ThunderGP: Fast Graph Processing for HLS-based FPGAs

Xinyu Chen[1], Hongshi Tan[1], Yao Chen[2],

Bingsheng He[1], Weng-Fai Wong[1], Deming Chen[3]

[1]National University of Singapore, [2]Advanced Digital Sciences Center,
[3]University of Illinois at Urbana-Champaign

Graph Processing on FPGAs

- **Graph processing is widely used in variety of application domains**
  - Social networks
  - Cybersecurity
  - Machine learning

- **Graph processing on FPGAs benefiting from**
  - Fine grained parallelism
  - Low power consumption
  - Extreme configurability

Graph Processing on HLS-based FPGAs

- **RTL-based FPGAs development**
  - Time-consuming
  - Deep understanding of hardware

- **To ease the use of FPGAs, HLS tools come**
  - High-level programming model
  - Hide hardware details
  - Both Intel and Xilinx have HLS tools

ThunderGP: Fast Graph Processing for HLS-based FPGAs

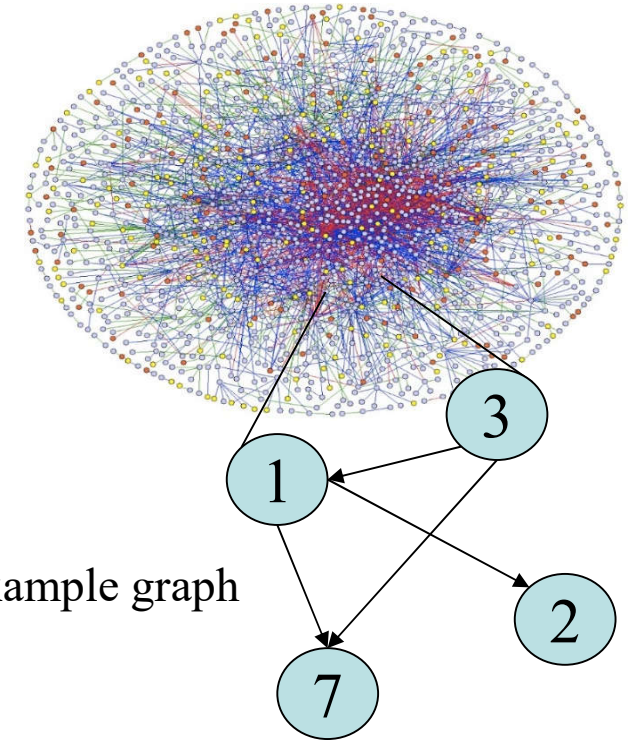
Outline

- On-the-fly Data Shuffling for Graph Processing Framework
- Improved Execution Flow and Memory Optimizations
- Scaling the Framework to Multiple SLRs
- APIs
- Evaluation

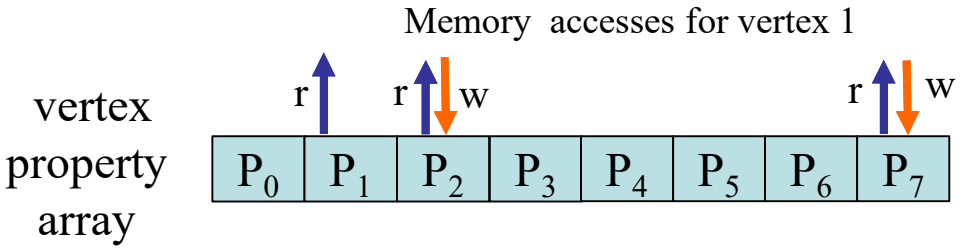
Outline

- **On-the-fly Data Shuffling for Graph Processing Framework**
- Improved Execution Flow and Memory Optimizations
- Scaling the Framework to Multiple SLRs
- APIs
- Evaluation

GAS Model [1] for Graph Processing



Example graph



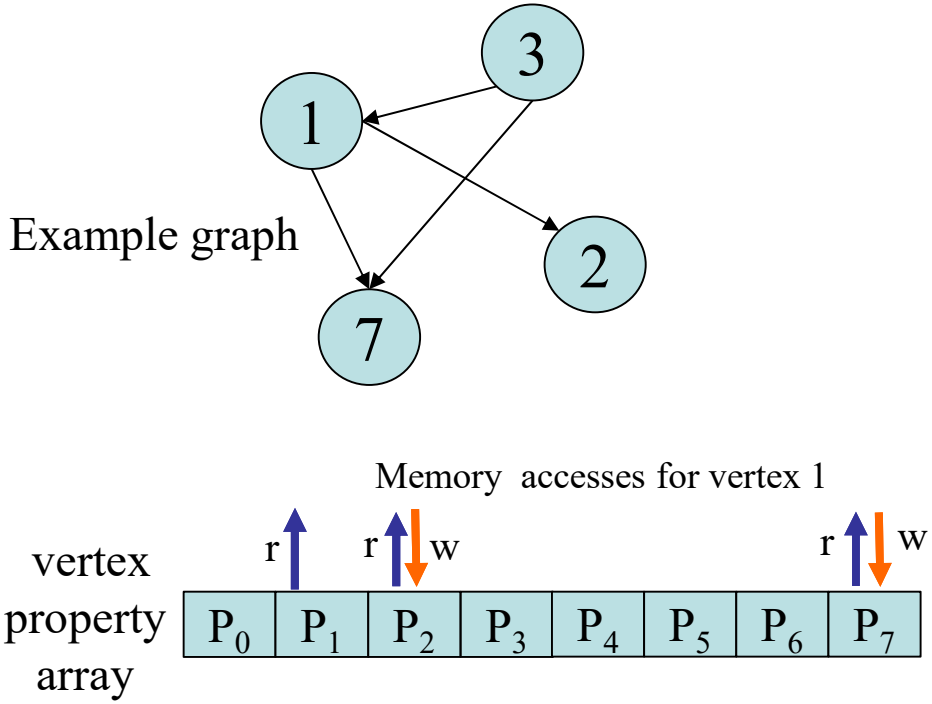
Three stages:

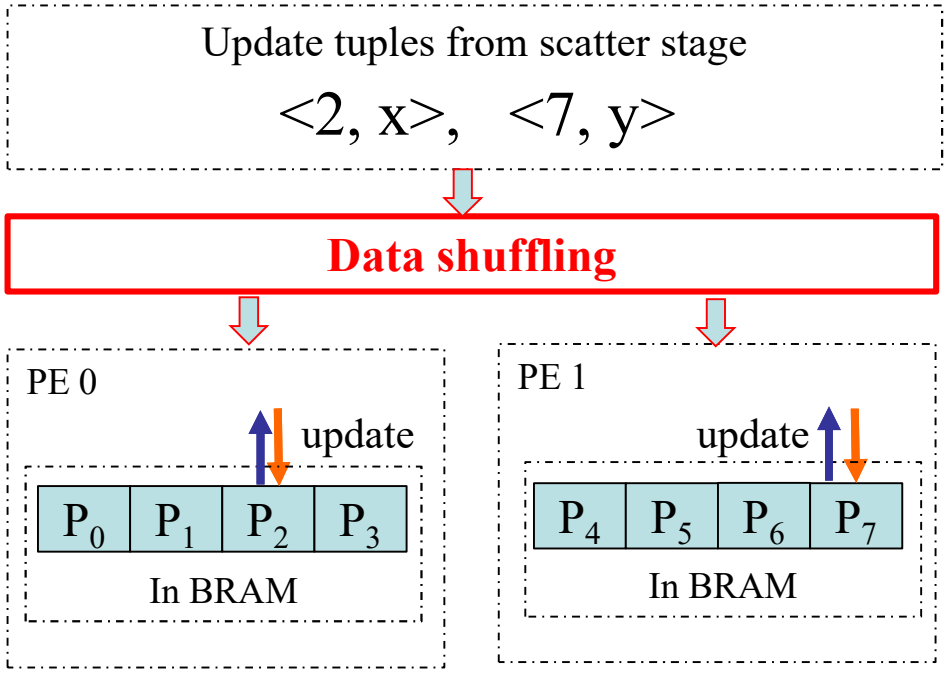
- **Scatter**: for an edge, an update tuple is generated with format of *<destination, value>*
  - E.g. <2, x>, <7, y> for vertex 1
- **Gather:** accumulates values to destination vertices
  - E.g. Op(P_2 , x), Op(P_7 , y)
- **Apply:** an apply function on all the vertices

[1] A. Roy, L. Bindschaedler, J. Malicevic, and W. Zwaenepoel, "Chaos: Scale-out graph processing from secondary storage," in *SOSP*, 2015

Common Techs for GAS Model on FPGAs

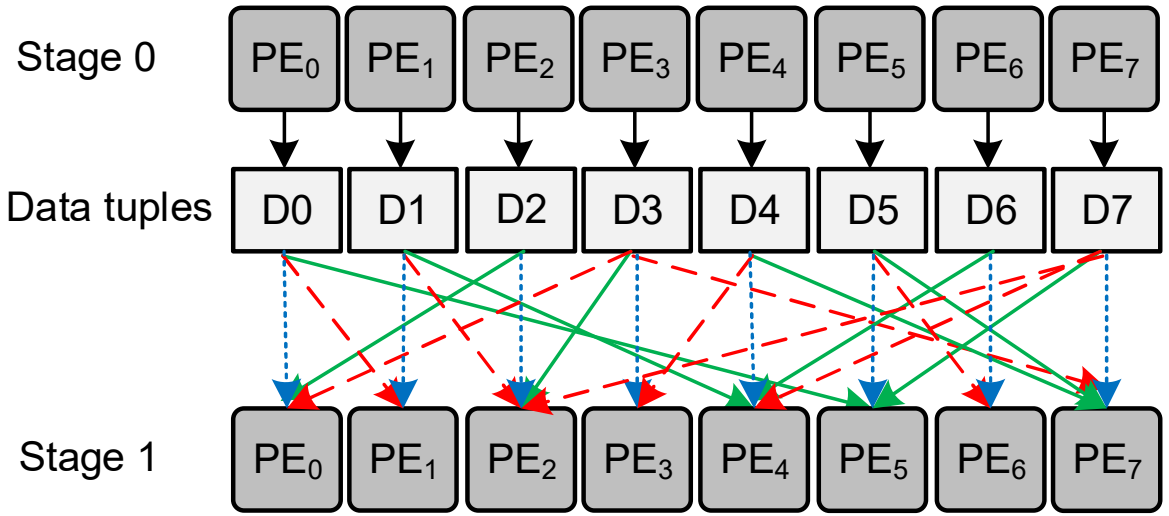
- **BRAM buffering**
  - Alleviate random memory accesses
- **Multiple PEs**
  - Each PE processes a part of buffered data to improve the throughput





Definition of Data Shuffling

- **Widely used for irregular applications**
- **The data with format <*dst, value*> goes to '*dst*' PE**
- **Challenges:**
  - Handling run-time data dependency
  - High parallelism



* Arrows with different colours show a few shuffling examples.

HLS not Natively Support Shuffling

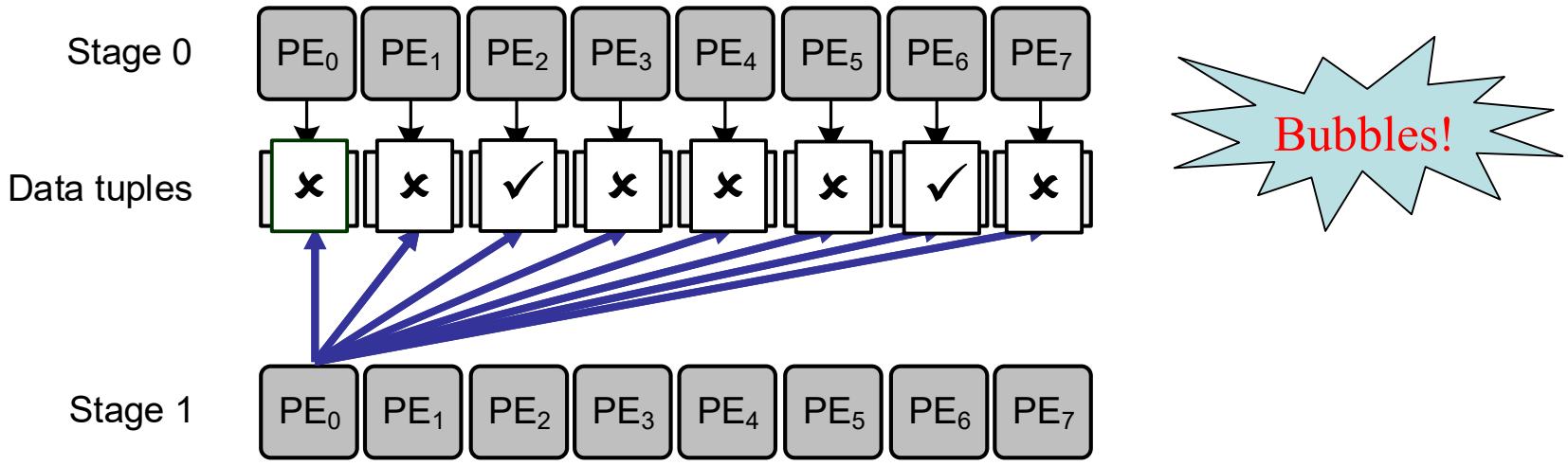
- **HLS lacks fine-grained control logic**
- **No vendor-specific extension for shuffling [1]**
- **Static analysis of HLS fails to extract parallelism in functions with run-time dependency [2]**

[1] Kapre, Nachiket, and Hiren Patel. "Applying Models of Computation to OpenCL Pipes for FPGA Computing." *Proceedings of the 5th International Workshop on OpenCL*. ACM, 2017.

[2] Z. Li, L. Liu, Y. Deng, S. Yin, Y. Wang, and S. Wei, "Aggressive pipelining of irregular applications on reconfigurable hardware," in *ISCA*, 2017.

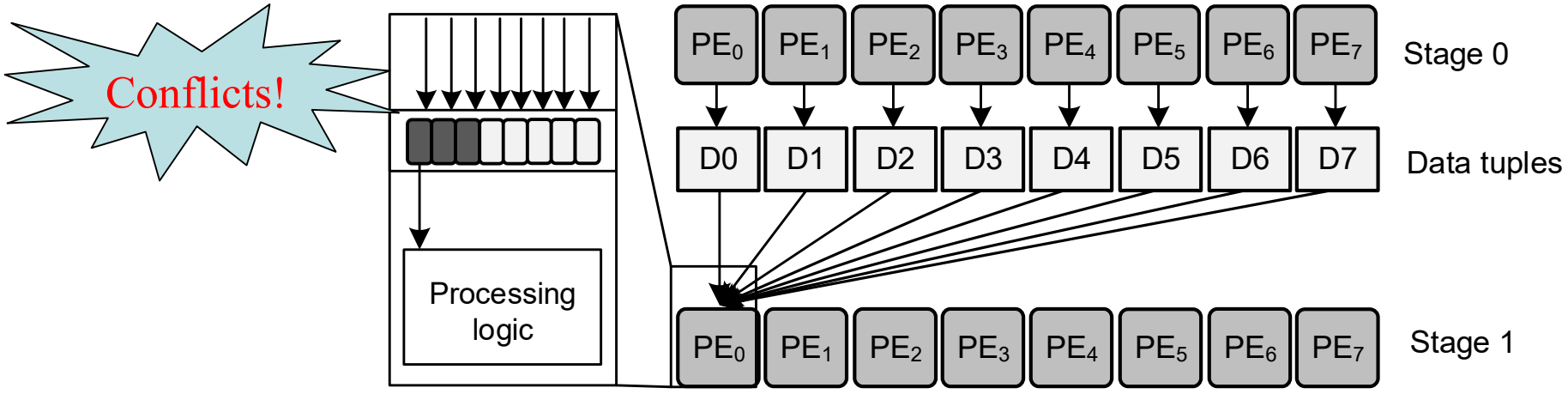
Potential Shuffling Solutions with HLS

- **Polling method**
  - Each PE checks the tuples serially
  - ‘Bubbles’ are introduced
  - 8 cycles for dispatching a set of 8 tuples



Potential Shuffling Solutions with HLS

- **Convergence kernel [1]**
  - Each PE writes wanted tuples to local BRAM in parallel
  - The run-time data dependency is left to compiler
  - Initiation interval (II) equals to 284 cycles



[1] Wang, Zeke, et al. "Multi-kernel Data Partitioning With Channel on OpenCL-Based FPGAs." *VLSI*, 2017 .

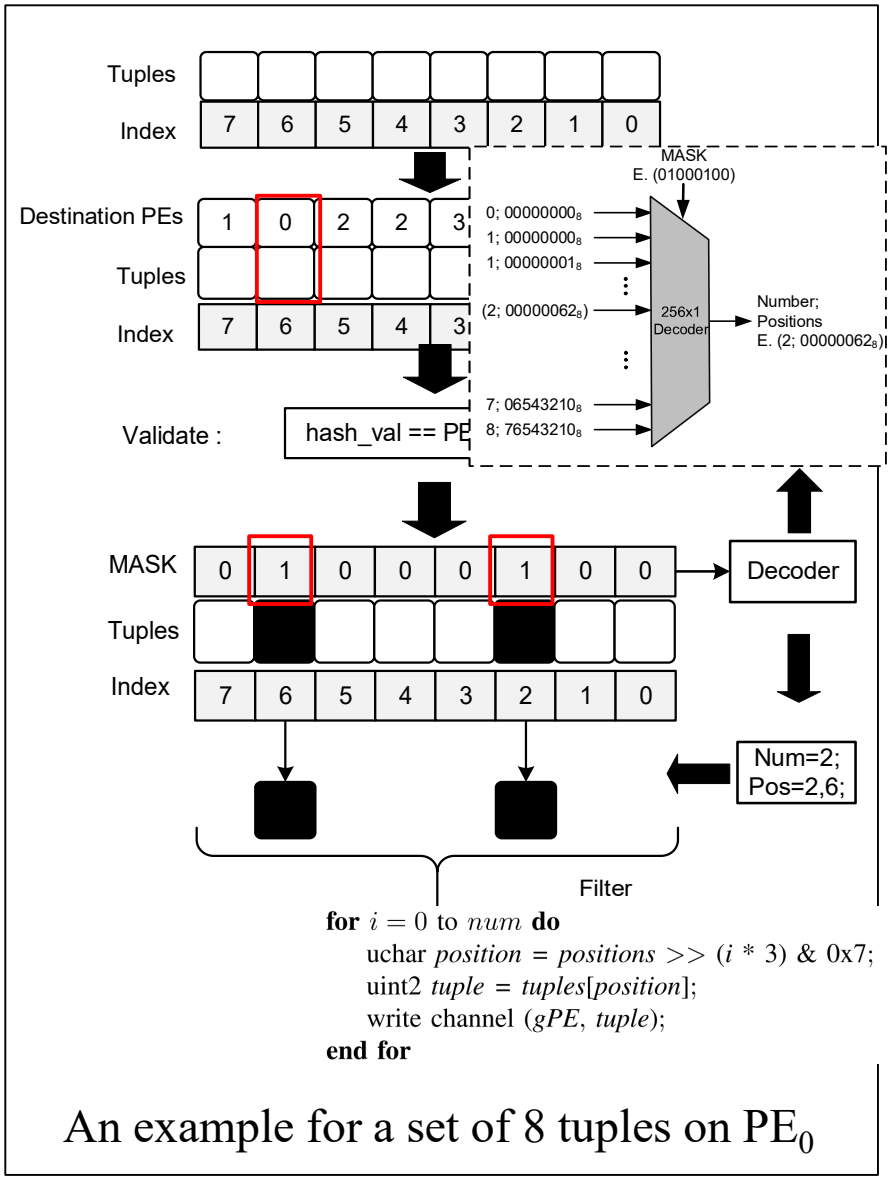
Insights

- **Polling: introduces ‘bubbles’**
- **Convergence kernel: run-time dependency is still there**

- **What if the PE knows the positions and number of wanted tuples?**
  - PEs can directly access the wanted tuples
  - Cycles needed equal to number of wanted tuples (no bubbles)
- **How to know the positions and number of wanted tuples?**
  - Decoder based solution
  - E.g. 2^8 possibilities for a set of 8 tuples, since each tuple has two statuses

Proposed Shuffling

- Calculate the destination PEs
- Compute an 8-bit MASK by comparing destination PEs with current PE id, 0
- Decode the positions and number of wanted tuples
- Collect the wanted tuples without “bubbles”



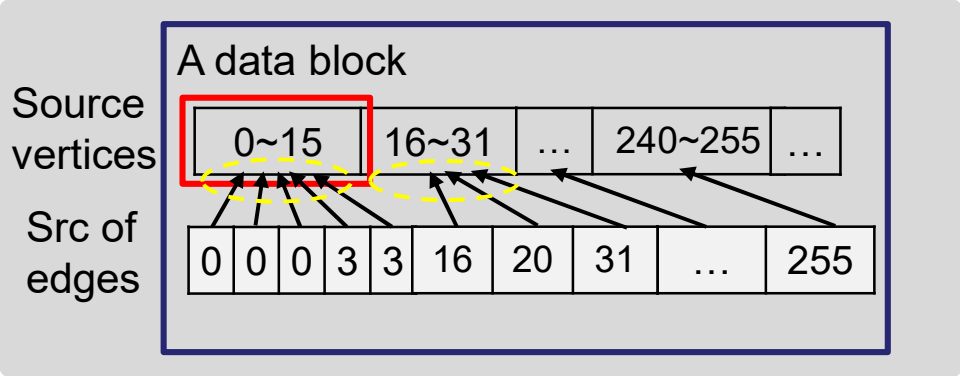
An example for a set of 8 tuples on PE_0

Outline

- On-the-fly Data Shuffling for HLS-based FPGAs
- **Improved Execution Flow and Memory Optimizations**
- Scaling to Multiple SLRs
- APIs
- Evaluation

Improved Execution Flow

- **ThunderGP pipelines scatter and gather inside FPGA**
  - Compared with serial execution [1], it:
    - **Reduces (random and sequential) memory accesses to global memory**
    - **Exploits pipeline parallelism better**
- **One side-effect: the access to source vertex introduces randomness**

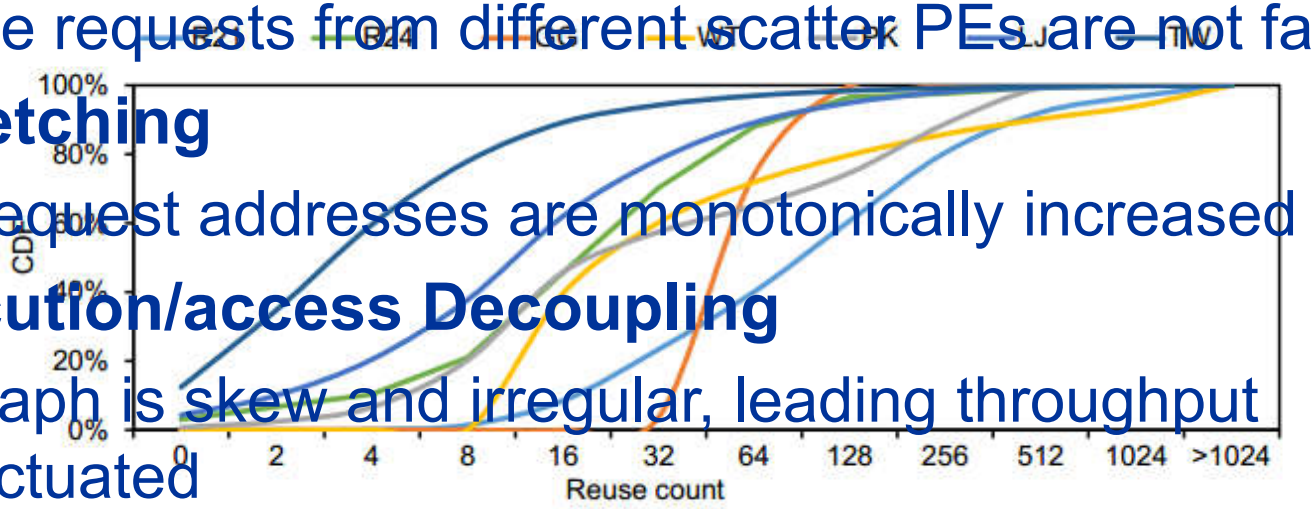


The access pattern to source vertices

[1] Shijie Zhou, Rajgopal Kannan, Viktor K Prasanna, Guna Seetharaman, and Qing Wu. 2019. HitGraph: High-throughput Graph Processing Framework on FPGA. *TPDS* (2019)

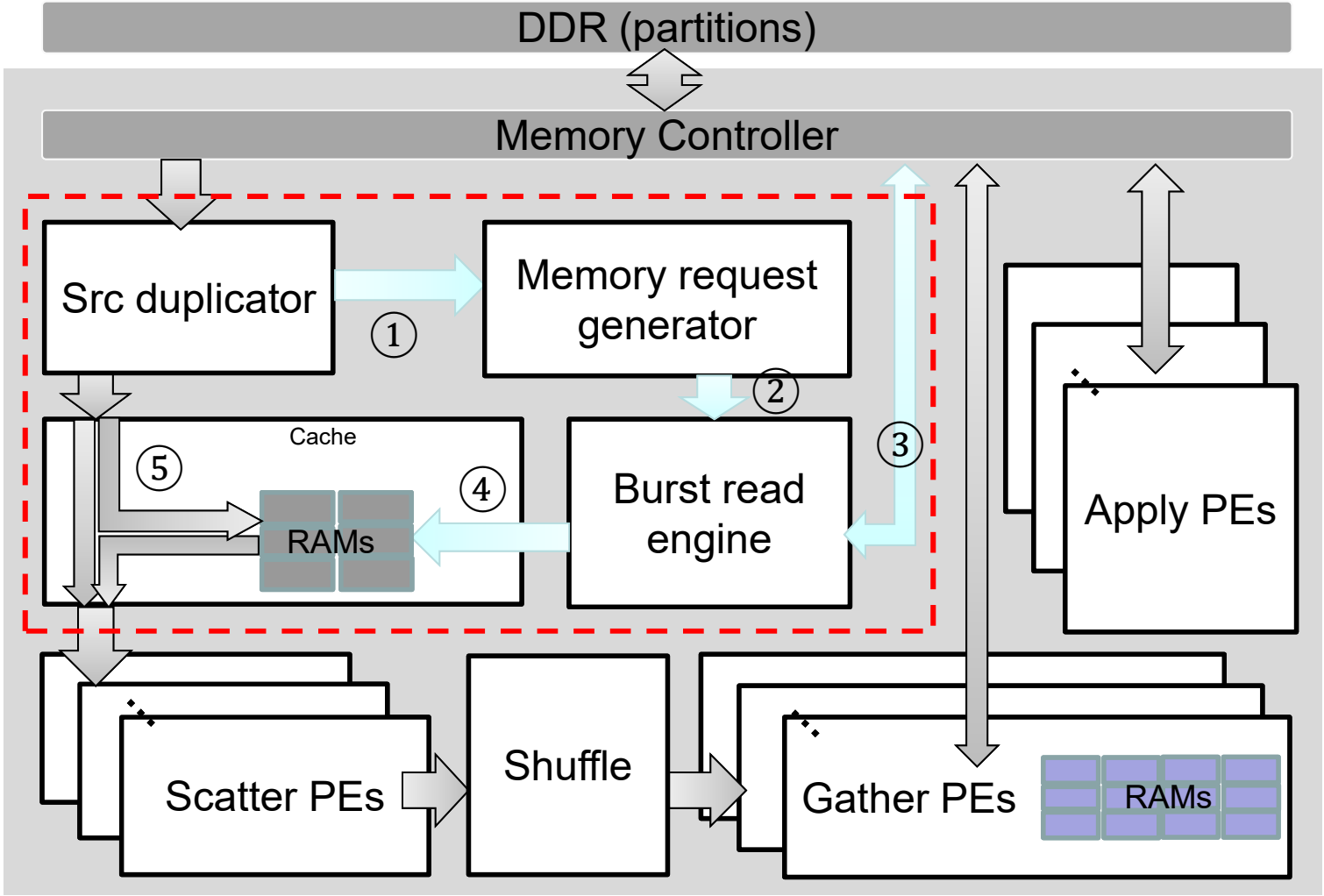
Memory Optimizations

- **Caching**
  - Data blocks build good locality
- **Coalescing**
  - The requests from different scatter PEs are not far away
- **Prefetching**
  - Request addresses are monotonically increased
- **Execution/access Decoupling**
  - Graph is skew and irregular, leading throughput fluctuated



The CFD of reuse count of data blocks for graphs.

Overall Architecture on Single SLR

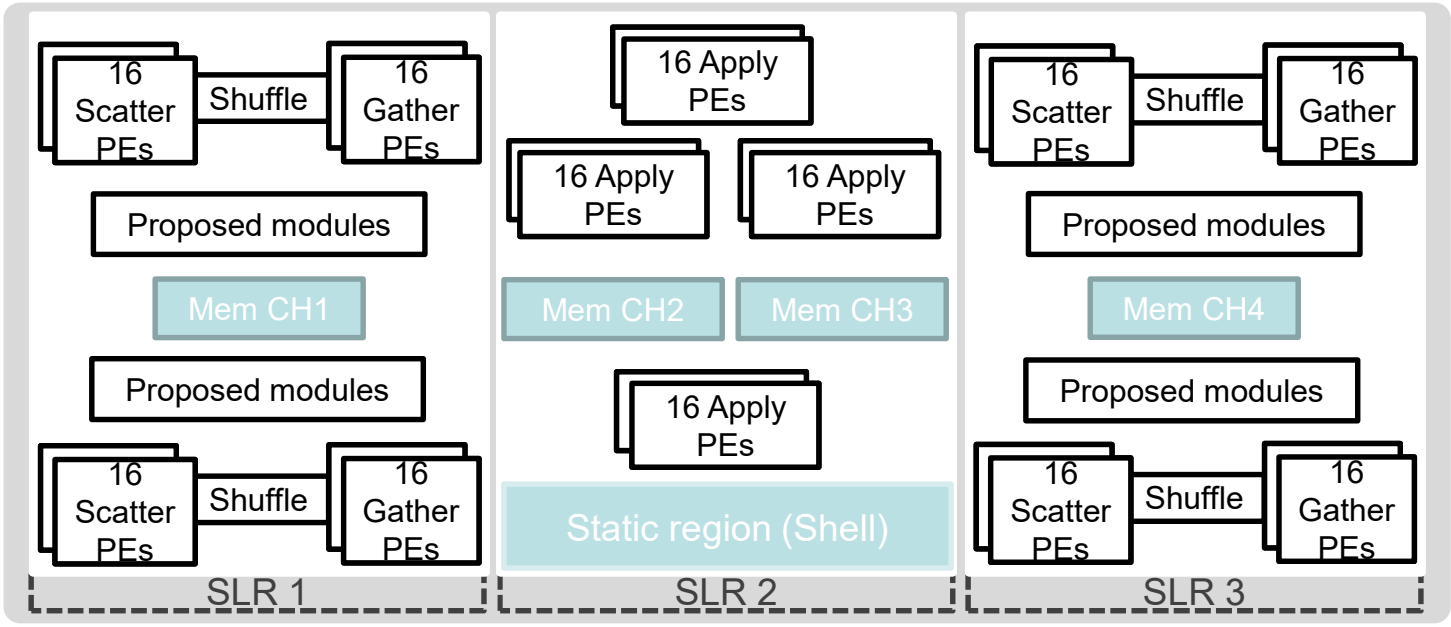


Outline

- On-the-fly Data Shuffling for Graph Processing Framework
- Improved Execution Flow and Memory Optimizations
- **Scaling the Framework to Multiple SLRs**
- APIs
- Evaluation

Scaling to Multi-SLR

- **Fitting kernels to multiple SLRs**
- **Scheduling graph partitions to SLRs**



The implementation on FPGAs with three SLRs and four memory channels

Outline

- On-the-fly Data Shuffling for Graph Processing Framework
- Improved Execution Flow and Memory Optimizations
- Scaling the Framework to Multiple SLRs
- **APIs**
- Evaluation

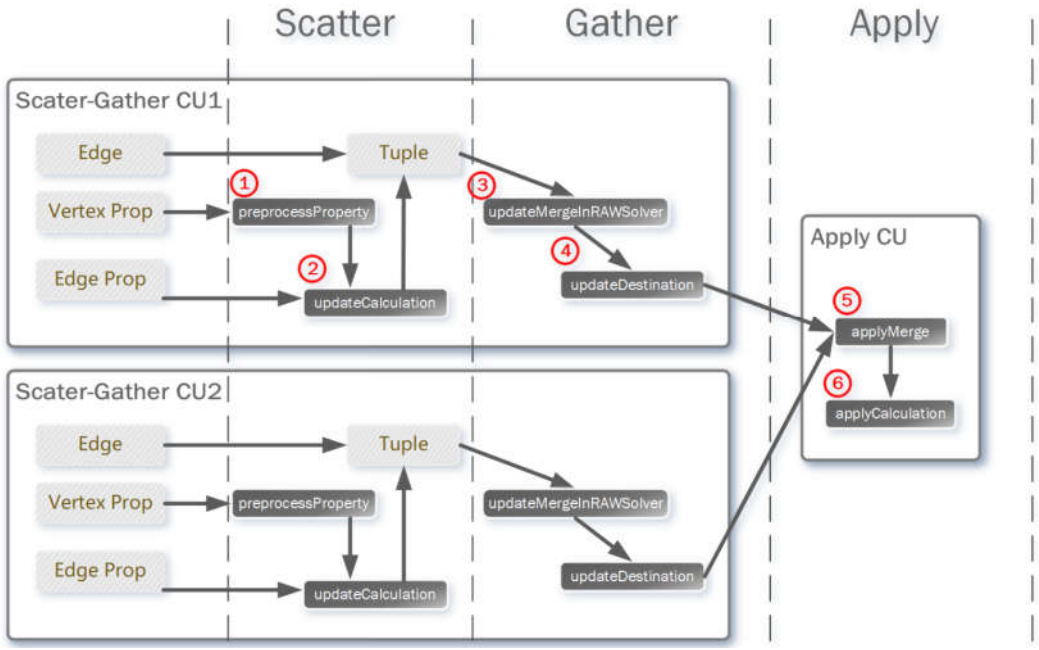
APIs (programmability)

- **ThunderGP provides three levels APIs**
  - L1: constructs the basic modules to build the dataflow
  - L2: provides hooks for mapping new algorithms
  - L3: wraps partition scheduling and memory management interface for multiple SLRs

*More details can be found on https://github.com/Xtra-Computing/ThunderGP/

APIs – L2

The hooks need to be tailored to user's algorithms



Hooks	ID	Description
preprocessProperty	1	Per-process the source vertex property.
updateCalculation	2	Calculate the update value by using the edge property and source vertex property.
updateMergeInRAWSolver	3	Destination property update in RAW solver.
updateDestination	4	Destination property update.
applyMerge	5	Destination property merge from all of the scatter-gather CUs.
applyCalculation	6	Calculate the new property of vertices.

*Details can be found on https://github.com/Xtra-Computing/ThunderGP/blob/master/docs/algorithm_mapping.md

APIs – L2

SpMV as an example:

```
/* directly pass the value, do not need in SpMV */
inline prop_t preprocessProperty(prop_t srcProp)
{
    return (srcProp);
}

/*  calculate the  A.val[i] * x[A.col[i]]; */
inline prop_t updateCalculation(prop_t srcProp, prop_t edgeProp)
{
    return ((srcProp) * (edgeProp));
}

/* accumulate y[r] in rawSolver */
inline prop_t updateMergeInRAWSolver(prop_t ori, prop_t update)
{
    return ((ori) + (update));
}

/* accumulate y[r] in on-chip memory */
inline prop_t updateDestination(prop_t ori, prop_t update)
{
    return ((ori) + (update));
}
```

Scatter-gather

```
/* accumulate y[r] among CUs */
inline prop_t applyMerge(prop_t ori, prop_t update)
{
    return ((ori) + (update));
}
/* no other calculation in apply phase */
inline prop_t applyCalculation( prop_t tProp,
                                prop_t source,
                                prop_t outDeg,
                                unsigned int &extra,
                                unsigned int arg
                              )
{
    return tProp;
}
```

Apply

*Details can be found on https://github.com/Xtra-Computing/ThunderGP/blob/master/docs/algorithm_mapping.md

Outline

- On-the-fly Data Shuffling for Graph Processing Framework
- Improved Execution Flow and Memory Optimizations
- Scaling the Framework to Multiple SLRs
- APIs
- **Evaluation**

Experimental Configuration

- **Hardware platforms: VCU1525 board and Terasic DE5-Net board**
- **Applications: BFS, SSSP, PageRank and SpMV**
- **Dataset shown as below**

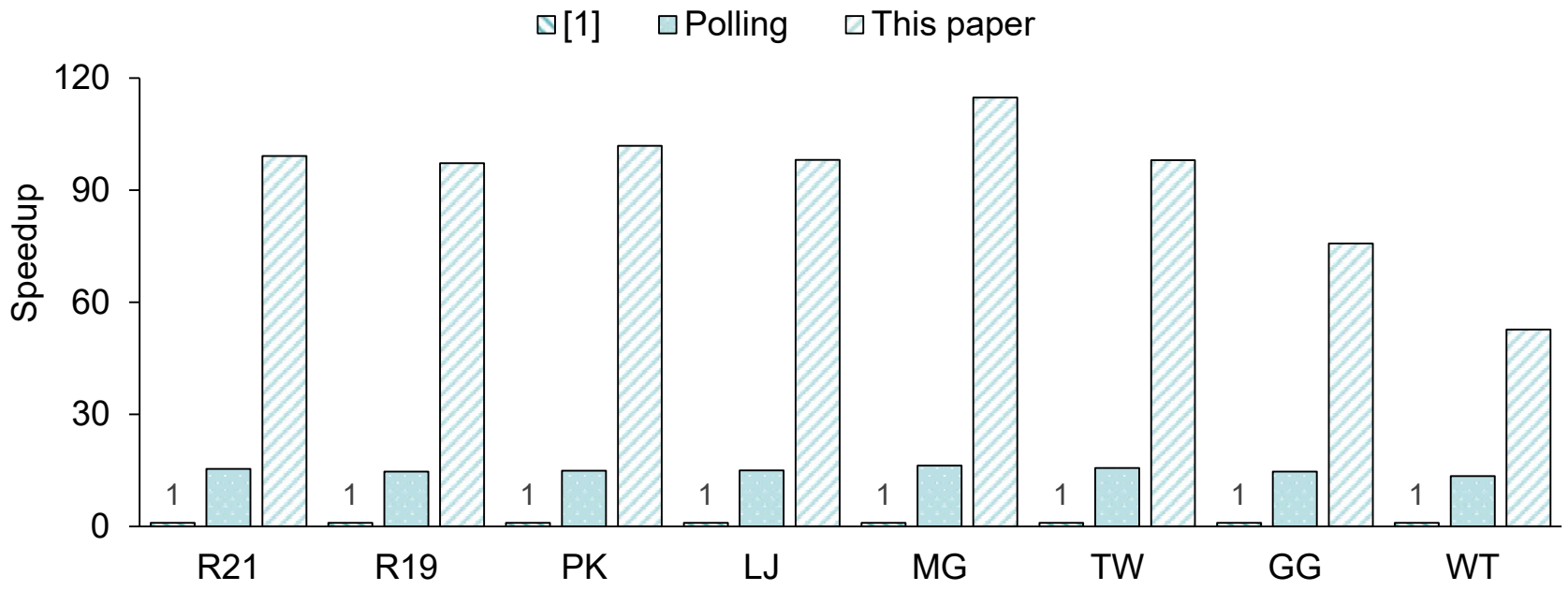
TABLE II: Graph dataset.

Graphs	$\|V\|$	$\|E\|$	D_{avg}	D_{max}
rmat-19-32 (R19) [34]	524K	17M	32	90K
rmat-21-32 (R21) [34]	2M	67M	32	211K
mouse-gene (MG) [35]	43K	14.5M	670	8K
web-google (GG) [35]	875K	5.1M	11	6.4K
pokec (PK) [35]	1.6M	31M	37	20K
wiki-talk (WT) [35]	2M	5M	4	100K
live-journal (LJ) [35]	4.8M	69M	13	3K
twitter-2010 (TW) [35]	41M	1.4B	35	770K

[34] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," JMLR, 2010.
[35] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in AAAI, 2015.

Evaluation on Proposed Shuffling

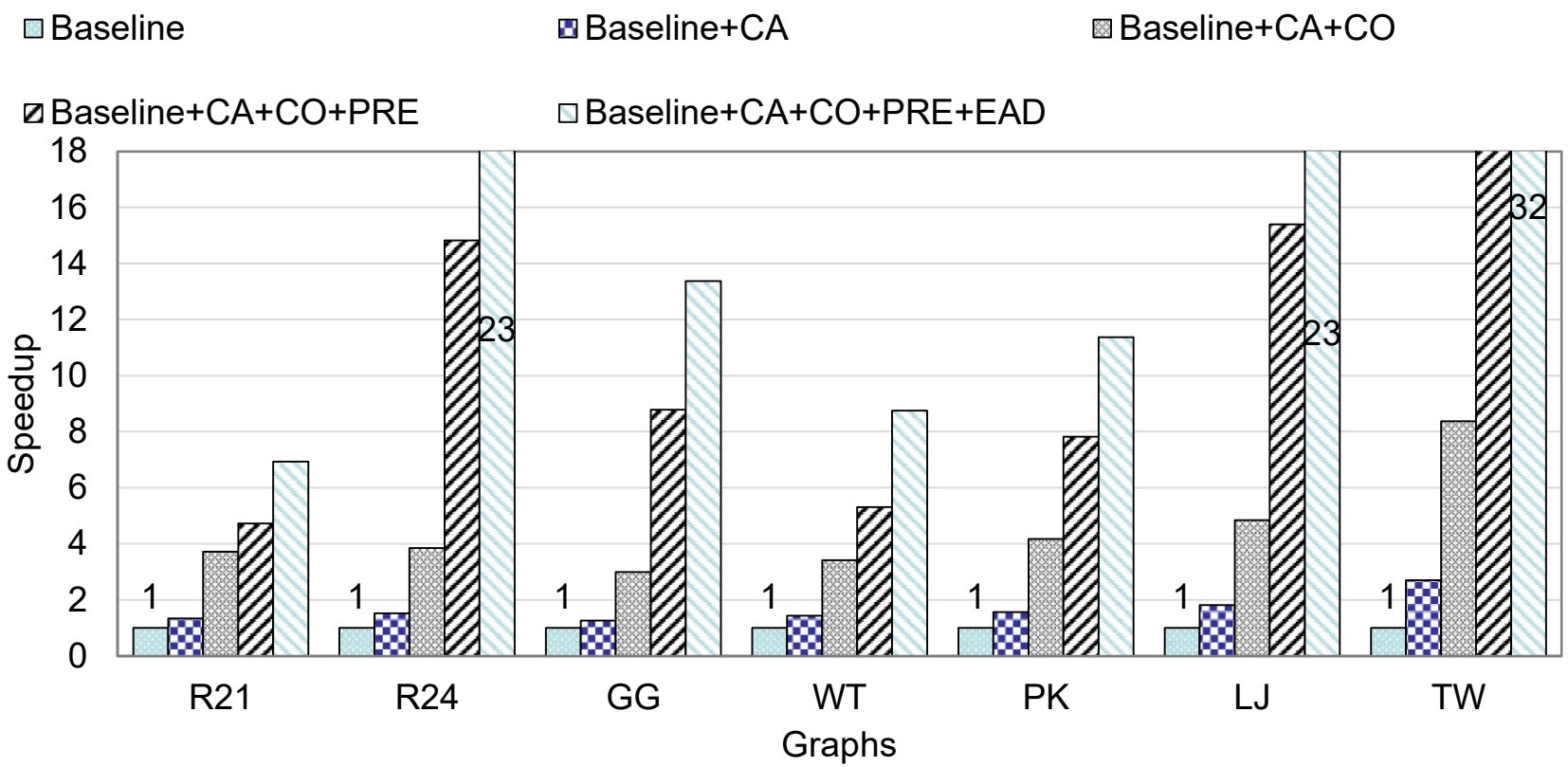
- **Compare the performance of graph frameworks with different shuffling solutions**
- **Speedup of PageRank is up to 100× of [1], and 6× of Polling**



[1] Wang, Zeke, et al. "Multi-kernel Data Partitioning With Channel on OpenCL-Based FPGAs." *VLSI*, 2017 .

Evaluation on Memory Optimizations

- **Four methods cumulatively contribute to performance**
- **Speedup can be up to 32×**



The performance speedup from four optimizations:
caching (CA), coalescing (CO), prefetching (PRE) and access/execute decoupling (DAE)

ThunderGP vs. the State-of-the-art [1]

Algorithm	Dataset	Throughput of work [1]	Our throughput	Speedup
SpMV	WT	1,004	3,138	**3.1x**
SpMV	LJ	1,906	2,860	**1.5x**
PR	R21	3,410	4,759	**1.4x**
PR	LJ	2,110	3,133	**1.5x**
SSSP	WT	2,156	2,954	**1.4x**

Performance (MTEPS) comparison on Xilinx VCU1525 board

*Notes:

1. Since work [1] is not open sourced, performance is collected from their paper.
2. Our performance is measured on-board implementations while their performance is presented with simulation.

[1] Shijie Zhou, Rajgopal Kannan, Viktor K Prasanna, Guna Seetharaman, and Qing Wu. 2019. HitGraph: High-throughput Graph Processing Framework on FPGA. *TPDS* (2019)

ThunderGP is Publicly Available

- **GitHub:** https://github.com/Xtra-Computing/ThunderGP
- **Also featured at** Xilinx Apps and Libraries

ThunderGP

license apache2 issues 0 open

ThunderGP: Fast Graph Processing for HLS-based FPGAs

What's new?

ThunderGP is featured at Xilinx Apps and Libraries

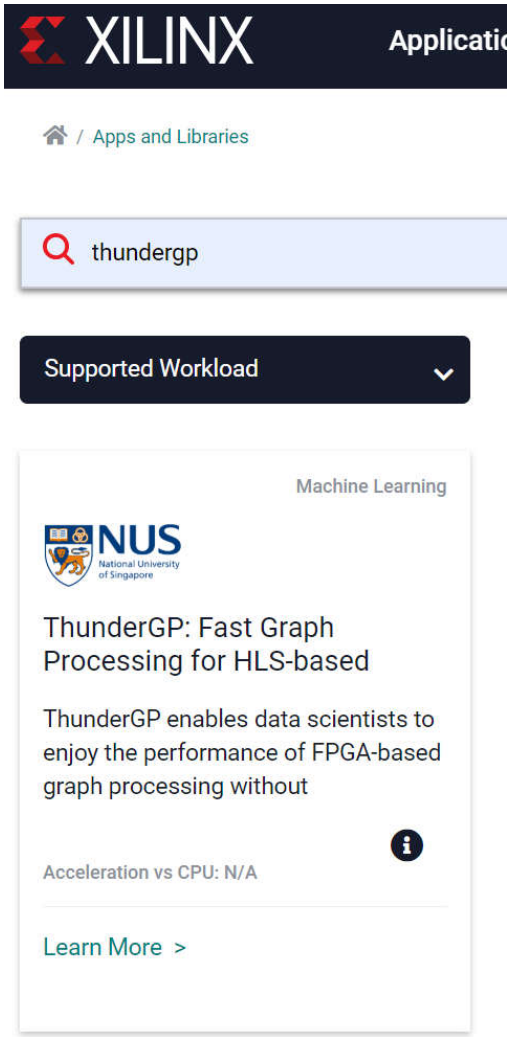
Introduction

ThunderGP enables data scientists to enjoy the ***performance*** of FPGA-based graph processing without compromising ***programmability. To our best knowledge and experiments, this is the fastest graph processing framework on HLS-based FPGAs.***

Two aspacts make the ThunderGP deliver superior performance. On the one hand, ThunderGP embraces an improved execution flow to better exploit the pipeline parallelism of FPGA and alleviate the data access amount to the global memory. On the other hand, the memory accesses are highly optimized to fully utilize the memory bandwidth capacity of the hardware platforms.

ThunderGP can run on both Xilinx and Intel platforms:

- Check the implementation on Intel platform out.
- On Xilinx multi-SLR based FPGAs, it is running at 250Mhz, and the performance can be up to ***5300 MTEPS (million traversed edges per second)***, or a ***2 times speedup*** over the state-of-the-art.



Conclusion

- There is a pressing need for high-performance graph processing frameworks
- We propose ThunderGP, which enables data scientists to enjoy the *performance* of FPGA-based graph processing without compromising *programmability*
- To our best knowledge and experiments, this is the fastest graph processing framework on HLS-based FPGAs

Future Work

- **How to beat CPU/GPU?**
  - More powerful hardware? (HBM-based boards)
  - It is promising since we achieve full potential of memory bandwidth
- **Go beyond simple graph model (GAS)**
  - GAS model is too simple to be efficient for many algorithms
- **Go beyond simple graphs**
  - Graphs in real applications are far more complex than static graphs in experiments

Acknowledgement

- Singapore MoE Tier 2 grant (MOE2017-T2-1-122).

- Xilinx Adaptive Compute Clusters (XACC) program

Thanks