Towards Energy-Efficient Real-Time Scheduling of Heterogeneous Multi-GPU Systems

Yidi Wang, Mohsen Karimi and Hyoseung Kim

University of California, Riverside

IEEE Real-Time Systems Symposium (RTSS) 2022

Motivation

- In a multi-GPU system, workload allocation methods can be categorized to:
  - Load distribution
    - Idle energy consumption from computing units causes energy inefficiency
  - Load concentration
    - Different tasks have different *energy-preferred* GPU

- The problem is more complicated in a real-time system
  - Real-time tasks have different arriving patterns with different timing constraints

Related Work

- Real-time GPU Scheduling
  - Temporal multitasking[1, 2, 3]: focus on the time-sharing of the GPU
    - Poor energy efficiency and lack of support for heterogeneous GPUs
  - Spatial multitasking[4]
    - No consideration of energy efficiency as well as multi-GPUs
- GPU Energy Efficiency[5, 6, 7]
  - Focuses on regulating the number of active SMs
    - Problem: SM-level power gating is not yet available in today's GPUs
- Our previous work – sBEET framework[8]
  - Combines spatial and temporal multitasking to balance energy consumption and schedulability
    - We extend this work to a heterogeneous multi-GPU system through offline task allocation and runtime job migration

[1] G. Elliott and J. Anderson. Globally scheduled real-time multiprocessor systems with GPUs. *Real-Time Systems, 48:34–74*, 05 2012
[2] H. Kim, P. Patel, S. Wang, and R. Rajkumar. A server-based approach for predictable GPU access control. *RTCSA*, 2017
[3] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A responsive GPGPU execution model for runtime engines. *RTSS*, 2011
[4] S. K. Saha, Y. Xiang, and H. Kim. STGM: Spatio-temporal GPU management for real-time tasks. *RTCSA*, 2019
[5] P. Aguilera, K. Morrow, and N. S. Kim, "QoS-aware dynamic resource allocation for spatial-multitasking GPUs," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014
[6] Z.-G. Tasoulas and I. Anagnostopoulos, "Improving GPU performance with a power-aware streaming multiprocessor allocation methodology," *Electronics*, vol. 8, no. 12, 2019.
[7] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng. Power gating strategies on GPUs. *TACO*, 2011
[8] Y. Wang, M. Karimi, Y. Xiang, and H. Kim, "Balancing energy efficiency and real-time performance in GPU scheduling," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021

Contributions

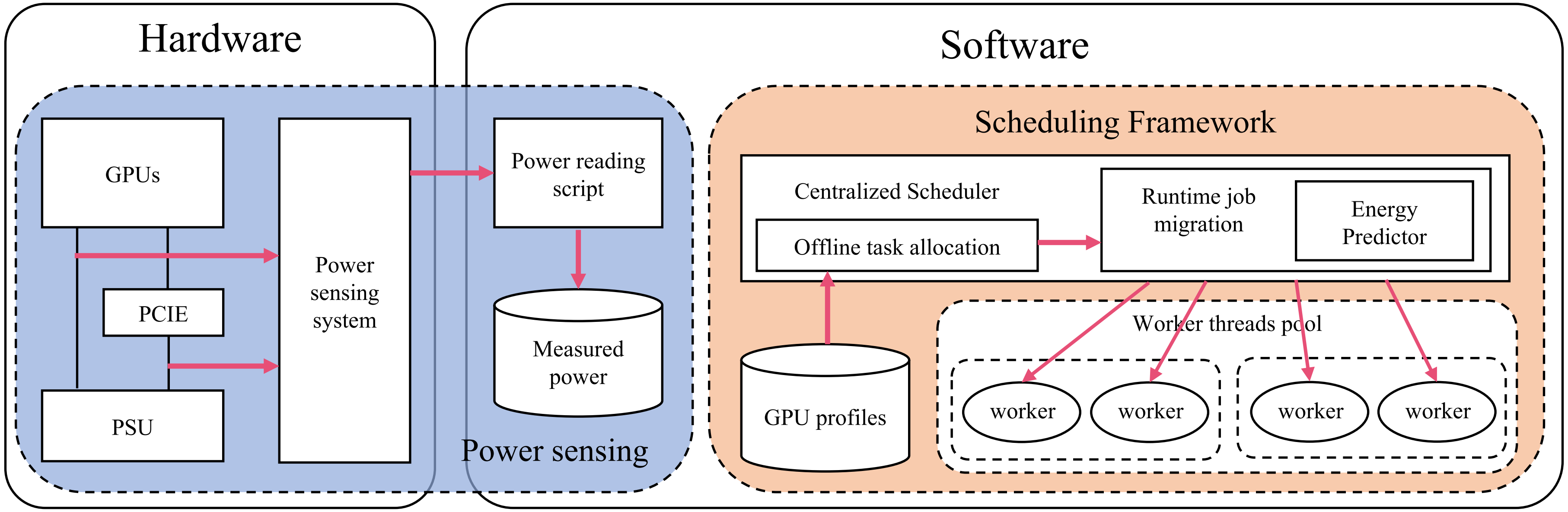
We propose sBEET-mg:

✓ An energy-efficient real-time GPU scheduling framework for heterogeneous multi-GPU systems

- Analyzed the power usage characteristics on a multi-GPU system with our customized power monitoring tool
- Proposed a framework to address the timeliness and energy efficiency simultaneously in a heterogeneous multi-GPU environment
- Developed a custom power monitoring tool that obtains precise power measurements
- The proposed work outperforms the conventional load concentration and distribution approaches in both real hardware and simulation

Proposed Work Overview

- Custom power sensing tool
- Scheduling framework
  - Centralized scheduler (offline task allocation + runtime job migration)
    - One single CUDA context
  - Two worker threads dedicated for each GPU



System Model

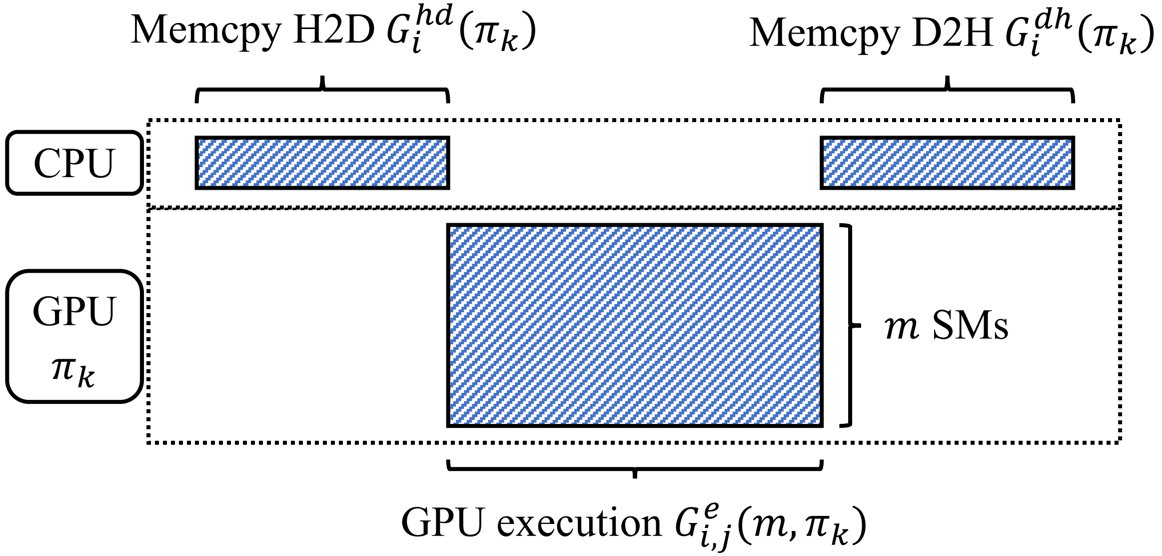
- Platform Model
  - A single-ISA system $\boldsymbol{\Pi}$ consisting with $\boldsymbol{\omega}$ heterogeneous GPUs
  - A GPU $\boldsymbol{\pi_k}$ containing $\boldsymbol{M_k}$ SMs
- Task Model
  - A taskset $\boldsymbol{\Gamma}$ consists of $\boldsymbol{n}$ periodic GPU tasks:
    - Non-preemptive
    - W/ Constrained deadlines

      $\tau_i \coloneqq (G_i,\ T_i,\ D_i)$

      WCET, period, deadline
  - Each task τ_i consists of a sequence of jobs $J_{i,j}$
  - Each job can execute with a different number of SMs on a different GPU

WCET of a job $J_{i,j}$:

$$G_{i,j}(m, \pi_k) = G_i^{hd}(\pi_k) + G_{i,j}^{e}(m, \pi_k) + G_i^{dh}(\pi_k)$$



Power and Energy Model

- Power model
  - Power model: $P = P^s + P^d + P^{idle}$
  - For a set of jobs J = $\{J_1,\ J_2, \dots,\ J_n\}$:

$$P = P^s + \sum_{i=1}^{n} P_i^d(m_i) + P^{idle}\left(M - \sum_{i=1}^{n} m_i\right)$$

  - For a taskset Γ, energy consumption in [t1, t2]:

$$E_k(t_1, t_2) = \int_{t_1}^{t_2} \left(P_k^s + \sum_{J_i \in J} \left(P_{k,i}^d \left(\sum_{m=1}^{M_k} x_i^m(t) \right) \right) + P_k^{idle} \left(M_k - \sum_{J_i \in J} \sum_{m=1}^{M_k} x_i^m(t) \right) \right) dt$$

  - Energy consumption of all GPUs:

$$E([t_1, t_2]) = \sum_{\forall \pi_k \in \Pi} E_k([t_1, t_2])$$

$$x_i^m(t) = \begin{cases} 0, \tau_i \text{ is not active on } SM_k \\ 1, \tau_i \text{ is active on } SM_k \end{cases}$$

Insights on Conventional Approaches (1)

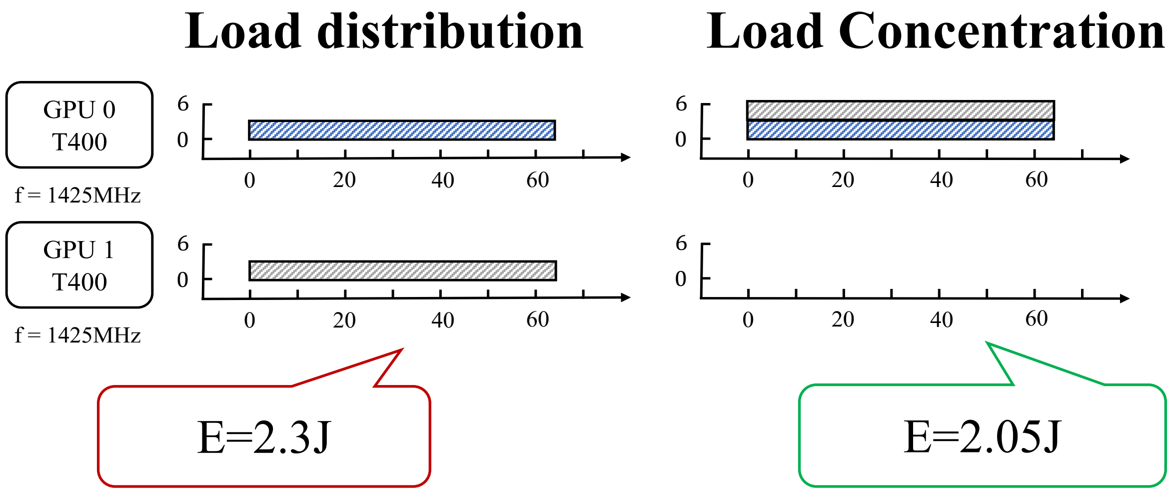
- Baseline Scheduling Approaches
  - **Load Concentration**
    - It assigns a GPU job to the most packed GPU
  - **Load Distribution**
    - It chooses an idling GPU first (or a GPU with the highest number of idling SMs)

Insights on Conventional Approaches (2)

- Homogeneous GPUs
  - Example 1

Table III: Taskset in Examples 1 and 2

Task	Application	$G_i^e(\pi_0, 6)$	$G_i^e(\pi_0, 4)$	$G_i^e(\pi_0, 3)$	$G_i^e(\pi_0, 2)$
$\tau_1 = \tau_2$	Histogram	32.67 ms	47.95 ms	63.724 ms	95.53 ms



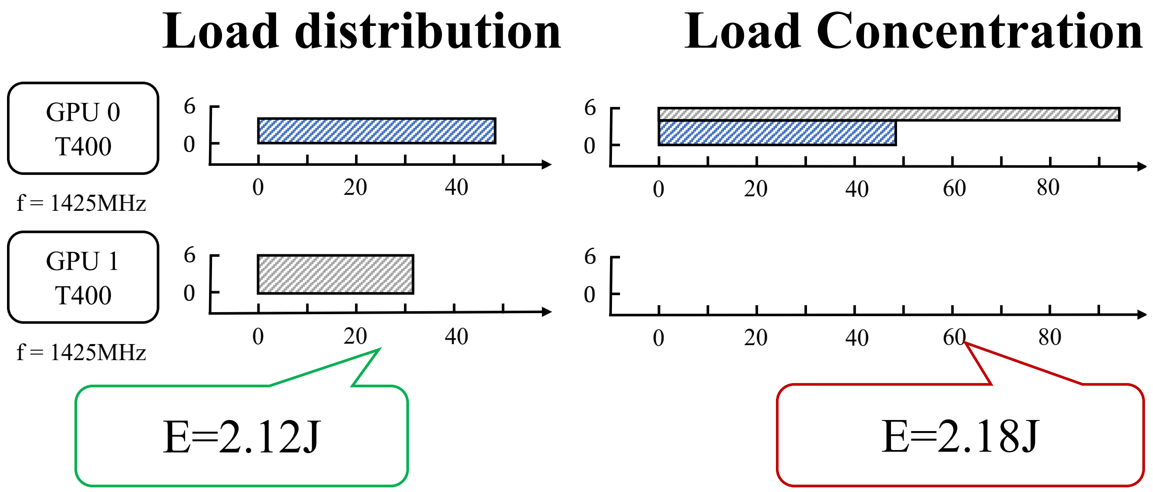
Load concentration is better in this case

Insights on Conventional Approaches (3)

- Homogeneous GPUs
  - Example 2
    - Same taskset, but τ_1 executes slightly earlier with 4 SMs

Table III: Taskset in Examples 1 and 2

Task	Application	$G_i^e(\pi_0, 6)$	$G_i^e(\pi_0, 4)$	$G_i^e(\pi_0, 3)$	$G_i^e(\pi_0, 2)$
$\tau_1 = \tau_2$	Histogram	32.67 ms	47.95 ms	63.724 ms	95.53 ms



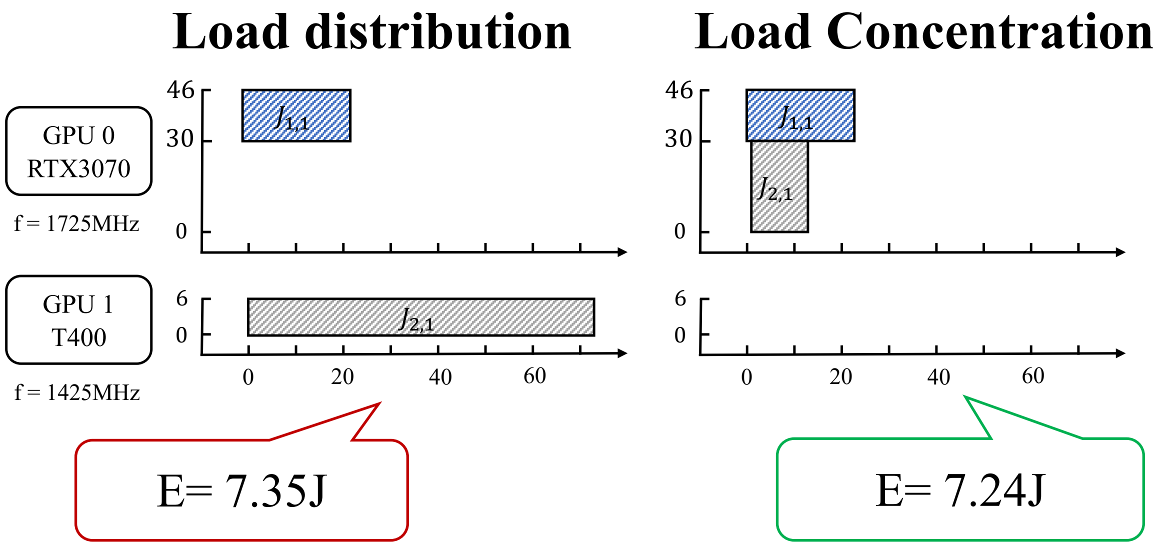
A small difference made load distribution the winner

Insights on Conventional Approaches (4)

- Heterogeneous GPUs
  - Example 1

Table IV: Taskset in Example $\boxed{3}$ and $\boxed{4}$

Task	Application	$G_i^e(30, \pi_0)$	$G_i^e(16, \pi_0)$	$G_i^e(6, \pi_1)$
τ_1	MatrixMul	11.98 ms	21.55 ms	-
τ_2	Hotspot	12.00 ms	22.31 ms	73.188 ms

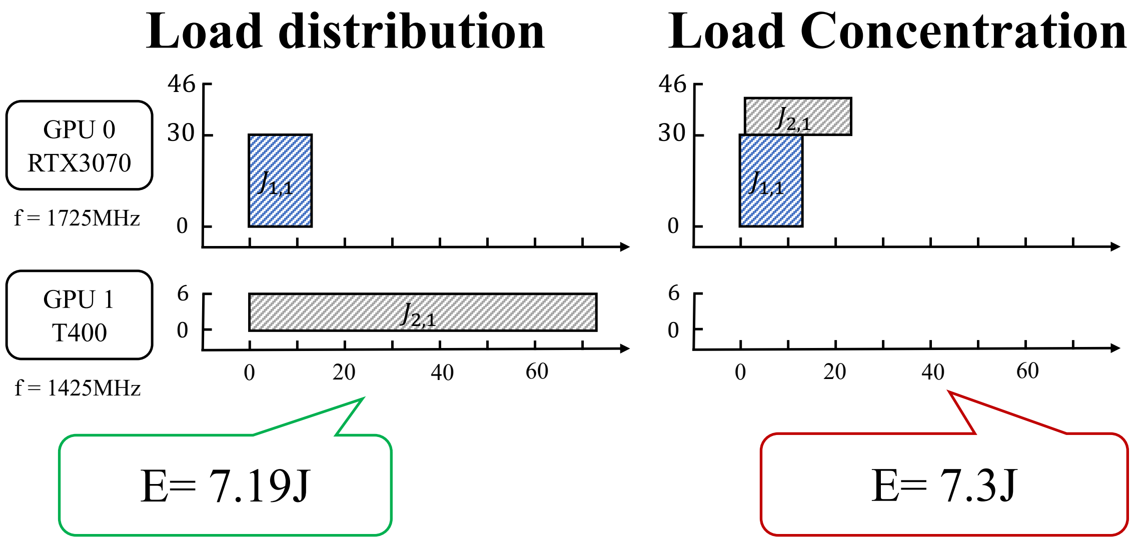


Insights on Conventional Approaches (5)

- Heterogeneous GPUs
  - Example 2

Table IV: Taskset in Example 3 and 4

Task	Application	$G_i^e(30, \pi_0)$	$G_i^e(16, \pi_0)$	$G_i^e(6, \pi_1)$
τ_1	MatrixMul	11.98 ms	21.55 ms	-
τ_2	Hotspot	12.00 ms	22.31 ms	73.188 ms



Insights on Conventional Approaches (6)

- To improve energy efficiency…
  - **Neither approaches should be preferred** regardless of whether the GPUs are homogeneous or not
  - If we can make all tasks on the same GPU finish at similar time, active-idle power consumption of unused SMs can be minimized
  - **However, it is hard to realize with real-time tasks** since they have different arrival patterns and timing constraints

Energy-Efficient Multi-GPU Scheduling (1)

- Energy Optimality:

  - **Definition 1. (*Energy optimal SMs*)** The energy-optimal number of SMs $m_{k,i}^{opt}$, for a task τ_i on a GPU π_k is defined as the number of SMs that leads to the lowest energy consumption when it executes in isolation on the GPU during an arbitrary time interval.

  - **Definition 2. (*Energy preferred GPU*)** The energy-preferred GPU for a task τ_i in a multi-GPU system Π is the GPU that consumes the least amount of energy when τ_i executes with $m_{k,i}^{opt}$ SMs on it.

$$\underset{\pi_k \in \Pi}{\operatorname{argmin}} \int_0^{\delta} P_k^s + P_{k,i}^d(m_{k,i}^{opt}) + P_k^{idle}(M_k - m_{k,i}^{opt})dt$$

Energy-Efficient Multi-GPU Scheduling (2)

- sBEET-mg Overview:
  - Adaptively chooses the GPU and SM configuration of each job of real-time GPU tasks such that it brings the minimum expected energy consumption to all GPUs in the system

- Approach:
  - An offline task distribution algorithm
    - ➢ As a guideline for the runtime scheduler
  - A heuristic runtime scheduler
    - ➢ Two worker threads per GPU to enable parallel execution of jobs
    - ➢ Decides whether to execute a job on the preassigned GPU or migrate it to another GPU

Energy-Efficient Multi-GPU Scheduling (3)

- Offline Task Distribution:
  - Main idea: For each task, the algorithm tries to assign it to the energy-preferred GPU
  - Step 1: Sort all tasks in the decreasing order of priority
  - Step 2: For each task, it obtains a list of GPUs in an order of energy-preference
  - Step 3: Simple utilization check for admission
  - Step 3: Assign the unassigned tasks in Step 3 to the GPUs that will have the minimum utilization

Algorithm 1 Offline Task Distribution

procedure TASK DISTRIBUTION
    Sort tasks in Γ in decreasing order of priority
    for $\tau_i \in \Gamma$ **do**
        Get a list Π_i of GPUs in non-increasing order of expected energy consumption for τ_i
        for $\pi_k \in \Pi_i$ **do**
            if $U(\pi_k) + U_i(\pi_k, m_{k,i}^{opt}) \leq 1$ **then**
                Assign τ_i to π_k
                break
            end if
        end for
        if τ_i is not assigned **then**
            Assign τ_i to the GPU that has a minimum utilization after τ_i is assigned
        end if
    end for
end procedure

Energy-Efficient Multi-GPU Scheduling (4)

- Runtime Job Migration:
  - Main idea: Migrate and pack jobs at runtime to further reduce energy consumption since the GPUs are not SM-level power-gated
  - Decide at runtime:
    - Consider the energy consumption of a given job on each GPU
    - Choose the one that can meet all deadlines with the minimum predicted energy consumption
    - If no GPU can meet the deadline, select the one with the minimum energy consumption

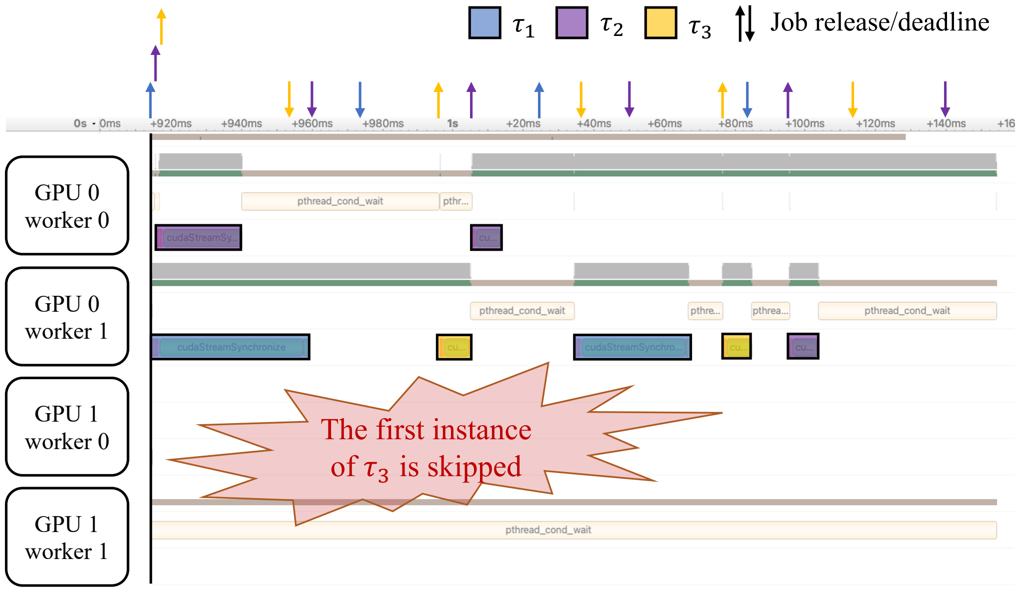
Energy-Efficient Multi-GPU Scheduling (5)

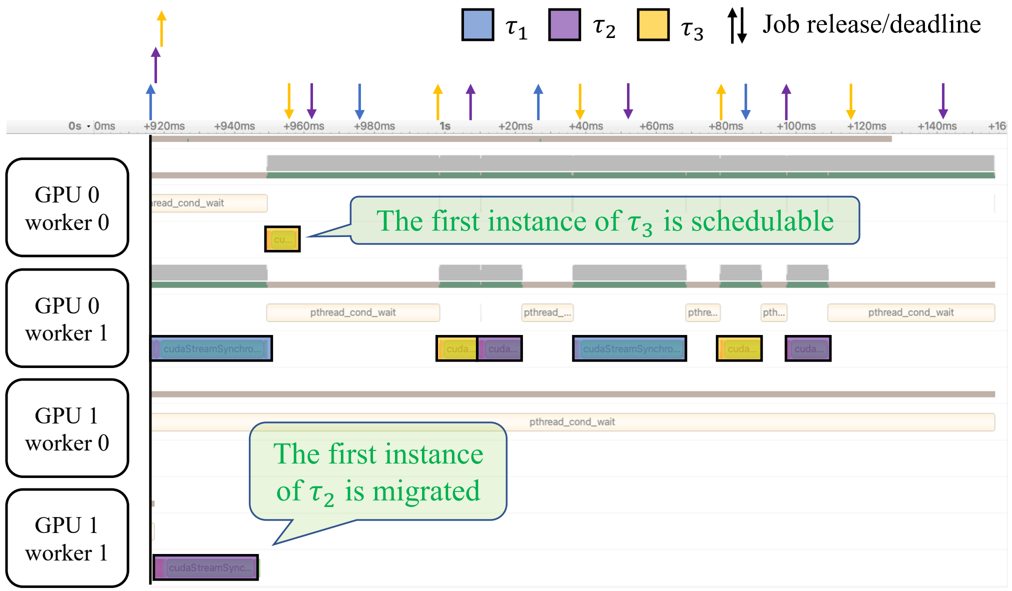
- Runtime Job Migration – Case Study 1

Table VII: Taskset used in case study 1

Task	$D_i = 0.5 * T_i$ (ms)	Offset (ms)	GPU assigned by Alg. 1
τ_1	60	0	RTX3070
τ_2	45	1	RTX3070
τ_3	40	2	RTX3070

✓ All three jobs are schedulable w/ migration



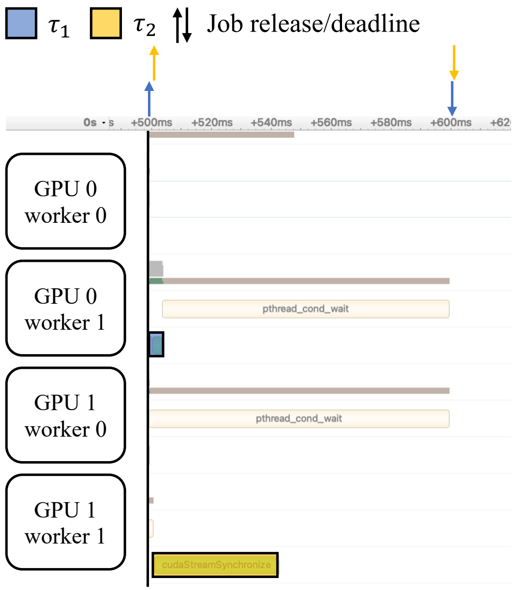


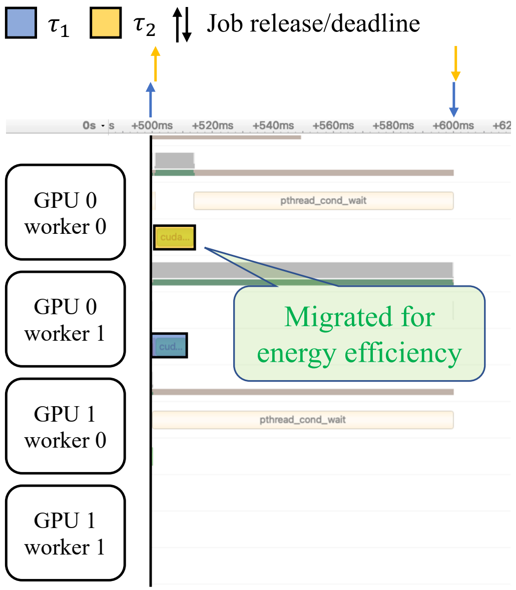
Energy-Efficient Multi-GPU Scheduling (6)

- Runtime Job Migration – Case Study 2

Table VIII: Taskset used in case study 2

Task	$D_i = 0.5 * T_i$ (ms)	Offset (ms)	GPU assigned by Alg. $\boxed{1}$
τ_1	100	0	RTX3070
τ_2	100	1	T400





✓ Energy consumption in two schedules:

- w/o migration - 6.51 J
- w/ migration - 6.49 J

Evaluation

- Multi-GPU System
  - NVIDIA RTX3070 + NVIDIA T400
  - Ubuntu 18.04 + CUDA 11.6
- Benchmark pool & Power parameters

(a) Dynamic power of benchmarks

Benchmark_i	$P^d_{0,i}(1)$	$P^d_{1,i}(1)$
MatrixMul	3.77 W	2.06 W
Stereodisparity	1.63 W	0.98 W
Hotspot	1.14 W	0.81 W
DXTC	1.67 W	1.15 W
BFS	0.98 W	1.07 W
Histogram	0.91 W	1.19 W

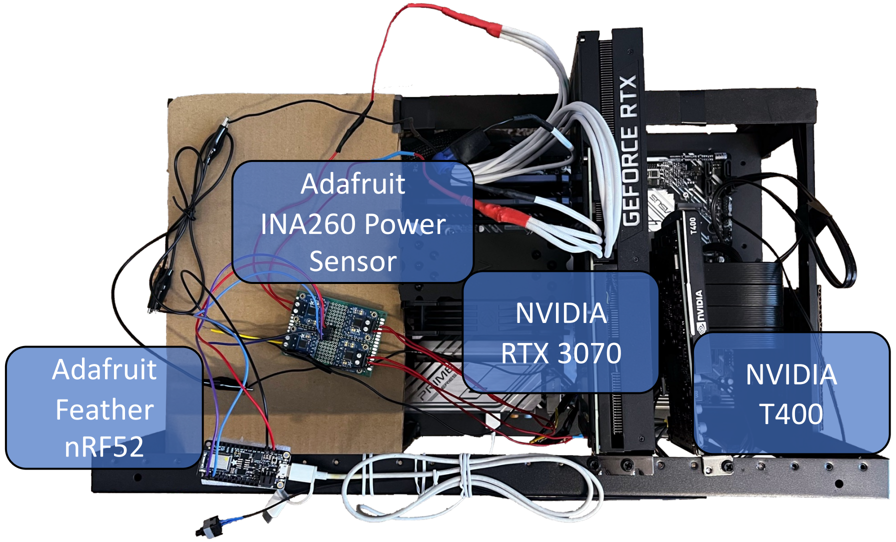
(b) Idle and static power of each GPU

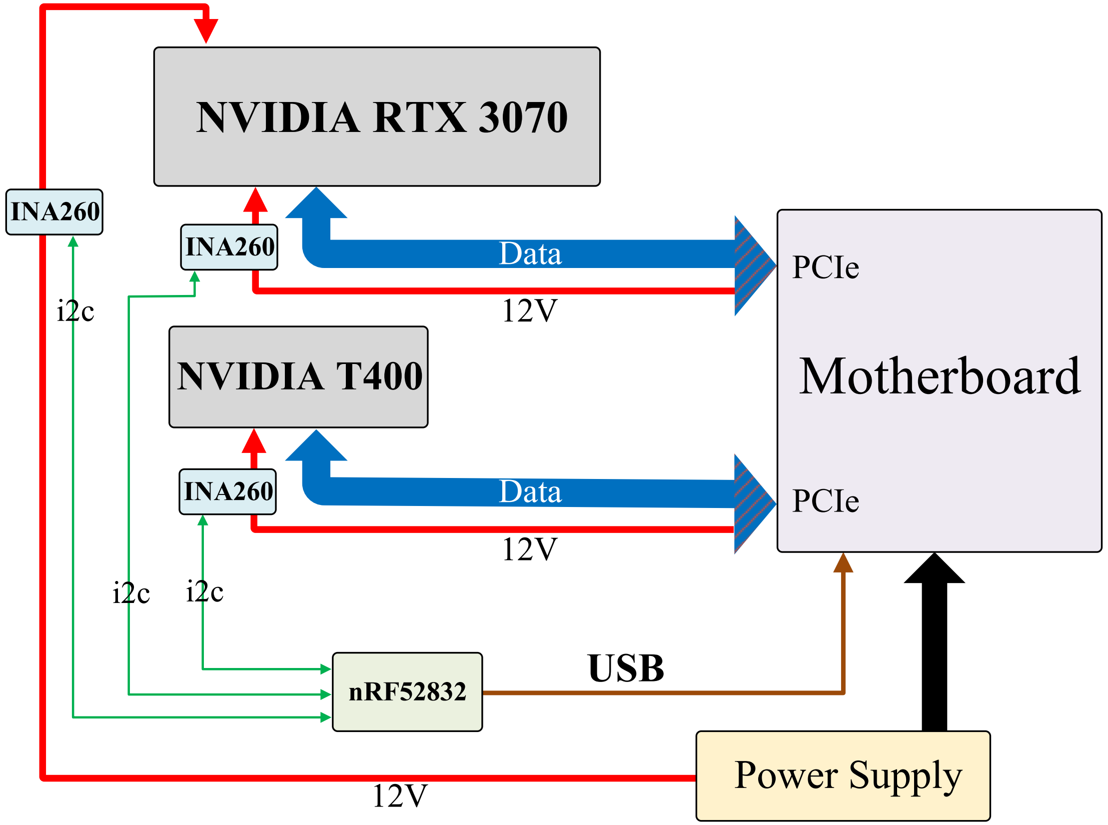
GPU_k	P^s_k	P^{idle}_k
π_0 (RTX 3070)	46 W	0.445 W
π_1 (T400)	8 W	0.652 W

- Scheduling Approaches
  - **sBEET-mg**
    - ➢ The complete version of the proposed framework
  - **sBEET-mg Offline Only**
    - ➢ The offline part of the proposed framework
  - **LCF ("Little-Core-First")**
  - **BCF ("Biggest-Core-First")**
    - ➢ Load concentration
  - **Load-Dist (load distribution):**
    - ➢ Load distribution

Hardware Setup

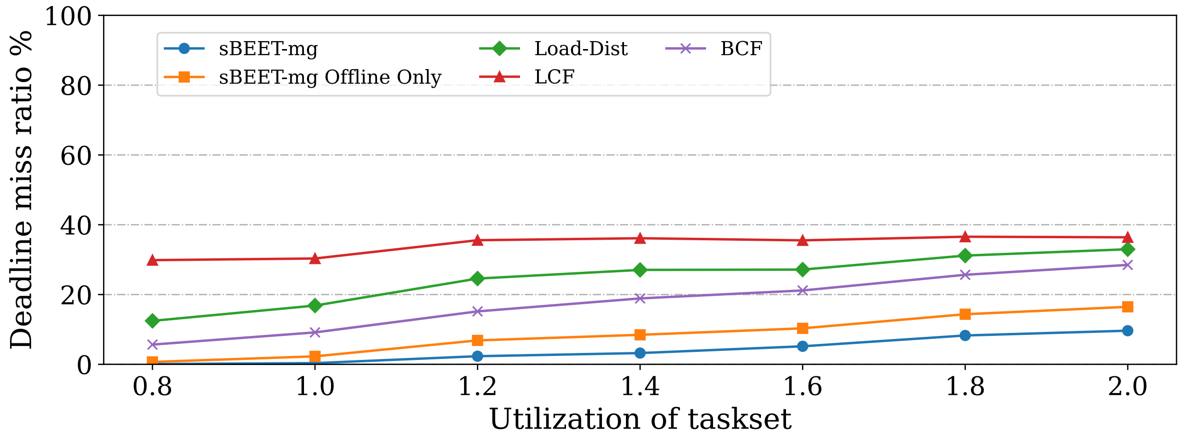
- Multi-GPU System
  - NVIDIA RTX3070 @ 1725 MHz
  - NVIDIA T400 @ 1425 MHz
- Custom Power Measurement Tool
  - nRF52832 SoC
  - INA260 power sensor

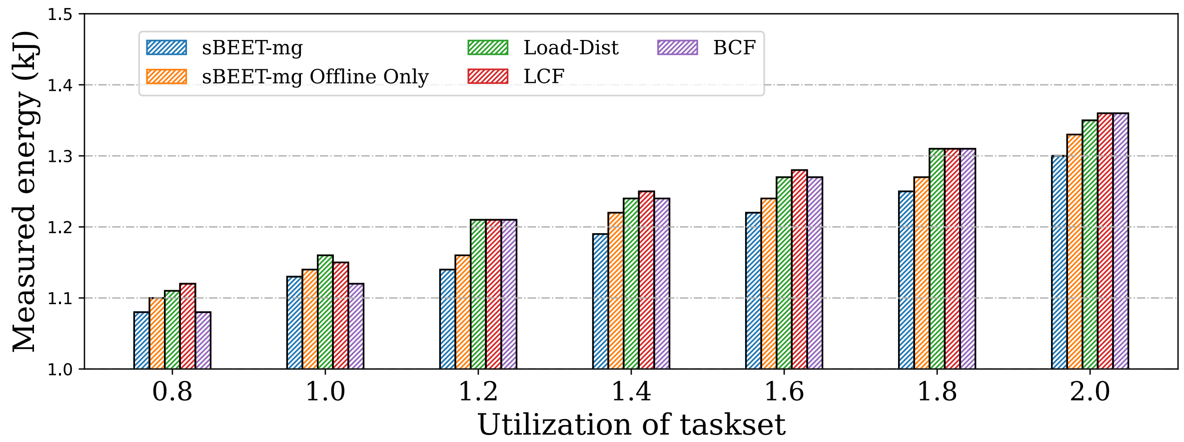




Performance Evaluation

- Taskset Generation
  - 100 randomly generated tasksets
  - Running for 15s on our multi-GPU system
- Experiment Settings
  - 24 SMs are allowed on RTX3070
  - Results of other settings can be found in the paper

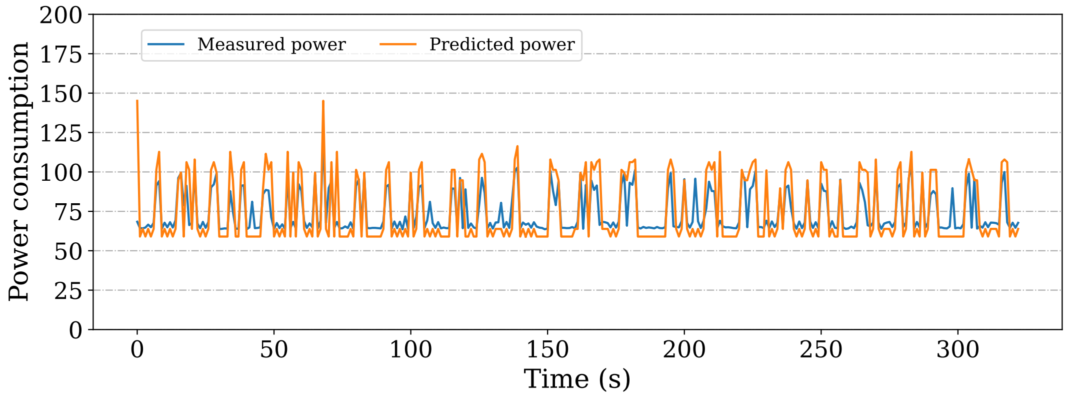




✓ Up to 23% and 18% less deadline misses compared to Load-Dist and BCF
✓ sBEET-mg has lower energy consumption

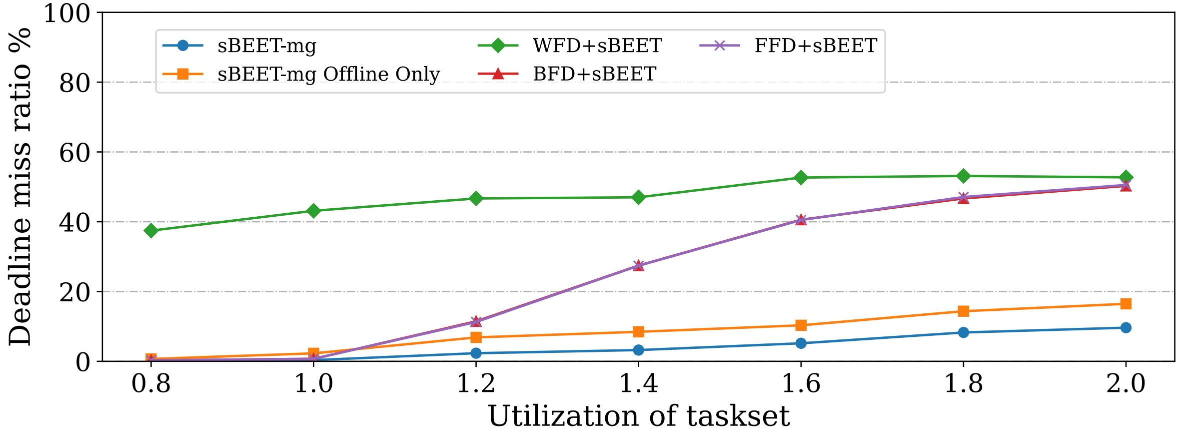
Power Prediction Accuracy

- Randomly generated one taskset under each utilization
- Average mean-absolute-error is 10.80 W (≈6% of 180W)
- More results can be found in the paper



Comparison with Previous Work - sBEET

- Taskset Generation
  - 100 randomly generated tasksets
  - Running for 15s on our multi-GPU system
- Experiment Settings
  - 24 SMs are allowed on RTX3070
- Scheduling Approaches
  - Proposed approaches
    - sBEET-mg, sBEET-mg Offline Only
  - sBEET w/ other allocation methods
    - WFD, FFD, BFD

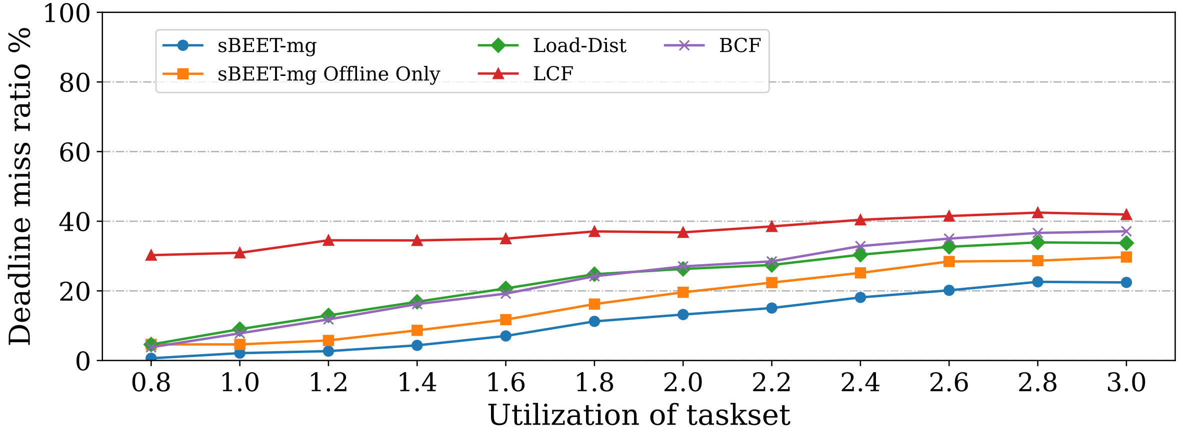


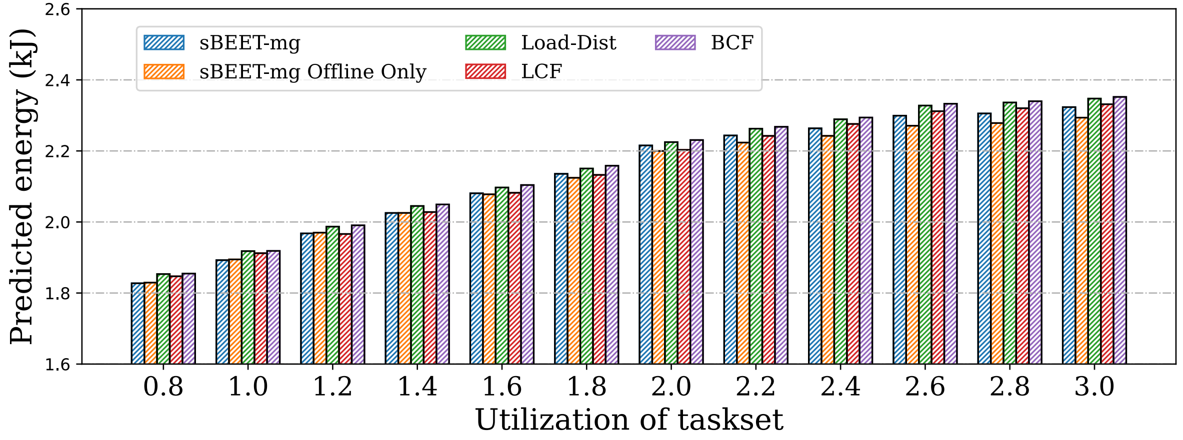
✓ Note that the results of BFD+sBEET and FFD+sBEET are overlapped

✓ sBEET-mg has the lowest deadline miss ratio

Simulation w/ Multiple GPUs

- Simulating a Multi-GPU System
  - RTX3070 w/ 12 SMs
  - RTX3070 w/ 12 SMs
  - T400 w/ all 6 SMs





Conclusion

- We observed that the existing simple task allocation approaches are not a preferred option for energy efficiency regardless of whether the GPU is homogeneous or heterogeneous
- We extended the prior work and proposed sBEET-mg, the multi-GPU scheduling framework that improves both schedulability and energy efficiency
- We designed a power monitoring setup for precise power measurement for our experiments
- Various experiments on both real hardware and simulation shows our proposed work can simultaneously reduce deadline misses and energy consumption

Source code available at https://github.com/rtenlab/sBEET-mg/

Towards Energy-Efficient Real-Time Scheduling of Heterogeneous Multi-GPU Systems

Yidi Wang, Mohsen Karimi, and Hyoseung Kim

Thank you!

https://github.com/rtenlab/sBEET-mg/