Dynamic Front-End Sharing In Graphics Processing Units

Xiaoyao Liang

Shanghai Jiao Tong University

Presented by: Xiaoyao Liang

MPSoC 2016

Nara, Japan

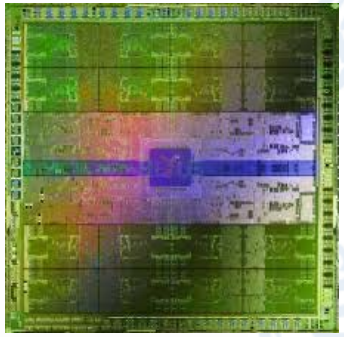
Agenda

- Motivation
- Introduction
- Related work
- Front-end sharing architecture
- Experimental methodology
- Results and analysis
- Conclusion

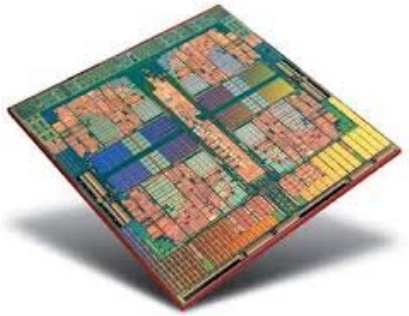
Motivation

Graphics Processing Units (GPUs) are now widely used in general purpose computing, can we reduce their power ?

- A many-core GPU consumes several times power of a multi-core CPU
- Front-end power is a major portion of a GPU [S. Hong et al.]



GPU : Nvidia GTX480, 40nm node,
15-16 streaming multiprocessors,
250W TDP

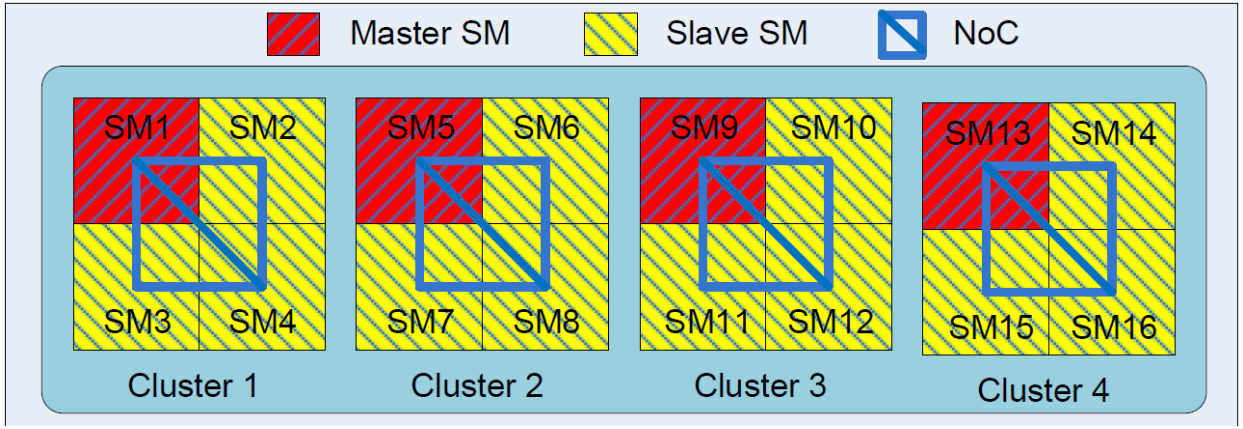


CPU : Intel Core i5-750s, 45nm node,
Quad-core, 72W TDP

Introduction

We propose a novel front-end sharing architecture to share front-end units among several adjacent streaming multiprocessors (SMs) opportunistically

- A GPU is split into several sharing clusters
- There is a master SM in each cluster; the front-end unit of the master SM is active working for all SMs in the cluster
- The front-end units of the slave SMs are turned off to save power



Example: Splitting a 16-SM GPU into four sharing clusters

Related Work

Combine several small CPU cores into a big, powerful core

- Core Fusion [Ipek et al.]
- Core Federation [Tarjan et al.]
- Composable Lightweight Processors [C. Kim et al.]

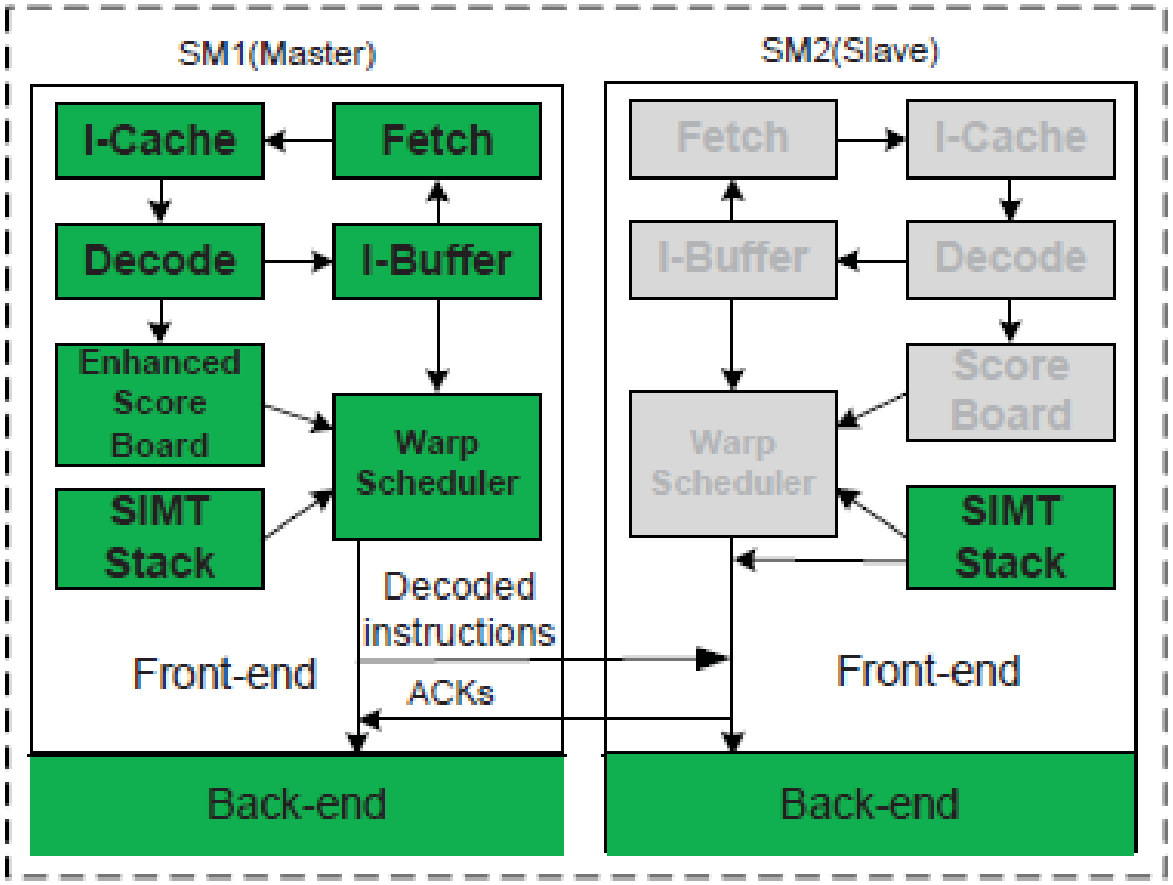
Save power in GPU components

- Adding a RF cache to reduce the number of accesses to the conventional power-hungry RF [Gebhart et al.]
- Using eDRAM as the replacement of SRAM for RFs in GPUs [Jing et al.]
- Integrating STT-RAM into GPU as RFs [Goswami et al.]
- Adding a filter cache to eliminate 30%-100% of instruction cache requests [Lashgar et al.]

Our work is the first to arrange several SMs to work in the lock-step manner in GPUs

Front-end sharing architecture(1/5)

Every S (eg. 2 or 4) adjacent SMs are grouped to work in the lock-step manner



A two-SM front-end sharing cluster

In the master SM:

- All front-end components are active
- Exploits an enhanced scoreboard to track the memory operation for all SMs in the cluster
- Sends decoded inst. to slaves

In slave SMs:

- Only the SIMT stack is active.
- Receive decoded inst. from the master

Front-end sharing architecture(2/5)

Grouping

- Splitting a GPU of multiple SMs into clusters
- Happening at every kernel launch
- The SM with the least index in each cluster become the master

Ungrouping

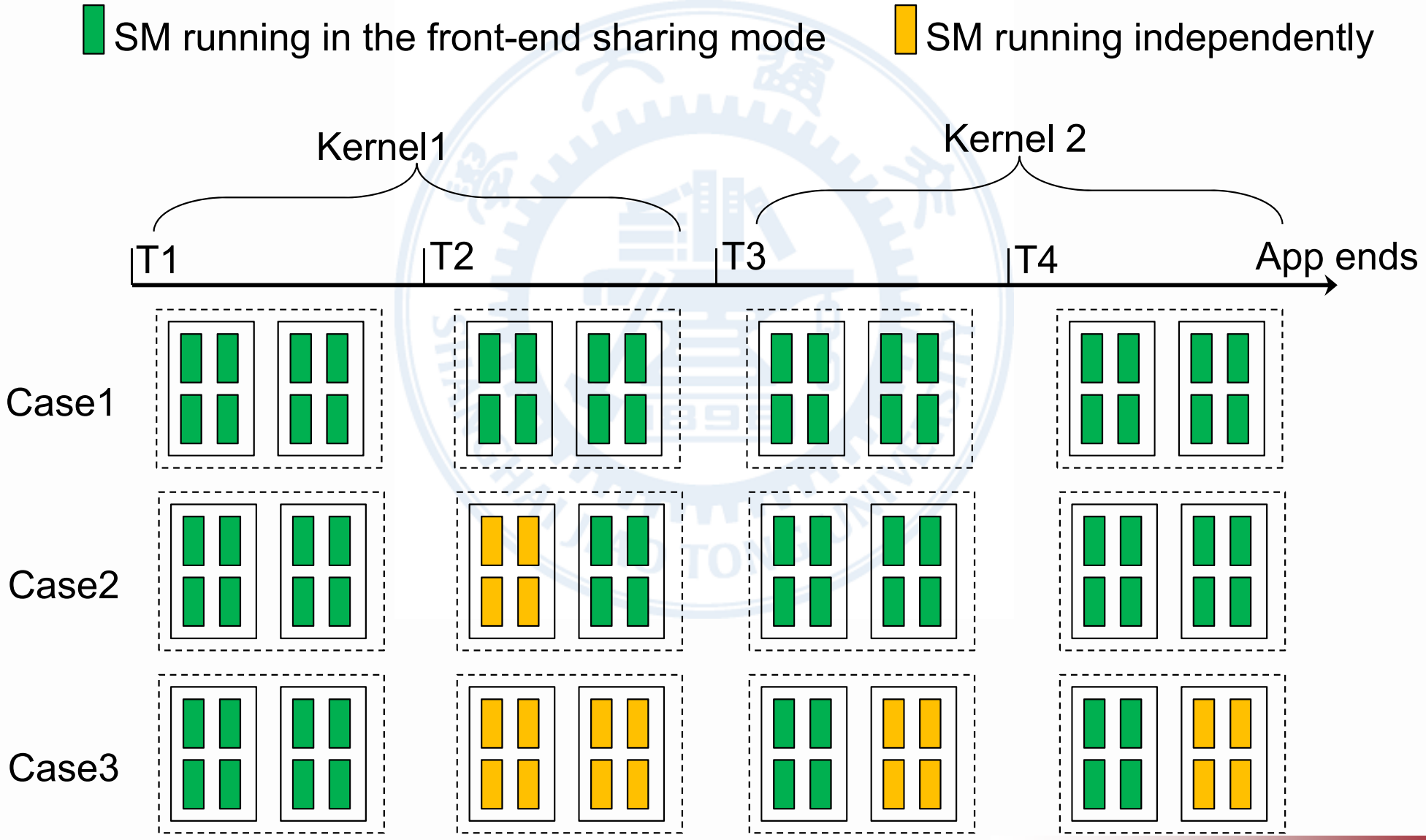
- When SMs in a cluster taking different instructions (called SM divergence), ungroup this cluster and then SMs work independently
- Happening at most once in each kernel (Clusters once ungrouped will never regroup until the end of the kernel).

Regrouping

- Normally, a GPU application consists of multiple kernels, each implementing certain function. At the beginning of the new kernel, SMs will have the opportunity to be grouped again even if they are just ungrouped in the last kernel.

Front-end sharing architecture(3/5)

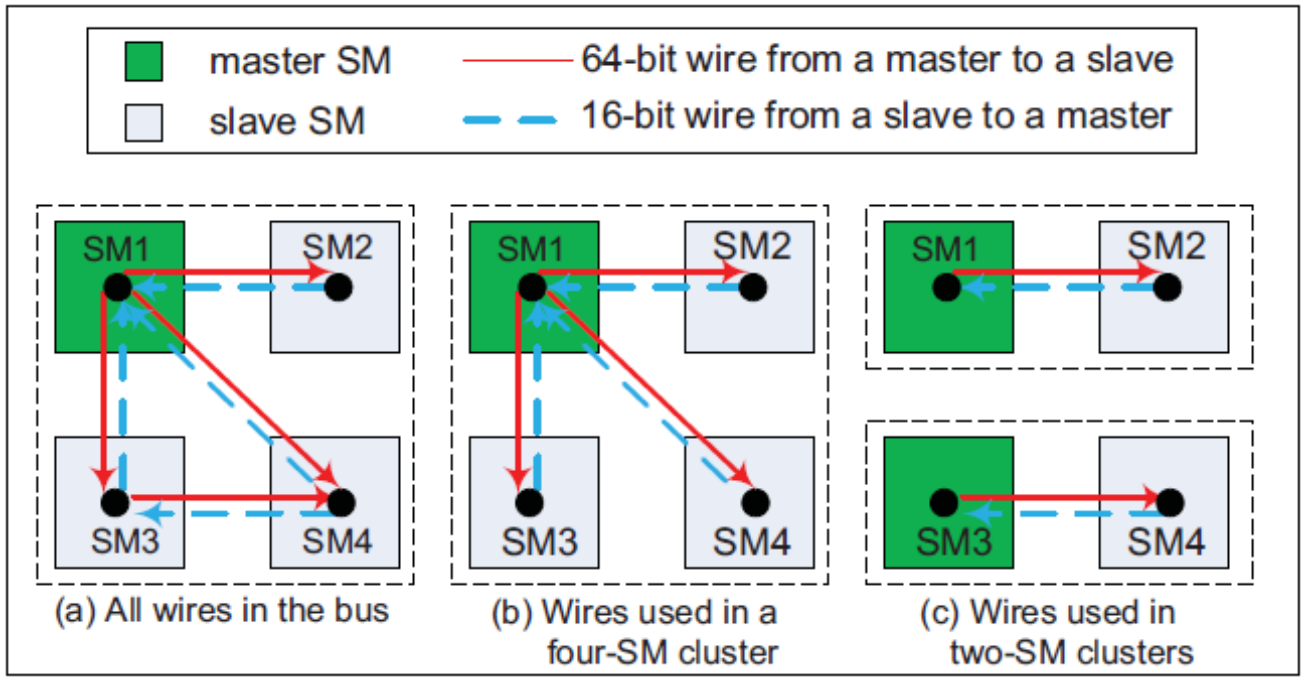
- **Several execution scenarios in the front-end sharing architecture**



Front-end sharing architecture(4/5)

NoC in the front-end sharing clusters

- There is a pair of wires connecting the master and every slave
- 64-bit from a master to a slave, 16-bit from a slave to a master
- Operates at twice the frequency of SM cores
- Totally 10 bytes wide, which is only 1/3 of the width of the GPU main interconnection network between the SMs and L2 cache



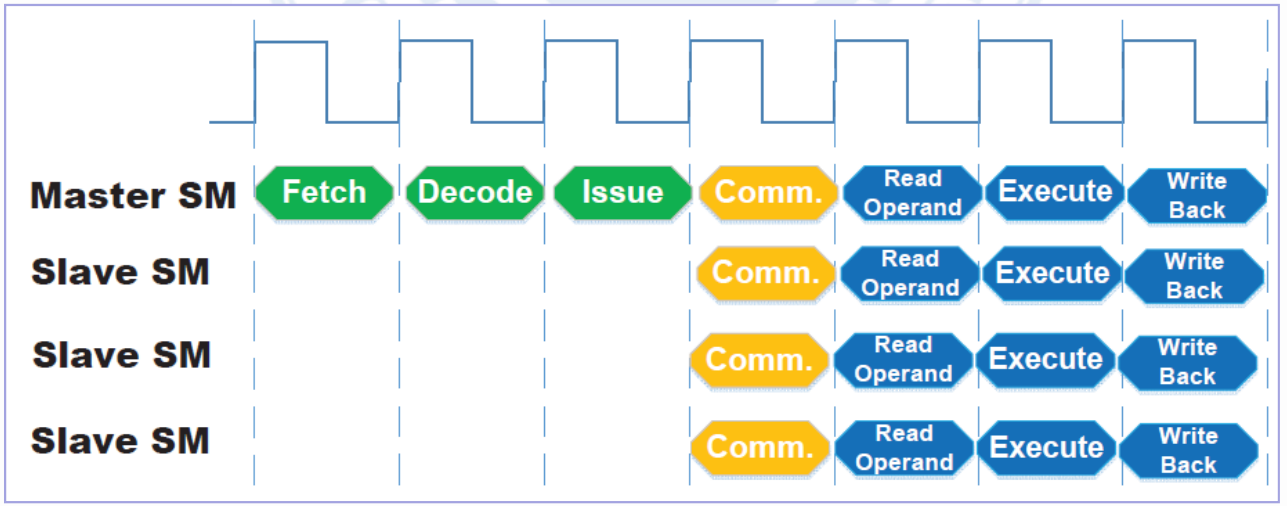
Front-end sharing architecture(5/5)

Pipeline stages in the front-end units

- A new ”communicate” stage is inserted between the issue and the read operand stage to transfer the packets between the master and its slaves

There are three types of data packets

- InstPacket: containing instruction information
- MemPacket: containing memory access “ACK” messages
- CtrlPacket: controling the cluster behavior such as ungrouping or regrouping



上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Experimental methodology (1/2)

Simulator architectural configuration: we simulated a Nvidia GTX480 GPU architecture using GPGPU-Sim 3.2.1

Configuration items	Value
Shaders (SMs)	16
Warp Size	32
Capacity / Core	MAX. 1536 Threads, 8 CTAs
Core / Memory Clock	700 MHz / 924 MHz
Interconnection Network	1.4 GHz, 32 bytes wide, crossbar
Registers / Core	32768
Shared Memory / Core	48KB
Constant Cache / Core	8KB, 2-way, 64B line
Texture Cache / Core	4KB, 24-way, 128B line
L1 Data Cache / Core	32KB, 4-way, 128B line
L1 I-Cache / Core	4KB, 4-way, 128B line
L2 Cache	64KB, 16-way, 128B line
warp scheduler	Greedy then Oldest(GTO)
DRAM Model	FR-FCFS memory scheduler, 6 memory modules

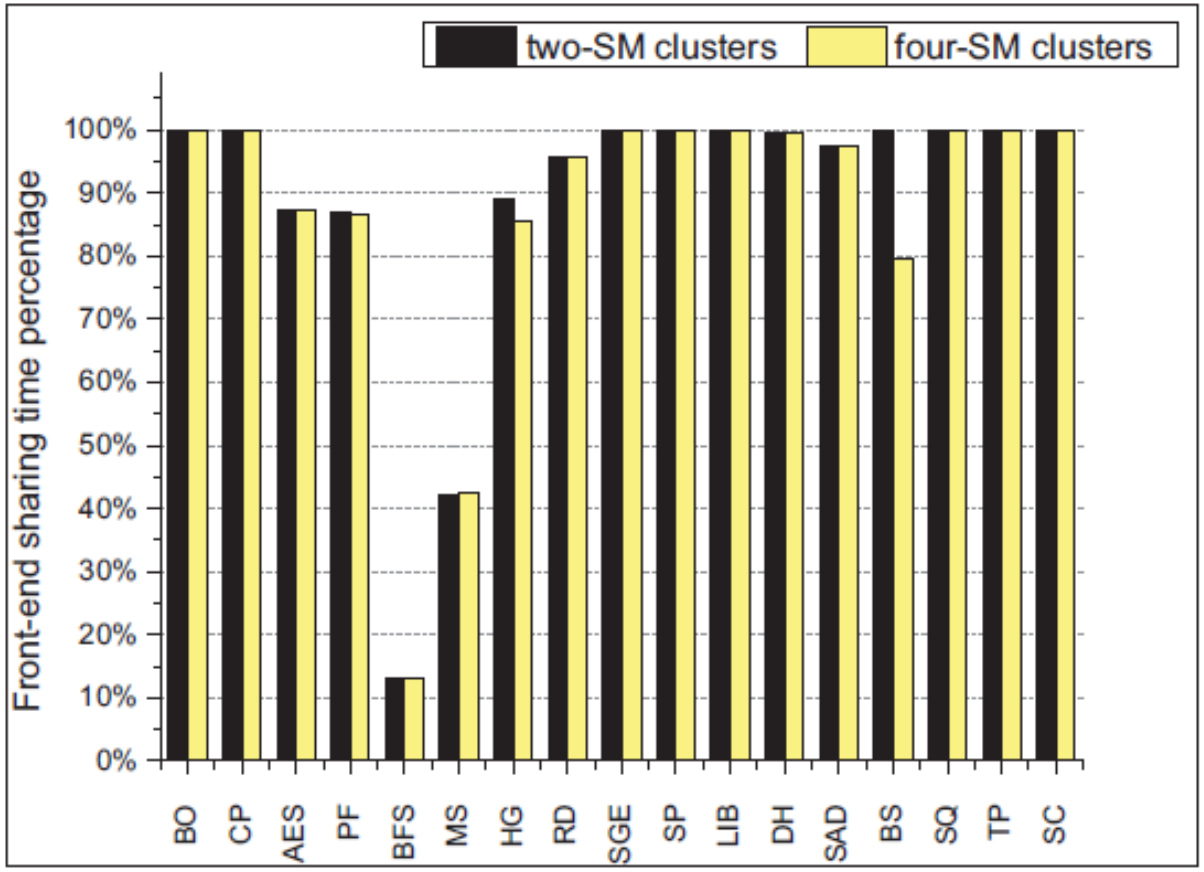
Experimental methodology (2/2)

- **Benchmarks: mixed benchmarks from various sources:**
  - NVIDIA CUDA SDK 4.1 [4]: BinomialOptions (BO), MergeSort (MS), Histogram (HG), Reduction (RD), ScalarProd (SP), dwtHarr1D (DH), BlackScholes (BS), SobolQRNG (SQ), Transpose (TP), Scan (SC)
  - Parboil [18]: sgemm (SGE), Sum of Absolute Difference (SAD)
  - Rodinia: PATH Finder (PF)
  - GPGPU-Sim benchmark suite [1]: Coul Potential (CP), AES Encryption (AES), BFS Search (BFS), Swap Portfolio (LIB)
- **Diverse application characteristics**
  - Memory-intensive apps: BS, SQ, TP, SC
  - Compute-intensive apps: BO, CP, AES, PF
  - Irregular apps: BFS, MS, HG

Results and analysis (1/4)

Front-end sharing percentage

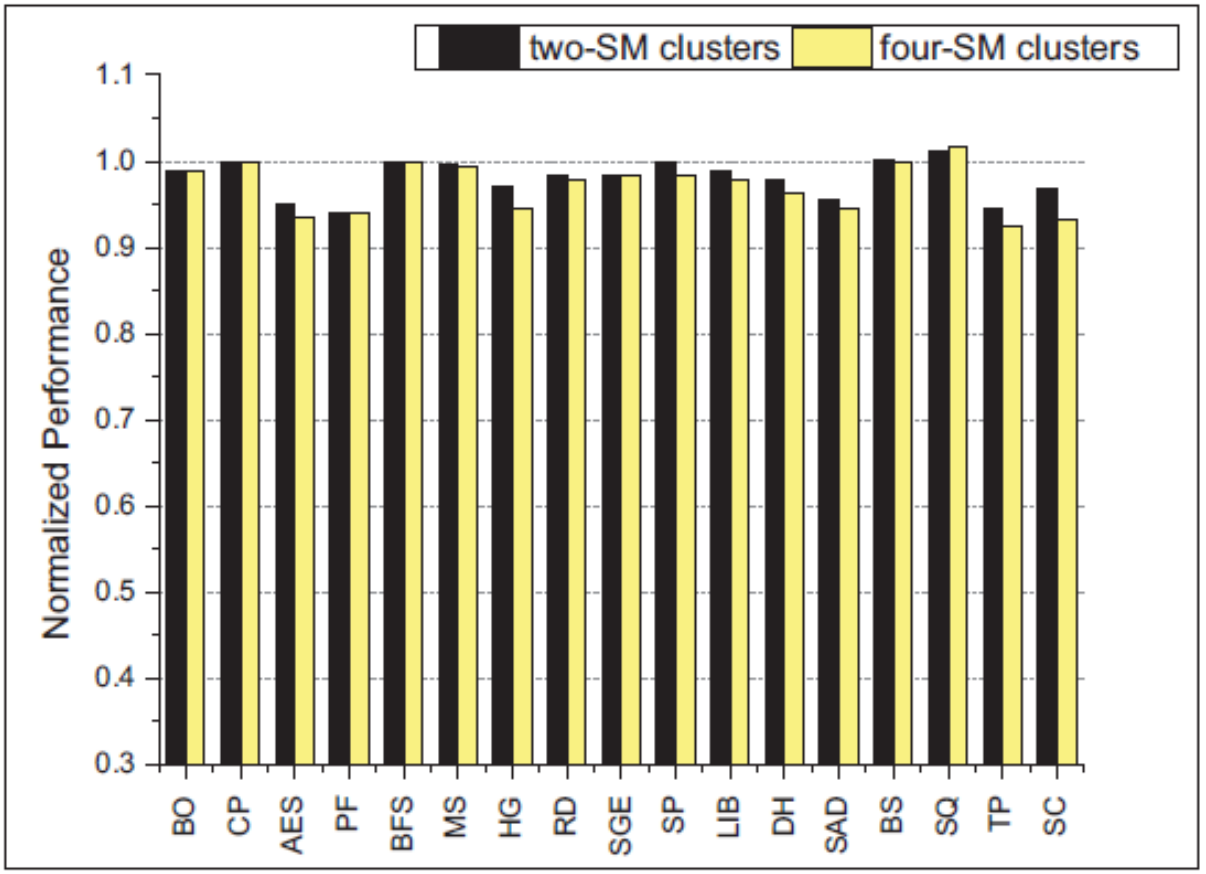
- Most applications are always in front-end sharing execution (no SM divergence)
- Irregular applications have small sharing time percentage



Results and analysis (2/4)

Performance

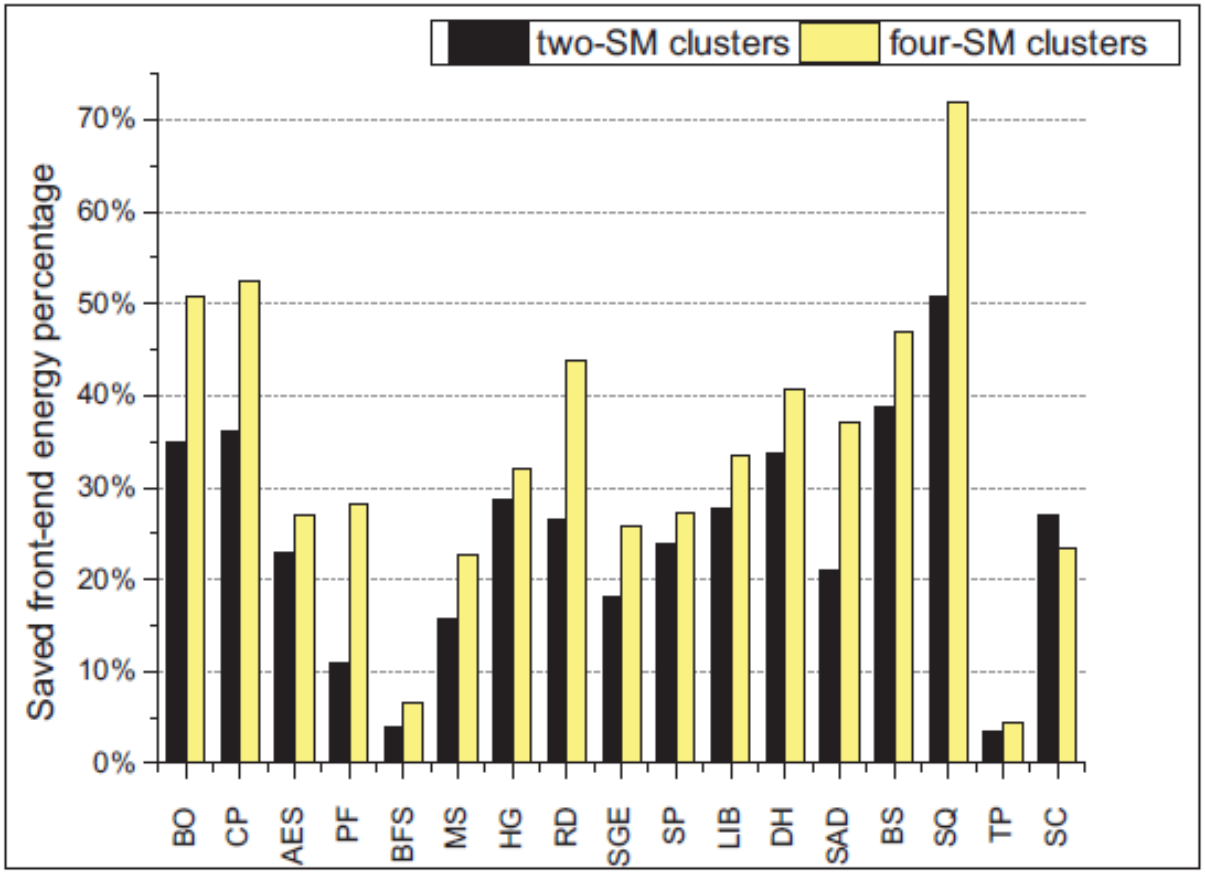
- The architecture achieves 98.0% and 97.1% normalized performance on average for two-SM cluster and four-SM cluster, respectively.
- Some applications suffer performance degradations due to increased memory access latency or instruction issue stalls.



Results and analysis (3/4)

Front-end energy savings

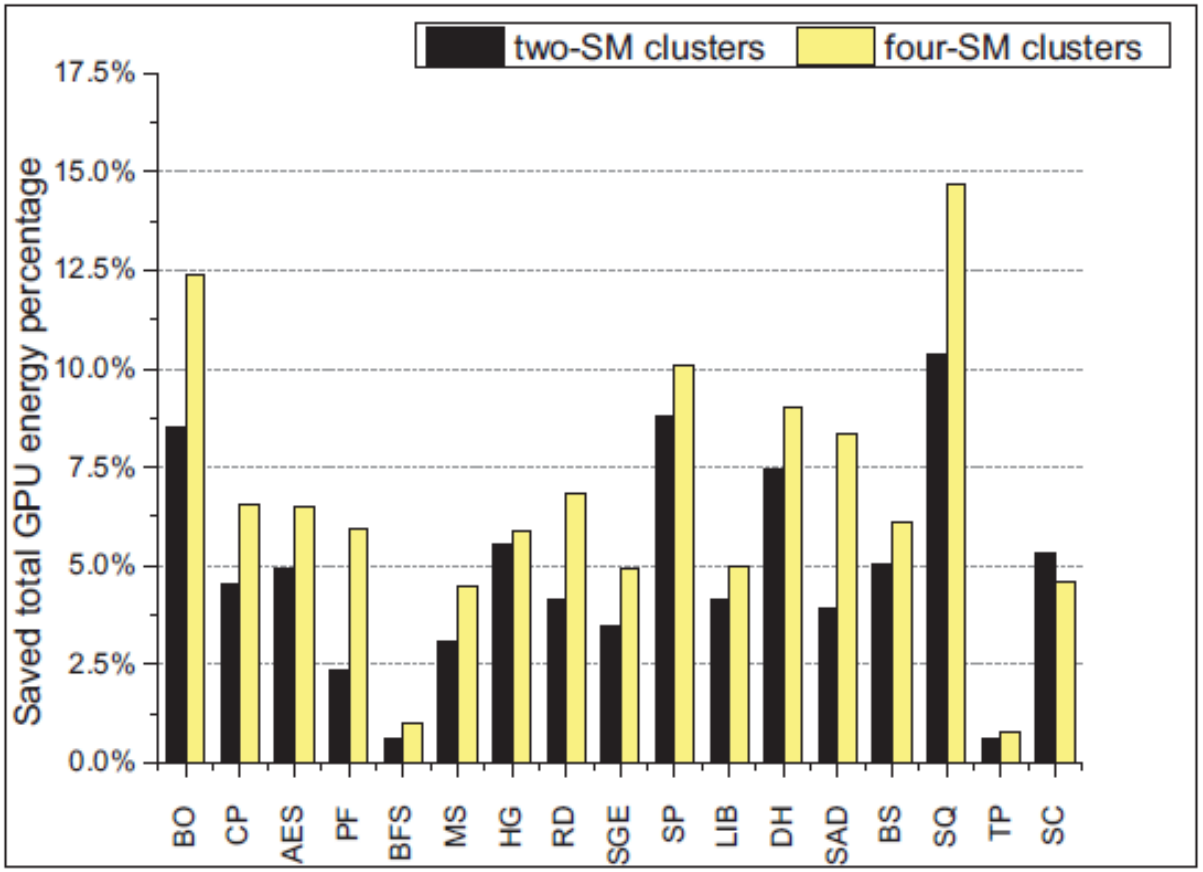
- On average, 24.9% and 33.7% front-end energy can be saved under two-SM cluster and four-SM cluster, respectively
- Four-SM cluster formation saves more energy since there are more power-gated slave SMs.



Results and analysis (4/4)

Total GPU energy savings

- On average, 4.9% and 6.8% total energy savings are obtained
- SQ saves the highest energy while BFS and TP save the least energy
- Three applications save more than 10% total energy



Conclusion

- We proposed a front-end sharing architecture to improve the energy efficiency in GPUs
- The architecture can save 6.8% on average and up to 14.6% of total GPU energy
- Experiments show that this architecture is effective for compute-intensive applications, memory-intensive applications and some irregular applications

Thanks

Q & A