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

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 – one single CUDA context
  - Two worker threads dedicated for each GPU
System Model

- **Platform Model**
  - A single-ISA system $\Pi$ consisting with $\omega$ heterogeneous GPUs
  - A GPU $\pi_k$ containing $M_k$ SMs

- **Task Model**
  - A taskset $\Gamma$ consists of $n$ periodic GPU tasks:
    - Non-preemptive
    - W/ Constrained deadlines
    - Each task $\tau_i$ 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 $J_{i,j}$:
$$G_{i,j}(m, \pi_k) = G_i^{hd}(\pi_k) + G_i^e(m, \pi_k) + G_i^{dh}(\pi_k)$$

- Memcpy H2D $G_i^{hd}(\pi_k)$
- Memcpy D2H $G_i^{dh}(\pi_k)$

$$\tau_i := (G_i, T_i, D_i)$$

WCET, period, deadline
Power and Energy Model

- **Power model**
  - Power model: \( P = P_s + P^d + P^{idle} \)
  - For a set of jobs \( J = \{ J_1, J_2, \ldots, J_n \} \):
    \[
    P = P_s + \sum_{i=1}^{n} P^d_i(m_i) + P^{idle}(M - \sum_{i=1}^{n} m_i)
    \]
  - For a taskset \( \Gamma \), energy consumption in \([t_1, t_2]\):\[
  E_k(t_1, t_2) = \int_{t_1}^{t_2} \left( P^s_k + \sum_{j_i \in J} \left( p_{k,i}^d \left( \sum_{m=1}^{M_k} x_i^m(t) \right) \right) + p_k^{idle} \left( M_k - \sum_{j_i \in J} \sum_{m=1}^{M_k} x_i^m(t) \right) \right) dt
  \]
  - 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>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>$G^c_i(\pi_0, 6)$</th>
<th>$G^c_i(\pi_0, 4)$</th>
<th>$G^c_i(\pi_0, 3)$</th>
<th>$G^c_i(\pi_0, 2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1 = \tau_2$</td>
<td>Histogram</td>
<td>32.67 ms</td>
<td>47.95 ms</td>
<td>63.724 ms</td>
<td>95.53 ms</td>
</tr>
</tbody>
</table>

**Load distribution**

- GPU 0 T400
  - $f = 1425$ MHz
  - E = 2.3 J

- GPU 1 T400
  - $f = 1425$ MHz

**Load Concentration**

- GPU 0 T400
  - $f = 1425$ MHz
  - E = 2.05 J

Load concentration is better in this case
Insights on Conventional Approaches (3)

- Homogeneous GPUs
  - Example 2
    - Same taskset, but $\tau_1$ executes slightly earlier with 4 SMs

Table III: Taskset in Examples 1 and 2

<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>$G^c_1(\pi_0, 6)$</th>
<th>$G^c_1(\pi_0, 4)$</th>
<th>$G^c_1(\pi_0, 3)$</th>
<th>$G^c_1(\pi_0, 2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1 = \tau_2$</td>
<td>Histogram</td>
<td>32.67 ms</td>
<td>47.95 ms</td>
<td>63.724 ms</td>
<td>95.53 ms</td>
</tr>
</tbody>
</table>

Load distribution

GPU 0
T400
$f=1425$MHz

GPU 1
T400
$f=1425$MHz

Load Concentration

E=2.12J

E=2.18J

A small difference made load distribution the winner
Heterogeneous GPUs
Example 1

Table IV: Taskset in Example 3 and 4

<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>$C_t^n(30, \pi_0)$</th>
<th>$C_t^n(16, \pi_0)$</th>
<th>$C_t^n(6, \pi_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>MatrixMul</td>
<td>11.98 ms</td>
<td>21.55 ms</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>Hotspot</td>
<td>12.00 ms</td>
<td>22.31 ms</td>
<td>73.188 ms</td>
</tr>
</tbody>
</table>

Load distribution

GPU 0
RTX3070

f = 1725MHz

Load Concentration

GPU 1
T400

f = 1425MHz

$E = 7.35J$

$E = 7.24J$
Insights on Conventional Approaches (5)

- Heterogeneous GPUs
- Example 2

**Table IV: Taskset in Example 3 and 4**

<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>$G_{\pi}^m(30, \pi_0)$</th>
<th>$G_{\pi}^m(16, \pi_0)$</th>
<th>$G_{\pi}^m(6, \pi_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>MatrixMul</td>
<td>11.98 ms</td>
<td>21.55 ms</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>Hotspot</td>
<td>12.00 ms</td>
<td>22.31 ms</td>
<td>73.188 ms</td>
</tr>
</tbody>
</table>

**Load distribution**
- GPU 0: RTX3070, $f = 1725$ MHz
- GPU 1: T400, $f = 1425$ MHz

- $E = 7.19$ J

**Load Concentration**

- $E = 7.3$ J
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 $\tau_i$ on a GPU $\pi_k$ 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 $\tau_i$ in a multi-GPU system $\Pi$ is the GPU that consumes the least amount of energy when $\tau_i$ executes with $m_{k,i}^{opt}$ SMs on it.

\[
\arg\min_{\pi_k \in \Pi} \int_0^\delta P_k^s + P_{k,i}^d(m_{k,i}^{opt}) + P_{k,\text{idle}}(M_k - m_{k,i}^{opt})dt
\]
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**

1. **procedure** TASK DISTRIBUTION
2. Sort tasks in $\Gamma$ in decreasing order of priority
3. for $\tau_i \in \Gamma$ do
4. Get a list $\Pi_i$ of GPUs in non-increasing order of expected energy consumption for $\tau_i$
5. for $\pi_k \in \Pi_i$ do
6. if $U(\pi_k) + U_i(\pi_k, m_{opt}) \leq 1$ then
7. Assign $\tau_i$ to $\pi_k$
8. break
9. end if
10. end for
11. if $\tau_i$ is not assigned then
12. Assign $\tau_i$ to the GPU that has a minimum utilization after $\tau_i$ is assigned
13. end if
14. end for
15. **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

<table>
<thead>
<tr>
<th>Task</th>
<th>$D_i = 0.5 \times T_i$ (ms)</th>
<th>Offset (ms)</th>
<th>GPU assigned by Alg. [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>60</td>
<td>0</td>
<td>RTX3070</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>45</td>
<td>1</td>
<td>RTX3070</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>40</td>
<td>2</td>
<td>RTX3070</td>
</tr>
</tbody>
</table>

- All three jobs are schedulable w/ migration

The first instance of $\tau_3$ is migrated

The first instance of $\tau_3$ is schedulable

The first instance of $\tau_2$ is migrated

The first instance of $\tau_3$ is skipped
Energy-Efficient Multi-GPU Scheduling (6)

- Runtime Job Migration – Case Study 2

Table VIII: Taskset used in case study 2

<table>
<thead>
<tr>
<th>Task</th>
<th>$D_i = 0.5 \times T_i$ (ms)</th>
<th>Offset (ms)</th>
<th>GPU assigned by Alg. 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>100</td>
<td>0</td>
<td>RTX3070</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>100</td>
<td>1</td>
<td>T400</td>
</tr>
</tbody>
</table>

✔ 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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$P_{d,i}(1)$</th>
<th>$P_{d,i}^{d}(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatrixMul</td>
<td>3.77 W</td>
<td>2.06 W</td>
</tr>
<tr>
<td>Stereodisparity</td>
<td>1.63 W</td>
<td>0.98 W</td>
</tr>
<tr>
<td>Hotspot</td>
<td>1.14 W</td>
<td>0.81 W</td>
</tr>
<tr>
<td>DXTC</td>
<td>1.67 W</td>
<td>1.15 W</td>
</tr>
<tr>
<td>BFS</td>
<td>0.98 W</td>
<td>1.07 W</td>
</tr>
<tr>
<td>Histogram</td>
<td>0.91 W</td>
<td>1.19 W</td>
</tr>
</tbody>
</table>

(b) Idle and static power of each GPU

<table>
<thead>
<tr>
<th>GPU_k</th>
<th>$P_s^k$</th>
<th>$P_{idle}^k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_0$ (RTX 3070)</td>
<td>46 W</td>
<td>0.445 W</td>
</tr>
<tr>
<td>$\pi_1$ (T400)</td>
<td>8 W</td>
<td>0.652 W</td>
</tr>
</tbody>
</table>

- 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/rttenlab/sBEET-mg/