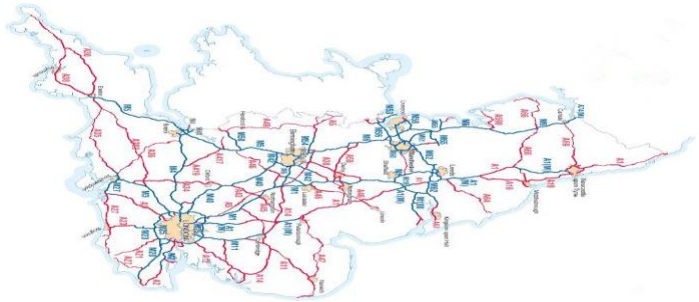
- **Origin**: Seven Bridges of Königsberg
- Graph: describing the relationships (edges) between different entities (vertices) with high flexibility and accuracy
  - ✓ Rest Area -> Vertex, Bridge -> Edge
  - ✓ An undirected graph with four vertices and seven edges

What is Graph?

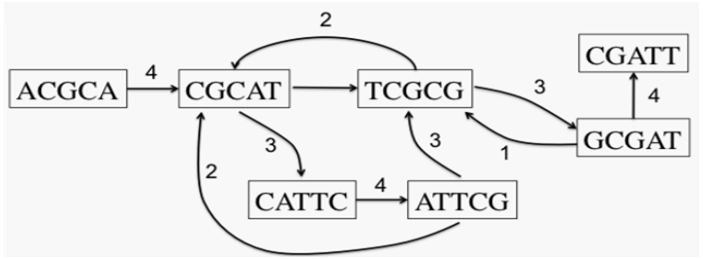


World Wide Web

Social Networks



Traffic Networks

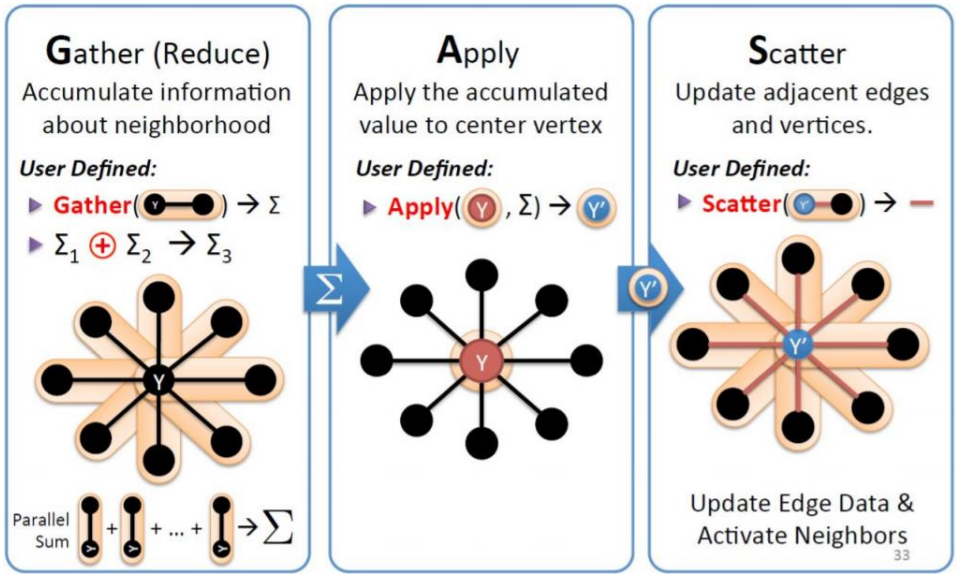


Biology

Graph is ubiquitous!

What is Graph Analytics?

- **Graph analytics**: solving real-world problems by analyzing graphs



Each **vertex** in the input graph **iteratively updates itself** based on the properties of its **neighbors**

- ✓ **Graph processing**
  - ✓ Computing vertex/edge properties to analyze graph property, e.g., BFS, SSSP, PageRank
  - ✓ Gather-Apply-Scatter: iterative computation based on neighbors

What is Graph Analytics?

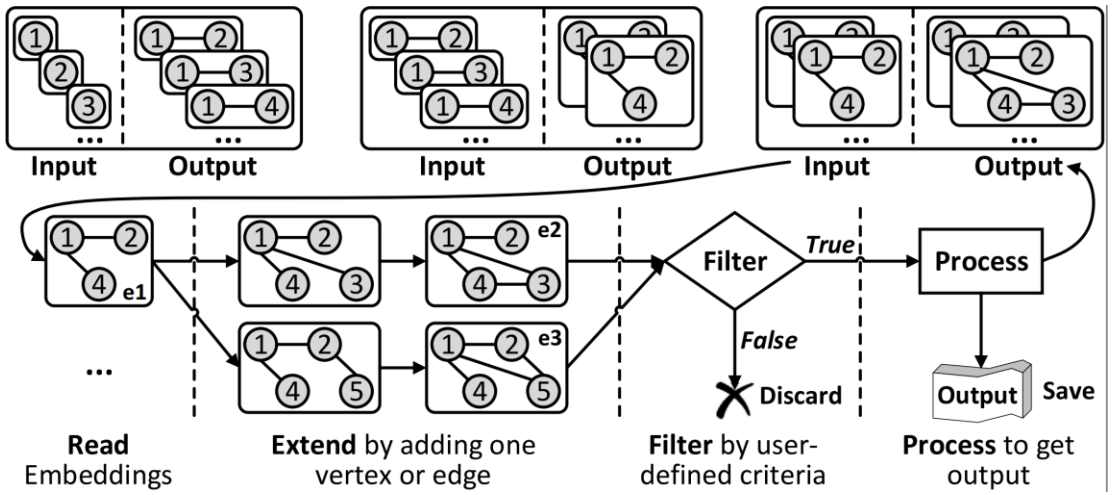
- **Graph analytics**: solving real-world problems by analyzing graphs

Model how people interact and influence each other to predict COVID-19 outbreaks

- ✓ **Graph processing**
  - ✓ Computing vertex/edge properties to analyze graph property, e.g., BFS, SSSP, PageRank
  - ✓ Gather-Apply-Scatter: iterative computation based on neighbors
  - ✓ Widely adopted to analyze relationships

What is Graph Analytics?

- **Graph analytics**: solving real-world problems by analyzing graphs

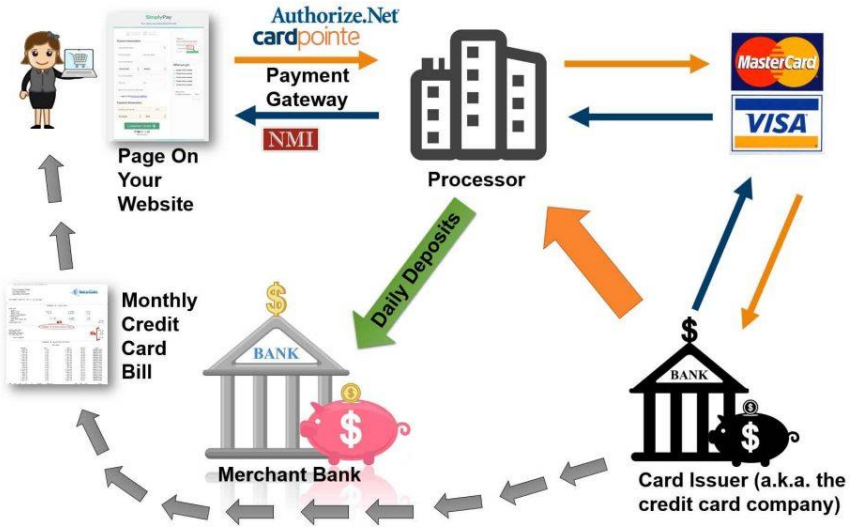


Each **subgraph** in the input graph **iteratively extends itself** based on the **edges** of its **neighbors**

- ✓ **Graph mining**
  - ✓ Searching interesting graph patterns to analyze graph structure, e.g., subgraph matching, motif counting
  - ✓ Extend-Filter-Process: iterative extension based on neighbors

What is Graph Analytics?

- **Graph analytics**: solving real-world problems by analyzing graphs



Detecting suspicious account by **searching specific cash flow**

- ✓ **Graph mining**
  - ✓ Searching interesting graph patterns to analyze graph structure, e.g., subgraph matching, motif counting
  - ✓ Extend-Filter-Process: iterative extension based on neighbors
  - ✓ Widely used in social network, finance, and biology

Why We Need Graph Analytics?

NLP

Image Processing

Autopilot

IOT

We are in a golden age of AI:
Machine Learning (ML) technologies are ubiquitous in our life

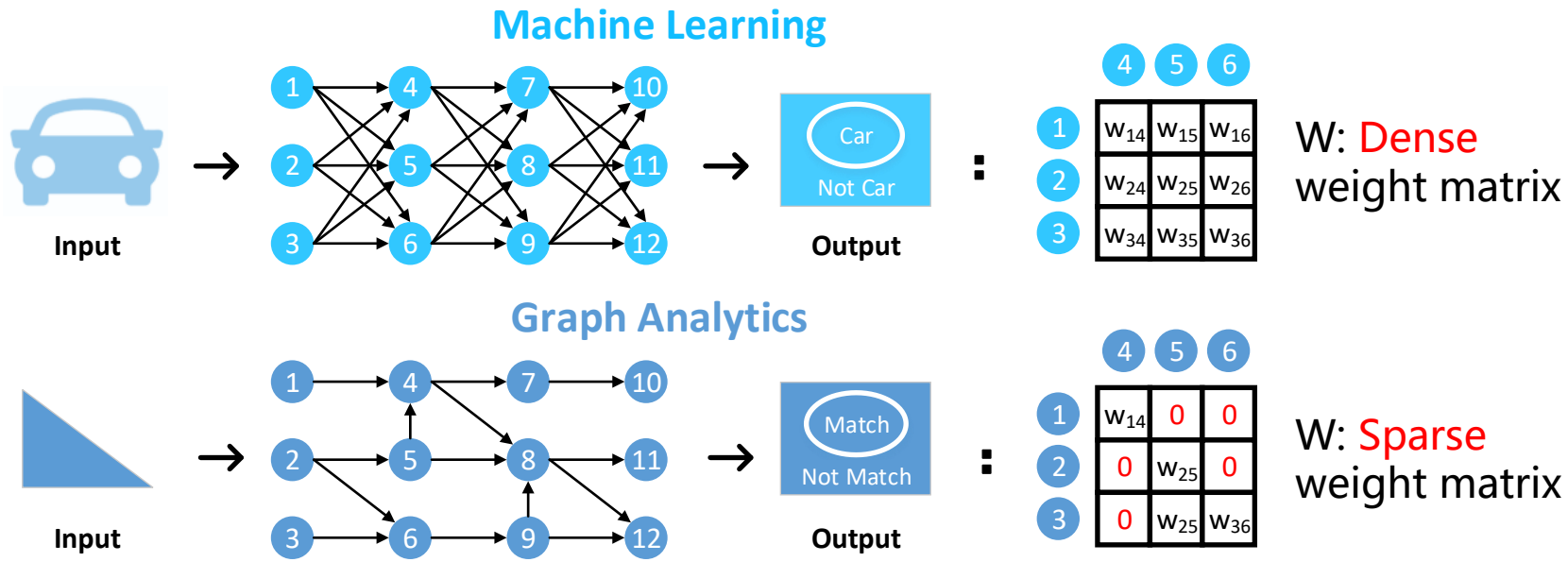
ASIC Designs

EDA Tools

Neuroscience

Why we need to research graph analytics (GA), which seems to be out of date?

Why We Need Graph Analytics?



- **Reason:** the need of processing sparse data in real-world applications
- Both of ML and GA can be abstracted as matrix computation
  - ✓ **ML:** Dense matrix computation, where most of the elements are nonzero (e.g., 100% in DNN)
  - ✓ **GA:** Sparse matrix computation, where most of the elements are zero (e.g., 99.9999997% in Twitter Graph)

Why We Need Graph Analytics?

Machine Learning

NLP

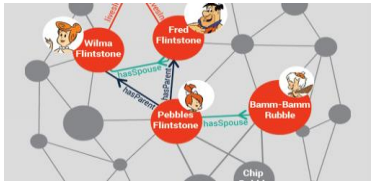
Image Processing

Autopilot

Speech Recognition

Graph Analytics

Social Network Analysis

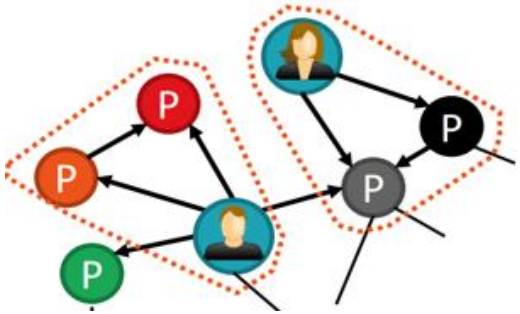


RDF Database

- **ML and GA are adopted in different areas**
  - ✓ **ML**: a representative of regular applications which process dense data
  - ✓ **GA**: a representative of irregular applications which process sparse data
  - ✓ Both of them are necessary for solving real-world problems

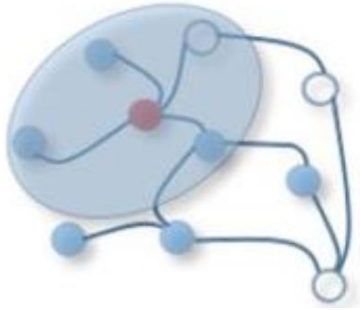
Why We Need Graph Analytics?

Sparsity is ubiquitous



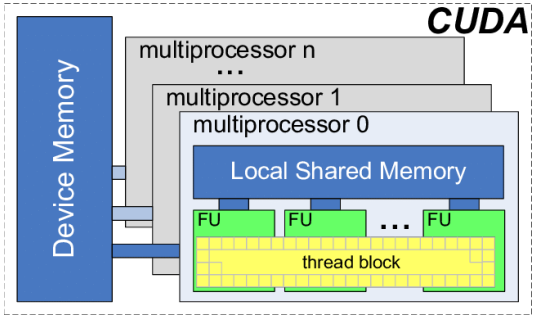
We only interact with a limited number of items

Sparsity also exists in dense applications



ML also has sparsity (Sparse NN, GNN)

Efficient processing technologies are missing

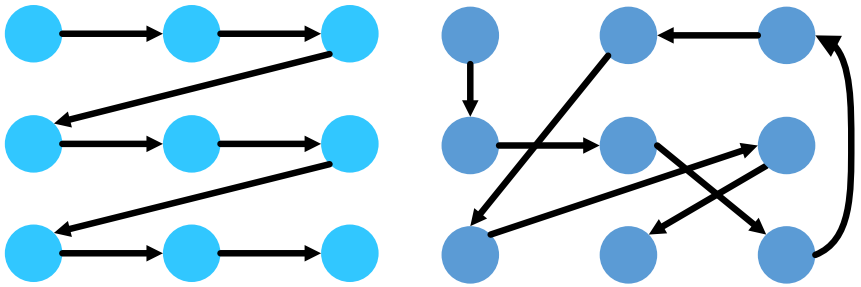


Existing SWs & HWs are designed for dense data

- **Researches on sparsity are increasingly important**
  - ✓ Sparsity is ubiquitous in our life
  - ✓ Even typical dense applications might have some kind of sparsity
  - ✓ Most of existing SWs and HWs are designed for dense data
  - ✓ Efficient processing technologies for sparse data are essential

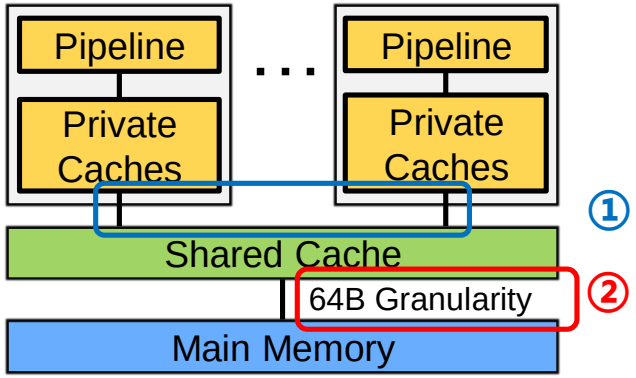
Why We Need Graph Accelerators?

Sparsity leads to irregular computation



- **ML: dense data process is regular and sequential**
- **GA: sparse data process is irregular and random**

Irregularity is inefficient on general-purpose architecture



① **Data contention on the same irregular data**

② **Mismatched access granularity**

- **Sparsity in graph data causes significant irregularity**
  - ✓ Sparse data leads to irregular computations and memory accesses
  - ✓ General-purpose processors (e.g., CPU, GPU) are designed to efficiently process dense data
  - ✓ Sparse data cases significant data contentions and random accesses

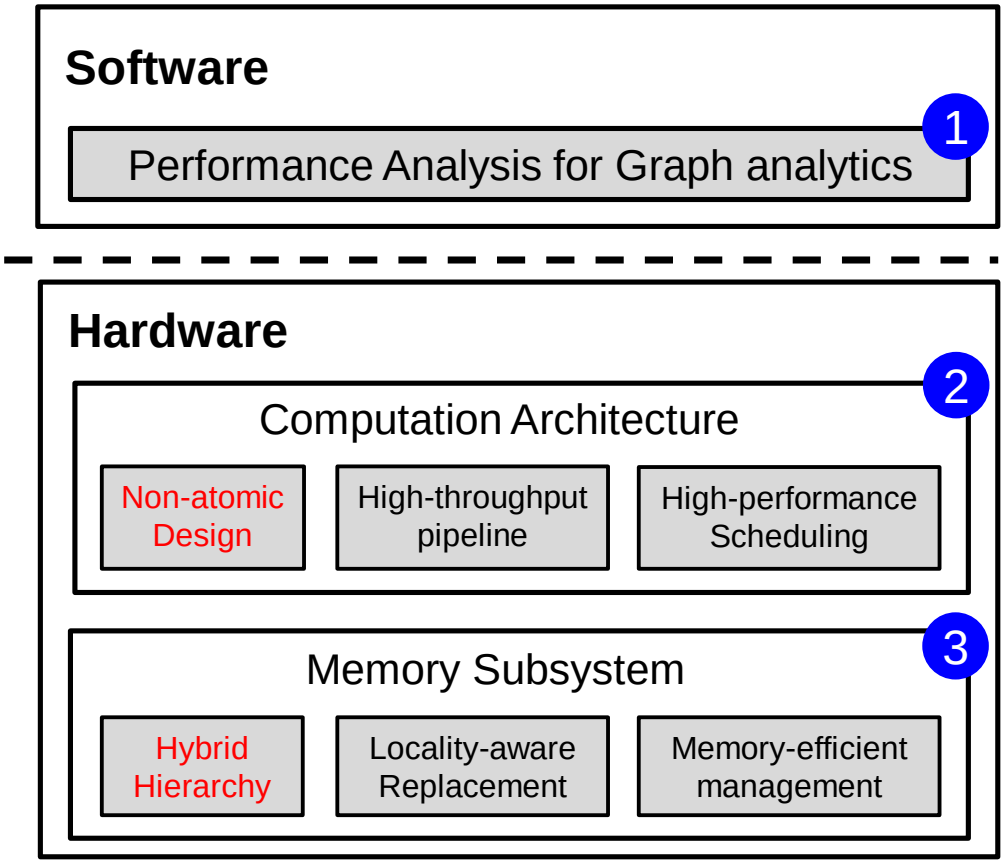
Outline

- Introduction of Graph Analytics
  - Background of Graph Analytics
  - The importance of Graph Analytics
  - Why we need Graph accelerator
- **Introduction of our work**
  - Performance analysis
  - Computation Architecture Designs
  - Memory System Designs
- Conclusion

Our Objective

Building A General-Purpose Graph Analytics Computer

Our Work in Accelerator Designs



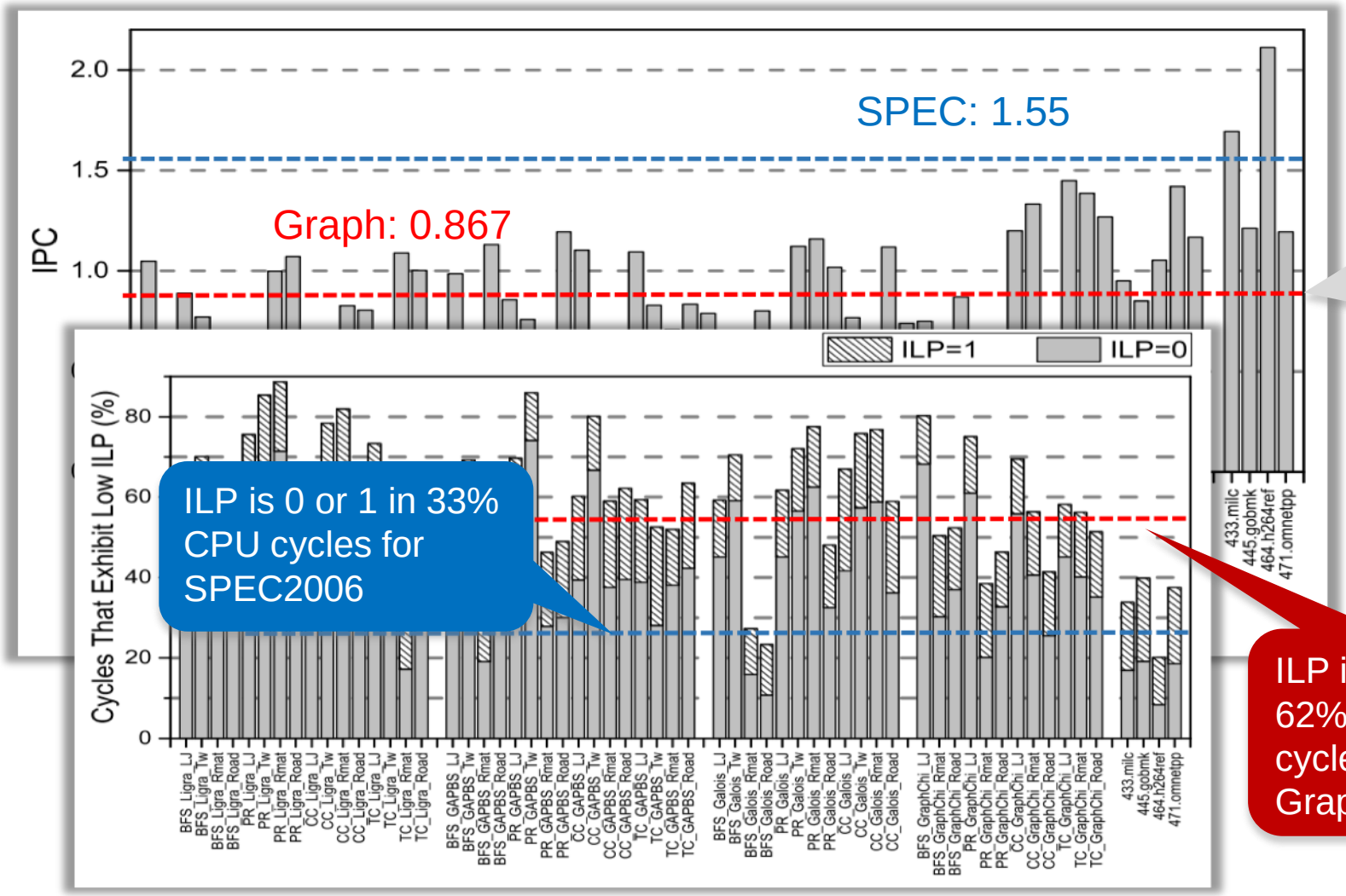
1. Towards Dataflow-based Graph accelerator (**ICDCS'17**)
2. An Efficient Graph Accelerator with Parallel Data Conflict Management (**PACT'18**)
3. A Locality-Aware Energy-Efficient Accelerator for Graph Mining Applications (**MICRO'20**)

Performance Analysis — ICDCS'17

		Prior Work	Our Work
Algorithm	*Not Stalled*	*Memory Sys.*	*Inside Core*
Breadth-First Search	25.486	38.022	36.492
PageRank	23.704	46.683	29.613
Connected Components	20.691	43.751	35.558
Triangle Counting	23.674	37.750	38.576

- **Understanding the Performance bottlenecks**
  - ✓ Existing work believes that graph analytics is bottlenecked by memory accesses
  - ✓ We find that most (i.e., 41%) of the CPU cycles are stalled on memory accesses
  - ✓ There are still 35% of the CPU cycles are stalled on core execution

Performance Analysis — ICDCS'17

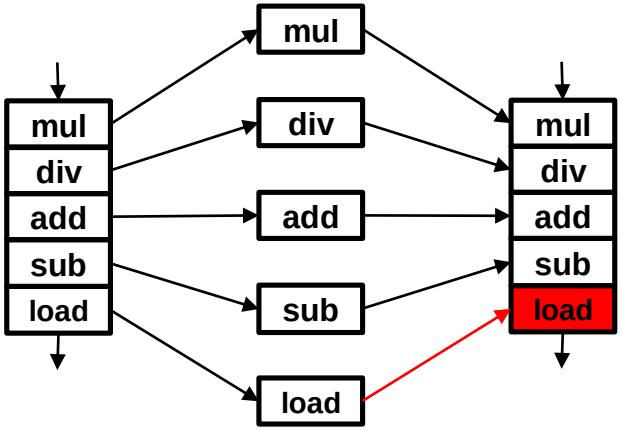


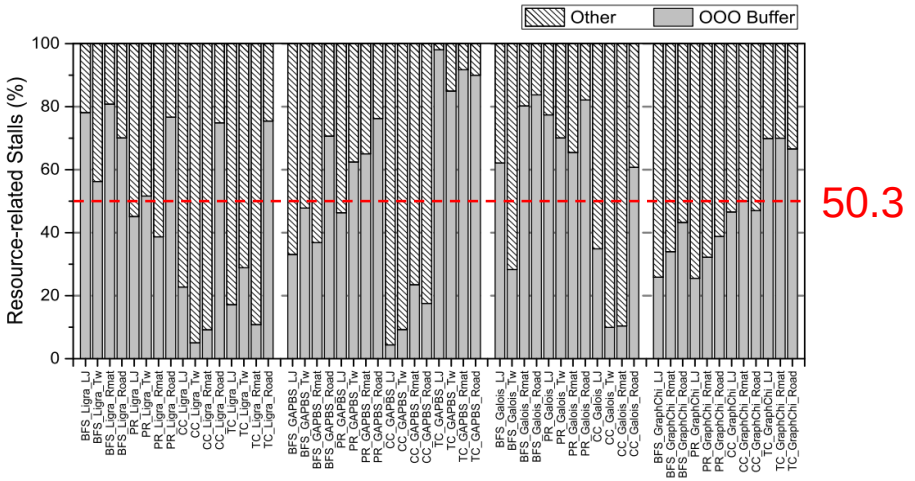
Average IPC is only 0.87

ILP is 0 or 1 in 33% CPU cycles for SPEC2006

ILP is 0 or 1 in 62% CPU cycles for Graph Analytics

Performance Analysis — ICDCS'17



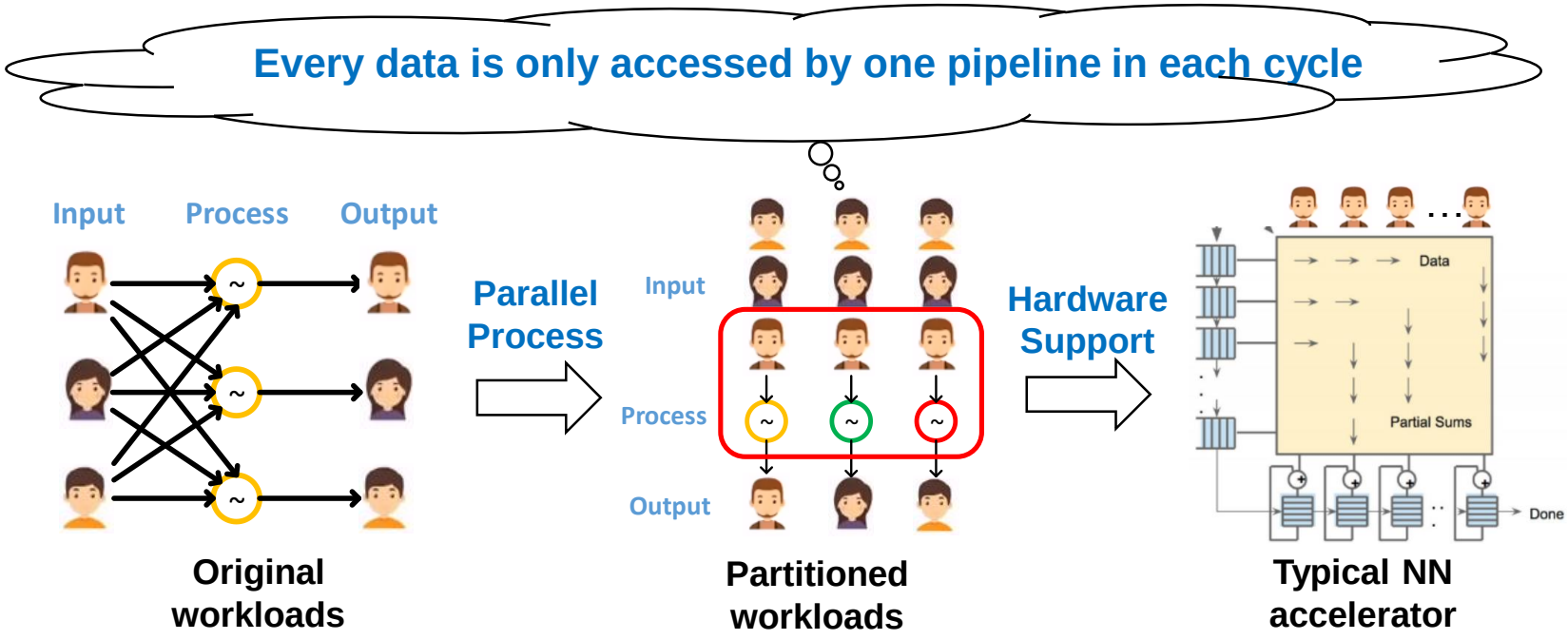


- **Limitation of Out-of-Order Execution**
  - ✓ Out-of-order execution, but sequential retirement
  - ✓ When facing a long latency instruction, OOO buffers can soon be clogged by succeeding instructions that are already executed
  - ✓ OOO buffers account for 50.3% of the total resource-related stalls
- **Long latency instructions in graph analytics**
  - ✓ **Computation**: atomic operations; **Memory**: random vertex/edge accesses

More details in "Towards Dataflow-based Graph accelerator"

Computation Architecture — PACT'18

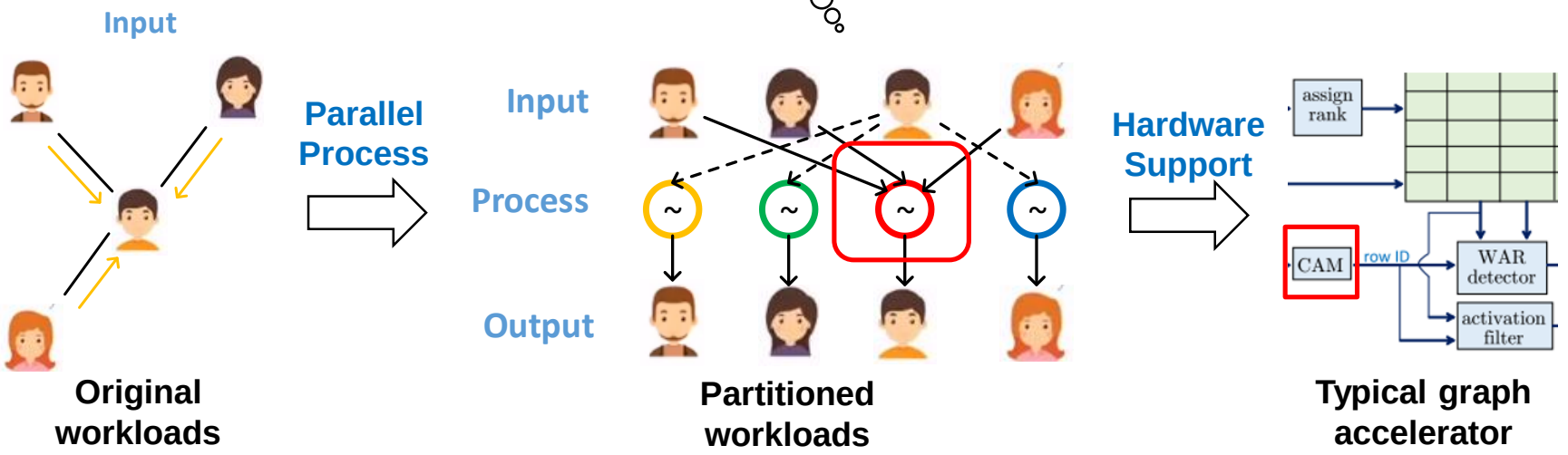
- **Inefficiencies of atomic operations in graph analytics**
  - ✓ Sparsity cause irregular computations, which lead to data contention



Computation Architecture — PACT'18

- **Inefficiencies of atomic operations in graph analytics**
  - ✓ Sparsity cause irregular computations, which lead to data contention
  - ✓ To ensure the correctness of results, existing work uses atomic operations to serialize the process of conflict data

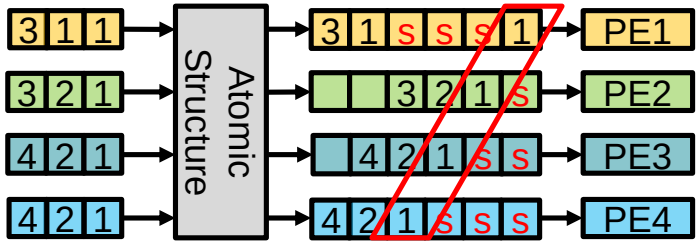
Every data might be simultaneously accessed by multiple pipelines



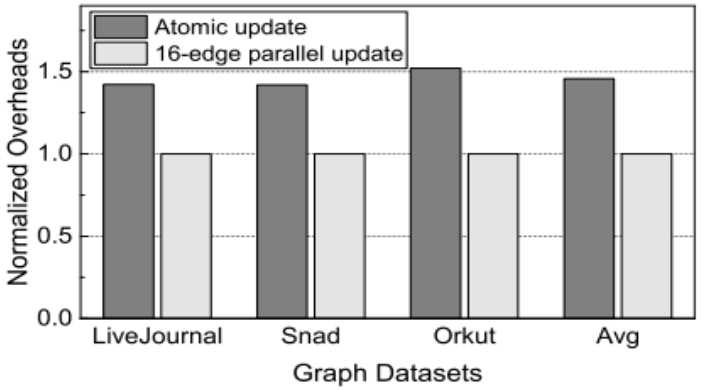
Graph Analytics

Computation Architecture — PACT'18

- **Inefficiencies of atomic operations in graph analytics**
  - ✓ Sparsity cause irregular computations, which lead to data contention
  - ✓ To ensure the correctness of results, existing work uses atomic operations to sequentially access the conflict data
  - ✓ Overheads of atomic operations: 45%



The process of vertex 1 is blocked

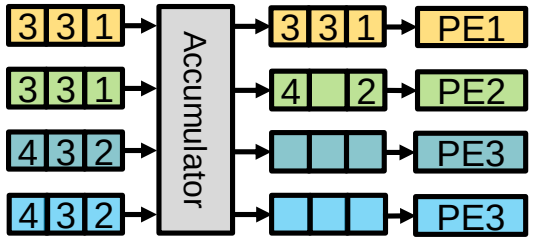


Whether we can parallelize the process of conflict data while ensuring the accuracy of results?

Computation Architecture — PACT'18

- **Insight: Parallel accumulation**

Algorithm	Operation Type
Breadth-First Search	CAS if less
Weakly Connected Components	CAS if less
Shortest Path	CAS if less
PageRank	Atomic add
Triangle Counting	Atomic add
Degree Centrality	Atomic add
Collaborative Filtering	Atomic add

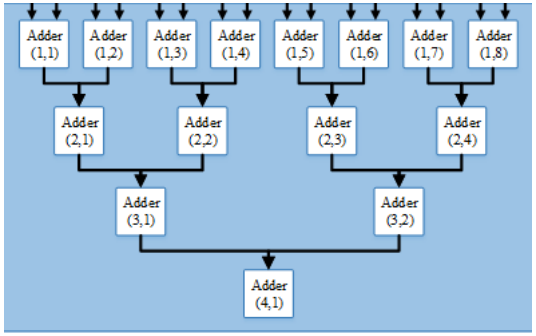


- ✓ **Characteristics of graph atomic operations:** following commutative law and associative law
- ✓ **Insight:** accumulating the atomic operations in parallel before writing back the results

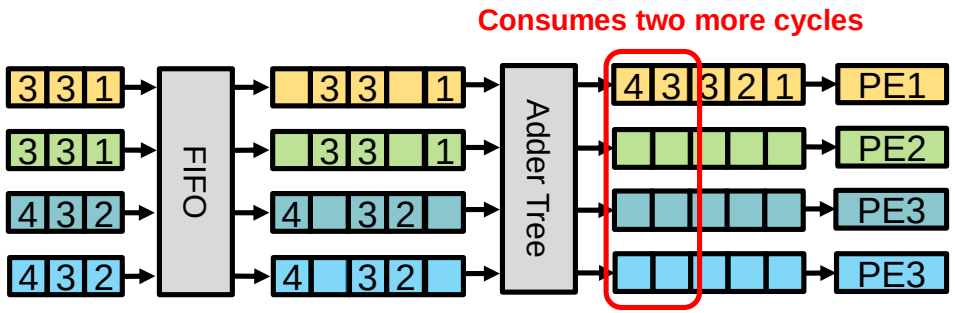
The insight sounds to be promising, but not easy to be efficiently implemented

Computation Architecture — PACT'18

- **Challenge: Irregular vertex degrees**



Traditional accumulator (adder tree)



Suboptimal performance of adder tree

- ✓ Different vertex has different number of degrees (i.e., the data received in each cycle belongs to **multiple** vertices)
- ✓ Traditional adder tree **can only accumulate for one vertex**, leading to suboptimal performance
- ✓ Using multiple adder trees is kind of a **cop-out**. It works, but **significantly increases hardware fanouts**

Computation Architecture — PACT'18

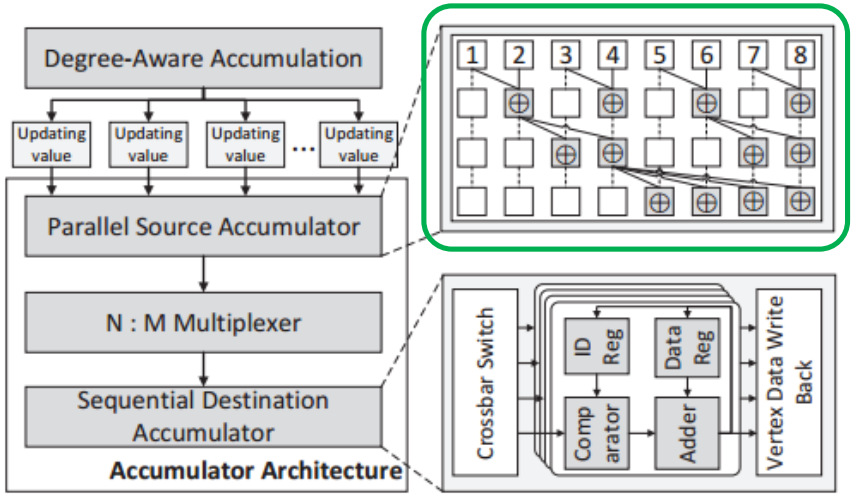
- **Problem Formulization**
  - ✓ **Original (integer programming)**: $p_j = \sum_{1 \le i \le N} a_i \cdot b_{ij}, 1 \le j \le M$
  - ✓ **Characteristic**
    - partial order in pull model, i.e., all requests are sent in an ascending order of destination vertex
    - $b_{ij} \le b_{kj}$, if $i \le k$
  - ✓ **Simplified (Prefix Sum)**: $p_j = f(c_j^2)$, where

$$f(i) = \begin{cases} f(i-1) + a_i, & i \notin \{c_1^1, c_2^1, \ldots, c_M^1\} \\ a_i, & i \in \{c_1^1, c_2^1, \ldots, c_M^1\} \end{cases}$$

- **Differences between traditional prefix sum and our problem**
  - ✓ **Breakpoint**
    - $f(i)$ is initialized to a_i when a break point c^1_m appears
    - Accumulator needs to find all dynamically changing c^1_m
  - ✓ **Filter**:
    - Only $f(c^2_j)$ is the valid results, where c^2_j also dynamically changes
    - Accumulator needs to precisely select the accumulated results

Computation Architecture — PACT'18

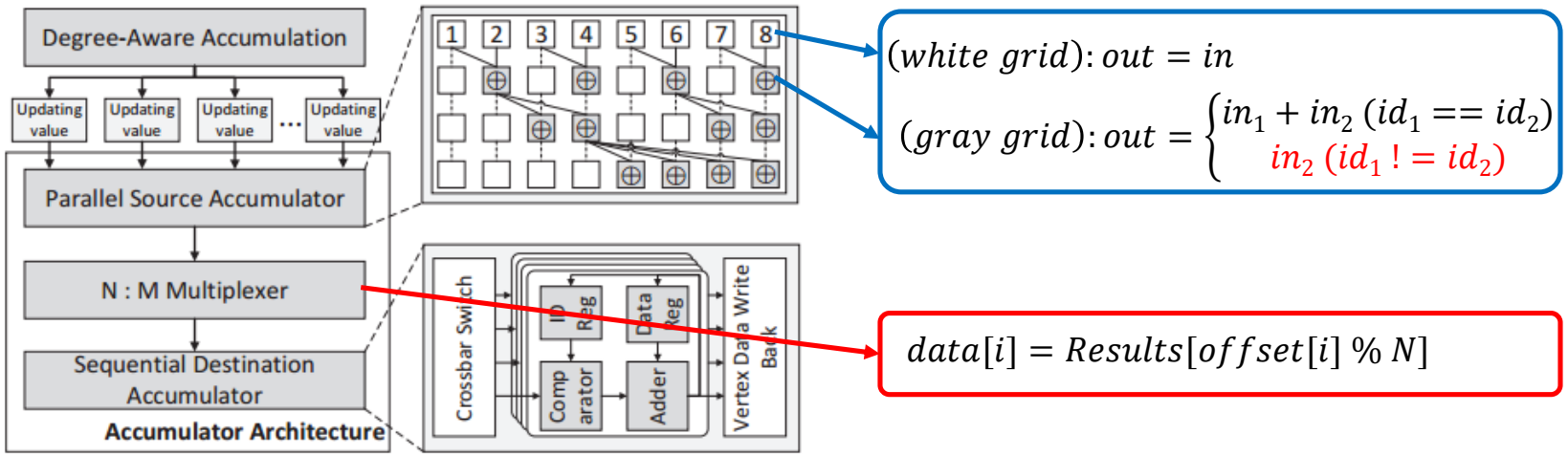
- **Methodology**
  - ✓ Optimize traditional prefix sum adders to mitigate the gaps
  - ✓ We choose **Ladner-Fischer adders** for minimized latency and resources
  - ✓ **Partial order** in c^1_m ($c^1_i \le c^1_j$ if $i \le j$)
  - ✓ **Adding log(|Pipeline|) bits** to each update as vertex ID
  - ✓ c^1_m can be generated in runtime by comparing the vertex ID of consecutive updates



$$p_j = f(c^2_j), \text{ where}$$

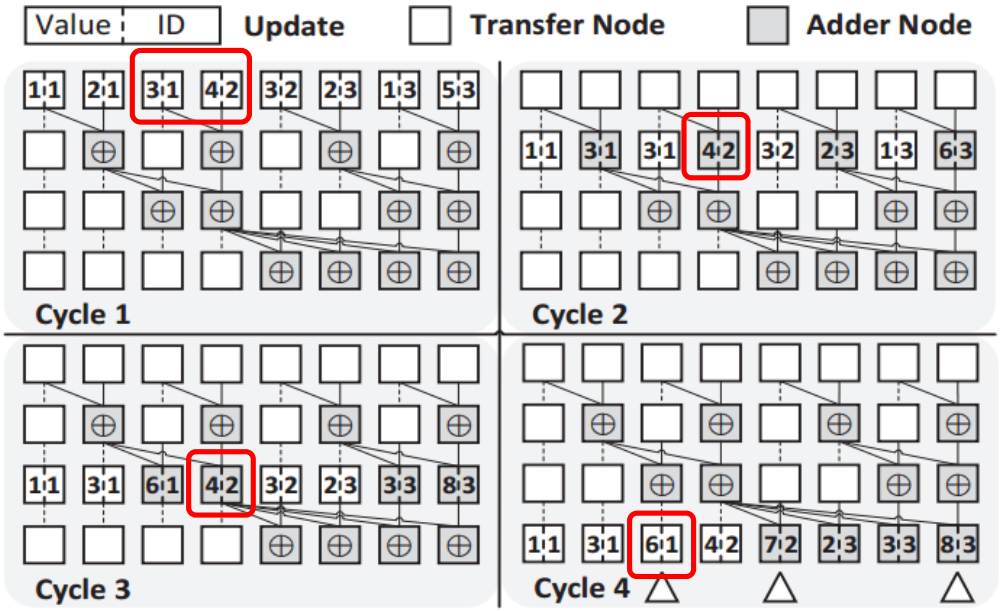
$$f(i) = \begin{cases} f(i-1) + a_i, & i \notin \{c^1_1, c^1_2, \ldots, c^1_M\} \\ a_i, & i \in \{c^1_1, c^1_2, \ldots, c^1_M\} \end{cases} \Rightarrow f(i) = \begin{cases} f(i-1) + a_i, & v_i = v_{i-1} \\ a_i, & v_i \neq v_{i-1} \end{cases}$$

Computation Architecture — PACT'18



- **Hardware Designs**
  - ✓ Breakpoint recognition: replacing original adder logic with the conditional adding logic
  - ✓ Data filtering: select the accumulated results in the location of each vertex's last update

Computation Architecture — PACT'18

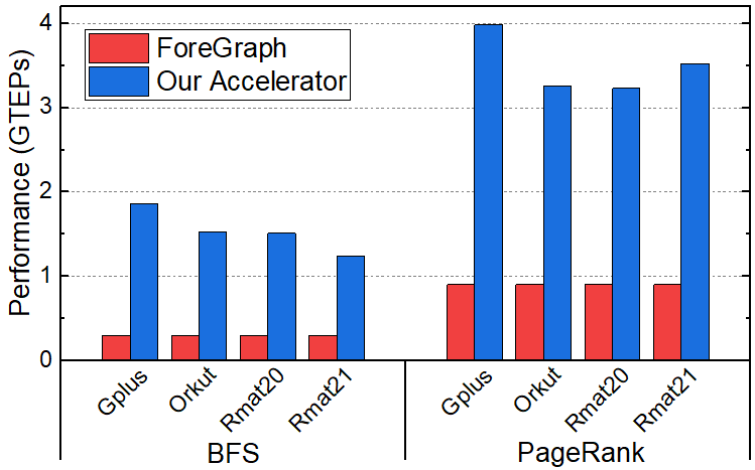


- **Example**
  - ✓ Fully pipelined accumulation
  - ✓ Accumulator receives 8 updates in the 1st cycle
  - ✓ Adder directly sends the second data when finding a breakpoint in the 2nd and 3rd cycle
  - ✓ Because the location of the last updates of vertex 1 is 3, accumulator select the data in the 3rd port as the results for vertex 1

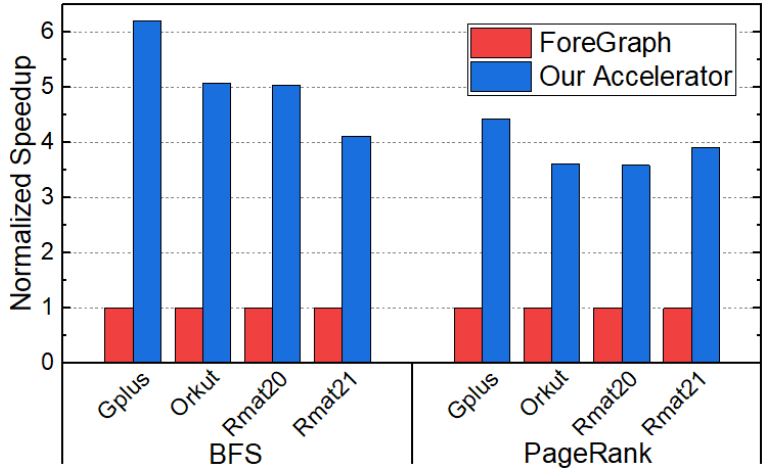
Computation Architecture — PACT'18

➢ Evaluation

- ✓ **Platform:** Xilinx U250 Acceleration card, use 2 channel memory only
- ✓ **Performance (**170MHz ~ 250MHz**):**
  1.23 - 1.86 GTEPs (Overall) or **3.69 – 5.58** GTEPs (Single Iteration) for BFS
  3.22 - 3.98 GTEPs for PageRank
- ✓ **Normalized Speedup: 3.6-6.2x** speedup comparing to state-of-the-art
- ✓ **3.96x** speedups for finance anti-fraud applications from Ping An Technology



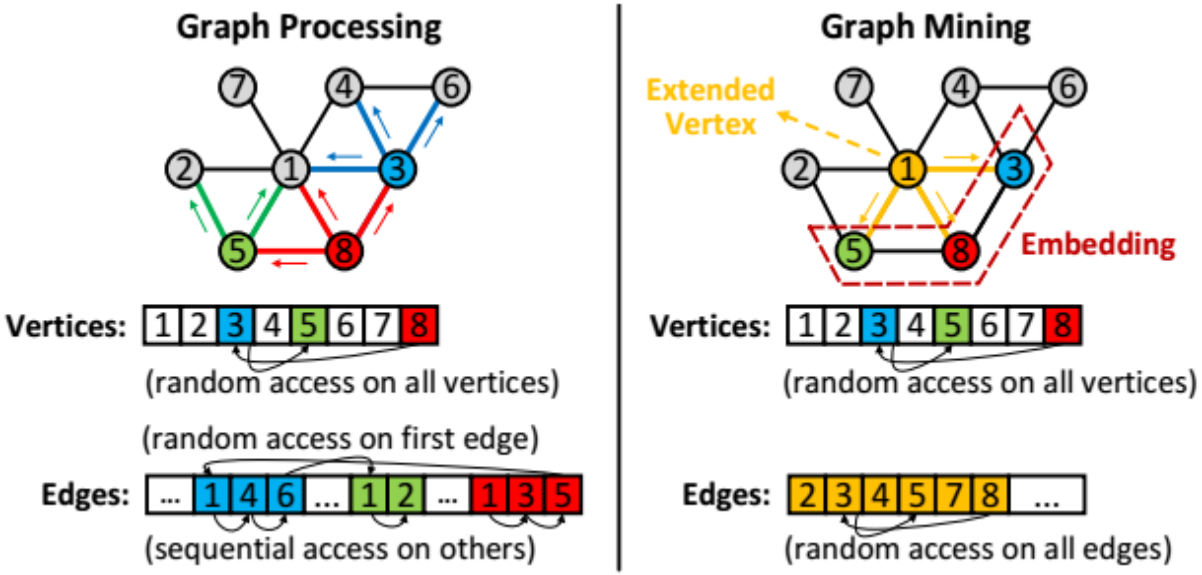
Performance



Speedup

More details in "An Efficient Graph Accelerator with Parallel Data Conflict Management "

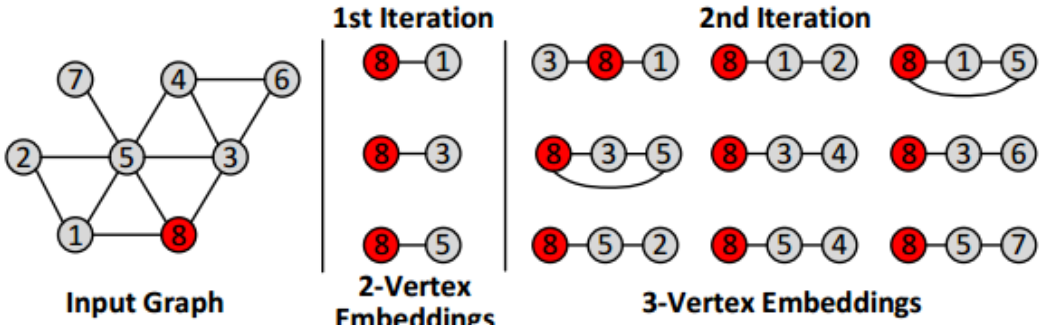
Memory Subsystem — MICRO'20



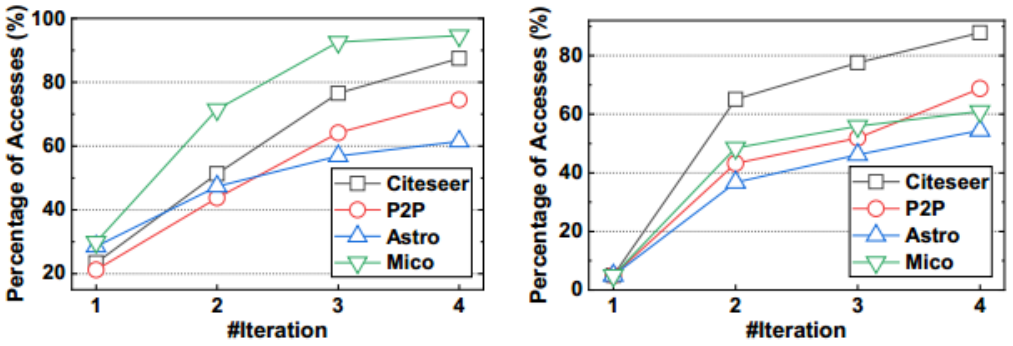
- **Memory Access Characteristics**
  - ✓ Graph processing: random **vertex** access
  - ✓ Graph Mining: random **vertex and edge** access
  - ✓ Existing accelerators **statically hold all vertex data on-chip**
  - ✓ While this mechanism is efficient for graph processing, it is **not practical** to hold all edge data for graph mining **with limited on-chip memory capacity**

Memory Subsystem — MICRO'20

➢ **Insight: Exploration Locality**



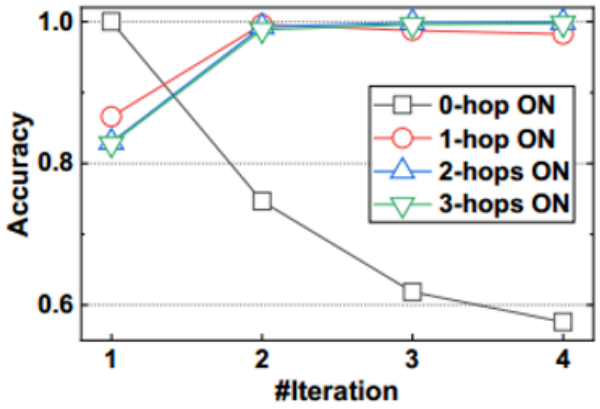
✓ An intuition is whether we can achieve considerable performance by **storing only a small portion of data**?

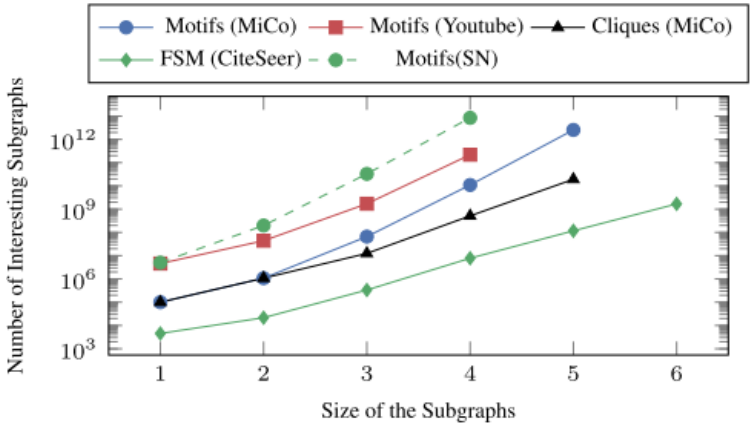


(a) Vertex access (b) Edge access

✓ **Exploration locality:** the power-law distribution in natural graphs is amplified during the process. **5% data can generate at most 95% memory accesses**

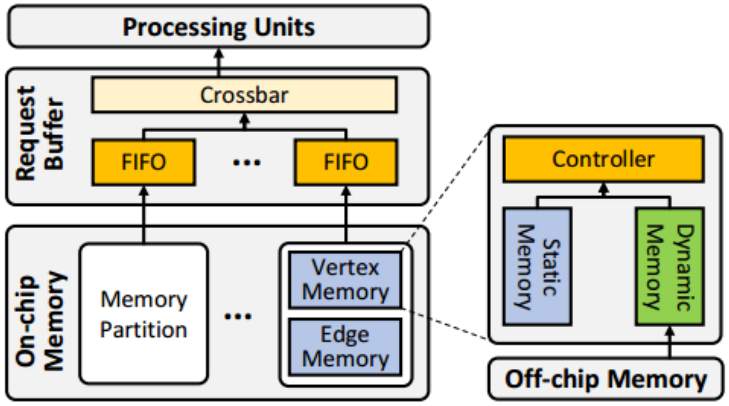
Memory Subsystem — MICRO'20



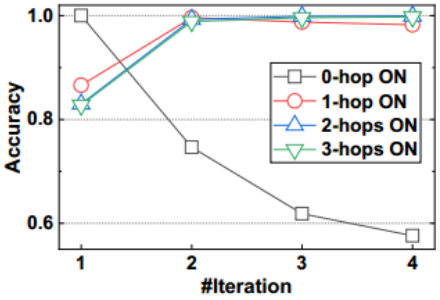


- **Methodology**: Divide-and-Conquer
  - ✓ **Statically retaining frequently accessed data, and dynamically replacing the left**
- **Challenge**
  - ✓ Frequency calculation: hard to precisely locate the frequently accessed data
  - ✓ Intermediate results: hard to be stored in off-chip memory

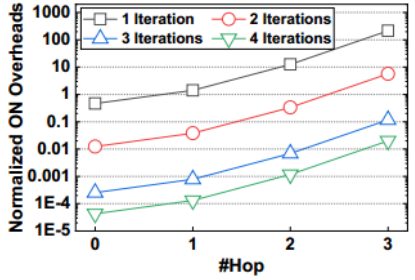
Memory Subsystem — MICRO'20



$$victim = \arg\max\{Rank(\mathrm{ON}_1(v)) + \lambda Rec(v)\}, v \in V_{Low}$$



(a) Accuracy

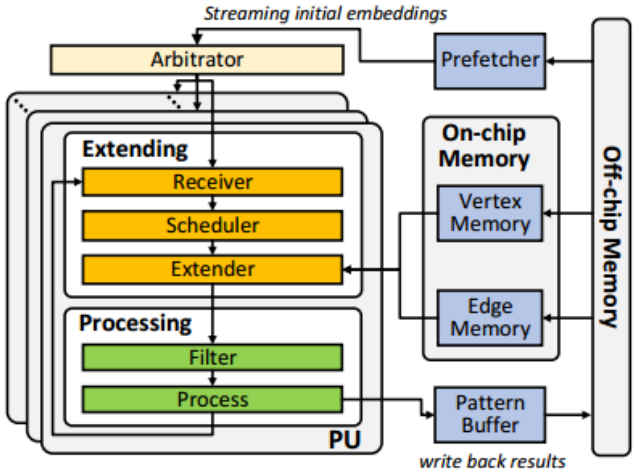


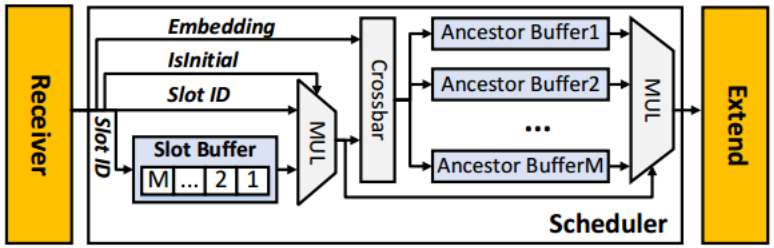
(b) ON-computation overheads

- **Hardware Design**
  - ✓ Memory hierarchy: static memory (scratchpad memory) + dynamic memory (cache)
  - ✓ **Observation**: access frequency of a vertex is mainly determined by the degrees of itself and its 1-hop neighbors
  - ✓ Data addressing: predict access frequency with high accuracy and low overheads
  - ✓ Replacement policy: Locality-aware replacement

Memory Subsystem — MICRO'20

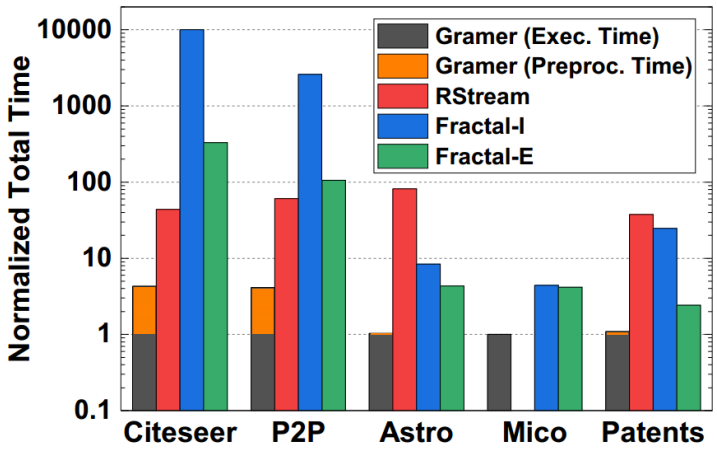
- **Hardware Design**
  - ✓ DFS-based architecture: extending the subgraphs in DFS manner to avoid writing back intermediate results
  - ✓ Slot-based pipeline: efficiently process different subgraphs in parallel in a dataflow manner
  - ✓ Data compression: compacting the parameters of DFS traverse
  - ✓ Load-balance: aggressive and precise work-stealing mechanism



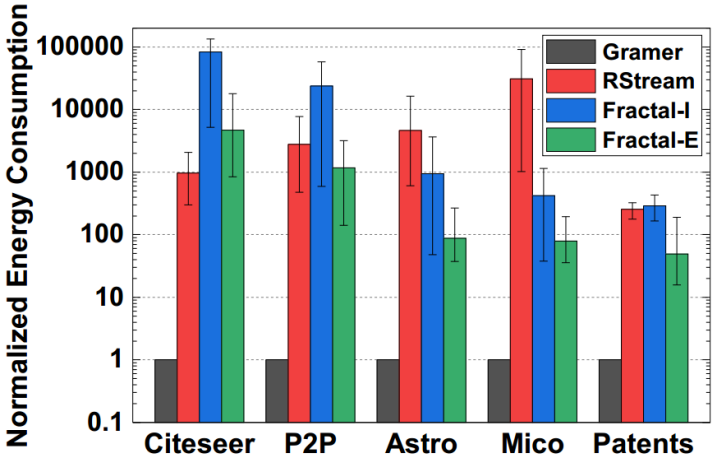


Memory Subsystem — MICRO'20

- **Evaluation**
  - ✓ **Platform:** Xilinx U250 FPGA acceleration card
  - ✓ **Performance:** ≥ 2.4x speedups comparing to state-of-the-art Fractal
  - ✓ **Energy efficiency:** ≥ 33.6x comparing to state-of-the-art Fractal



Performance



Energy Efficiency

More details in "A Locality-Aware Energy-Efficient Accelerator for Graph Mining Applications"

Outline

- Introduction of Graph Analytics
  - Background of Graph Analytics
  - The importance of Graph Analytics
  - Why we need Graph accelerator
- Introduction of our work
  - Performance analysis
  - Computation Architecture Designs
  - Memory System Designs
- **Conclusion**

Conclusion

- Graph is ubiquitous in our life
- Researching graph analytics is important
- Graph processing is bottlenecked by both computation and memory access
- Atomic operations in graph analytics can be accumulated in parallel to avoid pipeline stalls
- Exploration locality can be leveraged to achieve consider memory performance on large volume of graph data

END

Thanks!