ThunderGP: HLS-based Graph Processing Framework on FPGAs

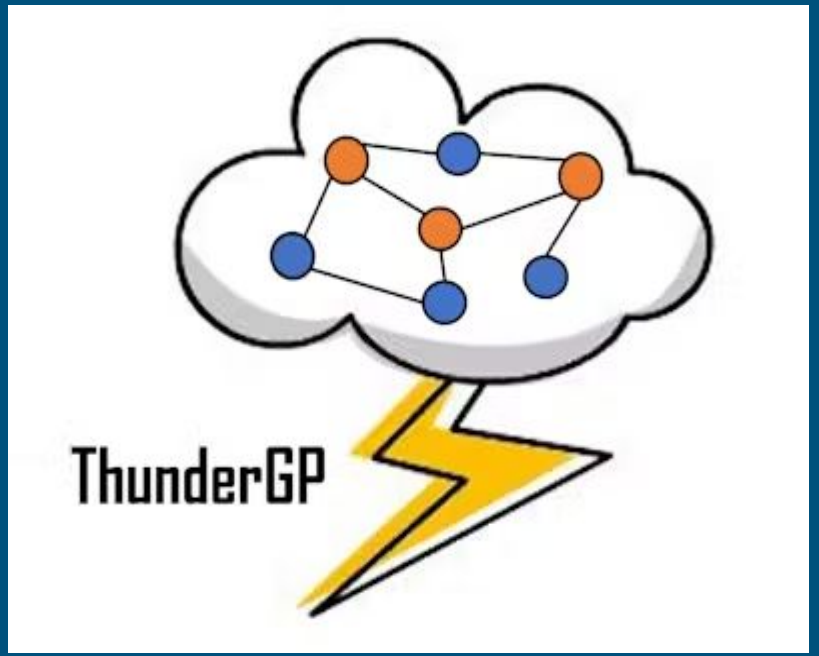
William Yin

Background

- Graph processing performance in large demand with rapid growth of data
- Many attempts have been made to tackle the challenges of designing efficient FPGA-based accelerators for graph processing.
- Still requires hardware expertise and development efforts from developers

ThunderGP

- Open-source HLS-based graph processing framework on FPGAs
- Allows for FPGA-accelerated graph processing with high level functions - no knowledge of hardware needed
- Utilizes Gather-Apply-Scatter (GAS) model



HDL & HLS Difficulties

HLS for FPGAs

Verilog	VHDL
ASIC Designs	FPGA Designs
Weakly Typed	Strongly Typed
Low Verbosity	High Verbosity
Partially Deterministic	Very Deterministic
More “C” like	Non “C” like

Existing Studies Similar to ThunderGP

Works	API[1]	PL[2]	Auto[3]	Eva[4]	App[5]	Public[6]
(F) GraphGen [29]	✘	HDL	✔	HW	2	✘
(F) FPGP [20]	✘	HDL	✘	HW	1	✘
(F) HitGraph [24]	✘	HDL	✔	SIM	4	✔
(F) Foregraph [23]	✘	HDL	✘	SIM	3	✘
(F) Zhou et al. [21]	✘	HDL	✔	SIM	2	✘
(F) Chen et al. [32]	✘	HLS (OpenCL)	✘	HW	4	✔
(L) GraphOps [27]	✘	HLS (MaxJ)	✘	HW	6	✔
(A) FabGraph [26]	✘	HDL	✘	SIM	2	✘
(A) Zhou et al. [22]	✘	HDL	✘	SIM	3	✘
(A) AccuGraph [25]	✘	HDL	✘	SIM	3	✘
(F) ThunderGP	✔	**HLS (C++)**	✔	**HW**	**7**	✔

[1] Whether the system provides explicit application programming interface;
[2] Required programming language for development;
[3] Whether the system supports automated design flow;
[4] Evaluation based on simulation (SIM) or real hardware implementation (HW);
[5] Number of evaluated applications with the system;
[6] Whether the system is publicly available.

Downsides of Similar Works

- Poor programmability
  - No existing APIs
  - Usage of HDLs such as Verilog/VHDL
- No automated design flow
  - Requires manual performance tuning
- Poor usability
  - Majority of works not available to public

Existing Challenges to be Addressed

Challenge 1: Be able to support various graph processing algorithms (ex. bfs (breadth first search), sssp (single source shortest path), etc.)

Solution: Design architectural template with the GAS model

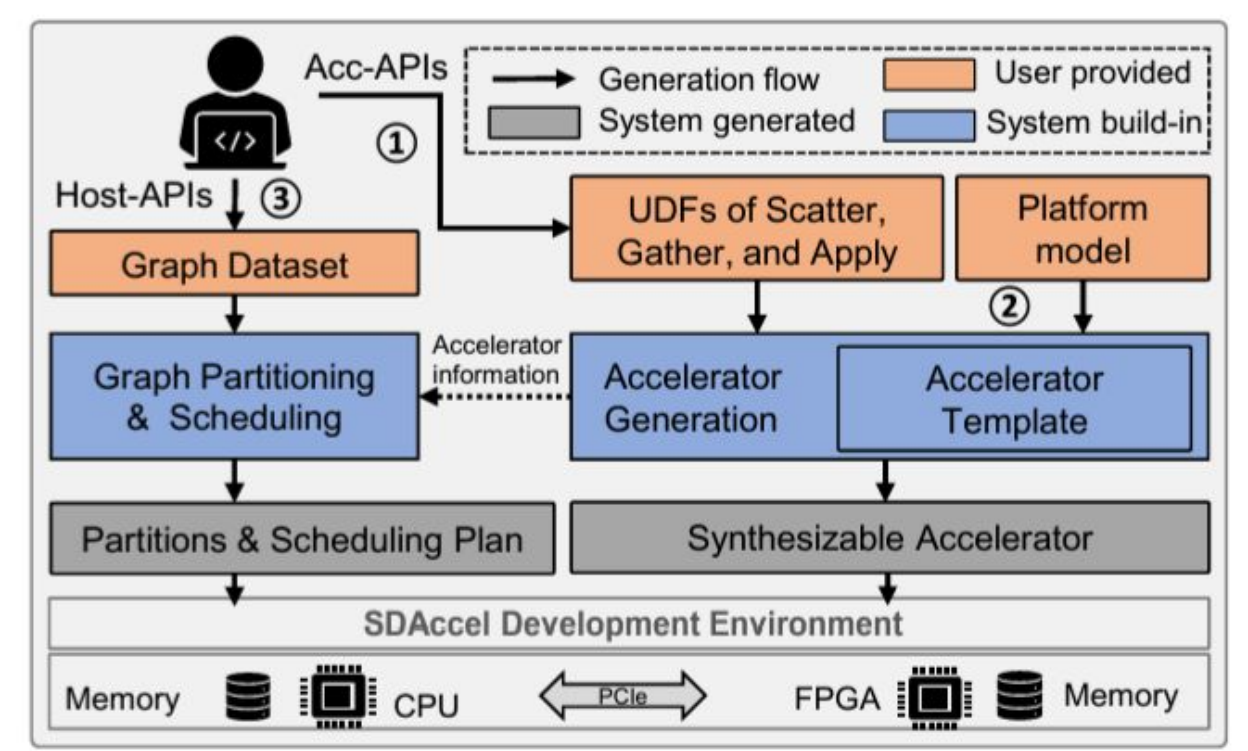
Challenge 2: Programming without hardware expertise

Solution: Only expose high-level APIs to developers and take everything else to be turned into synthesizable code

Challenge 3: Utilize high-performance FPGAs efficiently

Solution 3: Scale to Multi-SLR FPGAs and schedule graph data appropriately

ThunderGP Overview



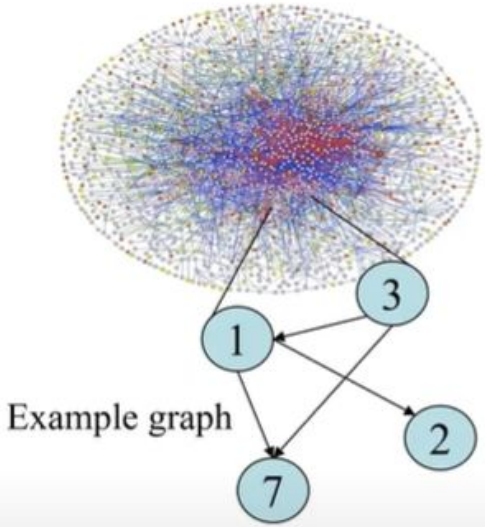
Gather-Apply-Scatter Model

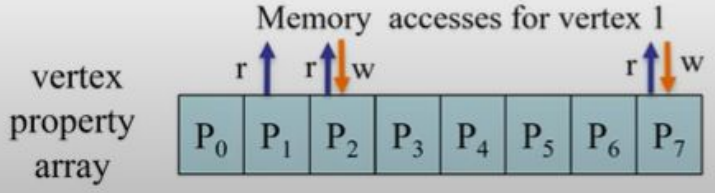
- Provides high level abstraction for various graph processing algorithms

Algorithm 1 The GAS Model

```
1:  while not done do
2:      for all e in Edges do                    ▷ The Scatter stage
3:          u = new update
4:          u.dst = e.dst
5:          u.value = Scatter(e.w, e.src.value)
6:      end for
7:      for all u in Updates do                  ▷ The Gather stage
8:          u.dst.accum = Gather(u.dst.accum, u.value)
9:      end for
10:     for all v in Vertices do                 ▷ The Apply stage
11:         Apply(v.accum, v.value)
12:     end for
13: end while
```

Gather-Apply Supply-Model Further Defined





GAS model's three stages [1]:

- **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.

GAS Model API Functions

Table 2: Acc-APIs (user defined functions).

APIs	Parameters	Return	Description
prop_t scatterFunc ( )	vertex property, edge property.	update value	Calculates update value for destination vertices
prop_t gatherFunc ( )	update tuple, buffered destination vertices.	accumulated value	Gathers update values to buffered destination vertices
prop_t applyFunc ( )	vertex property*, outdegree*, etc*.	latest vertex property	Updates vertex properties for next iteration

Execution Flow

- Vertex buffering with RAMs
- Multiple PEs (processing elements) with shuffle

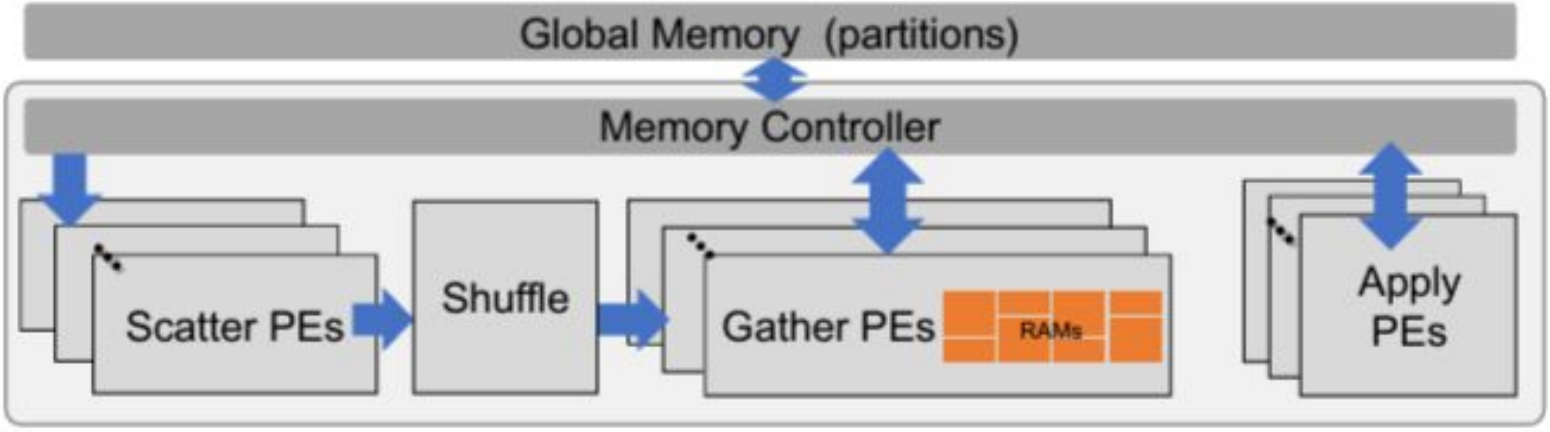
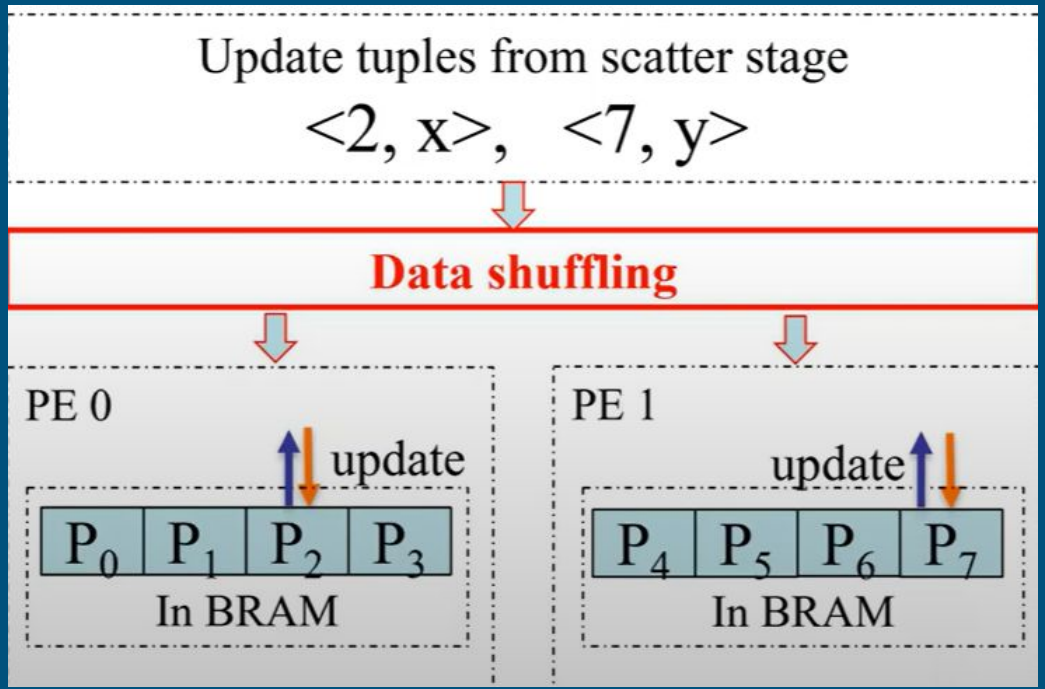
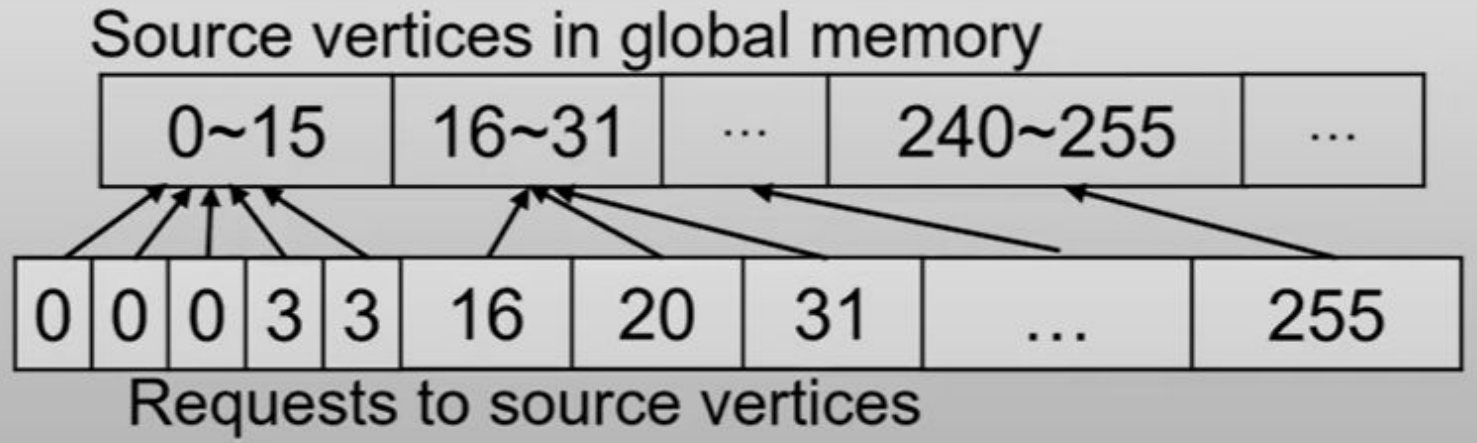


Figure 2: The overview of the accelerator template.

Benefits of Multiple PEs



Memory Access Optimizations



The accesses to source vertices from scatter PEs

Automated Generation

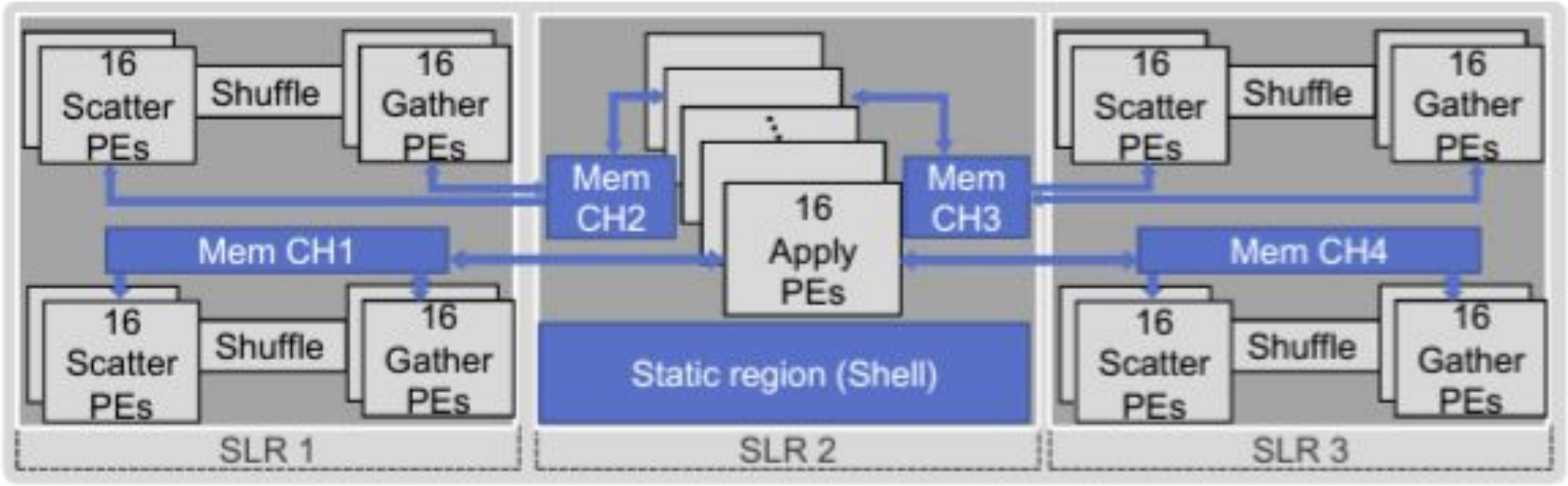


Figure 5: The example implementation on VCU1525 with three SLRs and four memory channels.

Graph Partitioning

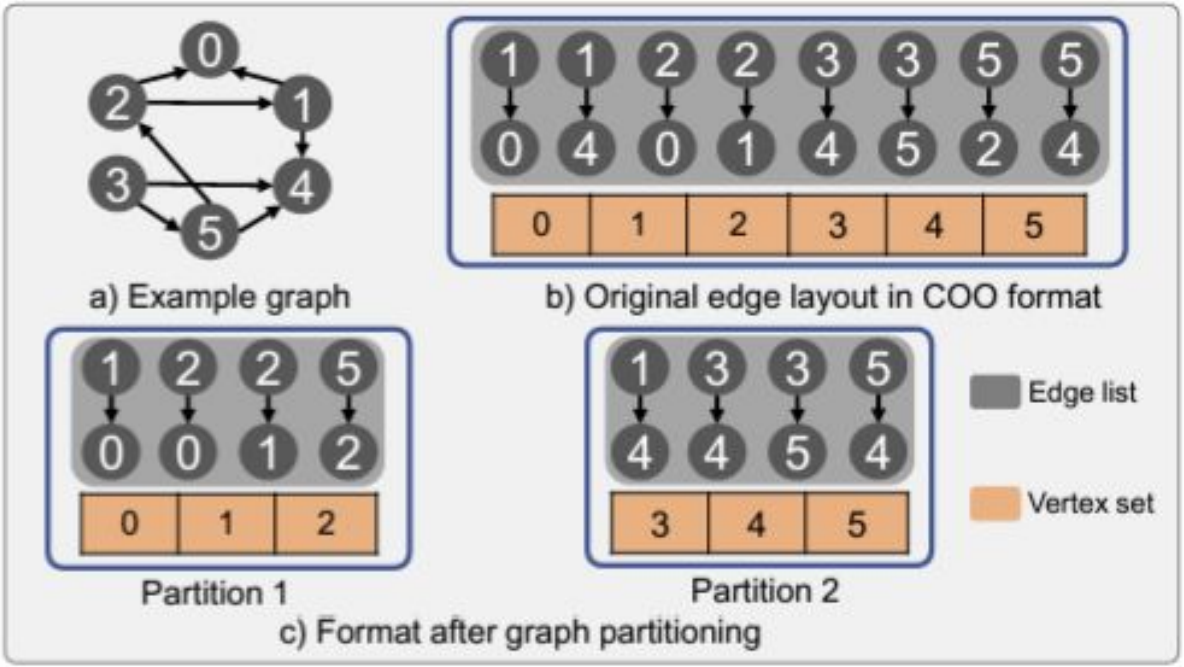
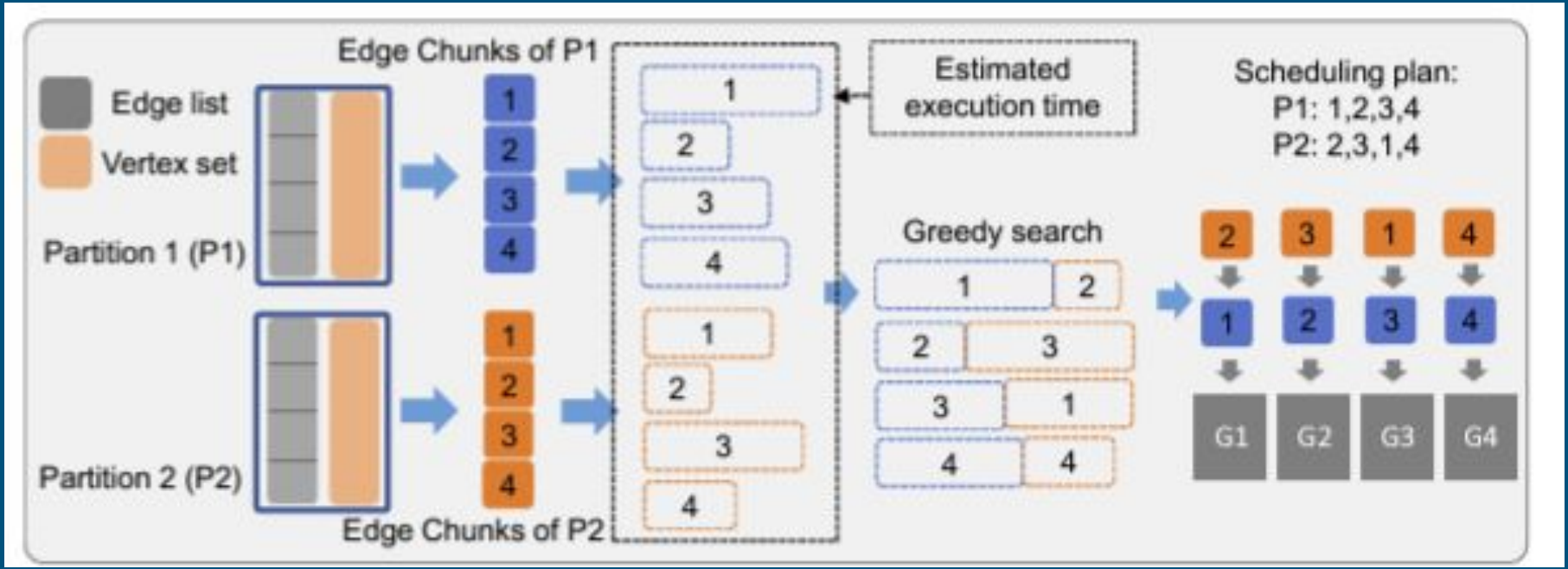


Figure 6: The example of graph partitioning.

Scheduling Partitions



Evaluation

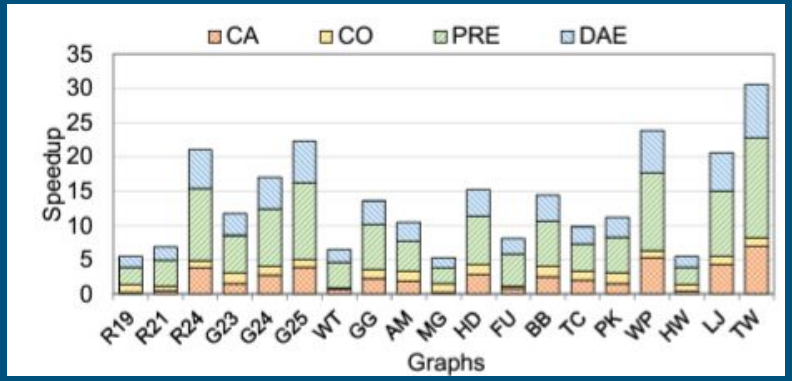
Table 5: The graph applications.

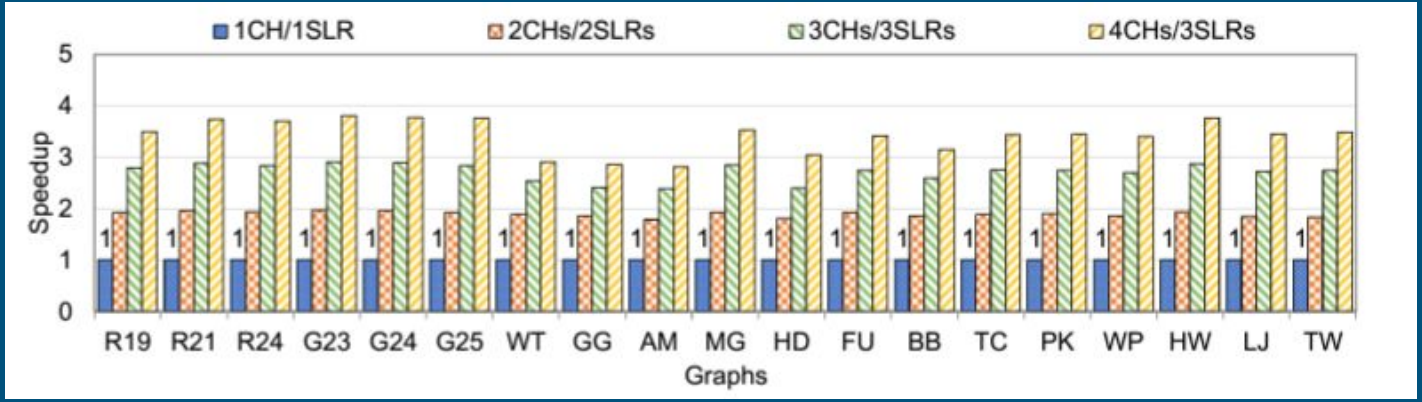
App.	Description
PR	Scores the importance and authority of a website through its links
SpMV	Multiplies a sparse matrix (represented as a graph) with a vector
BFS	Traverses a graph in a breadth ward from the selected node
SSSP	Finds the shortest path from a selected node to another node
CC	Detects nodes which could spread information very efficiently
AR	Measures the transitive influence or connectivity of nodes
WCC	Finds maximal subset of vertices of the graph with connection

Table 7: Frequency (MHz) improvement on a single SLR.

Freq.	PR	SpMV	BFS	SSSP	CC	AR	WCC
Baseline	168	253	257	184	198	173	247
SS	242	286	281	231	267	273	243
SS+MDD	**297**	**296**	**299**	**300**	**287**	**301**	**296**
Improvement	**77%**	**17%**	**16%**	**63%**	**45%**	**74%**	**20%**

Overall Performance





Conclusion

Benefits

- FPGA graph-processing ability without hardware expertise needed

Strengths

- Delved into the algorithms used and explained them fairly well
- Great overview figure of ThunderGP

Weaknesses

- Figures were extremely out of place
- Little to no elaboration on Host-APIs provided by system

References

- Chen, Xinyu, et al. “ThunderGP: The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.” ACM Conferences, 17 Feb. 2021, dl.acm.org/doi/10.1145/3431920.3439290.
- Pregel, GraphLab, and Xstream, id2221kth.github.io/slides/2018/11_graph_processing_part1.pdf.
- “Nimbix: Empowering Enterprises with HPC-Enabled Secure Cloud Solutions.” Eviden, 30 Oct. 2023, www.nimbix.net/.