Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik

Xulong Tang, Adwait Jog, Onur Kayıran, Asit Mishra,

Mahmut Kandemir, Onur Mutlu, Chita Das

PACT ‘16

Era of Energy-Efficient Architectures

Peak Performance increased by ~27x in past 6 years

Energy Efficiency increased by ~7x in past 6 years

Future: 1 ExaFlops/s at 20 MW Peak power

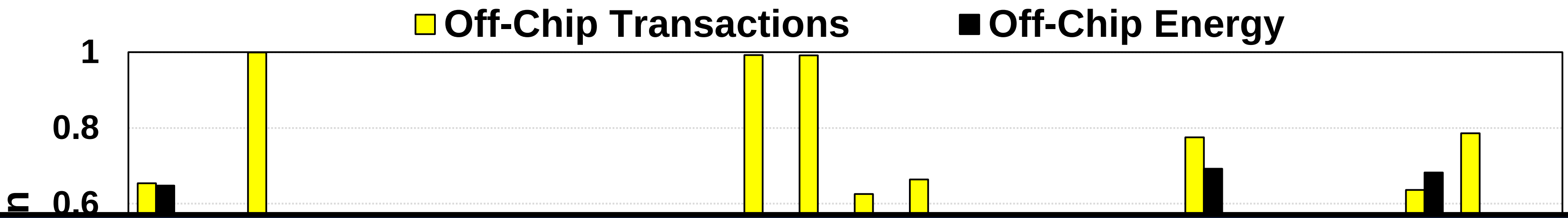
- **Greatly need to improve energy efficiency as well as performance!**

2010:	2013:	2016:
Tianhe-1A	Tianhe-2	Sunway TaihuLight
4.7 PFlop/s, 4 MW	54.9 PFlop/s, 17.8 MW	125.4 PFlop/s, 15.4 MW
~1.175 TFlops/W	~3.084 TFlops/W	~8.143 TFlops/W

Bottleneck

- Continuous energy-efficiency and performance scaling is not easy.
- Energy consumed by a floating-point operation is scaling down with technology scaling.
- Energy consumption due to data transfer overhead is not scaling down!

Bottleneck

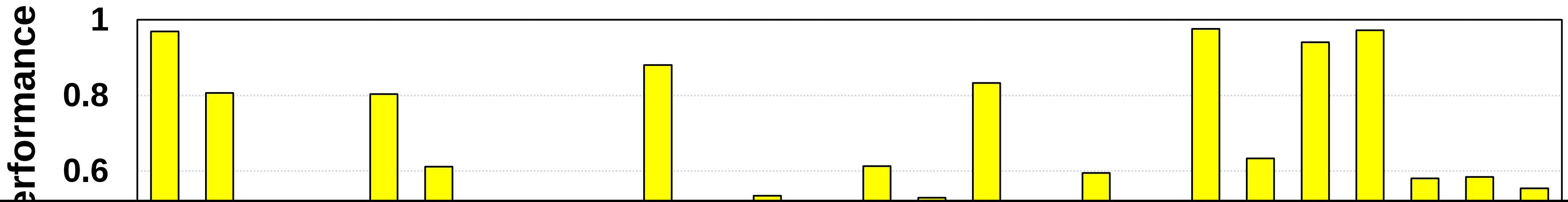


Across these 25 GPGPU applications:

- **49% of all transactions are off-chip.**
- **This is responsible for 41% of total energy consumption of the system.**

Data movement and system energy consumption caused by off-chip memory accesses.

Bottleneck



Main memory accesses lead to 45% performance degradation!

Performance normalized to a hypothetical GPU where all the off-chip accesses hit in the last-level cache.

Outline

- Introduction and Motivation
- Background and Challenges
- Design of Kernel Offloading Mechanism
- Design of Concurrent Kernel Management
- Simulation Setup and Evaluation
- Conclusions

Revisiting Processing-In-Memory (PIM)

- It’s a promising approach to minimize data movement.
- The concept dates back to the late 1960s
- Technological limitations of integrating fast computational units in memory was a challenge

- Significant advances in adoption of 3D-stacked memory has
  - enabled tight integration of memory dies and logic layer
  - brought computational units into the memory stack

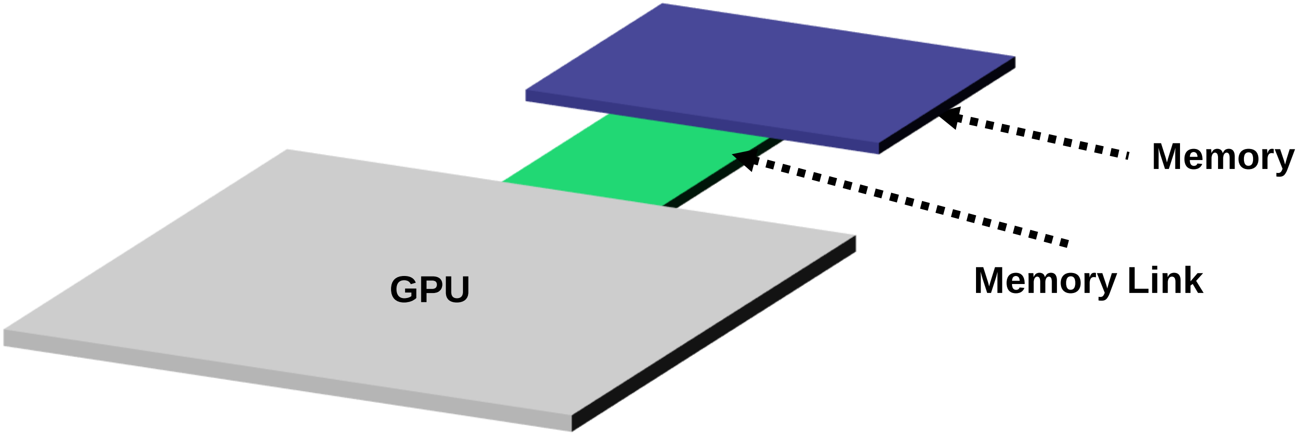
PIM-Assisted GPU architecture

- We integrate PIM units to a GPU based system and we call this as "*PIM-Assisted GPU architecture*".

- At least one 3D-stacked memory is integrated with PIM units and is placed adjacent to a traditional GPU design.

PIM-Assisted GPU architecture

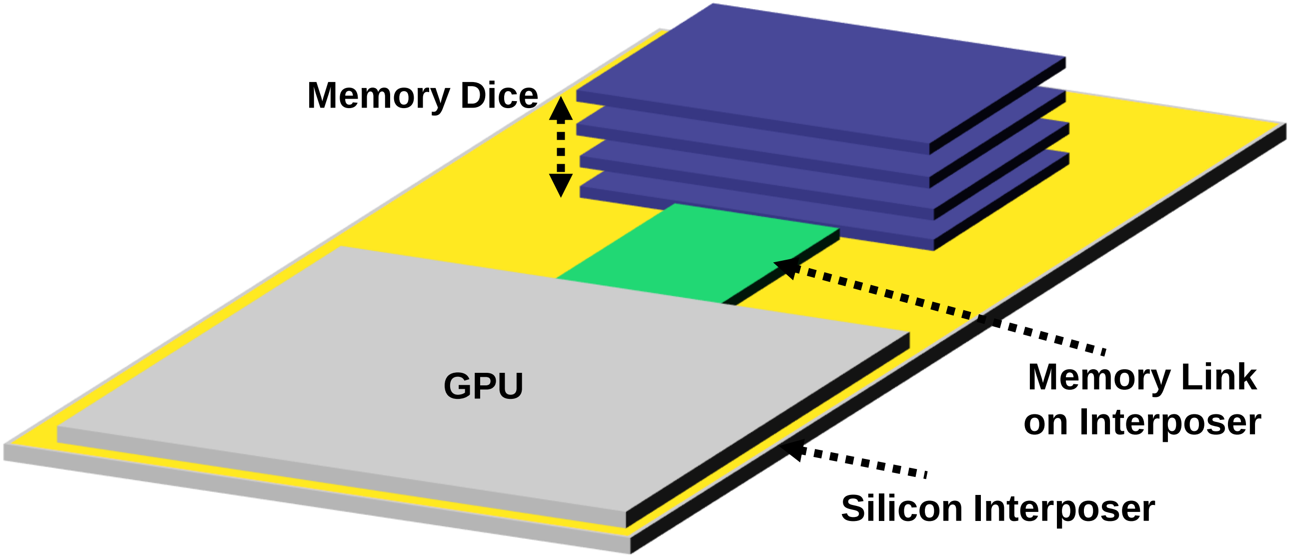
- Traditional GPU architecture*



* Only a single DRAM partition is shown for illustration purposes

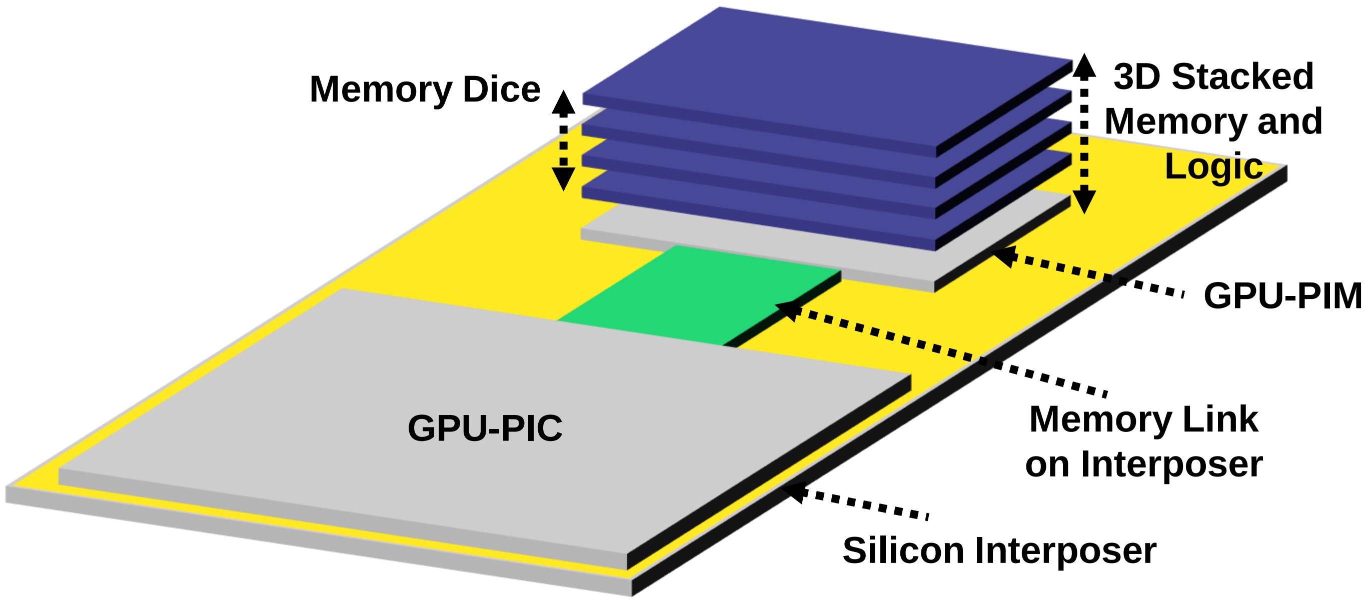
PIM-Assisted GPU architecture

- GPU architecture with 3D-stacked memory on a silicon interposer



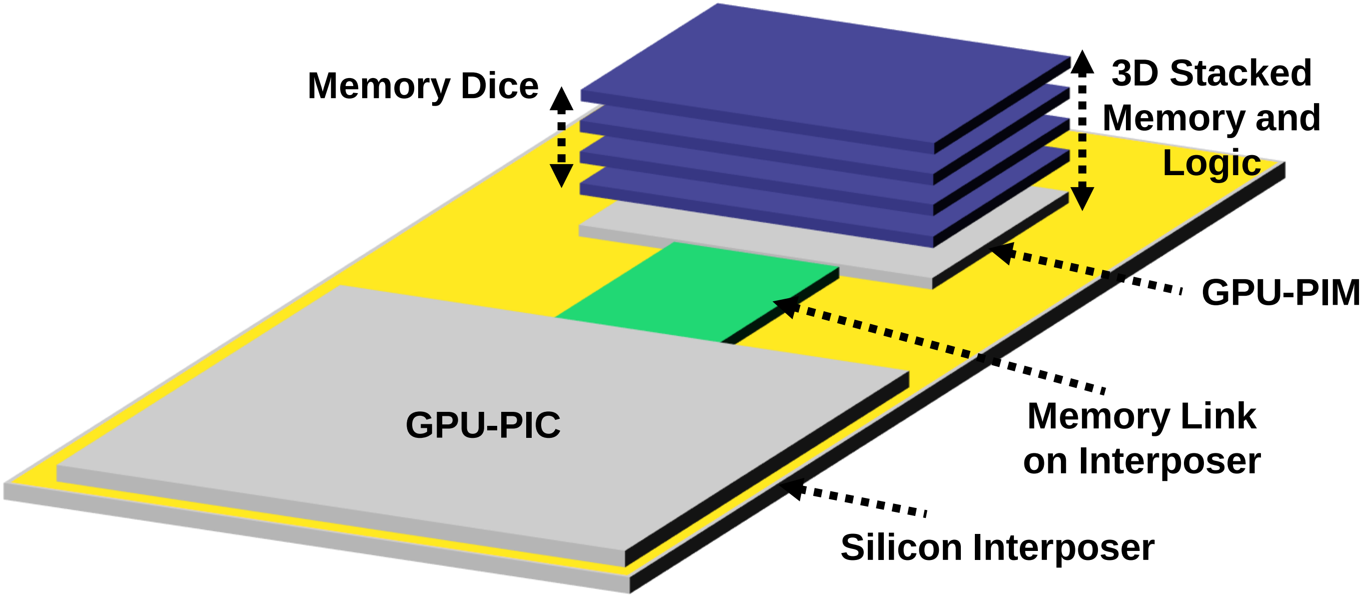
PIM-Assisted GPU architecture

- Now we add a logic layer to the 3D-stacked memory and we call this logic layer as GPU-PIM.
- The traditional GPU logic is now called GPU-PIC.



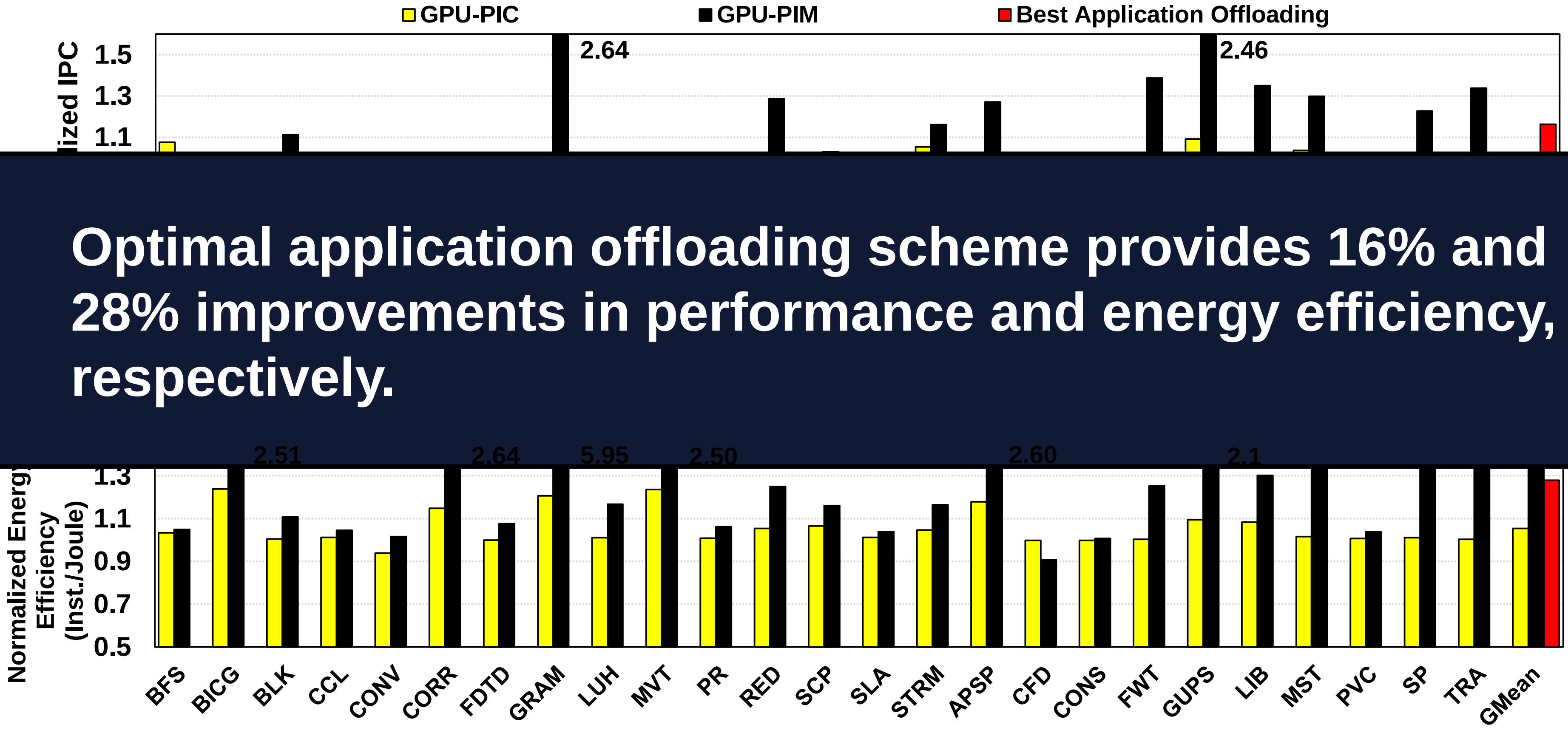
PIM-Assisted GPU architecture

- Application can now be run on both GPU-PIC and GPU-PIM
- Challenge: Where to execute the application on?



Application Offloading

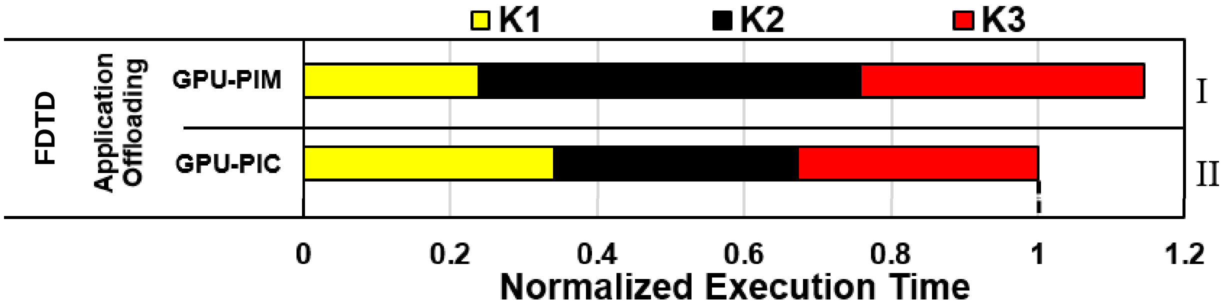
- We evaluate application execution on either GPU-PIC or GPU-PIM



Optimal application offloading scheme provides 16% and 28% improvements in performance and energy efficiency, respectively.

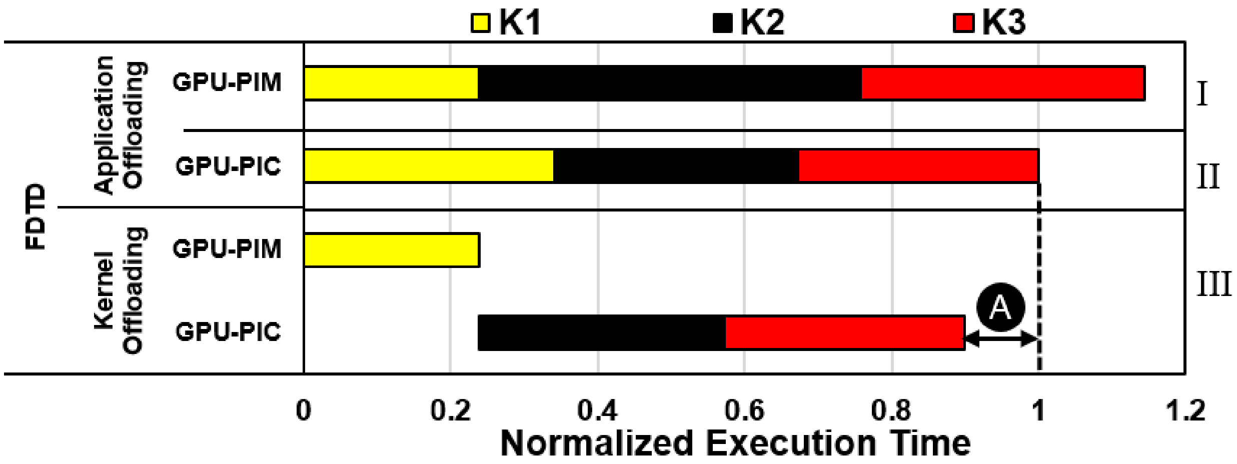
Limitations of Application Offloading

- Limitation 1: Lack of Fine-Grained Offloading



Limitations of Application Offloading

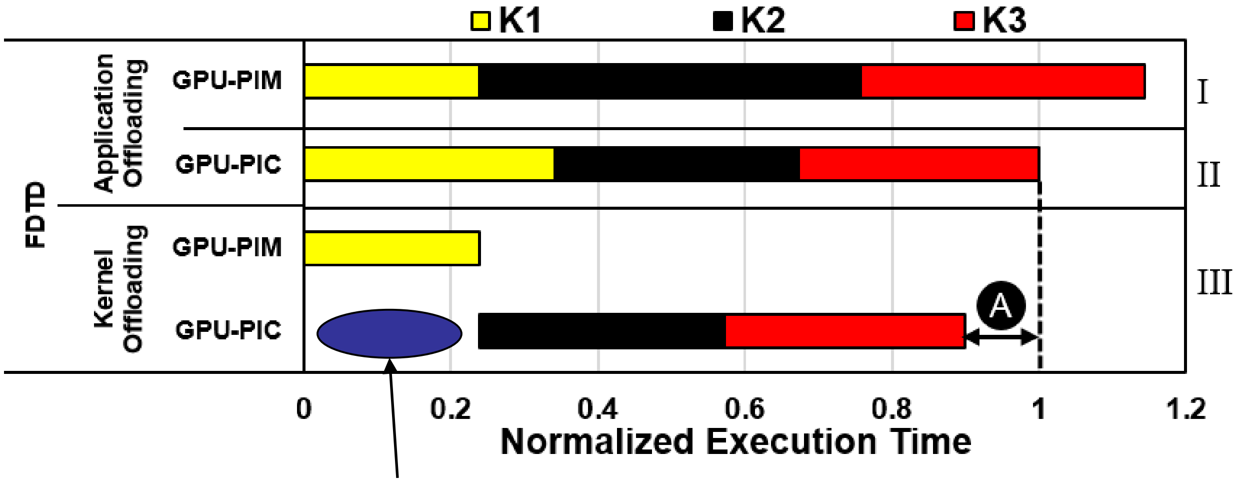
- Limitation 1: Lack of Fine-Grained Offloading



Running K1 on GPU-PIM, and K2 and K3 on GPU-PIC provides the optimal kernel placement for improved performance.

Limitations of Application Offloading

- Limitation 1: Lack of Fine-Grained Offloading
- Limitation 2: Lack of Concurrent Utilization of GPU-PIM and GPU-PIC

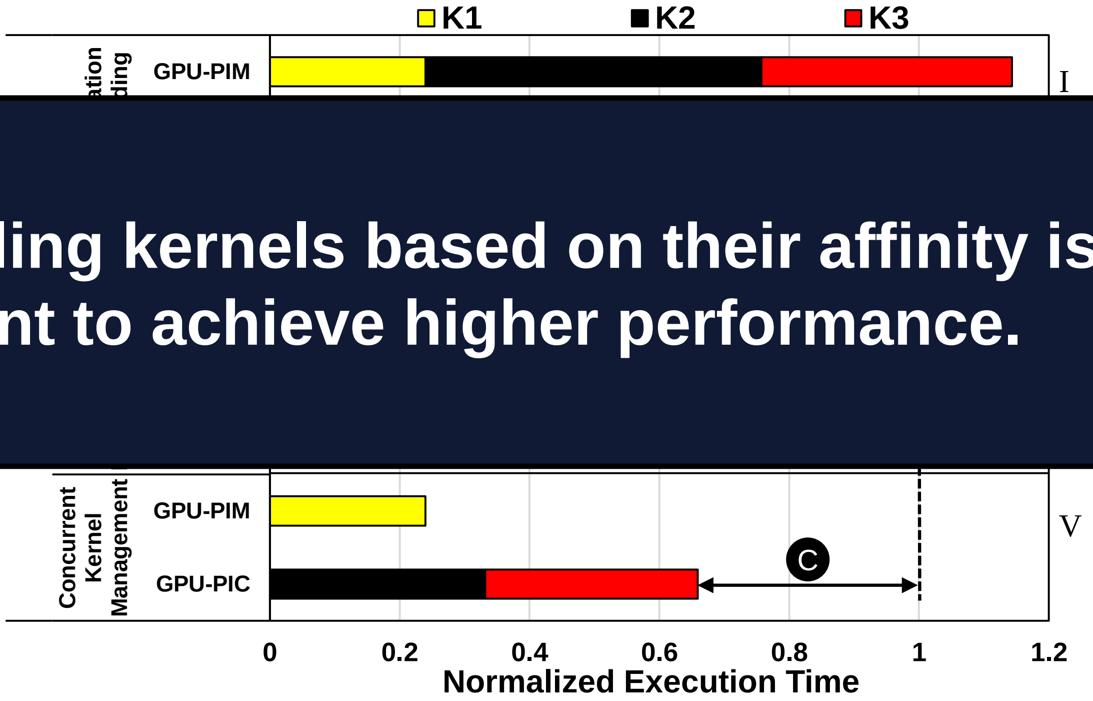


GPU-PIC is idle!

- From the application we find that kernel K1 and K2 are independent from each other.

Limitations of Application Offloading

- Limitation 1: Lack of Fine-Grained Offloading
- Limitation 2: Lack of Concurrent Utilization of GPU-PIM and GPU-PIC



Scheduling kernels based on their affinity is very important to achieve higher performance.

K1 -> GPU-PIM
K2 -> GPU-PIC

Our Goal

To develop runtime mechanisms for

- automatically identifying architecture affinity of each kernel in an application
- scheduling kernels on GPU-PIC and GPU-PIM to maximize for performance and utilization

Outline

- Introduction and Motivation
- Background and Challenges
- Design of Kernel Offloading Mechanism
- Design of Concurrent Kernel Management
- Simulation Setup and Evaluation
- Conclusions

Design of Kernel Offloading Mechanism

- Goal: Offload kernels to either GPU-PIC or GPU-PIM to maximize performance
- Challenge: Need to know the architecture affinity of the kernels
- We build an architecture affinity prediction model

Design of Kernel Offloading Mechanism

- Metrics used to predict compute engine affinity and GPU-PIC and GPU-PIM execution time.

Category	Predictive Metric	Static/Dynamic
I: Memory Intensity of Kernel	Memory to Compute Ratio	Static
	Number of Compute Inst.	Static
	Number of Memory Inst.	Static
II: Available Parallelism in the Kernel	Number of CTAs	Dynamic
	Total Number of Threads	Dynamic
	Number of Thread Inst.	Dynamic
III: Shared Memory Intensity of Kernel	Total Number of Shared Memory Inst.	Static

Design of Kernel Offloading Mechanism

- Logistic Regression Model for Affinity Prediction

$$\sigma(t) = \frac{e^t}{e^t + 1}$$

where:

$\sigma(t)$ = model output ($\sigma(t)$ < 0.5 => GPU-PIC, $\sigma(t)$ ≥ 0.5 => GPU-PIM)

t = $\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 + \alpha_5 x_5 + \alpha_6 x_6 + \alpha_7 x_7$

α_i = Coefficients of the Regression Model

x_i = Predictive Metrics

Design of Kernel Offloading Mechanism

- Training Set: we randomly sample 60% (15) of the 25 GPGPU applications considered in the paper.
- These 15 applications consists of 82 unique kernels that are used for training the affinity prediction model.

- Test Set: the remaining 40% (10) of the applications are used as the test set for the model

- Accuracy of the model on the test set: 83%

Outline

- Introduction and Motivation
- Background and Challenges
- Design of Kernel Offloading Mechanism
- Design of Concurrent Kernel Management
- Simulation Setup and Evaluation
- Conclusions

Design of Concurrent Kernel Management

- Goal: Efficiently manage the scheduling of concurrent kernels to improve performance and utilization of the PIM-Assisted GPU architecture

- For efficiently managing kernel execution on both GPU-PIM and GPU-PIC, we need
  - Kernel-level Dependence Information
  - Architecture Affinity Information
  - Execution Time Information

Design of Concurrent Kernel Management

- For efficiently managing kernel execution on both GPU-PIM and GPU-PIC, we need
  - Kernel-level Dependence Information
    - Obtained through exhaustive analysis to find RAW dependence for all considered applications and input pairs
  - Architecture Affinity Information
  - Execution Time Information

Design of Concurrent Kernel Management

- For efficiently managing kernel execution on both GPU-PIM and GPU-PIC, we need
  - Kernel-level Dependence Information
    - Obtained through exhaustive analysis to find RAW dependence for all considered applications and input pairs
  - Architecture Affinity Information
    - Utilizes the affinity prediction model built for kernel offloading mechanism
  - Execution Time Information

Design of Concurrent Kernel Management

- For efficiently managing kernel execution on both GPU-PIM and GPU-PIC, we need
  - Kernel-level Dependence Information
    - Obtained through exhaustive analysis to find RAW dependence for all considered applications and input pairs
  - Architecture Affinity Information
    - Utilizes the affinity prediction model built for kernel offloading mechanism
  - Execution Time Information
    - We build linear regression models for execution time prediction on GPU-PIC and GPU-PIM
    - We use the same “Predictive metrics” and training set used for affinity prediction model

Design of Concurrent Kernel Management

- Linear Regression Model for Execution Time Prediction Model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7$$

where:

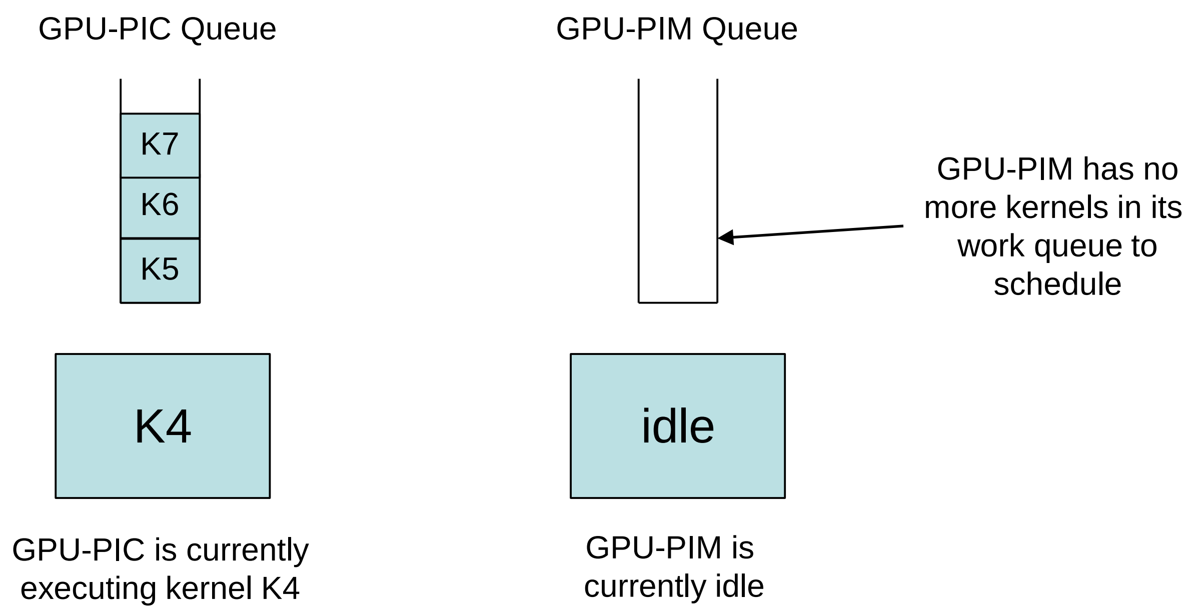
y = model output (predicted execution time)

β_i = Coefficients of the Regression Model

x_i = Predictive Metrics

Design of Concurrent Kernel Management

- Lets run through an example



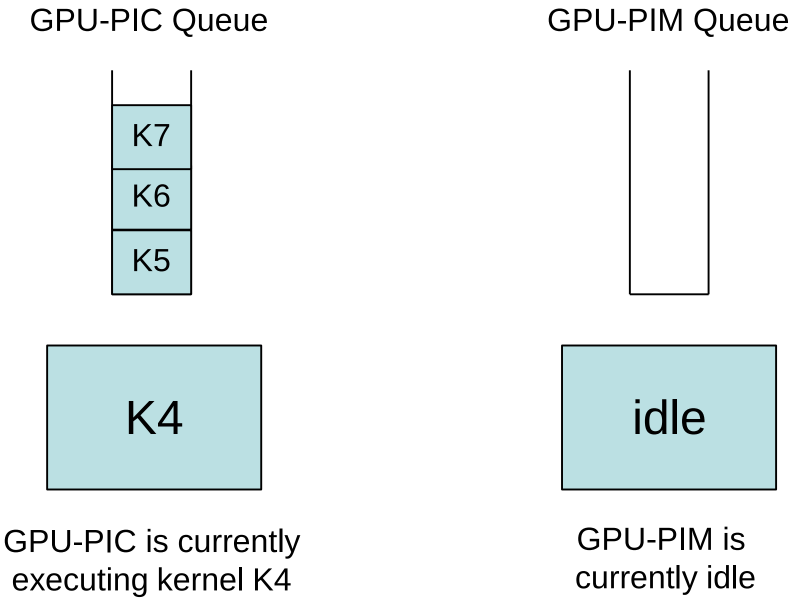
GPU-PIM has no more kernels in its work queue to schedule

GPU-PIC is currently executing kernel K4

GPU-PIM is currently idle

Design of Concurrent Kernel Management

- We can potentially pick any kernel (assuming no data dependence among themselves and K4) from GPU-PIC Queue and schedule them onto GPU-PIM


GPU-PIC Queue
GPU-PIM Queue
K7
K6
K5
K4
idle
GPU-PIC is currently executing kernel K4
GPU-PIM is currently idle

- But which one to pick?

Design of Concurrent Kernel Management

- We steal the first kernel that satisfies a given condition and schedule it on to GPU-PIM Queue.

- Pseudocode:
- *time(kernel, compute_engine)* returns the estimated execution time of “kernel” when executed on ”compute_engine”

Estimated execution time of currently executing kernel K4 on GPU-PIC

```
for X in GPU-PIC’s Queue
  if $(time\ (X, GPU - PIM) \leq \{\ time(K4, GPU - PIC) - time_{executed}(K4) + time(X, GPU - PIC)\}$
      $steal\ and\ schedule\ X\ to\ GPU - PIM;$
      $break;$
```

Outline

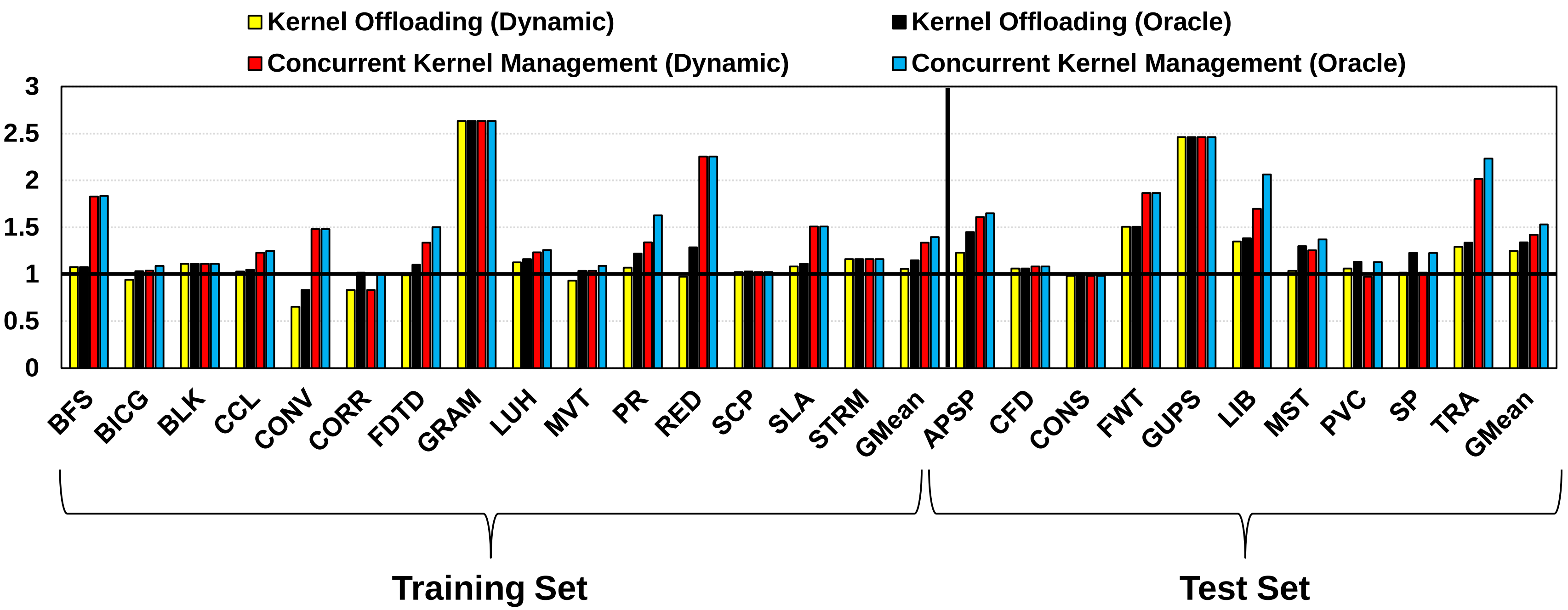
- Introduction and Motivation
- Background and Challenges
- Design of Kernel Offloading Mechanism
- Design of Concurrent Kernel Management
- Simulation Setup and Evaluation
- Conclusions

Simulation Setup

- Evaluated on GPGPU-Sim, a cycle accurate GPU simulator
- Baseline configuration
  - 40 SMs, 32-SIMT lanes, 32-threads/warp
  - 768 kB L2 cache
- GPU-PIM configuration
  - 8 SMs, 32-SIMT lanes, 32-threads/warp
  - No L2 cache
- GPU-PIC configuration
  - 32 SMs, 32-SIMT lanes, 32-threads/warp
  - 768 kB L2 cache

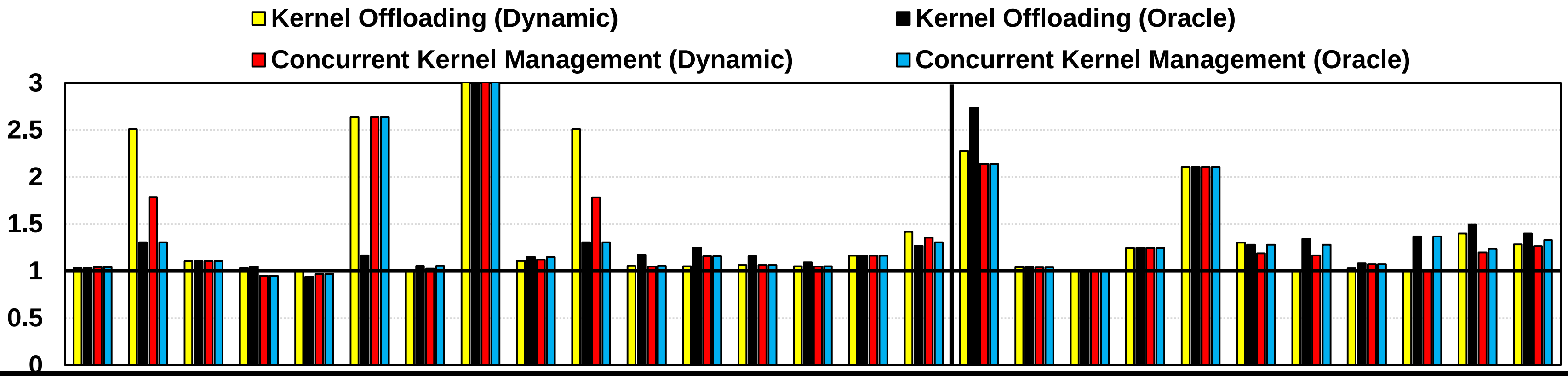
- 25 GPGPU Applications classified into 2 exclusive sets
  - Training Set: The kernels are used as input to build the regression models
  - Test Set: The regression models are only tested on these kernels

Performance (Normalized to Baseline)



- Performance improvement for Test Set applications
    - Kernel Offloading = 25%
    - Concurrent Kernel Management = 42%

Energy-Efficiency (Normalized to Baseline)



More results and detailed description of our runtime mechanisms are in the paper.

- Energy-Efficiency improvement for Test Set applications
  - Kernel Offloading = 28%
  - Concurrent Kernel Management = 27%

Conclusions

- Processing-In-Memory is a key direction in achieving high performance with lower power budget.
- Simply offloading applications completely onto PIM units is not optimal.
- For effective utilization of PIM-Assisted GPU architecture, we need to
  - Identify code segments for offloading onto GPU-PIM
  - Efficiently distribute work between GPU-PIC and GPU-PIM
- Our kernel-level scheduling mechanisms can be an effective runtime solution for exploiting processing-in-memory in modern GPU-based architectures.

Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayıran,
Asit Mishra, Mahmut Kandemir, Onur Mutlu, Chita Das.

PACT '16

PennState WILLIAM & MARY AMD ETH zürich Carnegie Mellon University intel