# Cache Capacity Aware Thread Scheduling for Irregular Memory Access on Many-Core GPGPUs

Hsien-Kai Kuo, Ta-Kan Yen, Bo-Cheng Charles Lai and Jing-Yang Jou

Department of Electronics Engineering
National Chiao Tung University, Taiwan
Email : hkkuo[at]ee.eda.nctu.edu.tw

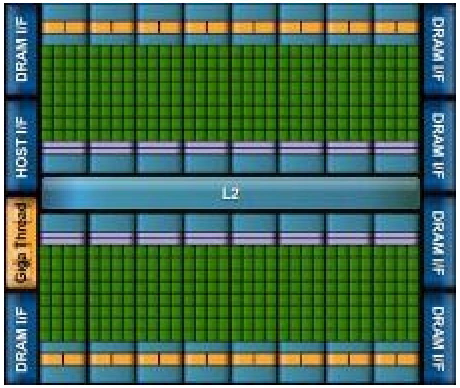
ASP-DAC 2013

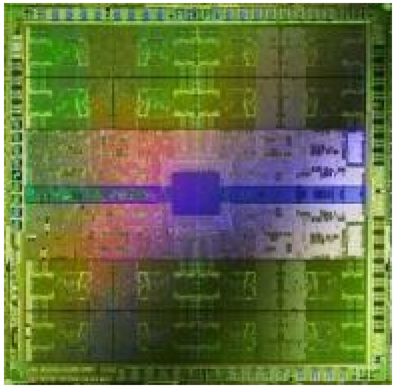
# Outline

- ☐ Introduction
- ☐ GPGPU Background
- ☐ Motivational Examples
- ☐ Cache Capacity Aware Thread Scheduling
- ☐ Experimental Results
- ☐ Conclusions

# Introduction – GPGPU

- **G**eneral **P**urpose **G**raphic **P**rocessing **U**nit
  - An accelerator for general computing
  - Numerous computing cores (> 512 cores/chip)
  - Throughput-oriented





- Techniques to alleviate memory bottleneck
  - **Memory Coalescing**
  - **On-chip Shared Cache**

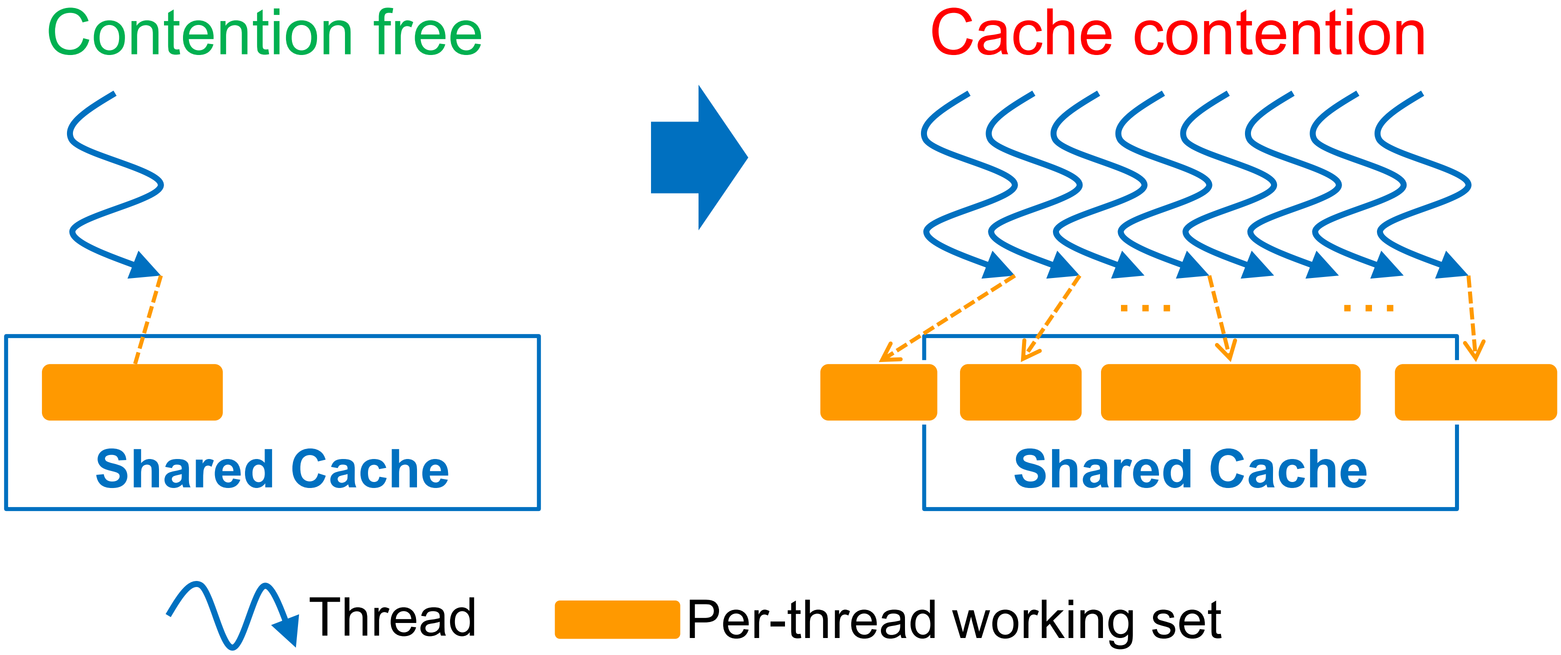
Source: Nvidia, http://http://www.nvidia.com

# Introduction – Alleviate Memory Bottleneck

- **Memory Coalescing**
  - Combine several narrow accesses into **a single wide** one
  - Effective and widely used in regular applications
    - Fast Fourier Transform (FFT) and Matrix Multiplications
- **On-chip Shared Cache**
  - Shared among several computing cores
  - Automatically exploit **data reuse**
- However, in **Irregular Applications**
  - Lack of coordinated memory access (**Non-Coalescing**)
  - Numerous threads with limited cache capacity (**Cache Contention**)

# Introduction – Cache Contention

- **Cache Contention**
  - Happen when the cache capacity is insufficient for all the concurrent threads
  - Example :



# Introduction – Previous Studies

- Previous studies
  - **Deng, et al. (ICCAD'09)**
    - Scratch-pad memory to enhance coalescing
  - **Zhang, et al. (ASPLOS'11)**
    - Data and computation reordering to improve coalescing
  - **Kuo, et al. (ASPDAC'12)**
    - Thread clustering to enhance coalescing and mitigate cache contention
- Without considering the **Cache Capacity**
  - Cannot fully resolve the **Cache Contention** issue

Y. Deng, et al., "Taming Irregular EDA Applications on GPUs," in *ICCAD*, 2009

E. Z. Zhang, et al., "On-the-Fly Elimination of Dynamic Irregularities for GPU Computing," in *ASPLOS*, 2011

H.-K. Kuo, et al., "Thread Affinity Mapping for Irregular Data Access on Shared Cache GPGPU," in *ASPDAC*, 2012

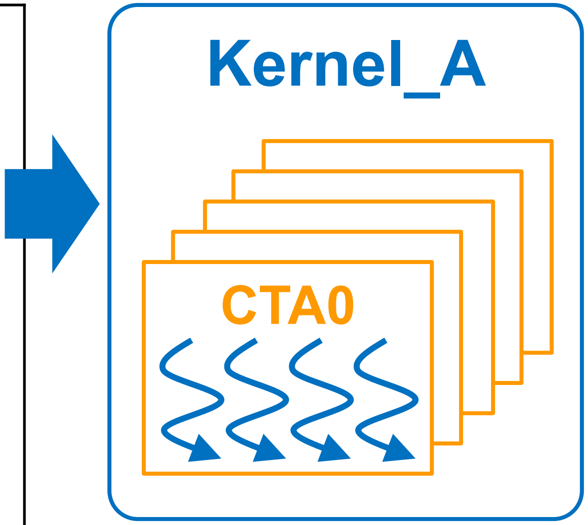
# Introduction – Contributions

- This paper
  - Formulate a general thread scheduling problem on GPGPUs
    - **Cache Capacity Aware Thread Scheduling Problem**
  - Carry out a comprehensive analysis on the variants of the problem
    - Nvidia's Fermi architecture is modeled as a special variant
  - Propose thread scheduling algorithms for different variants
    - An average of **44.7%** cache misses reduction
    - An average of **28.5%** runtime enhancement

# GPGPU Background – Programming Model

- Nvidia's CUDA Programming Model
  - **C**ooperative **T**hread **A**rray (**CTA**)
    - A collection of threads
  - **Kernel**
    - A collection of CTAs

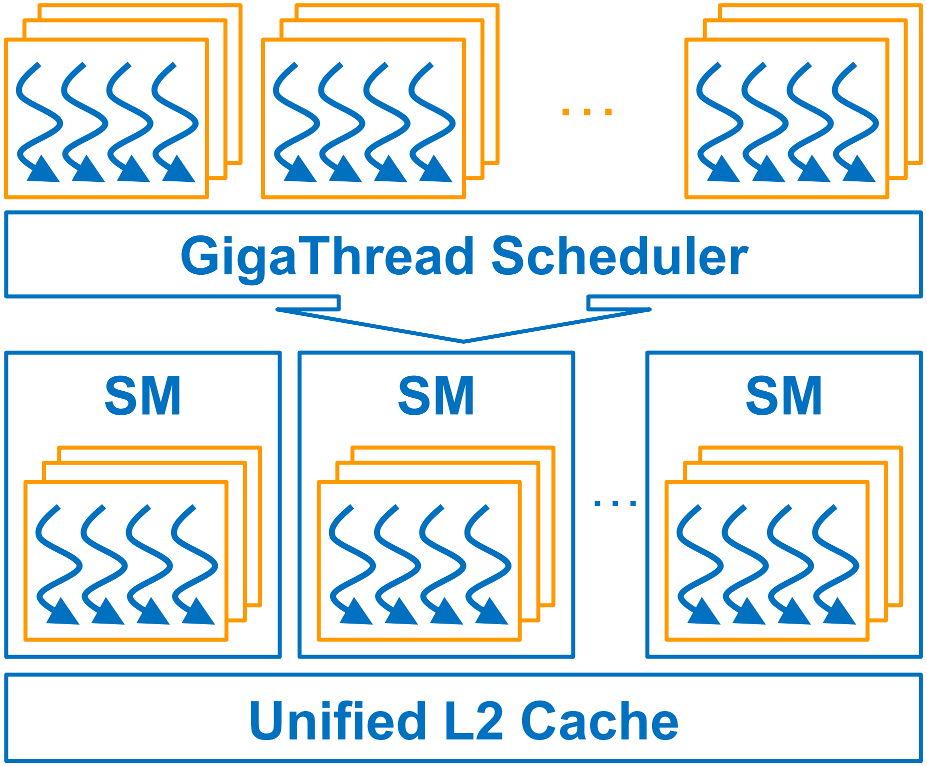
```
int main(){
        /* serial code*/
        …
        kernel_A<<<192, 256>>>(arg0, arg1, …)
        …
        /* serial code*/
        …
        kernel_B<<<256, 192>>>(arg0, arg1, …)
        …
}
```



Source: Nvidia, http://http://www.nvidia.com

# GPGPU Background – GPGPU Architecture

- Nvidia’s Fermi GPGPU Architecture
  - **Streaming** **Multiprocessor (SM)**
  - **Unified L2 Cache**
  - **GigaThread Scheduler**
    - Fixed number of concurrent CTAs
- This paper
  - Consider **re-configuring** the number of concurrent CTAs
    - Need **synchronizations**



Source: Nvidia, http://http://www.nvidia.com

# Motivational Examples – Example 1

- Assume that
  - A collection of CTAs = {A, B, C, D, E, F, G, H, I, J, K, L}
  - Working set sizes = {1, 8, 3, 1, 2, 2, 1, 7, 4, 4, 2, 5}
  - Cache capacity = 10
  - Maximum number of concurrent CTA = 4
- Example 1

| Example 1 : Cache Capacity Agnostic Scheduling | | |
|---|---|---|
| **Scheduling Steps** | **Concurrent CTAs** | **Cache Contention Evaluation** |
| Step1 | A, B, C, D | 1 + 8 + 3 + 1 = 13 > 10 (Contention) |
| Step2 | E, F, G, H | 2 + 2 + 1 + 7 = 12 > 10 (Contention) |
| Step3 | I, J, K, L | 4 + 4 + 2 + 5 = 15 > 10 (Contention) |

# Motivational Examples – Example 2

- Example 2
  - Too restrictive to schedule more concurrent CTAs

**Example 2 : Cache Capacity Aware Scheduling with Fixed Number of Concurrent CTAs**

| Scheduling Steps | Concurrent CTAs | Cache Contention Evaluation |
|---|---|---|
| Step1 | B, E | 8 + 2 = 10 ≤ 10 (Contention free) |
| Step2 | C, H | 3 + 7 = 10 ≤ 10 (Contention free) |
| Step3 | L, J | 5 + 4 = 9 ≤ 10 (Contention free) |
| Step4 | F, I | 2 + 2 = 6 ≤ 10 (Contention free) |
| Step5 | A, K | 1 + 2 = 3 ≤ 10 (Contention free) |
| Step6 | D, G | 1 + 1 = 2 ≤ 10 (Contention free) |

# Motivational Examples – Example 3

- Example 3
  - Should also consider the synchronization cost

**Example 3 : Cache Capacity Aware Scheduling with Reconfigurable Number of Concurrent CTAs**

| Scheduling Steps | Concurrent CTAs | Cache Contention Evaluation |
|---|---|---|
| Step1 | B, E | 8 + 2 = 10 ≤ 10 (Contention free) |
| Step2 | C, H | 3 + 7 = 10 ≤ 10 (Contention free) |
| **Synchronize and re-configure the number of concurrent CTAs** | | |
| Step3 | L, K, F, J | 5 + 2 + 2 + 1 = 10 ≤ 10 (Contention free) |
| Step4 | J, I, D, G | 4 + 4 + 1 + 1 = 10 ≤ 10 (Contention free) |

# Cache Capacity Aware Thread Scheduling – Problem Formulation (1/4)

- Input
  - $\boldsymbol{c^n}$ : a collection of CTAs
    - $\boldsymbol{c^n} = \{c_1, c_2 \cdots, c_n\}$
    - $\boldsymbol{w(c_i)}$ : working set size of the CTA $c_i$
- Output
  - $\boldsymbol{s^m}$ : a schedule of CTAs (a series of scheduling step)
    - $\boldsymbol{s^m} = \{s_1, s_2 \cdots, s_m\}$
      - Each scheduling step $s_i$ is a subset of $c^n$
    - $\boldsymbol{conc(s_i)}$ : concurrency of the scheduling step $s_i$
      - Number of CTAs belongs to $s_i$

# Cache Capacity Aware Thread Scheduling – Problem Formulation (2/4)

- Constraint (**Cache Capacity**)
  - $\forall s_i : \sum_{c_j \in s_i} w(c_j) \leq \boldsymbol{Cap\_unified\_L2}$
- Cost Function
  - $\boldsymbol{m} + \boldsymbol{sync\_cost(s^m)}$ : overall cost of the schedule $s^m$
    - $\boldsymbol{m}$ : total number of scheduling steps
    - $\boldsymbol{sync\_cost(s^m)}$ : total synchronization cost
      - $\boldsymbol{sync\_cost(s^m)} = \boldsymbol{cps} \times \sum_{i=0}^{m-1} \boldsymbol{sync(s_i, s_{i+1})}$
        - $\boldsymbol{sync(s_i, s_{i+1})}$ : necessity of synchronization
          - $\boldsymbol{sync(s_i, s_{i+1})} = \begin{cases} 0, & conc(s_i) = conc(s_{i+1}) \\ 1, & conc(s_i) \neq conc(s_{i+1}) \end{cases}$
        - $\boldsymbol{cps}$ : cost per synchronization
          - $\boldsymbol{cps} \in \mathbb{R}, 0 < cps \leq 1$

# Cache Capacity Aware Thread Scheduling – Problem Formulation (3/4)

- Problem Definition
  - **Cache Capacity Aware Thread Scheduling Problem** : Given a collection of CTAs $\boldsymbol{c^n}$ with working set size $\boldsymbol{w(c_i)}$, the problem is to find a schedule $\boldsymbol{s^m}$ where the overall cost is minimized subject to cache capacity constraint:

**minimize** $\boldsymbol{m} + \boldsymbol{sync\_cost(s^m)}$

**subject to** $\forall s_i: \sum_{c_j \in s_i} w(c_j) \leq \boldsymbol{Cap\_unified\_L2}$

$$\forall s_i \neq s_j : s_i \cap s_j = \emptyset$$

$$s_1 \cup s_2 \cdots s_m = c^n$$

# Cache Capacity Aware Thread Scheduling – Problem Formulation (4/4)

- NP-hardness
  - Lemma 1 : The Cache Capacity Aware Thread Scheduling Problem is NP-hard
  - Proof : The NP-hard problem, **Bin Packing Problem** can be reduced to this problem
- P ≠ NP
  - No optimal algorithm in polynomial time
  - Acceptable quality in polynomial time
    - Approximation algorithms

# Cache Capacity Aware Thread Scheduling – Fixed Concurrency (1/2)

- Fixed Concurrency Constraint
  - $\forall s_i \neq s_j : conc(s_i) = cons(s_j)$
    - Imply no synchronization cost
    - Reduced to k-Cardinality Bin Packing Problem
- **k-Cardinality Bin Packing Problem**
  - Given : a set of items $a_1, a_2, \cdots, a_n$, each with sizes $s(a_i)$ and the bin capacity $cap$
  - Result : a division of the items into to a minimum number of bins
  - Constraints : each bin contains at most $k$ items and its aggregated size cannot exceed the capacity $cap$

# Cache Capacity Aware Thread Scheduling – Fixed Concurrency (2/2)

- k-Cardinality Bin Packing Algorithms
  - Largest Memory First (LMF) and Iterated Worst-Case Decreasing (IWFD)
    - Constant approximation ratio

**Algorithm 1 : Thread Scheduling for Fixed Concurrency**

```
$k \leftarrow$ maximum possible concurrency
sort $c^n$ in decending order by working set size
repeat
    $cap \leftarrow w(c_1) + w(c_2) + \cdots + w(c_k)$
    $k \leftarrow k - 1$
until $cap \leq Cap\_unified\_L2$
$cap \leftarrow Cap\_unified\_L2$
$s^m \leftarrow$ K-CARDINALITY-BIN-PACKING($c^n, cap, k$)
return $s^m$
```

M. R. Garey, et al., "Worst-Case Analysis of Memory Allocation Algorithms," in *ACM Symp. Theory of Computing*, 1972

K. L. Krause, et al., "Analysis of Several Task-Scheduling Algorithms for a Model of Multiprogramming Computer Systems," *J. ACM*, vol. 22, pp. 522-550, 1975

# Cache Capacity Aware Thread Scheduling – Variable Concurrency (1/2)

- Cost Function: $m + sync\_cost(s^m)$
  - Trade-off between the number of scheduling steps ($m$) and synchronization cost ($sync\_cost(s^m)$)

- Interesting Findings
  - Lemma 2 : For any schedule $s^m$, the overall cost, $m + sync\_cost(s^m)$ is lesser or equal to $2m - 1$
  - Lemma 3 : For any schedule $s^m$, the synchronization cost is minimum if the scheduling steps are sorted by the concurrency ($conc(s_i)$)

# Cache Capacity Aware Thread Scheduling – Variable Concurrency (2/2)

- Algorithm Design
  - Lemma 2 → Minimize the number of steps ($m$)
  - Lemma 3 → Minimize sync. cost ($sync\_cost(s^m)$)

**Algorithm 2 : Thread Scheduling for Variable Concurrency**

Iterative Refinement

```
$k \leftarrow$ maximum possible concurrency
$cap \leftarrow Cap\_unified\_L2$
repeat
    $s^m \leftarrow$ K−CARDINALITY−BIN−PACKING$(c^n, cap, k)$        Lemma 2
    sort $s^m$ by concurrency to minimize synchronization cost
    $old\_cost \leftarrow m + sync\_cost(s^m)$                       Lemma 3
    $k \leftarrow k - 1$
    $s^{m'} \leftarrow$ K−CARDINALITY−BIN−PACKING$(c^n, cap, k)$
    sort $s^{m'}$ by concurrency to minimize synchronization cost
    $new\_cost \leftarrow m + sync\_cost(s^{m'})$
until $new\_cost \geq old\_cost$
return $s^m$
```

# Experimental Results – Experiment Setup (1/2)

- GPGPU-Sim (ISPASS'09) Simulation Setup

| Fermi's Architectural Configurations in GPGPU-Sim | |
|---|---|
| **Number of SMs** | 15 |
| **SM configuration** | 32-wide pipeline, 32 threads/warp, 1536 threads/SM, 32768 registers/SM, **number of CTAs/SM (dynamic reconfigurable, default 8)** |
| **L2 cache** | **unified 768KB**, 8-way, 64 byte/block |
| **DRAM** | 6 GDDR5 channels, 2 chips/channel, 16 banks, 16 entries/chip, FR-FCFS policy |
| **Interconnection network** | single stage butterfly, 32-byte flit size |

- Thread clustering for CTA generation
  - Kuo, et al. (ASPDAC'12)
- Ocelot for working set size analysis
  - Ocelot (PACT'10)

A. Bakhoda, et al., "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *ISPASS*, 2009
H.-K. Kuo, et al., "Thread Affinity Mapping for Irregular Data Access on Shared Cache GPGPU," in *ASPDAC*, 2012
G. F. Diamos, et al., "Ocelot: A Dynamic Optimization Framework for Bulk-Synchronous Applications in Heterogeneous Systems," in *PACT*, 2010

# Experimental Results – Experiment Setup (2/2)

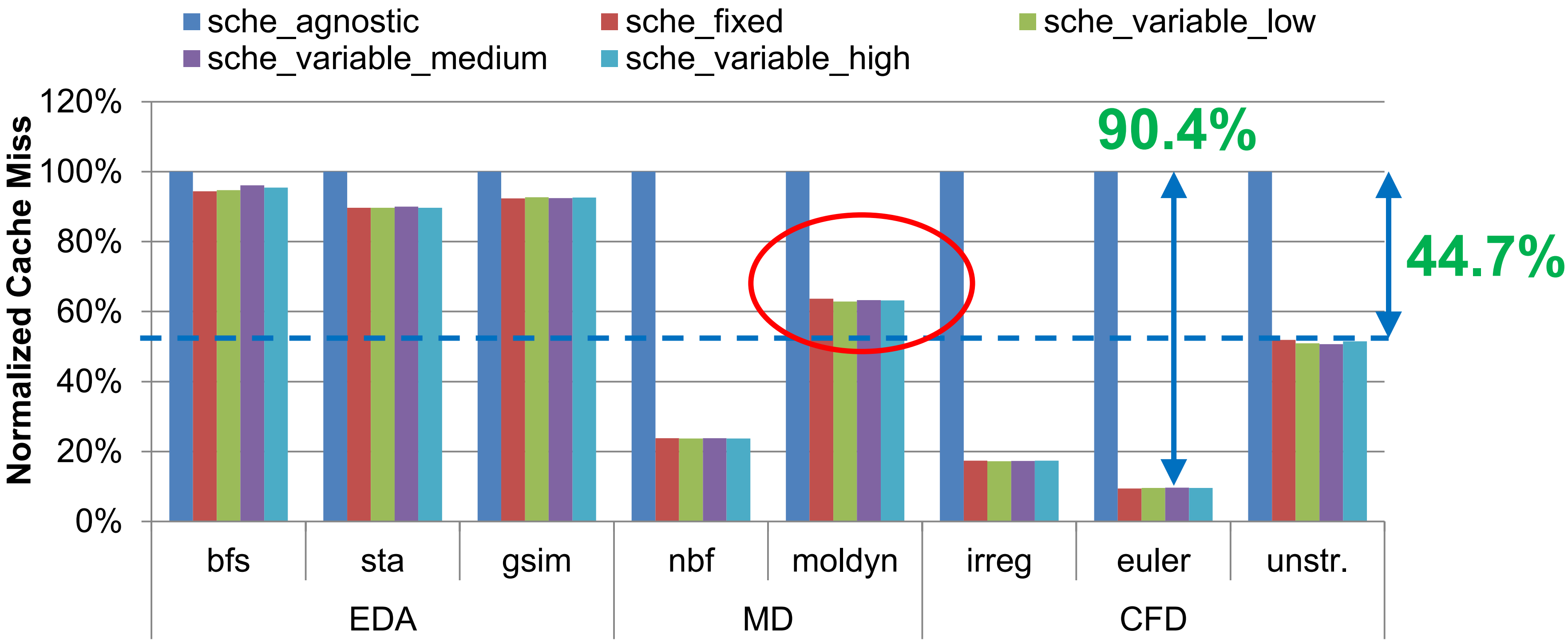
- Application Domains

| Irregular Massive Parallel Applications | | | | |
|---|---|---|---|---|
| **Applications** | **Fields** | **Descriptions** | **Sources** | **Data set sizes** |
| bfs | Electronic Design Automation (EDA) | breadth first search | Kuo, et al. | 2.6 MB |
| sta | | static timing analysis | | 3.0 MB |
| gsim | | gate level logic simulation | | 3.5 MB |
| nbf | Molecular Dynamics (MD) | kernel abstracted from the GROMOS code | Cosmic | 6.3MB |
| moldyn | | force calculation in the CHARMM program | | 10.2MB |
| irreg | Computational Fluid Dynamics (CFD) | kernel of Partial Differential Equation solver | | 6.3MB |
| euler | | finite-difference approximations on mesh | Chaos | 8.5MB |
| unstructured | | fluid dynamics with unstructured mesh | | 10.2MB |

H.-K. Kuo, et al., "Thread Affinity Mapping for Irregular Data Access on Shared Cache GPGPU," in *ASPDAC*, 2012
H. Han, et al., "Exploiting Locality for Irregular Scientific Codes," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, pp. 606-618, 2006
R. Das, et al., "Communication Optimizations for Irregular Scientific Computations on Distributed Memory Architectures," *J. Parallel Distrib. Comput.*, vol. 22, pp. 462-478, 1994.

# Experimental Results – Cache Misses Reduction

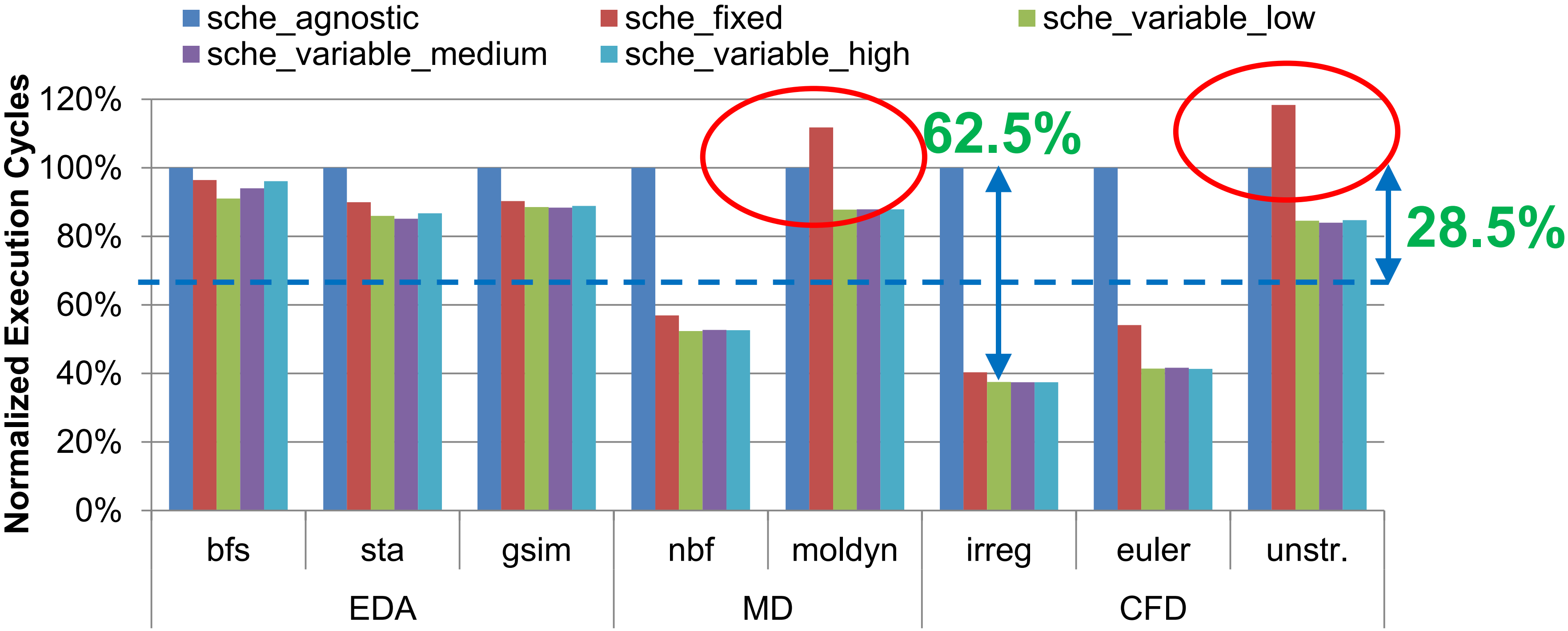
- sche_agnostic, sche_fixed and sche_variable
  - cps : low (50 cycles), medium (100 cycles) and high (200 cycles)



W.-C. Feng , et al., "To GPU Synchronize or not GPU Synchronize?," in *ISCAS*, 2010

# Experimental Results – Execution Time Improvement

- sche_fixed
  - Too restrictive to schedule more concurrent CTAs (moldyn and unstructured)



# Conclusions

- This paper
  - Formulate a general thread scheduling problem, Cache Capacity Aware Thread Scheduling Problem
  - Not only prove the NP-hardness, but also propose two thread scheduling algorithms
  - Achieve an average of
    - **44.7%** cache misses reduction
    - **28.5%** runtime enhancement
      - Up to **62.5%** for applications with more threads and higher complexity

# THANK YOU
# FOR YOUR ATTENTION

WE WELCOME YOUR QUESTIONS, COMMENTS AND SUGGESTIONS

Hsien-Kai Kuo
hkkuo[at]ee.eda.nctu.edu.tw