Balancing Energy Efficiency and Real-Time Performance in GPU Scheduling

Yidi Wang, Mohsen Karimi, Yecheng Xiang and Hyoseung Kim

University of California, Riverside

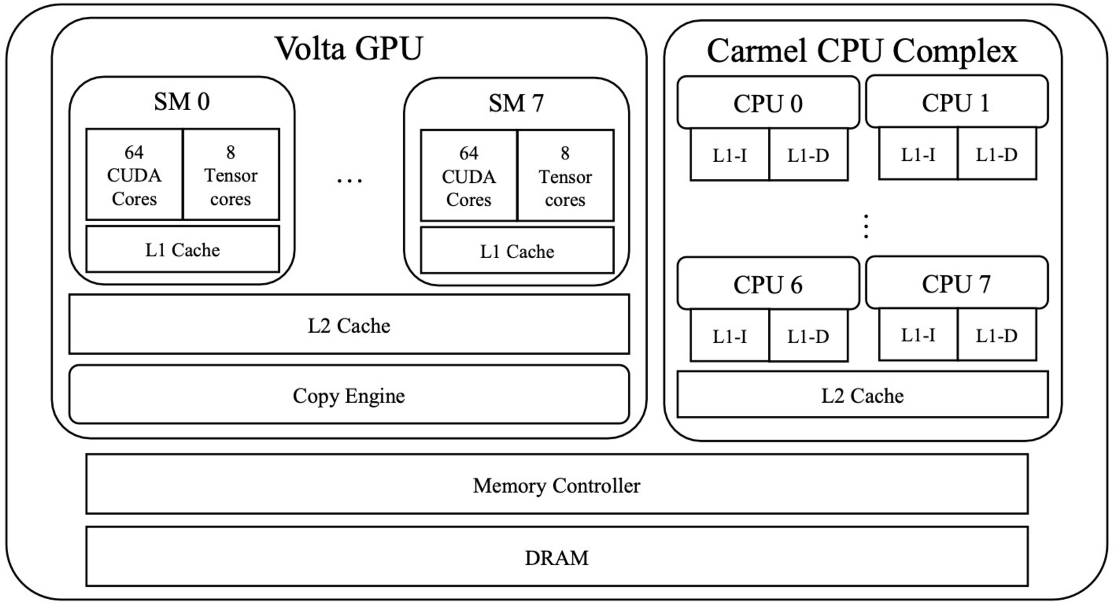
IEEE Real-Time Systems Symposium (RTSS) 2021

Introduction

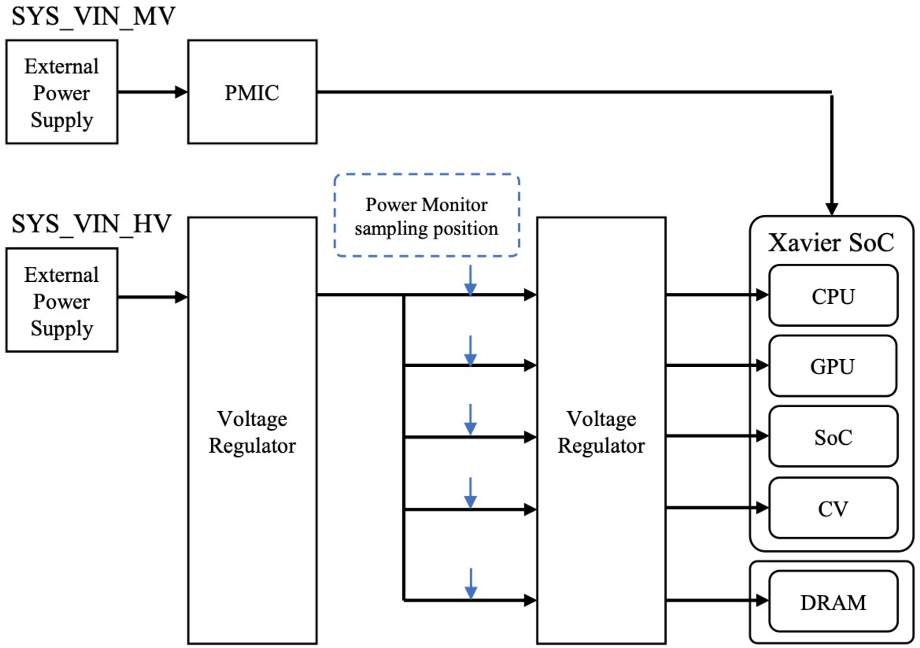
- GPU power management is important in CPS
  - GPUs are designed for better performance, with dramatically increased power consumption
  - Benefits of GPU power management:
    - Reliability, feasibility, scalability, etc.

- Partitioning the GPU can improve real-time performance and resource efficiency
  - Spatial multitasking partitions the GPU into computing units, so that multiple kernels can run simultaneously

NVIDIA Jetson AGX Xavier



(a) Xavier SoC Architecture



(b) Block diagram of Xavier power rails

Figure 1: Architecture and module power rails of NVIDIA Jetson AGX Xavier

- **The GPU is rail-gated and clock-gated, but not SM level power-gated**

Related Work

- Temporal Multitasking on GPU – Prior works specifically for real-time systems
  - Non-preemptive scheduling[1,2]: makes GPU access and blocking time predictable
  - Preemptive scheduling[3,4]: decomposes big kernel into smaller segments
  - GPU resources may be underutilized
- Spatial Multitasking on GPU[5]
  - It can reduce contention on computing resources between tasks
  - It may not lead to the most energy-efficient schedule
- Resource Allocation for GPU Energy Saving[6,7]
  - Turns off idling resources (i.e., SMs)
  - But SM-level power gating is not yet available even on the latest embedded GPUs

[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] H. Zhou, G. Tong, and C. Liu. GPES: a preemptive execution system for GPGPU computing. *RTAS*, 2015
[5] S. K. Saha, Y. Xiang, and H. Kim. STGM: Spatio-temporal GPU management for real-time tasks. *RTCSA*, 2019
[6] S. Hong and H. Kim. An integrated GPU power and performance model. *ACM SIGARCH*, 2010
[7] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng. Power gating strategies on GPUs. *TACO*, 2011

Contributions

We proposed sBEET:

✓ Real-time scheduling framework that Balances Energy Efficiency and Timeliness of GPU kernels on embedded GPUs

- Derive a power and energy consumption analysis for GPU kernels scheduled w/ and w/o spatial multitasking on the GPU
- Develop a runtime scheduler that balance the deadline misses and the energy consumption of non-preemptive GPU kernels
- Implement the scheduler on NVIDIA Jetson AGX Xavier
- The proposed work outperforms the existing spatial multitasking approach in real-time performance and energy consumption

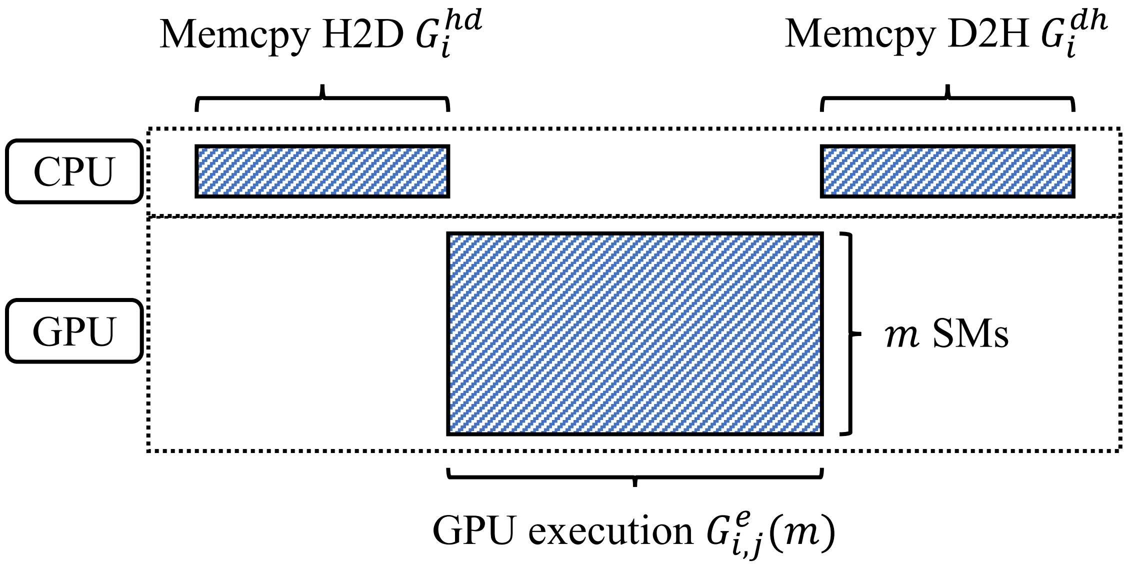
System Model

- System Model
  - A GPU containing $\boldsymbol{M}$ SMs
  - Single Memory Copy Engine
- Task Model
  - A taskset $\boldsymbol{\Gamma}$ consists of $\boldsymbol{n}$ periodic tasks:
    - Non-preemptive
    - W/ Constrained deadlines

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

WCET, period, deadline

- Job Model
  - Each task τ_i consists of a sequence of jobs $J_{i,j}$
  - Job are running exclusively on the assigned number of SMs



Power and Energy Analysis (1/5)

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

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

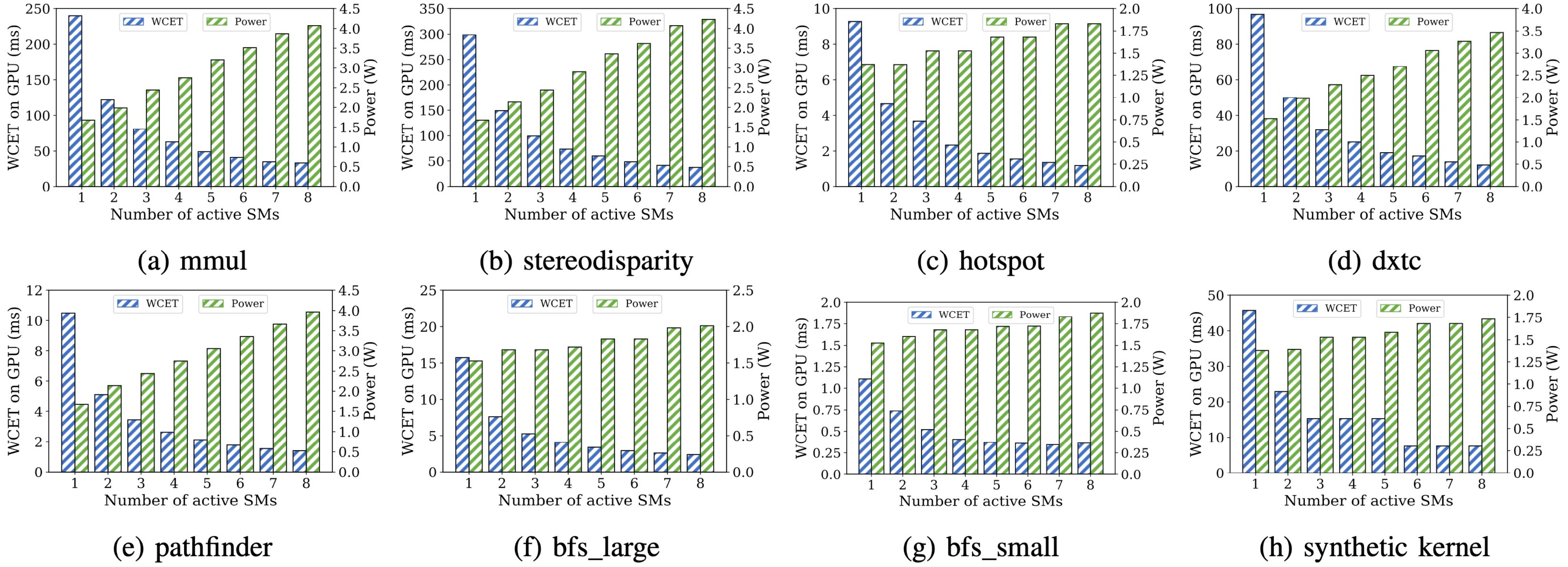
  - For a taskset Γ, energy consumption in [t1, t2]:

$$E(t_1, t_2) = \int_{t_1}^{t_2} \left(P^s + \sum_{i=1}^{n} \left(P_i^d \left(\sum_{k=1}^{M} x_i^k(t) \right) \right) + P^{idle} \left(M - \sum_{i=1}^{n} \sum_{k=1}^{M} x_i^k(t) \right) \right)$$

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

Power and Energy Analysis (2/5)

- WCET and power consumption profiling
  - Obtain power parameters for each application



(a) mmul (b) stereodisparity (c) hotspot (d) dxtc

(e) pathfinder (f) bfs_large (g) bfs_small (h) synthetic kernel

Figure 4: Profiling results of WCET and power consumption

Power and Energy Analysis (3/5)

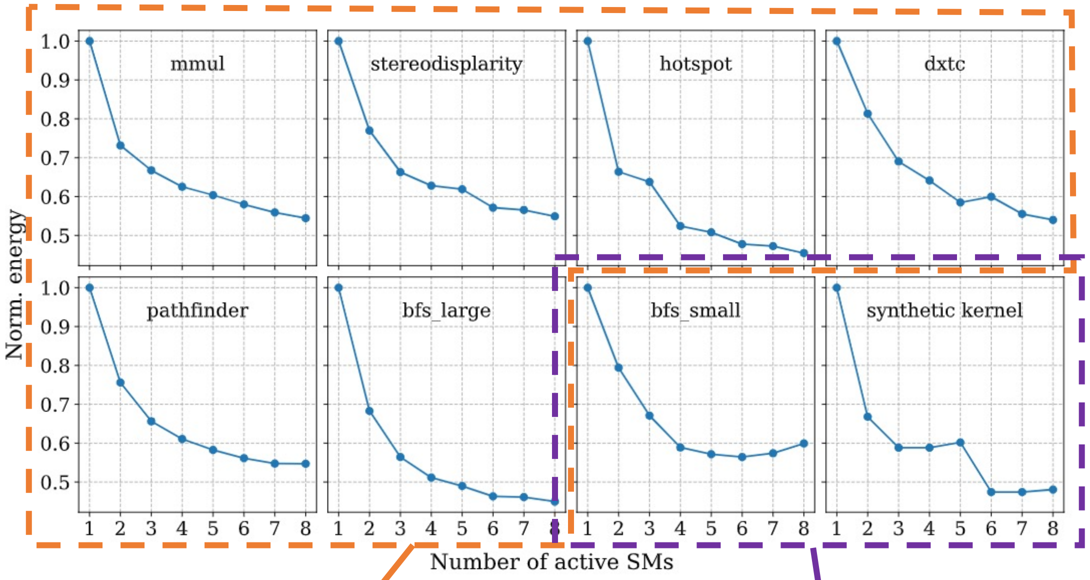


Figure 5: Normalized energy consumption in time window

Linear-speedup ($m^{opt} = M$)

Nonlinear-speedup ($m^{opt} < M$)

- **Definition 1. (m^{opt})** The energy-optimal number of SMs m^{opt} for a task τ_i 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.

Power and Energy Analysis (4/5)

Theorem 1

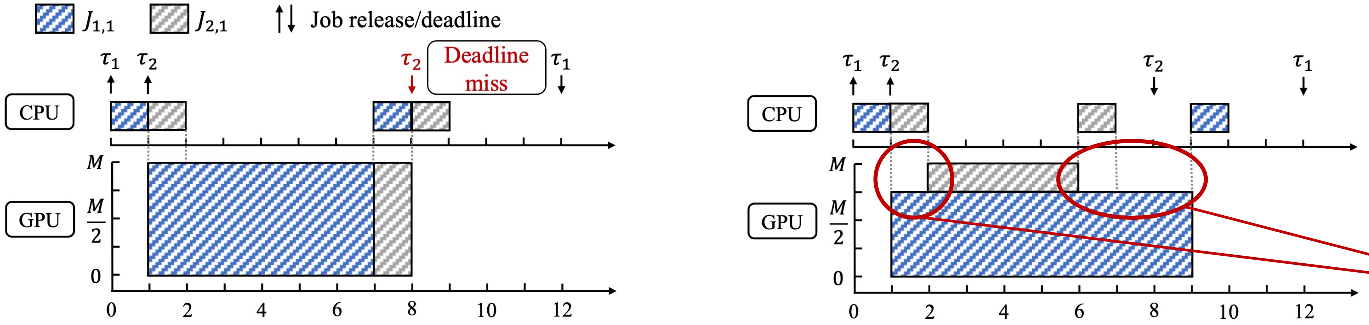
The schedule of a job set J with spatial multitasking **cannot be more energy-efficient** than the schedule without spatial multitasking if the jobs in J are ***linear-speedup*** jobs.

Schedule in (b):

- ✓ **Less energy efficient**
- ✓ **Better schedulability**

Consider two linear-speedup tasks:

Task	D_i	$G_i^e(M)$	G_i^{hd}	G_i^{dh}	Offset
τ_1	12	6	1	1	0
τ_2	7	1	1	1	1



(a) Schedule w/o spatial multitasking

(b) Schedule w/ spatial multitasking

Extra consumed energy due to idle SMs

Power and Energy Analysis (5/5)

Theorem 1

The schedule of a job set J with spatial multitasking **cannot be more energy-efficient** than the schedule without spatial multitasking if the jobs in J are ***linear-speedup*** jobs.

Schedule in (b):

- ✓ **Less energy efficient**
- ✓ **Better schedulability**

Lemma 2

Theorem 1 does not necessarily hold for *nonlinear-speedup* jobs.

- To reduce energy consumption:
  - ✓ For *linear-speedup* jobs, **execute them as fast as possible**
  - ✓ For *nonlinear-speedup* jobs, try to **assign the right number of SMs (m^{opt})** to them

Framework (1/3)

- Goals:
  - Minimize deadline misses
  - Maximize the opportunity to reduce energy consumption

- Approach:
  - A heuristic runtime scheduler:
    - ➢ Improve deadline misses by exploiting spatial multitasking technique
    - ➢ Reduce energy consumption by running each job with m^{opt} whenever possible
  - Two workers are created to parallelize the kernels
    - ➢ Motivated by hyperthreading on CPU

Framework (2/3)

- SM allocation policy:
  - The decision is made dynamically for each job
    - It is called when a new job arrives or a running job completes
  - When the GPU is idling:
    - ➢ Consider all the jobs that will arrive before $f_{i,j}(m)$
    - ➢ Generate all feasible schedules
    - ➢ Choose the schedule with the minimum predicted energy consumption
  - When the GPU is partially occupied:
    - ➢ Decide which one is more energy efficient:
      - o launch the job right away
      - o or wait until the current running job completes execution

Algorithm 2 SM Allocation

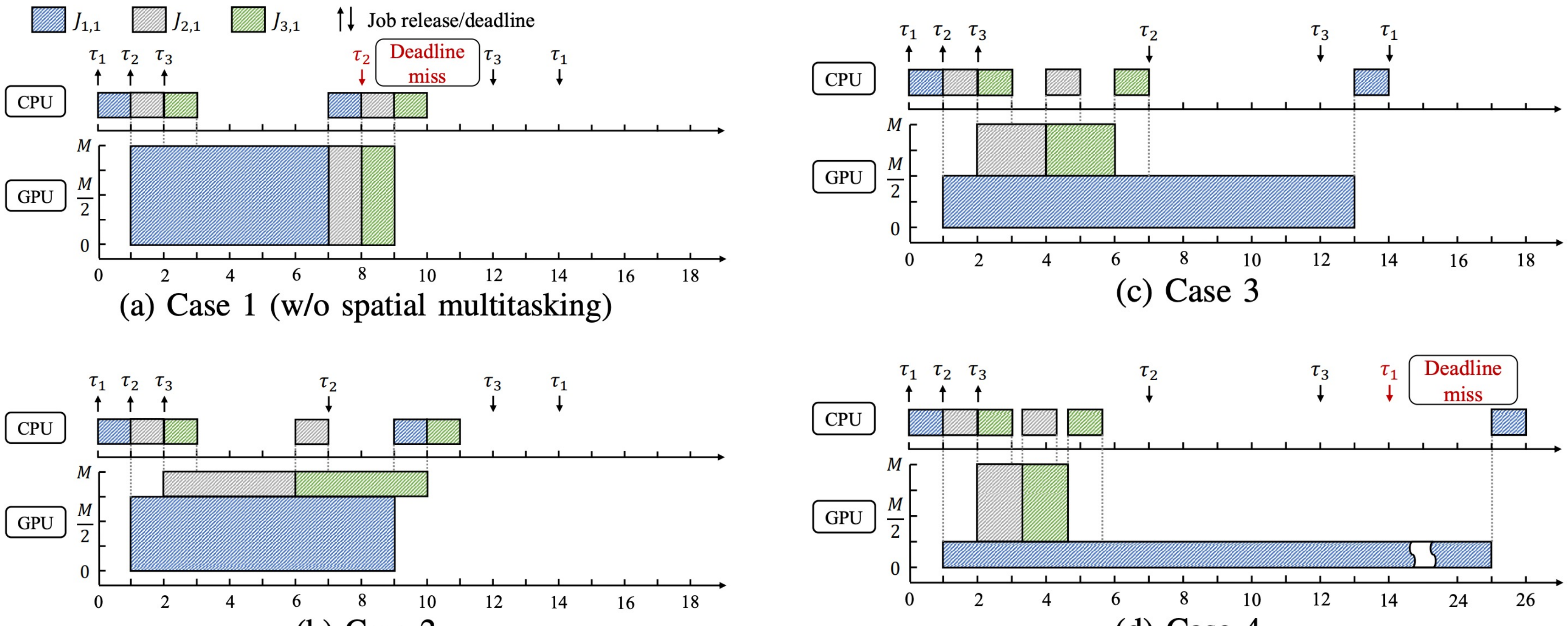
function ALLOCATION($J_{i,j}$, $J_{q,r}$)
 $t_{now} \leftarrow$ current time
 if $J_{q,r}$ is *nullptr* **then** ▷ $\Longrightarrow$ GPU is idling
  for $m \leftarrow M$ to 1 **do**
   $m' \leftarrow \min(m, m_i^{opt})$
   $Q_{i,j}^w \leftarrow \{J_{k,p} \mid \forall p, (\tau_k \neq \tau_q) \wedge (r_{k,p} < f_{i,j}(m'))\}$
   SCHEDGEN($J_{i,j}$, $J_{q,r}$, m', $[t_{now}, f_{i,j}(m')]$, $Q_{i,j}^w$)
   Compute $E_{pred} = E(t_{now}, f_{i,j}(m'))$ by Eq. (5)
  end for
  if no generated schedule is feasible **then**
   Choose the schedule with the minimum E_{pred}
  else
   Choose the feasible schedule with the min. E_{pred}
  end if
  return $S_{i,j}^{cfg}$ ▷ the corresponding SM allocation for $J_{i,j}$
 else ▷ the GPU is partially occupied
  $m' \leftarrow \min(|S_{avail}|, m_i^{opt})$
  if $f_{i,j}(m') > f_{q,r} + G_{i,j}^e(M)$ **then**
   return $\emptyset$ ▷ Do not run $J_{i,j}$ in parallel with $J_{q,r}$
  else
   $Q_{i,j}^w \leftarrow \{J_{k,p} \mid \forall p, (\tau_k \neq \tau_q) \wedge (r_{k,p} < f_{i,j}(m'))\}$
   SCHEDGEN($J_{i,j}$, $J_{q,r}$, m', $[t_{now}, f_{i,j}(m')]$, $Q_{i,j}^w$)
   if the generated schedule is not feasible **then**
    return $\emptyset$
   else
    return $S_{i,j}^{cfg}$ ▷ the corresp. SM allocation
   end if
  end if
 end if
end function

Framework (3/3)

- High-level idea of the scheduler:
  - generates the possible schedules
  - Then choose the one with minimum energy consumption and w/o deadline violation

Table II: Taskset in Example 2

Task	D_i	$G_i^e(M)$	G_i^{hd}	G_i^{dh}	Offset
τ_1	14	6	1	1	0
τ_2	7	1	1	1	1
τ_3	10	1	1	1	2



(a) Case 1 (w/o spatial multitasking)

(b) Case 2

(c) Case 3

(d) Case 4

- Time complexity: O(nlogn)

Evaluation

- Experiment Setup
  - NVIDIA Jetson AGX Xavier with Ubuntu 18.04 and CUDA 10.0
    - ➢ 670 MHz GPU clock frequency
    - ➢ All CPU cores are enabled
  - GPU power consumption is measured from the built-in power sensor
- Scheduling Approaches
  - **sBEET:**
    - ➢ the proposed approach
  - **FCFS, RM:**
    - ➢ temporal-multitasking
  - **STGM[1]:**
    - ➢ temporal-multitasking and spatial-multitasking

[1] S. K. Saha, Y. Xiang, and H. Kim. STGM: Spatio-temporal GPU management for real-time tasks. *RTCSA*, 2019

Power Model Evaluation

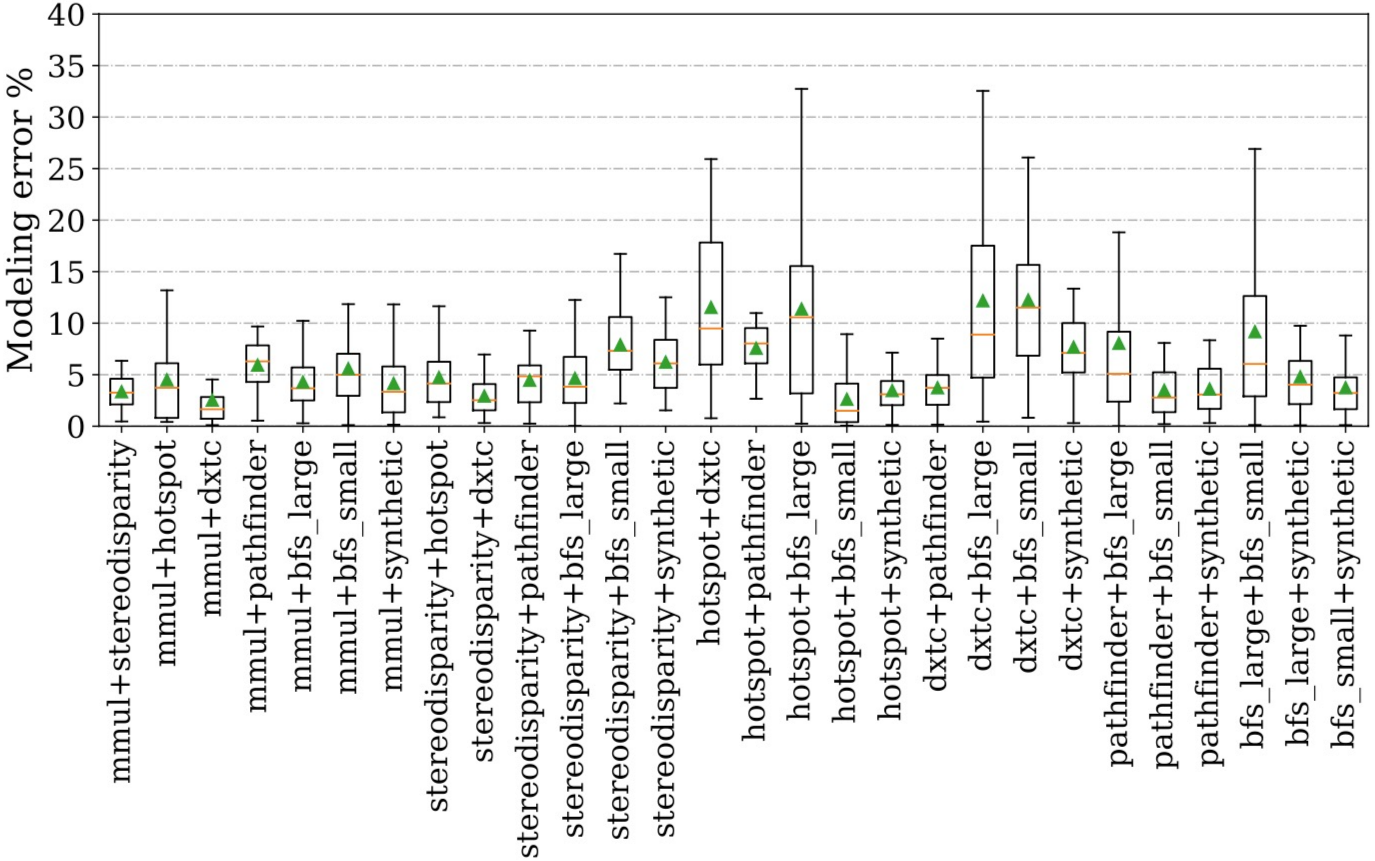
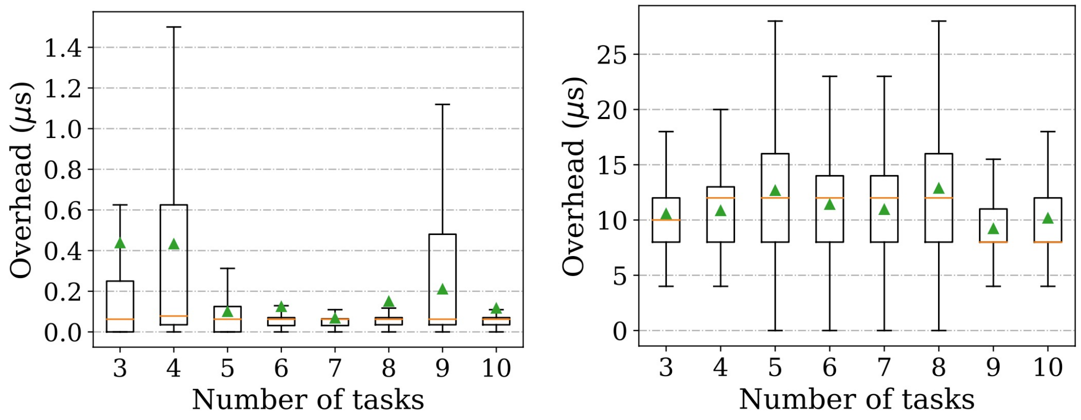


Figure 6: Error of predicted GPU power consumption

- The average error is 5.93%
- R2 score is 0.87

Overhead Measurement

- The overhead comes from the decision-making of the scheduler
- A taskset of total utilization of 1.0 is executed for 10 minutes



(a) Overhead of Alg. 1 (b) Overhead of Alg. 2 and 3

Figure 7: Runtime overhead w.r.t number of tasks

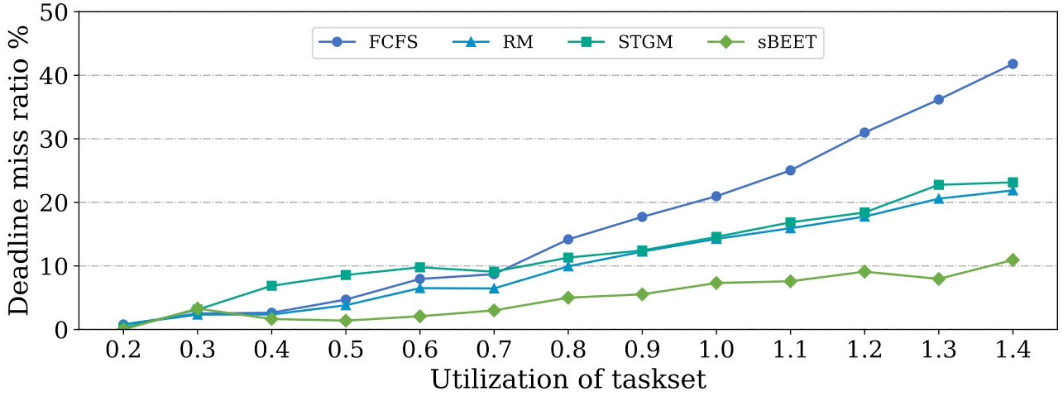
✓ The overhead is acceptable for our target embedded platform

Effect of Taskset Utilization

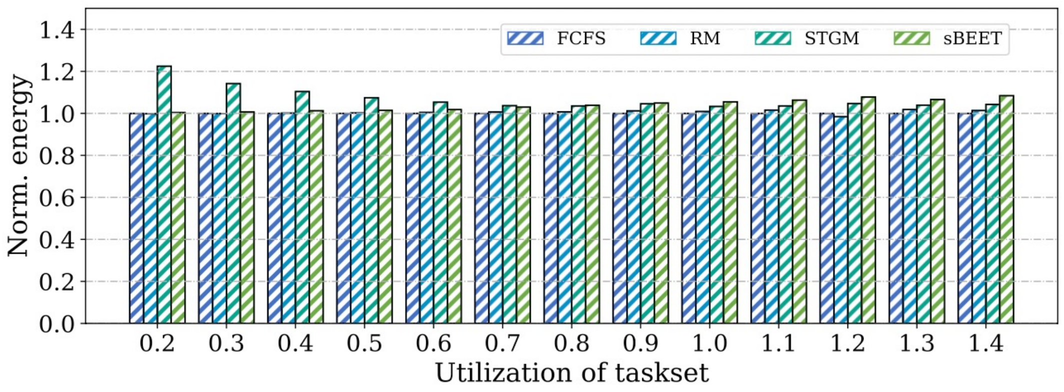
- To see the schedulability and energy consumption of different approaches when the system is overloaded
- Taskset generation
  - 1,000 randomly-generated tasksets
  - Running for 10 secs on real hardware

✓ sBEET has the lowest deadline miss ratio

✓ When the utilization gets larger, the energy consumption of sBEET becomes the highest due to the use of spatial multitasking and sBEET has more completed jobs than others



(a) Deadline miss ratio



(b) Overall energy consumption

Figure 8: Runtime results w.r.t. the utilization of taskset

Effect of Heavy/Light Task Ratio

- Heavy tasks are likely to have negative impact on schedulability
- Task categorization
  - Heavy tasks: MMUL, Stereodisparity, DXTC
  - Light tasks: Hotspots, Pathfinder, BFS, the synthetic kernel

✓ sBEET is better at meeting the deadlines since the long blocking by heavy tasks can be avoided

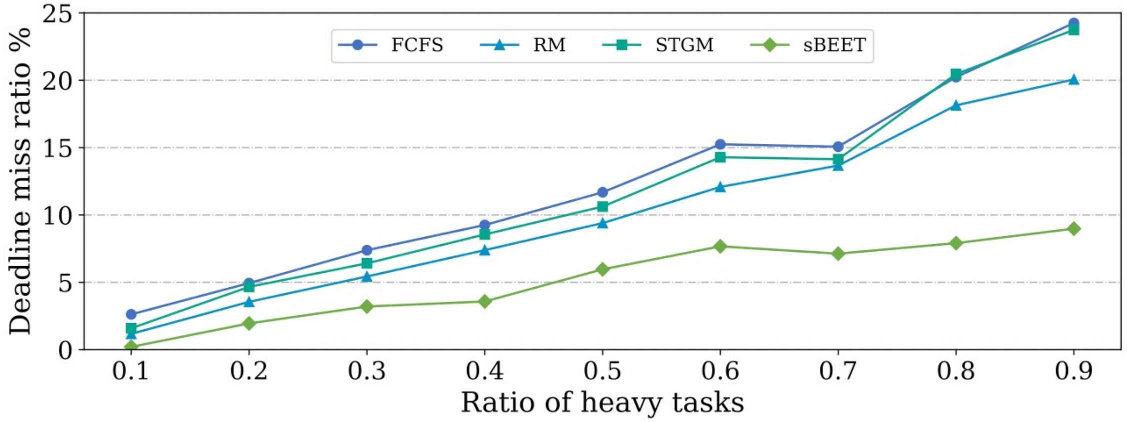


Figure 9: Runtime deadline miss ratio of light tasks w.r.t. ratio of heavy tasks

Effect of Spatial Multitasking

- Focus on the energy efficiency w/ spatial-multitasking
- All the tasksets can pass the original STGM offline schedulability test which guarantees no deadline miss

✓ Both have 0% deadline miss ratio

✓ sBEET can save up to 21% of the energy

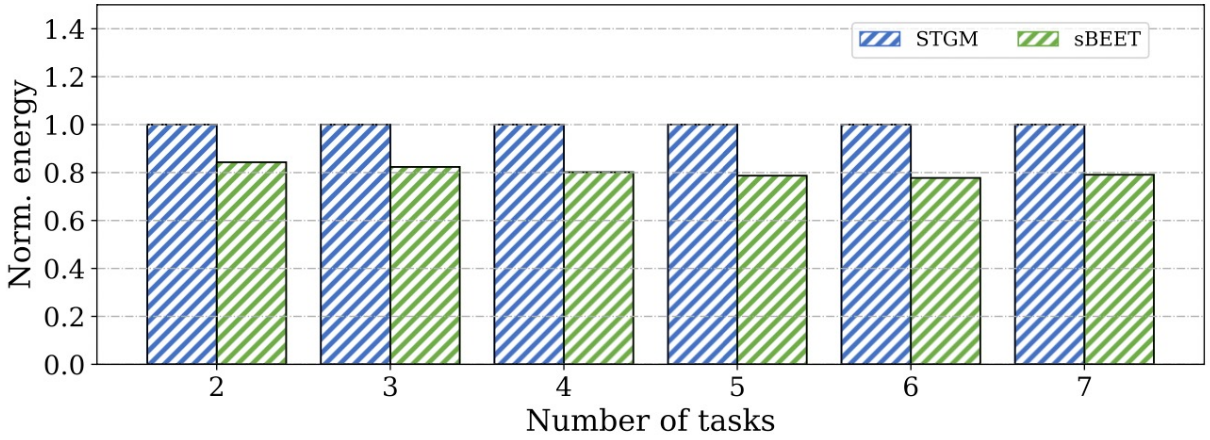


Figure 10: Comparison of runtime energy consumption of STGM and the proposed work

Discussion

- Shared memory resource contention
  - Co-scheduled kernels may experience additional timing interference due to contention on shared memory resources of the GPU
  - We did not observe any discernible slowdown:
    - ➢ The target platform has a small number of SMs and a high memory bandwidth
  - Can be co-used with *Fractional GPUs*[1]

- Energy consumed by other hardware components
  - Including CPU, memory, etc.
  - It will be more challenging to optimize the energy consumption of the whole hardware

[1] S. Jain, I. Baek, S. Wang, and R. Rajkumar. Fractional GPUs: Software-based compute and memory bandwidth reservation for GPUs. *RTAS*, 2019

Conclusion and Future Work

- Conclusion
  - Our power and energy analysis shows that spatial multitasking on the GPU benefits schedulability, but may lead to energy inefficiency due to the energy consumed by idle SMs
  - The proposed runtime scheduler balances the schedulability and energy efficiency
  - We implemented the scheduler on NVIDIA Jetson AGX Xavier
  - Experimental results show that the proposed scheduler can achieve better energy efficiency in meeting tasks' deadlines

- Future work
  - Extend the current work to more powerful GPUs
  - Consider heterogeneous multi-GPU systems
  - Consider the energy consumption of the whole hardware
  - Extend our idea to other systems, e.g., DNN inference servers and autonomous driving

Balancing Energy Efficiency and Real-Time Performance in GPU Scheduling

Yidi Wang, Mohsen Karimi, Yecheng Xiang and Hyoseung Kim

Thank you!