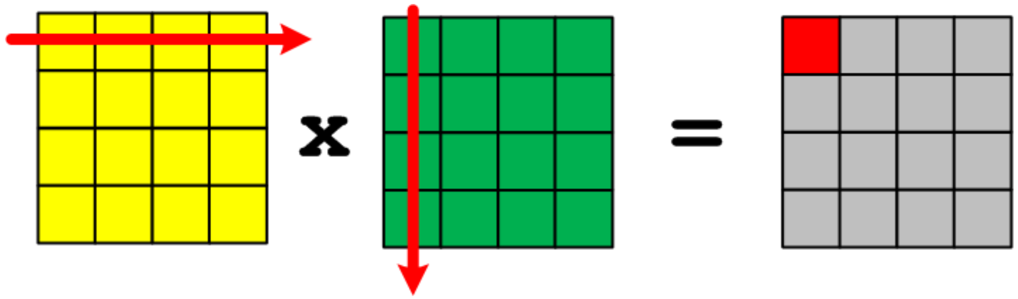
A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures

Minsoo Rhu Michael Sullivan Jingwen Leng Mattan Erez

The University of Texas at Austin

Introduction – (1)

- "General purpose computing" with GPUs
  - Enabled by non-graphics APIs (e.g., CUDA, OpenCL)
    - 'GP'GPU == massively multithreaded *throughput* processor
- Excellent for executing *regular* programs
  - Simple control flow
  - Regular memory access patterns (with high locality)
    - e.g., matrix-multiplication



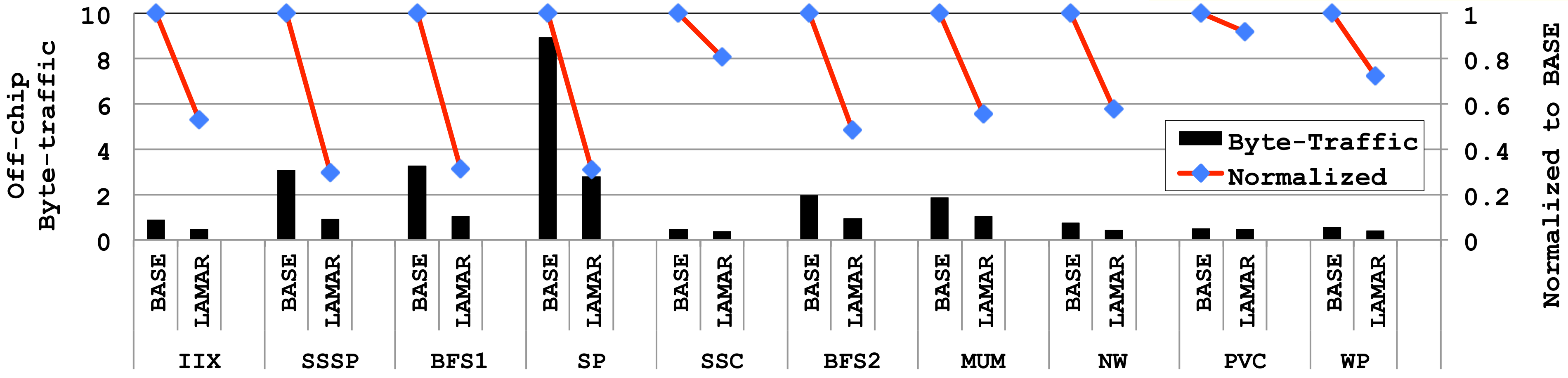
Introduction – (2)

- Importance in executing *irregular* apps increasing
  - Still embarrassingly parallel … but *complex* control flow and *irregular* memory access behavior
    - Molecular dynamics
    - Computer vision
    - Analytics
    - And much more …
- GPU's massive multi-threading + irregular memory
  - Small *per-thread* cache space available
  - Efficient caching becomes extremely challenging
    - Low hit rates
    - Low block reuse
    - Significant waste in off-chip bandwidth utilization

Problem / Our solution

- **Problem** : Waste in off-chip bandwidth utilization
- **Solution** : Locality-aware memory hierarchy (**LAMAR**)

***Lower* is better**



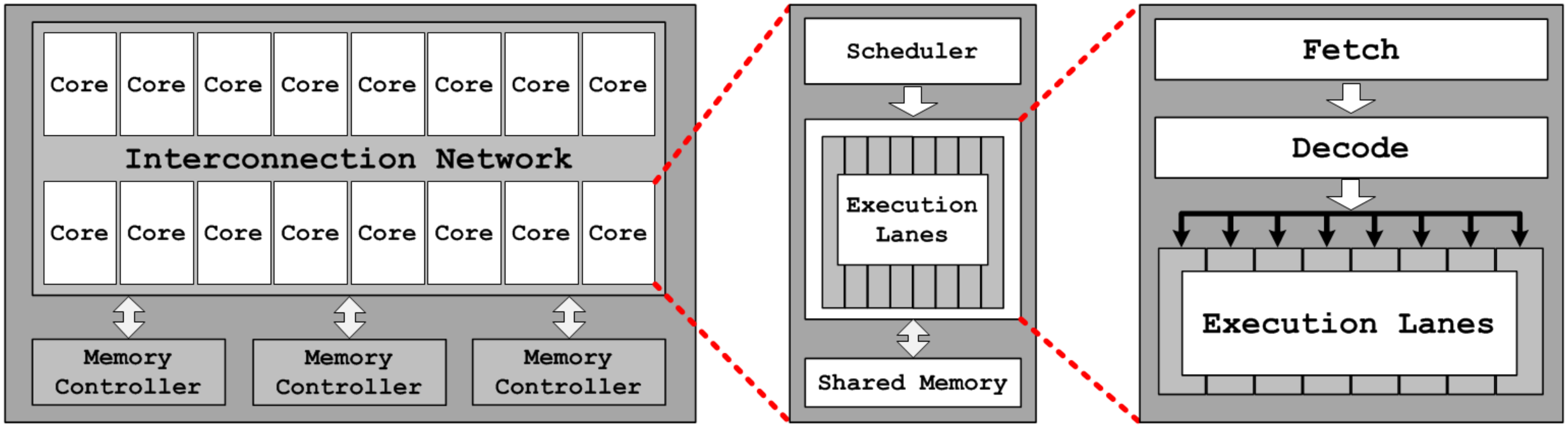
(a) [Byte-traffic/num_of_instrs] (left-axis) and the normalization to baseline memory system (right-axis).

BASE : *Baseline* memory system
LAMAR : *Proposed* solution

Background

Graphic Processing Units (GPUs)

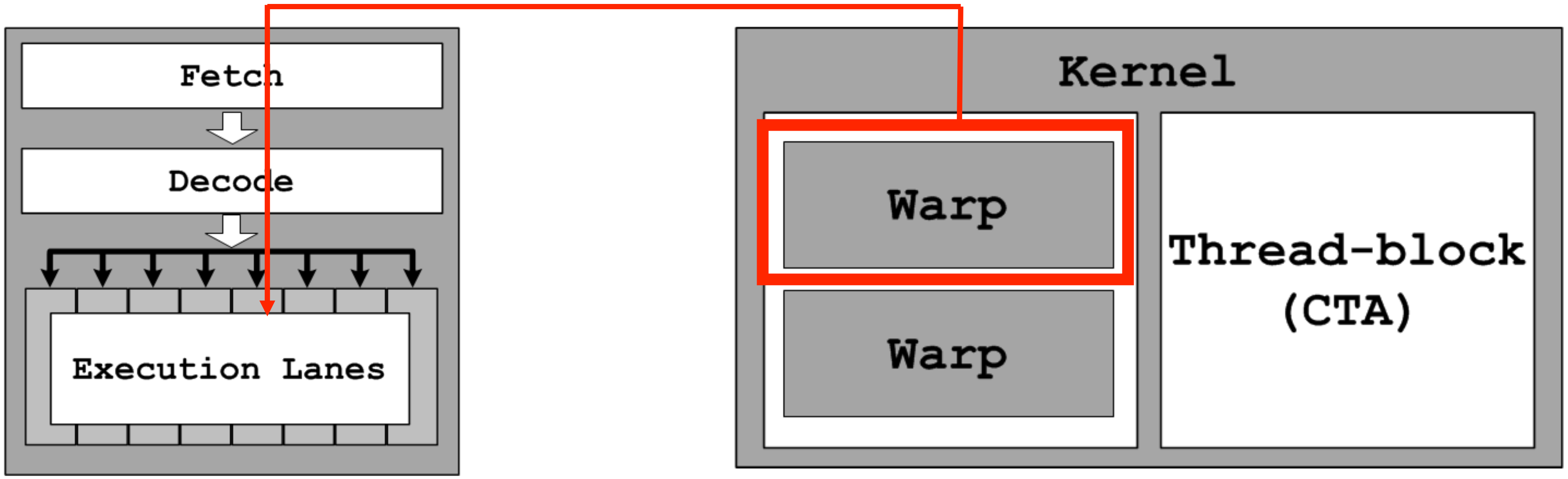
- General-purpose many-core accelerators
  - Use simple in-order shader cores in 10s / 100s numbers
- Scalar frontend (fetch & decode) + parallel backend
  - Amortizes the cost of frontend and control



CUDA exposes hierarchy of data-parallel threads

- **SPMD model:** single ***kernel*** executed by all _scalar_ threads
- **Kernel / Thread-block / *Warp* / *Thread***
  - Multiple **warps** compose a **thread-block**
  - Multiple **threads (32)** compose a warp

A warp is scheduled as a *batch* of scalar threads

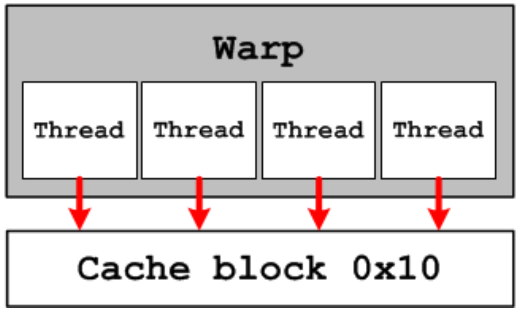


SIMT: Single-Instruction Multiple-Thread

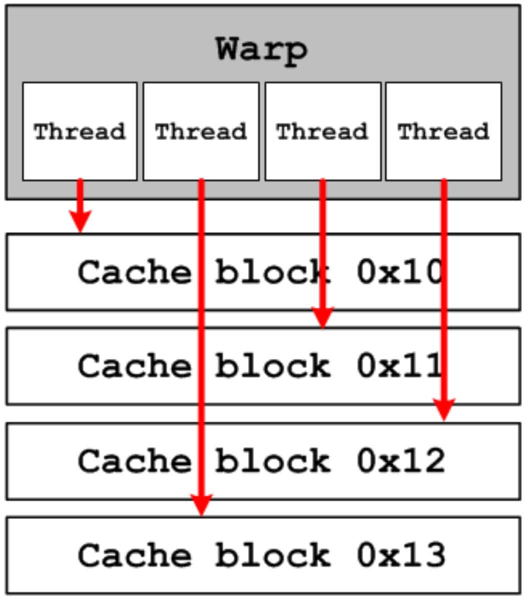
- Programmer writes code to be executed by ***scalar*** threads in a ***vector*** SIMD hardware.
  - HW/SW supports ***conditional branches***
    - Each thread can follow its own control flow
  - Fine-grained ***scatter-gather*** LD/STs
    - Each thread can LD/ST from arbitrary memory address

Memory divergence (*a.k.a address* divergence)

- Well-structured memory accesses are *coalesced* into a *single* memory transaction
  - e.g., all the threads in a warp access the ***same*** cache block
  - Minimized cache port contention, save port-BW
- A single warp (32 threads) can generate up to 32 memory transactions
  - Memory divergence
  - e.g., each thread within a warp accesses a ***distinct*** cache block region



(a) Regular memory accesses



(b) Irregular memory accesses

The problem:

GPU caching inefficiencies

GPUs leverage massive multithreading

- It is ***what makes them*** throughput processors!
  - Core of its latency-tolerance
    - Thread-level parallelism >> Instruction-level parallelism

- Massive multithreading + irregular memory accesses
  - ***Bursts*** of concurrent cache accesses within a given time frame
  - Orders of magnitude higher cache contention than CMP counterparts
    - Efficient caching becomes extremely challenging!

On-chip cache hierarchy of GPUs

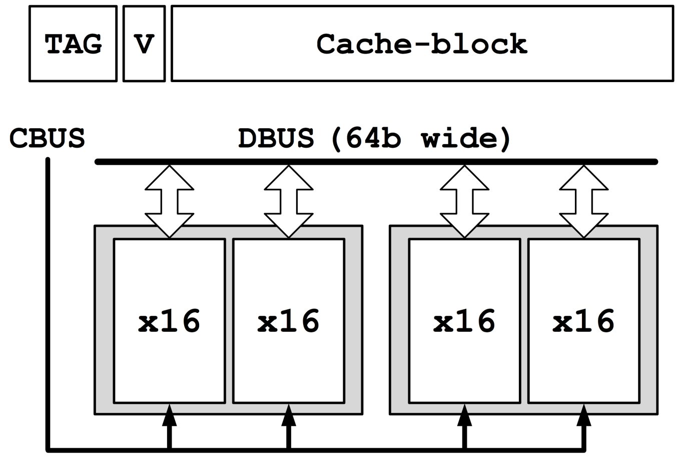
- Per-thread cache capacity of modern CPUs/GPU

Intel Core i7-4960x	IBM Power7	Oracle UltraSparc T3	NVIDIA Kepler GK 110
32KB L1 2 threads/core **16KB/thread**	32KB L1 4 threads/core **8KB/thread**	8KB L1 8 threads/core **1KB/thread**	48KB L1 2K threads/core **24B/thread**

- NVIDIA GPU cores issue LD/STs in **[32B – 128B]** granularity
  - Each scalar thread can request data of **[1B – 16B]***
  - HW address coalescer generates *one or more* **[32B – 128B]*** memory transactions per warp, depending on:
    - Control divergence (*subset* of active threads within a warp)*
    - Memory divergence

** 1B (char) – 16B (vector of floats)*
** NVIDIA, "CUDA C Programming Guide", 2013*
** Fung et al., "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow", MICRO-2007*

Baseline GPU memory hierarchy (Coarse-grained [CG] fetch-only)



(a) Baseline cache and memory system (V: valid)

- ***128B*** L1/L2 cache blocks
  - Cache misses invoke data requests in cache-block granularity (**CG**-fetches)
- Width of each memory-channel: 64b (two 32b **GDDR5** chips)
  - Fermi (GF110) / Kepler (GK110) / Southern-Island
- ***64B*** (64b x 8-bursts) minimum access granularity

Why is GPU caching so challenging?

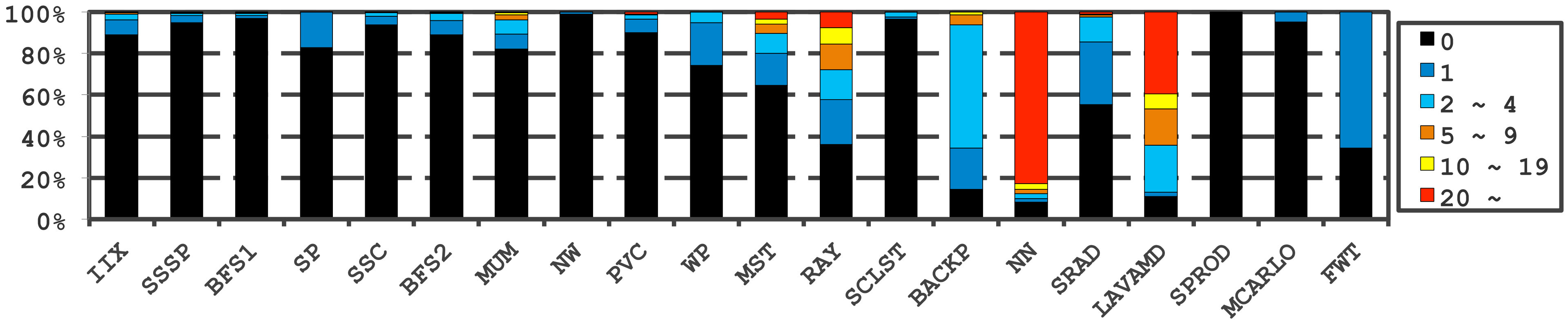
- Small per-thread cache capacity
  - 24B per thread (worst case)

- Larger cache block size
  - 128B per cache block

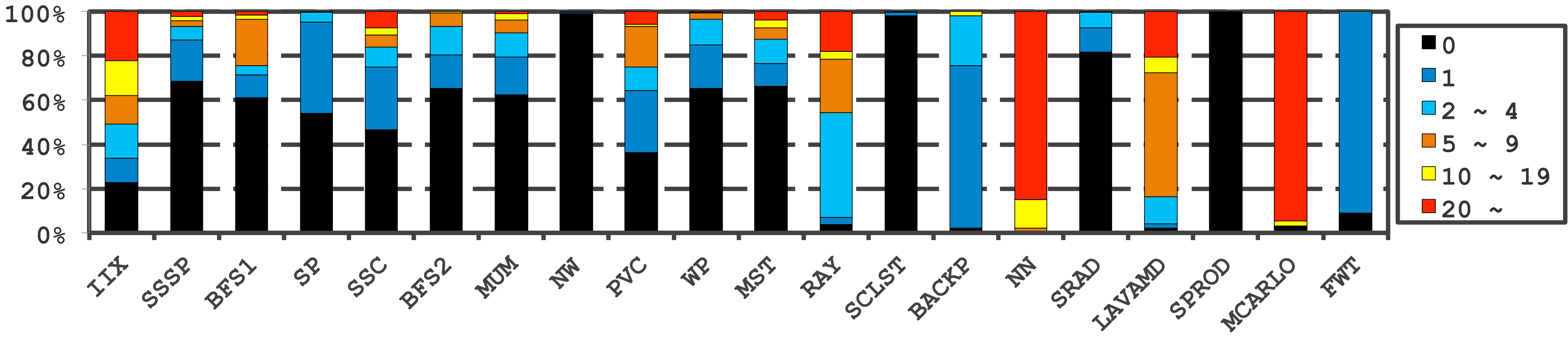
- Caching efficiency is significantly compromised!
  - High miss rate
  - Low block reuse
  - Sub-optimal utilization of off-chip bandwidth

Cache block reuse (*temporal*)

- Number of repeated accesses to L1/L2 cache blocks, after fill (before eviction)



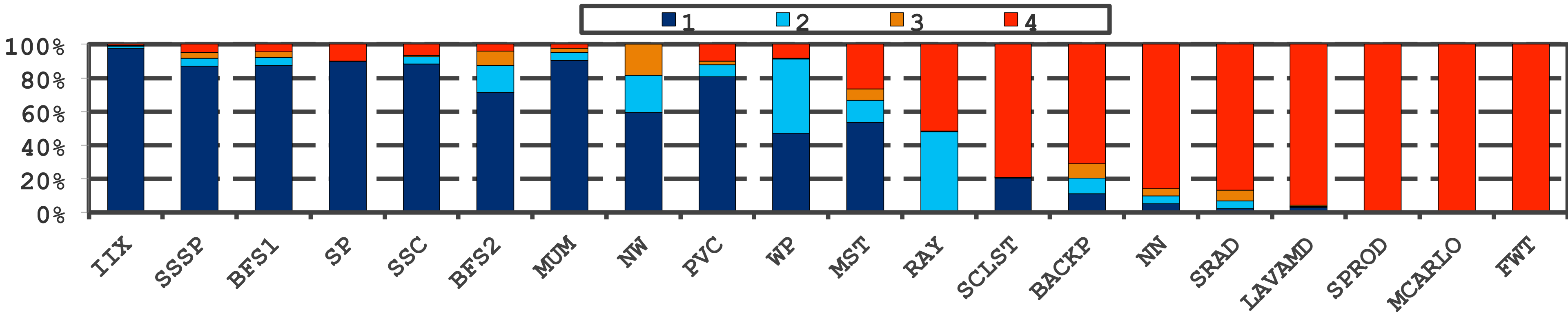
(a) L1 cache



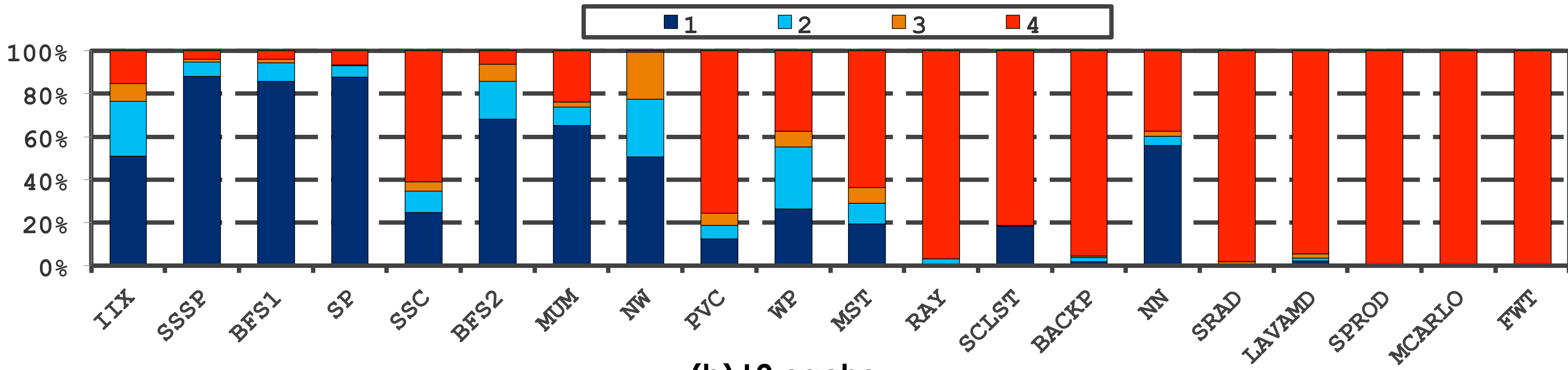
(b) L2 cache

Cache block reuse (*spatial*)

- Number of 32B chunks (or ***sectors***) in a cache block actually referenced in L1/L2 (128B cache block)



(a) L1 cache

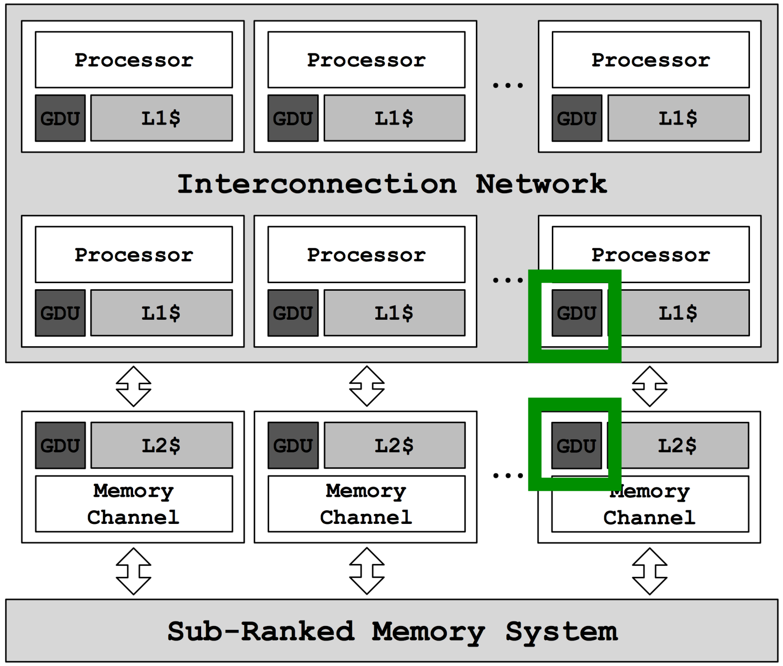


(b) L2 cache

Our solution:

Locality-aware memory hierarchy

LAMAR: Locality-aware memory hierarchy



Static-GDU: *Always* fetch CG or FG
Dynamic-GDU : choose at runtime

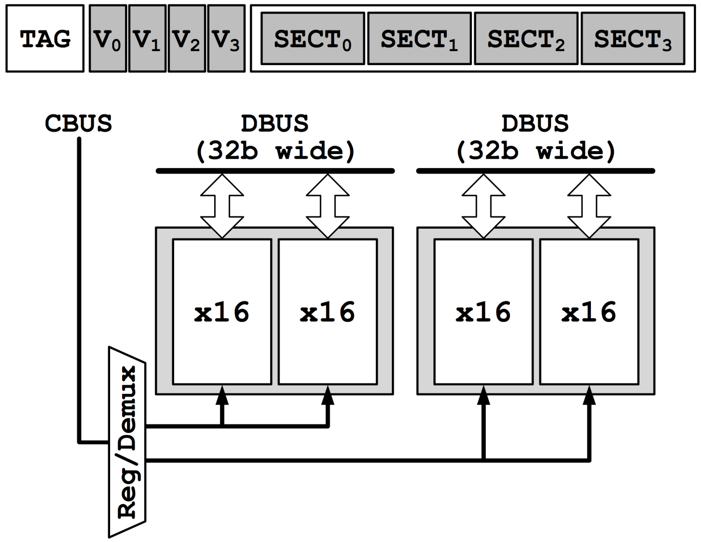
GDU: Granularity Decision Unit

Determines whether to fetch missed block in CG or in FG

- Provide ***FG*** data fetching capability to save BW
  - ***Motivation***: massive multithreading limits cache block lifetime, so the likelihood of block reuse is very low in GPUs
    - Effectiveness of prefetching (e.g., filling all 128B) decreases
    - Sectored caches* + sub-ranked* memory system

** Liptay, "Structural aspects of the system/360 model 85, Part II: The cache", IBM Journal, 1968*
** Zheng et al., "Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency", MICRO-2008*

LAMAR memory systems (Fine-grained [FG]-enabled)



(a) Memory hierarchy with a sectored cache and a sub-ranked memory system (128B cache block, V: valid)

- Sectored cache: amortize tag-overhead
  - 32B sector, 4-sectors per cache block (hence a single tag)
- Each channel divided into ***two*** sub-ranks (retrieve data from 1 chip)
  - 32B (32b x 8-bursts) minimum access granularity
  - Allows finer control of data fetches from DRAM (64B vs 32B)

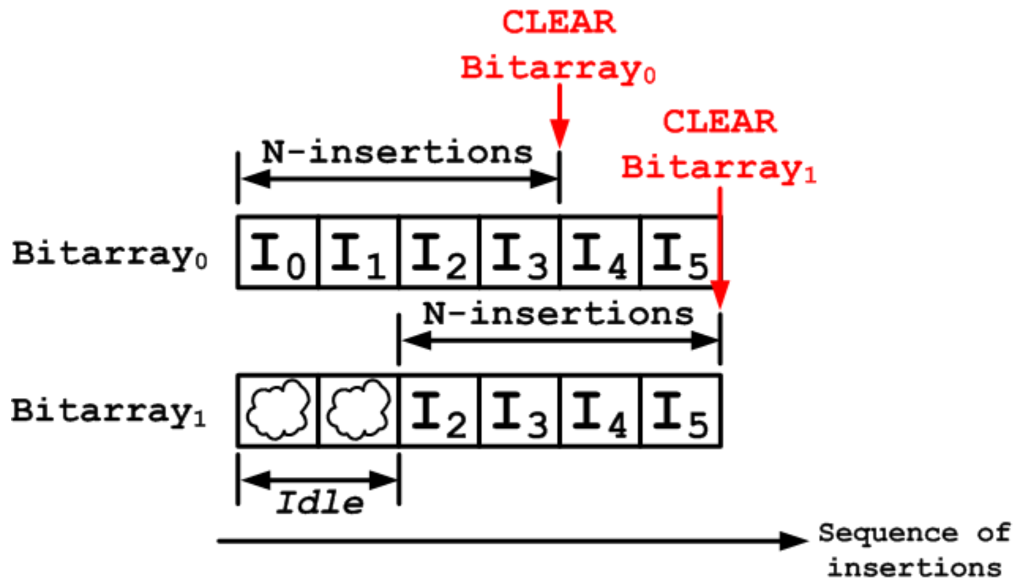
Predicting Access Granularity

Bi-modal granularity predictor (BGP)

- Predictor provides *bi-modal* prediction
  - **Coarse granularity (CG)**
  - **Fine granularity (FG)**

- Fetch ***all*** block-wide data (**CG**) or ***just enough to service*** the data requested from the GPU core (**FG**)
  - Reduce the number of RD/WR commands sent to DRAM
  - Reduce byte-traffic sent to off-chip memory

Light-weight BGP microarchitecture



- Bloom-filter based design with dual-bitarrays
  - ***Temporally delayed/overlapped*** for history insertions
    - The one with more insertions are used for **TEST**ing membership
    - The older bitarray is **CLEAR**ed at regular intervals (***N***-ins.)
    - When old bitarray is cleared, **other one** used for **TEST**ing
  - Balance size, false positive rate, and history depth

Granularity prediction algorithm

- Predictor has a **default** prediction (like ***agree predictor****)
  - Initialized as CG (or FG) at kernel initialization

- Each cache miss queries the bloom-filter
  - **INTUITION.** check whether missing block's *optimal* fetching-prediction **agrees*** with the default prediction

- **What**/**when** are elements inserted into the bloom-filter?
  - **What**: Evicted block-address (+ spatial reuse information)
  - **When**: Evicted block's reuse information **disagrees*** with the default prediction

***** *Sprangle et al., "The agree predictor: A mechanism for reducing negative branch history interference", ISCA-1997***

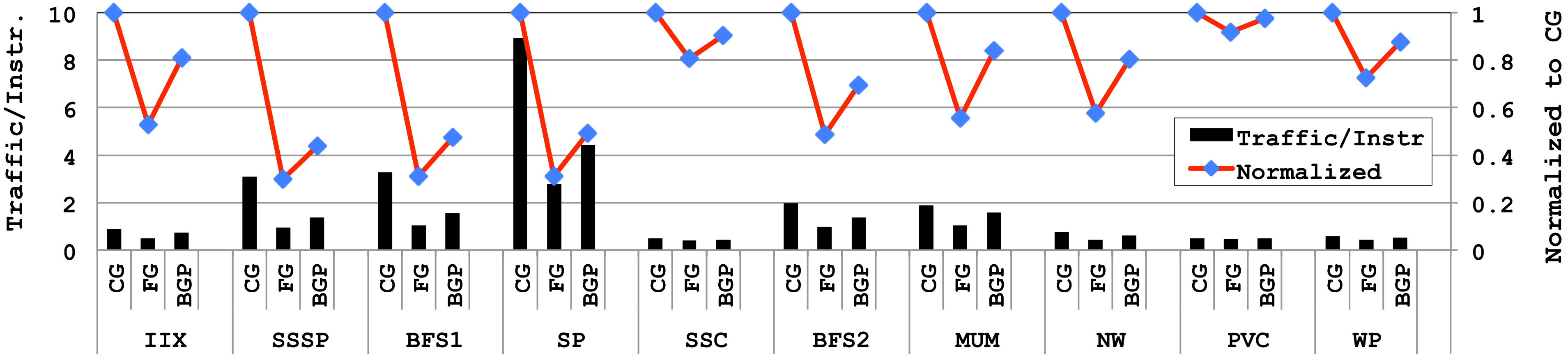
Key experiments and analysis

Simulation environment

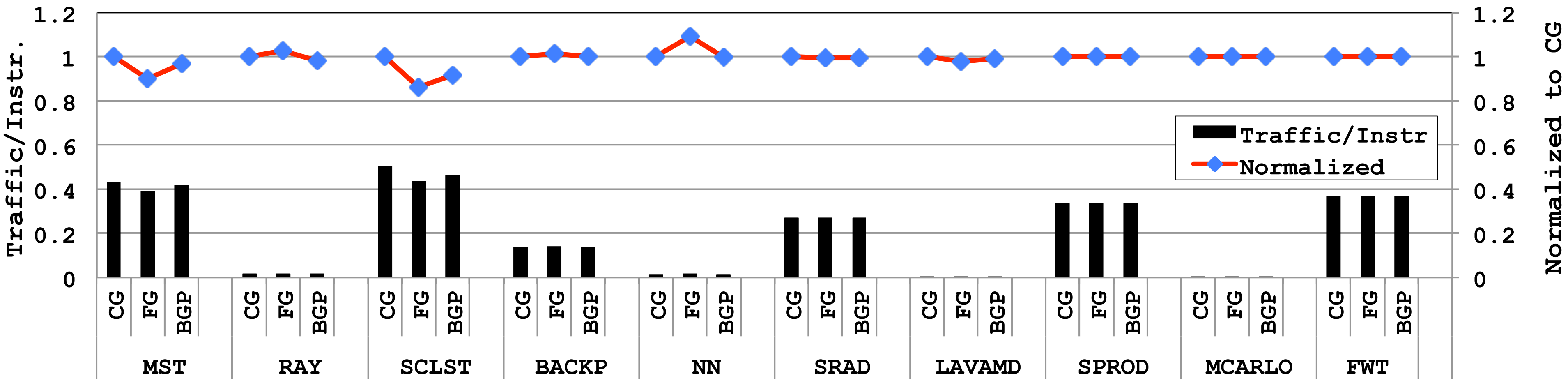
- GPGPU-Sim
  - Processor architecture
    - Similar to GTX 480
  - Memory hierarchy
    - Sectored cache
    - Sub-ranked memory system (using ***DrSim****)
    - Memory bandwidth
      - 179.2 GB/s overall (through 8 channels)
- Applications
  - CUDA-SDK, Rodinia, LonestarGPU, MapReduce
  - Characterization
    - FG-leaning (avg. number of sectors referenced ≤ 2.0)
    - CG-leaning (avg. number of sectors referenced > 2.0)

*** http://lph.ece.utexas.edu/public/Main/DrSim**

Off-chip byte traffic (normalized to # of instructions)



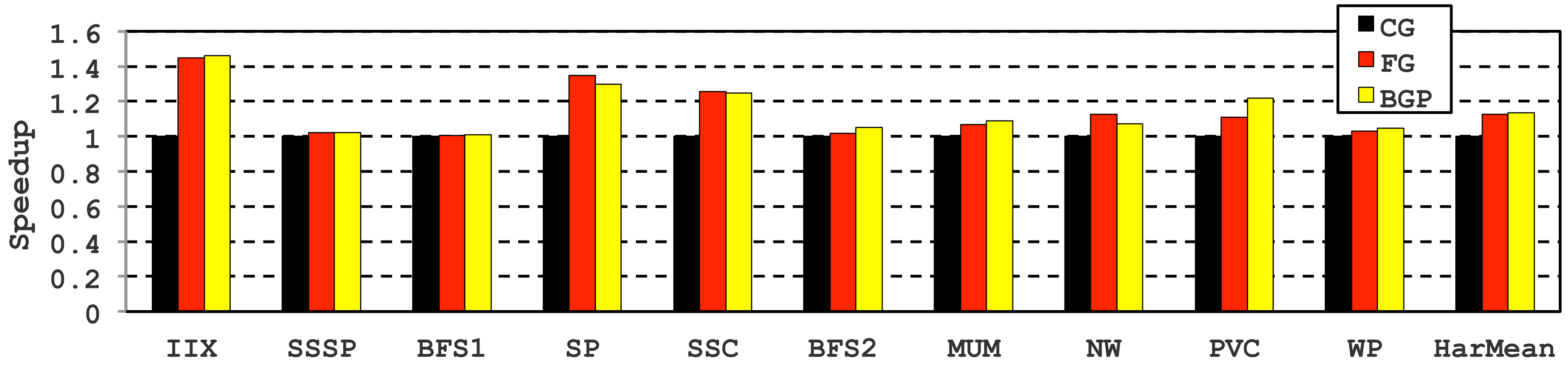
(a) FG-leaning applications



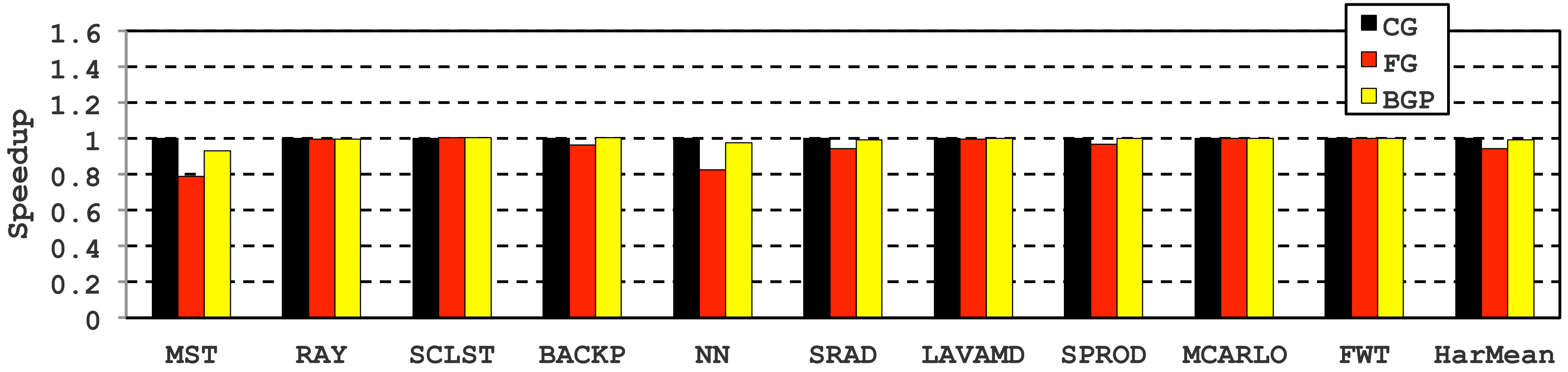
(b) CG-leaning applications

Performance improvements

Higher is better

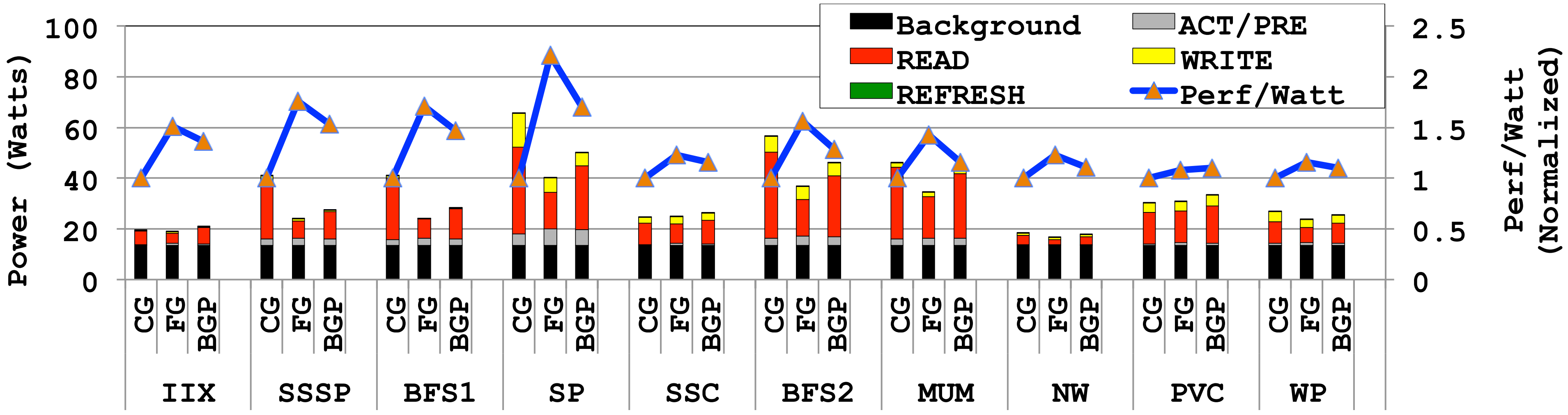


(a) FG-leaning applications

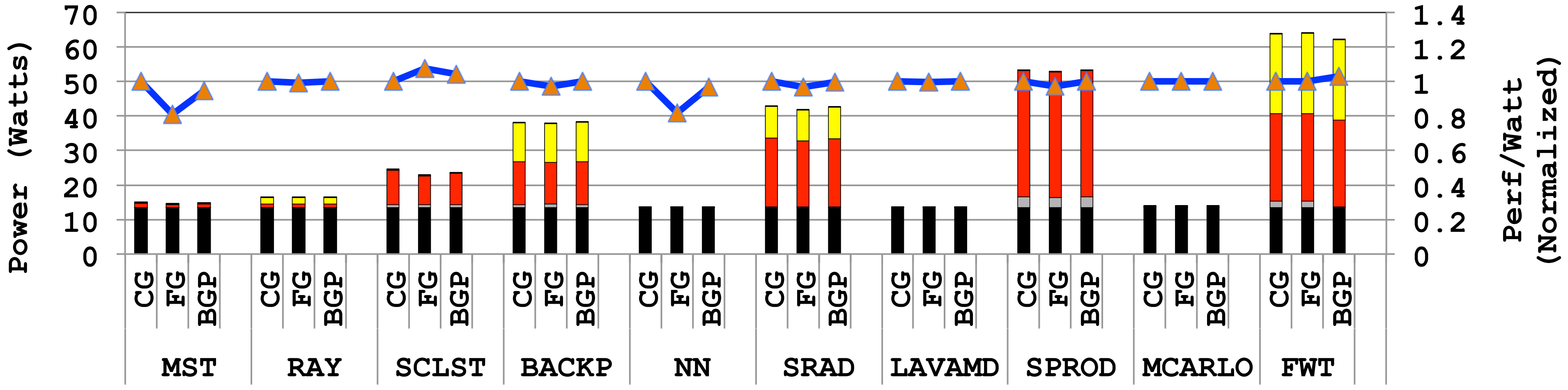


(b) CG-leaning applications

DRAM power consumption



(a) FG-leaning applications



(b) CG-leaning applications

LAMAR conclusions

- Memory hierarchy of GPUs necessitates fine-grained access granularity
  - Inherent data locality is rarely captured in GPUs
  - Minimizing useless overfetching (within cache block) improves BW utilization
    - Reduces DRAM power consumption
    - Improves performance
    - *Perf/Watt* enhanced