

CycleTandem: Energy-Saving Scheduling for Real-Time Systems with Hardware Accelerators

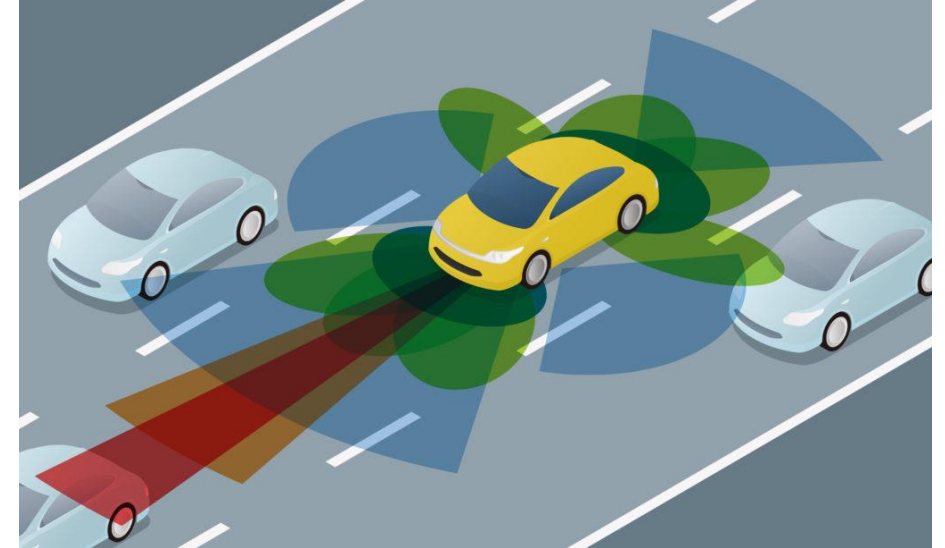
Sandeep D'souza and Ragunathan (Raj) Rajkumar
Carnegie Mellon University

**Carnegie
Mellon
University**



High (Energy) Cost of Accelerators

- **Modern-day CPS**
 - real-time decision making
- **GP-GPUs, ASICs, FPGAs, DSPs ...**
 - *high computational* power
 - with *significant energy* requirements



Energy Budget is *limited*

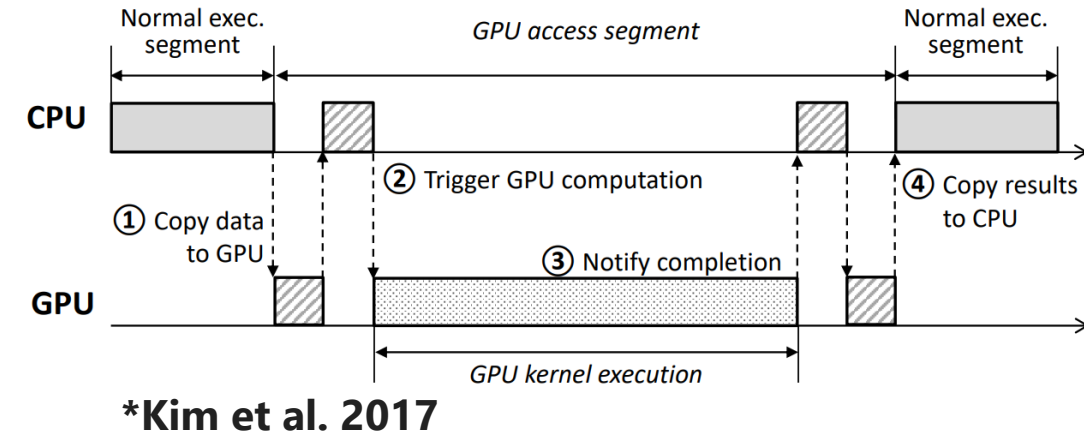
→ Focus on *reducing* hardware-accelerator power consumption

Outline

- Motivation
- **Background & Related Work**
 - *Schedulability Analysis*
 - *Energy Management*
- *CycleSolo*: Uniprocessor + Accelerator
- *CycleTandem*: Uniprocessor + Accelerator
- Multi-core Extensions
- Results
- Conclusion

System Model

- **Sporadic Hard Real-Time Tasks** $\tau_i: (C_i, G_i, D_i, T_i)$
 - C_i : CPU WCET of any job of task τ_i
 - G_i : Accelerator WCET of any job of task τ_i
 - G_i^e : Accelerator Execution
 - G_i^m : CPU intervention
 - D_i : Deadline
 - T_i : Period (Minimum inter-arrival time)
- **Fixed-priority** and **Fully-Partitioned** Multi-core Scheduling



Assumptions: Each task has a **single** accelerator segment, tasks **self-suspend** on the CPU & the accelerator is **non-preemptive**

Related Work: Schedulability Analysis

- **Lock-based Approach**

- lock to *arbitrate* accelerator access
- [Elliott et al. 2012][Elliott et al. 2013]
[Chen et al. 2016][Patel et al. 2018] ...



- **Server-based Approach**

- server *accesses* accelerator on behalf of tasks
- [Kim et al. 2017]



Utilize the *lock-based* approach with MPCP [Rajkumar 1990]
→ proposed techniques can be extended to the server-based approach

MPCP-based Analysis

- *with self-suspensions*

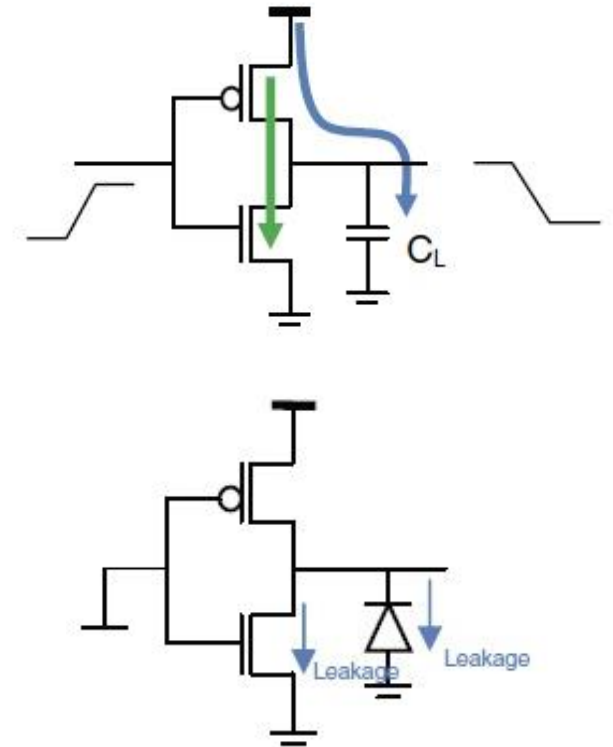
- $W_i^0 = C_i + G_i + B_i$, $W_i^{k+1} = C_i + G_i + B_i + \sum_{h=1}^{i-1} I_{i,h}$
- B_i : worst-case blocking, $I_{i,h}$: worst-case preemption by τ_h on τ_i
- **Blocking Calculation: *No* exact analysis (*pessimistic*)**
 - *request-driven* approach
 - uses **number of accelerator requests** when **task blocks**
 - *job-driven* approach
 - uses **number of jobs which arrive** during **task response time**
 - *hybrid* analysis
 - best of both worlds

Single Accelerator segment per-task

→ *request-driven* approach *dominates* the job-driven approach

CMOS Energy Model

- **Energy Model:** $P_{total} = P_{dynamic} + P_{static}$
- **Dynamic Switching Power**
 - $P_{dynamic} = K * C_L * V_{dd}^2 * f$
 - reduced using **Voltage and Frequency Scaling (VFS)**
- **Static Leakage Power**
 - $P_{static} = V_{dd} * I_{leakage}$
 - reduced using **low-power sleep states**
 - power gating and/or clock gating



Most GPUs (and hardware accelerators) only **expose** VFS to users
→ focus on **reducing frequency** to **reduce power consumption**

Related Work: RT Energy Management

- ***VFS-based***: [Saewong and Rajkumar '03][Hakan and Yang '03]
[Devadas and Aydin '10][Kandhalu et al. '11] ...
- ***Sleep-state based***: [Chang, Gabow and Khuller '12][Rowe et al. 2010]
[Fu et al. '15][Dsouza et al. 2016] ...
- **GPU Energy Management**
 - **MERLOT** [Santriaji and Hoffmann 2018]
 - hardware approach, **dynamically** exploit GPU slack to reduce frequency
 - does not consider schedulability, CPU+GPU execution

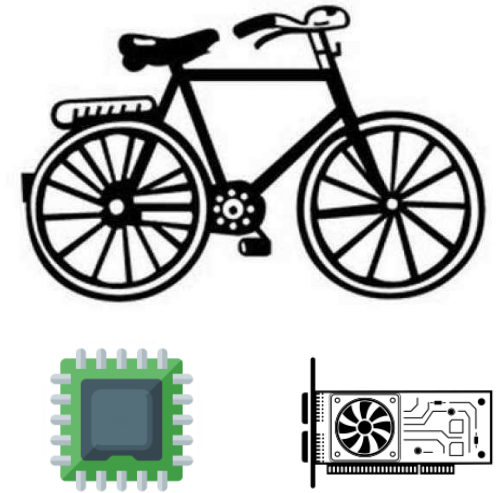
Focus on ***static frequency scaling*** (taskset-wide single frequency)
→ more ***predictable*** operation

Outline

- Motivation
- Background & Related Work
- **CycleSolo: Uniprocessor**
 - *CycleSolo Variants*
 - *SysClock [Saewong et al. 2003]*
 - *Ratchet Search*
- *CycleTandem*: Uniprocessor
- Multi-core Extensions
- Results
- Conclusion

CycleSolo: Variants

- Uniprocessor + Hardware Accelerator combination
 - only **one frequency** can be set
- **CycleSolo-CPU**
 - only the **CPU** frequency can be scaled
- **CycleSolo-Accelerator**
 - only the **accelerator** frequency can be scaled
- **CycleSolo-ID**
 - the **CPU & accelerator** can only be scaled by a **common** factor



To find the **smallest frequency**
→ Find the **maximum slack** available in the schedule

CycleSolo: Impact of Blocking



- Tasks block while *waiting* to access the *non-preemptive* accelerator
- Undefined Critical Instant
 - *Does not occur* when all high-priority tasks come in together
 - Due to blocking and self-suspensions
- Existing analysis assume the same critical instant, add pessimism by:
 - considering the *worst-case blocking*
 - modeling *self-suspensions as release jitter*
- All analyses are *pessimistic*, none are exact

*We can only find the **smallest frequency** which guarantees schedulability,
given a schedulability-analysis technique*

CycleSolo: Impact of Self-Suspensions

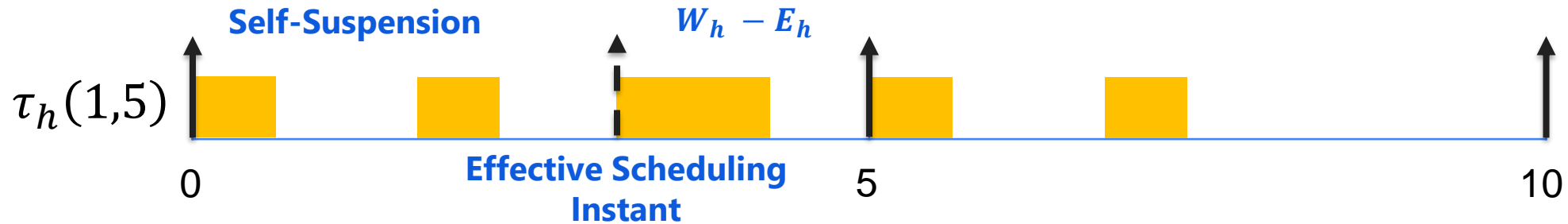
- ***SysClock*: Independent Tasks, No blocking & self-suspensions***
 - Calculate slack at each scheduling instant (**job arrivals, deadlines**)
- **In the presence of self-suspensions**
 - Slack calculation (*frequency calculation*) depends on
 - interference and blocking
 - *response time* and *WCET* of *high-priority* tasks[^]
 - *operating frequency*



In the *presence* of self-suspension
→ model the *pessimism* as *effective scheduling instants*

Effective Scheduling Instants

- $W_i^{k+1} = C_i + G_i + B_i + \sum_{h=1}^{i-1} I_{i,h}$
- **Interference of τ_h on τ_i :** $I_{i,h} = \alpha_{i,h} * E_h$
 - $\alpha_{i,h} = \text{ceil}((W_i + W_h - E_h)/T_h) \rightarrow$ depends on high-priority response time & WCET
- **For τ_i , from an analysis perspective**
 - It appears that τ_h which self-suspends arrives at:
 - $S_h := \{j * T_h - (W_h - E_h) \mid j > 0\}$



Calculate Slack at all *Effective Scheduling* Instants

CycleSolo: Key Ideas

- Find a **tight range** $[f_{low}, f_{high}]$ in which the **best frequency** f_{min} lies
- Perform a **binary search** over the range
 - Yields the **lowest frequency guaranteeing schedulability**
- **How it works:**
 - Consider tasks in **decreasing order of priority**
 - For each τ_i compute a range $[f_{low}^i, f_{high}^i]$ containing the lowest frequency
 - ensures that τ_i **and all its higher-priority tasks are schedulable**

Iteratively *refine* the range as we go through the tasks → **Ratchet Search**

Ratchet Search

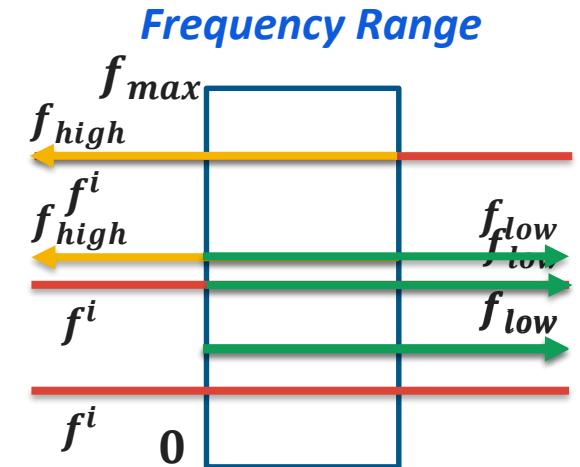
- Consider task τ_i , initial range $[f_{low}^{i-1}, f_{high}^{i-1}]$
 - Assume **high-priority tasks** are running at f_{high}^{i-1}
 - **minimizes** interference, **guarantees** schedulability
 - **Slack calculation yields frequency f^i**



τ_i **schedulable** with frequency f^i
when the high-priority interference is **minimized**

Ratchet Search: Range Update

- **Case 1:** $f^i < f_{low}^{i-1}$
 - At least one task misses deadlines at f^i
 - $f_i^{min} \in [f_{low}^{i-1}, f_{high}^{i-1}] \rightarrow$ *No change to the range*
- **Case 2:** $f_{low}^{i-1} < f^i < f_{high}^{i-1}$
 - τ_i misses deadlines at frequencies $< f^i$
 - $f_i^{min} \in [f^i, f_{high}^{i-1}] \rightarrow$ *Lower bound ratcheted up \uparrow*
- **Case 3:** $f^i > f_{high}^{i-1}$
 - τ_i not schedulable at frequencies f_{high}^{i-1}
 - $f_i^{min} \in [f_{high}^{i-1}, f^i] \rightarrow$ *Lower & Upper bounds ratcheted up \uparrow*



Range bounds always *ratcheted* up \uparrow

CycleSolo: Putting it all together

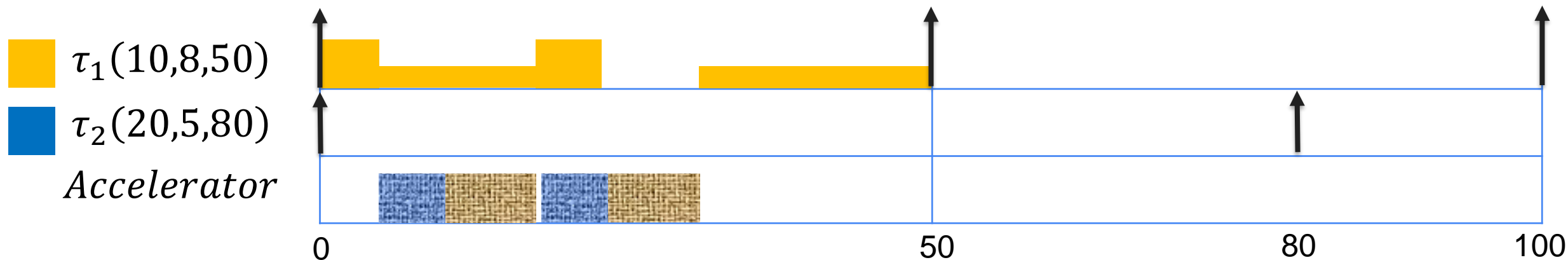
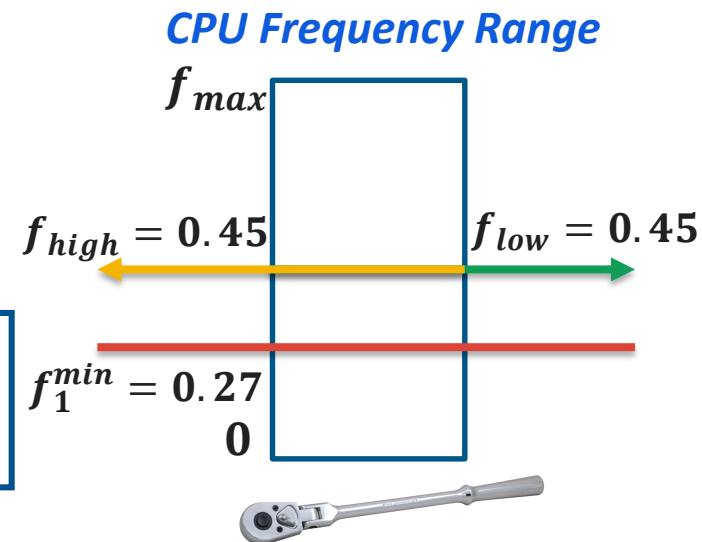
- **CycleSolo-CPU Example:** $\tau_1: (C_1 = 10, G_1 = 8, T_1 = 50), \tau_2: (C_2 = 20, G_2 = 5, T_2 = 80)$
- **Initial CPU Frequency Range set to:**
 - $[f_{low}^{init} = 0.45, f_{high}^{init} = 0.45]$ (CPU Utilization)
- **Consider effective scheduling points for τ_1**
 - $\tau_1 \rightarrow 50$

$$\text{Consider } \tau_1 \rightarrow \alpha_1^{50} = \frac{C_1}{50 - G_1 - G_2} = 0.27$$

$$f_1^{min} = \min(\alpha_1^{50}) = 0.27$$

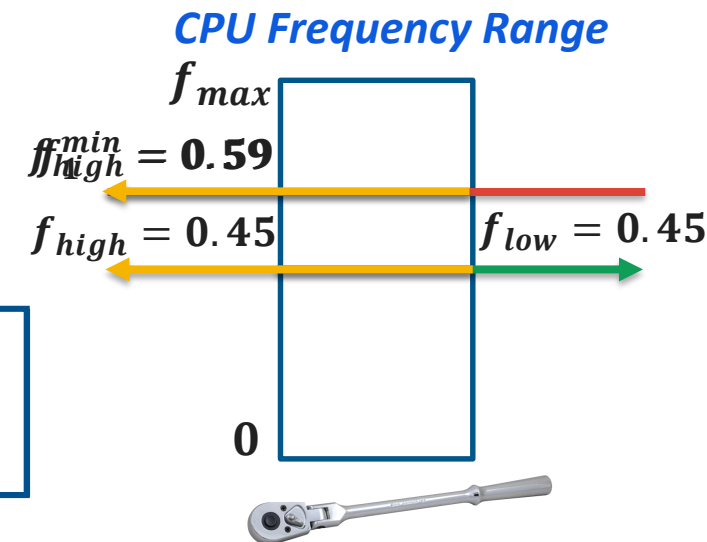
Case 1: $f_i < f_{low}$
No Change

CPU Frequency = 0.27



CycleSolo: Putting it all together



- **CycleSolo-CPU Example:** $\tau_1: (C_1 = 10, G_1 = 8, T_1 = 50), \tau_2: (C_2 = 20, G_2 = 5, T_2 = 80)$
- **CPU Frequency Range set to:**
 - $[f_{low}^0 = 0.45, f_{high}^0 = 0.45]$
- **Consider effective scheduling points for τ_2**
 - $\tau_2 \rightarrow 42$ and 80

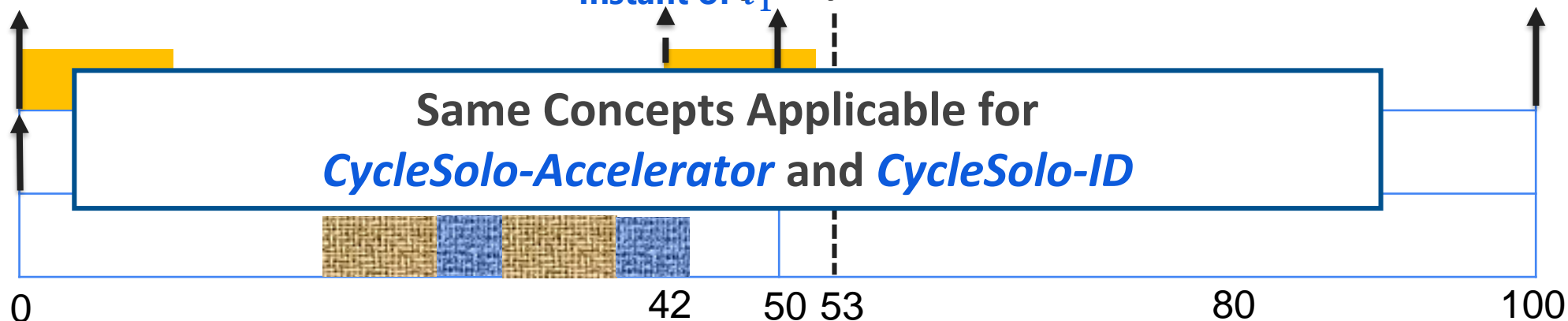


$f_{cpu}^{solo} \in [0.45, 0.59]$
Binary Search $\rightarrow f_{min} = 0.59$

Effective Scheduling
 CPU Frequency = 0.59
 Instant of τ_1

Same Concepts Applicable for
CycleSolo-Accelerator and *CycleSolo-ID*

 $\tau_1(10,8,50)$
 $\tau_2(20,5,80)$
Accelerator

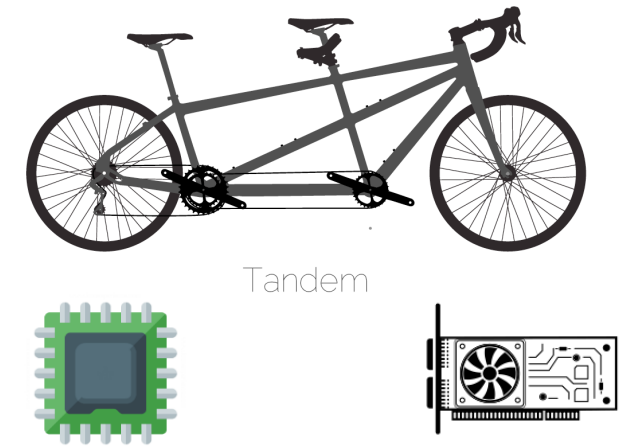


Outline

- Motivation
- Background & Related Work
- *CycleSolo*: Uniprocessor
- ***CycleTandem*: Uniprocessor**
 - ***See-Saw Theorem***
 - ***Slack Squeezing***
- Multi-core Extensions
- Conclusion

CycleTandem

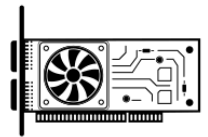
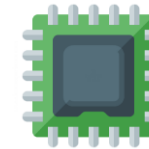
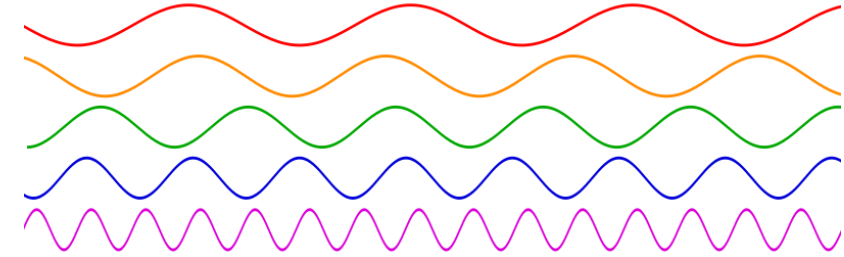
- Uniprocessor + Hardware Accelerator combination
 - each can have *their own* frequency
- Slack in the system is finite:
 - relationship between the CPU & accelerator frequency
- every CPU frequency \exists min. accelerator frequency
 - guaranteeing schedulability
 - and vice versa



To *minimize* energy consumption
→ Find the *frequency pair* (f_{cpu} , f_{acc}) which *minimizes energy*
while *guaranteeing schedulability*

Feasible Frequencies

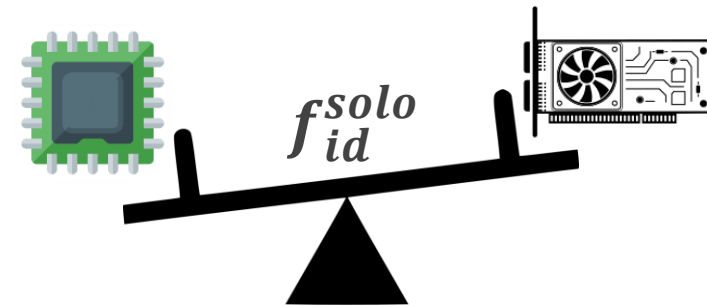
- **Lowest Feasible CPU frequency**
 - returned by CycleSolo-CPU: f_{cpu}^{solo}
- **Lowest Feasible Accelerator frequency**
 - returned by CycleSolo-Acc: f_{acc}^{solo}
- **Highest Useful CPU Frequency** f_{cpu}^{high}
 - Lowest CPU frequency corresponding to accelerator frequency f_{acc}^{solo}
- **Highest Useful Accelerator Frequency** f_{acc}^{high}
 - Lowest Accelerator frequency corresponding to CPU frequency f_{cpu}^{solo}



Energy-Optimal CPU Frequency $f_{cpu}^{opt} \in [f_{cpu}^{solo}, f_{cpu}^{high}]$
Energy-Optimal Accelerator Frequency $f_{acc}^{opt} \in [f_{acc}^{solo}, f_{acc}^{high}]$
→ **CycleSolo helps bootstrap CycleTandem**

CycleTandem: See-Saw Theorem

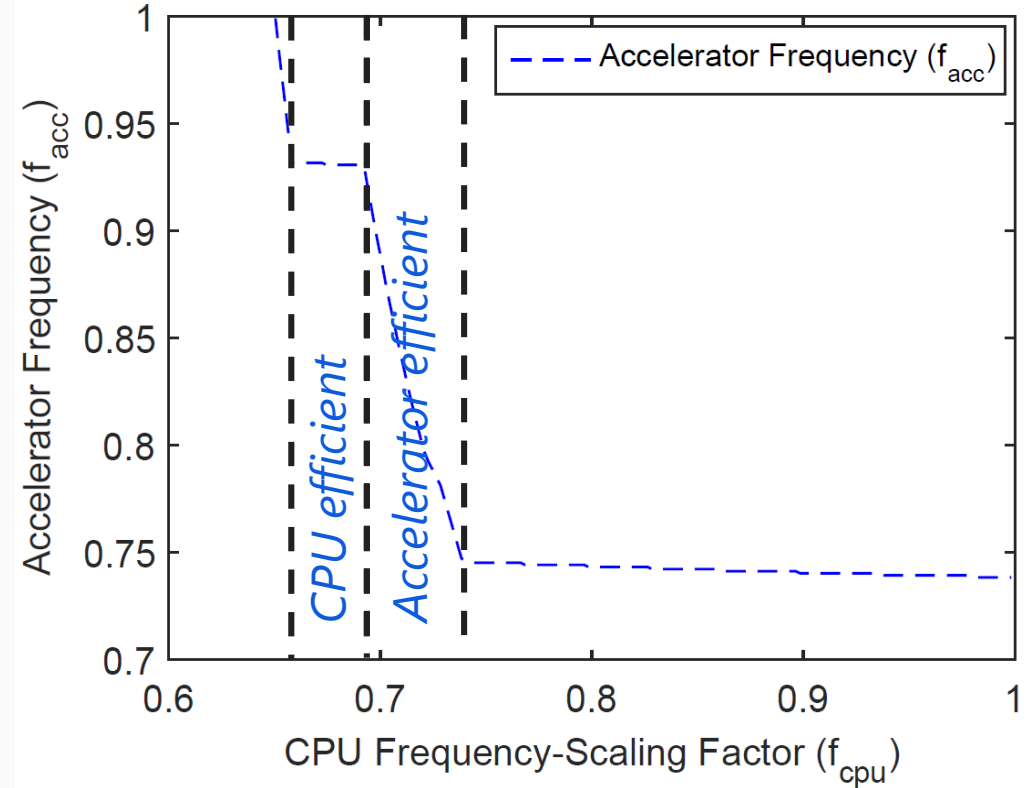
- The CPU and accelerator frequencies cannot be
 - *simultaneously lesser* than f_{id}^{solo} (CycleSolo-ID)
 - to *ensure schedulability*
- f_{id}^{solo} : CycleSolo-ID Frequency
 - *Lowest Common* CPU and Accelerator Frequency
- CPU frequency increases
 - Accelerator frequency decreases
 - and vice versa
- Mapping between $f_{cpu} \rightarrow f_{acc}$ (and vice versa)



*Find the optimal CPU frequency f_{cpu} or Accelerator frequency f_{acc}
which minimizes energy \rightarrow search the feasible range*

CycleTandem: Slack Squeezing

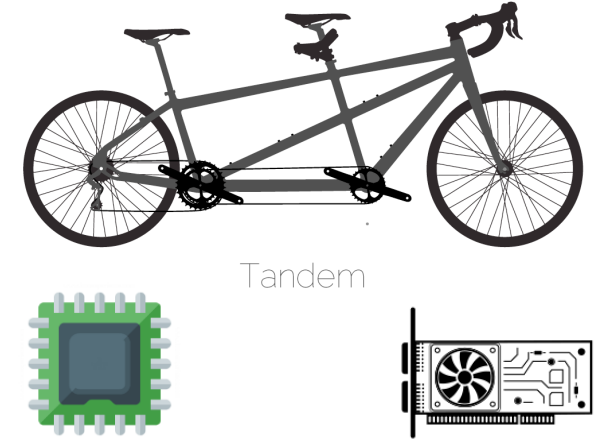
- For a **given δ increase** in CPU (accelerator) frequency f_{cpu} (f_{acc})
- if the accelerator (CPU) frequency **decreases by $\delta' > \delta$**
- then the accelerator (CPU)
 - **Squeezes** slack more efficiently than the CPU (accelerator)



Energy is a **non-linear** & **non-convex** function of the accelerator & CPU frequencies

CycleTandem: Key Idea

- **Non-Linear Non-Convex Energy Function**
 - **difficult** to obtain the best solution
 - **search** the feasible range using a heuristic
- **How it works:**
 - Compute the **feasible CPU frequency range** (CycleSolo-CPU)
 - Compute the **feasible Accelerator frequency range** (CycleSolo-Accelerator)
 - Search (heuristic or brute force) over the **smaller** of the two ranges



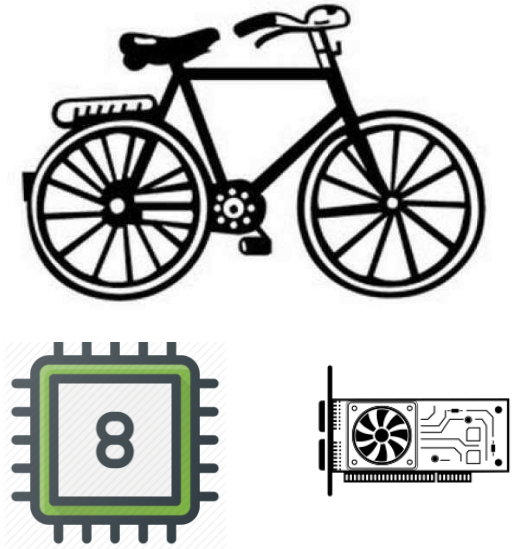
Greedy-search heuristic: (1) Choose the endpoint with the **lower energy**,
(2) increase/decrease the frequency **till a local minimum is reached**

Outline

- Motivation
- Background & Related Work
- *CycleSolo*: Uniprocessor
- *CycleTandem*: Uniprocessor
- **Multi-core Extensions**
 - *Assumptions*
 - *Extensions*
 - *SyncAware-WFD*
- Results
- Conclusion

Multi-Core Assumptions

- *all CPU cores* are in the *same power domain*
- can only be set to the *same frequency*
- *fully-partitioned* scheduling

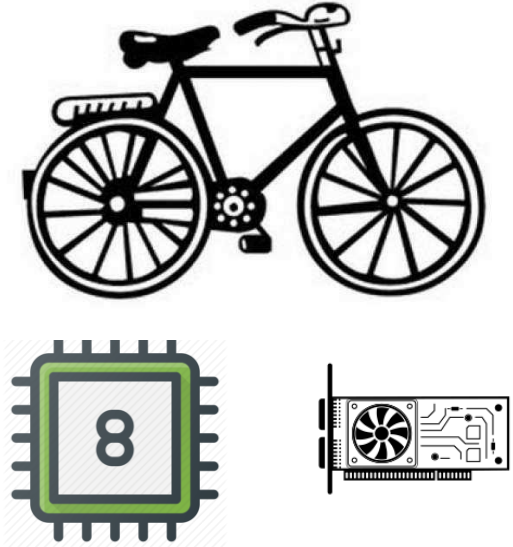


If each core has its own frequency

- *Response Time* of task on a core can depend on *frequency of another core*
- *impact of self suspensions on remote blocking*

Multi-Core Extensions

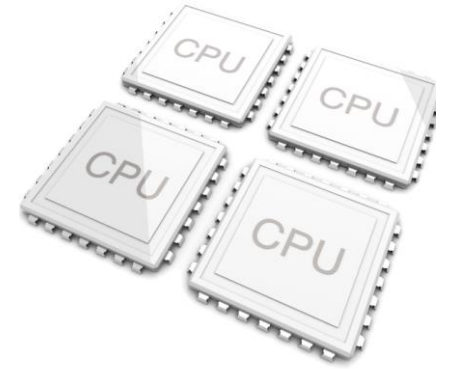
- Key Observations:
 - *RatchetSearch* still applicable → CycleSolo works
 - *See-Saw Theorem* still holds → CycleTandem works
- Difference:
 - *Interference* only by *tasks on the same core*
 - *Remote Blocking* by *tasks on other core*



CycleSolo & CycleTandem can be used for Multi-Core Processors,
which have a *single power domain* across all CPU cores

The Impact of Task Partitioning

- All CPU cores can only be set to the same frequency
 - *Load balancing* is useful
 - *Worst-Fit Decreasing (WFD)* known to yield **balanced partitions**
- **Blocking & Self-Suspensions**
 - *Effects* can be felt by tasks on other cores
- **Sync-Aware WFD:** based on [Lakshmanan et al. 2011]
 - **Load Balance** while *constraining* self-suspending tasks to $\psi = \text{ceil}(\gamma_{acc} * m)$ cores
 - γ_{acc} is the fraction of the CPU load belonging to self-suspending tasks



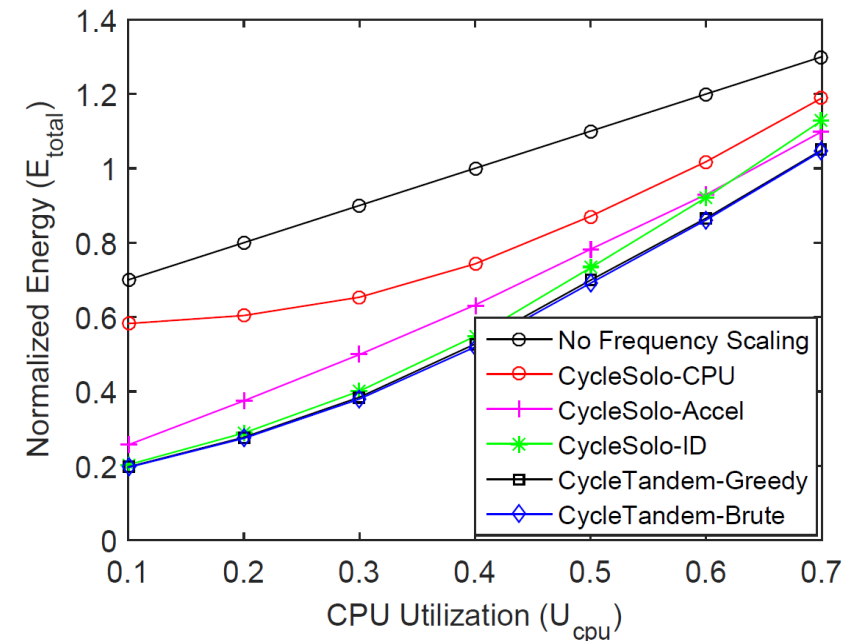
Sync-Aware WFD can *restrict* the *schedulability pessimism* of blocking and self-suspensions to a few cores

Outline

- Motivation
- Background & Related Work
- *CycleSolo*: Uniprocessor
- *CycleTandem*: Uniprocessor
- Multi-core Extensions
- **Results**
 - *Analytical Results*
 - *TX2 Results*
- Conclusion

Analytical Results

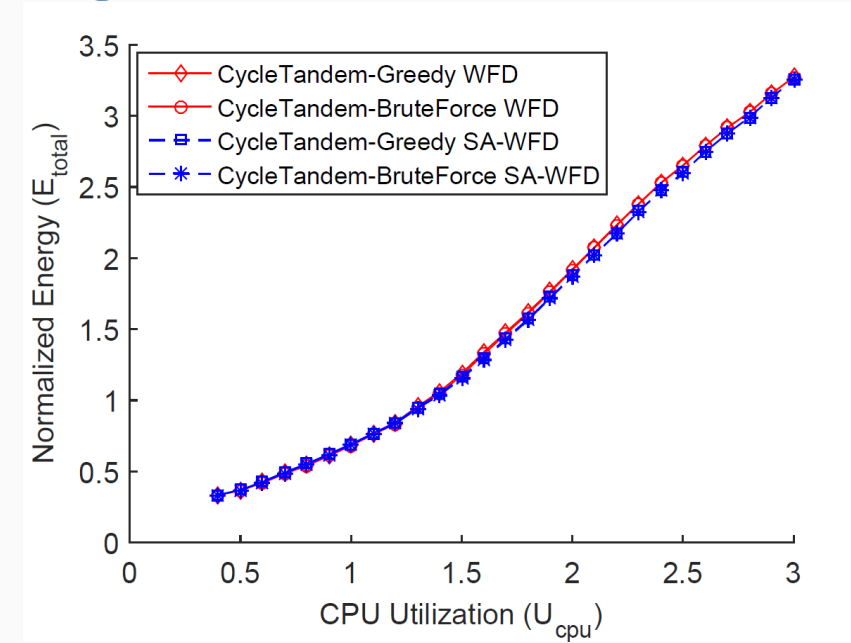
- Randomly-generated Tasksets: *UUnifast-Discard*
- Energy: Compared to without Energy Management
 - *CycleTandem* : up to 71.88% lower
 - *CycleSolo-ID* : up to 71.03% lower
 - *CycleSolo-Accel* : up to 63.41% lower
 - *CycleSolo-CPU* : up to 27.42% lower
- CycleTandem greedy-search heuristic vs brute force:
 - Worst-case 1.53% greater energy than brute force



CycleSolo and CycleTandem can deliver *significant* energy savings

Multi-core Results: Partitioning

- 4-core processor considered
- CycleTandem: WFD vs SyncAware-WFD
 - **Schedulability** : SyncAware-WFD 6.3% more tasksets
 - **Energy-Savings**: SyncAware-WFD up to 3.3% greater



SyncAware-WFD yields *marginally better* schedulability and energy savings than WFD, with the *same algorithmic complexity*

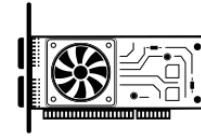
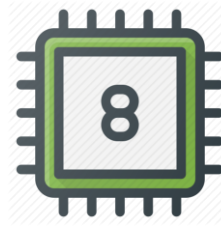
Real-Platform Evaluation: NVIDIA TX2

- **4-core ARM processor, 256-core Integrated GPU**
 - CPU cores can only be set to the *same frequency*
 - GPU frequency can be set *independently*
- **Energy: Compared to without Energy Management**
 - *CycleTandem* : up to 44.29% lower → 1.78x battery life
 - *CycleSolo-ID* : up to 44.18% lower
 - *CycleSolo-Accel* : up to 7.81% lower
 - *CycleSolo-CPU* : up to 32.01% lower



***Significant* real-world energy savings are possible**

Conclusion



- **CycleSolo: “Slowest Speed is Best”**
 - scenarios where *only the CPU/accelerator/common frequency* can be set
 - **slack computation** in the presence of *blocking* and *self-suspensions*
- **CycleTandem: “Better frequency pair, lower energy”**
 - CPU and accelerator frequency can be *independently* set
 - *non-convex* optimization problem: *search* the feasible range
- **Multi-Core Extensions:**
 - for scenarios where all CPU cores have the *same frequency*
 - CycleSolo & CycleTandem are still *applicable*

Analysis framework for *Energy-Saving* Scheduling with *Hardware Accelerators*
→ lays the groundwork for analyzing more *complex* scenarios

Thank You ! Questions ?