COL862: Low Power Computing

Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques

Authors: Huazhe Zhang and Henry Hoffmann, Published: ASPLOS '16 Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating System.

- Presented By: Suman M, Radhika D.

Paper Overview

- Explores the Tradeoffs between Timeliness and Efficiency
- Proposes PUPiL (Hybrid) approach that includes both software and hardware components
- Develops a decision framework to maximize performance under a power cap.
- Experimental Evaluation of the PUPiL approach for Single-Application and Multi-Application workloads
- Comparative analysis with other Hardware and Software Techniques

Motivational Example

- Comparison between Hardware based (RAPL) and Software based (Soft-Decision)
- Test Setup is to run a x264 Video encoder in a Intel Linux/x86 system
- Performance is measured as frames processed per second and plotted for both hardware and software approach

- Specifications
  - Sandy Bridge Xeon E5 Processor
  - Dual Socket
  - 8 cores each with hyper-threading provides 32 cores
  - 64 GB RAM
  - TDP - 135W
  - 15 frequency settings

- Hardware approach:
  - RAPL's only mechanism for power enforcement is processor voltage and frequency
  - Completely based on Hardware Performance counters and power estimation models by Intel
- Software Approach:
  - It can configure following parameters:
    - How many sockets to use
    - How many cores to use on each socket
    - Whether to use hyper-threads or not
    - How many memory controllers to use
    - The frequency of each socket

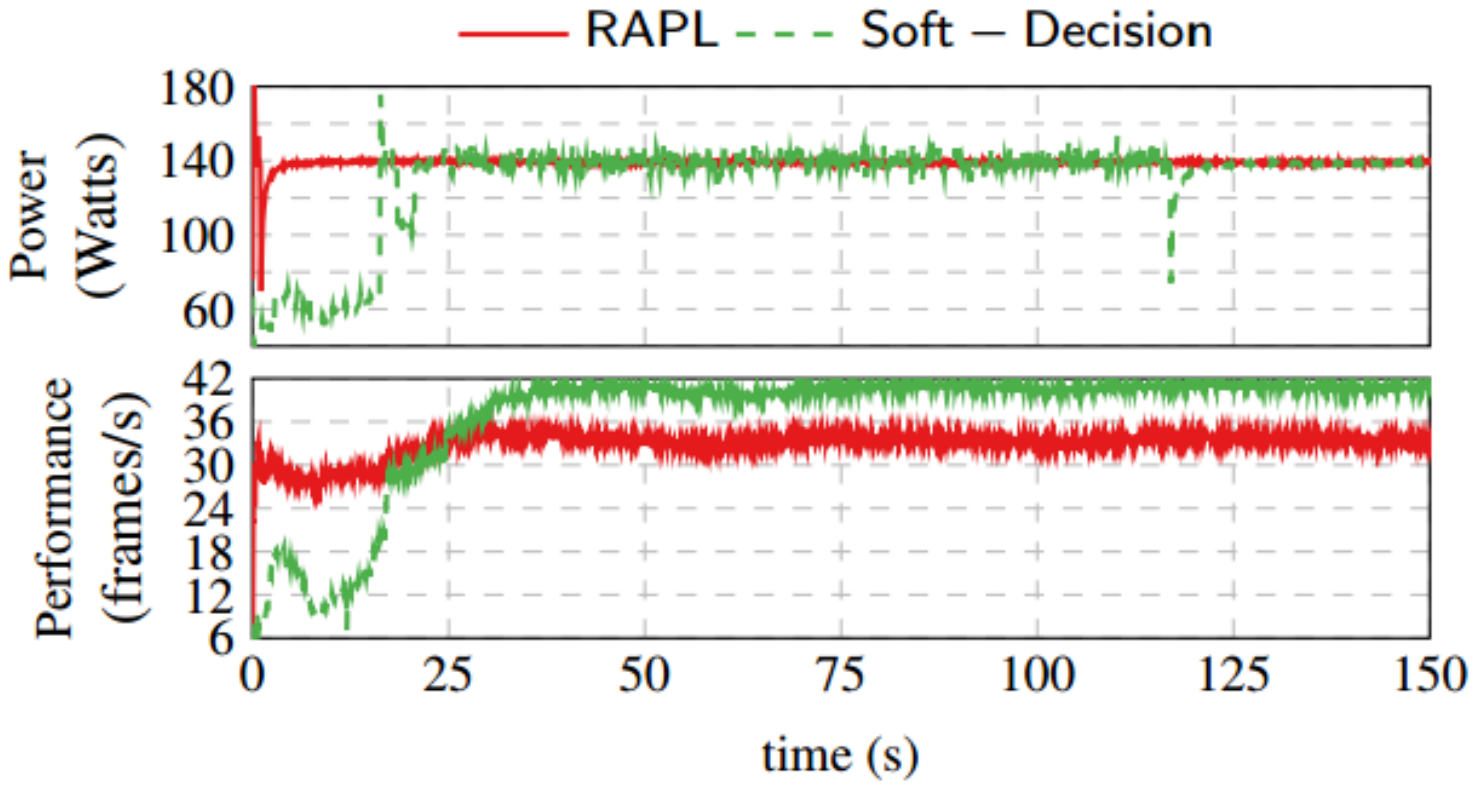


Figure 1. Tradeoff between timeliness and efficiency from hardware and software power capping, running x264.

Observation

- Performance:
  - RAPL averages approximately 33.5 frames per second
  - Once the software approach converges, it delivers 20% more performance than RAPL approximately 41 frames per second
- Power:
  - RAPL hits the cap quickly
  - Software approach operates below the cap for approximately 20 seconds and then converges

- Software outperforms hardware because it recognizes that hyperthreads do not help this application.
- The software approach recognizes that it should not make use of hyperthreads and instead it increases the speed of the cores it is using without hyperthreads.
- These results demonstrate the need for a hybrid approach that enforces power caps with hardware's speed, but has software's flexibility to adapt resource usage.

- Power Capping Methodologies
  - Hardware Power Capping - RAPL
  - Software Power Capping
  - PUPiL (Hybrid) Power Capping
- All three approaches operate on feedback and *observe-decide-a*ct Framework
- observe their environment, decide on a response, and act to implement their decisions

Software Power Capping

Observe

- Collection of Power and Performance Feedback
  - Power feedback can come from power monitoring mechanisms like
    - external power meters such as a WattsUp,
    - on-board power monitoring devices,
    - or on-chip power monitoring
    - through research prototypes.
  - Performance feedback sources:
    - Instrumented applications
    - Hardware counters like floating point computation rate or simply instructions per second

- Problem with feedback is that real systems are noisy.
- A power capping system should ensure that it is reacting to persistent phenomena and not some transient effect that momentarily disturbs performance
- Solution :
  - a deviation based filter to remove outliers

- The software approach measures performance over a window, filters any data that falls more than 3-standard deviations from the mean, and averages the rest

$$\begin{aligned} \mu &= \frac{\sum_i X_i}{N} \\ \sigma &= \sqrt{\frac{\sum_i (X^i - \mu^2)}{N}} \\ X_{feedback} &= \frac{\sum_{j \in A} X_j}{size(A)} \\ A &= \{j \mid |X_j - \mu| < 3\sigma\} \end{aligned}$$

- X is the list of performance measurements collected
- μ is the average of unfiltered X
- σ is the standard deviation of unfiltered X
- $X_{feedback}$ is the performance feedback

Decide

- In the decide phase, the software selects a resource configuration.
- One possibility is **Exhaustive search** until we find the highest performance configuration that respects the power cap
- Drawback :
  - Fails to meet the timeliness challenge
  - It may fail to respect the power cap
  - As Resources increases the configuration grows exponentially

Decision Framework Algorithm

- The system first orders the available resources
- It then starts in the lowest resource configuration.
- Put the next resource into its highest setting. Feedback is measured in this new configuration.
- The software compares the performanc feedback of the current configuration tc that of last configuration to decide whether
  - 1) performance has improved by using this new resource and
  - 2) the resource usage respects the power cap
- The order is determined by Order() (Algorithm 2)

Algorithm 1 Walking the decision framework.

Require: Set of ordered resources R
Require: Power cap P
Put system in minimal resource configuration
$U \leftarrow R$ ▷ the set of untested resources
while $U \neq \emptyset$ **do** ▷ While untested resources
$\langle perf_{old}, pow_{old} \rangle \leftarrow$ GetFeedback()
$r \leftarrow$ RemoveNext(U) ▷ next resource in order
set r to highest setting
wait $r.d$ time units ▷ Account for resource delay
$\langle perf_{cur}, pow_{cur} \rangle \leftarrow$ GetFeedback()
if $perf_{cur} < perf_{old}$ **then**
return r to lowest setting
else
if $pow_{cur} > P$ **then**
$s \leftarrow$ BinarySearchResourceSettings(r)
set r to s
▷ This may return the resource to its lowest setting.

Decision Framework Algorithm

- While this ordered set of untested resources is non-empty, the algorithm measures power and performance
- It then takes the next resource in order and sets it to its highest configuration setting.
- Waits a resource specific amount of time, and then measures the feedback again
- If this resource provided higher performance then
  - the algorithm fine tunes the resource setting,
- Else
  - it returns to the lowest setting for this resource
- The fine tuning process involves performing a binary search on resource settings to find the highest performance setting that is under the power cap

Algorithm 1 Walking the decision framework.

Require: Set of ordered resources R
Require: Power cap P
Put system in minimal resource configuration
$U \leftarrow R$ ▷ the set of untested resources
while $U \neq \emptyset$ **do** ▷ While untested resources
$\langle perf_{old}, pow_{old} \rangle \leftarrow$ GetFeedback()
$r \leftarrow$ RemoveNext(U) ▷ next resource in order
set r to highest setting
wait $r.d$ time units ▷ Account for resource delay
$\langle perf_{cur}, pow_{cur} \rangle \leftarrow$ GetFeedback()
if $perf_{cur} < perf_{old}$ **then**
return r to lowest setting
else
if $pow_{cur} > P$ **then**
$s \leftarrow$ BinarySearchResourceSettings(r)
set r to s
▷ This may return the resource to its lowest setting.

Ordering Algorithm

- The software approach establishes the ordering based on the potential impact of each resource.
- Higher impact resources have precedence over lower impact resources.
- Allocate power first to higher impact resources so that we can tune the performance from coarse-grained knobs to fine-grained knobs
- Impact of a resource by the performance improvement that it delivers when activated individually
- Impact is determined by calibrating the system using a well-understood, embarrassingly parallel application

Algorithm 2 Ordering Resources in Calibration.

Require: Set of resources R excluding DVFS
Require: a calibration benchmark without inter-thread communication
Put system in minimal resource configuration
$U \leftarrow R$ ▷ the set of disordered resources
while $U \neq \emptyset$ **do** ▷ While disordered resources
$r \leftarrow \mathsf{RemoveNext}(U)$ ▷ next resource in random order
set r to highest setting
wait $r.d$ time units ▷ Account for resource delay
$perf_r \leftarrow \mathsf{GetFeedback}()$
return r to lowest setting
add r to O
Sort r in O by $perf_r$
Add DVFS to the last in O **return** O ▷ The set of ordered resources

ACT

- The act phase, the software implements the resource allocation proposed by the decision phase
- The software requires two pieces of external information
  - **Timing information** about how long to expect from when the resource is allocated to when its effects can be observed.
  - A **function** that implements the resource allocation to maintain generality of the approach

ACT

- Simply -
  - Set the **resource configuration** to that specified by the decision phase
  - Put the decision framework to **sleep** for the time it will take to see the resource effects.
  - To increase efficiency, the software **keeps track** of the previous resource allocation and only changes those resource settings which changed since the last decision

Hardware Power Capping

- Power Capping is based on the RAPL counters
- Power limit (cap) and time interval for RAPL is set through MSRs.
- RAPL observes various low-level hardware events and estimates power consumption from those event counts.
- It then determines an energy budget that would meet the desired power cap during the specified time interval.

Hardware Power Capping

- RAPL sub-divides the user-specified time interval into a set of smaller intervals.
- For each of sub interval it then calculates the remaining energy budget for the remaining time in the user-specified interval.
- Based on this the best possible processor speed and voltage is choosen
- RAPL then sets DVFS accordingly and wait for the next sub interval.

Hardware Power Capping

- RAPL observes only power feedback and not performance.
- Decisions are made by solving a linear equation, and acts by only tuning voltage and frequency only.
- All three steps (feedback/decision making and action) can be done within milliseconds and this ensures the timeliness of hardware approach.
- But RAPL lacks performance feedback and considers only DVFS, thus hardware approach cannot deliver the highest performance for many applications

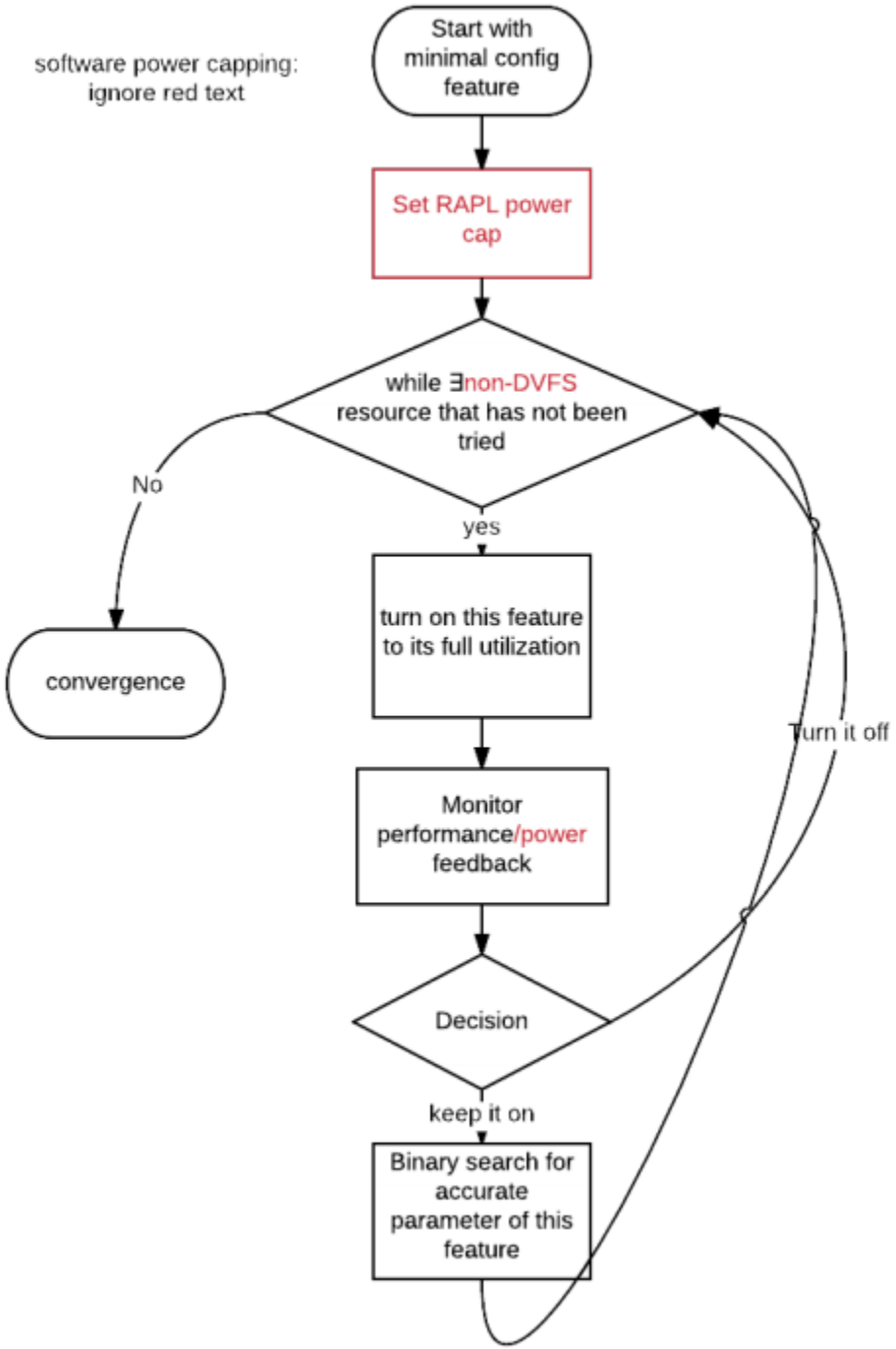
PUPiL

Obtain the efficiency of the software approach and the timeliness of hardware approach

Hybrid Approach:

- **Hardware power capping**(RAPL) approach has to be in charge power capping.
- Removed processor **speed** and **voltage** from the set of resources controlled by software.
- Power distribution among different chips in a multi-socket environment needs to be considered.
- It uses a core-number based power distribution across different chips (dynamic power dist.)

Flow Chart for PUPiL Approach



Experimental Setup

- Benchmarks
  - 20 benchmark applications from three different suites
    - PARSEC (x264, swaptions, vips, fluidanimate, blackscholes, bodytrack)
    - Minebench (ScalParC, kmeans, HOP, PLSA, svmfe, btree, kmeans fuzzy).
    - Rodinia (cfd, nn, lud, particlefilter),
    - partial differential equation solver (jacobi)
    - swish++ search webserver
    - dijkstra

Experimental Setup

- Platform Details:
  - SuperMICRO X9DRL-iF motherboard
  - 2 Xeon E5-2690 processors
  - Linux 3.2.0.
  - 8 cores
  - TDP – 135 W
- Resource Configurability - Supports ***1024*** user-accessible configurations with its own power/performance tradeoffs
  - 15 DVFS settings
  - Hyper-threading
  - TurboBoost
  - *msr* module for RAPL Settings
  - *cpufrequtils* for clock speed
  - numactl for Memory control

Table 2. System configurations.

Configuration	Settings	Max Speedup	Max Powerup
cores per socket	8	7.9	2.1
sockets	2	2.0	1.7
hyperthreading	2	1.9	1.2
mem controllers	2	1.8	1.1
clock speeds	16	3.2	3.4

Experimental Setup

- Evaluation Metrics
  - Evaluate the timeliness and efficiency of various power capping approaches
    - Timeliness – measured through *settling time*
    - Efficiency - measured by performance achieved by a workload under a power cap. It is measured as the *weighted speedup.*

Methods Compared

- **RAPL**: Hardware Approach
- **Soft-DVFS**: This is a software approach that sets the DVFS settings using the cpufrequtils package.
- **Soft-Modeling**:
  - Predictive Model - uses multiple regression to estimate the power and performance of an application as a function of assigned resources (clock speed, memory controllers, sockets, cores per socket and hyper-threads)
  - No Feedback Information
- **Soft-Decision**: This is the software-only decision framework discussed previously
- **Optimal:** This is determined by running each application in every possible system configuration and measuring its performance. The optimal configuration achieves the best speed for a given power cap.

Experiments

- Experiments are divided into two category
  - Single application workloads
  - Multi-application workloads
- Single Application Workloads
  - Each application is launched under a power cap and both performance and settling time is measured.
  - Evaluation is done for 5 different processor power caps: 60, 100, 140, 180 and 220 watts

Experiments : Single application workloads

Table 3. Comparison of Harmonic Mean Performance.

Power Cap	RAPL	Soft-DVFS	Soft-Modeling	Soft-Decision	PUPiL
60W	.54	-	-	.70	.71
100W	.68	.66	.66	.80	.85
140W	.74	.71	.65	.87	.89
180W	.78	.74	.76	.88	.92
220W	.79	.75	.85	.91	.94

- When setting power cap for RAPL Soft-DVFS, the power budget is divided equally among the sockets as that is the optimal setting.
- Soft-Decision and PUPiL decision is taken as they see the fit when the threads are migrated.
- For Soft-DVFS and Soft-Modelling there is not data for 60W power cap since the minimum P-state exceed 60W power cap when using all cores and hyperthreads.

Experiments : Single application workloads

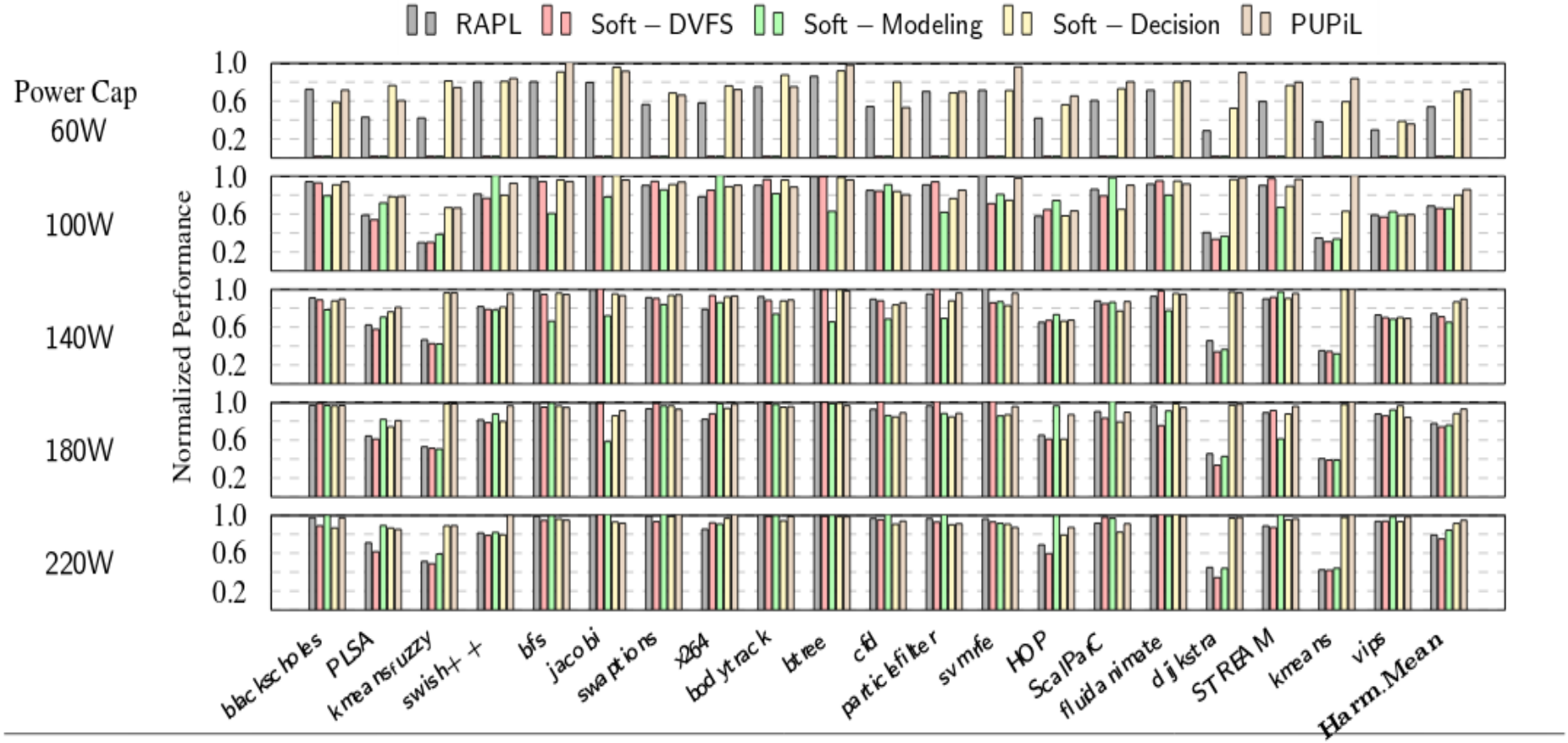


Figure 3. Performance of several power control techniques normalized to optimal.

Experiments : Single application workloads

- In general Soft-Decision provides higher performance than RAPL
- While Soft-DVFS and Soft-Modeling are comparable to RAPL
- Results shows that in majority of the benchmarks PUPiL's performance is better to other appraches
- Soft-Modeling uses historical power data and configures the machine based on the predicted power. Thus results are unpredictable since configuration is not based on feedback.
- For some applications and power cap like HOP, swish++ at 100W it out performs all the other approaches.
- Yet the average performance of this is still not good when compared to Soft-Decision and PUPiL.
- While RAPL performs well on some applications like btree and svmfe and poorly on others like dijkstra and kmeans.
- Soft-Decision is very close to PUPiL.
- Thus these result confirm that approaches that manage multi-resource out perform systems that only manipulate DVFS (software (soft-DVFS) or hardware (RAPL)).

Experiments : Single application workloads

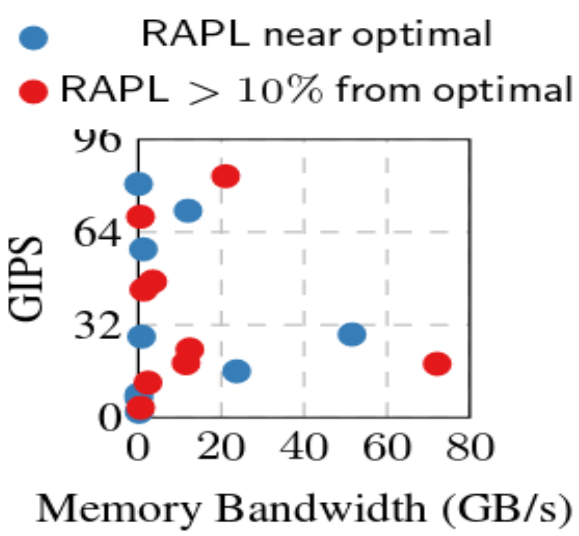


Figure 5. Benchmark characteristics.

- Notions like memory-bound or compute bound are not good predictors of RAPL efficiency.
- Since RAPL performs poorly on STREAM, which has the highest memory bandwidth, yet does well with jacobi which has the second highest memory bandwidth.
- It generally performs well for applications that have ample parallelism and scale well to use all 32 virtual cores poorly on applications with scaling issues or limited parallelism.

Experiments : Single application workloads

- Kmeans works well with more cores on a single socket (limited parallelism).
- While using cores on both the socket, inter socket communication creates bottleneck.
- To handle this RAPL and Soft-DVFS reduce clock speed to meet the power cap which hits performance.
- But Soft-Decision and PUPiL recognize that the second socket decreased performance nd they restrict kmeans to a single socket and increase its speed, which results in higher performance

Experiments : Single application workloads

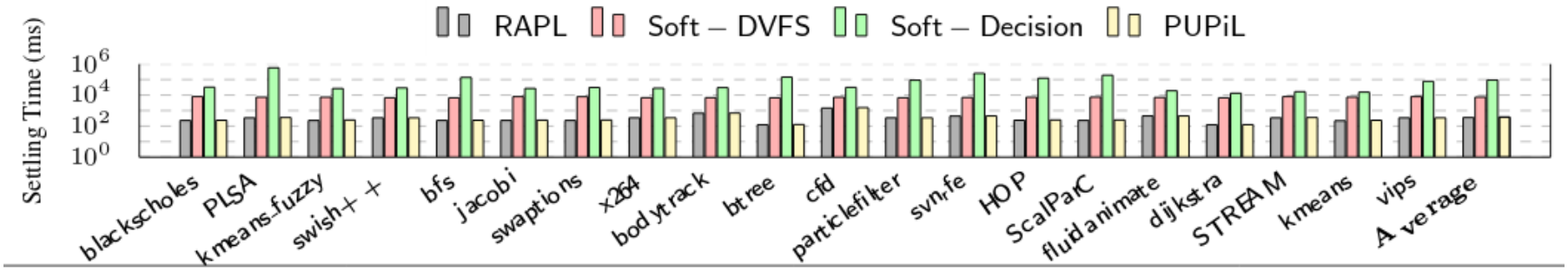


Figure 4. Settling times for several power control techniques.

- Soft modelling is offline approach so no setting time is noted.
- The settling times shown for all approaches and applications are for 140 Watt cap.
- RAPL shows the hardware timeliness advantage over Soft-DVFS.
- PUPiL is in par with RAPL. However the small overhead is due to the fact that PUPiL's software controls power cap rather than directly setting the register in hardware.
- Thus PUPiL exhibits the performance of Software approach and timeliness of Hardware approach.

Experiments:Multi-Application Workloads

Table 4. Multi-application Workloads.

Name	Benchmarks
mix1	jacobi, swaptions, bfs, particlefilter
mix2	cfd, bfs, fluidanimate, jacobi
mix3	blackscholes, cfd, jacobi, fluidanimate
mix4	particlefilter, blackscholes, swaptions, btree
mix5	x264, dijkstra, vips, HOP
mix6	STREAM, fuzzy-kmeans, HOP, dijkstra
mix7	STREAM, kmeans, vips, HOP
mix8	kmeans, dijkstra, x264, STREAM
mix9	jacobi, swaptions, fussy-kmeans, vips
mix10	cfd, bfs, x264, HOP
mix11	jacobi, blackscholes, dijkstra, fuzzy-kmeans
mix12	btree, particlefilter, kmeans, STREAM

- Benchmarks are divided into two sets : ones for which RAPL delivered near-optimal performance (blue dots) and ones for which RAPL is more than 10% from optimal.
- Then multiapplication workloads by randomly selecting applications from the two sets.
- For the first four mixes (1–4), all applications are drawn from the blue set.
- The mixes 5–8 are all applications taken from red set
- The applications in mixes 9–12 include two applications from each set.

Experiments:Multi-Application Workloads

- Two separate secenarios are studied:
  - Co-operative
    - All applications know that they are running with other applications;
    - Each is launched with only 8 threads, so that the total number of active threads is equal to the number of virtual cores.
  - Oblivious
    - Each application is launched without regard to the other applications in the system.
    - Each requests 32 threads, So total of 128 treads are alive in the system
- Performance of PUPiL to RAPL is considered.

Experiments : Multi-Application Workloads

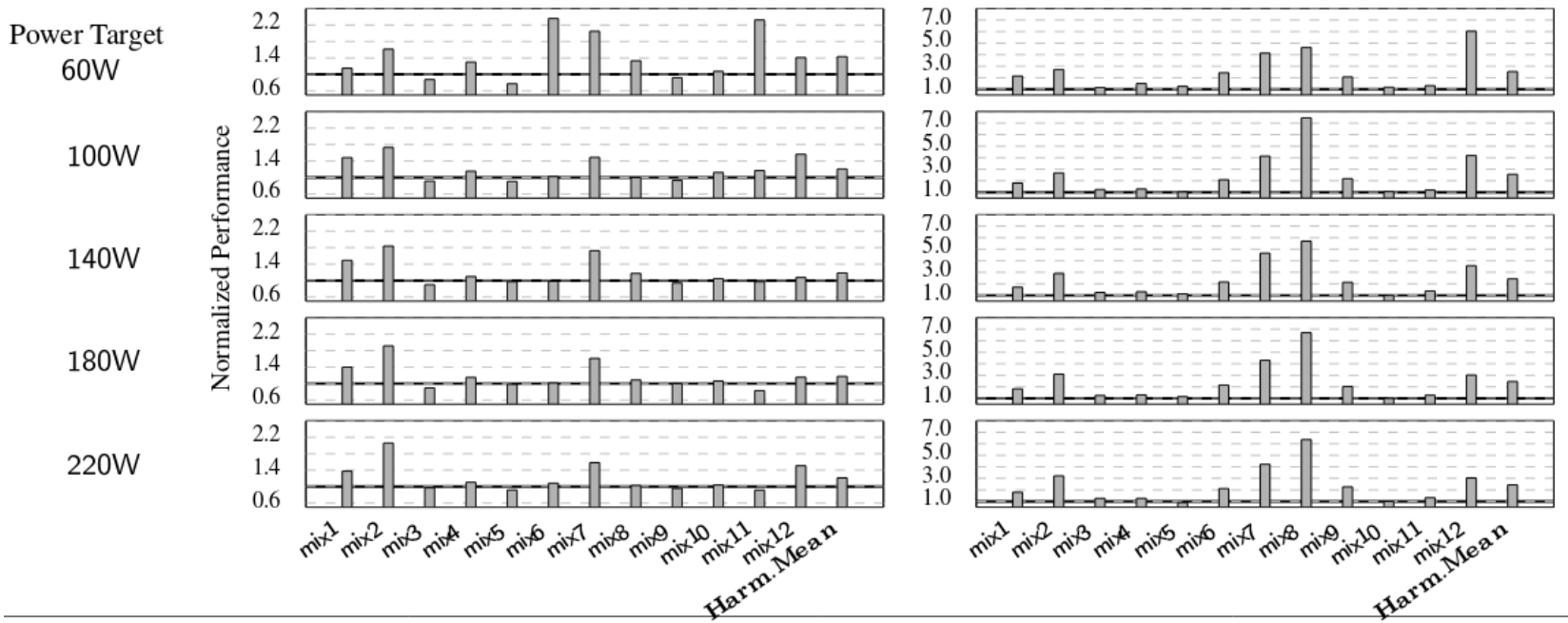


Figure 6. Ratio of PUPiL to RAPL performance in cooperative (left) and oblivious (right) multiapp scenarios.

Table 5. Ratio of PUPiL to RAPL Performance.

Power Cap	Cooperative	Oblivious
60W	1.43	2.53
100W	1.21	2.56
140W	1.18	2.44
180W	1.18	2.46
220W	1.21	2.43

Experiments : Multi-Application Workloads

- In the cooperative scenario, PUPiL outperforms RAPL by atleast 18% across all power budgets.
- Single-application performance is not necessarily a good indicator of multi-application performance. Where RAPL out performed PUPiL for some of the workloads.
- Since, though Mix 2 had all the application from blue set, PUPiL outperformed RAPL.
- The result shows that multi-application workloads can have complicated behavior and it justifies the need for an adaptive approach, like PUPiL.

Experiments : Multi-Application Workloads

- In case of oblivious multi-application workload, every application is trying to claim as many resources as possible.
- In such cases RAPL by itself cannot provide high performance under the power cap.
- Thus a flexible system like PUPiL is needed to carefully manage resource usage and deliver high performance under the power cap.
- PUPiL's higher performance is mainly due to the non computational resource bottle neck like inter socket communication.

Experiments : Multi-Application Workloads

Table 6. PUPiL and RAPL Multiapp Performance.

Workload	Spin Cycles (%)		Memory Bandwidth (GB/s)	
	RAPL	PUPiL	RAPL	PUPiL
mix7	15	0.23	14.6	23.8
mix8	54	.48	17.5	30.3
mix12	33	.40	14.3	27.0

- Mix 7, Mix 8 and Mix 12 shows that PUPiL out performs RAPL by greatest amount.
- Vtune was used to collect data on applications, and Spin cycles and memory BW stood out.
- Under RAPL control these mixes spend large portion of time spinning and achive very less memory BW
- Assumption is that one of the application in these mixes use polling synchronization and when it gets CPU holds it for the entire quantum.
- When mix is scheduled on fewer cores such that they run on single socket, the polling benchmark
  - Has much less contention
  - Finshes work faster
  - Yeilds core to other apllications

Experiments: Energy efficiency

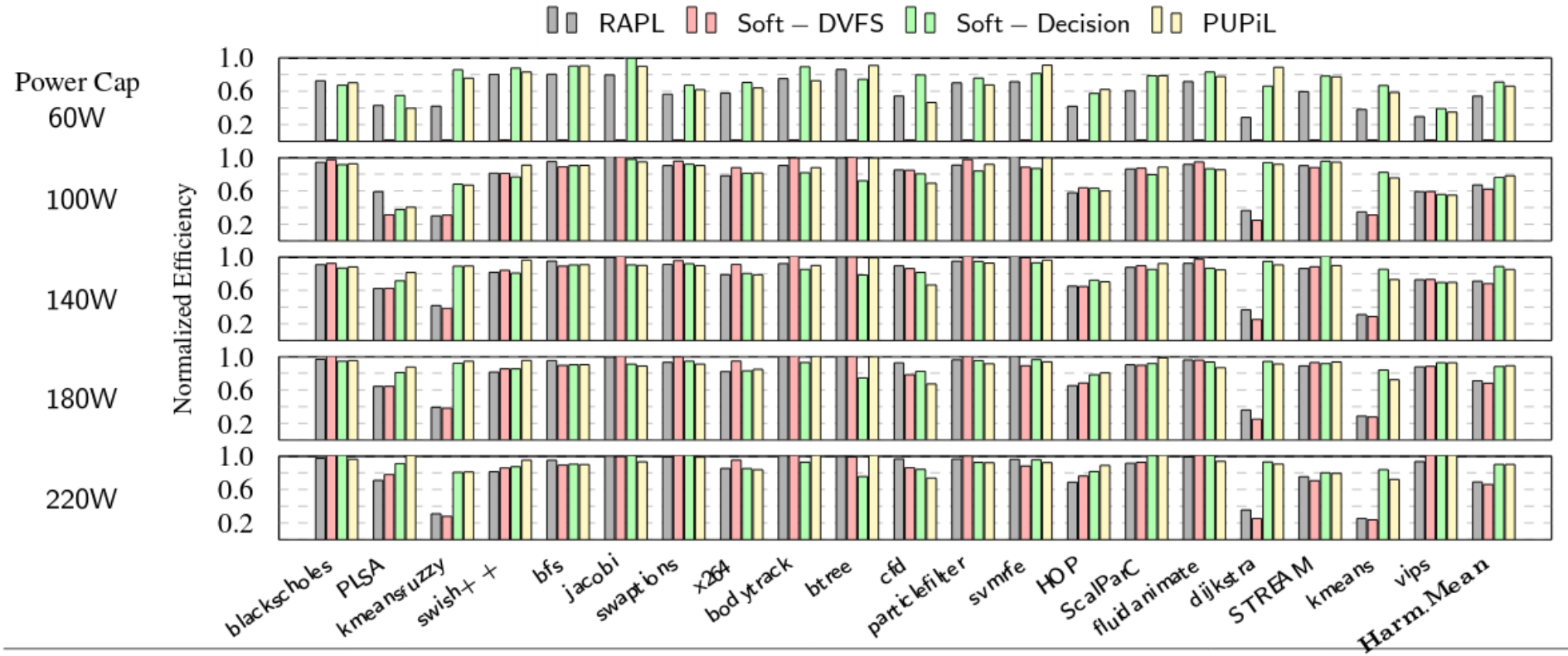


Figure 7. Energy efficiency of several power control techniques normalized to optimal.

Experiments: Energy efficiency

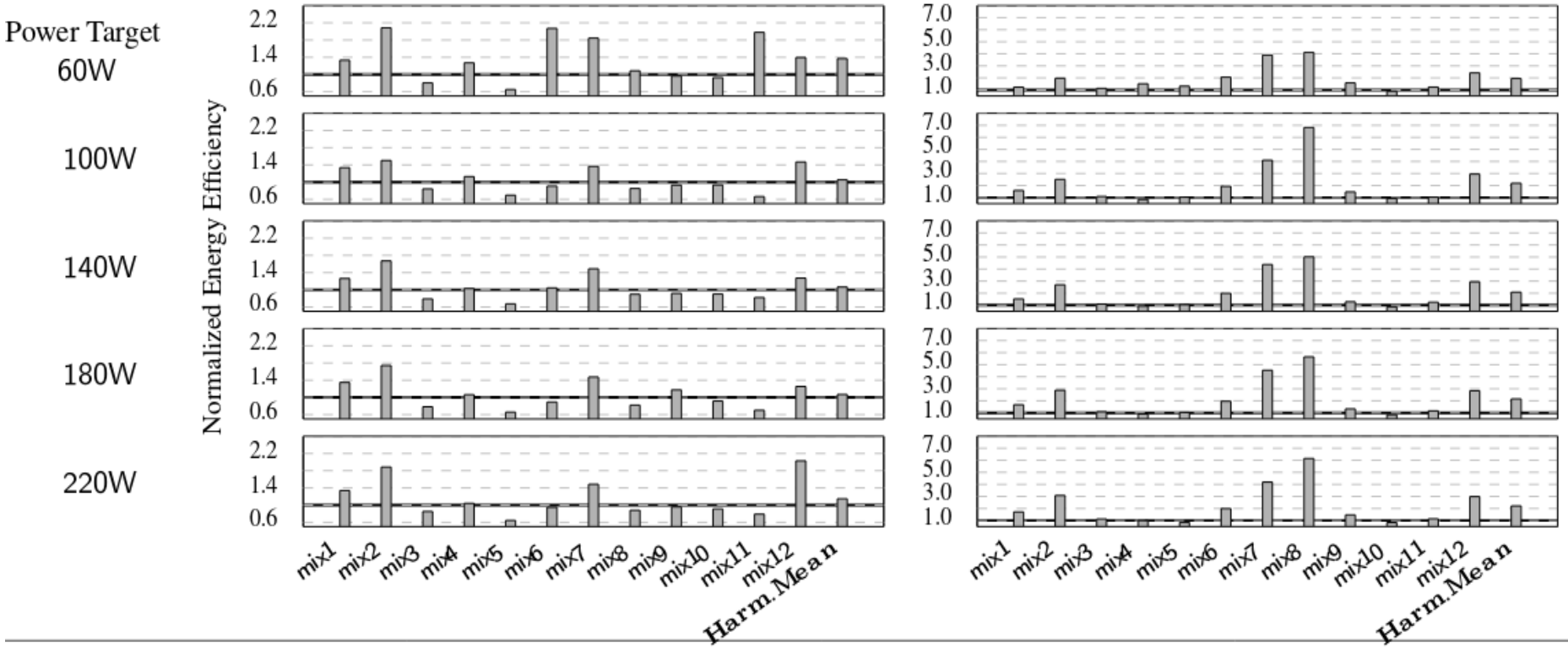


Figure 8. Ratio of PUPiL to RAPL energy efficiency in cooperative (left) and oblivious (right) multiapp scenarios.

Experiments : Sensitivity and Overhead analysis

- As seen all the approaches sensitivity is tested against various power cap.
- Also tests where carried with multi-application scenarios.
- In feedback system like Soft decision and PUPiL, two kinds of overheads are considered
  - Number of measurements to be taken before system converges
  - Impact on the converged system
- All the results reported include impact on the converged system
- Settling time shown covers the first type of overhead.

Conclusion

- Hardware Technique provide quick response time by quickly enforcing power limits
- While Software technique provides greater flexibility by tailoring resources to the current application workload.
- PUPiL which is hybrid approach is PUPiL's response time is comparable to Hardware Technique and flexibility and performance is comparable to software technique.
- Thus delivering performance under power cap cannot be left to just hardware or software approach but needs co-operation between both.

THANK YOU