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

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

(a) Xavier SoC Architecture
(b) Block diagram of Xavier power rails

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

---

Contributions

We proposed sBEET:
✓ Real-time scheduling framework that \textit{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 $M$ SMs
  - Single Memory Copy Engine

- Task Model
  - A taskset $\Gamma$ consists of $n$ periodic tasks:
    - Non-preemptive
    - W/ Constrained deadlines
    $\tau_i := (G_i, T_i, D_i)$
    WCET, period, deadline

- Job Model
  - Each task $\tau_i$ 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 \( J = \{J_1, J_2, \ldots, J_n\} \):
    \[
    P = P_s + \sum_{i=1}^{n} P_d^i(m_i) + P_{idle}(M - \sum_{i=1}^{n} m_i)
    \]
  - For a taskset \( \Gamma \), energy consumption in \([t_1, t_2]\):
    \[
    E(t_1, t_2) = \int_{t_1}^{t_2} \left( P_s + \sum_{i=1}^{n} \left( P_d^i \left( \sum_{k=1}^{M} x_{i}^{k}(t) \right) \right) + P_{idle} \left( M - \sum_{i=1}^{n} \sum_{k=1}^{M} x_{i}^{k}(t) \right) \right)
    \]

\( x_{i}^{k}(t) = \begin{cases} 
0, & \text{\( \tau_i \) is not active on SM}_k \\
1, & \text{\( \tau_i \) is active on SM}_k 
\end{cases} \)
Power and Energy Analysis (2/5)

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

![Graphs showing WCET and power consumption for different applications](image)

(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)

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

Figure 5: Normalized energy consumption in time window

- Linear-speedup \((m^{opt} = M)\)
- Nonlinear-speedup \((m^{opt} < M)\)
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.

Consider two linear-speedup tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>$D_i$</th>
<th>$G_r^c(M)$</th>
<th>$G_d^h$</th>
<th>$G_d^{dh}$</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Schedule in (b):
- Less energy efficient
- Better schedulability

Extra consumed energy due to idle SMs
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.

**Theorem 1**

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

**Schedule in (b):**

- Less energy efficient
- Better schedulability

**Lemma 2**

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:
    - launch the job right away
    - or wait until the current running job completes execution

---

**Algorithm 2 SM Allocation**

1. function `ALLOCATION(J_{i,j}, J_{q,r})`
2. $t_{now} \leftarrow$ current time
3. if $J_{q,r}$ is `nullptr` then
   4. for $m \leftarrow M$ to 1 do
   5.   $m' \leftarrow \min(m, m_{opt})$
   6.   $Q_{i,j}^w \leftarrow \{J_{k,p} \mid \forall p, (r_k \neq r_q) \wedge (r_{k,p} < f_{i,j}(m'))\}$
   7.   $SCHEDEGEN(J_{i,j}, J_{q,r}, m', [t_{now}, f_{i,j}(m')], Q_{i,j}^w)$
   8.   Compute $E_{pred} = E(t_{now}, f_{i,j}(m'))$ by Eq. (5)
   9. end for
10. if no generated schedule is feasible then
   11.   Choose the schedule with the minimum $E_{pred}$
   12. else
   13.   Choose the feasible schedule with the min. $E_{pred}$
   14. end if
15. return $S_{i,j}^{c,f}$ > the corresponding SM allocation for $J_{i,j}$
16. if $J_{q,r}$ is `nullptr` then
17.   $m' \leftarrow \min(|S_{avail}|, m_{opt})$
18. else if $f_{i,j}(m') > f_{q,r} + G_{i,j}(M)$ then
19.   return $\emptyset$ > Do not run $J_{i,j}$ in parallel with $J_{q,r}$
20. else
21.   $Q_{i,j}^w \leftarrow \{J_{k,p} \mid \forall p, (r_k \neq r_q) \wedge (r_{k,p} < f_{i,j}(m'))\}$
22.   $SCHEDEGEN(J_{i,j}, J_{q,r}, m', [t_{now}, f_{i,j}(m')], Q_{i,j}^w)$
23.   if the generated schedule is not feasible then
24.     return $\emptyset$
25. else
26.     return $S_{i,j}^{c,f}$ > the corresponding SM allocation
27. end if
28. end if
29. end if
30. 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

- Time complexity: $O(n \log n)$
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
  - **STGM¹:**
    - temporal-multitasking and spatial-multitasking

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

Figure 6: Error of predicted GPU power consumption
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
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

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 with 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](image)

1/12/22
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 GPUs¹*

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

---


1/12/22
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!