HW-aware mapping of Graph Neural Networks on RISC-V GPGPU A Work-in-Progress

Giuseppe M. Sarda[1,2], Nimish Shah[1], Debjyoti Bhattacharjee[2],
Peter Debacker[2], Marian Verhelst[1,2]
[1]KU Leuven, Leuven, Belgium
[2]imec, Leuven, Belgium

Introduction

Giuseppe

- **Graduated** from **Politecnico di Torino** Electronics engineering
- Joined **imec – KU Leuven** Sept. 2020 **Ph.D. researcher**

- Passionate about **on-chip ML**, **computer architectures**, **computer arithmetic**

email: Giuseppe.Sarda@imec.be

Outline

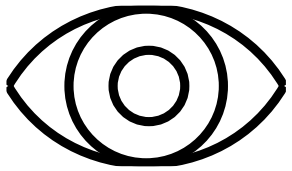
- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

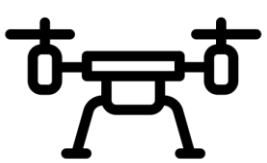
Outline

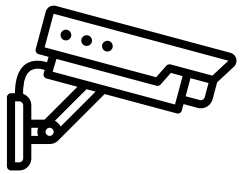
- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

Machine learning ...

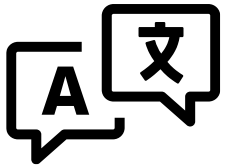
Computer vision



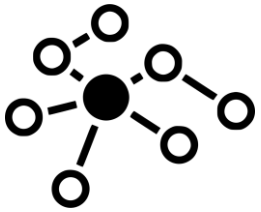


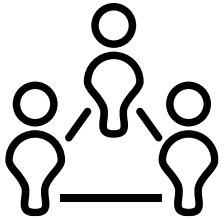


Natural language process



Graph-based problems



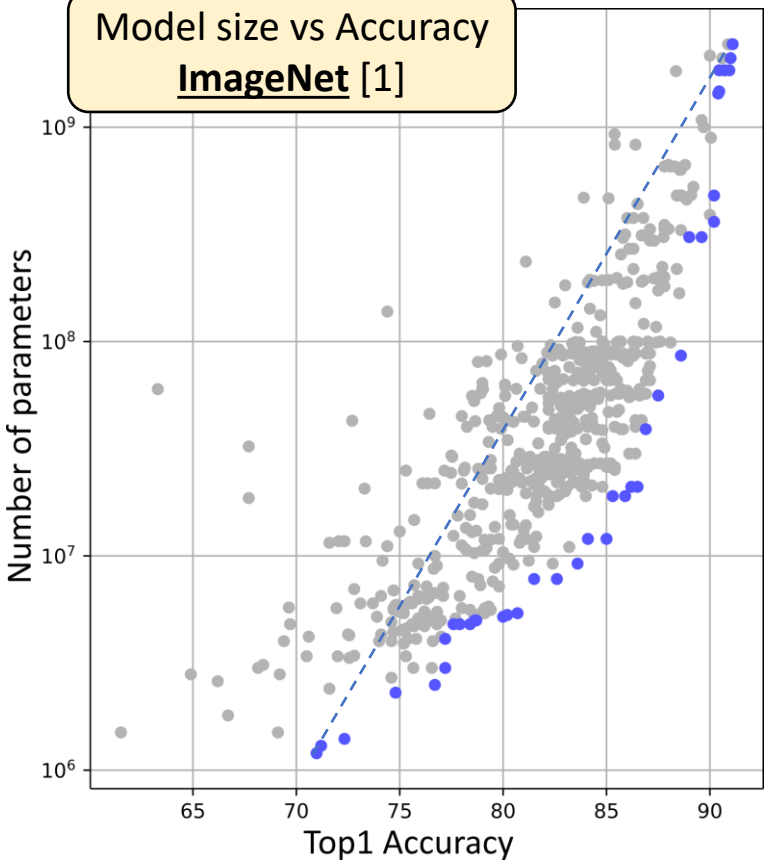


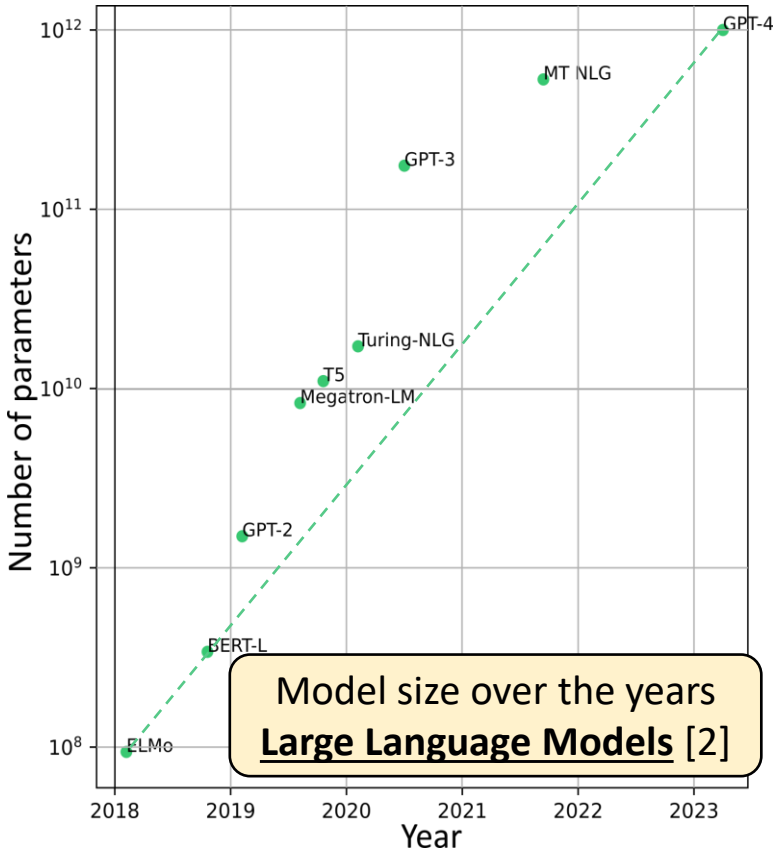
Machine Learning can achieve outstanding results on many tasks ...

Machine learning ...

[1] “Image Classification on ImageNet”, papers with code, June 2023
[2] “Large Language Models: A New Moore's Law?”, Hugging Face blog, 2022
[3] “Graph Neural Networks: Methods, Applications, and Opportunities”, Lilapati et al., AI Open, 2020

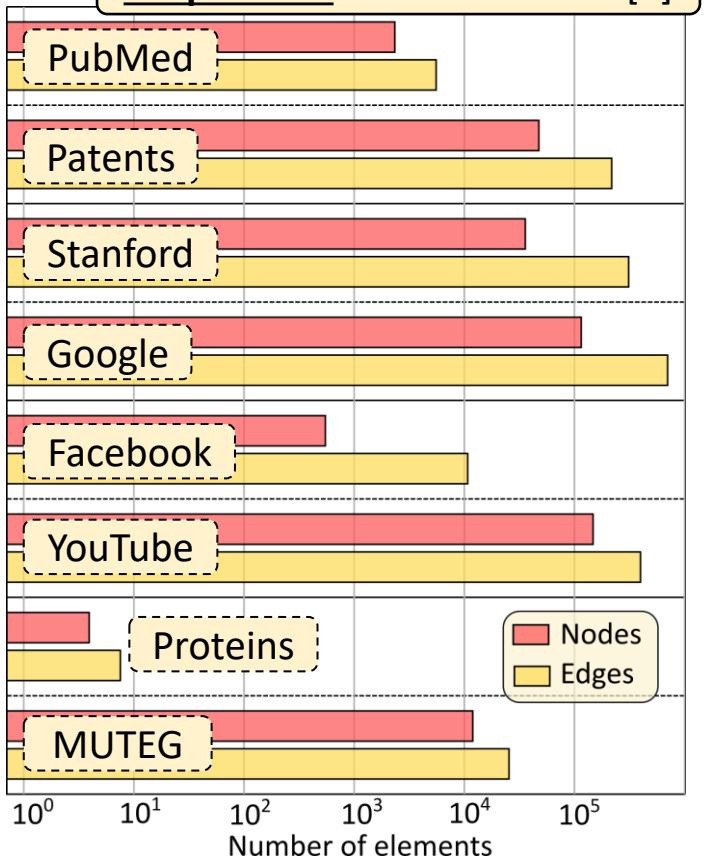
Model size vs Accuracy
ImageNet [1]





Model size over the years
Large Language Models [2]

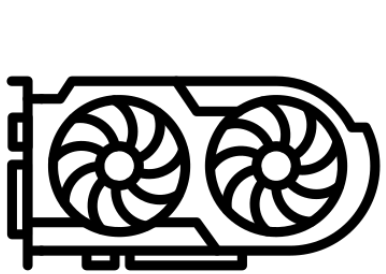
Graphs size of GNN datasets [3]

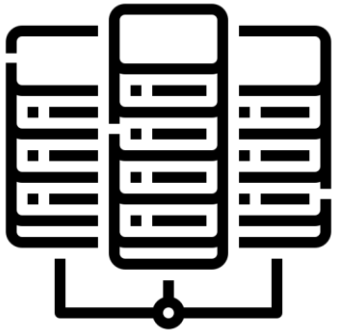


Machine Learning can achieve outstanding results on many tasks ...
with **increasing requirements** 🚀

… and computers!

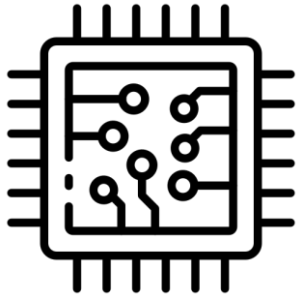
GPUs (and compute clusters)

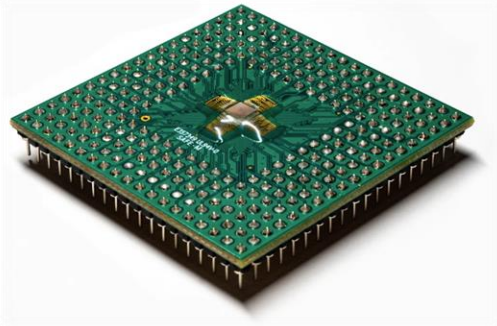




- ✓ High, scalable parallelism
- ✓ **General purpose**
- ✓ Easy to program
- ✗ Low efficiency
- ✗ **Difficult to further specialize**

ASIC

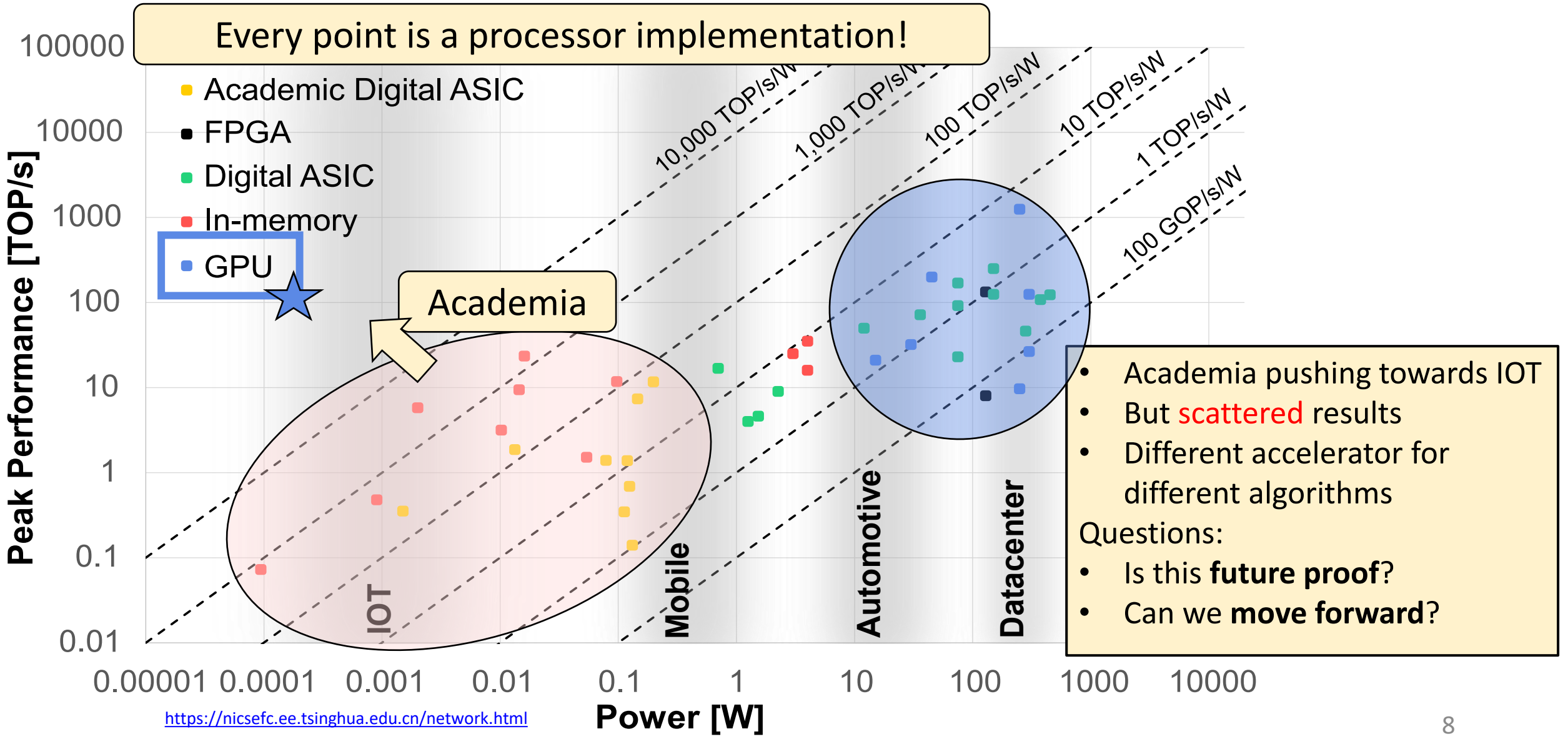




- ✓ Optimal **efficiency**
- ✓ High, scalable parallelism
- ✗ Specific for an application class
- ✗ Non-standard programming model

Field moving towards ASIC…

Trends for ML Acceleration



- Academia pushing towards IOT
- But scattered results
- Different accelerator for different algorithms

Questions:

- Is this **future proof**?
- Can we **move forward**?

https://nicsefc.ee.tsinghua.edu.cn/network.html

GPUs?

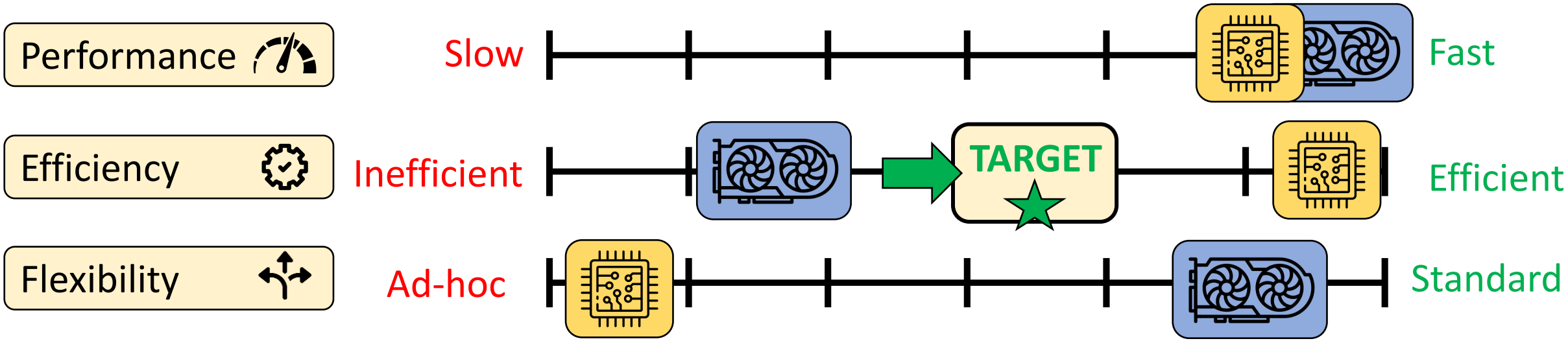
- GPUs are the **only alternative** to application specific designs (ASIC or FPGA)
  (in ML acceleration context)
- But need further **specialization to improve efficiency**

Where do we start?

- GPUs are:
  - **complex** architectures
  - Most solutions are commercial, based on proprietary **design**, **ISA**, and **SW-stack** 😔
- But **open-source**, academic **alternatives** out there!
  Not as mature as commercial solution, but **enable architecture research**

Motivation

Qualitative comparison: ASIC vs Open-Source GPU



- **OS-GPUs** have **limited efficiency**, but providing **great flexibility** and **programmability**
- **ASIC** are **very efficient**, but for a **specific task** and with **ad-hoc programming model**

TARGET

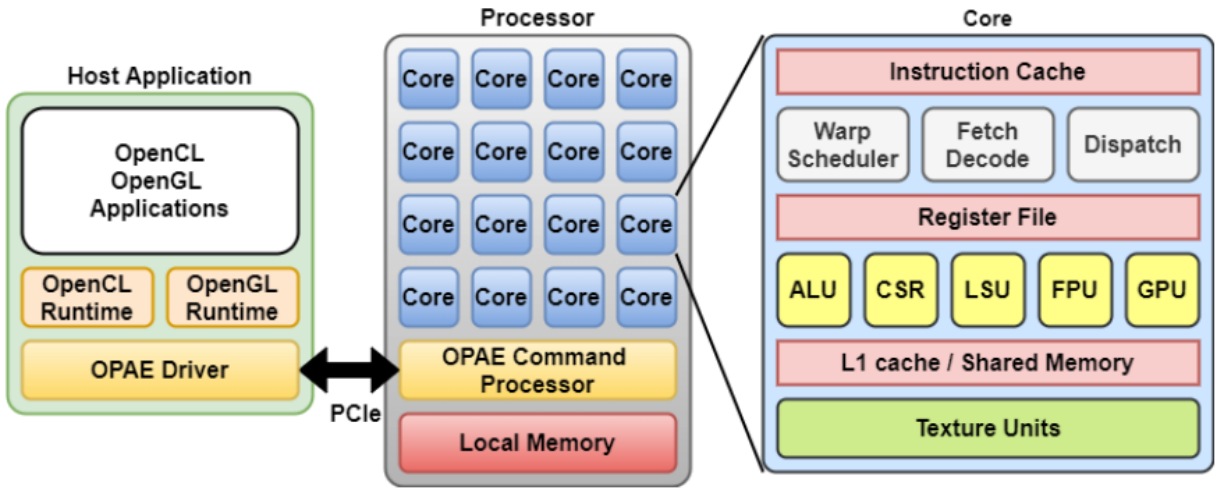
1. **Improve OS-GPU efficiency** to reduce the gap with ASIC
2. Make our result **accessible**, **modular**, **incremental**, **open-source**!

Outline

- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

The Vortex GPGPU

The Vortex GPGPU High-level architecture



- **RTL level** implementation of GPGPU
- Based on the **RISC-V ISA**
- Scalable in **cores**, **warps** and **threads**
- Uses **open-source software stack**
- Support for **OpenCL**

GPGPU control RISC-V ISA extension

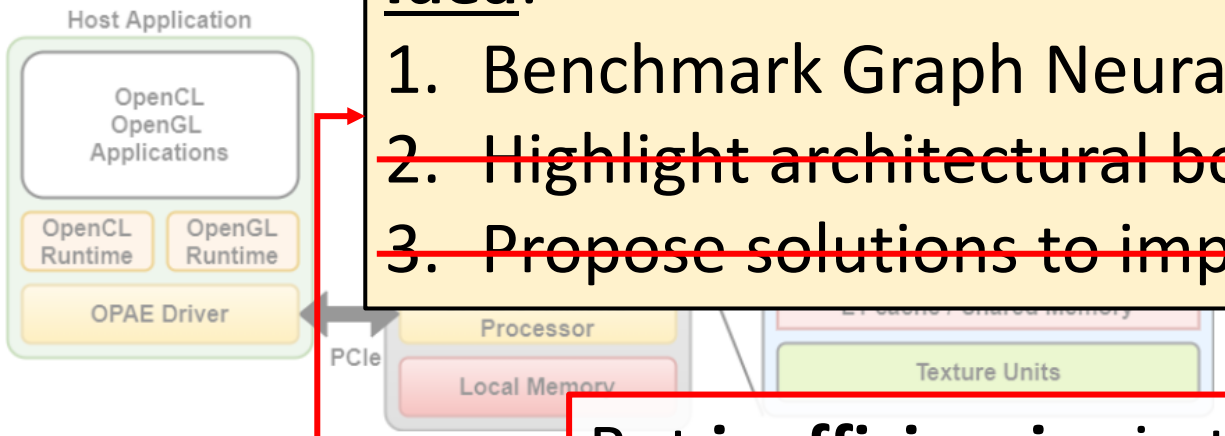
- **`wspawn`** – wavefront generation
- **`tmc`** – apply thread mask
- **`split`**/**`join`** – control flow divergence/reconvergence
- **`bar`** – wavefront barrier

Why Vortex?

1. Closed HW-SW loop + HW validation
2. Complete memory hierarchy implementation
3. Open-source SW stack: **extendable**!

The Vortex GPGPU

The Vortex GPGPU High-level architecture



Idea:

1. Benchmark Graph Neural Network (e2e) on Vortex
2. ~~Highlight architectural bottlenecks~~
3. ~~Propose solutions to improve execution~~

But **inefficiencies** in the kernel execution (@ **SW level**!)

GPGPU ISA control extension

- **`wspawn`** – wavefront generation
- **`tmc`** – apply thread mask
- **`split/join`** – control flow divergence/reconvergence
- **`bar`** – wavefront barrier

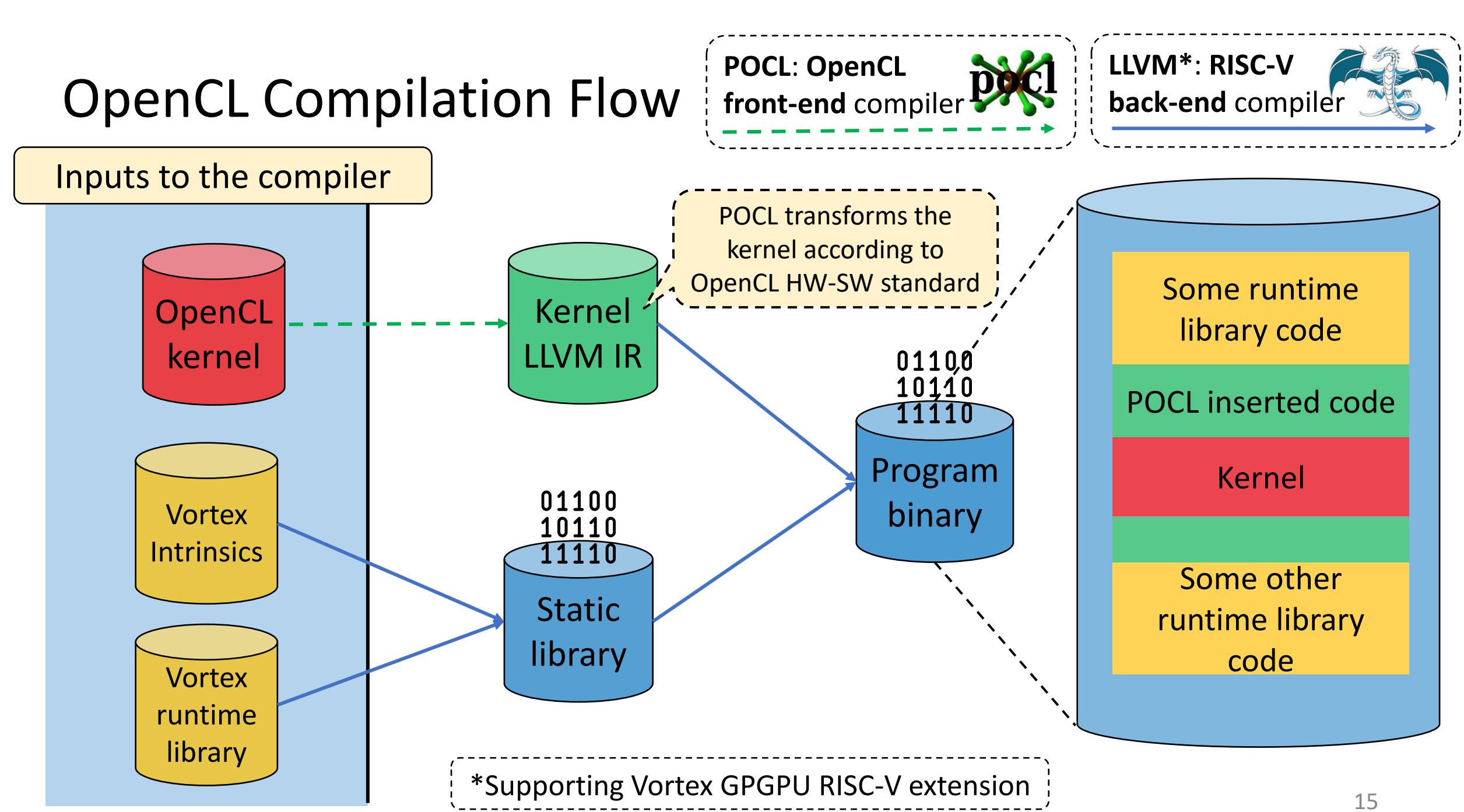
Why Vortex?

1. Closed HW-SW loop + HW validation
2. Complete memory hierarchy implementation
3. Open-source SW stack: **extendable**!

Outline

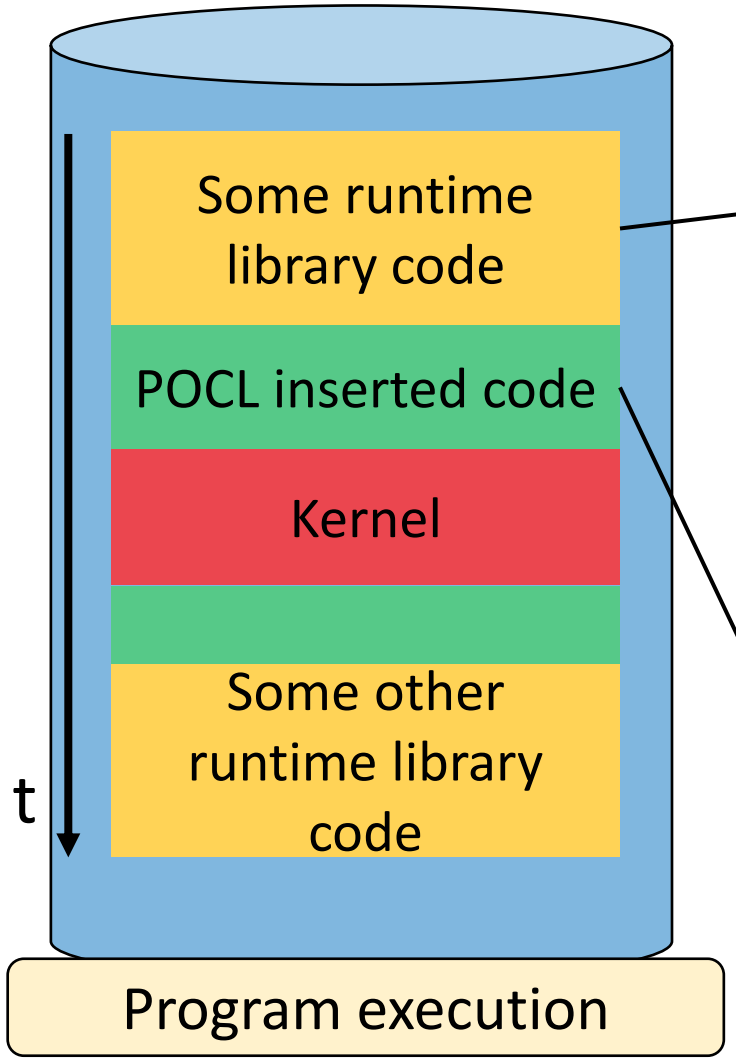
- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

OpenCL Compilation Flow

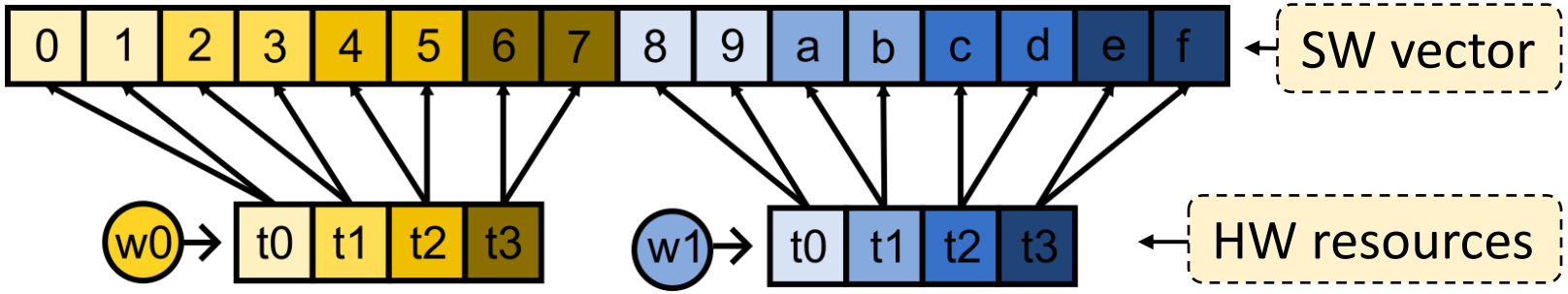


Workload distribution on vortex

Example HW configuration:
1 core, 2 warps, 4 threads/warp



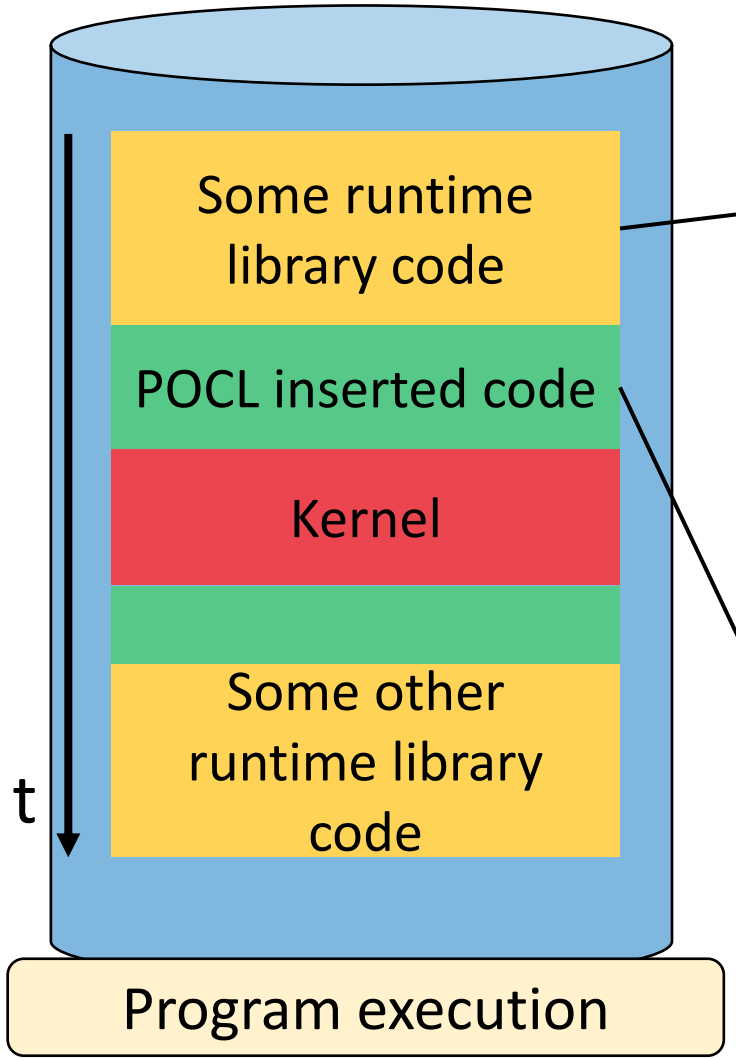
STEP 1 – **Spatial workload distribution**
Workload is evenly distributed among HW resources
Example!
vector addition C = A + B with **16** elements
Each thread will add 2 elements (*typical case*)

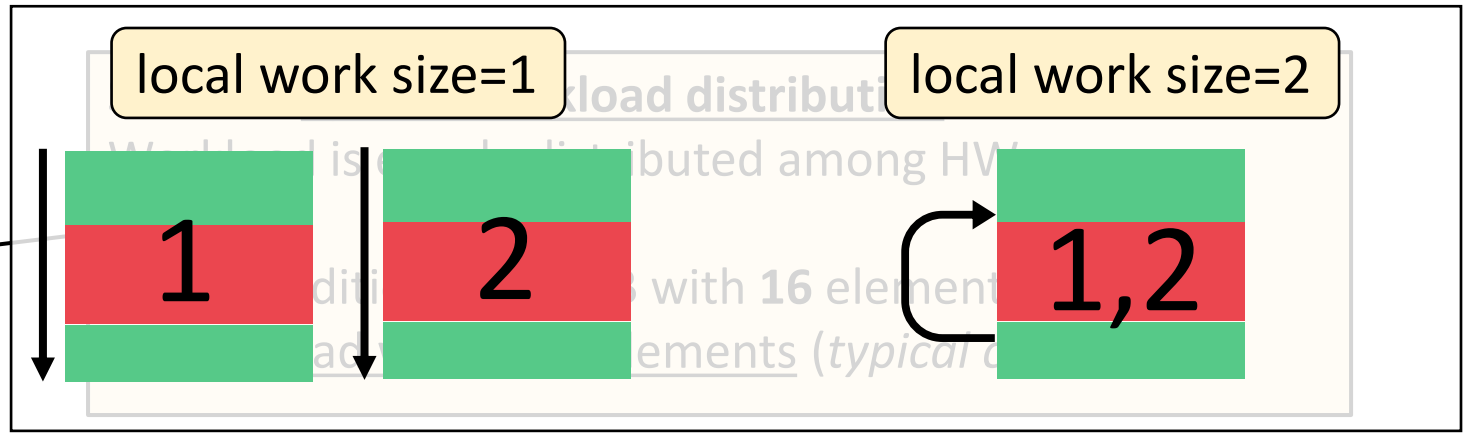


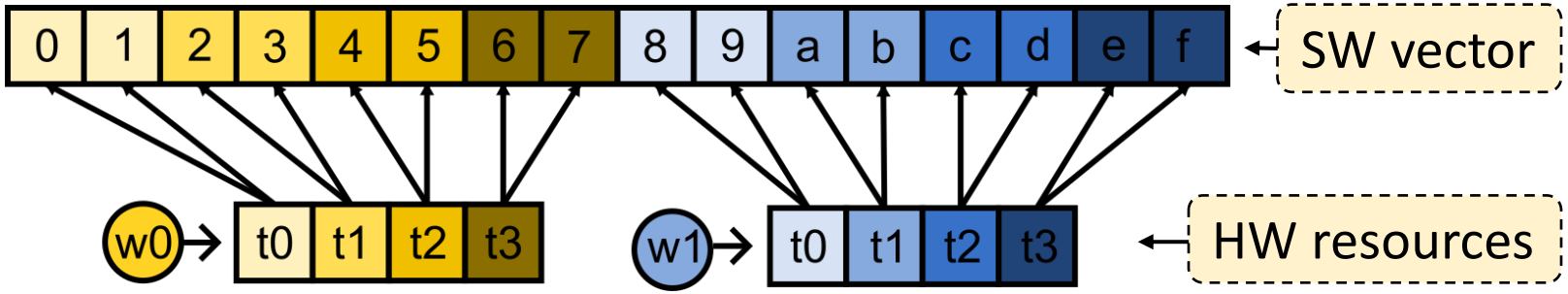
STEP 2 – **Temporal Unrolling**
Determines the elements executed by every spawned (SW) warp
This depends on a runtime parameter passed by the host global kernel call → *local work size*

Workload distribution on vortex

Example HW configuration:
1 core, 2 warps, 4 threads/warp





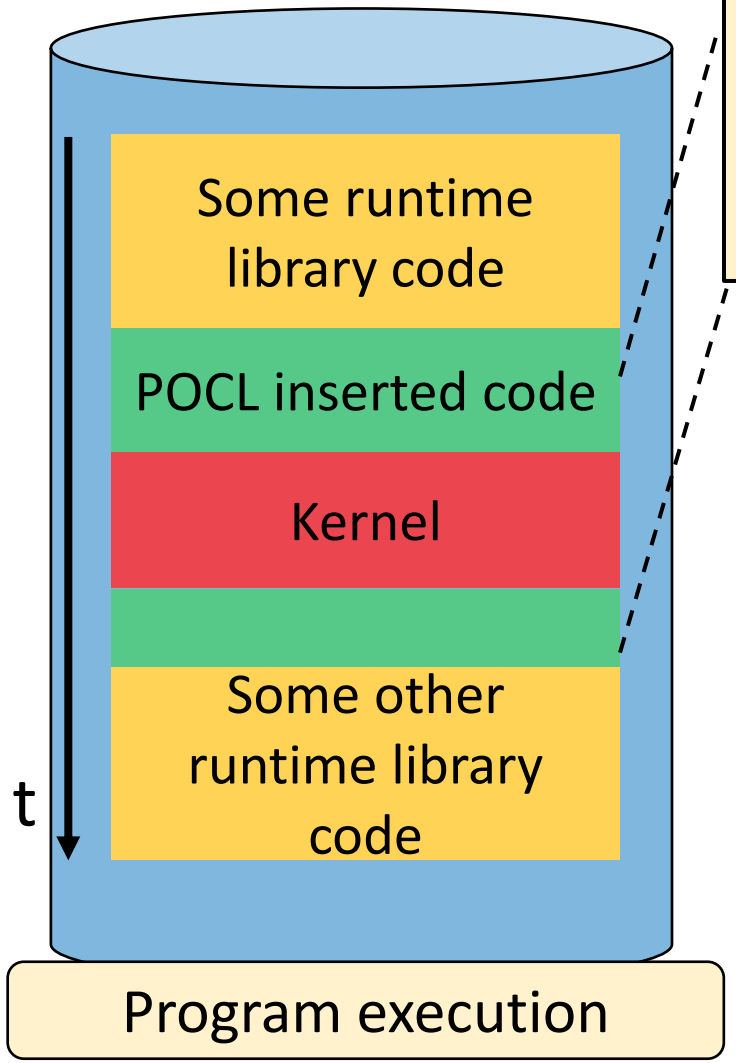


STEP 2 – **Temporal Unrolling**
Determines the elements executed by every spawned (SW) warp
This depends on a runtime parameter passed by the host global kernel call → *local work size*

Outline

- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

Temporal Unrolling



Example HW configuration:
1 core, 2 warps, 4 threads/warp

Pseudo code (executed by every thread)

```
# local_work_size (lws) specified by host
for i in range(local_work_size):
    #(tid → assigned first iteration)
    C[tid + i] = C[tid + i] + C[tid + i]
```

Example – Execution changing *lws* vecadd 16 elements

lws	wspawn	tmask
1	4	1111
2	2	1111
4	2	0011
8	2	0001

Underutilization

Problem

- The *lws* parameter impacts the execution
- "Wrong" values lead to **suboptimal HW utilization** (slower exec, more instr. issues)

How to determine optimal *lws* **dynamically** for **different kernels**?

Optimal HW-aware mapping

gws → global workload size (16 in the example)

$$lws_{opt} = \frac{gws}{cores \times warps \times threads}$$

SW

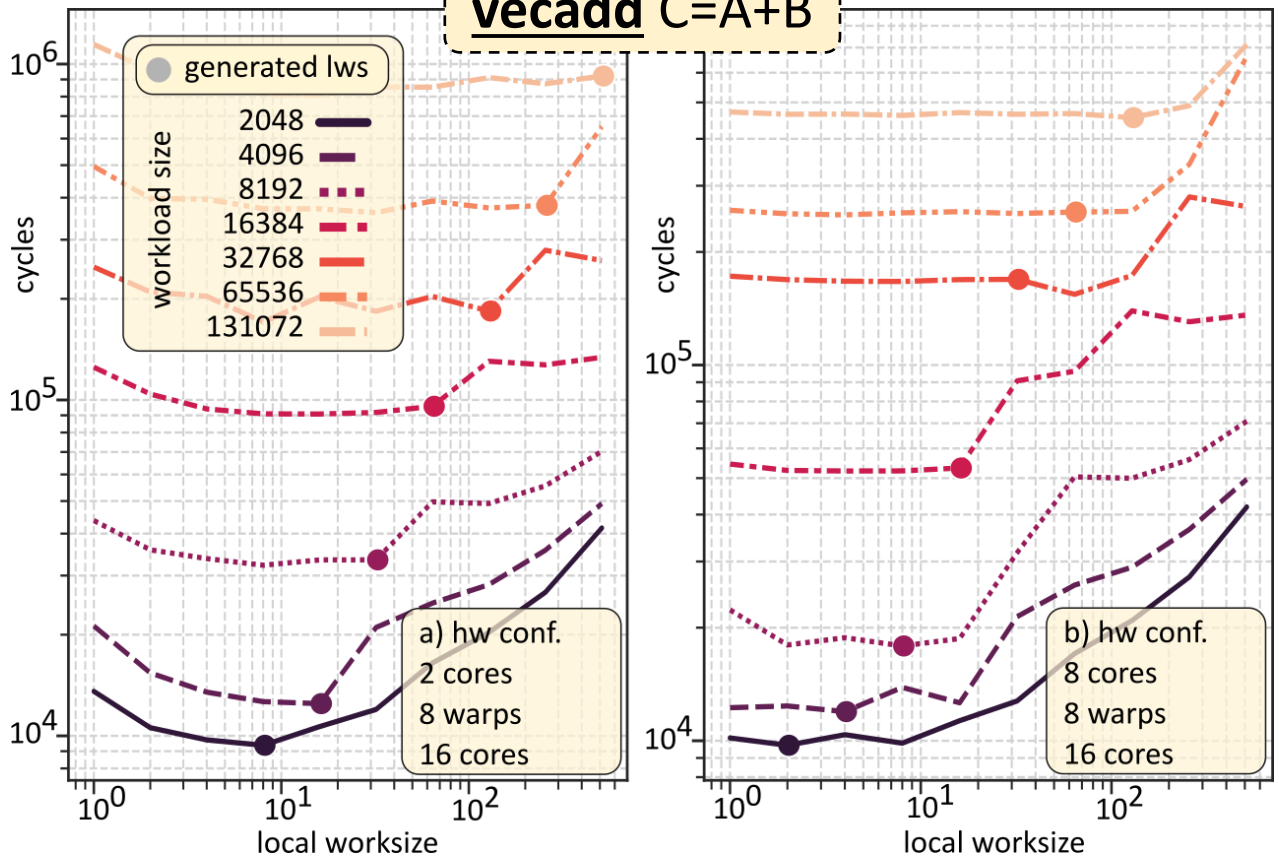
HW

Example – Execution changing *lws*

- HW configuration:
  1 core, 2 warps, 4 threads
- SW: vecadd 16 elements

lws	wspawn	tmask
1	4	1111
2	2	1111
4	2	0011
8	2	0001

vecadd C=A+B



Optimal HW-aware mapping

gws → global workload size (16 in the example)

$$lws_{opt} = \frac{gws}{cores \times warps \times threads}$$

SW

HW

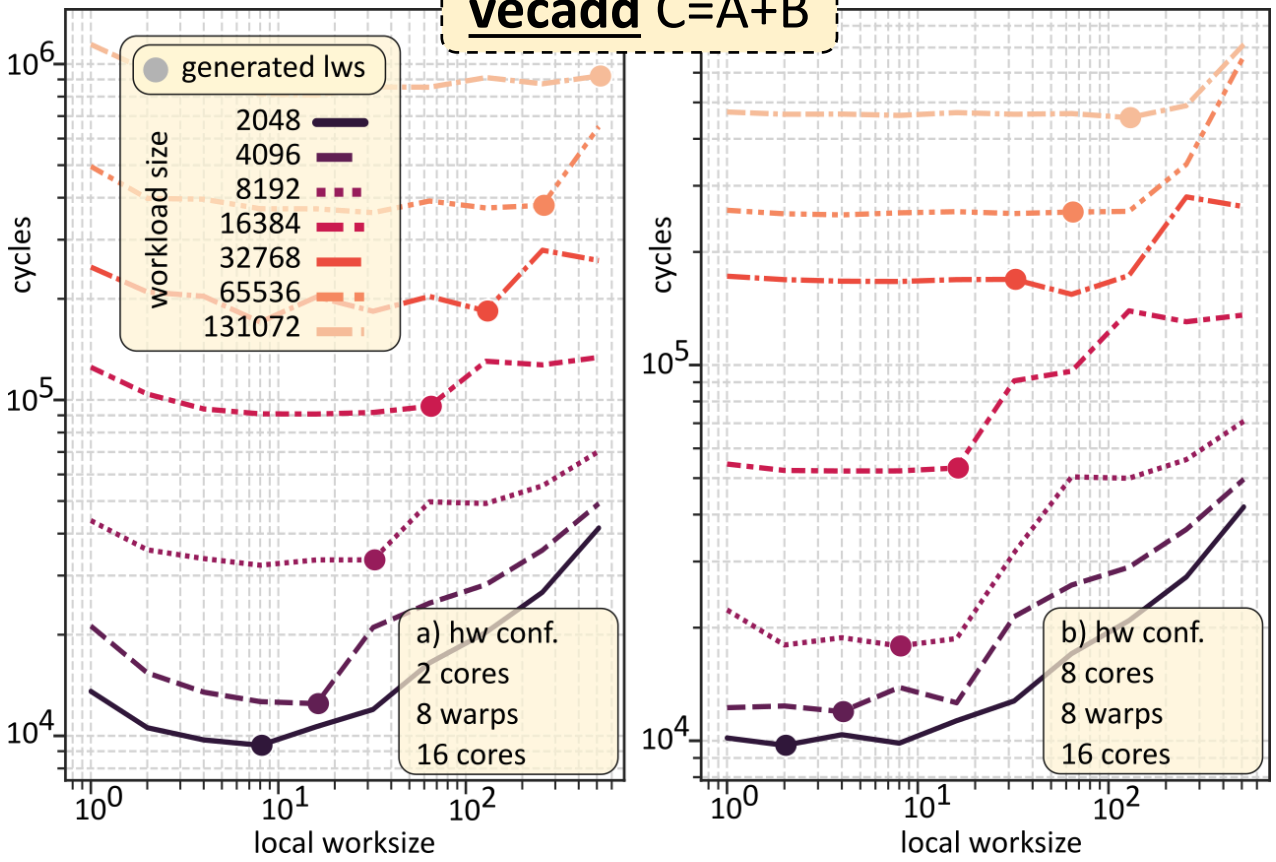
Example – Execution changing *lws*

- HW configuration:
  - 1 core, 2 warps, 4 threads
- SW: vecadd 16 elements

lws	wspawn	tmask
1	4	1111
2	2	1111
4	2	0011
8	2	0001

- Optimal mapping needs **HW** and **SW information**
- It can be evaluated **dynamically** at **runtime**
- Our mapping **abstracts** HW to programmer

vecadd C=A+B



Outline

- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

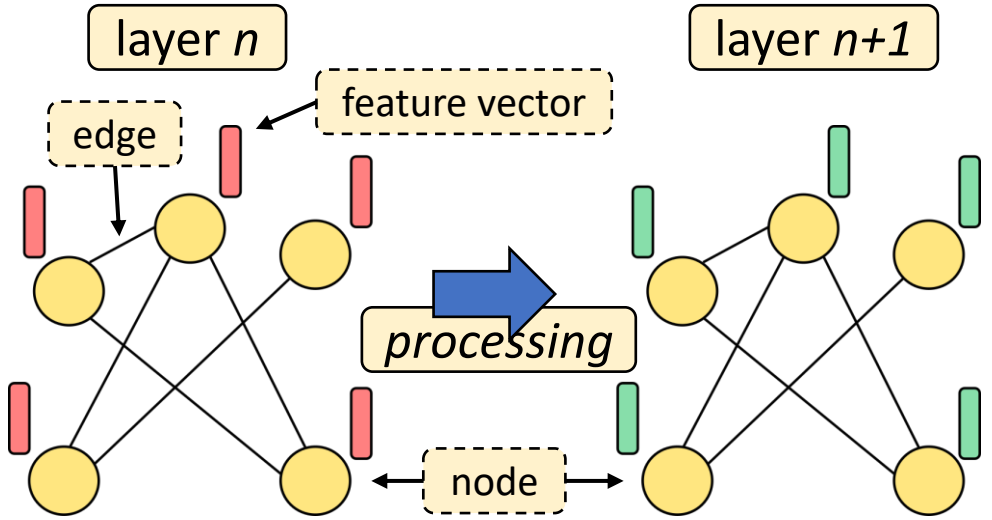
Graph Neural Networks: Overview

Graph Neural Networks (GNN)

Class of Machine Learning models

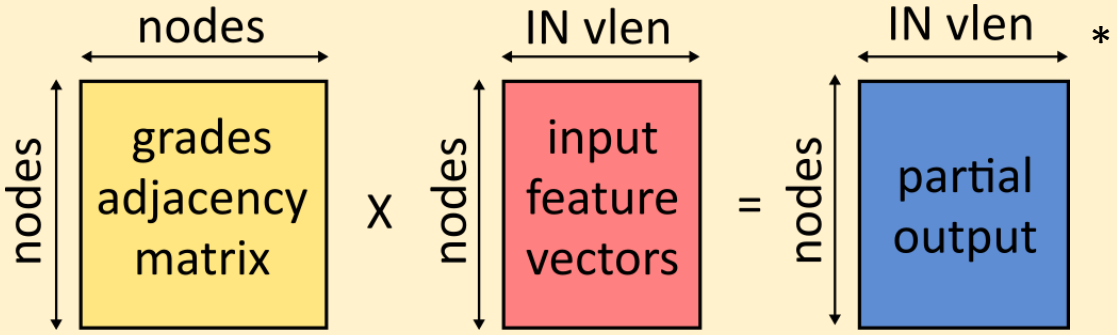
- Exploit **information embedded in graph structure**
- Combine with **neural networks**

to perform specific tasks (classification, prediction, …)



Aggregation – gathers info from neighbors

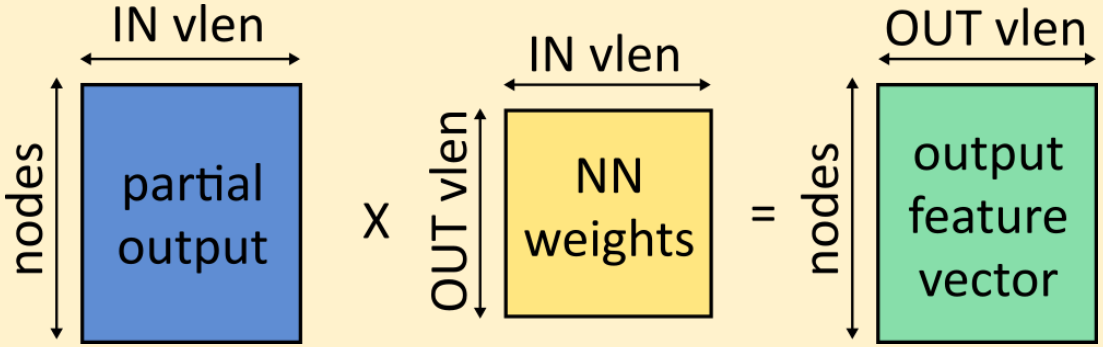
Graph Conv Networks → **Weighted average**



* we use CSR sparse sgemm in our benchmarks

Combination – applies some processing

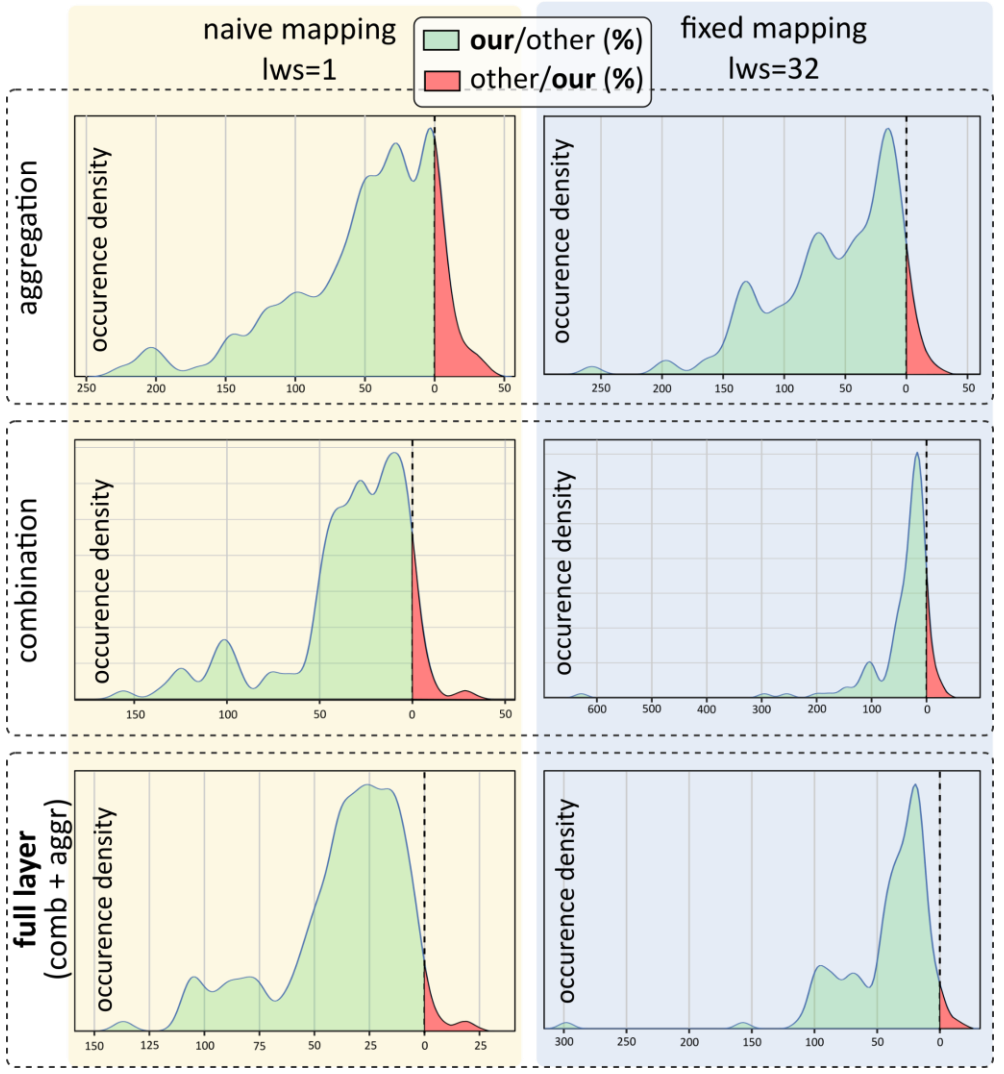
Graph Conv Networks → **Fully connected layer**



Validation methodology

1. **Sampled 15** different Vortex **architecture configurations**
2. **Evaluated** our mapping on GCN layers **over**:
   a. cora, citeseer and pubmed *datasets* (different graph size, different structure)
   b. 16, 32, and 64 *hidden feature size*
   c. on single *aggregation*, *combination* and *full layer*
3. **Compared execution latency** results with:
   a. naïve mapping (lws=1)
   b. fixed mapping (lws=32)

FoM (lower is better)	**aggr**	**sgemm**	**full layer**
Our mapping slower than others (>5%)	10/135 (7.4%)	9/135 (6.7%)	**2**/135 (**1.5%**)



Validation methodology

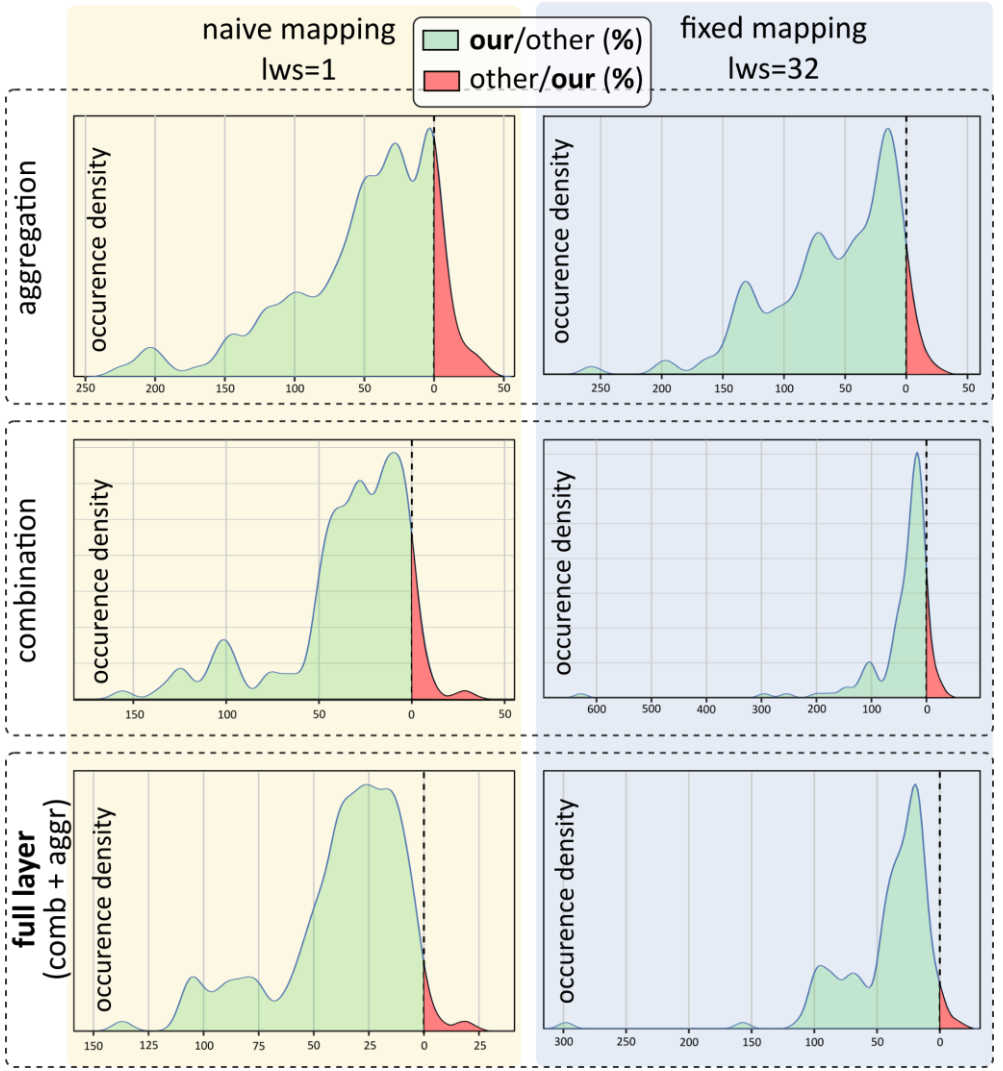
1. **Sampled 15** different Vortex **architecture configurations**
2. **Evaluated** our mapping on GCN layers **over**:
   - cora, citeseer and pubmed *datasets*
   - (different graph size, different structure)
   - 16 and 64 hidden features
3. **Compared execution latency** results with:
   a. naïve mapping (lws=1)
   b. fixed mapping (lws=32)

Results

Our mapping:

1. **Always** deliver comparable execution latency
2. Shows higher **benefits with combined** kernel calls

FoM (lower is better)	**aggr**	**sgemm**	**full layer**
Our mapping slower than others (>5%)	10/135 (7.4%)	9/135 (6.7%)	**2**/135 (**1.5%**)

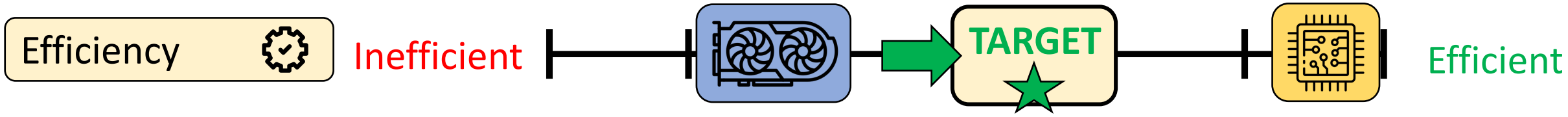


Outline

- Background and motivation
- The Vortex GPGPU
  - Compilation flow and workload partitioning
- HW-aware optimal workload mapping
- Validation on Graph Neural Networks
- Conclusion and Future Work

Conclusion

- ML acceleration research optimizes architectures for specific models
- A different approach is bridging the gap between open-source GPUs and ASIC



We:

- Investigated **limitations** of the Vortex GPGPU platform
- Proposed an **optimal, HW-aware mapping** (dynamic at runtime) that ensures an **efficient execution**, minimizing cycle latency
- Validated on several configurations of **GCN layers**
  Our mapping performs better in 98.5% of the cases

Pull request to the Vortex repo

We fixed a bug 🐛 in the Vortex runtime library (prevents correct execution!)

Pull request → Fixed #79 bug in vx_spawn.c for rT kernel iteration allocation

Now we can validate in way more HW configurations

Apply our patch if you are going to use Vortex 😜

Future Work

- Improve the loop overhead
- Focus where we started → GNN execution

Thank you for your attention!