# PowerStar: Improving Power Efficiency in Heterogenous Processors for Bursty Workloads with Approximate Computing

Sai Santosh Dayapule
*George Washington University*
saisantoshd@gwu.edu

Fan Yao
*University of Central Florida*
fan.yao@ucf.edu

Guru Venkataramani
*George Washington University*
guruv@gwu.edu

***Abstract*—Modern Data Centers have increasingly adopted heterogeneous processors in their server nodes to maximize power efficiency. However, there are still challenges in how to properly configure these processors such that throughput can be maximized under fluctuating workload while optimizing system power consumption. In this paper, we propose PowerStar, a framework that maximizes power efficiency and reduces the number of reconfigurations needed in heterogeneous processors during periods of fluctuations in job arrival patterns while handling latency-critical workloads. *PowerStar* is built based on the following two key observations: (i) reconfiguration of heterogeneous processors to add more cores and enable higher performance and/or re-allocation of computing cores can be costly due to the extra latency involved and the associated energy overheads; (ii) a considerable amount of energy savings can be achieved by keeping the system in most power-efficient configurations capable of absorbing short bursts in job arrivals without needing to reconfigure the system. *PowerStar* operates by carefully choosing the most power-efficient configurations (states) and judiciously maximizing the state residency through controlled use of approximate computing, when feasible. We implement PowerStar as a prototype on a 6-core ARM big.LITTLE heterogeneous platform and evaluate it with a variety of workloads. Our results show that, compared to a baseline of performance-driven power management policy, our power efficiency-aware PowerStar can reduce the average power by upto 11% under tight QoS ($95^{th}$ percentile latency under $3\times$ job execution latency), and can save even higher average power of upto 32% under relaxed QoS ($95^{th}$ percentile latency under $10\times$ job execution latency) constraints when compared to the baseline.**

***Index Terms*—Power aware computing, Heterogeneous processors, Approximate Computing, Workload management.**

## I. Introduction

Large-scale server farms and data centers account for more than 2% of the US domestic energy consumption [1]. As they are often over-provisioned to meet the peak demand, a considerable amount of power is spent on just keeping the server systems powered up. Improving power efficiency of data centers is very challenging as service providers need to meet the Quality of Service (QoS) requirements from the user, which is typically defined by tail latencies (e.g., $95^{th}$, $99^{th}$) percentile latencies) for the workloads.

We note that heterogeneous multiprocessing (HMP) offers even greater opportunity for optimizing power efficiency using cores with differing performance and power profiles. Owing to this reason, heterogeneous computing systems provides better adaptability to tradeoff performance against a given power budget. Prior work have developed mechanisms that allocate heterogeneous core resources for latency critical workloads to achieve energy savings [2], [3]. However, there are still two major challenges in heterogeneous computing environments: 1) Designing an effective resource allocation strategy in heterogeneous processor environments under temporally fluctuating workload arrival rates can be challenging. 2) Core reclamation and reassignment typically involves wakeup and shutdown of core and un-core hardware components, that can incur non-trivial performance and energy overheads if done frequently.

In this paper, we propose PowerStar, a framework for reconfiguring heterogeneous processors and load scheduling to improve power efficiency when there are temporal variations in job arrival patterns especially for latency-critical workloads. Unlike prior techniques that enable or disable cores based on performance needs of the current workload, PowerStar carefully selects the next configuration with higher performance that delivers higher *power efficiency* as well. This reduces the amount of reconfigurations needed when there are temporary spikes in job arrival patterns, while improving system power efficiency. Ocassionally, such high bursts over short period are commonly seen in today's latency-critical workloads [4]. To tackle this problem, PowerStar leverages approximate computing to moderately sacrifice acceptable levels of in accuracy for jobs (as permitted by corresponding application domains), so that the current configuration can still meet the latency constraints. Recent research has shown that many data center workloads, including web-search and web-service, are amenable to approximation [5]. *To the best of our knowledge, PowerStar is the first work that leverages both heterogeneity and dynamic approximation jointly to improve power efficiency for* latency-critical workloads.

In summary, the main contributions of our paper are as follows:

- We analyze various workloads under different arrival patterns, and our experimental results show that the power efficiency profiles can exhibit different trend patterns across HMP configurations. Our study shows the need for workload-specific analysis of power-efficiency profiles and reconfigurations to accommodate changes in workload arrival patterns.
- We design PowerStar, a framework that considers the cost of reconfiguration and the current workload arrival patterns along with QoS constraints, to judiciously choose the next HMP configuration especially during short bursts in workload arrival patterns.
- We explore mechanisms to enable adaptive job execution, that can further boost energy savings and reduce the

number of HMP reconfigurations through leveraging approximate computing techiniques. In particular, we study two different reconfiguration strategies to increase power efficiency of the system while maintaining QoS.

- We implement a prototype of our PowerStar framework on an ARM big.LITTLE processor-based physical testbed, and evaluate PowerStar's efficacy using PARSEC benchmarks [6]. Our results show that, compared to a baseline of performance-driven power management policy, our power efficiency-aware PowerStar can reduce average power by upto 11% under tight QoS ($95^{th}$ percentile latency under $3\times$ job execution latency), and can save even higher average power of upto 32% under relaxed QoS ($95^{th}$ percentile latency under $10\times$ job execution latency) constraints without adversely affecting application tail latencies.

## II. Background

Heterogeneous computing is one of the most promising ways to improve peak performance of the system while achieving energy proportionality as other techniques such as frequency scaling provide limited power reduction [7]. In Single-ISA asymmetric multiprocessing, the processing cores can be differentiated by various micro-architectural features such as cache and pipeline structure, clock frequencies, special instruction accelerators, out-of-order execution capability, etc. to provide significantly different performance and power characteristics for the same application binaries. A heterogeneous processor can be a valuable resource in data center environments, where the workload of different levels of performance sensitivity can be scheduled accordingly.

Linux's *race-to-idle* scheduling policy maintains an ordered list of CPU configurations (numbers of cores and related frequency settings) based on performance. It merely switches to next higher CPU configuration if the current one does meet the QoS demands. We note that such a policy may not work well for scheduling on heterogeneous processors, since a configuration with higher performance may not be the most power efficient. Also, other heterogeneous multiprocessing scheduler policy, such as the ARM's Global Task scheduler for big.LITTLE processor [8], schedules CPU-intensive tasks on the big cores and I/O intensive tasks on the small cores to improve task efficiency. However, this task may not be power-efficient when multiple jobs run in the system.

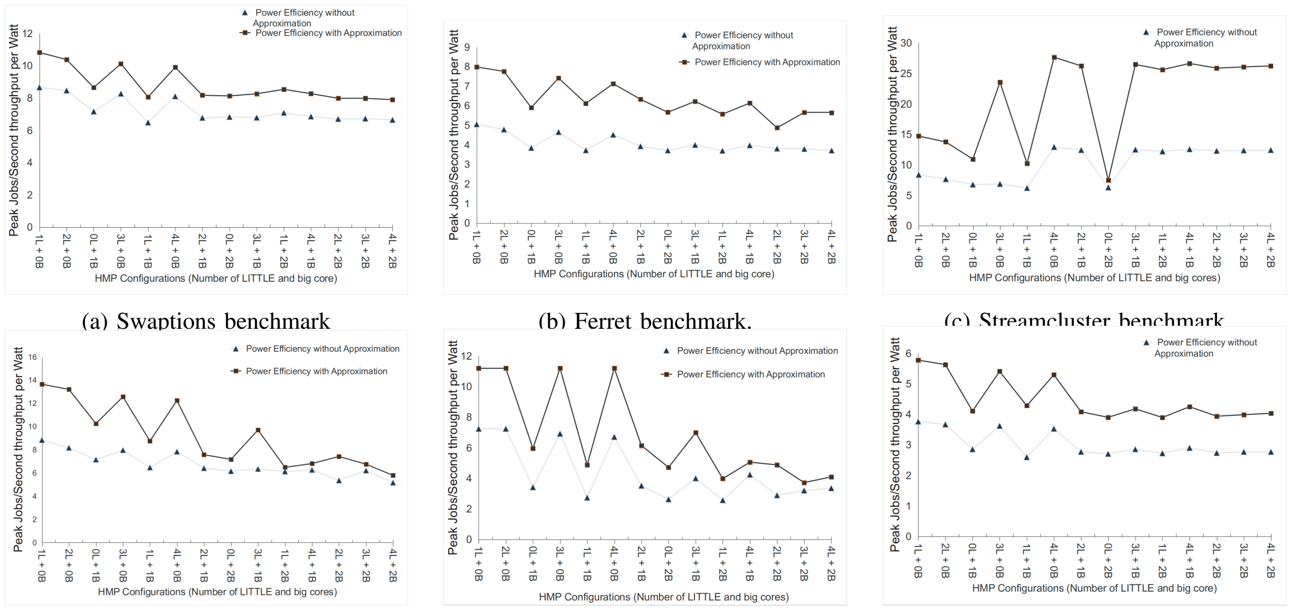
We quantify the performance requirements of the data center users in terms of *latency* of any given job to be serviced by the system as specified by the Quality-of-service constraints. The system targets to satisfy, for instance, the $95^{th}$ percentile target latency for all the jobs irrespective of the number of jobs arriving at any given point of time to the server. We can consider the latency target parameter to be a multiple of the job's execution time, that includes the communication time between client and server, queuing delays, CPU allocation time and time taken by the client to process the return request.

In order to meet performance demands from higher workload arrival, a processor reconfiguration is necessary to enable more number of active cores. Similarly, cores must be turned off when the workload arrival rate decreases. Such changes to active and low power status of cores is managed by the OS power governor, and during transition, they add both energy and latency costs. This is because, the low power states involve clock gating certain architectural features such as instruction pipelines and caches, and re-enabling them costs additional clock cycles. Prior work show that the transition time between active and sleep states can be high (around 20ms) [9]. Apart from these overheads, there is additional performance degradation due to warmup times of processor data and instruction caches depending on the application as established by [10]. This performance loss was measured to be around 200% additional time for next 10,000 instructions in SPEC benchmarks [10].

## III. Motivation

Under performance-driven power management policies, when the job workload exceeds the peak performance capacity of current configuration, the processor is reconfigured to enable for higher performance. This is accomplished through powering on an additional core, or setting a higher frequency or both. We note that such reconfiguration does not always correspond to improved power efficiency, i.e, Performance Per Watt (PPW) does not necessarily improve when simply moving to a higher configuration. Under fluctuating workload patterns, it may not be prudent to just pick the next performance configuration to adjust to increased workload.

In this section, we perform motivational study to understand the relationship between power efficiency and performance in heterogeneous multiprocessing architectures. We consider a job processing data center-like environment where each job takes a certain amount of resources to execute on a compute server and that it needs to satisfy the QoS constraints set forth by the user. In this experiment, we consider a 6-core ARM big.LITTLE processor, with 2 high performance (Big cores, denoted by, 'B') and 4 high power-efficiency (Little cores, denoted by 'L') processor cores. Each little core and big core consumes 0.2W and 0.6W peak power respectively on our ODROID XU4 board with the Exynos 5422 processor [11]. Each individual core could be dynamically shut down and will consume zero watt in power-off state. We use six benchmarks from the PARSEC benchmark suite [6] to build an online web-serving style applications on top of Apache (Refer to Section V for more details). Figure 1 shows the power efficiency (maximum jobs execution throughput per watt) for the benchmarks under various HMP configurations (the X-axis is sorted based on the power consumption). The maximum job throughput is obtained by gradually increasing the arrival rate until the QoS constraint (i.e., $95^{th}$ percentile latency is less than $3\times$ job execution latency) is no longer satisfied. From our experimental results, we can see that *the power efficiency does not linearly scale with the increasing performance of each successive configuration*. In fact, we can see that the peaks and valleys in power efficiency at each of the configuration, indicating considerable fluctuations of power efficiency as capacity of the configuration increases. This is significantly



(a) Swaptions benchmark (b) Ferret benchmark. (c) Streamcluster benchmark

(d) Bodytrack benchmark (e) Canneal benchmark (f) Fluidanimate benchmark

Fig. 1: Peak Power efficiency for different HMP configurations of 6-core big.LITTLE procesor for non-approximate and approximate versions of benchmarks.

different from homogeneous platforms. Also, by comparing the power efficiency curves across multiple benchmarks, we can find that the choice of power efficient states may slightly vary as well. For instance, in swaptions benchmark, 1L+2B is more power efficient than 4L+1B, whereas the trend is reversed in Ferret benchmark for the same two configurations.

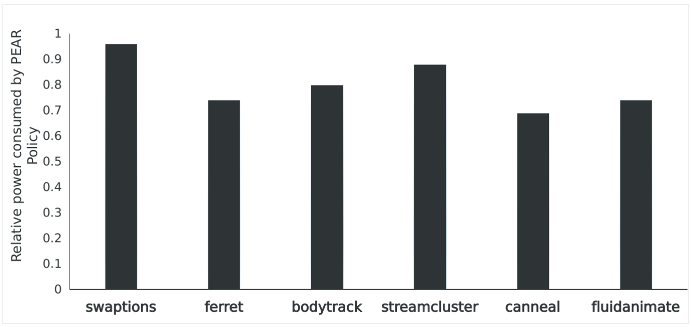


Fig. 2: Relative power consumed by PEAR policy over a baseline PAR policy.

We implement a baseline policy, Performance-Aware Reconfiguration (denoted as PAR) that simply transitions to the successive higher performance state when QoS constraints are not met. We conduct study to quantify the benefit of Power Efficiency-Aware Reconfiguration of HMP processors (denoted as PEAR) that transitions to next power-efficient HMP configuration when job arrival patterns change and QoS constraints are violated. We run the benchmarks for a total of 500s with diurnal (sinusoidal) job arrival pattern, and QoS ($95^{th}$ percentile latency) constraint of $3\times$ job execution latency. Figure 2 shows the relative power consumed by PEAR over the baseline PAR policy. We observe that the power under

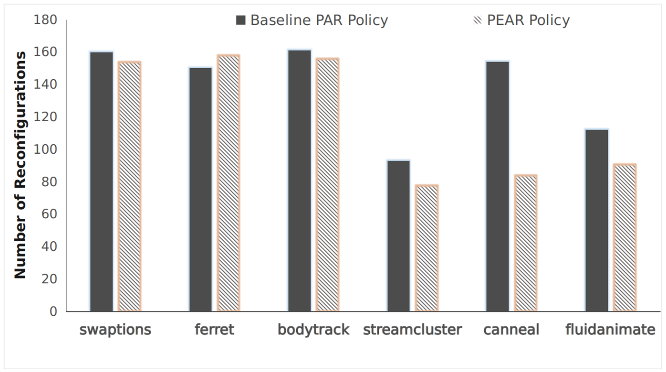


Fig. 3: Comparison of number of HMP reconfigurations required by PAR and PEAR policies.

PEAR policy could be upto 30% lower in certain benchmarks (e.g., canneal). Also, in Figure 3, we observe that PEAR reduces the number of HMP reconfigurations to satisfy QoS in 5 out of 6 applications (sometimes, as high a reduction by almost 50% e.g., canneal). In ferret, however, there is a slight increase in the number of HMP reconfigurations. While PEAR offers better energy savings than PAR in general, we note that our motivational study offers good insights into the promise of pursuing power efficiency-aware reconfiguration of HMP to accommodate short-lived fluctuations in workload arrivals.

Based on our motivational study, our intuition is that, through avoiding the processor configurations into power-inefficient states, we can significantly reduce the energy consumption of a server. Additionally, we enable approximation (as permitted by the application domain) in order to achieve faster execution time and higher power efficiency. Note that approximate computing for PARSEC benchmarks has been studied by prior work [12].

## IV. PowerStar Design Overview

As evidenced by our experimental results in Section III, HMP configurations that offer the necessary performance does not always correspond to increased power efficiency. Unfortunately, existing OS process schedulers typically coordinates tasks and manages heterogeneous cores merely based on the projected performance needs. For instance, Utilization-aware load balancers [8] categorize tasks into computation-intensive and I/O- intensive types, and schedule tasks onto high-performance and slow cores respectively.

In this work, we propose PowerStar, a power efficiency aware reconfiguration and load scheduling framework in heterogeneous processor environments, especially under short bursts or fluctuations in job arrival rates while satisfying QoS constraints. At a high level, PowerStar incorporates two critical components: 1) a workload-specific power efficiency profiling module for different HMP configurations; 2) a runtime power-efficiency aware reconfiguration module that dynamically allocates heterogeneous cores and leverages approximate computing for jobs to meet QoS constraints.

**Workload profiler for power efficiency of a HMP configuration.** In the offline analysis stage, we systematically profile the maximum arrival rate that a particular HMP configuration could serve under target QoS constraints. This could be computed by gradually increasing the job arrival rate in small increments until the tail latency (QoS) constraints are violated. After this step, we are able to obtain the maximum throughput (performance) per watt for a given configuration. To understand the effect of approximation that could enable higher power efficiency, we perform the same procedure for the approximate versions of the job workload under a different approximation levels. Essentially, after the completion of this offline analysis phase, our profiler generates a *power efficiency vector* that is a six-element tuple: $(W, Q, C, \lambda_{max}, E)$, where $W$ indicates the workload type, $Q$ is the QoS constraint (i.e., target tail latency), $C$ represents the HMP configuration. $\lambda_{max}$ and $E$ represent the maximum job arrival rate that can be serviced while not violating QoS constraints and, and E denotes the highest power efficiency under this setting.

**Runtime power efficiency-aware HMP reconfiguration.** Figure 4 illustrates the overall design of the power efficiency aware resource reconfiguration for PowerStar. At a high level, the reconfiguration comprises of three major modules: *the load monitor module*, *the reconfiguration controller* and *the job approximation manager*. The load monitor keeps track of the arrival rate and profiles the service times for the workload. It is worth noting that typically the average service times for latency-critical workload is steady over time, therefore the monitor only needs to capture the service time statistics for an initial warmup period of the system. However, the job arrivals may exhibit either short-term spikes or long-term diurnal fluctuations. Thus our monitor will continuously track the historical job arrival rates, and estimate job arrival traffic for the next decision window. PowerStar uses a moving average-based prediction, similar to TS-BatPro [13]. The reconfiguration controller receives the job arrival rates and service time statistics from the load monitor, and then makes a determination on the HMP configuration that will yield highest power efficiency while still satisfying the tail latency (QoS) objective. The reconfiguration can either perform reallocation of heterogeneous cores and/or request adjustment of approximation levels for the jobs to be served in the current period. Finally, the job approximation manager receives notifications about approximation target from the reconfiguration controller.

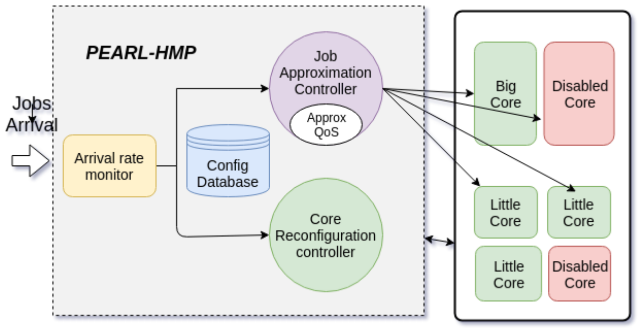


Fig. 4: High Level Design of PowerStar

### A. Conservative Switching

The conservative switching policy will prioritize leveraging approximation as much as possible without having to switch to a higher performance HMP reconfiguration. In other words, the conservative switching policy aims to remain *closer to the current HMP configuration* as possible, with the anticipation that load spikes are short-lived. Specifically, for the current configuration with power efficiency vector, denoted by $V_c$, if the predicted job arrival $\lambda_p$ exceeds $V_c.\lambda_{max}$, this policy first attempts to find an approximation level that can still guarantee the QoS (i.e., soft reconfiguration). If soft reconfiguration is not suitable (i.e., approximate computing still violates latency constraints), then HMP reconfiguration is performed (i.e., hard reconfiguration). The conservative switching mechanism is built based on the observation that the load fluctuation for latency-critical workloads often undergoes short-lived burstiness [4]. Therefore, a slight change in HMP configuration (e.g., an additional small core) with job approximation retained for a short period of time is expected to absorb such short-term spikes. This avoids considerable hardware reconfiguration (e.g, allocation of a big core) that may severely deteriorate the power efficiency of the system. We note that the approximation level needs to be controlled judiciously so that the accuracy of the latency-critical workload would not be excessively influenced. Accordingly, we monitor the number of jobs which were executed approximately and employ a approximation threshold to bound the worst-case accuracy loss for the jobs. Algorithm 1 describes the conservative switching policy.

### B. Aggressive Switching

Different from Conservative Switching, Aggressive Switching prioritizes HMP configuration that can serve the incoming jobs that avoids sacrificing accuracy. That is, whenever the current HMP configuration is unable to bound the tail latency due to increased load with the highest allowable approximation level (per user's QoS constraints), it will elect the next most power efficient HMP allocation for the non-approximate job execution to meet the QoS. This algorithm is more applicable when there are *high variations in job arrival rate patterns*. By choosing the next most efficient state without sacrificing accurancy loss, we still leave room in the new transition state to absorb any future variations in arrival rate through approximation. The aggressive switching policy is shown in Algorithm 2.

**Algorithm 1:** Conservative Switching algorithm

```
Input: ;
List of HMP configuration in order of performance:$C_1, C_2, C_3 ... C_n$;
New estimated job arrival rate: $R_{new}$;
Current job arrival rate: $R$;
Current HMP configuration: $C_i$;
Number of Approximated Jobs in current time window:$NumAprrox$;
Current level of Approximation:$A_{in}$;
Power for the corresponding HMP configurations $P_1, P_2, P_3 ... P_n$;
Output: New HMP configuration: $C_o$;
New level of Approximation: $A_{out}$
let $A_{none}$ and $A_{high}$ be the lowest and highest approx. levels;
if $R_{new}$ >MaxRate($C_i$, $A_{in}$) then
    if $A_{in}$ = $A_{low}$ and $R_{new}$ <MaxRate($C_i$, $A_{high}$) and
      $numApprox + 1 < QoS_{approx}$ then
        $A_{out} \leftarrow A_{high}$;
    else if $numApprox + 1 < QoS_{approx}$ then
        Find HMP configuration $C_x$ with lowest power $P_x$ such that
          MaxRate($C_x$, $A_{high}$) <$R_{new}$;
        $A_{out} \leftarrow A_{high}$;
    else
        if $numApprox + 1 < QoS_{approx}$ then
            $C_o \leftarrow C_x$;
            $A_{out} \leftarrow A_{high}$;
        else
            Find HMP configuration $C_x$ with lowest power $P_x$ such that
              MaxRate($C_o$, $A_{none}$) <$R_{new}$;
            $A_{out} \leftarrow A_{none}$;
else if $R_{new}$ <MaxRate($C_i$, $A_{in}$) then
    $CApprox_{min} \leftarrow$
      $FindstateC_x withlowestP_x suchthatMaxRate(C_x, A_{high}) < R_{new}$;
    $CNoApprox_{min} \leftarrow$
      $FindstateC_x withlowestP_x suchthatMaxRate(C_x, A_{none}) < R_{new}$;
    if $CApprox_{min} \neq CNApprox_{min}$ then
        if $numApprox + 1 < QoS_{approx}$ then
            $A_{out} \leftarrow A_{high}$;
        else
            $C_o \leftarrow CNApprox_{min}$;
            $A_{out} \leftarrow A_{none}$;
    else
        $C_o \leftarrow CNApprox_{min}$;
        $A_{out} \leftarrow A_{none}$;
else if $A_{in} = A_{high} andNumApprox + 1 > QoS_{approx}$ then
    $A_{out} \leftarrow A_{none}$
return $A_{out}$;
```

*C. Implementation*

The resource allocation of the heterogeneous processor is implemented by dynamically mapping active threads to the available cores in the current configuration, and by putting the rest of the unused cores into low-power mode [14]. We implement the reconfiguration algorithm as a separate module within httperf load generator [15]. The reconfiguration module performs load prediction within the next time window, and determines the *right HMP configuration* as well as *job approximation levels* using algorithms described earlier in Section IV. It then issues commands to the Apache server that performs the soft and hard reconfiguration of HMPs as needed. Our typical monitor window in-between decisions for reconfiguration is *1 second*, although a rapidly changing arrival pattern may trigger early reconfiguration if necessary. HMP reconfiguration is done by sending special HTTP requests to the application server to perform remapping of the Apache working threads as well as modifying the approximation parameters of specific applications in the CGI scripts.

**Algorithm 2:** Aggressive Switching algorithm

```
Input: ;
List of HMP configuration in order of performance:$C_1, C_2, C_3 ... C_n$;
New estimated job arrival rate:$R_{new}$;
Current job arrival rate:$R$;
Current HMP configuration:$C_i$;
Number of Approximated Jobs in current time window:$NumAprrox$;
Current level of Approximation:$A_{in}$;
Power for the corresponding HMP configurations$P_1, P_2, P_3 ... P_n$;
Output: New HMP configuration:$C_o$;
New level of Approximation:$A_{out}$
let $A_{none}$ and $A_{high}$ be the lowest and highest approximation levels;
if $R_{new}$ >MaxRate($C_i$, $A_{in}$) then
    if $A_{in}$ = $A_{low}$ and $R_{new}$ <MaxRate($C_i$, $A_{high}$) and
      $numApprox + 1 < QoS_{approx}$ then
        $A_{out} \leftarrow A_{high}$;
    else
        Find HMP configuration $C_o$ with lowest power $P_x$ such that
          MaxRate($C_o$, $A_{none}$) <$R_{new}$;
        $A_{out} \leftarrow A_{none}$;
else if $R_{new}$ <MaxRate($C_i$, $A_{in}$) then
    $CApprox_{min} \leftarrow FindthestateC_x withlowestpowerP_x$
      $suchthatMaxRate(C_x, A_{high}) < R_{new}$;
    $CNoApprox_{min} \leftarrow FindthestateC_x withlowestpowerP_x$
      $suchthatMaxRate(C_x, A_{none}) < R_{new}$;
    if $CApprox_{min} \neq CNApprox_{min}$ then
        $C_o \leftarrow CNApprox_{min}$;
        $A_{out} \leftarrow A_{none}$;
else if $A_{in} = A_{high} andNumApprox + 1 > QoS_{approx}$ then
    $A_{out} \leftarrow A_{none}$
return $A_{out}$;
```

As part of the QoS constraints, we also consider an *approximation threshold* to avoid noticeable loss of accuracy due to prolonged execution of approximated jobs. Currently, our policy would not allow more than a sequence of $k$ jobs to be executed continuously in approximation mode (where $k$ is set by the user). We note that $k$ is a configurable parameter that could be set by system administrators based on user's needs. By default, we set $k$ to be 5 in our experiments.

## V. Experimental Setup

**Client and server configuration.** We build a prototype of PowerStar using Apache web server. The web server is deployed on an ODROID XU4 board equipped with the Samsung Exynos 5422 ARM big.LITTLE processor [16]. The key hardware configuration and the power model of the ARM board are illustrated in Table I. In our evaluation, we let PowerStar to use 4 LITTLE and 2 big cores while the rest two big cores are allocated exclusively to run other system background processes. This ensures that other system processes will not interfere with latency-critical workloads. Each request from the client side is served by the Apache server internally via executing CGI scripts that run a particular native job.We setup a client-side load generator on a separate machine running *httperf* [15]. We modified httperf so that it is able to generate server traffic based on real-world job arrival traces. We utilize a real system job arrival trace from NLANR [17] to model our workload arrival patterns to the system. Additionally, we also use a synthetically generated sinusoidal (diurnal) job arrival pattern that exhibits higher variation in the job arrivals to stress-test our system.

TABLE I: Exynos 5422 big.LITTLE processor specs

| Specification | LITTLE cluster | big cluster |
|---|---|---|
| Core Type | Cortex A7 | Cortex A15 |
| Number of Cores | 4 | 4 |
| Max. Frequency | 1.4 Ghz | 2 Ghz |
| Pipeline | In-order | Out-of-order |
| Peak Power | 0.2W | 0.6W |
| Memory | 2 GB LPDDR3 RAM | |

### A. Benchmarks

We consider six different applications from PARSEC-2.1 [6]. Each application support approximate computing via tuning the application parameters [18]. Currently, we only model one level of approximation in PowerStar, and note that our analysis can be extended to multiple approximation levels as well. We describe our applications in Table II:

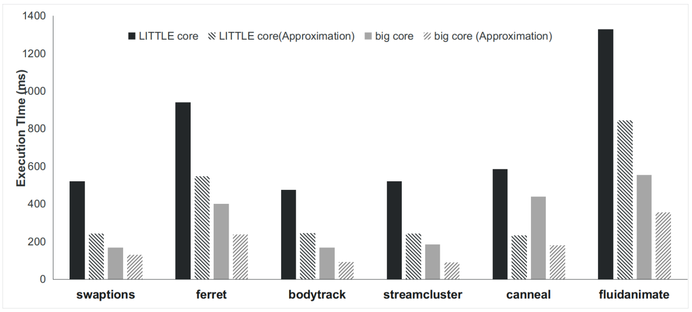


Fig. 5: Benchmark execution times (non-approximated and approximated) when run on big and little cores.

## VI. Evaluation

In our experimental evaluation, we study the effectiveness of PowerStar using applications from various domains, arrival traffic patterns and QoS constraints.

### A. Power Efficiency and Performance

First, we study the effect of approximation on the execution time of benchmarks as well its power efficiency profiles. We use the approximation parameters outlined in Section V. Figure 5 shows the execution times of the various benchmarks in LITTLE and big cores under both non-approximate and approximate settings. As expected, big cores offer higer performance in general over little cores even over approximation. The only exception is canneal, where execution time is reduced significantly when run under approximate mode. Figure 1 presents corresponding power efficiency profiles for various HMP configurations in different benchmarks. As expected the power efficiency profiles of approximate versions are much higher, and in particular, benchmarks like streamcluster, canneal, bodytrack and ferret show a sharply increased power efficiency for certain HMP configurations under approximate mode.

### B. Conservative vs. Aggressive Switching

We study the average power consumption for conservative and aggressive switching policies in our PowerStar framework for both tight QoS ($95^{th}$ percentile latency under $3\times$ job execution latency) and under relaxed QoS ($95^{th}$ percentile latency under $10\times$ job execution latency) . For these experiments, we model load arrival patterns from real traces such as NLANR, as well as synthetic diurnal job arrival pattern. Our experimental results are shown in Figure 6. We find that Conservative switching offers higher power savings in general, and can result in upto 11% savings compared to baseline PAR policy (See Section III). With the diurnal arrival pattern, the conservative switching policy provides much higher power savings of up to 32% in certain benchmarks such as canneal.

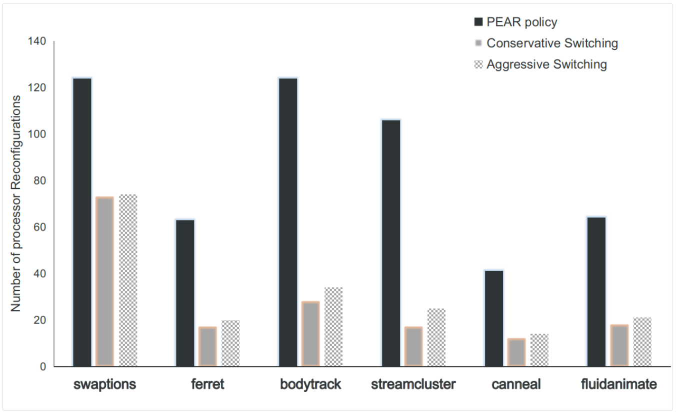


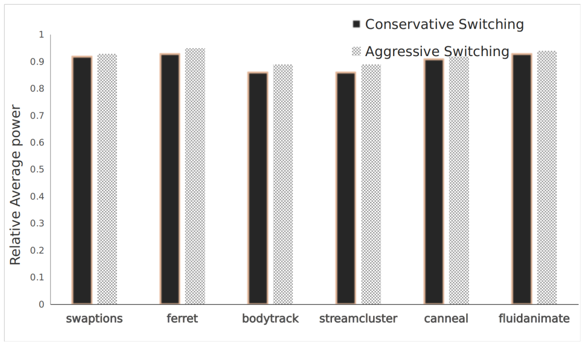
Fig. 7: Comparison of number of reconfigurations required for tight (3x) QoS threshold NLANR job arrival.

### C. HMP reconfigurations

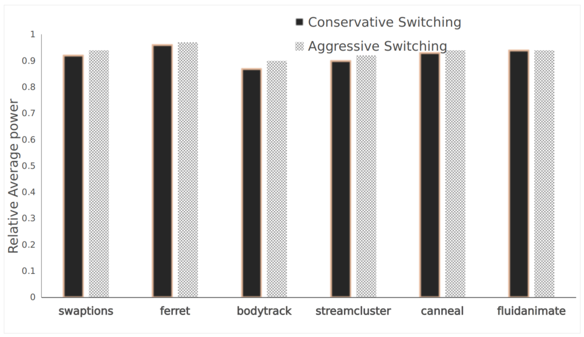
One important benefit of using approximation is the reduction in the number of reconfiguration required. Using either Conservative or Aggressive policies, PowerStar is able to reduce the number of reconfigurations by 41%-84% as compared to the baseline PAR policy for NLANR job arrival pattern as shown in figure 7. Moreover, by switching to the non-approximate jobs when current load exceed the capability of the hardware configuration, the *conservative* policy offers a slightly lower number of reconfiguration when compared to the *aggressive* policy. Obviously, this preference of non-approximation state offers more room to absorb the fluctuations in job arrival rate, and thus fewer transitions. Additionally, when the workload is more varying as in the diurnal job arrival pattern, we see that aggressive policy is preferable.

TABLE II: Benchmarks and approximation parameter details.

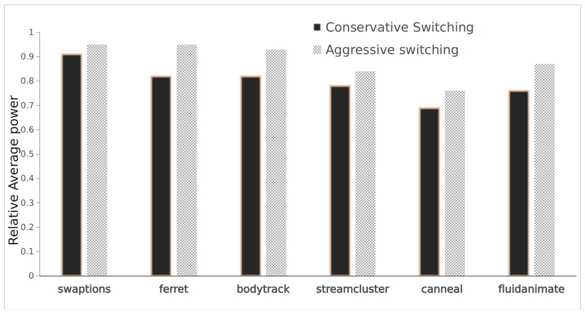
| Benchmark | Application domain | Approximation Parameter |
|---|---|---|
| Swaptions | Financial Analysis | Number of Montecarlo simulations reduced from 200 to 100. |
| Bodytrack | Computer Vision | Limiting the number of input particles processed 20 to 10. |
| Ferret | Similarity search | Number of top results queried from 5 to 2. |
| Streamcluster | Data Mining and Clustering | Number of clusters required from 1000 to 750. |
| Canneal | Simulated Annealing for Optimum routing cost | Number of Swapping steps from 1000 to 500. |
| Fluidanimate | Physics Computation | Number of frames to simulate from 10 to 6 in the 15K particle testcase. |



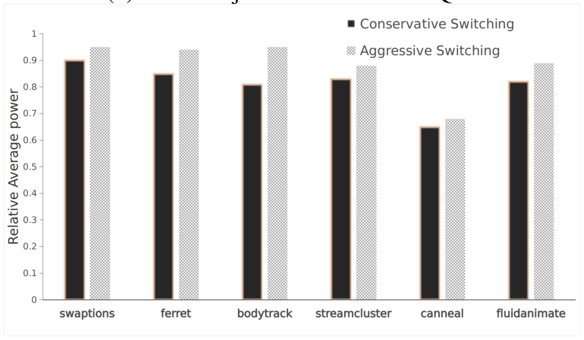
(a) NLANR job arrival: Tight QoS



(b) NLANR job arrival: Relaxed QoS



(c) Diurnal job arrival: Tight QoS



(d) Diurnal job arrival: Relaxed QoS

Fig. 6: Average Relative Power under different job arrival patterns and QoS constraints (Baseline: PAR policy).

## VII. Related Work

Energy management for latency critical workload has been well studied in the past decade. Typically, the proposed techniques could be categorized into two classes: (i) idle power management that improves energy savings by putting system hardware components to low-power modes ( [13], [14], [19]–[21]), (ii) active power managements leveraging Dynamic Voltage and Frequency Scaling (DVFS) to save processor or server dynamic power ( [4], [22]), (iii) Application power debugging [23]–[25]. These techniques exploit the tail latency slack to dynamically trade-off performance for higher energy efficiency while still meeting the service level agreement. Most of these mechanisms consider homogeneous multi-core processors that exhibit consistent power efficiency across cores.

During the past few years, several studies have shown that heterogeneous processors offer even greater opportunity for enhancing computing efficiency through cores with differentiating performance characteristics. Some prior works have proposed energy efficient scheduling frameworks for heterogeneous processor. Specifically, Octopus-Man [2] dynamically schedules tasks on either high performance cores or power-efficient cores to enhance energy efficiency with QoS awareness. Hipster [3] utilizes reinforcement learning to adaptively map workload at certain utilization level to a heterogeneous core allocation as well as DVFS configuration that meet QoS and achieve higher energy savings. Note that these techniques do not explicitly model or leverage the power efficiency across various HMP configurations. Differently, PowerStar directly optimizes energy efficiency through intentionally choosing the most *power efficient* configurations, which has been shown to be extremely effective in achieving considerable energy savings. Haque et al. [26] use techniques that dynamically change migration thresholds for transitioning application processes from slow to fast processing cores based on length of execution. This approach may incur non-trivial performance overhead due to the fine-grained per-process monitoring and scheduling mechanism.

Finally, there are some emerging studies on using approximation to trade off quality of result for better performance and/or energy efficiency [5], [12], [27]. CLAP [28] adjusts the data aggregation granularity for online search workload to achieve reduced tail latency with minimal accuracy losses. PowerDial [18] dynamically adapts the application behavior through approximate computing so that applications can meet task deadlines. MEANTIME [12] applies control theory to perform resource allocation based on the current jobs, and adjusts job approximation levels to avoid performance violations. Pliant [29] improves server utilization for mixed workloads by controlling approximation of non-latency critical jobs to satisfy the latency requirement of co-located latency critical jobs. This approach does not consider server heterogeneity and cost of processor core reconfiguration especially in highly variable workload patterns.

Additionally, there have been studies utilizing both heterogeneity and job approximation techniques. Tan et al. [30] propose a heterogeneity aware, per task scheduler to improve performance under a strict DP constraint. Recently, Kanduri et al. [31] show how a coordinated management of approximation and processor performance control is necessary as controlling one without aware of the other will output incorrect results. Compared to these works, the key difference of our proposed framework is that PowerStar makes use of approximation as the control knob to optimize duration of the system in the most power efficient state even under bursty workloads. PowerStar is able to achieve higher energy saving due to reduced number of hard reconfiguration as well as prolonged stay in power efficient settings.

## Conclusion

In this paper, we proposed PowerStar, a power efficiency-aware reconfiguration and local scheduling framework that aims to optimize power efficiency for latency critical workload on heterogeneous multi-processors. PowerStar carefully chooses the most power-efficient configurations and maximizes their state residency. Through a coordinated reconfiguration mechanism, PowerStar is able to significantly reduce the number of hardware reconfigurations even when the workload exhibits short-lived bursty job arrivals. We have built a prototype of PowerStar on a real-world HMP platform, and extensively evaluated the efficacy of our proposed framework using a variety of workload and traffic patterns. Our results show that, compared to a baseline of performance-driven power management policy, our power efficiency-aware PowerStar can reduce as much as 28% power reduction under tight QoS ($95^{th}$ percentile latency under $3\times$ job execution latency), while reducing as much as 50% processor reconfigurations.

## Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant Number CNS-178133.

## References

[1] J. Whitney and P. Delforge, "Data Center Efficiency Assessment," National Resources Defense Council (NRDC) Issue Paper IP:14-08-A, Aug. 2014.

[2] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mossé, J. Mars, and L. Tang, "Octopus-Man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers," in *HPCA*, 2015.

[3] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell, "Hipster: Hybrid Task Manager for Latency-Critical Cloud Workloads," in *HPCA*, 2017.

[4] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *MICRO*, Dec. 2015.

[5] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "MobiQoR: Pushing the Envelope of Mobile Edge Computing Via Quality-of-Result Optimization," in *ICDCS*, 2017.

[6] C. Bienia, "Benchmarking Modern Multiprocessors," PhD Thesis, Princeton University, 2011.

[7] A. Lukefahr, S. Padmanabha, R. Das, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Heterogeneous microarchitectures trump voltage scaling for low-power cores," in *PACT*, 2014.

[8] M. Kim, K. Kim, J. R. Geraci, and S. Hong, "Utilization-aware load balancing for the energy efficient operation of the big.LITTLE processor," in *DATE*, 2014.

[9] S. Yoo, Y. Shim, S. Lee, S.-A. Lee, and J. Kim, "A Case for Bad Big.LITTLE Switching: How to Scale Power-performance in SI-HMP," in *HotPower '15*. ACM, 2015.

[10] A. Sarkar, F. Mueller, H. Ramaprasad, and S. Mohan, "Push-assisted Migration of Real-time Tasks in Multi-core Processors," in *ACM SIGPLAN/SIGBED*, 2009.

[11] A. Butko, F. Bruguier, A. Gamatié, G. Sassatelli, D. Novo, L. Torres, and M. Robert, "Full-System Simulation of big.LITTLE Multicore Architecture for Performance and Energy Exploration," in *MCSOC*, 2016.

[12] A. Farrell and H. Hoffmann, "MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing," in *USENIX ATC*, 2016.

[13] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "TS-BatPro: Improving Energy Efficiency in Data Centers by Leveraging Temporal-spatial Batching," *IEEE TGCN*, 2018.

[14] B. Lu, S. S. Dayapule, F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "PopCorns: Power Optimization Using a Cooperative Network-Server Approach for Data Centers," in *(ICCCN)*, 2018-07.

[15] D. Mosberger and T. Jin, "Httperf—a Tool for Measuring Web Server Performance," *SIGMETRICS Perform. Eval. Rev.*, no. 3, Dec. 1998.

[16] "ODROID-XU4," https://www.hardkernel.com/shop/odroid-xu4/.

[17] "National Lab of Applied Network Research." http://www.nlanr.net/.

[18] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic Knobs for Responsive Power-aware Computing," in *ASPLOS*. ACM, 2011.

[19] Y. Liu, S. C. Draper, and N. S. Kim, "SleepScale: Runtime Joint Speed Scaling and Sleep States Management for Power Efficient Data Centers," in *ISCA*, 2014.

[20] F. Yao, J. Wu, S. Subramaniam, and G. Venkataramani, "WASP: Workload Adaptive Energy-Latency Optimization in Server Farms Using Server Low-Power States," in *2017 IEEE 10th Int'l Conf. on Cloud Computing (CLOUD)*, Jun. 2017.

[21] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A dual delay timer strategy for optimizing server farm energy," in *(CloudCom)*, 2015.

[22] M. Alian, A. H. M. O. Abulila, L. Jindal, D. Kim, and N. S. Kim, "NCAP: Network-Driven, Packet Context-Aware Power Management for Client-Server Architecture," in *HPCA*, Feb. 2017.

[23] J. Chen and G. Venkataramani, "enDebug: A hardware–software framework for automated energy debugging," *Journal of Parallel and Distributed Computing*, Oct. 2016.

[24] J. Chen, F. Yao, and G. Venkataramani, "Watts-inside: A hardware-software cooperative approach for Multicore Power Debugging," in *(ICCD)*, 2013.

[25] J. Chen, G. Venkataramani, and G. Parmer, "The Need for Power Debugging in the Multi-Core Environment," *IEEE Computer Architecture Letters*, Jul. 2012.

[26] M. E. Haque, Y. He, S. Elnikety, T. D. Nguyen, R. Bianchini, and K. S. McKinley, "Exploiting Heterogeneity for Tail Latency and Energy Efficiency," in *MICRO '17*. ACM, 2017.

[27] M. Carbin, D. Kim, S. Misailovic, and M. C. Rinard, "Proving Acceptability Properties of Relaxed Nondeterministic Approximate Programs," in *PLDI '12*. ACM, 2012.

[28] R. Han, S. Huang, Z. Wang, and J. Zhan, "CLAP: Component-Level Approximate Processing for Low Tail Latency and High Result Accuracy in Cloud Online Services," *IEEE TPDS*, 2017.

[29] N. Kulkarni, F. Qi, and C. Delimitrou, "Pliant: Leveraging approximation to improve datacenter resource efficiency," in *HPCA*, Feb 2019.

[30] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju, "Approximation-aware scheduling on heterogeneous multi-core architectures," in *Asia and South Pacific DAC*, 2015.

[31] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt, "Approximation-aware Coordinated Power/Performance Management for Heterogeneous Multi-cores," in *DAC*. ACM, 2018.