High-performance Graph Processing on GPUs

Original slides by: Sreepathi Pai
University of Rochester, October 12, 2018

Adapted by Tyler Sorensen for CSE211 at UCSC
Nov. 15, 2022

Sreepathi Pai acks: Keshav Pingali, Alastair Donaldson, Muhammad Amber Hassaan, Tal Ben-Nun, Michael Sutton, Chad Voegele, Yi-Shan Lu, Ahmet Celik, Milos Gligoric, Sarfraz Khurshid

Announcements

Remote due to strike

- Let's plan to stay remote during the strike

Keep me updated of any issues that you might be facing due to strike disruption

Announcements

Grades have been posted for Midterm and attendance. Let us know if there are any issues.

We’re still aiming to get HW 2 graded this week. I will keep you updated

Homework 3 is due in less than 1 week (Nov. 21). Please get started if you haven’t.

Announcements

Moving office hours to Friday this week (3-5pm)

Announcements

Propose paper for paper reading review through Canvas (due today)

Determine if you are taking the Final or doing a Final Project. Please submit in Canvas. (due today)

Final project report can be a ”blog post” (it will be submitted as a PR to the class website). This is an option if you want to make the report public. Otherwise, it is a written report.

Announcements

Final project presentations:

Likely won’t have time for all of them. Instead, please prepare a 12 minute presentation (we will be strict on time!)

I will randomly select 6 groups to present on the last day of class.

The rest will record a screen recording of the presentation.

Announcements

Guest lecture: Felix Klock from amazon will be talking about the Rust compiler. (Nov. 29)

One more homework for the class (might be due after the last day of class to give you 2 weeks).

Get started early!

Review:

Halide (other presentation)

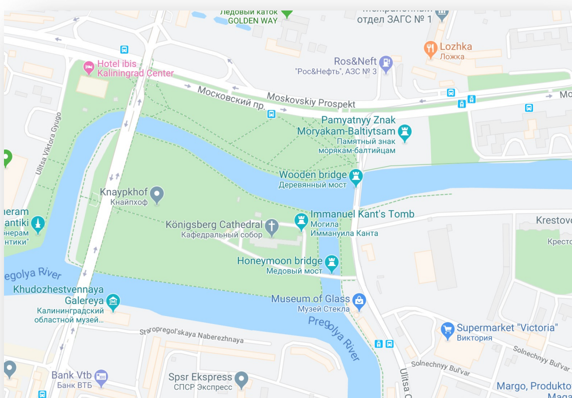
Remainder of class

Halide (other presentation)

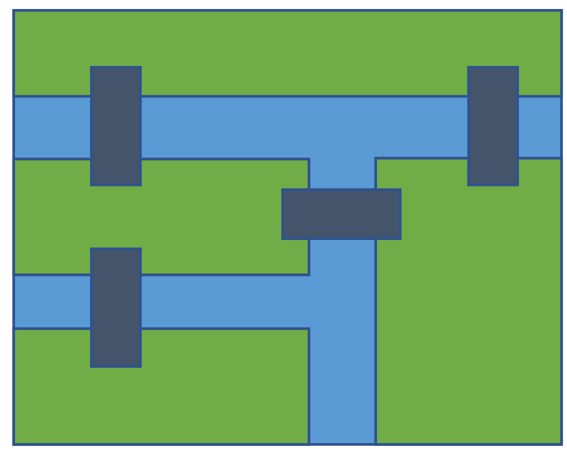
Graph Processing

Graphs (1736 Edition)

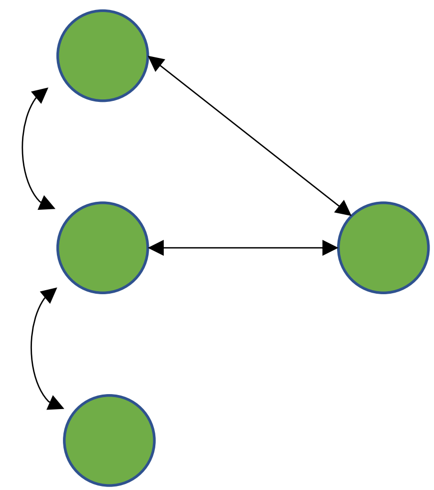
Euler’s Königsberg Bridges



Modern day



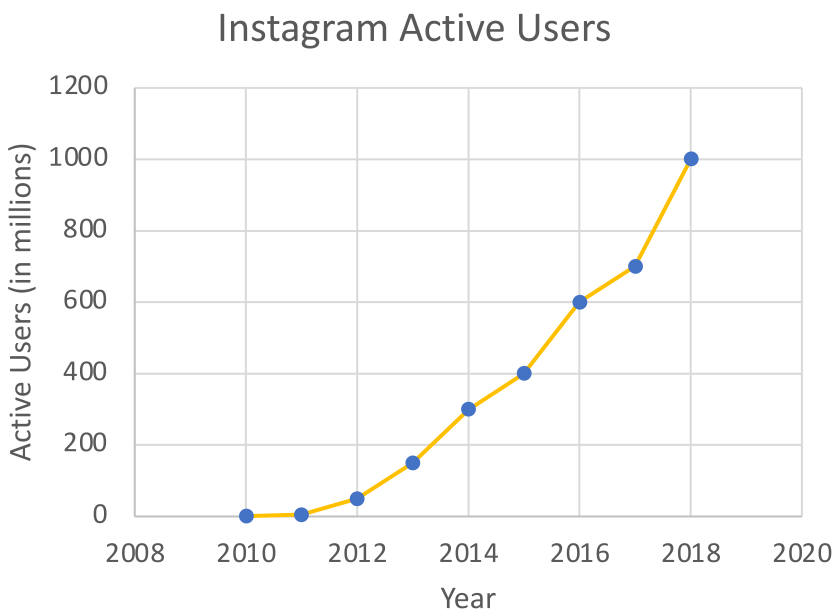
Abstract View

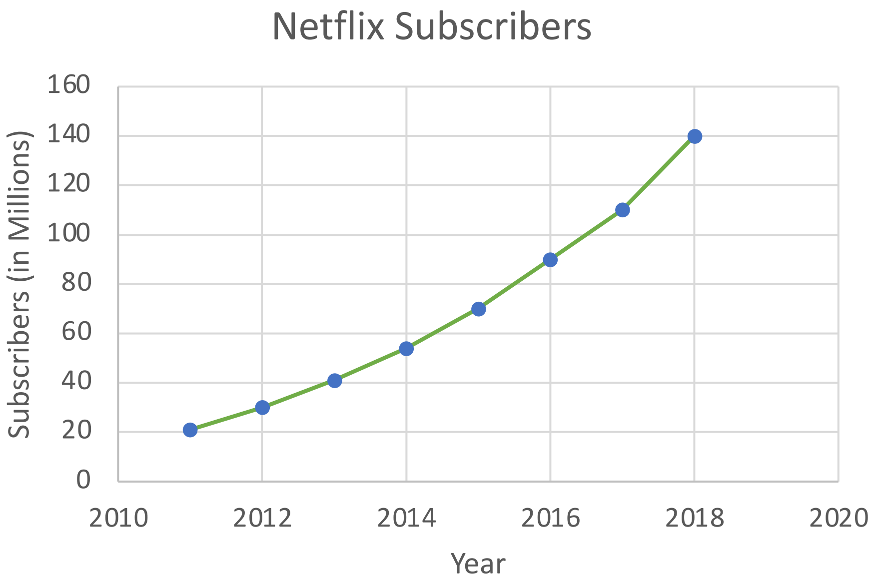


As a Graph

Graphs in 2019

Size/Growth of modern graphs



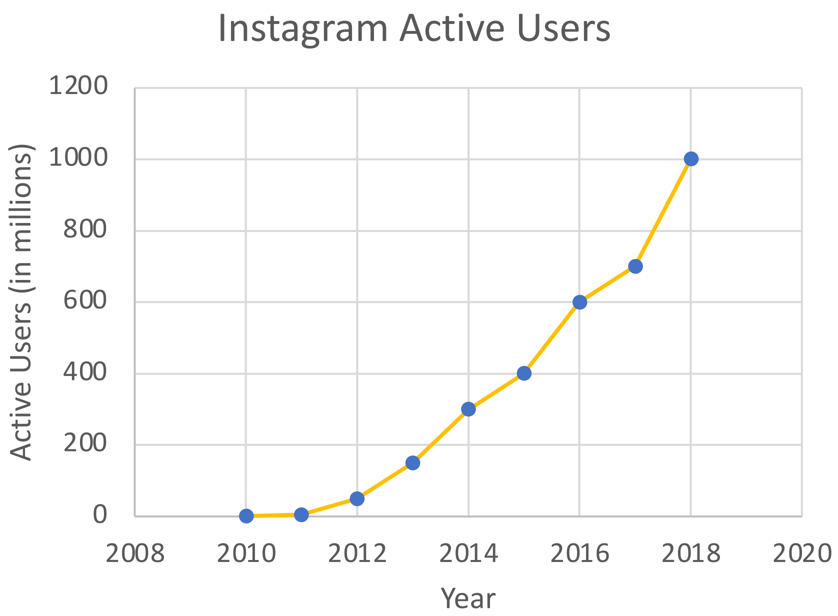


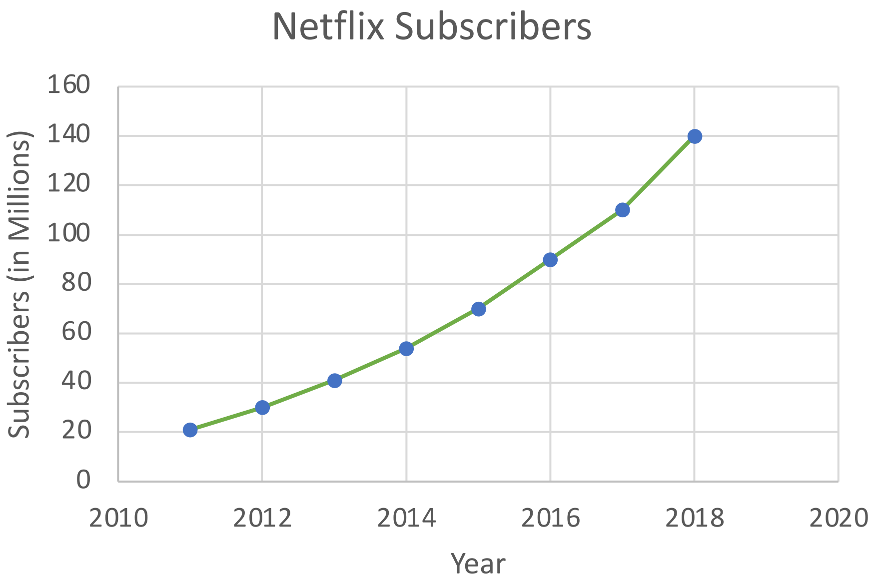
https://techcrunch.com/2018/06/20/instagram-1-billion-users/
https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/

Graphs in 2019

- Applications:
  - recommendation systems

Size/Growth of modern graphs



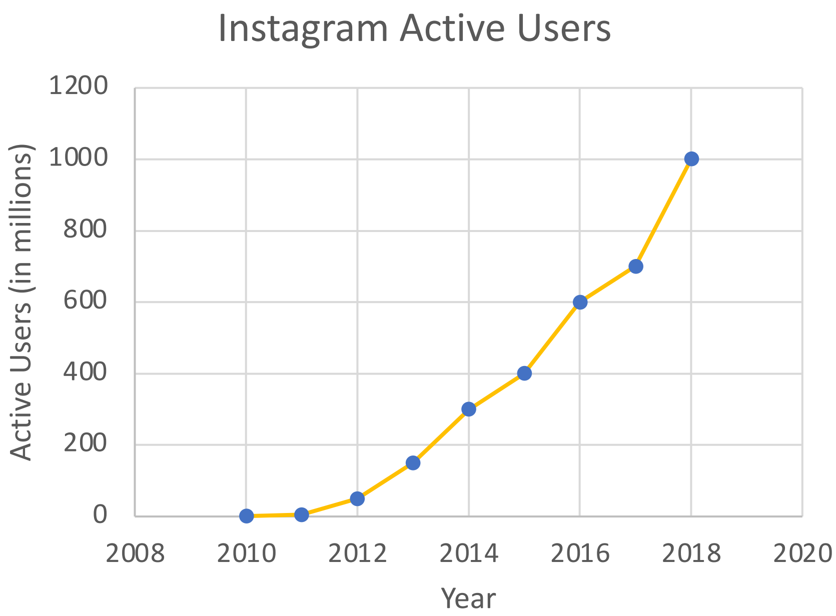


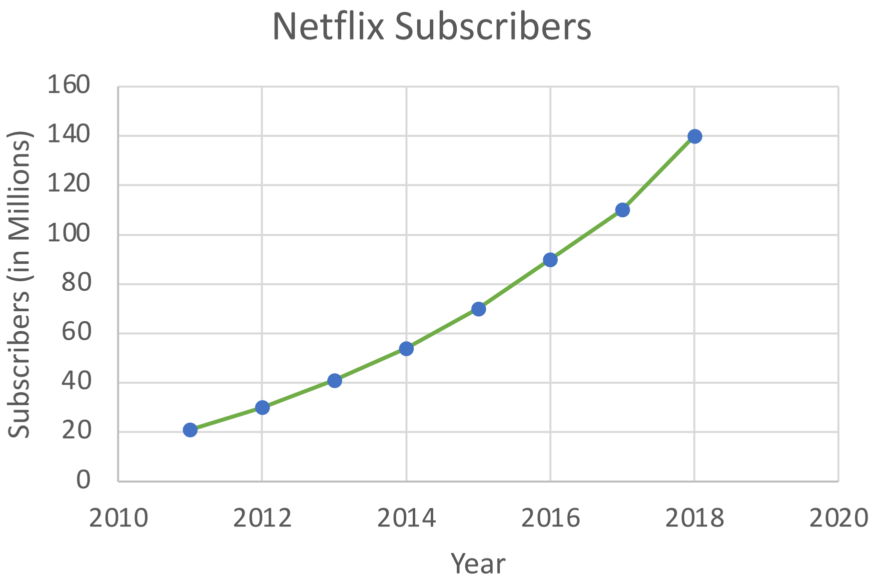
https://techcrunch.com/2018/06/20/instagram-1-billion-users/
https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/

Graphs in 2019

Size/Growth of modern graphs

- Applications:
  - recommendation systems
  - (mis)information spread



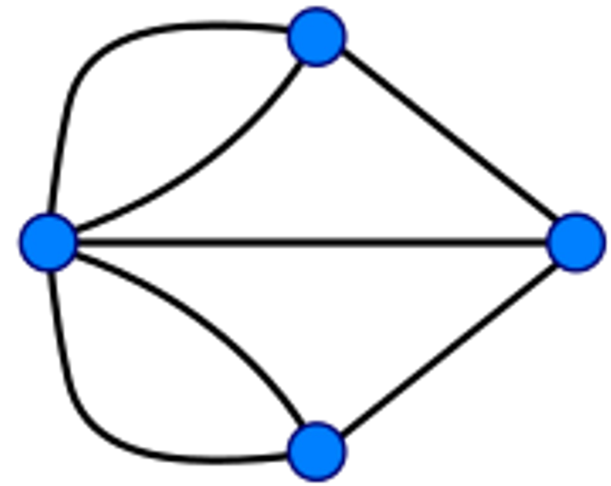


https://techcrunch.com/2018/06/20/instagram-1-billion-users/
https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/

What is graph processing?

Graphs are ubiquitous

- Social Networks
- Road Networks

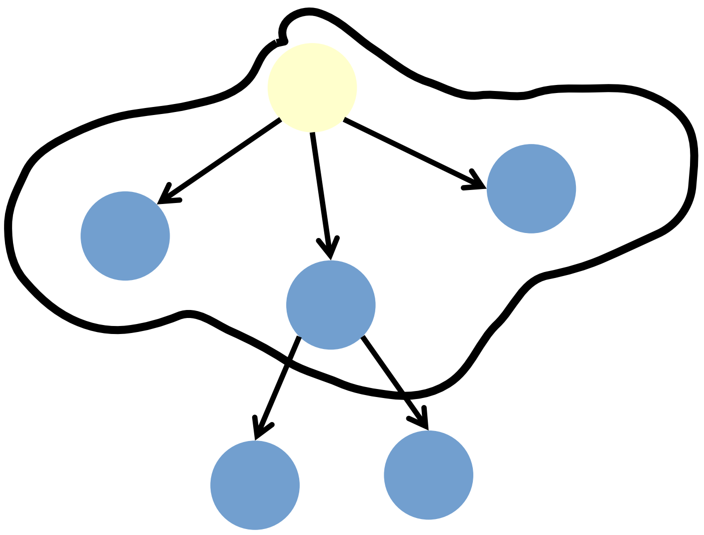


Graphs of interest are large:

- Millions of nodes, Billions of edges

Parallel Graph processing is necessary!

Graph Processing Platforms



Cluster Processing Systems

- Apache Giraph (Facebook)
- GraphLab (CMU)
- GraphX (UC Berkeley)

Vertex-centric Programming Model

- Highly parallelizable
- Limited expressivity
- Optimized for scale-free graphs
- Scalable, but not performant

Scalability, but at what COST? [McSherry et al. 2015]

Connected Components

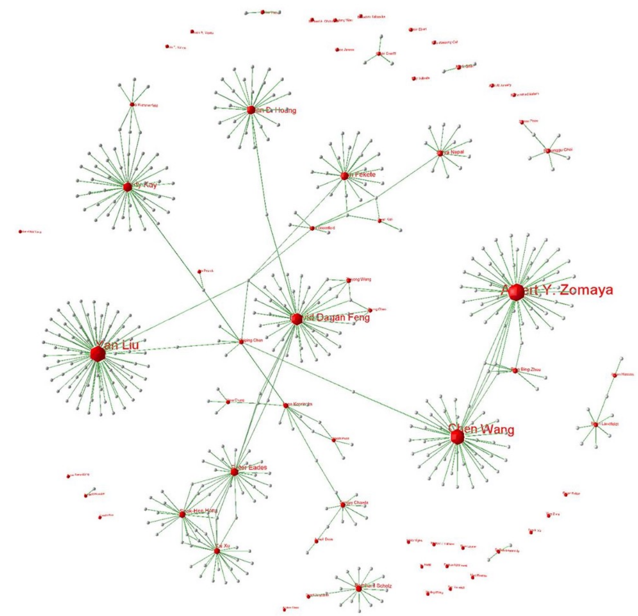
System/Algorithm	Cores	*Twitter*	*UK-2007-05*
GraphLab	128	242s	714s
GraphX	128	251s	800s
Label Propagation	1	153s	417s

Twitter: 41M vertices, 1.4B edges
UK-2007-05: 105M vertices, 3.7B edges

McSherry F., Isard M., and Murray D. G., Scalability, but at what COST?, HotOS 2015

Parallel Graph Processing Pitfalls

USA Road Network
24M nodes, 58M edges
High diameter, Low Uniform Degrees



LiveJournal Social Network
5M nodes, 69M edges
Low diameter, Highly-skewed Degrees

299ms	BFS(1)	84ms
692ms	BFS(2)	41ms

Perfect storm for a DSL

State-of-the-art DSLs massively underperform

Handwritten optimized code exists (guide for DSL)

Optimizations are not portable:

- i.e. it makes sense to decouple optimizations from algorithm, similar to Halide

IrGL (intermediate graph representation)

IrGL is a language for graph algorithm kernels

- *Slightly* higher-level than CUDA

IrGL kernels are compiled to CUDA code

- Incorporated into larger applications

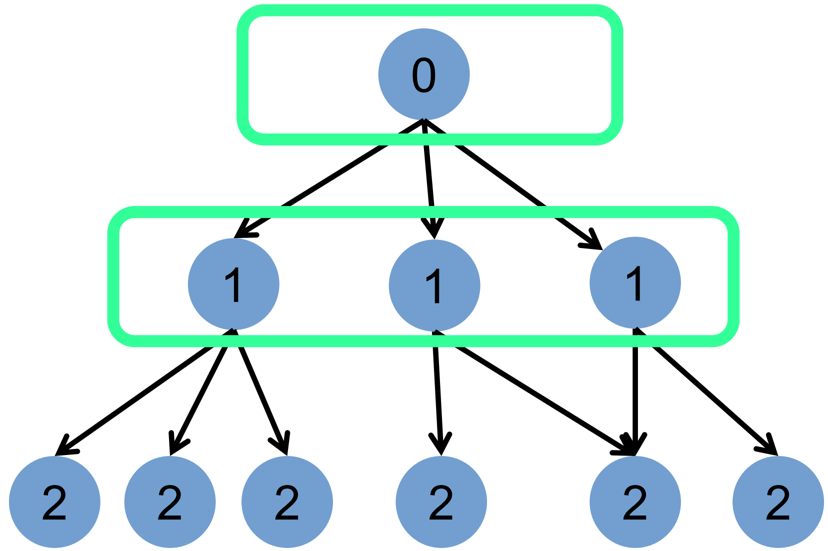
IrGL compiler applies 3 *throughput* optimizations

- User can select exact combination
- Yields multiple implementations of algorithm

Compiler generates all the interesting variants!

Bottlenecks in GPU Graph Processing

Example: Level-by-Level BFS



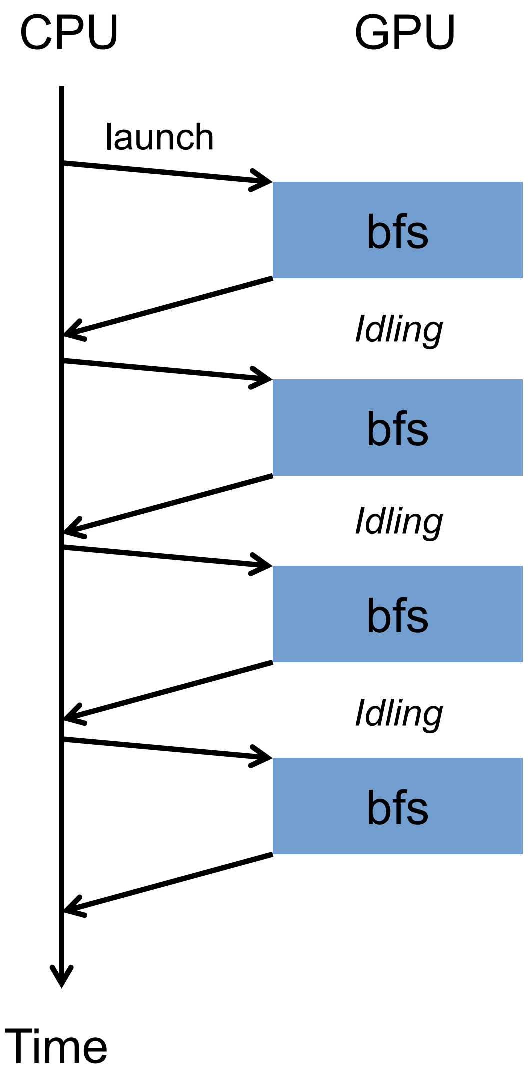

```
Kernel bfs(graph, LEVEL)
       ForAll(node in Worklist)
              ForAll(edge in graph.edges(node))
                     if(edge.dst.level == INF)
                            edge.dst.level = LEVEL
                            Worklist.push(edge.dst)

src.level = 0
Iterate bfs(graph, LEVEL) [src] {
       LEVEL++
}
```

Bottleneck #1: Short Kernels

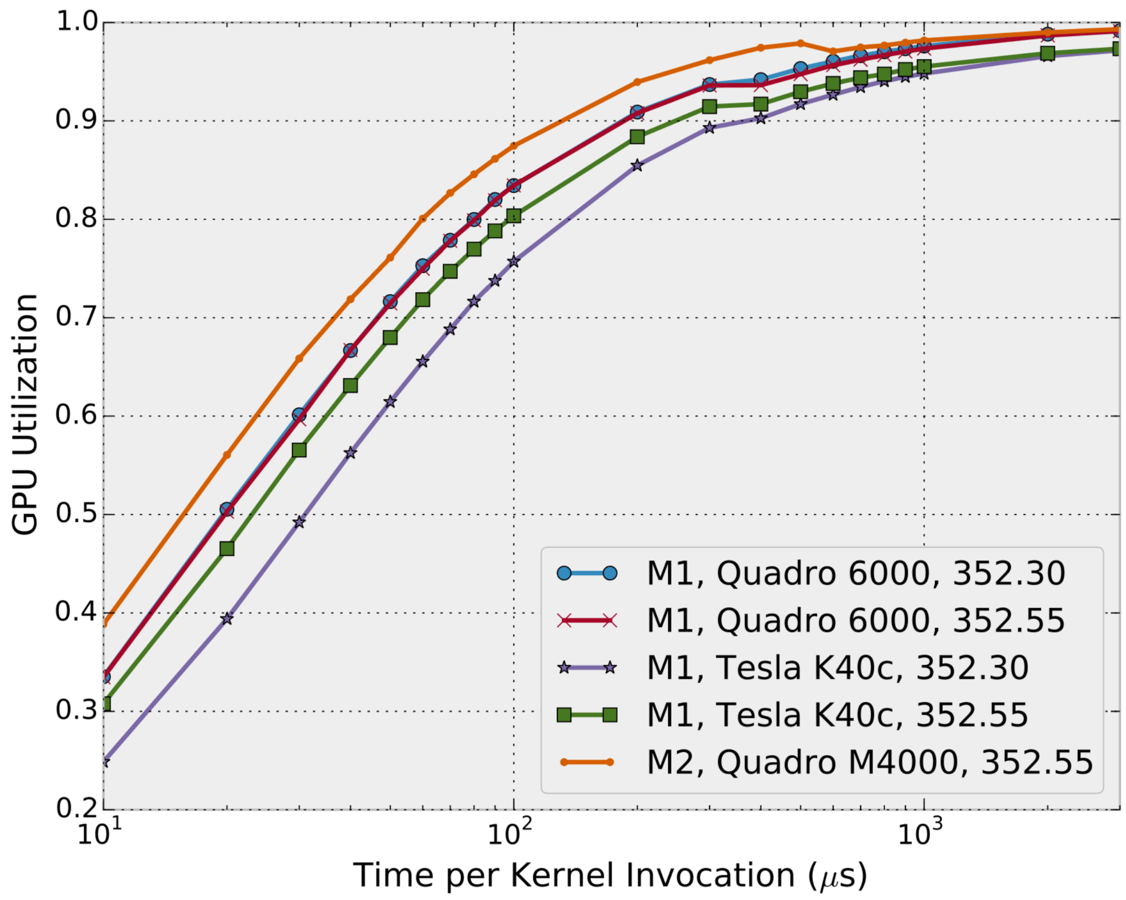
```
Kernel bfs(graph, LEVEL)
      ForAll(node in Worklist)
            ForAll(edge in graph.edges(node))
                  if(edge.dst.level == INF)
                        edge.dst.level = LEVEL
                        Worklist.push(edge.dst)

src.level = 0
Iterate bfs(graph, LEVEL) [src] {
      LEVEL++
}
```

- USA road network: 6261 `bfs` calls
- Average bfs call duration: 16 µs
- Total time should be 16*6261 = 100 ms
- Actual time is 320 ms: 3.2x slower!

Iterative Algorithm Timeline



GPU Utilization for Short Kernels



Improving Utilization

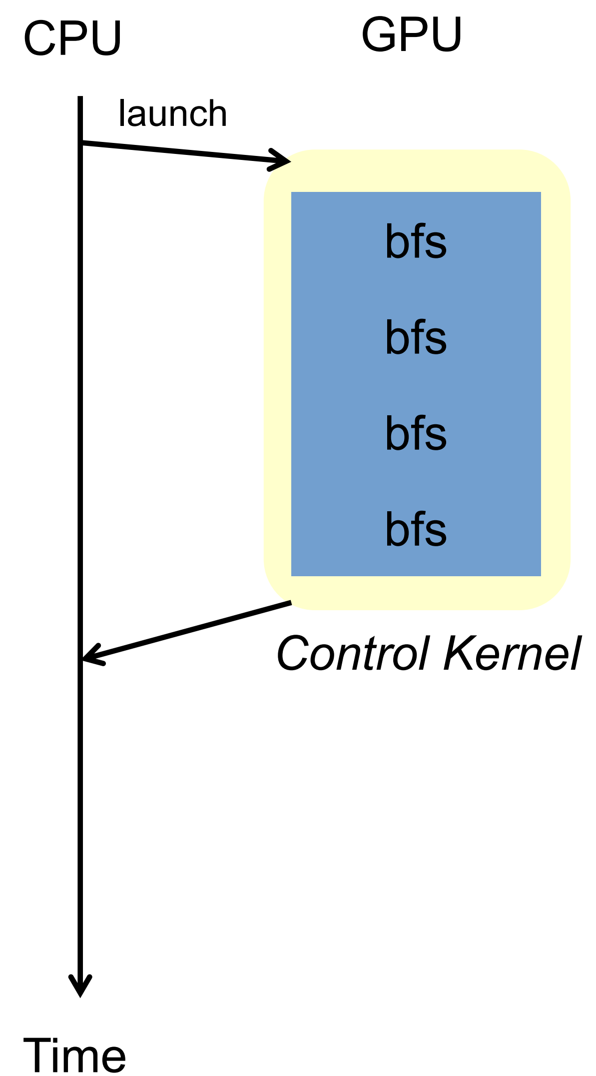
Generate Control Kernel to execute on GPU

Control kernel uses function calls on GPU for each iteration

Separates iterations with device-wide barriers

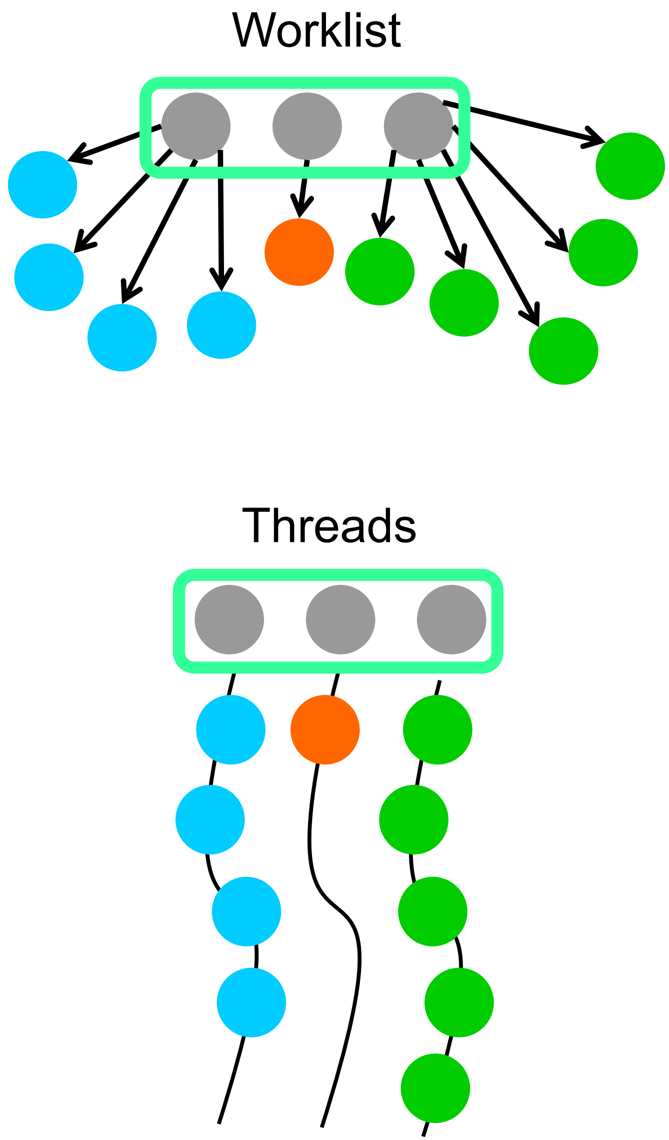
- Tricky to get right!

Device-wide barriers now supported in CUDA 9



Bottleneck #2: Load Imbalance from Inner-loop Serialization

```
Kernel bfs(graph, LEVEL)
  ForAll(node in Worklist)
    ForAll(edge in graph.edges(node))
      if(edge.dst.level == INF)
        edge.dst.level = LEVEL
        Worklist.push(edge.dst)

src.level = 0
Iterate bfs(graph, LEVEL) [src] {
  LEVEL++
}
```



Exploiting Nested Parallelism

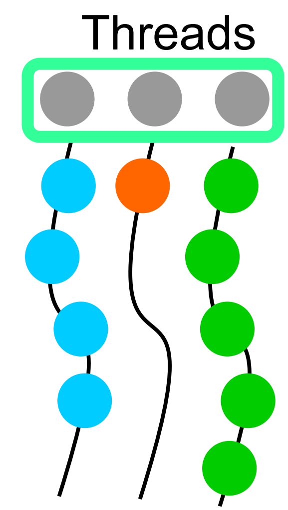
Generate code to execute inner loop in parallel

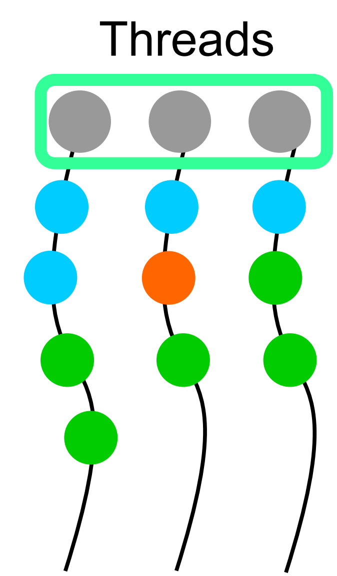
- Inner loop trip counts not known until runtime

Use Inspector/Executor approach at runtime

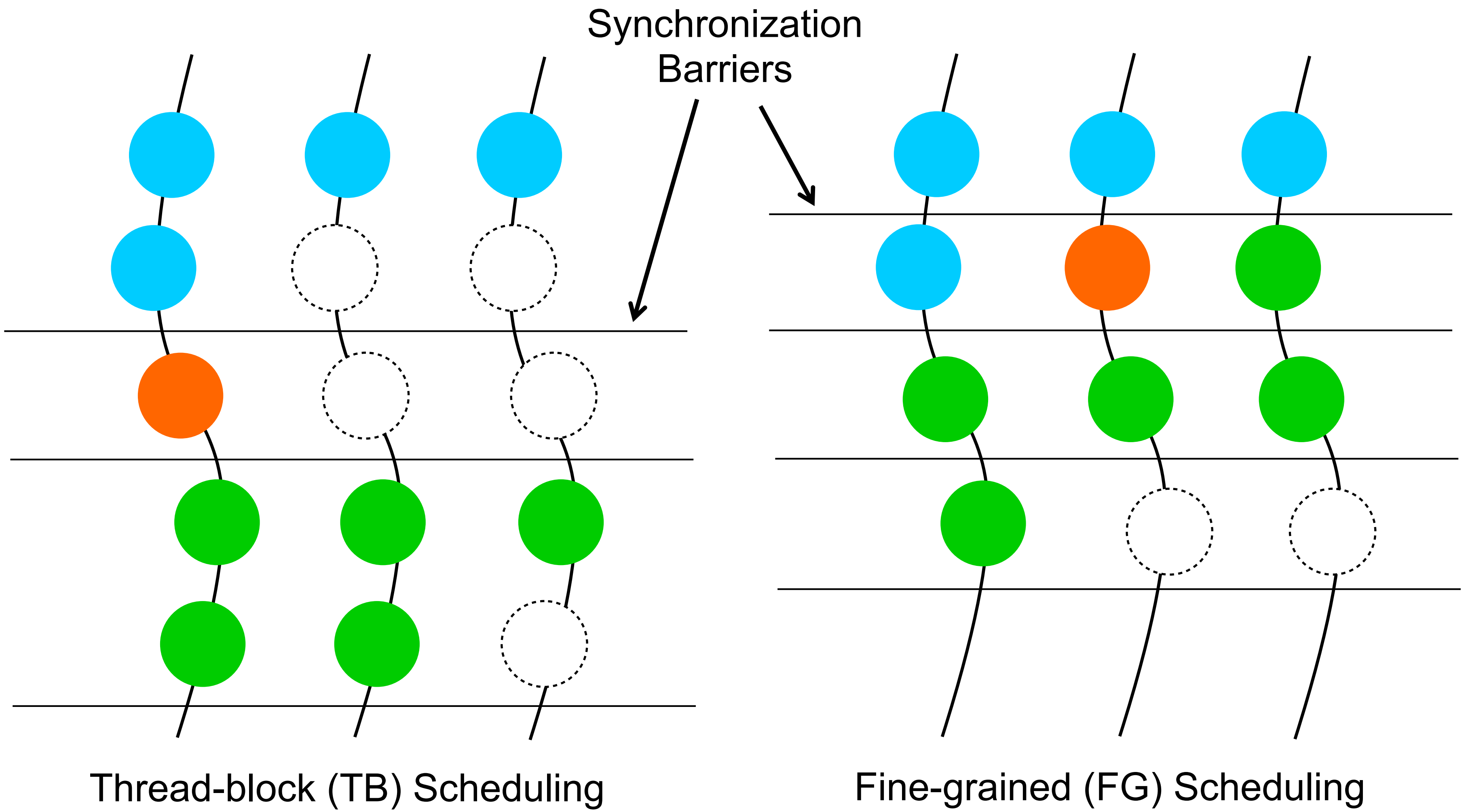
Primary challenges:

- Minimize Executor overhead
- Best-performing Executor varies by algorithm and input





Scheduling Inner Loop Iterations

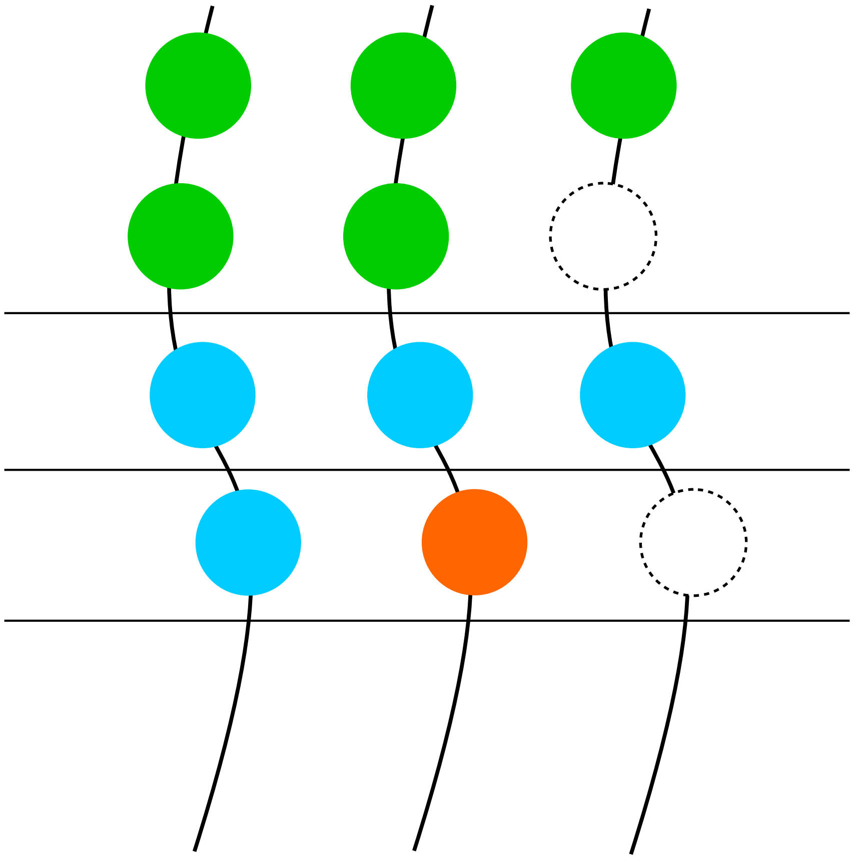


Example schedulers from Merrill et al., Scalable GPU Graph Traversal, PPoPP 2012

Multi-Scheduler Execution

Use thread-block (TB)
for high-degree nodes

Use fine-grained (FG)
for low-degree nodes



Thread-block (TB) + Finegrained (FG) Scheduling

Example schedulers from Merrill et al., Scalable GPU Graph Traversal, PPoPP 2012

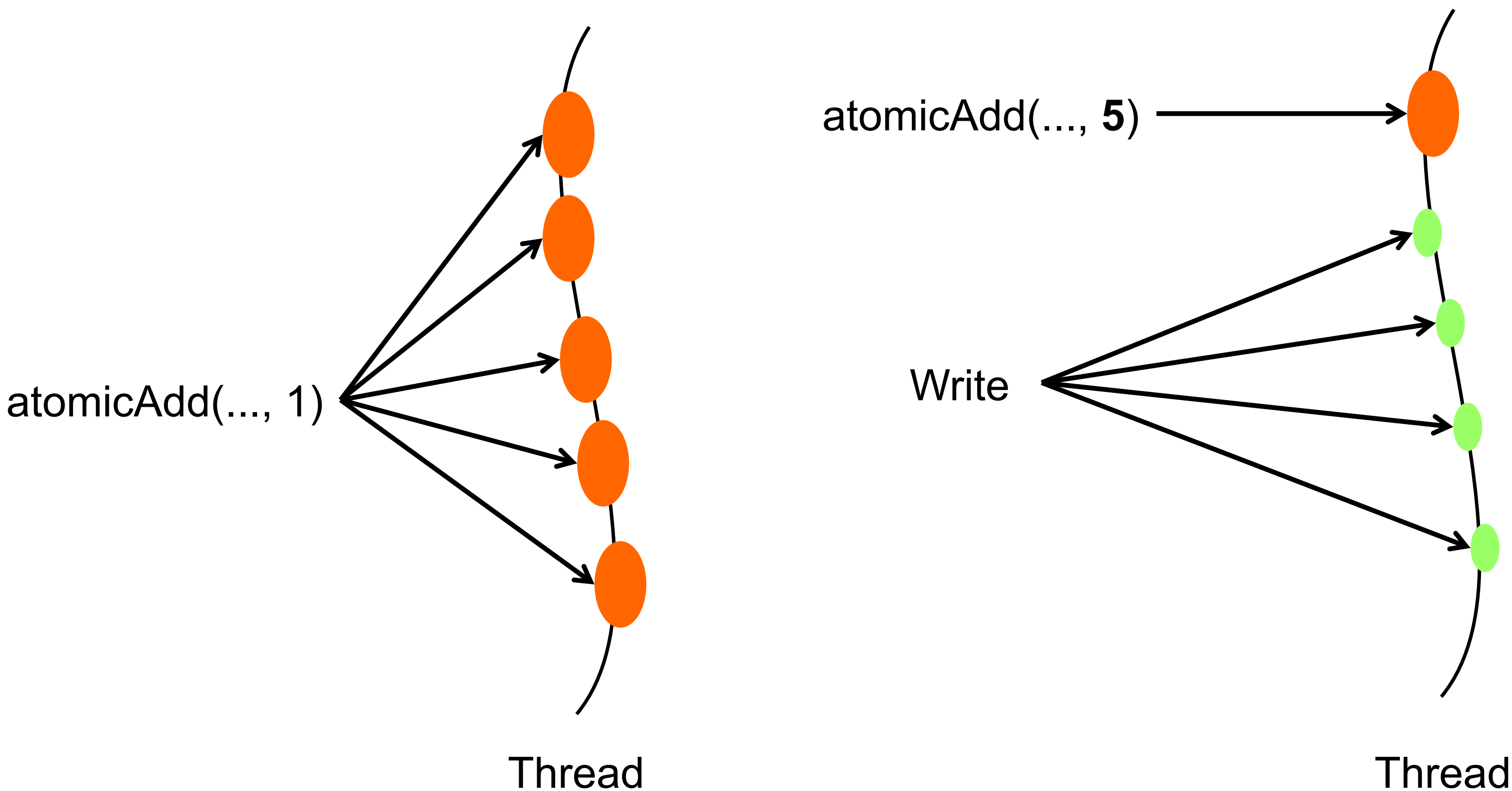
Bottleneck #3: Atomics

```
Kernel bfs(graph, LEVEL)
      ForAll(node in Worklist)
            ForAll(edge in graph.edges(node))
                  if(edge.dst.level == INF)
                        edge.dst.level = LEVEL
                        Worklist.push(edge.dst)


src.level = 0
Iterate bfs(gr
      LEVEL++
}
```

```
pos = atomicAdd(Worklist.length, 1)
Worklist.items[pos] = edge.dst
```

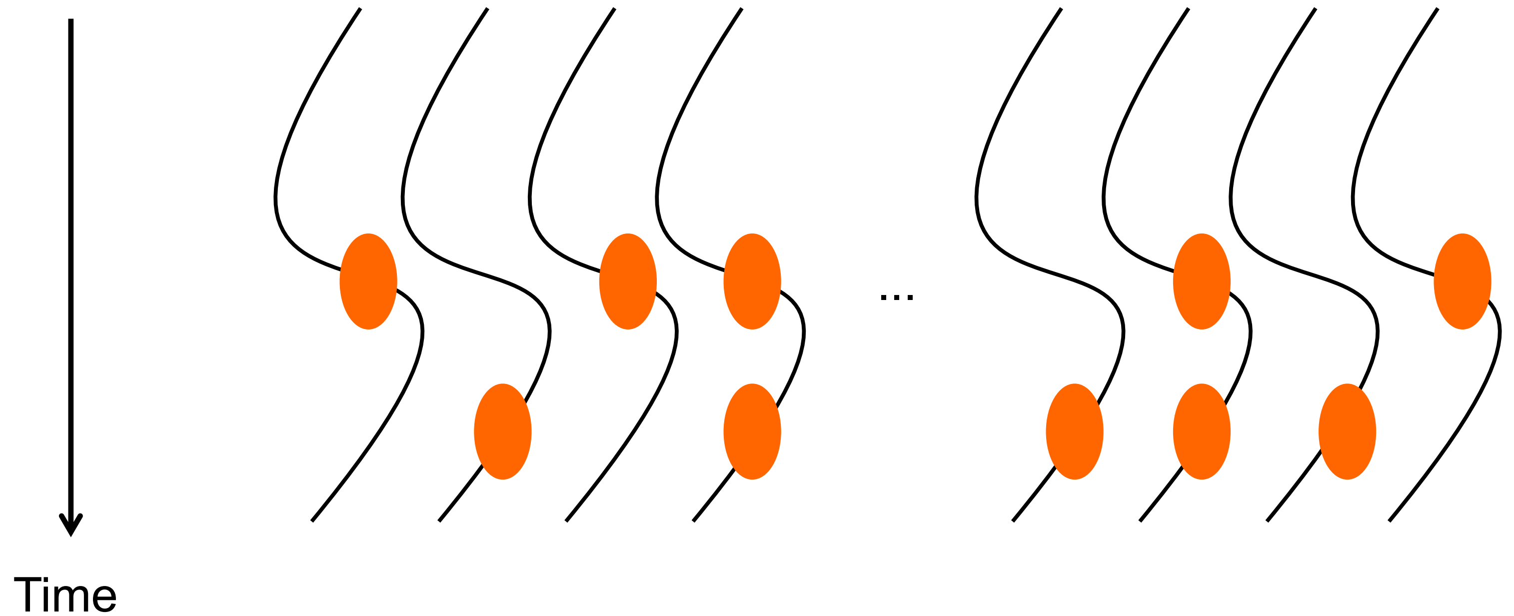
- Atomic Throughput on GPU: 1 per clock cycle
  - Roughly translated: 2.4 GB/s
  - Memory bandwidth: 288GB/s

Aggregating Atomics: Basic Idea



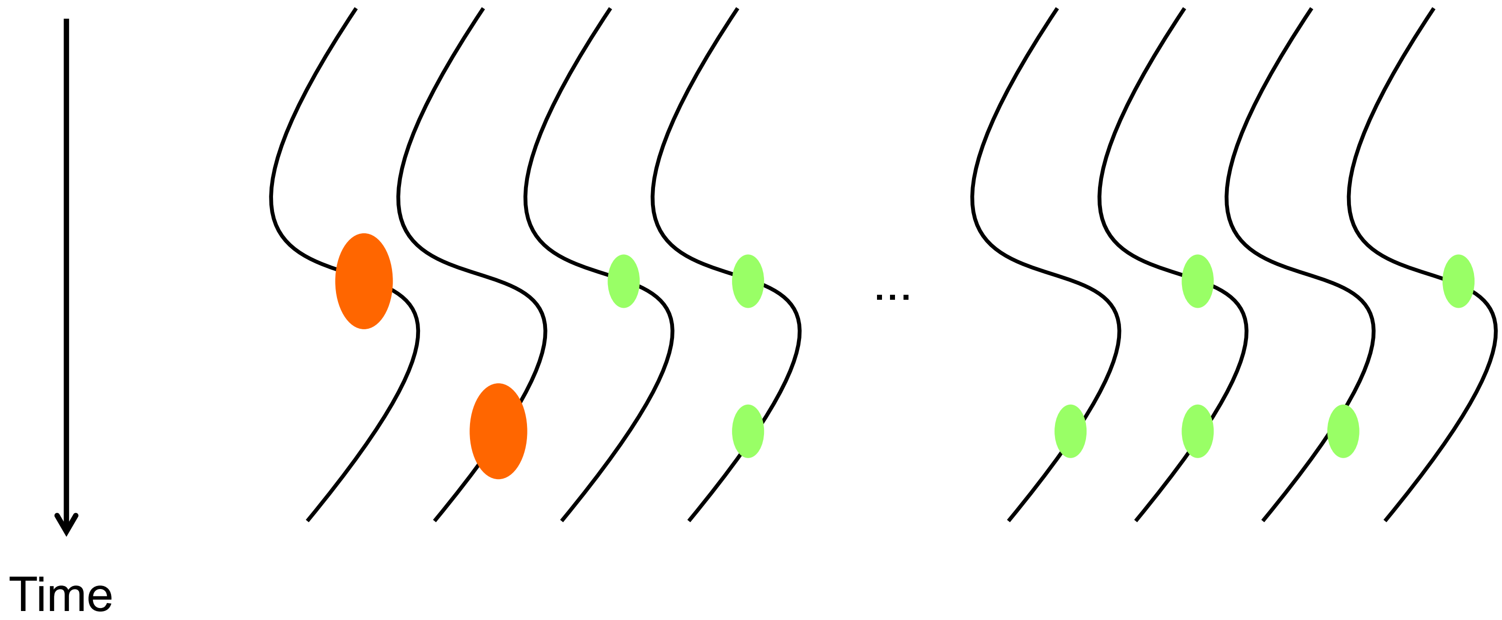
Challenge: Conditional Pushes

```
if(edge.dst.level == INF)
        Worklist.push(edge.dst)
```



Challenge: Conditional Pushes

```
if(edge.dst.level == INF)
        Worklist.push(edge.dst)
```



Must aggregate atomics *across* threads

Three Optimizations for Bottlenecks

1. Iteration Outlining
   - Improve GPU utilization for short kernels
2. Nested Parallelism
   - Improve load balance
3. Cooperative Conversion
   - Reduce atomics

Unoptimized BFS

- ~15 lines of CUDA
- 505ms on USA road network

Optimized BFS

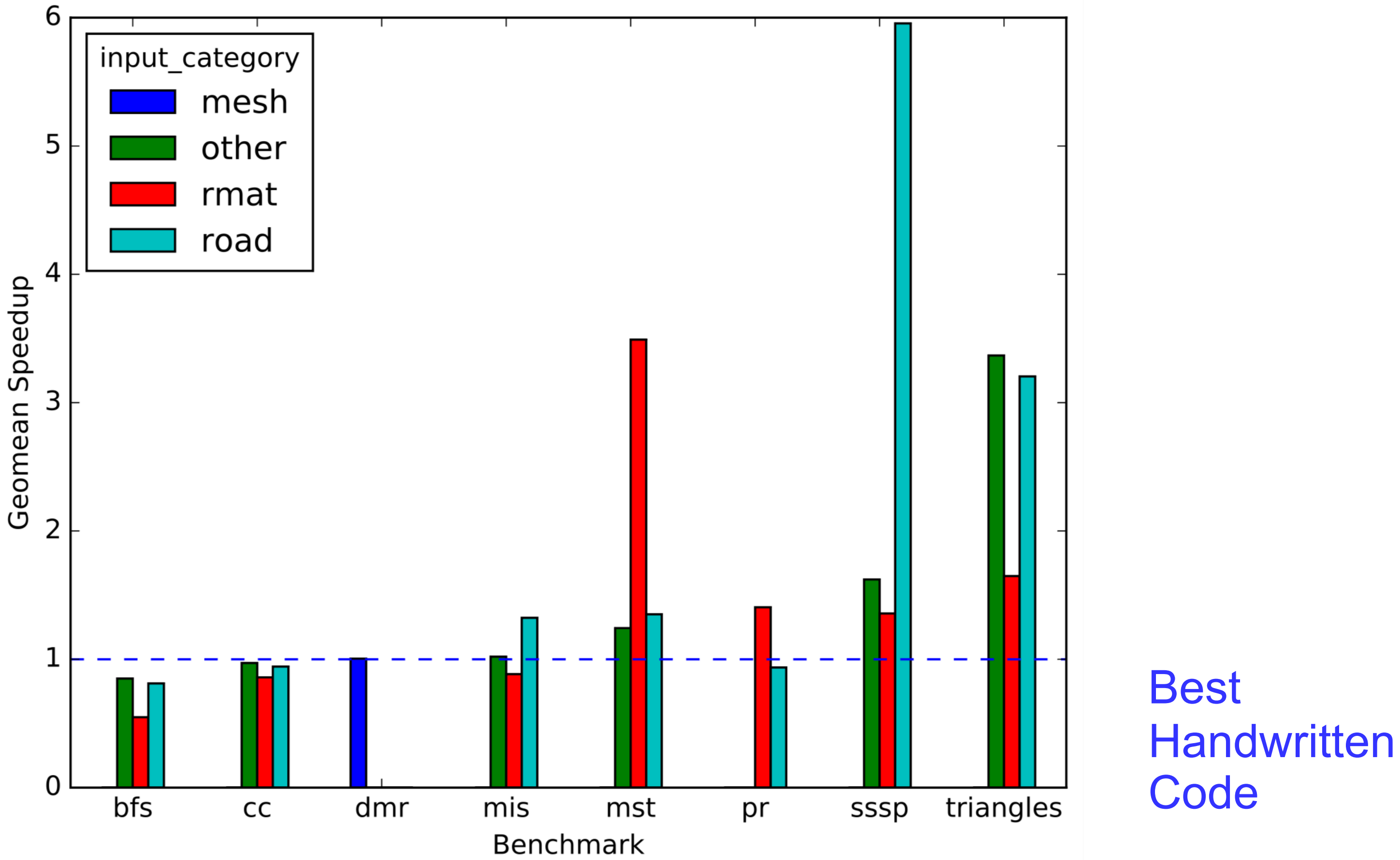
- ~200 lines of CUDA
- 120ms on the same graph

4.2x Performance Difference!

Evaluation

- Eight irregular algorithms
  - Breadth-First Search (BFS) [Merrill et al., 2012]
  - Connected Components (CC) [Soman et al., 2010]
  - Maximal Independent Set (MIS) [Che et al., 2013]
  - Minimum Spanning Tree (MST) [da Silva Sousa et al. 2015]
  - PageRank (PR) [Elsen and Vaidyanathan, 2014]
  - Single-Source Shortest Path (SSSP) [Davidson et al. 2014]
  - Triangle Counting (TRI) [Polak et al. 2015]
  - Delaunay Mesh Refinement (DMR) [Nasre et al., 2013]

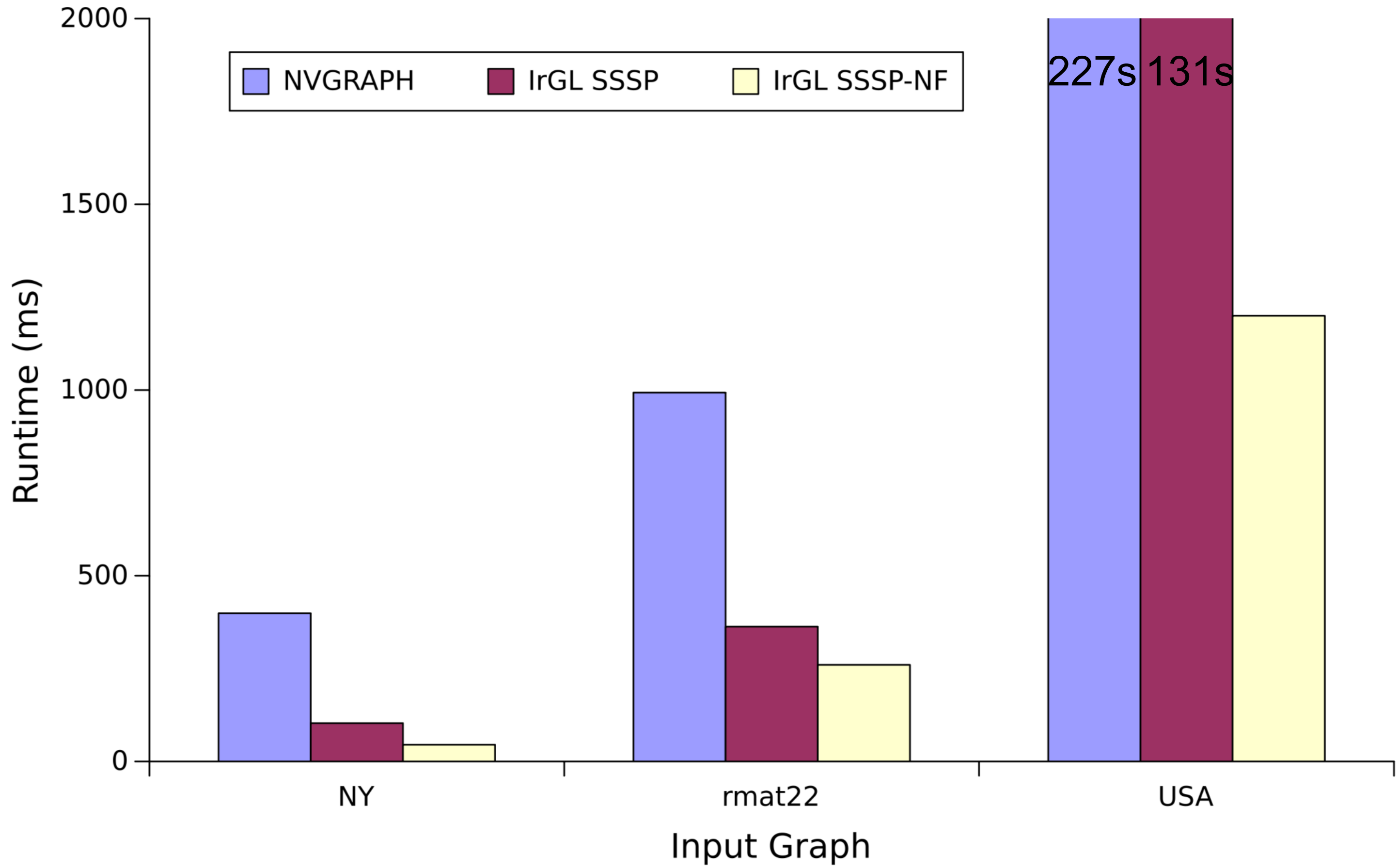
Overall Performance



Note: Each benchmark had a single set of optimizations applied to it

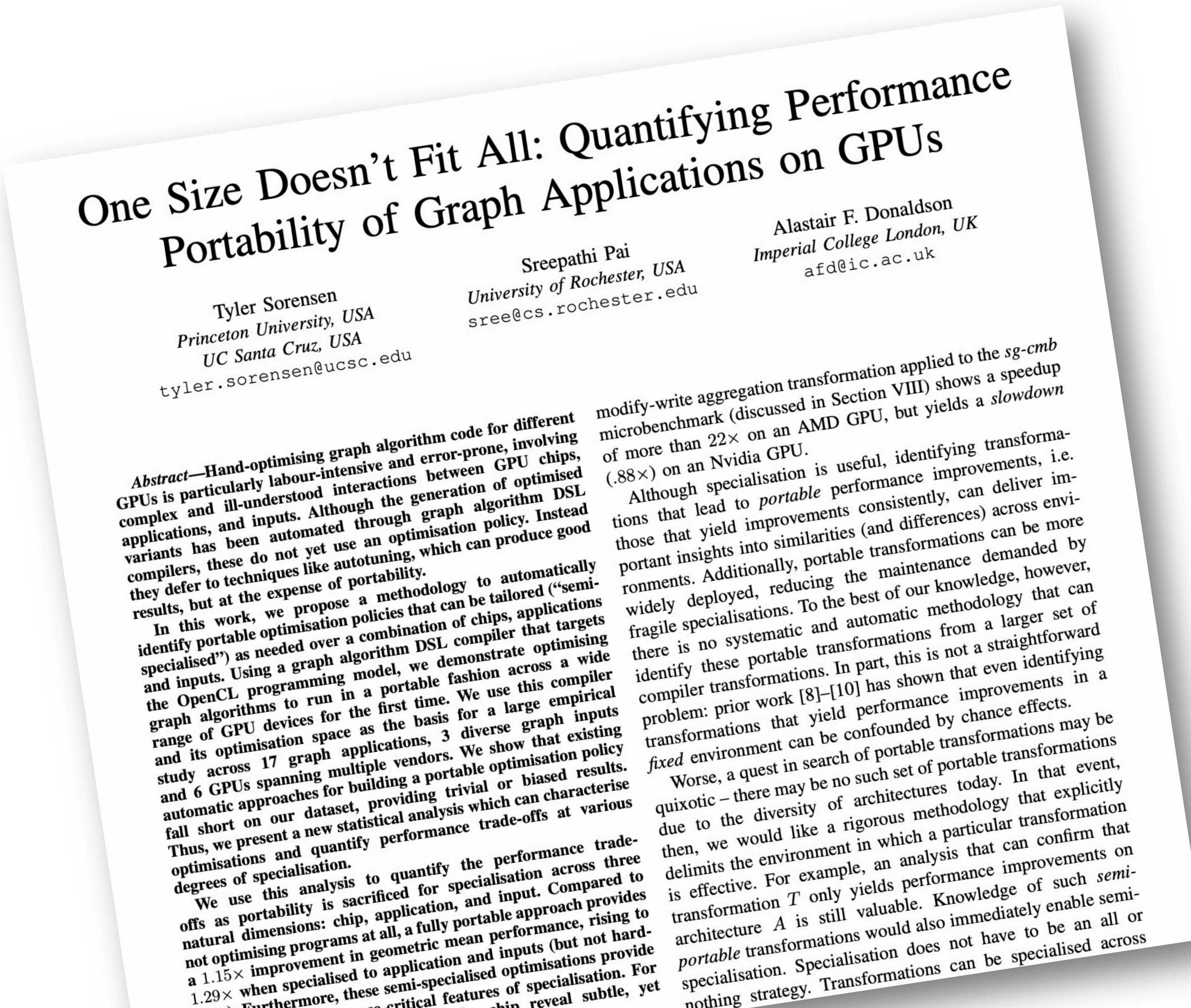
Pai and Pingali, OOPSLA 2016

Comparison to NVIDIA nvgraph SSSP



Pai, GTC 2017

Making IrGL portable



One Size Doesn't Fit All: Quantifying Performance Portability of Graph Applications on GPUs

Tyler Sorensen
Princeton University, USA
UC Santa Cruz, USA
tyler.sorensen@ucsc.edu

Sreepathi Pai
University of Rochester, USA
sree@cs.rochester.edu

Alastair F. Donaldson
Imperial College London, UK
afd@ic.ac.uk

***Abstract*—Hand-optimising graph algorithm code for different GPUs is particularly labour-intensive and error-prone, involving complex and ill-understood interactions between GPU chips, applications, and inputs. Although the generation of optimised variants has been automated through graph algorithm DSL compilers, these do not yet use an optimisation policy. Instead they defer to techniques like autotuning, which can produce good results, but at the expense of portability.**

In this work, we propose a methodology to automatically identify portable optimisation policies that can be tailored ("semi-specialised") as needed over a combination of chips, applications and inputs. Using a graph algorithm DSL compiler that targets the OpenCL programming model, we demonstrate optimising graph algorithms to run in a portable fashion across a wide range of GPU devices for the first time. We use this compiler and its optimisation space as the basis for a large empirical study across 17 graph applications, 3 diverse graph inputs and 6 GPUs spanning multiple vendors. We show that existing automatic approaches for building a portable optimisation policy fall short on our dataset, providing trivial or biased results. Thus, we present a new statistical analysis which can characterise optimisations and quantify performance trade-offs at various degrees of specialisation.

We use this analysis to quantify the performance trade-offs as portability is sacrificed for specialisation across three natural dimensions: chip, application, and input. Compared to not optimising programs at all, a fully portable approach provides a 1.15× improvement in geometric mean performance, rising to 1.29× when specialised to application and inputs (but not hard-... Furthermore, these semi-specialised optimisations provide ... critical features of specialisation. For ... chip reveal subtle, yet

modify-write aggregation transformation applied to the *sg-cmb* microbenchmark (discussed in Section VIII) shows a speedup of more than 22× on an AMD GPU, but yields a *slowdown* (.88×) on an Nvidia GPU.

Although specialisation is useful, identifying transformations that lead to *portable* performance improvements, i.e. those that yield improvements consistently, can deliver important insights into similarities (and differences) across environments. Additionally, portable transformations can be more widely deployed, reducing the maintenance demanded by fragile specialisations. To the best of our knowledge, however, there is no systematic and automatic methodology that can identify these portable transformations from a larger set of compiler transformations. In part, this is not a straightforward problem: prior work [8]–[10] has shown that even identifying transformations that yield performance improvements in a *fixed* environment can be confounded by chance effects.

Worse, a quest in search of portable transformations may be quixotic – there may be no such set of portable transformations due to the diversity of architectures today. In that event, then, we would like a rigorous methodology that explicitly delimits the environment in which a particular transformation is effective. For example, an analysis that can confirm that transformation T only yields performance improvements on architecture A is still valuable. Knowledge of such *semi-portable* transformations would also immediately enable semi-specialisation. Specialisation does not have to be an all or nothing strategy. Transformations can be specialised across

Believe it or not...

GPU != Nvidia

There are probably more Apple/Intel GPUs in this room than Nvidia GPUs

Headlines

We perform a massive empirical study (240 hours across 6 different GPUs)

Using a GPU graph application DSL and optimizing compiler, we find:

*Compiler optimizations can provide **speedups** of up to **16x** and a geomean across the domain of **1.5x***

*These optimizations can also provide **slowdowns** of up to **22x***

Headlines

Traditional ***performance portability*** fall short for graph applications on GPUs

- Previous approaches produce trivial or biased results

*All optimization combinations cause slowdowns **AND** speedups across the domain.*

*Magnitude-based approaches are **biased** towards more sensitive GPUs*

Headlines

Rank-based statistical procedures offer a new way of thinking about performance portability

Headlines

Rank-based statistical procedures offer a new way of thinking about performance portability

*Produces non-trivial performance portable optimization combination yielding a **max speedups** of **6x***

*Analysis can create **semi-specialized** optimization strategies, which yield greater speedups and **performance critical insights**.*

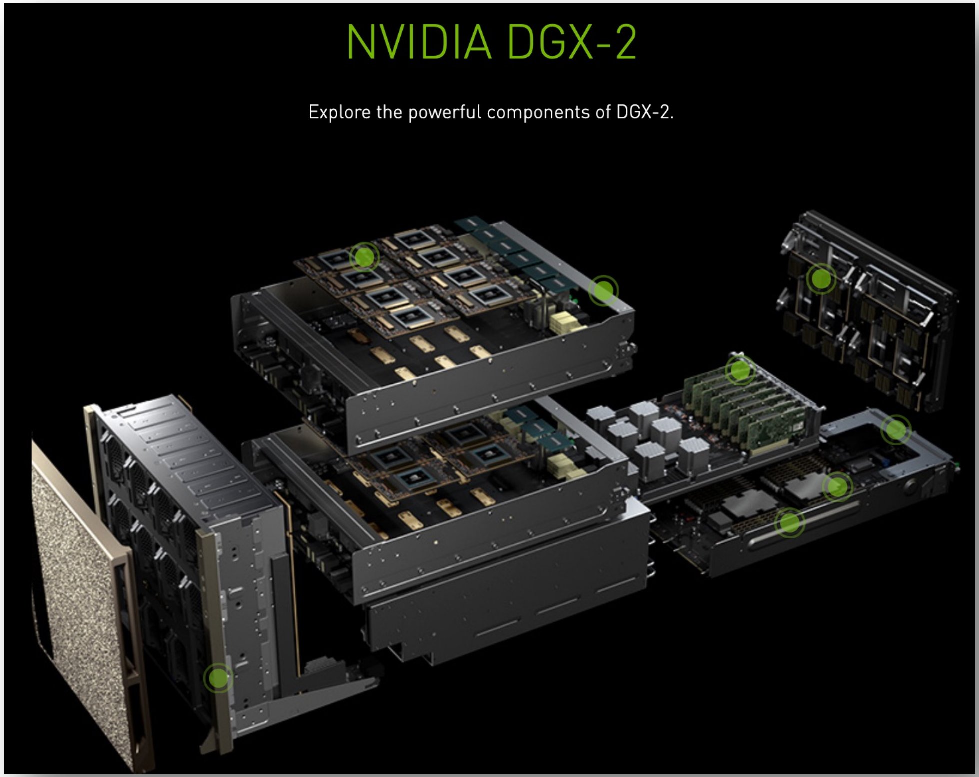
What is a GPU? (1999 Edition)

> The technical definition of a GPU is "a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second."

https://web.archive.org/web/20160408122443/http://www.nvidia.com/object/gpu.html

What is a GPU? (2019 Edition)

20 years later, Nvidia's homepage advertises GPUs without the ability to output graphics!



https://www.nvidia.com/en-us/data-center/dgx-2/

Trying to Define the Modern GPU

Still used for high-
end graphics

Trying to Define the Modern GPU

Still used for high-end graphics

Use in data centers for AI and scientific computing

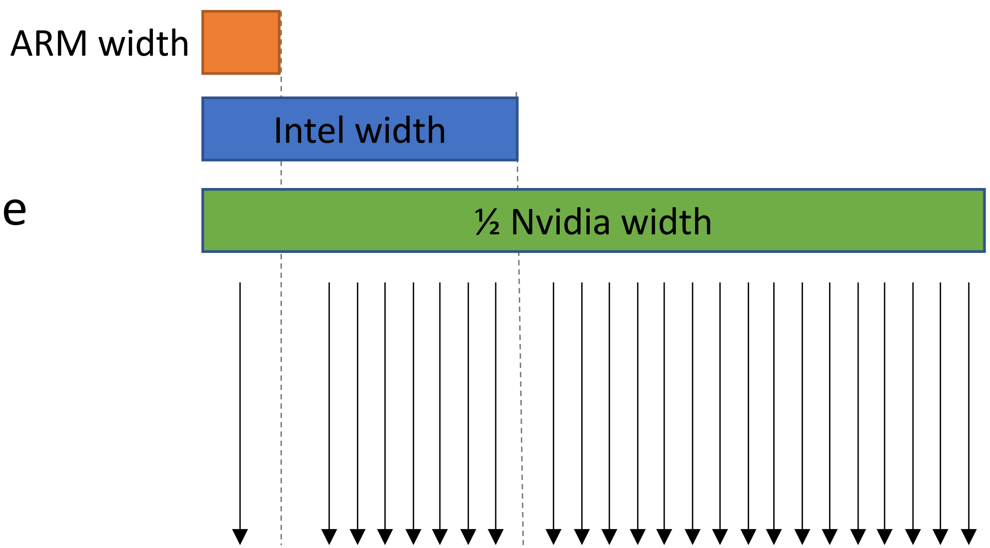
Trying to Define the Modern GPU

Still used for high-end graphics

Use in data centers for AI and scientific computing

Increasingly used in mobile devices

Trying to Define the Modern GPU



Programmable vector lanes?

- Nvidia GPUs have 32 threads per lane
- Intel GPUs have 8 threads per lane
- ARM GPUs have 1 thread per lane

High Bandwidth?

Highly parallel?

- Nvidia GPUs execute over 10K threads concurrently
- ARM GPUs execute 500 threads concurrently

Role of a compiler

As GPUs have diversified, it’s the compilers job to

- judiciously apply optimizations
  (apply transformations that cause speedups, not slowdowns)
- specialize when possible

This Work

Characterizing performance portability of Graph applications on GPUs

We Developed:

- A portable backend for a GPU graph application DSL and optimizing compiler

We Conducted:

- A large empirical study, collecting 240 hours of runtime data across 6 GPU

We Characterized:

- Performance portability in this domain using a rank-based statistical method

A GPU Graph DSL and Compiler

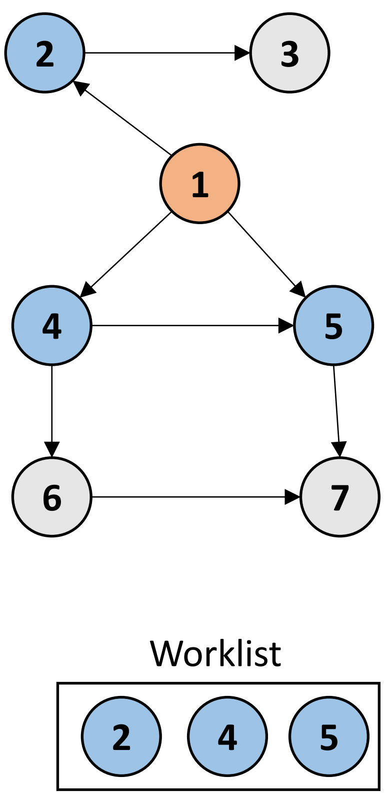
IrGL : Pai and Pingali, OOPSLA 2016

- Original work targets only Nvidia GPUs

First class support for nodes, edges, worklists

Optimizing compiler

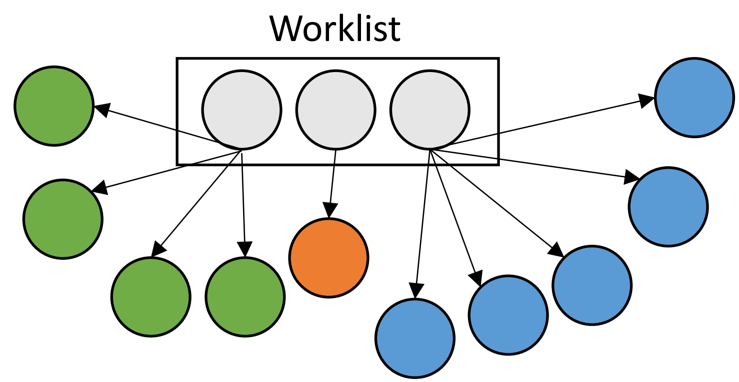
- Load balancing
- On-chip synchronization
- Atomic RMW coalescing

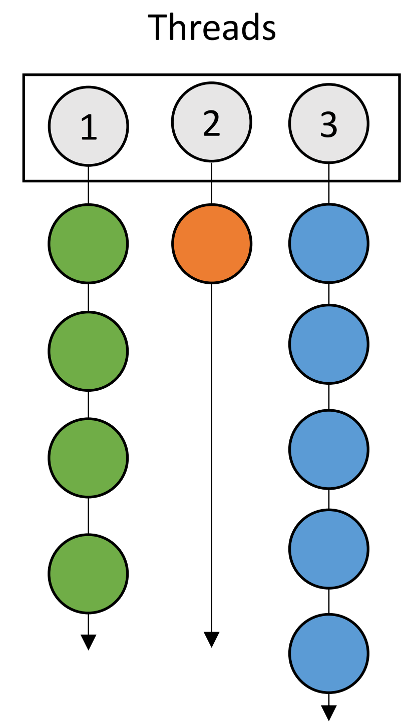


IrGL Optimizations

Load Balancing

Graphs have *irregular* parallelism leading to load imbalance



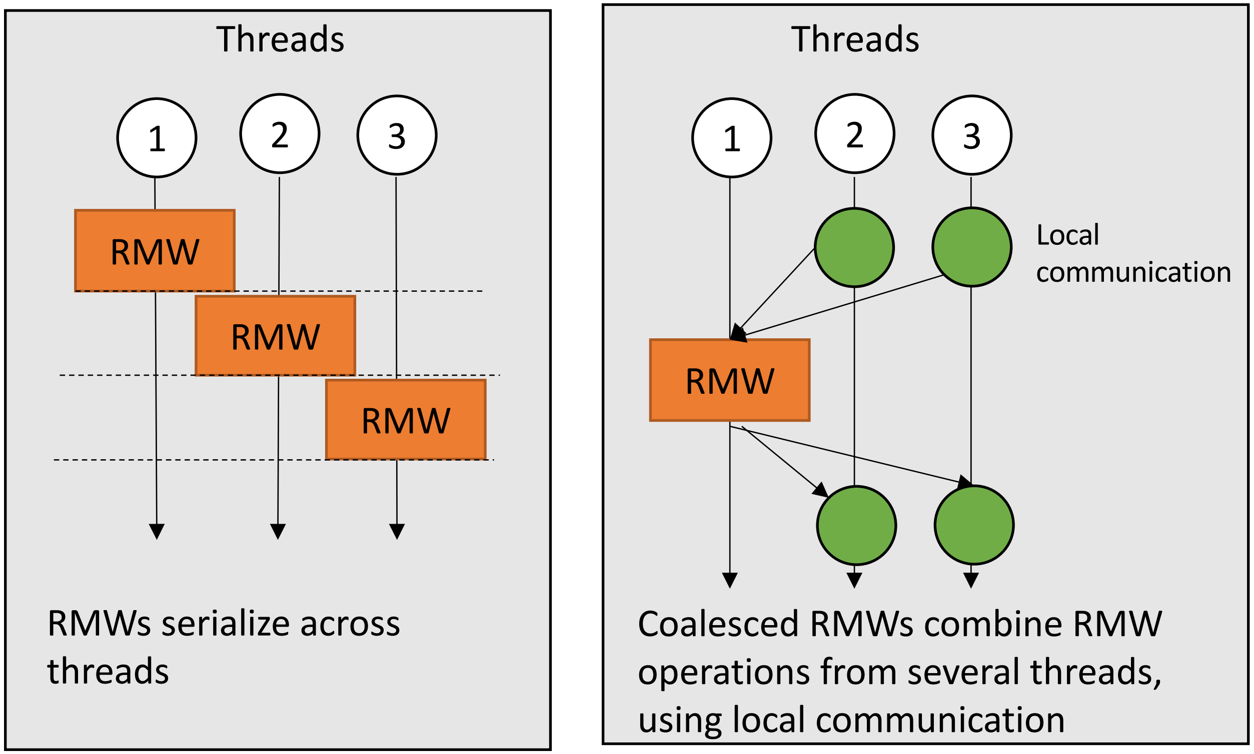


IrGL has 3 transformations to perform load balancing at 3 levels of the GPU hierarchy: Local, Subgroup, Workgroup

IrGL Optimizations

Atomic RMW Coalescing

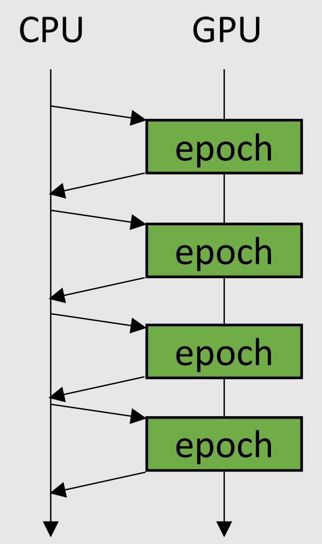
Graph applications require atomic RMWs to update the worklist for the next iteration



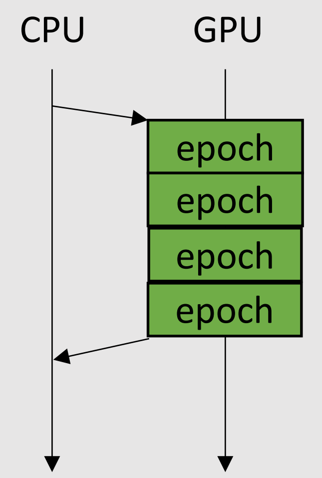
IrGL Optimizations

On-chip Synchronization

Many graph apps are iterative, requiring a global sync between iterations (epochs)



Traditionally GPU sync. involves CPU re-launch



Optimization to do on-chip sync. using experimental global barrier between epochs

Our Empirical Study

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
Nvidia-Quadro
Nvidia-1080
AMD-R9
Intel-Iris
Intel-HD5500
ARM-Mali T628

All combinations of above were run

Total runtime of **240 hours**

Over 10K individual runs

widest empirical study across GPUs that we are aware of!

Performance Portability

Which optimizations should be applied to provide best performance across the entire domain?

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space
(32 options)

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
Nvidia-Quadro
Nvidia-1080
AMD-R9
Intel-Iris
Intel-HD5500
ARM-Mali T628

Domain

Do No Harm

Only apply an optimization if it:

- Does not provide any slowdowns across the entire domain
- Provides at least one speedup

Easily to query from our data set, and we found…

Do No Harm

Only apply an optimization if it:

- Does not provide any slowdowns across the entire domain
- Provides at least one speedup

Easily to query from our data set, and we found…

NOTHING!!!

All optimizations provided at least one instance of a slowdown

Do the Least Harm

Relaxation of Do no Harm: Select the optimization combination that caused the fewest slowdowns.

Fewest slowdowns

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

36 Slowdowns
60 Speedups,
1.01x Geomean
2x max speedup

Do the Least Harm

Relaxation of Do no Harm: Select the optimization combination that caused the fewest slowdowns.

Fewest slowdowns

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

36 Slowdowns
60 Speedups,
1.01x Geomean
2x max speedup

From our exploration:

Compiler optimizations can provide ***speedups*** *of up to* ***16x*** *and a geomean across the domain of* ***1.5x***

Max Geomean

Select the optimization combination that provides the highest geomean across the domain

Highest Geomean

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

49 Slowdowns
66 Speedups,
1.18x Geomean

Max Geomean

Select the optimization combination that provides the highest geomean across the domain

Highest Geomean

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

49 Slowdowns
66 Speedups,
1.18x Geomean

GPUs	# Speedups	# Slowdowns
Nvidia-Quadro	10	21
Nvidia-1080	00	16
AMD-R9	12	3
Intel-Iris	10	2
Intel-HD5500	14	2
ARM-Mali T628	20	5

Max Geomean

Select the optimization combination that provides the highest geomean across the domain

Highest Geomean

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

49 Slowdowns
66 Speedups,
1.18x Geomean

GPUs	# Speedups	# Slowdowns
Nvidia-Quadro	10	21
Nvidia-1080	00	16
AMD-R9	12	3
Intel-Iris	10	2
Intel-HD5500	14	2
ARM-Mali T628	20	5

Our Approach: Rank-based

For a single chip,app,input combination, just compare confidence intervals

Optimizations
- LB – Local
- LB - Subgroup
- LB - Workgroup
- OC - Sync
- RMW-Cls

Optimization Space

Applications
- BFS
- SSSP
- PR
- CC
- MIS
- MST
- TRI

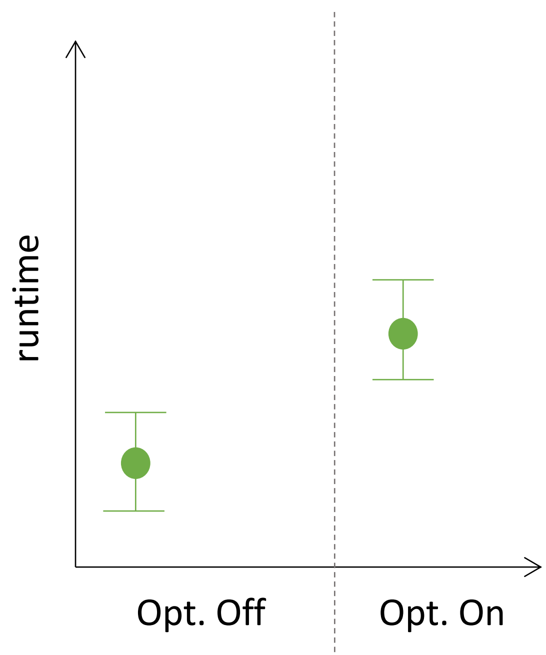
Inputs
- Uniform
- RMAT
- NY-Road

GPUs
- Nvidia-Quadro
- Nvidia-1080
- AMD-R9
- Intel-Iris
- Intel-HD5500
- ARM-Mali T628

Domain

Our Approach: Rank-based

For a single chip,app,input combination,
just compare confidence intervals



Optimizations
- LB – Local
- LB - Subgroup
- LB - Workgroup
- OC - Sync
- RMW-Cls

Optimization Space

Applications
- BFS
- SSSP
- PR
- CC
- MIS
- MST
- TRI

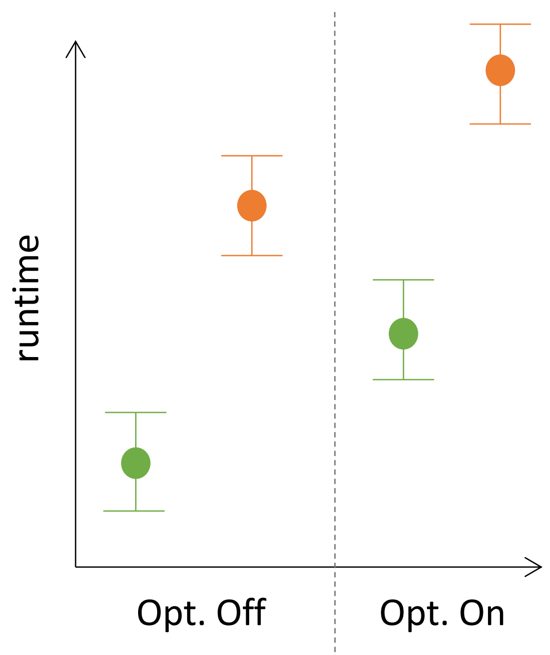
Inputs
- Uniform
- RMAT
- NY-Road

GPUs
- Nvidia-Quadro
- Nvidia-1080
- AMD-R9
- Intel-Iris
- Intel-HD5500
- ARM-Mali T628

Domain

Our Approach: Rank-based

Things become trickier when more chips are added



Optimizations
LB – Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

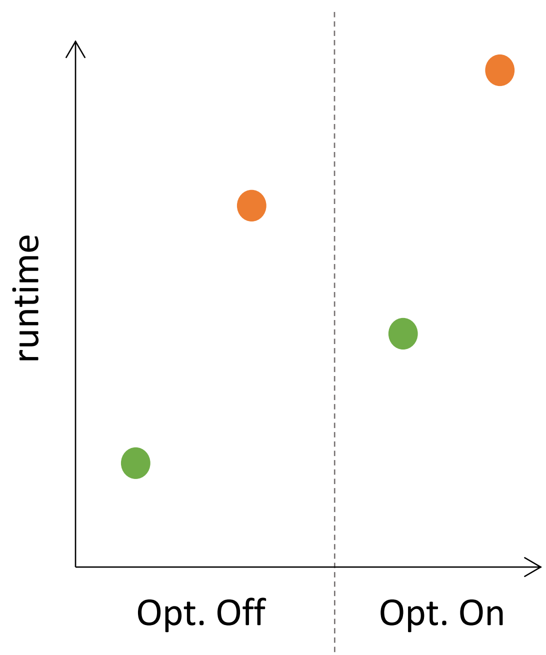
Inputs
Uniform
RMAT
NY-Road

GPUs
Nvidia-Quadro
Nvidia-1080
AMD-R9
Intel-Iris
Intel-HD5500
ARM-Mali T628

Domain

Our Approach: Rank-based

First, only consider points whose confidence
intervals don't overlap



Optimizations
LB – Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

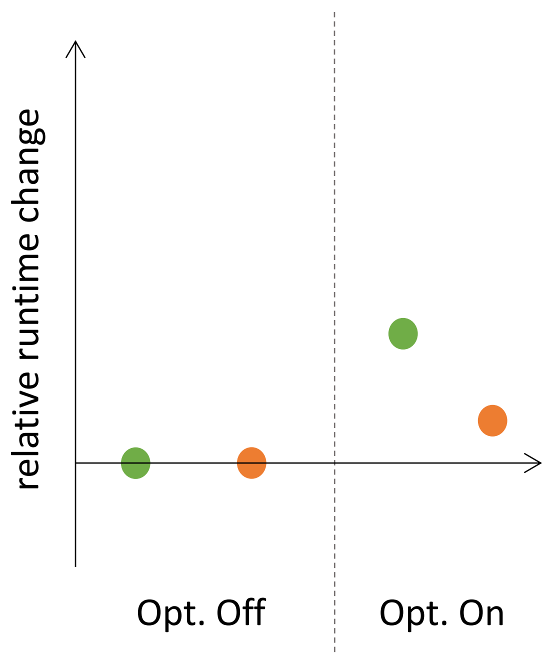
Inputs
Uniform
RMAT
NY-Road

GPUs
Nvidia-Quadro
Nvidia-1080
AMD-R9
Intel-Iris
Intel-HD5500
ARM-Mali T628

Domain

Our Approach: Rank-based

Normalize with respect to Opt. Off



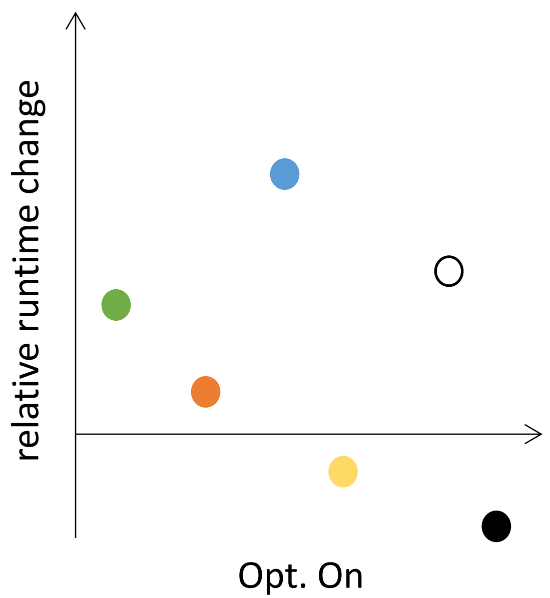
Optimizations	Applications	Inputs	GPUs
LB – Local	BFS	Uniform	Nvidia-Quadro
LB - Subgroup	SSSP	RMAT	Nvidia-1080
LB - Workgroup	PR	NY-Road	AMD-R9
OC - Sync	CC		Intel-Iris
RMW-Cls	MIS		Intel-HD5500
	MST		ARM-Mali T628
	TRI		

Optimization Space

Domain

Our Approach: Rank-based

Only consider relative *Opt. On* points, we can show more now visually



Optimizations
LB – Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space

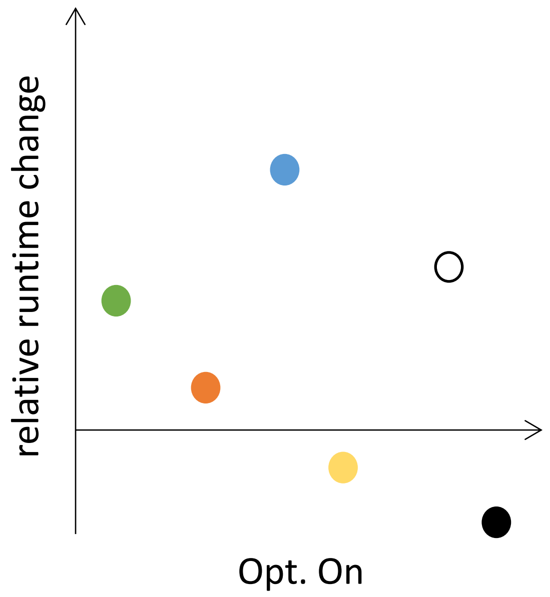
Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
Nvidia-Quadro
Nvidia-1080
AMD-R9
Intel-Iris
Intel-HD5500
ARM-Mali T628

Domain

Our Approach: Rank-based



We now use the ***Mann-Whitney U test*** to determine if points are ***stochastically more likely to be above*** the horizontal line.

The test is ***non-parametric***: it assumes nothing about the distribution.

Rank-based Results

Compared to fewest slowdowns, more slowdowns, also more speedups. Higher Geomean and higher max

Fewest slowdowns

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

36 Slowdowns
60 Speedups,
1.01x Geomean
2x max speedup

Rank-based

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

60 Slowdowns
66 Speedups,
1.15x Geomean
6x max speedup

Rank-based Results

Compared to highest geomean: No more bias against Nvidia GPUs

Highest Geomean

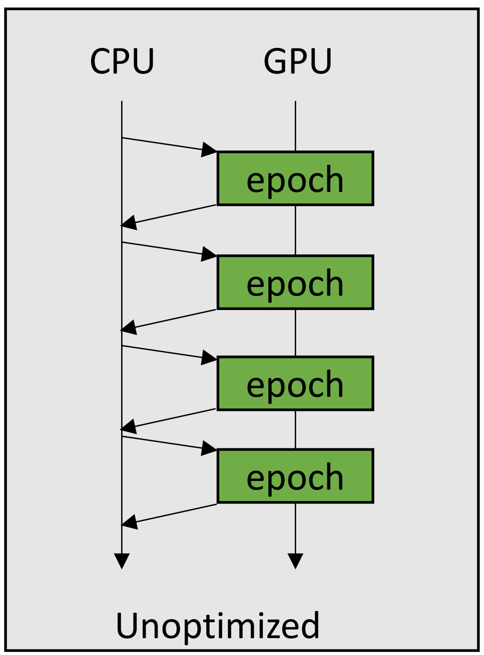
GPUs	# Speedups	# Slowdowns
Nvidia-Quadro	10	21
Nvidia-1080	00	16
AMD-R9	12	3
Intel-Iris	10	2
Intel-HD5500	14	2
ARM-Mali T628	20	5

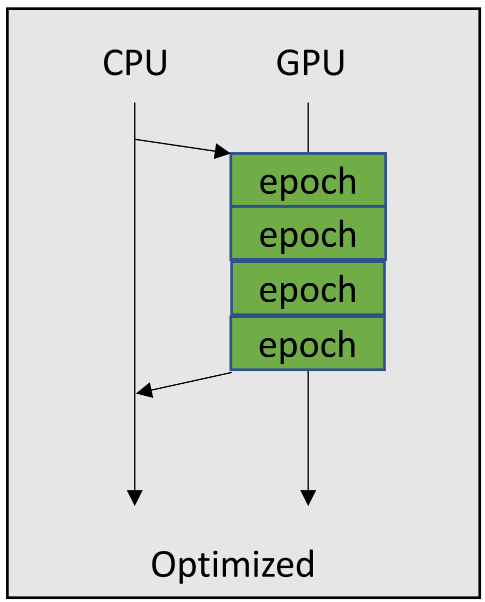
Rank-based

GPUs	# Speedups	# Slowdowns
Nvidia-Quadro	22	13
Nvidia-1080	13	07
AMD-R9	17	4
Intel-Iris	10	10
Intel-HD5500	21	12
ARM-Mali T628	20	04

Impact on GPU Programming Languages

Working with Khronos group to better specify a progress model that allows on-chip synchronization (OC-Sync)





Rank-based Global Optimizations

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-CIs

60 Slowdowns
66 Speedups,
1.15x Geomean
6x max speedup

KHRONOS GROUP

GPU Compiler Summary

GPUs and ***graph applications*** are important ***emerging domain***.

- We perform a massive empirical study (240 hours across 6 different GPUs)

Traditional ***performance portability*** fall short in this domain.

Rank-based statistical procedures offer a new way of thinking about performance portability

Semi-specialization per GPU

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space
(32 options)

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
Nvidia-Quadro
Nvidia-1080
AMD-R9
Intel-Iris
Intel-HD5500
ARM-Mali T628

Domain

Semi-specialization per GPU

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space
(32 options)

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
[redacted]
ARM-Mali T628

Domain

Semi-specialization per GPU

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space
(32 options)

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
[redacted]
[redacted]
[redacted]
[redacted]
Intel-HD5500
[redacted]

Domain

Semi-specialization per GPU

Optimizations
LB - Local
LB - Subgroup
LB - Workgroup
OC - Sync
RMW-Cls

Optimization Space
(32 options)

Applications
BFS
SSSP
PR
CC
MIS
MST
TRI

Inputs
Uniform
RMAT
NY-Road

GPUs
[redacted]
Intel-Iris
[redacted]

Domain

Semi-specialization

Provides 6 different optimization strategies, one per chip:

GPUs	LB-Local	LB-Subgroup	LB-Workgroup	OC - Sync	RMW-Cls
Nvidia-Quadro	.86	.68	.22	.47	.07
Nvidia-1080	.86	.78	.32	.22	.19
AMD-R9	.90	.74	.18	.65	.70
Intel-Iris	.58	.63	.09	.73	.67
Intel-HD5500	.54	.56	.12	.63	.41
ARM-Mali T628	.47	.76	.11	.71	.12

Semi-specialization

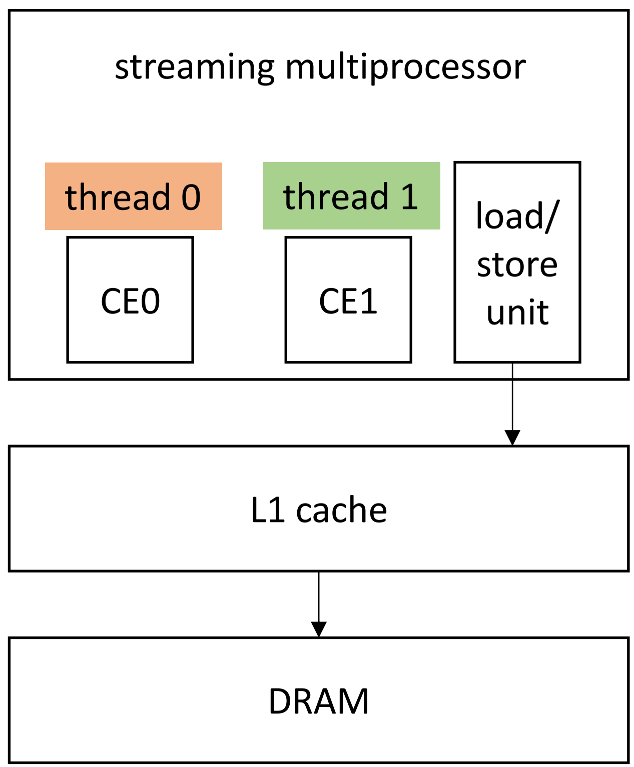
Provides 6 different optimization strategies, one per chip:

GPUs	LB-Local	LB-Subgroup	LB-Workgroup	OC - Sync	RMW-Cls
Nvidia-Quadro	.86	.68	.22	.47	.07
Nvidia-1080	.86	.78	.32	.22	.19
AMD-R9	.90	.74	.18	.65	.70
Intel-Iris	.58	.63	.09	.73	.67
Intel-HD5500	.54	.56	.12	.63	.41
ARM-Mali T628	.47	.76	.11	.71	.12

What about streaming multiprocessors (GPUs)?

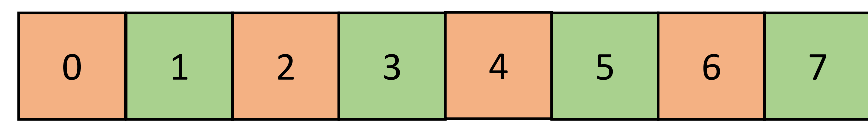
one streaming multiprocessor contains many small Compute Elements (CE)

CEs Can load adjacent memory locations simultaneously



ITER 0:

What about a striped pattern?



Semi-specialization

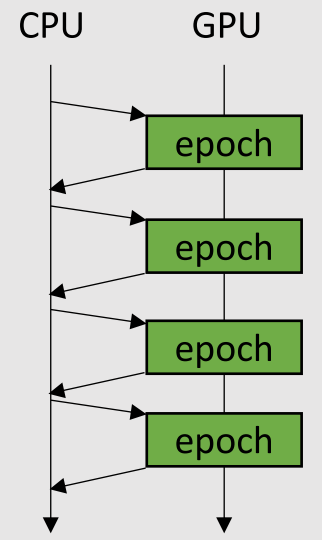
Provides 6 different optimization strategies, one per chip:

GPUs	LB-Local	LB-Subgroup	LB-Workgroup	OC - Sync	RMW-Cls
Nvidia-Quadro	.86	.68	.22	.47	.07
Nvidia-1080	.86	.78	.32	.22	.19
AMD-R9	.90	.74	.18	.65	.70
Intel-Iris	.58	.63	.09	.73	.67
Intel-HD5500	.54	.56	.12	.63	.41
ARM-Mali T628	.47	.76	.11	.71	.12

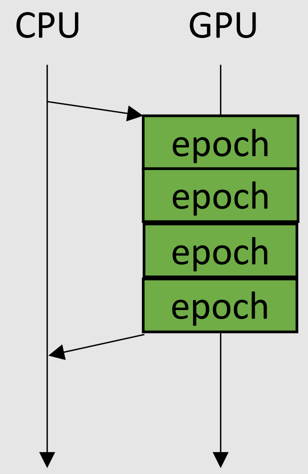
IrGL Optimizations

On-chip Synchronization

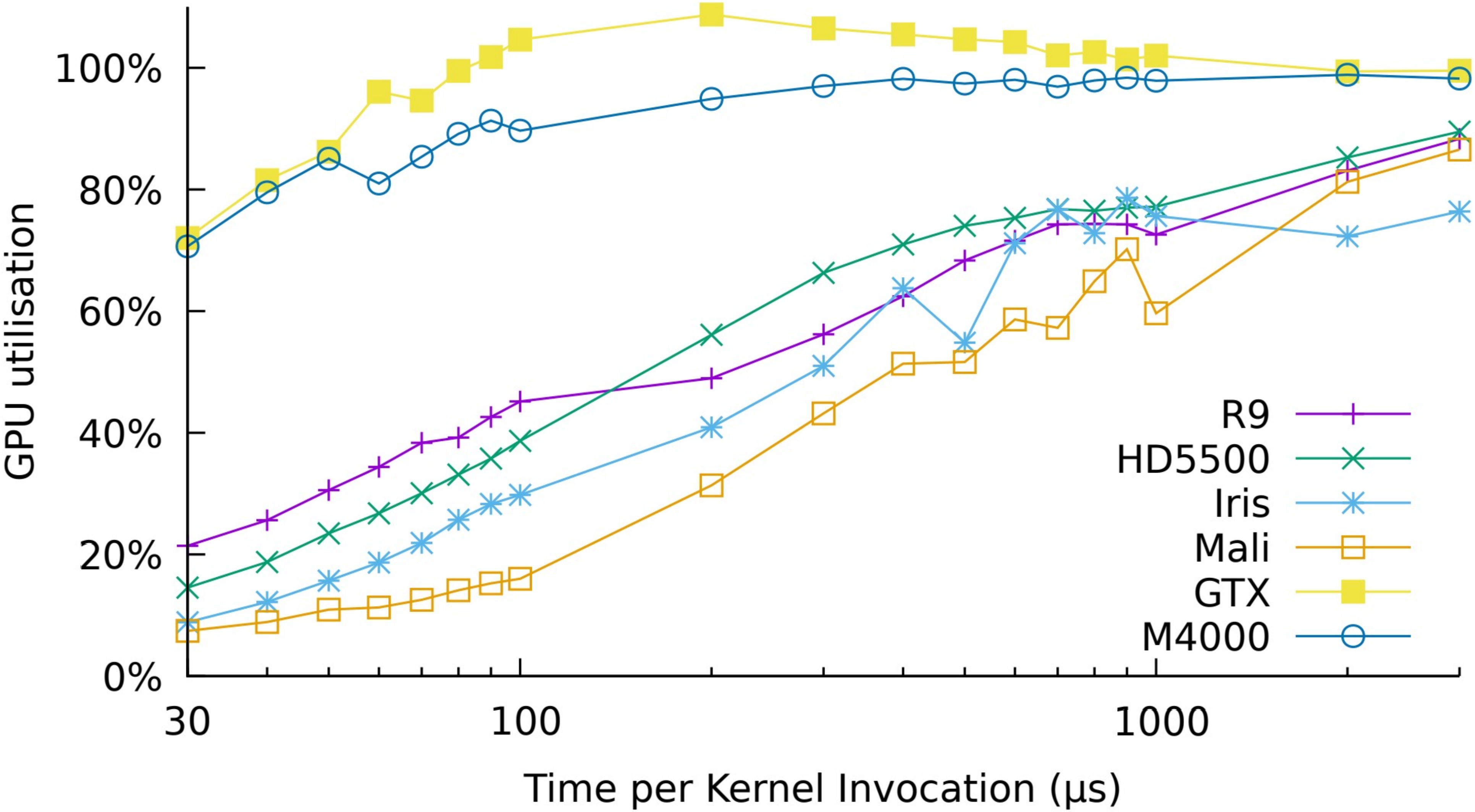
Many graph apps are iterative, requiring a global sync between iterations (epochs)



Traditionally GPU sync. involves CPU re-launch



Optimization to do on-chip sync. using experimental global barrier between epochs



GPU utilisation
100%
80%
60%
40%
20%
0%
30
100
1000
Time per Kernel Invocation (µs)
R9
HD5500
Iris
Mali
GTX
M4000

Next lecture

Optimization impact in general purpose languages!