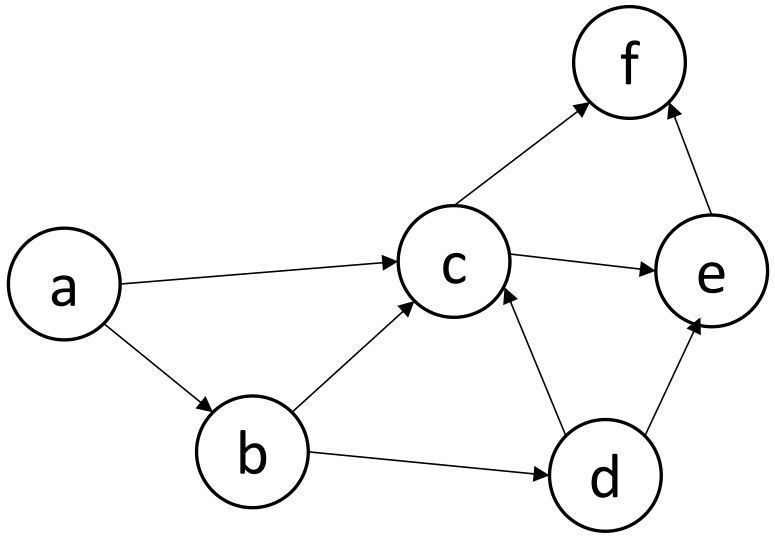
HyTGraph:GPU-Accelerated **Graph** Processing with **Hybrid Transfer** Management

Qiange Wang* , Xin Ai*, Yanfeng Zhang, Jing Chen, Ge Yu
School of Computer Science and Engineering
Northeastern University, Shenyang, China

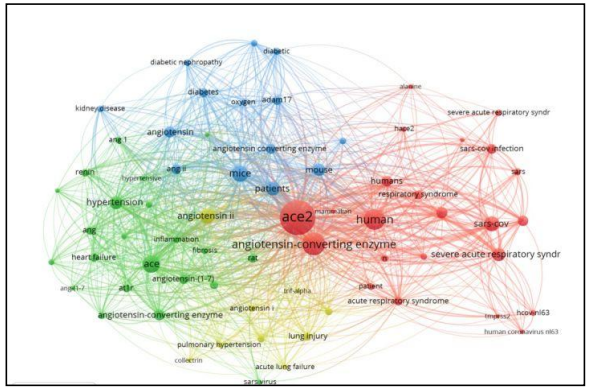
Graph Processing

Vertex: real-world entity

Edge: dependency between entities



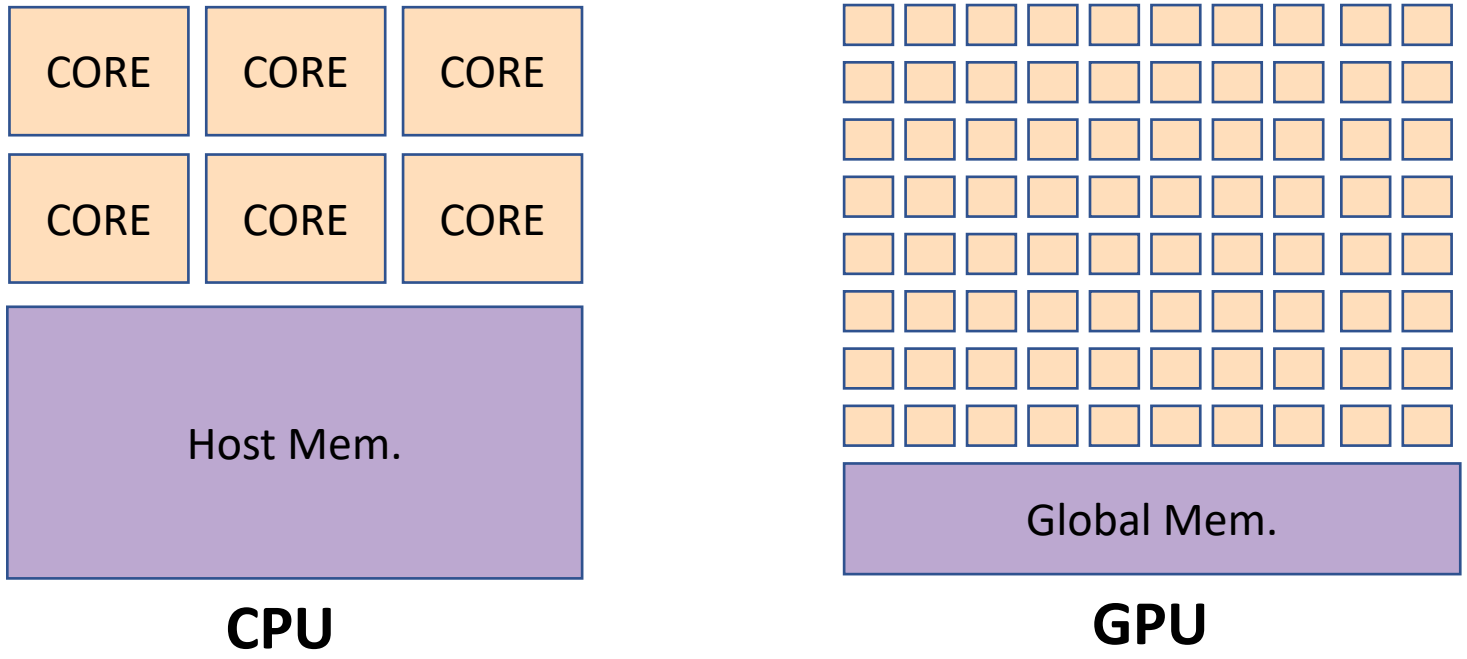
Social Network



Web Architecture

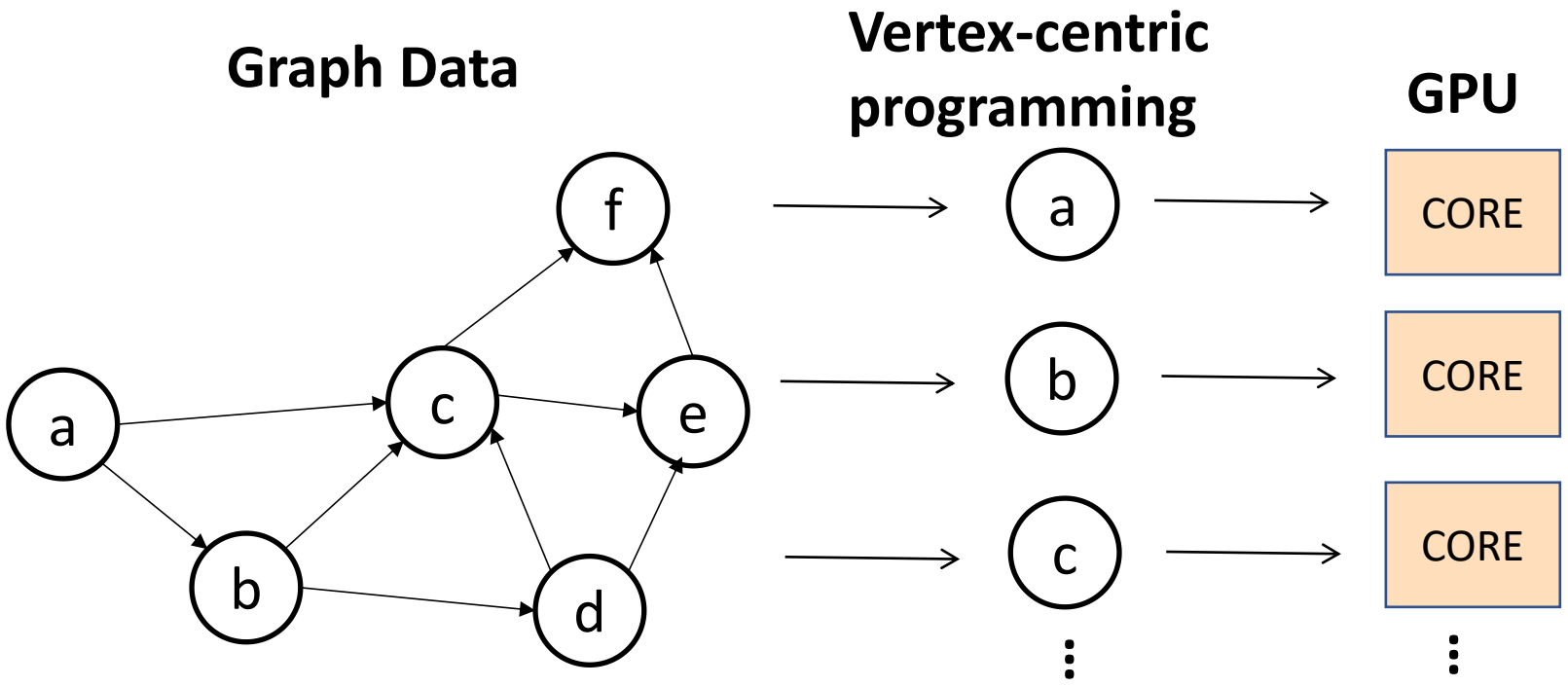
Human Genome

CPU and GPU



- CPU：Large Memory Capacity(main memory)；Low Parallelism
- GPU：Limited Memeory Capacity；High Parallelism

GPU-Accelerated Graph Processing



Suitable for GPU processing

Existing Works

Early works mainly focused on the internal-GPU bottlenecks

- Programming Model： **Harish**[HiPC'07], **MeduSa**[TPDS'14],**Gunrock** [HPDC'14]
- Irregular Memory Access: **CuSha**[HPDC'14],**Tigr** [ASPLOS'18],**CTA** [HPCA'14]
- Propagation Direction: **SEP-Graph**[PPOPP'19]
- Asynchronous Processing: **Groute**[PPOPP'17], **SEP-Graph**[PPOPP'19]

Existing Works

Early works mainly focused on the inner-GPU bottlenecks

- Programming Model: **Harish**[HiPC'07], **MeduSa**[TPDS'14],**Gunrock** [HPDC'14]
- All these works assume that the GPU(s) can accommodate the whole graph.
- Propagation Direction: **SEP-Graph**[PPOPP'19]
- Asynchronous Processing: **Groute**[PPOPP'17], **SEP-Graph**[PPOPP'19]

Existing Works

Early works mainly focused on the inner-GPU bottlenecks

- Programming Model: **Harish**[HiPC'07], **MeduSa**[TPDS'14],**Gunrock** [HPDC'14]
- All these works assume that the GPU(s) can accommodate the whole graph.
- Propagation Direction: **SEP-Graph**[PPOPP'19]
- Asynchronous Processing: **Groute**[PPOPP'17], **SEP-Graph**[PPOPP'19]

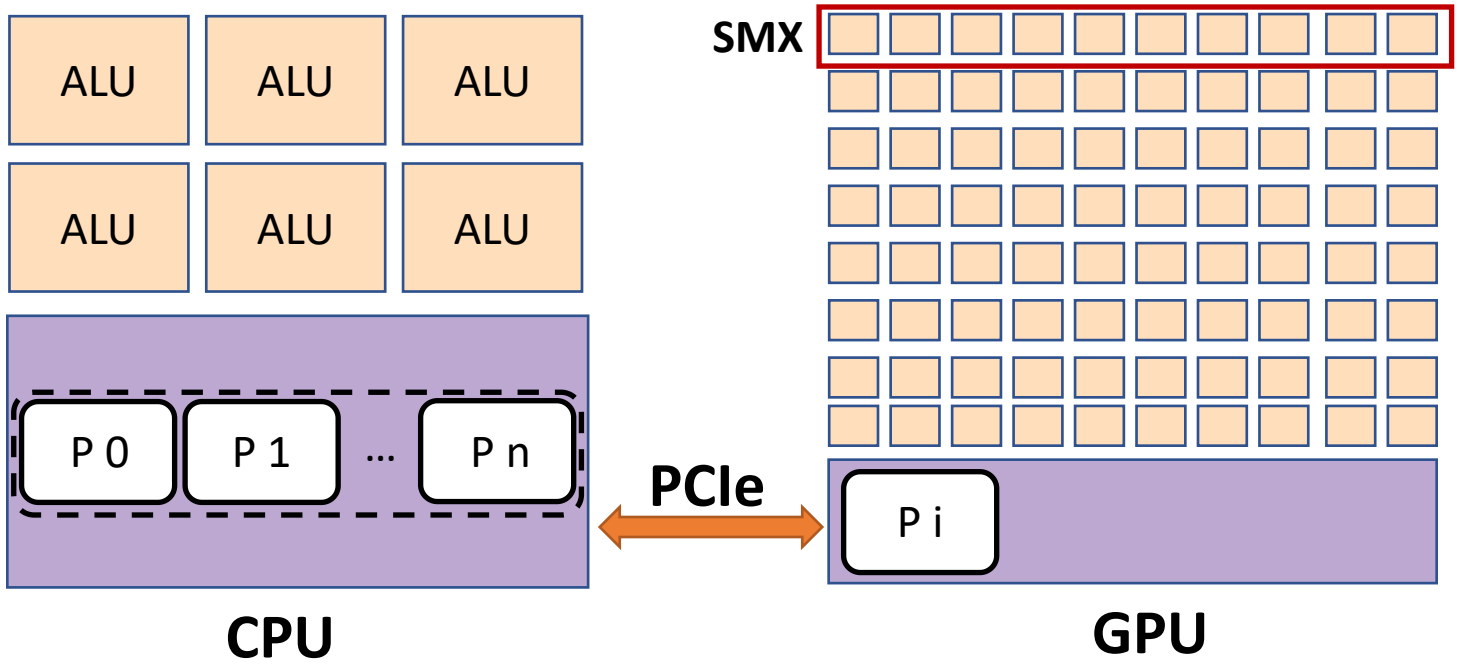
GPU Device Memory

GPU	Dev. Mem.
GTX2080Ti	11GB
Tesla-T4	16GB
Tesla-V100	16GB
GTX-3090Ti	24GB

Dataset description

Graph	|V|	|E|	Size
Twitter-14	52.5M	1.96B	32GB
Web-uk-07	105.1M	3.31B	55GB
Friendster-S	65.6M	3.61B	58GB
weibo-13	72.4M	6.43B	127GB

GPU-accelerated Graph Processing

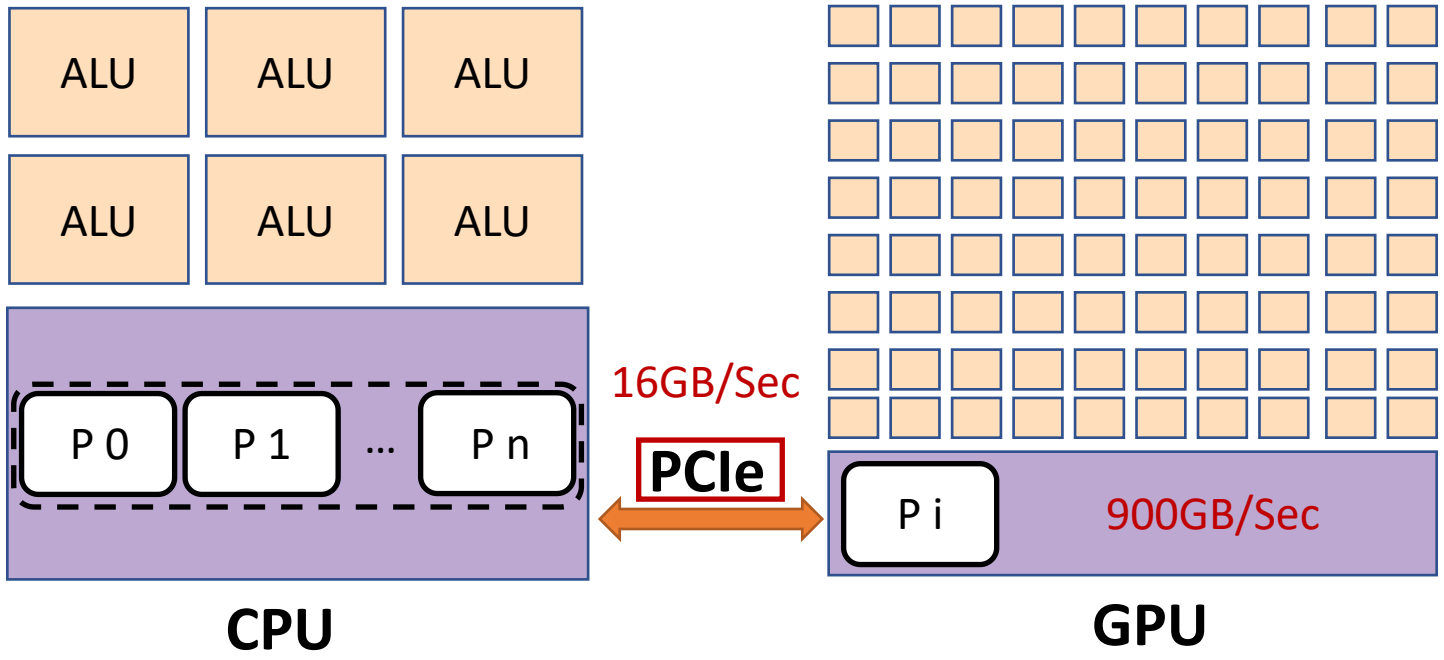


① Partitioning the graph in the host memory

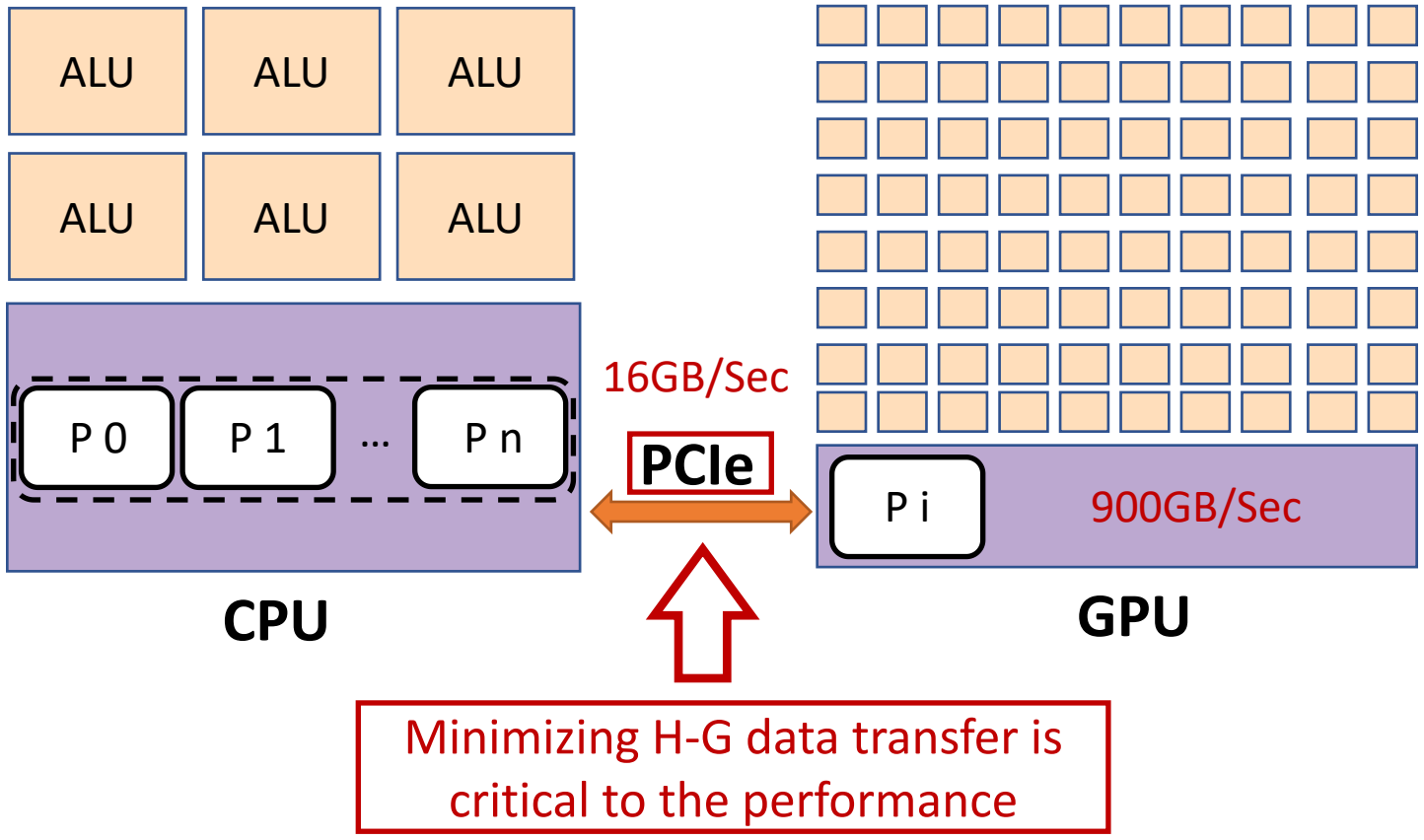
② Loading partitions

③ Parallel processing

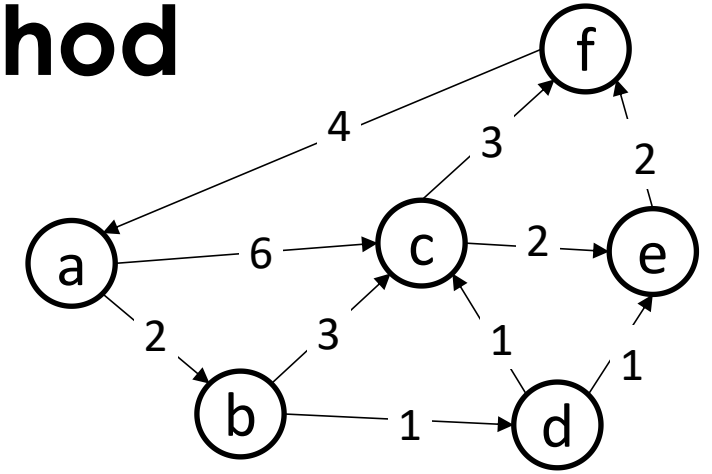
The Key Challenge: Data Transfer



The Key Challenge: Data Transfer



The Existing Transfer Reduction Method

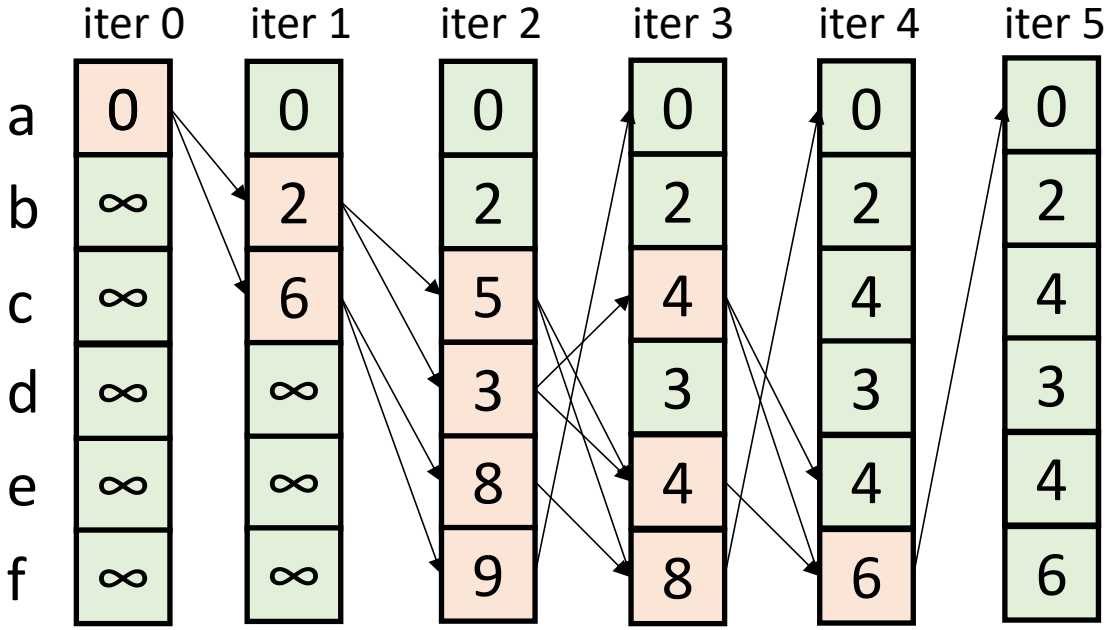


- **Explicit Transfer Management method**

GraphReduce [SC'15] **Graphie** [ATC'19], **GTS** [SIGMOD'16]
Scaph[ATC'20], **Subway** [Eurosys'20], **Ascetic** [ICPP'21]

- **Implicit Transfer Management method**

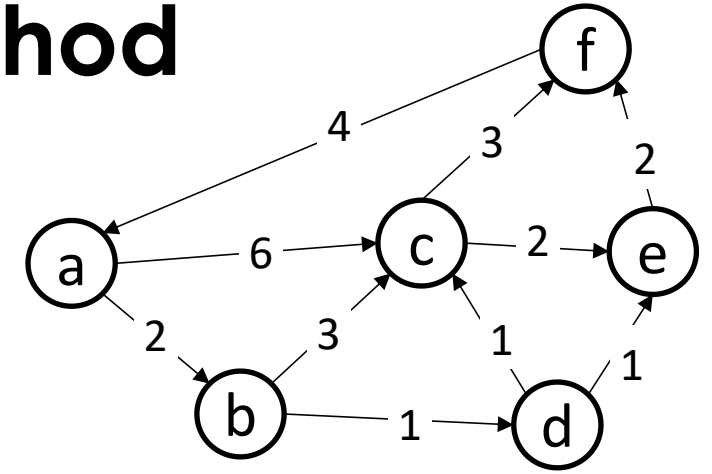
HALO [VLDB'20], **Grus**[TACO'21], **EMOGI** [VLDB'20]



The Existing Transfer Reduction Method

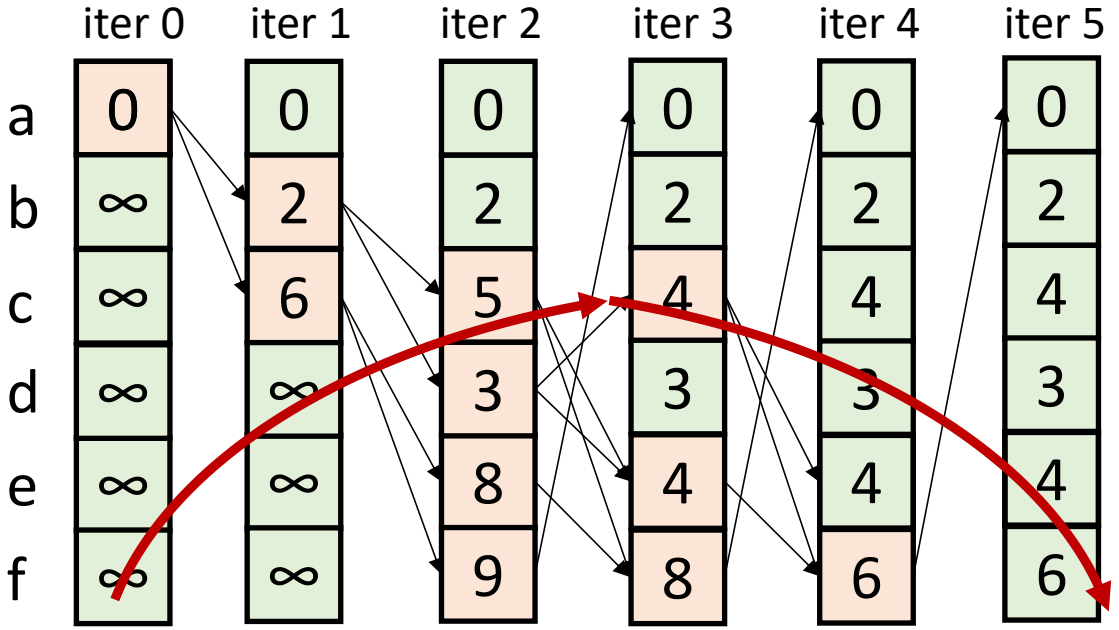
- **Explicit Transfer Management method**

GraphReduce [SC'15] **Graphie** [ATC'19], **GTS** [SIGMOD'16]
Scaph[ATC'20], **Subway** [Eurosys'20], **Ascetic** [ICPP'21]



- **Implicit Transfer Management method**

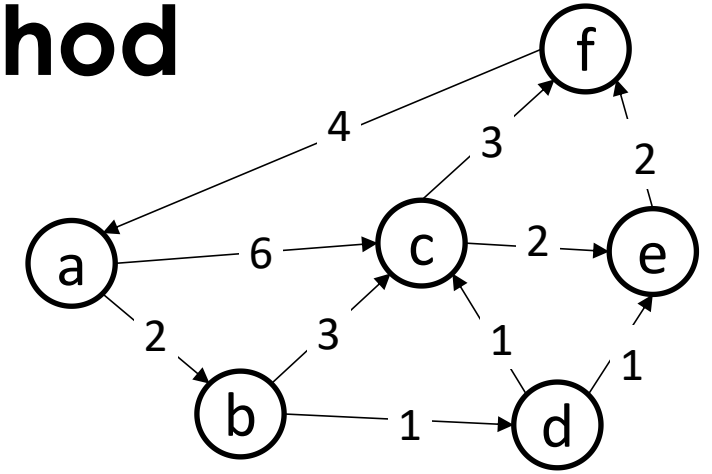
HALO [VLDB'20], **Grus**[TACO'21], **EMOGI** [VLDB'20]



The Existing Transfer Reduction Method

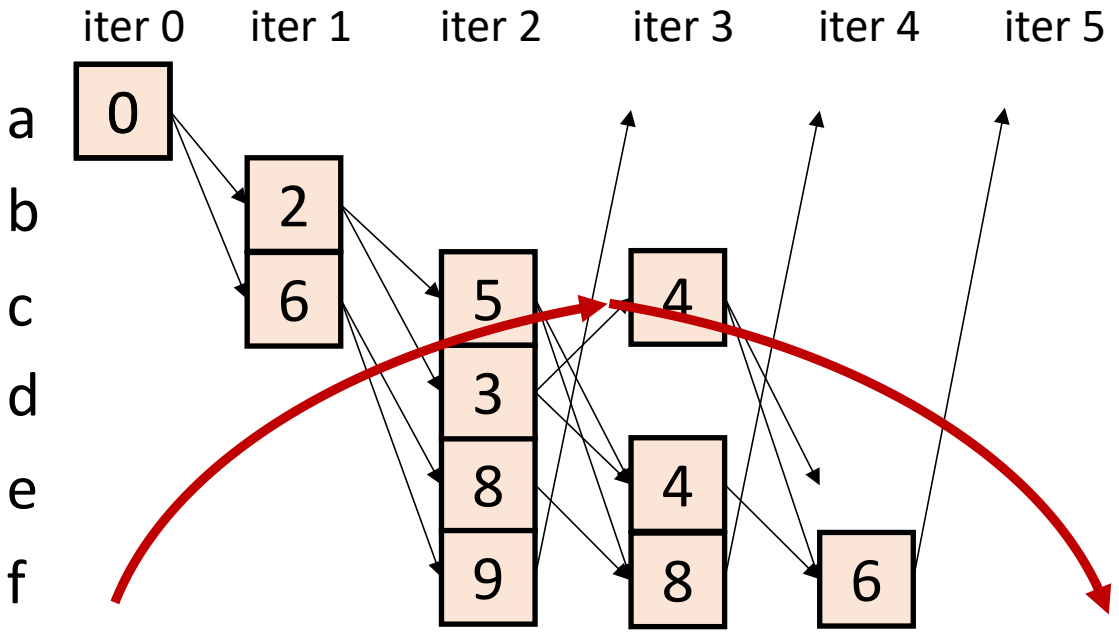
- **Explicit Transfer Management method**

GraphReduce [SC'15] **Graphie** [ATC'19], **GTS** [SIGMOD'16]
Scaph[ATC'20], **Subway** [Eurosys'20], **Ascetic** [ICPP'21]



- **Implicit Transfer Management method**

HALO [VLDB'20], **Grus**[TACO'21], **EMOGI** [VLDB'20]



Explicit Transfer Management

Explicit Transfer Management method

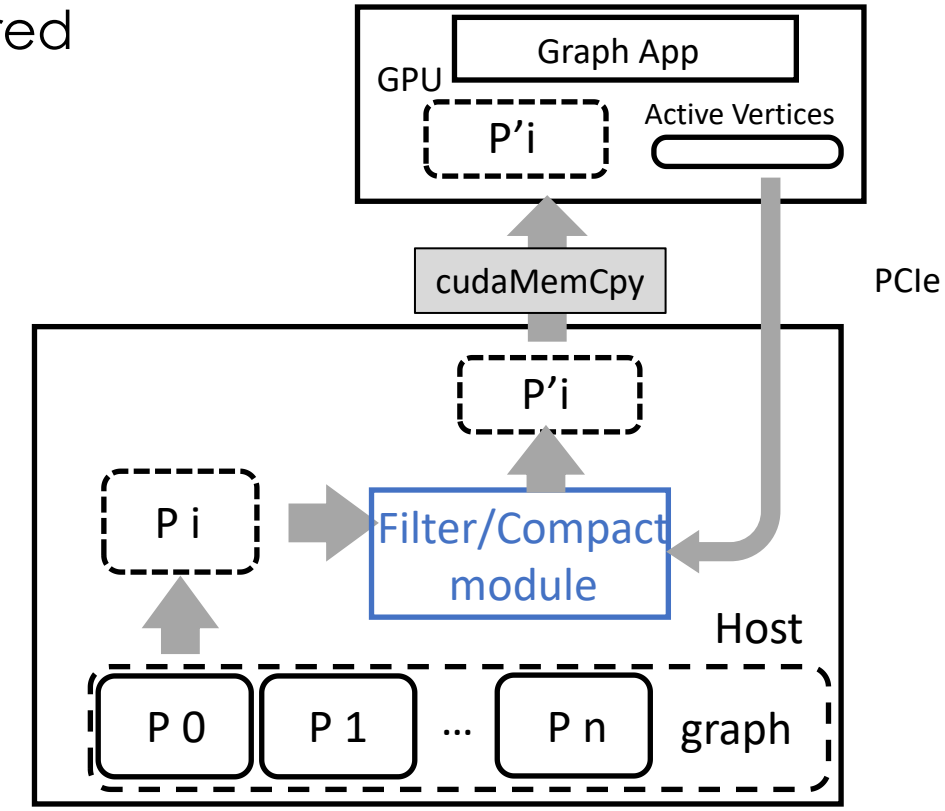
Using CPU(s) to actively compact/ filter to-be-transferred partitions on the host side.

GraphReduce [SC'15] **Graphie** [ATC'19], **GTS** [SIGMOD'16]
Scaph[ATC'20], **Subway** [Eurosys'20], **Ascetic** [ICPP'21]

Implicit Transfer Management method

Using unified memory or zero-copy (DMA) memory to manage the graph data, which enables GPU to transfer the required subgraph automatically.

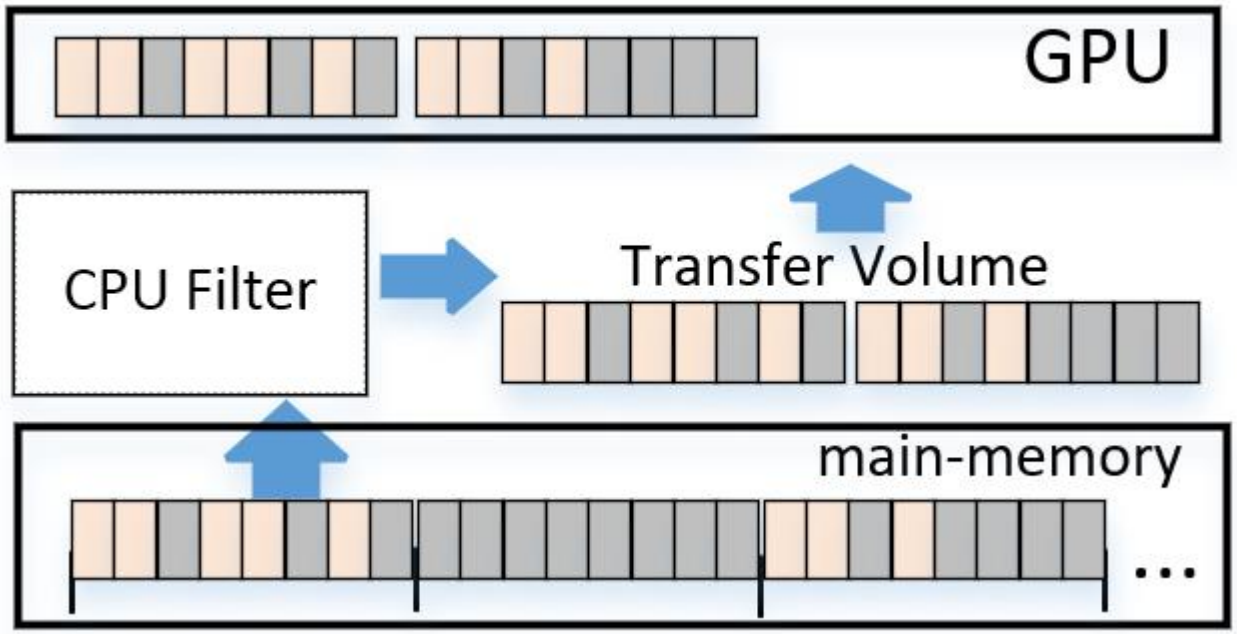
HALO [VLDB'20], **Grus**[TACO'21] **EMOGI** [VLDB'20]

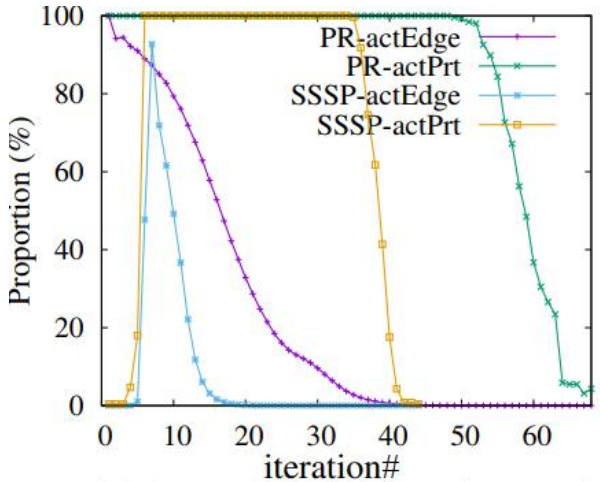


Explicit Transfer Management with Filtering

Identifying and transferring those partitions containing active edges

- CPU-based Filtering (with almost no cost)
- Explicit Memory Copy (*cudaMemCpy()*)



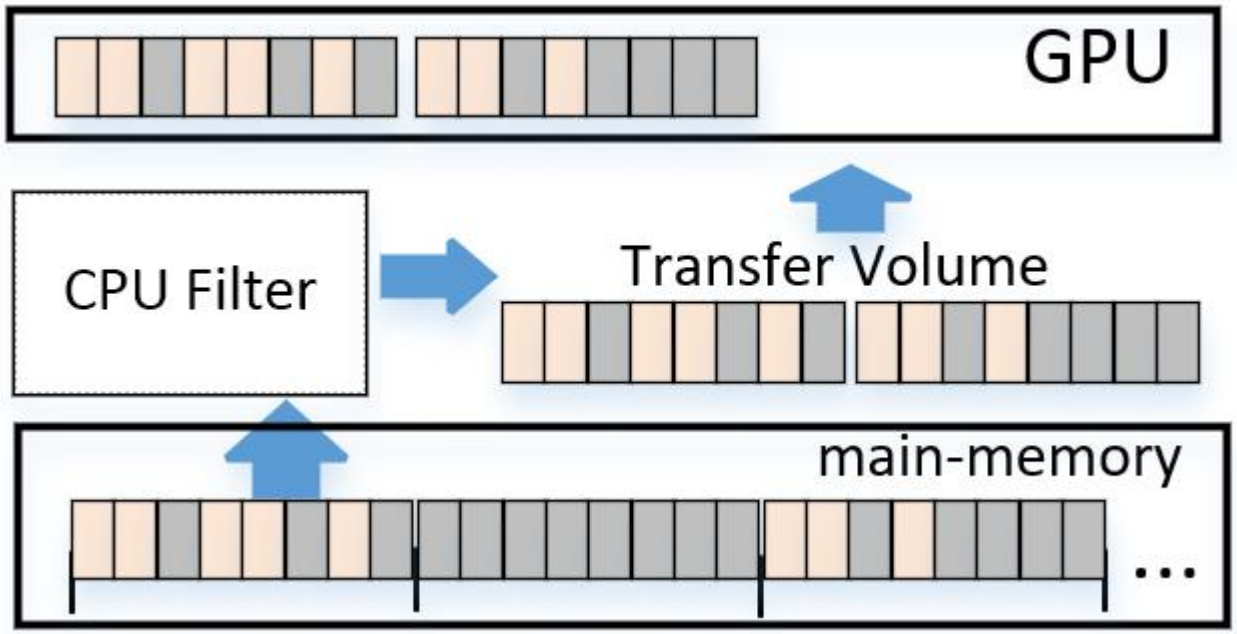


(a) Proportion of active edges and active partitions in ExpTM-filter

Explicit Transfer Management with Filtering

Identifying and transferring those partitions containing active edges

- CPU-based Filtering (with almost no cost)
- Explicit Memory Copy (*cudaMemCpy()*)

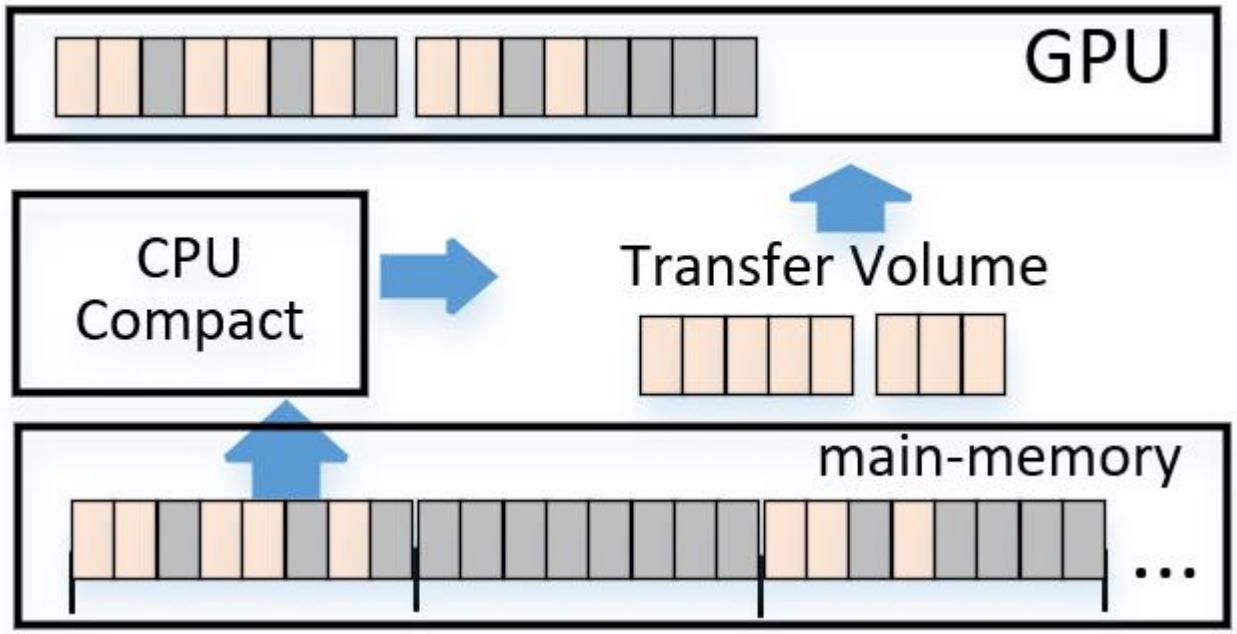


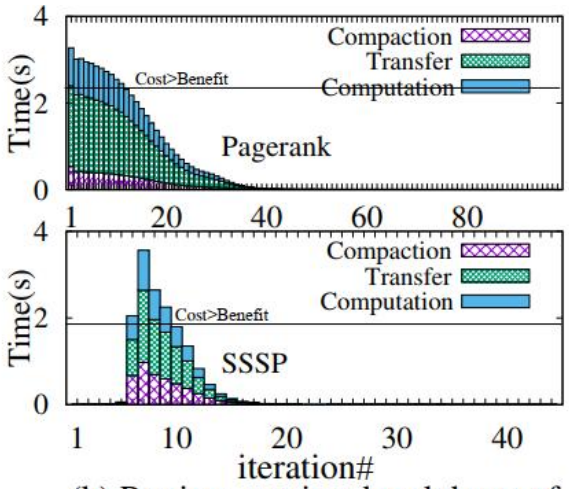
- High bandwidth utilization
- Low CPU processing overhead
- Heavy redundant data transfer

Explicit Transfer Management with Compaction

Compacting the active edges for each partition and transferring the shrunk partition.

- CPU-based Compaction
- Explicit Memory copy (*cudaMemCpy()*)



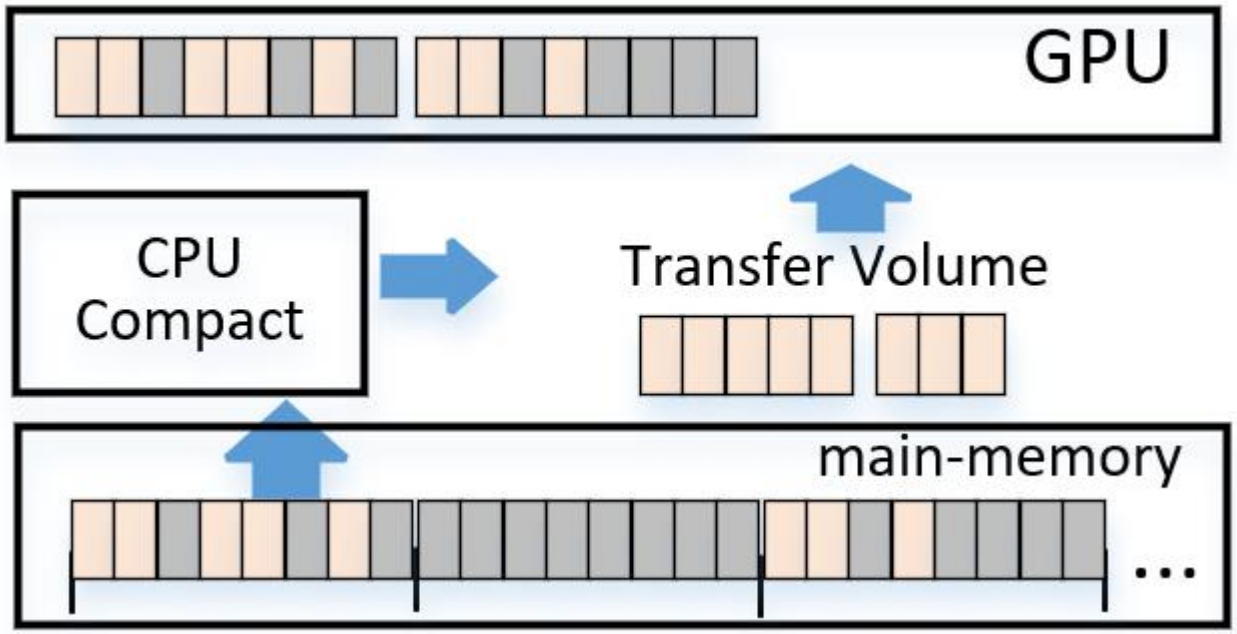


(b) Per-iter. runtime breakdown of ExpTM-compaction (Subway)

Explicit Transfer Management with Compaction

Compacting the active edges for each partition and transferring the shrunk partition.

- CPU-based Compaction
- Explicit Memory copy (*cudaMemCpy()*)



- High Bandwidth Utilization
- No redundant data transfer
- Expensive CPU processing overhead

Implicit Transfer Management

Explicit Transfer Management method

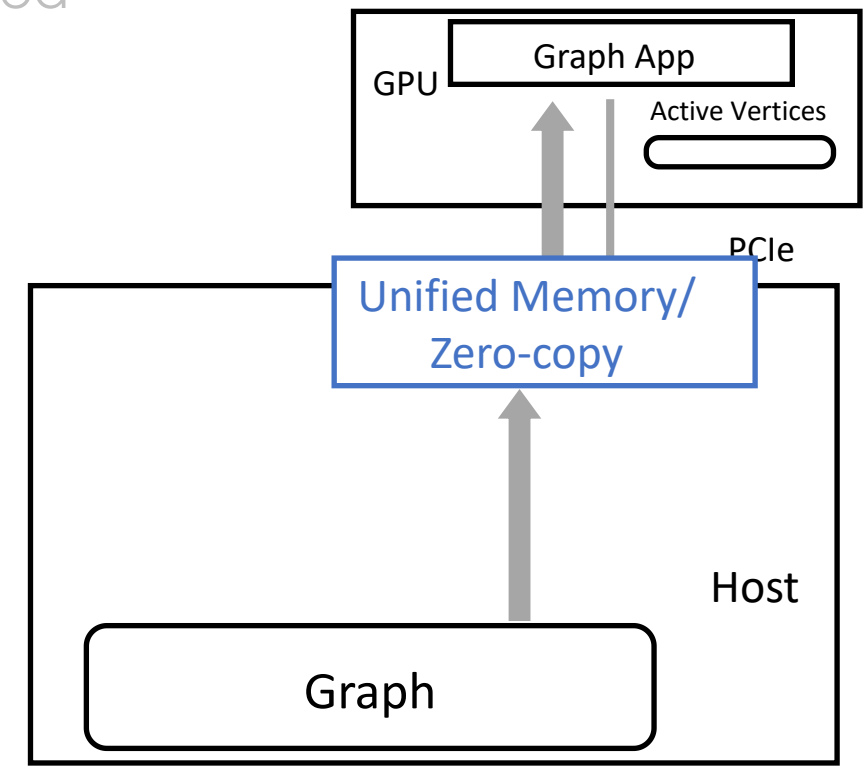
Using CPU(s) to actively compact/ filter to-be-transferred partitions on the host side.

GraphReduce [SC'15] **Graphie** [ATC'19], **GTS** [SIGMOD'16]
Scaph[ATC'20], **Subway** [Eurosys'20], **Ascetic** [ICPP'21]

Implicit Transfer Management method

Using unified memory or zero-copy (DMA) memory to manage the graph data, which enables GPU to transfer the required subgraph automatically.

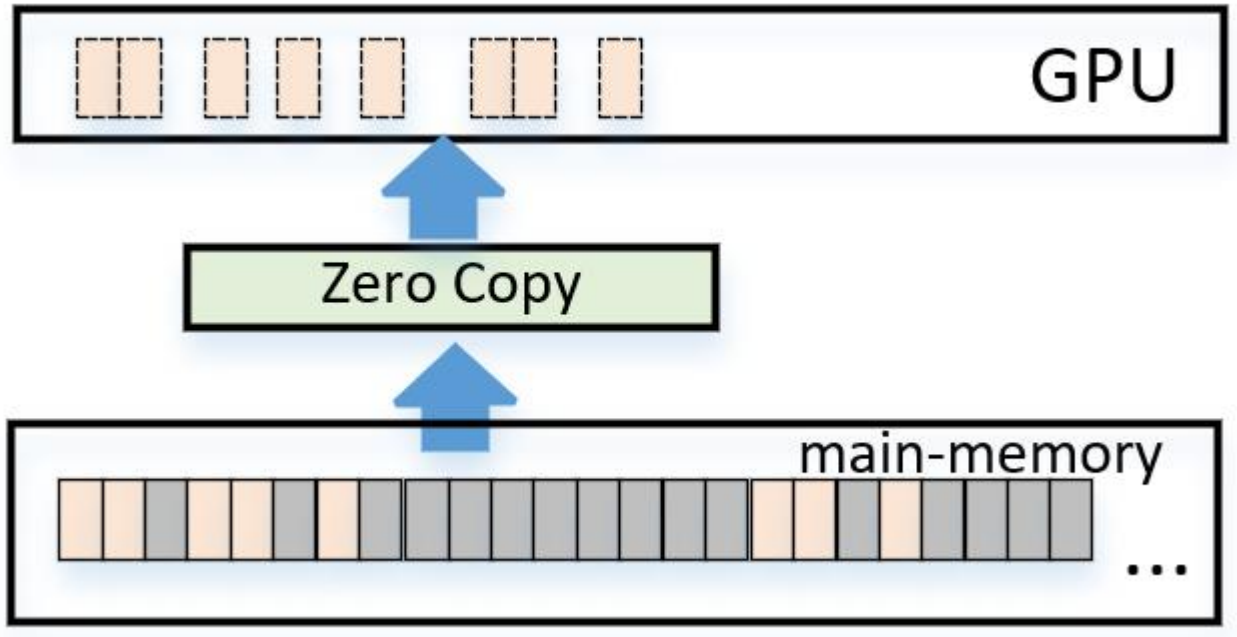
HALO [VLDB'20], **Grus**[TACO'21] **EMOGI** [VLDB'20]

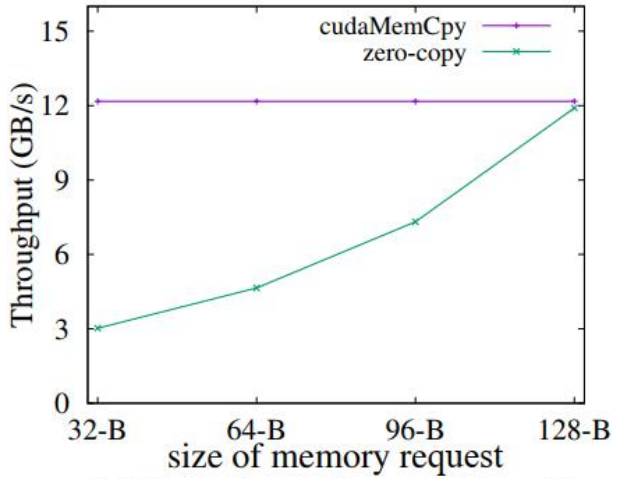


Implicit Transfer Management with Zero-Copy

Directly accessing the host graph with PCIe requests (DMA).

- Relying on the Transaction Layer Packet (TLP) of PCIe to achieve fine-grained on-demand memory access.



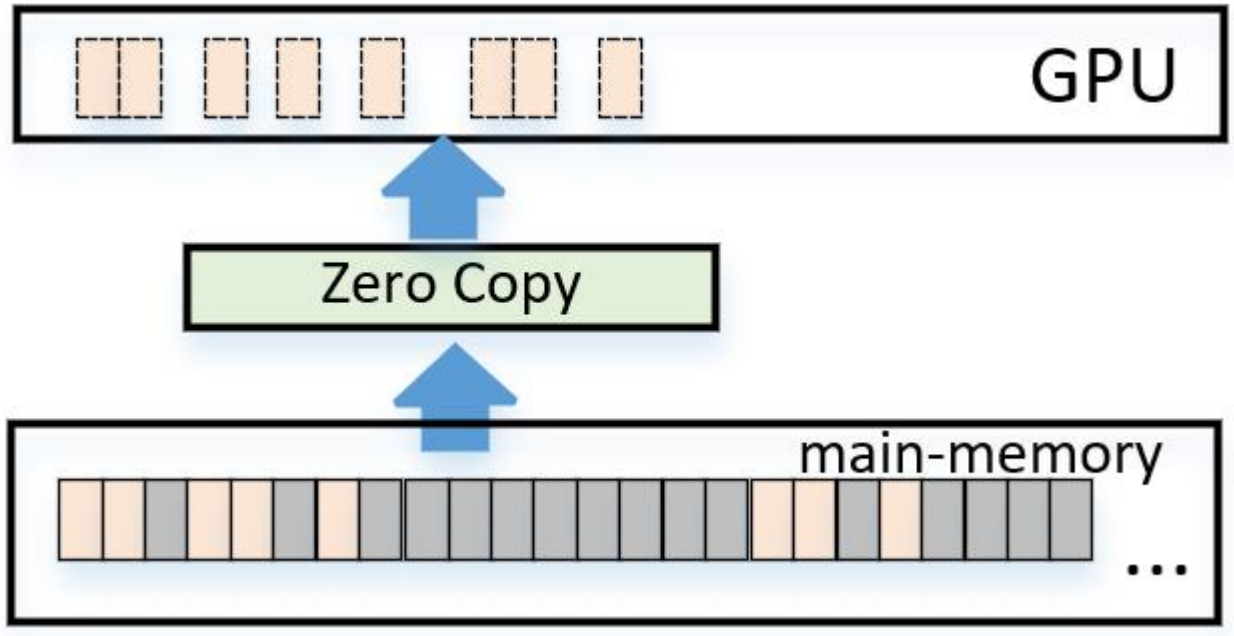


(e) Throughput of zero-copy with different access granularity

Implicit Transfer Management with Zero-Copy

Directly accessing the host graph with PCIe requests (DMA).

- Relying on the Transaction Layer Packet (TLP) of PCIe to achieve fine-grained on-demand memory access.

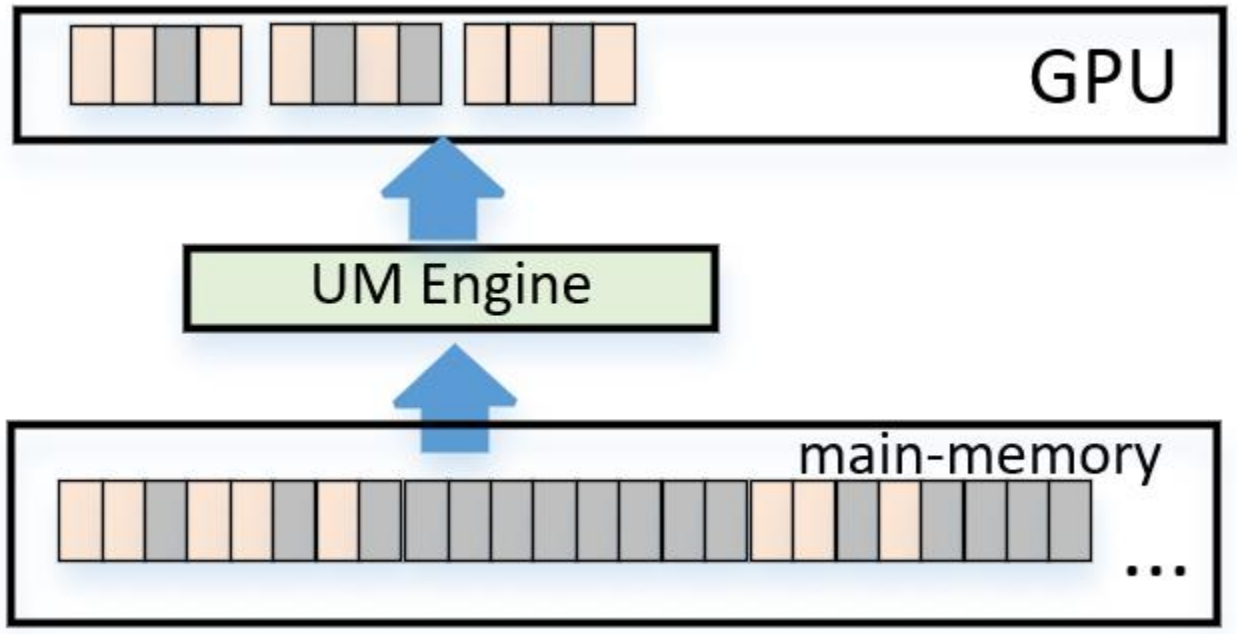


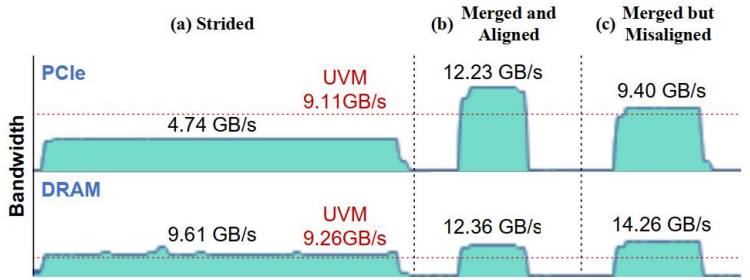
- No CPU processing overhead
- Negligible redundant data transfer
- Unstable bandwidth utilization

Implicit Transfer Management with UVM

Transferring the subgraph implicitly with unified-virtual-memory (UVM).

- UVM enables GPU to cache transferred data with device memory
- UVM can not fully utilize the PCIe bandwidth



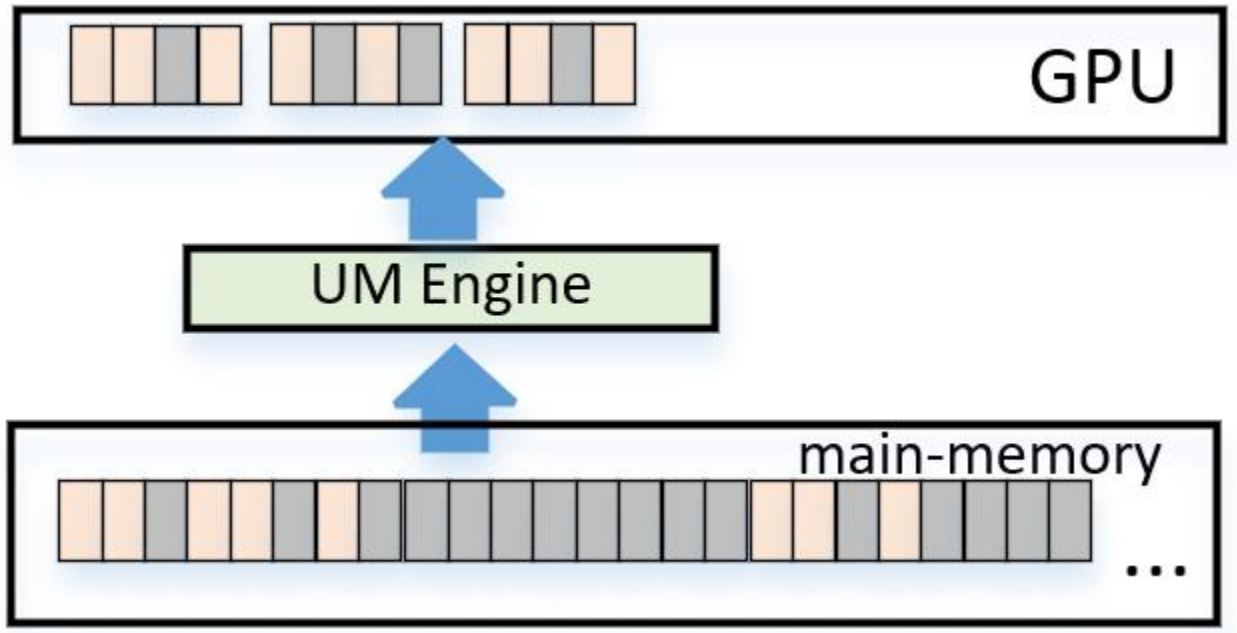


UVM bandwidth utilization [EMOGI:VLDB'20]

Implicit Transfer Management with UVM

Transferring the subgraph implicitly with unified-virtual-memory (UVM).

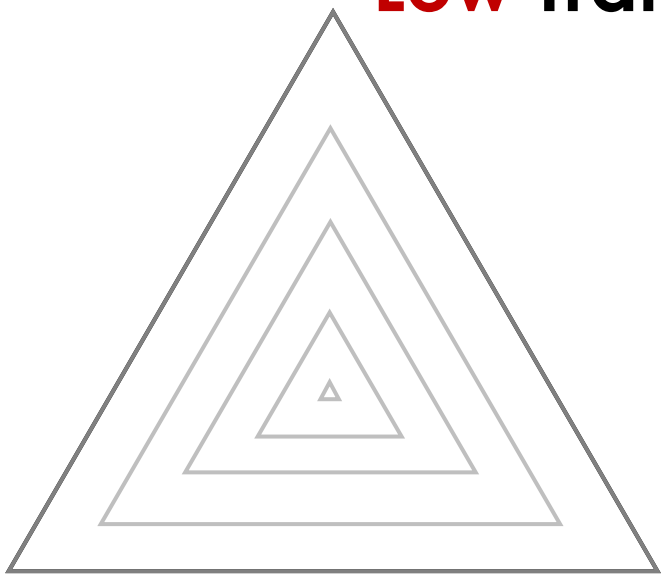
- UVM enables GPU to cache transferred data with device memory
- UVM can not fully utilize the PCIe bandwidth



- No CPU processing overhead
- Medium redundant data transfer
- Low bandwidth utilization

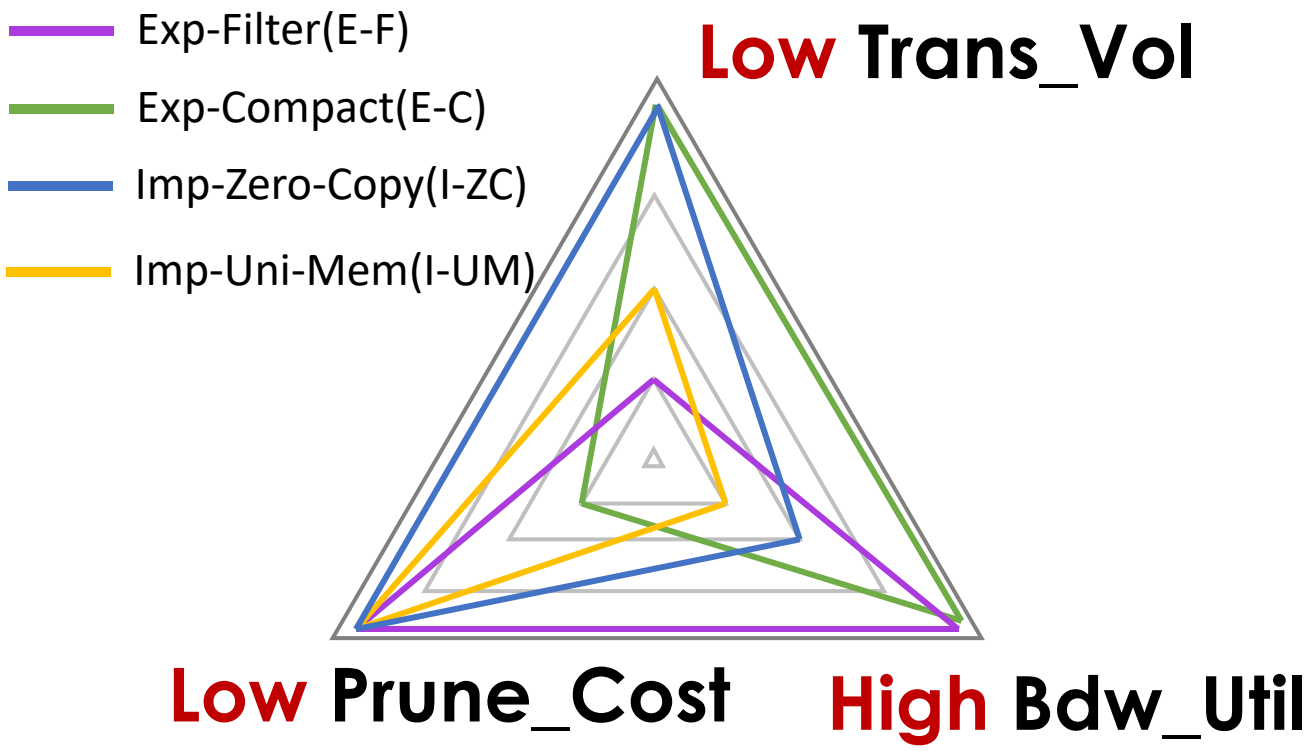
Comparison of the Existing Approaches

Total Transfer Cost= **Trans_Vol.**/(Bdw.*__Bdw_Util.__)+ **Prune_Cost**



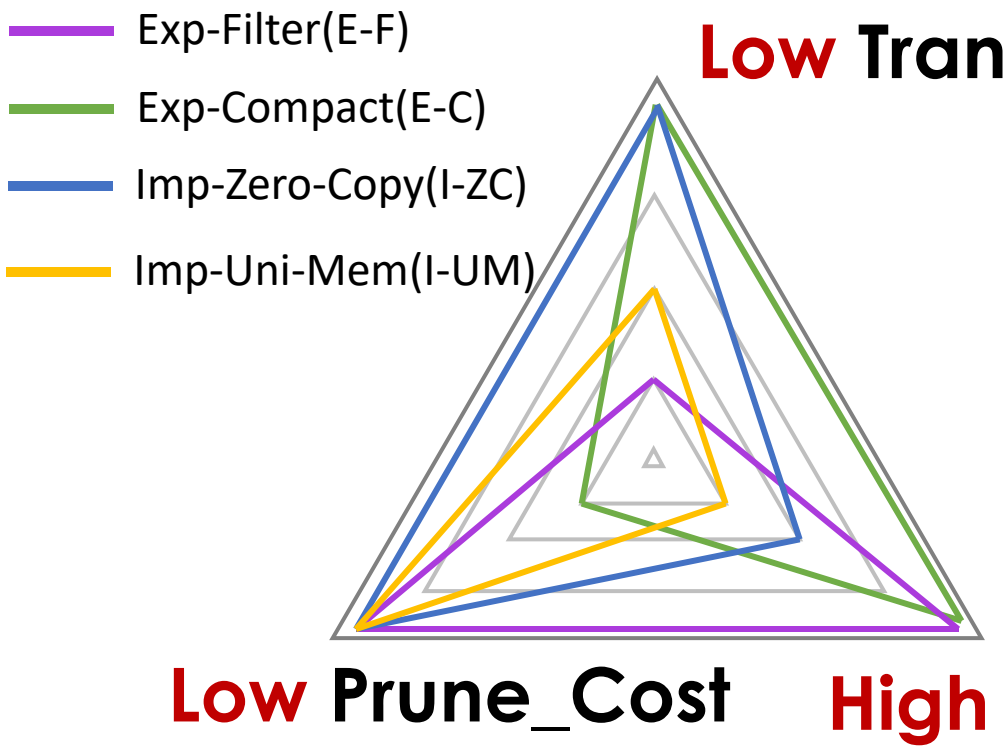
Comparison of the Existing Approaches

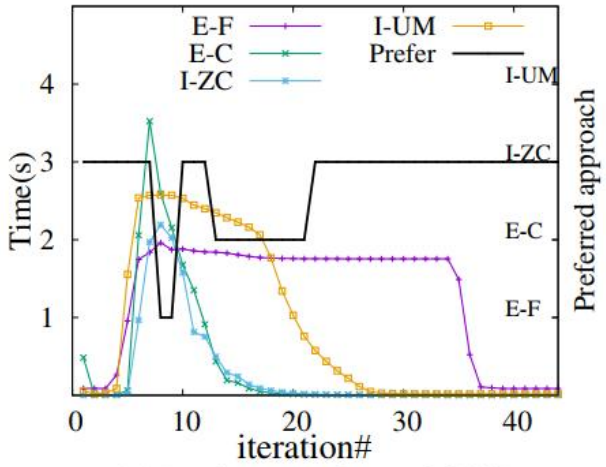
Total Transfer Cost= **Trans_Vol.**/(Bdw.***Bdw_Util.**)+ **Prune_Cost**



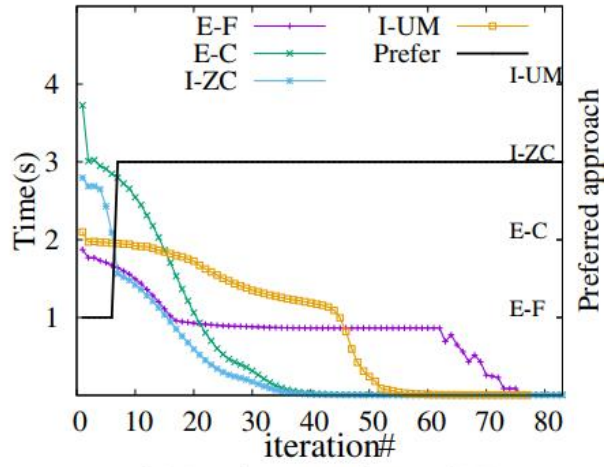
Comparison of the Existing Approaches

Total Transfer Cost= **Trans_Vol.**/(Bdw.*__Bdw_Util.__)+ **Prune_Cost**





(g) Per-iter. runtime of SSSP with the four approaches



(h) Per-iter. runtime of PR with the four approaches

Our Approach: Hybrid Transfer Management

Adaptively selecting the transfer method for each partition by using:

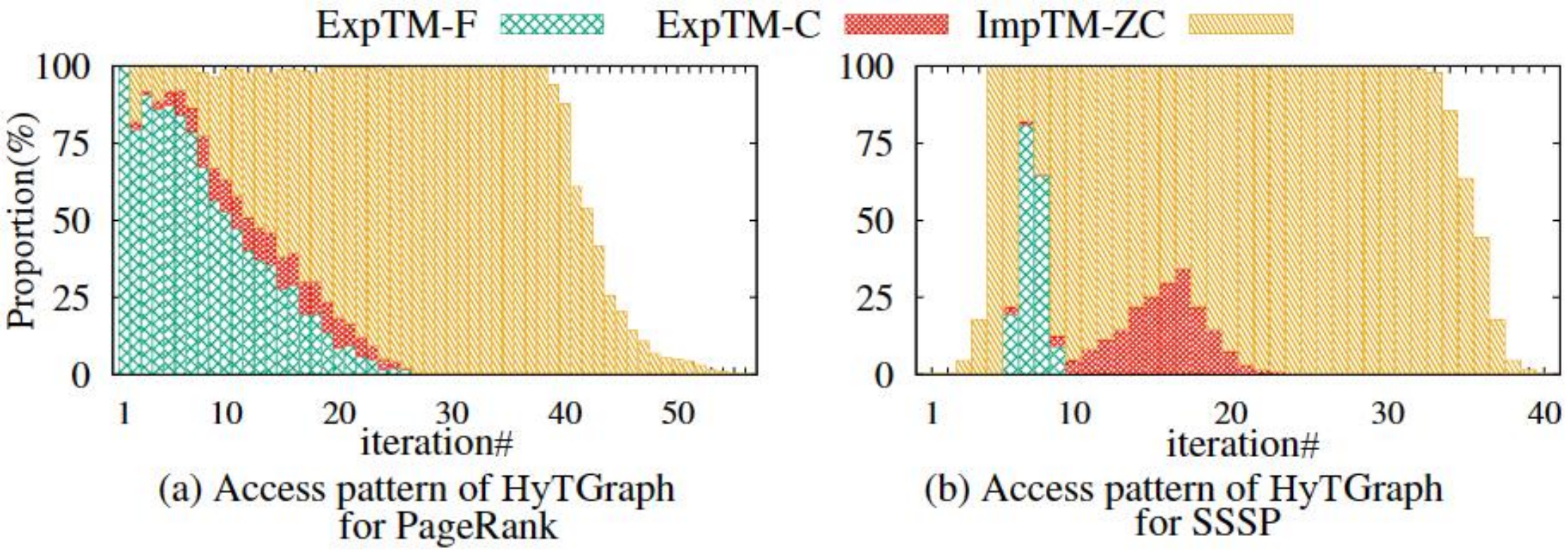
- Trans_Cost= Trans_Vol./(Bdw.*Bdw_Util.)+ Prune_Cost

(a) ExpTM-Filter
- Trans_Cost(E-F) = Trans_Vol. / Bdw.

(b) ExpTM-Compaction
- Trans_Cost(E-C) = Trans_Vol. / Bdw. + Prune_Cost

(c) ImpTM-Zero-Copy
- Trans_Cost(I-ZC) = Trans_Vol. / (Bdw. * Bdw_Util.)

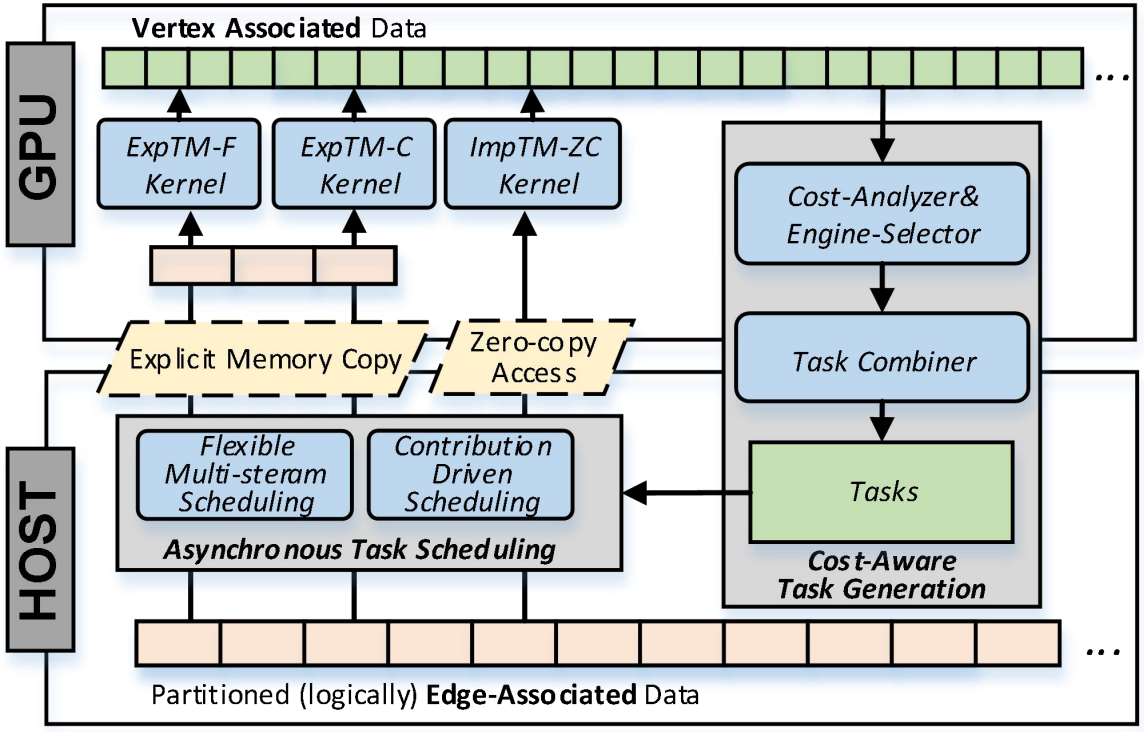
Our Approach: Hybrid Transfer Management



HyTGraph

HyTGraph uses a partitioning-based approach to manage the transfer

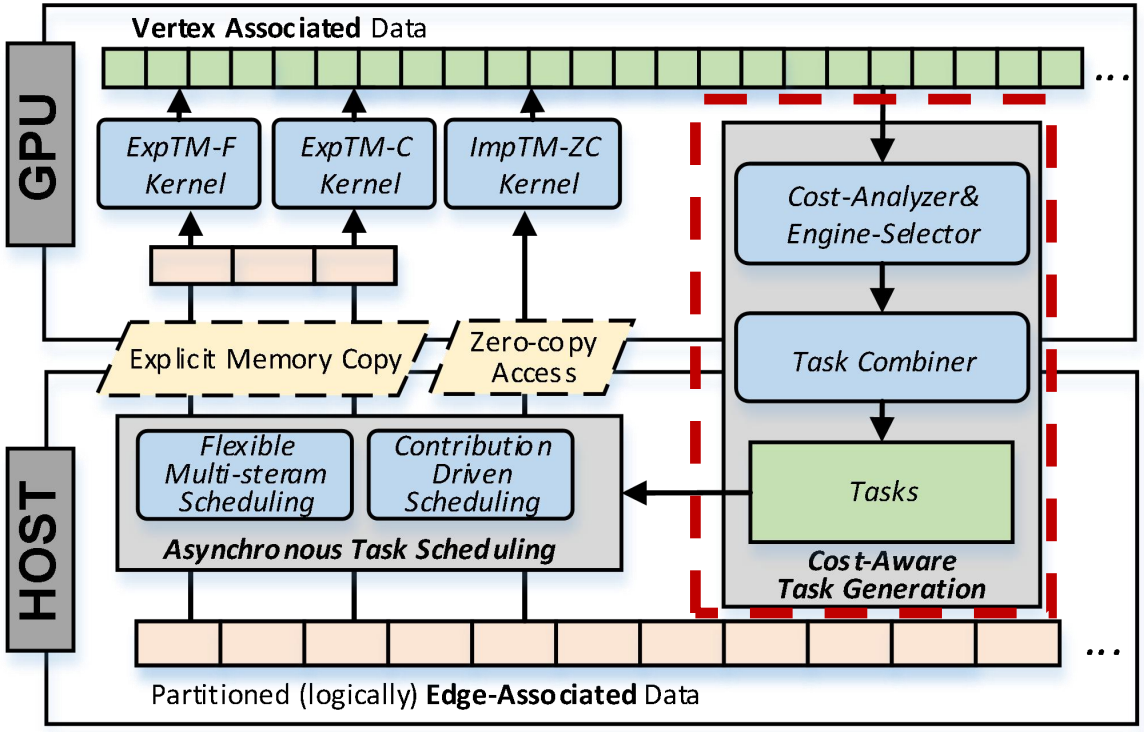
- Cost-Aware Task Generation:
  1. Cost Analyzer
  2. Engine Selector
  3. Task Combiner
- Asynchronous Task Scheduling:
  1. Contribution Driven Scheduling
  2. Multi-Stream Scheduling



HyTGraph

HyTGraph uses a partitioning-based approach to manage the transfer

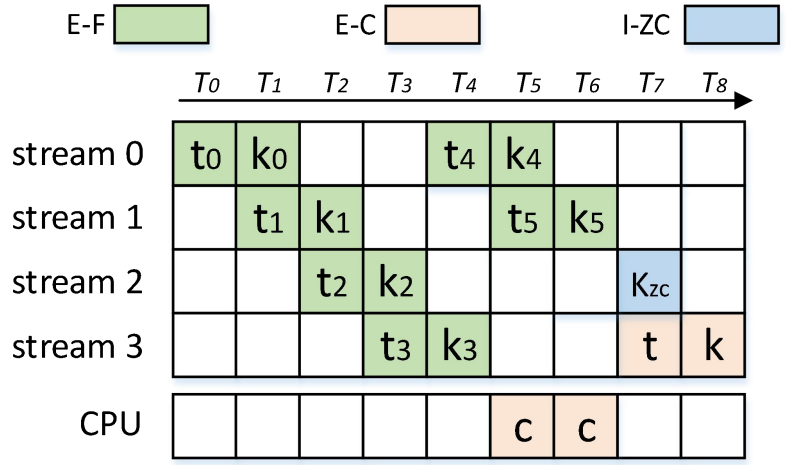
- **Cost-Aware Task Generation:**
  1. Cost Analyzer
  2. Engine Selector
  3. Task Combiner
- Asynchronous Task Scheduling:
  1. Contribution Driven Scheduling
  2. Multi-Stream Scheduling

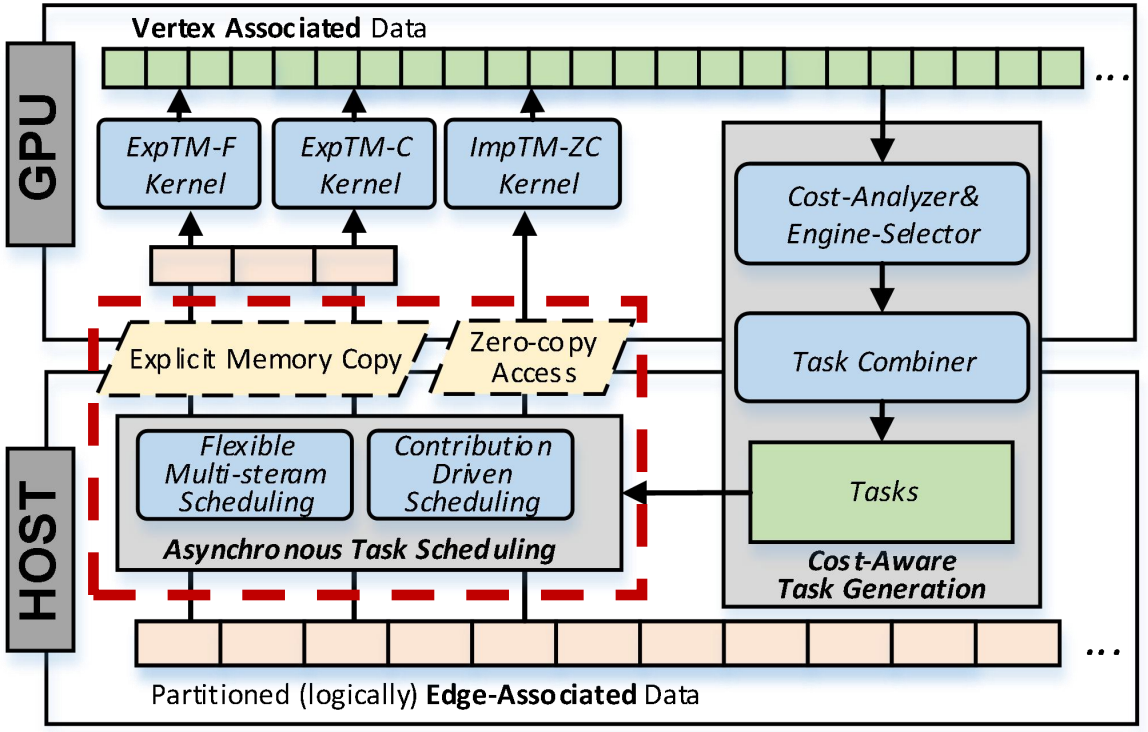


HyTGraph

HyTGraph uses a partitioning-based approach to manage the transfer

- Cost-Aware Task Generation:
  1. Cost Analyzer
  2. Engine Selector
  3. Task Combiner
- **Asynchronous Task Scheduling:**
  1. Contribution Driven Scheduling
  2. Multi-Stream Scheduling





Experimental Setting

Competitors: Grus [TACO'21], **Subway** [Eurosys'20], **EMOGI** [VLDB'20]. **ExpTM-Filter** (SEP-GRAPH), **ImpTM-UM**(SEP-GRAPH)

Test Platforms:

Intel Silver 4210 10-core CPU, 128GB DRAM, 1 NVIDIA-GTX 2080Ti GPU(34 SMXs, 4352 cores, 11GB GDDR6 RAM)

Algorithms and Datasets:

- 4 graph analytical algorithms: BFS, CC, SSSP, PageRank
- 5 real world graphs, and 1 synthesized graph with **RMAT**

Softeware Environment:

- Ubuntu 18.04 LTS
- CUDA 10.1 (418.67 driver)

TABLE II: Dataset description.

Dataset	|V|	|E|	|E|/|V|	Size
sk-2005 [2] (SK)	50.6M	1.93B	38	28GB
twitter [1] (TW)	52.5M	1.96B	37	32GB
friendster-konect [1] (FK)	68.3M	2.59B	37	42GB
uk-2007 [2] (UK)	105.1M	3.31B	31	55GB
friendster-snap [3] (FS)	65.6M	3.61B	55	58GB
RMAT [7]	1-100M	0.1-6.4B	-	-

Overall Results

HyTGraph shows better performance than the competitors on most cases.

- 2.01X-28.5X faster than **ExpTM-F**
- 2.4X-10.3X faster than **ExpTM-C** (**SubWay**)
- 1.1X-6.5X faster than **ImpTM-ZC** (**EMOGI**)
- 2.4-13.1X faster than **ImpTM-UM** (**Grus**)

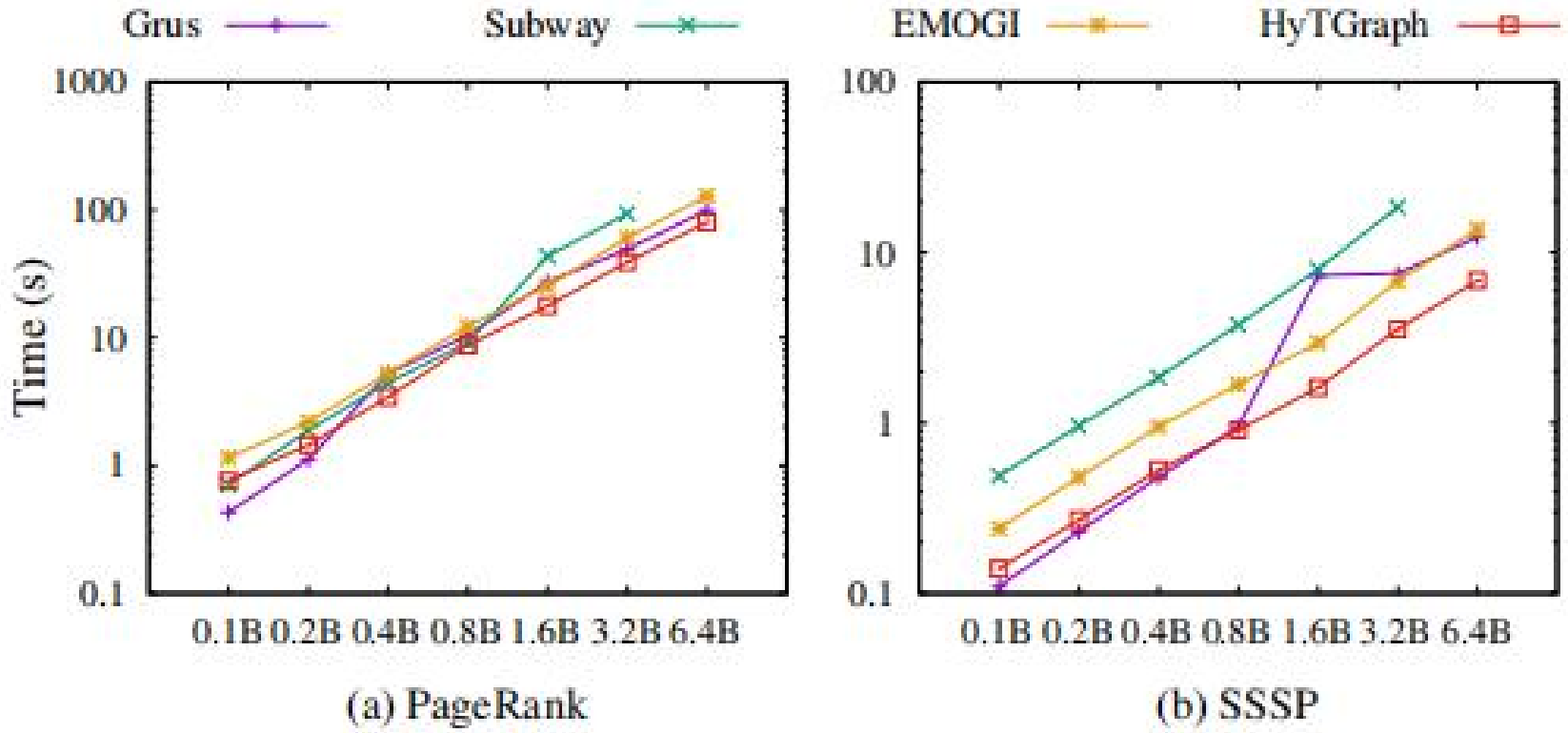
Overall runtime (s)						
Alg.	System	SK	TW	FK	UK	FS
PR	Galois	21.3	66.3	293.6	28.5	342.4
	ExpTM-F	37.7	34.8	60.7	34.3	162.8
	ImpTM-UM	6.89	16.5	75.4	22.4	102.7
	Grus	**1.72**	12.2	52.2	14.8	79.8
	Subway	8.68	38.1	73.7	16.9	108.4
	EMOGI	18.6	21.4	51.1	12.4	68.3
	HyTGraph	2.85	**11.5**	**30.1**	**4.71**	**40.8**
SSSP	Galois	26.7	12.9	51.5	15.2	33.1
	ExpTM-F	60.9	15.1	50.4	60.9	70.1
	ImpTM-UM	12.7	10.1	37.2	18.6	34.9
	Grus	25.2	11.2	70.8	5.32	16.9
	Subway	14.6	10.9	20.8	18.4	27.7
	EMOGI	7.46	4.09	14.9	4.71	11.8
	HyTGraph	**6.11**	**2.09**	**8.81**	**2.78**	**6.64**
CC	Galois	23.9	15.7	35.9	55.1	39.4
	ExpTM-F	21.9	5.47	10.9	41.6	11.8
	ImpTM-UM	**1.43**	1.49	3.27	7.88	4.16
	Grus	2.09	1.36	3.21	5.17	4.69
	Subway	11.67	6.52	8.61	14.7	14.1
	EMOGI	4.01	1.96	2.71	4.54	3.76
	HyTGraph	3.65	**1.19**	**2.01**	**3.86**	**2.59**
BFS	Galois	16.2	7.55	12.5	15.2	14.7
	ExpTM-F	20.3	3.86	8.87	25.1	9.54
	ImpTM-UM	1.13	1.29	1.97	2.33	6.25
	Grus	**0.83**	1.11	1.85	2.37	3.35
	Subway	7.39	5.79	6.85	9.04	13.49
	EMOGI	1.06	1.04	**1.44**	1.26	**1.97**
	HyTGraph	0.93	**0.85**	1.82	**0.88**	2.54

Transfer Reduction Analysis

Transfer volume / Edge volume					
Alg.	Dataset	ExpTM-F	Subway	EMOGI	HyTGraph
PR	SK	57.6X	2.46X	3.31X	**2.17X**
	TW	52.4X	**5.48X**	20.6X	10.9X
	FK	58.3X	**10.74**X	24.6X	12.01X
	UK	30.9X	1.79X	3.81X	**1.68X**
	FS	121.6X	**12.44X**	25.23X	12.62X
SSSP	SK	44.3X	4.23X	3.29X	**3.25X**
	TW	11.2X	2.07X	1.74X	**1.25**X
	FK	28.1X	**3.32**X	4.81X	4.60X
	UK	24.3X	1.78X	**1.11X**	1.13X
	FS	24.1X	3.19X	2.69X	**2.52X**
		45.3X	4.8X	9.1X	5.1X

- **HyTGraph** effectively reduces the data transfer, and its performance is close to the SOTA (**Subway**).
- **HyTGraph** achieves minimum data transfer on more graphs.

Scaling-Up Performance



The experiments on synthesized graphs show that **HyTGraph** shows the best scaling-up performance when expanding the graph size by 64 times.

PageRank: Grus(231.2X), Subway(OOM), EMOGI(111.6X), **HyTGraph** (**105.4X**)

SSSP: Grus(111.8X), Subway(OOM), EMOGI(57.1X), **HyTGraph** (**49.0X**)

Thanks for your listening

Questions