Share-a-GPU

Providing Simple and Effective Time-Sharing on GPUs

Shaleen Garg, Kishore Kothapalli, Suresh Purini

Computer Systems Group

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Needs ?

Expensive Hardware

- Users Contend for Same Resource
- Exorbitant Wait times
- Native GPUs follow leftover policy

Cloud Service Providers

- Usual to Overbook their hardware
- Rely on user non-consumption
- Fails when faces high traffic

E-Waste

- Hardware getting old rapidly
- Simulates a bigger GPU using Software
- Not explored much

Energy Impact

- No need for more hardware
- Less energy consumption
- Lesser heat generation

Literature Survey

- Wu et al. **FLEP** (ASPLOS ‘17) [Software]
  - ➢ MicroKernels for weighted Round-Robin Scheduling
  - ➢ Assumes that GPU Memory accommodates all programs
  - ➢ Experiments show at most three loads running concurrently

- Xu et al. **Warped-Slicer** (ISCA ‘16) [Simulator]
  - ➢ Pairs programs such that they use all the resources on SMs
  - ➢ Needs prior profiling to find suitable coexisting Kernels

- Anguilera et al. **Fair Share** (ICCD ‘14) [Simulator]
  - ➢ Proposes allocation policy based on Fairness to all programs
  - ➢ Uses Spatial multitasking instead of cooperative multitasking

Overview

- Introduction
- Our Technique
- Experiments
- Conclusion

Overview

- Introduction
- Our Technique
- Experiments
- Conclusion

“

The best ideas are often also the simplest

Some Definitions

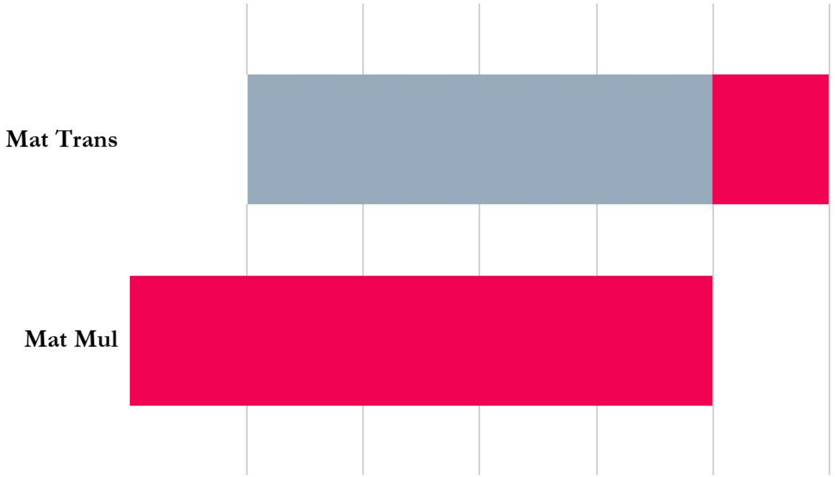
- **Compute Kernel** - Upto 3-D grid of GPU computations
- **Grid** - Consists of many Blocks (CTAs)
- **Compute Block** - Independent Unit of computations
- **Index** - Identity of each block (3 tuple of integers)
- **Timeslice** - Time of exclusive access to the GPU(s)

Stages of Preemption

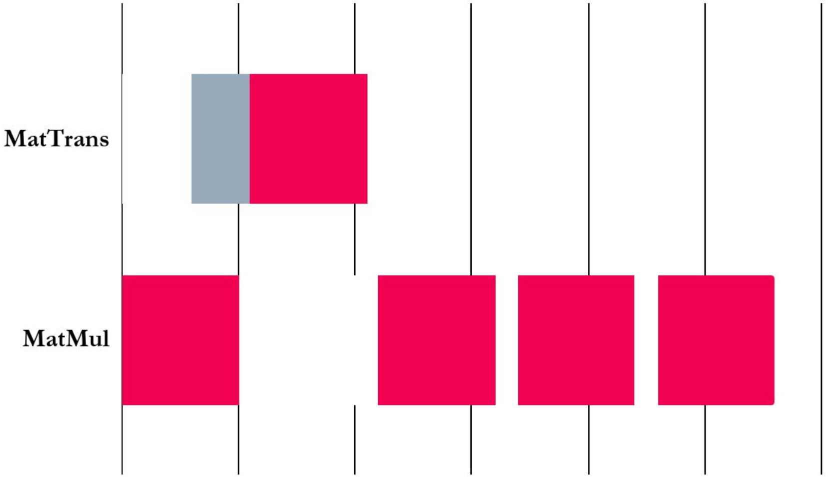
Pause	Save State	Resume
Pause the Kernel after its time slice for context switching	Snapshot of the current program state	Start from where we stopped last

The Changeover

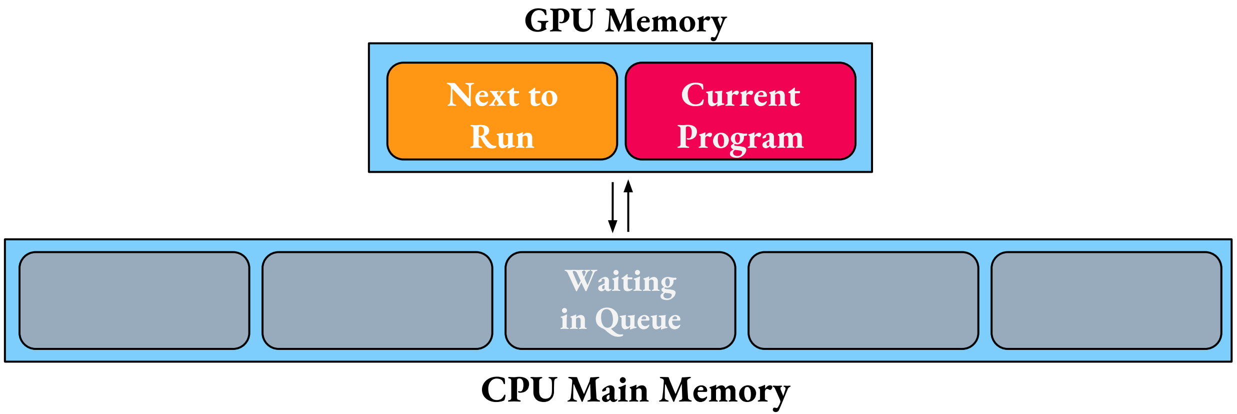
Native GPU (FIFO)



Scheduler (Round-Robin)

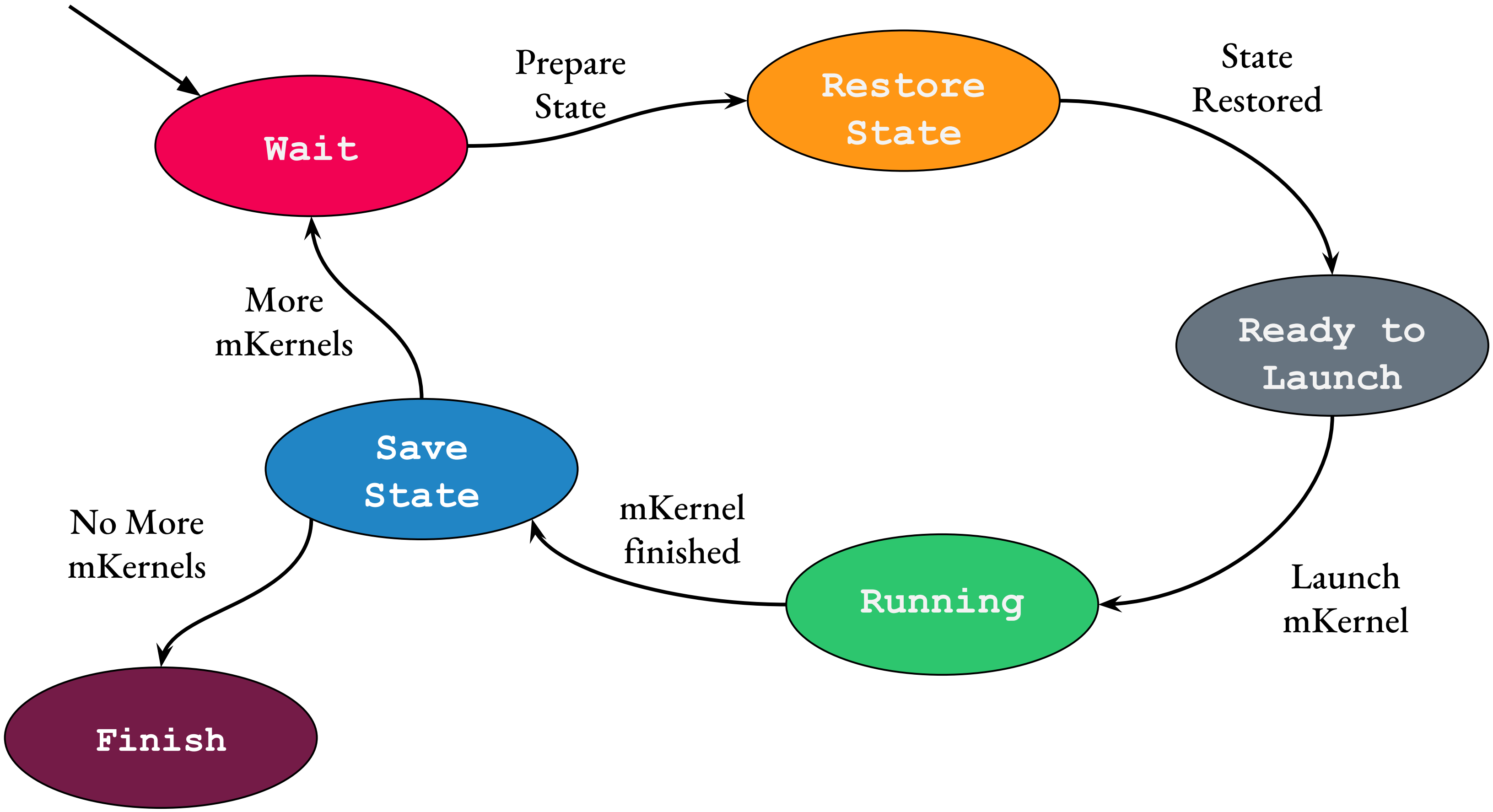


Memory Management

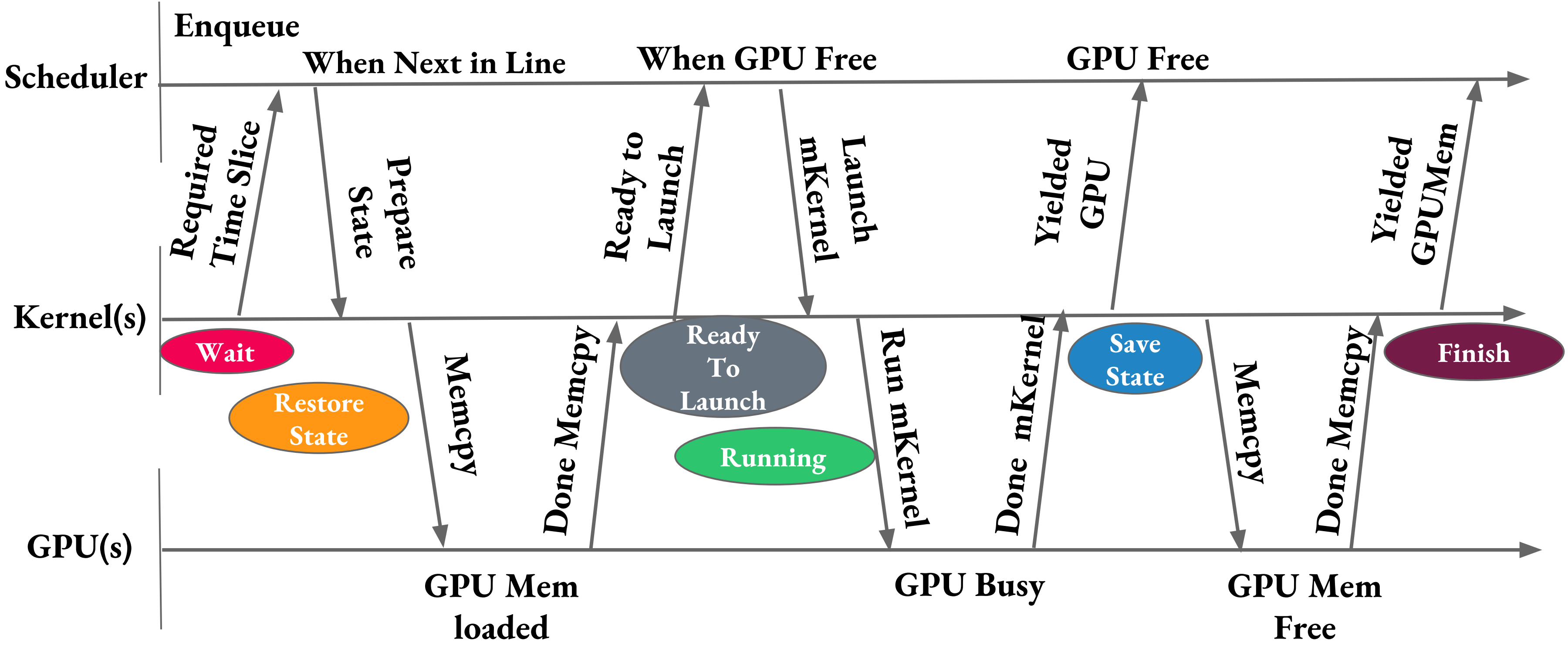


- Realistic Approach
- Double Buffering
- Memory transfers (PCI) concurrent to compute
- Guarantees ½ GPU memory to each kernel

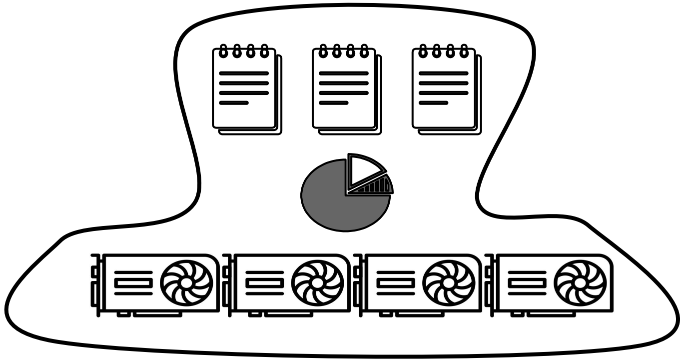
Program States

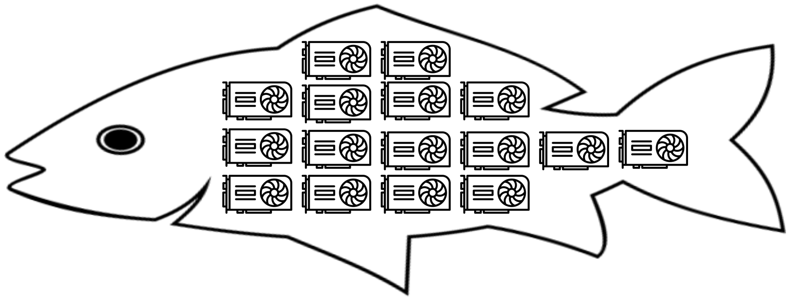


Scheduler - The Middleman



Multi-GPU Support



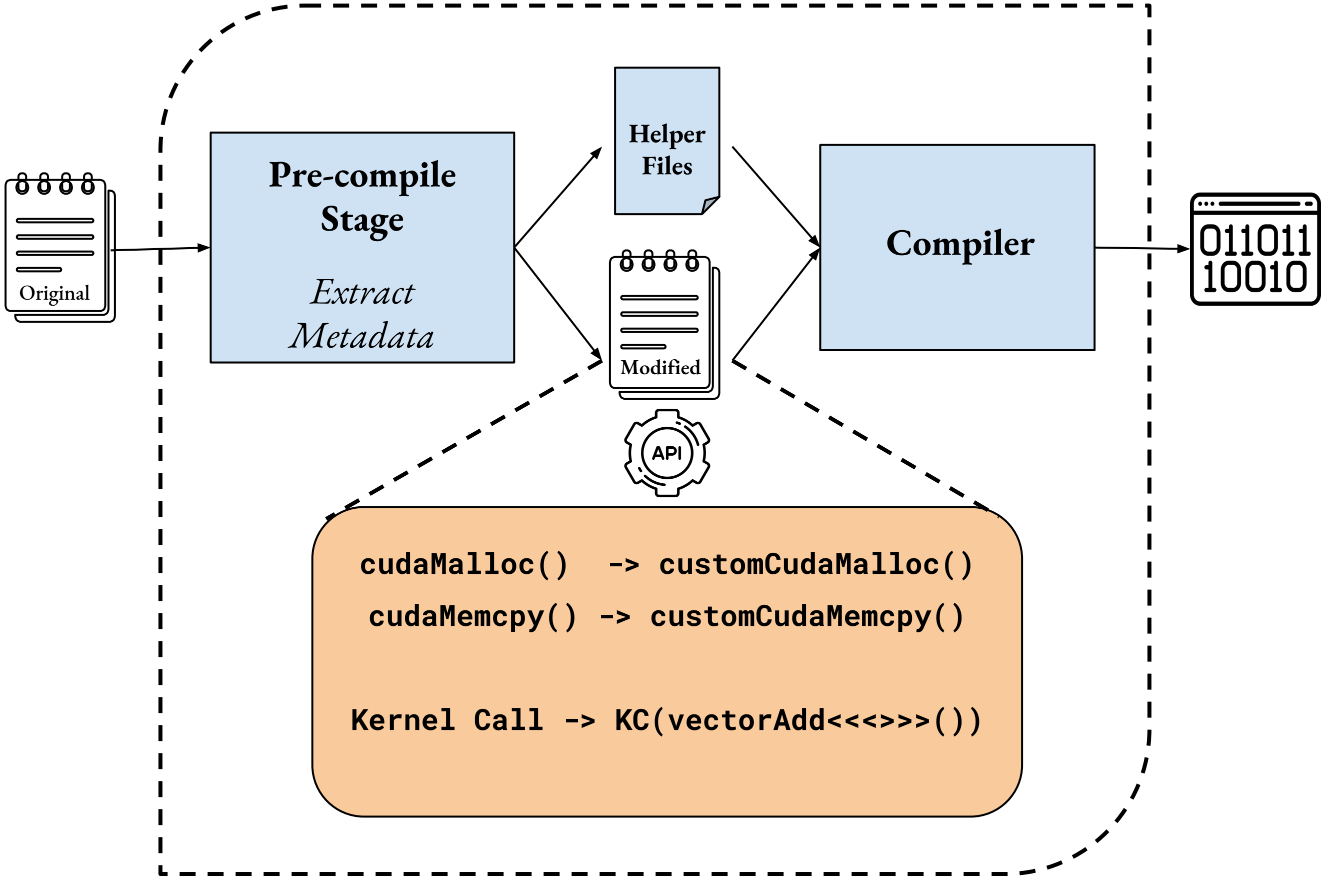


- Multi-GPU installations common
- Simulates an aggregate GPU from many small GPUs
- Intra-Kernel Parallelism
- Workload divided automatically

“

The best systems do not bother their users

User Ease



- Assisted Cooperation
- Modified API
- Usual Execution
- Transparent Working

Overview

- Introduction
- Our Technique
- Experiments
- Conclusion

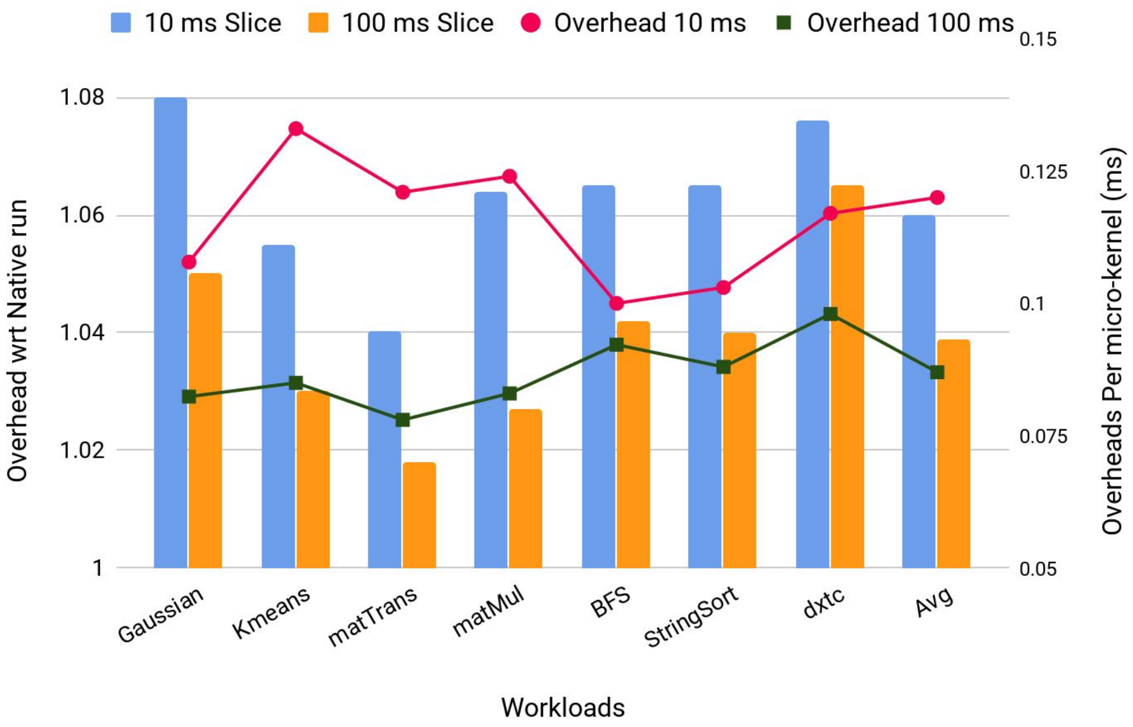
Overview

- Introduction
- Our Technique
- Experiments
- Conclusion

Experimental Programs

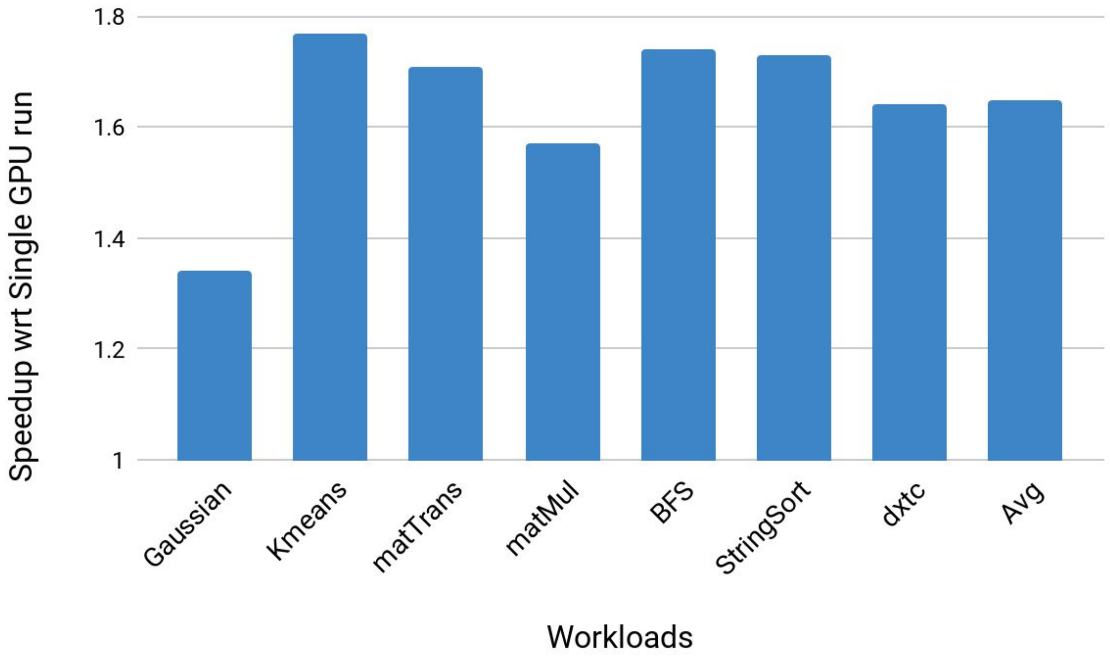
	Characteristics	Input	Memory footprint (MiB)	Native Runtime (sec)
BFS	Memory	Graph	1321	0.4
Gaussian	Compute	Matrix	883	327.45
Kmeans	Memory	Point Coord.	1006	0.07
matMul	Compute	Matrices	5424	54.19
matTrans	Memory	Matrix	5433	0.17
dxtc	Compute	Image	2280	20.78
StringSort	Memory	List of Strings	3072	0.203
TOTAL			**19420**	

Single Program on One GPU



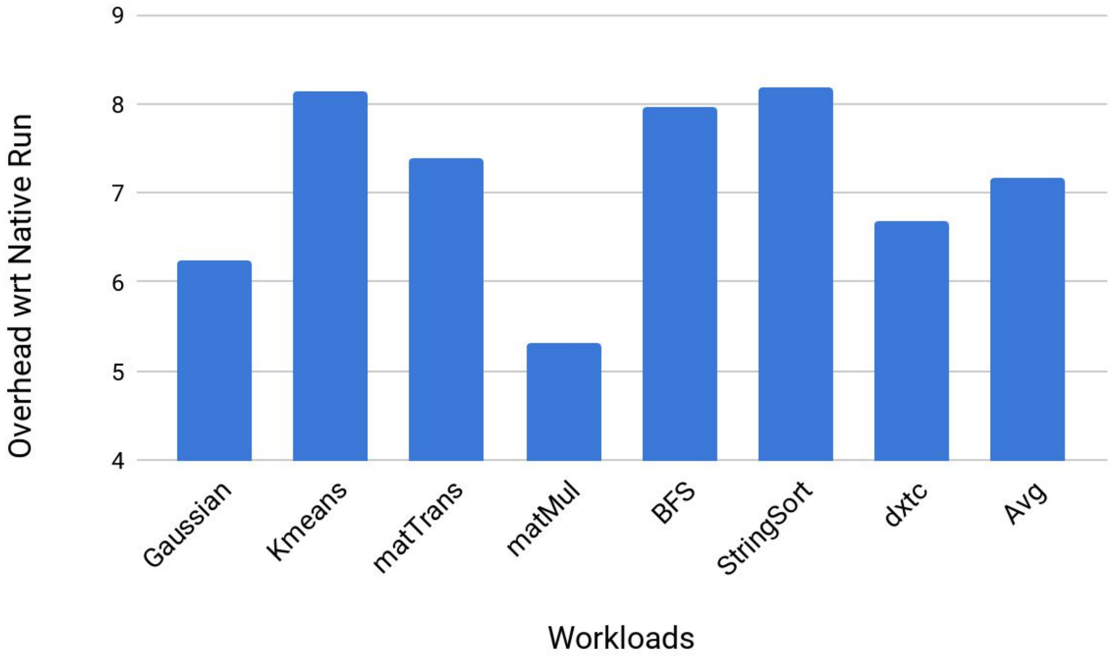
- Average slowdown is 4 - 5 %
- Overheads decrease with increase in timeslice
- Overheads ~0.1 ms

Single Program on Two GPUs



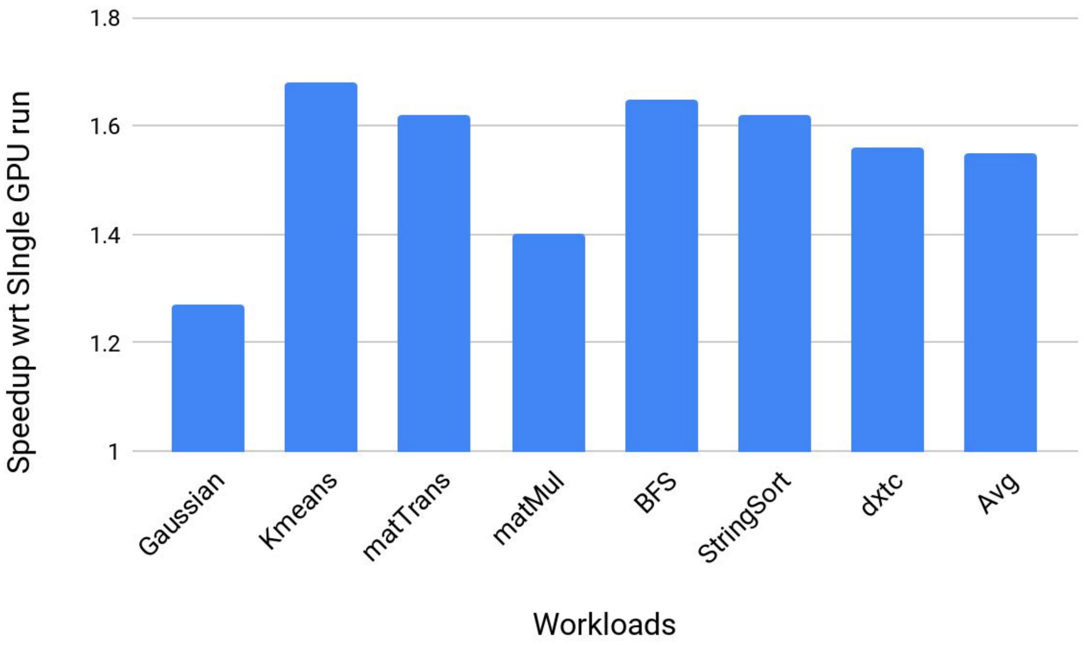
- Average speedup **~1.65 x**
- No additional user effort

All Programs on One GPU



- The real test of Fire
- First to show this !
- Overheads as anticipated

All Programs on Two GPUs



- Another test of Fire
- First to show this !
- Automatic speedup of **~1.5 x**

Future Work

- ▷ Extend to a cluster setting
- ▷ Support heterogeneous multi-GPUs
- ▷ Extend it to commonly used libraries

Conclusion

- ✓ Capable of things never done before
- ✓ Provides a deployable solution
- ✓ Transparent & Easy to use
- ✓ Flexible scheduling algorithms

Thanks!

Any Questions?

You can contact us at:

shaleen.garg@research.iiit.ac.in

kkishore@iiit.ac.in

suresh.purini@iiit.ac.in