CHIPS-AHOy: A Predictable Holistic Cyber-Physical Hypervisor for MPSoCs

Tiago Mück
University of California, Irvine
Irvine, CA, USA
tmuck@uci.edu

Antonio A. Fröhlich
Federal University of Santa Catarina
Florianópolis, SC, Brazil
guto@lisha.ufsc.br

Giovani Gracioli
Federal University of Santa Catarina
Florianópolis, SC, Brazil
giovani@lisha.ufsc.br

Amir M. Rahmani
University of California, Irvine
Irvine, CA, USA
Technische Universität Wien
Wien, Austria
amirr1@uci.edu

João Gabriel Reis
Federal University of Santa Catarina
Florianópolis, SC, Brazil
jgreis@lisha.ufsc.br

Nikil Dutt
University of California, Irvine
Irvine, CA, USA
dutt@ics.uci.edu

ABSTRACT

Virtual machines (VMs) are being deployed on embedded systems to integrate multiple applications with different run-time requirements on the same physical platform. In scenarios such as autonomous vehicles, these virtualized platforms must handle varying application requirements – from strict temporal predictability to high performance – while simultaneously satisfying disparate constraints such as energy efficiency, thermal bounds, and system lifetime. To address these challenges we present **CHIPS-AHOy**, a predi**C**table **H**ol**I**stic cyber-**P**hy**S**ic**A**l **H**ypervis**O**r that integrates VM isolation mechanisms with novel resource allocation approaches within a holistic observe-decide-adapt loop to achieve run-time predictability and simultaneously manage energy, thermal and wearout constraints. We present experimental results on several realistic MPSoC platforms that demonstrate the ability of CHIPS-AHOy to achieve predictable operation while conserving energy, managing temperature and extending system lifetime.

KEYWORDS

Heterogeneous Multi-core Processor, Power Management, Operating Systems, Virtualization, Real Time, Predictability

1 INTRODUCTION

The design of efficient computing platforms is essential for realizing future applications, such as complex cyber-physical systems and handheld mobile systems, ensuring high-performance and acceptable quality of service at low power consumption and cost. However, aggressive technology scaling has resulted in issues such as process variations, failure of Dennard's law and emergence of dark silicon [1]. In this context, virtualization is gaining more attention, fueled by the growth in single-chip core count and the need for embedded systems consolidation. Virtualization enables the integration of applications with different run-time requirements in the same physical platform, promotes cost reduction both in terms of development and maintenance, and improves resource utilization efficiency [2–4]. For many emerging CPS domains such as autonomous vehicles, such virtualized platforms have to deal with diverse, often conflicting, requirements (Fig. 1). Some applications, such as steering, fuel injection, and brake handling, are life/mission *critical* and pose hard real-time requirements to the system. Conversely, multimedia infotainment systems demand high-performance and are programmed to tolerate large variations in the quality of the services provided by *best-effort* operating systems such as Linux. In addition, a third rapidly growing class of *sensitive* applications call for both predictability and performance at the same time. Applications such as vision-based driver assistance and navigation have become too complex to fit within the traditional development cycle of critical embedded systems, yet they cannot be handled as best-effort ones. Sensitive applications needs to be served considering all the new challenges of the dark silicon era, including heterogeneity, energy proportionality, thermal issues, and wear-out [5].

Figure 1: Envisioned MPSoC consolidation scenario.

SAMOS XVIII, July 15–19, 2018, Pythagorion, Samos Island, Greece

ACM ISBN 978-1-4503-6494-2/18/07…$15.00
https://doi.org/10.1145/3229631.3229642

The run-time system in MPSoCs supporting such a hybrid workload needs to efficiently manage resources to satisfy: i) system-driven requirements such as reliability and power, ii) application-specific objectives such as predictability and high-performance, and iii) platform-wide co-management of different application categories and resources. Unfortunately, existing hypervisors are unable to assist ordinary operating systems to better manage diverse system resources without compromising response time and throughput.

To address these challenges, we propose CHIPS-AHOy: a holistic hypervisor that consolidates traditional predictable run-time systems with self-aware *computing-communication-control* (C3) platforms to maximize the overall system resource utilization while delivering an acceptable level of predictability to sensitive applications, as well as service guarantees for critical applications. We exploit cyber-physical system concepts applied to MPSoCs with on-chip and cross-layer sensing and actuation capabilities [6, 7] to enable adaptations within the *observe-decide-act* (ODA) [8] paradigm. We devise mechanisms to manage the major sources of unpredictability in modern architectures for applications exhibiting a mix of critical and sensitive tasks while enabling best-effort tasks – scenarios typical in emerging automotive and IoT worlds. The main contributions of this paper are:

- we comprehensively analyze major sources of unpredictability for runtime systems in emerging heterogenous MPSoCs, and highlight scalability challenges.
- we present CHIPS-AHOy: a holistic and predictable cyber-physical hypervisor capable of handling hybrid workloads comprising critical, sensitive and best effort tasks, while efficiently managing system resources.
- we demonstrate the efficacy of CHIPS-AHOy via a proof-of-concept prototype on three real multi-core platforms (Odroid, Zynq, and Intel i7) compared against a baseline Linux kernel.

2 RELATED WORK AND MOTIVATION

Modern multicore systems make intense use of latency hiding mechanisms to exploit locality while multiplexing an ever growing set of heterogeneous hardware resources to applications [7, 9]. In order to achieve high-performance, opportunistic and speculative algorithms have been developed, typically focusing on best performance or, more recently, on best performance per watt [5]. Here we discuss the most relevant sources of unpredictability in MPSoCs, as well as recent efforts to overcome them.

Shared Memory Hierarchy: the path between main memory, processors, and I/O devices is paved with latency hiding mechanisms, including caches, scratchpads, buffers, and FIFOs, that enable a variety of latency and bandwidth demands to coexist in a hierarchy at the price of poor predictability. Techniques such as private memory, coherent cache (cc-NUMA), and distributed shared memory boost performance, but suffer from limitations in scalability, energy efficiency, and timing. Consequently these techniques themselves become the primary sources of unpredictability in modern MPSoCs [10].

False sharing – the unintentional sharing of mid-hierarchy resources due to their placement and associativity – is the most notorious source of unpredictability with respect to memory. Even

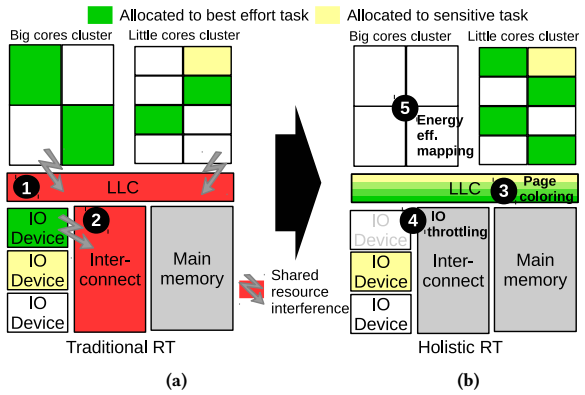


Figure 2: (a) traditional RTS, (b) our holistic approach.

if each VM gets its own share of private and global memories, the latency hiding mechanisms in the hierarchy are mostly unaware that some VMs are running critical or sensitive software which can't cope with time variations. Allocation and mapping at those intermediate levels are done in an application-agnostic manner, mostly focusing on system performance, using models and predictors (e.g., LRU approximation). VMs running data-intensive applications (e.g., multimedia) can easily exhaust FIFOs and buffers and evict a large number of cache lines. A VM running critical or sensitive applications needs to concur for those resources, suffering large variations in access time as illustrated in Fig. 2(a). While best-effort applications can easily trade predictability for performance, critical and sensitive ones perceive such variations as jitter, which eventually can violate their requirements. One approach to improve the predictability is the partitioning and isolation of shared resources. Cache partitioning has been widely studied [10]. Other approaches focus on the design of predictable hardware components, such as predictable memory controllers [11], or memory bandwidth reservation schemes in software [12]. Thus, it is important to have a holistic hypervisor to identify processor architecture bottlenecks that affect predictability.

Shared I/O Subsystem: Modern I/O subsystems are also partitioned and utilize latency hiding mechanisms but deliver lower throughput compared to those designed to feed data-hungry CPUs [13]. Many systems are designed assuming that just a few I/O devices will be active at any given time, which is often a wrong assumption for large MPSoCs. Contention in the I/O subsystem entails large variations in response time, causing delays and – more importantly for critical and sensitive applications – deadline misses. Indeed, the periods and deadlines defined for such applications at design-time are mostly based on the assumption that assessing the state of a physical component (e.g., reading a sensor or capturing an image) is a constant-time operation. This assumption has to be preserved when those applications are hosted on a conventional VM sharing the same set of I/O resources with others (Fig. 2(a)). To handle contention, the whole I/O subsystem can be managed via a traditional real-time scheduler, with time slots being dynamically allocated to tasks [14]. This approach, however, requires I/O delays to be estimated at design-time, which can be only done by

eliminating contentions through the reduction of parallelism or by over-provisioning, both of which fail to scale along with MPSoCs. I/O interference can be accounted within the execution time of tasks by profiling applications at design-time [15], which is infeasible for highly dynamic environments. Another alternative is QoS management performed by the interconnect itself. I/O flows are tagged and dynamically accounted, ensuring each one gets the specified share of the bandwidth [16]. However, effective resource reservation is infeasible for a large number of concurring VMs, particularly for network interfaces and storage devices being usually shared among VMs.

Hardware Variability: Variability in deep sub-micron technology is rapidly growing due to aggressive scaling. ITRS [17] predicts that over the next decade performance variability will increase from 48% to 66% and total power consumption variability will increase by up to 100%. Orthogonal to manufacturing induced process variability, emerging embedded and battery operated devices are moving towards MPSoC platforms with architecturally differentiated cores, thus introducing platform-level variability as well. Moreover, the dynamic nature of workloads and the way cores are allocated and stressed over the lifetime span of a chip also results in significant temperature gradients on the substrate leading to unbalanced cores' aging and high variance of *Mean Time To Failure* (MTTF) [18].

Previous works have proposed ways to deal with such variability by using adaptive guardbanding, adaptive reliability management, and novel task mapping techniques [5, 19–21], however none of them manage hardware unpredictability in a holistic manner to enable steady co-existence of critical, sensitive, and best effort tasks in the same system – which is one of our key contributions in CHIPS-AHOy.

Fig. 2 highlights a motivational example showing how a holistic hypervisor can open doors to new optimization opportunities. The scenario shown in Fig. 2(a) is fairly optimal for best-effort applications, however, critical and sensitive ones usually fail to get minimal guarantees from the platform w.r.t. their timing requirements. In this case, even though a sensitive task needs certain performance requirements to meet its deadlines, as new best-effort (BE) tasks enter the system, the competition to access shared resources may cause it to miss deadlines mainly due to **(1)** interference caused by conflicts in the shared cache and **(2)** contention in the main bus triggered by BE I/O. Our holistic approach (Fig. 2(b)) overcomes these problems by techniques such as cache coloring **(3)** and I/O throttling **(4)** that isolate the critical tasks from the BE tasks. In this case, since the BE tasks have less available resources and cannot achieve the same performance level, a smart task mapping approach which is aware of the platform heterogeneity can detect this scenario and move the BE tasks to lower power cores **(5)**, thereby saving power and reducing system temperature.

The rationale behind the introduction of a predictable hypervisor in this paper is the holistic integration and consolidation of traditional predictable mechanisms of an RTOS with the state-of-the-art best-effort centric approaches by leveraging the recent advances in the design and implementation of sensor-actuator-rich computing platforms (e.g., [6, 7]) to enable the realization of next generation hypervisors for MPSoCs.

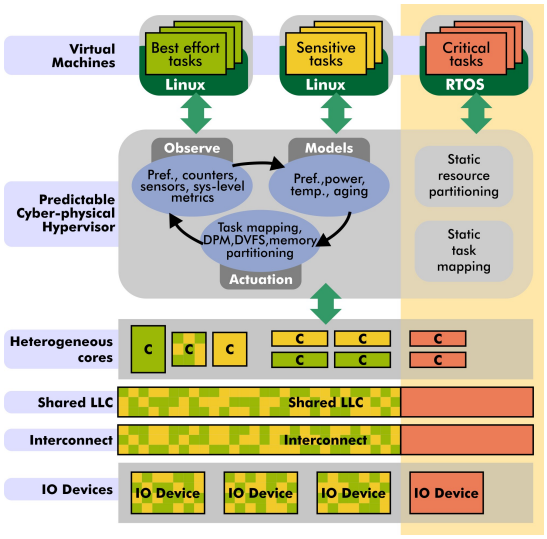


Figure 3: CHIPS-AHOy overview.

3 PREDICTABLE HOLISTIC CYBER-PHYSICAL HYPERVISOR

Providing means for management of multiple classes of applications with different timing and predictability requirements while utilizing the platform in an energy-efficient manner is a challenging task. We propose the *Predictable Cyber-Physical Hypervisor* (CHIPS-AHOy) as an exploratory attempt and proof-of-concept realization of a holistic hypervisor to address the challenges in such scenarios (Fig. 3). CHIPS-AHOy provides three classes of virtualized environments for applications with different predictability requirements: critical, sensitive, and best-effort. Resource partitioning, isolation, reliability, and power management are handled by the hypervisor in a holistic manner, following a self-aware C3 strategy similar to strategies in POET [6] and CPSoC [7]. This differentiates CHIPS-AHOy from traditional hypervisors in several ways. Traditional hypervisors multiplex hardware resources that are delivered to applications with minimal intervention under a strategy designed for performance and fairness [22, 23]. They largely ignore predictability, energy, and aging optimization opportunities, assuming guardbands typical of worst case design. They lack support for multi-level actuation and adaptation mechanisms to aggressively meet competing and conflicting application demands. CHIPS-AHOy overcomes these limitations through an ODA loop defined around three key ideas as shown in Fig. 3: **1) cross-layer sensing**; **2) predictive models**; and **3) self-aware actuation policies**. We describe these three concepts below.

Cross-layer sensing: CHIPS-AHOy leverages sensors available across multiple layers in the platform stack to take control decisions. At platform level, we leverage per-core PMUs available in most commercial platforms today to assess workload properties at runtime, such as number of instructions executed, cache misses, branch mispredictions, etc. CHIPS-AHOy also leverages physical sensors, such as temperature and power, to take actuation decisions. The notion of cross-layer sensing also includes virtual sensors, which

can be used to replace unavailable physical sensors (i.e., by using power/temperature models) and to encapsulate system-level metrics such as core utilization and core throughput [7].

Predictive models: Predictive models take as input sensed information and the current system state to predict the effect of an actuation action in the system. In contrast to previous works that use static models to predict the result of actuation [24], our models capture the dynamic behavior of system components (e.g. DVFS controller, multicore scheduler, I/O controller, etc), thus enabling different actuation policies to work in a holistic manner as described in Fig. 2(b).

Self-aware actuation: Self-awareness at the actuation level is enabled when policies are not only triggered by the instantaneous systems state, but also utilize inputs from the predictive models as described previously. In this work, we explore four main actuation mechanisms employed by CHIPS-AHOy to ensure predictability and improve the energy efficiency of the system: *real-time scheduling*, *cache partitioning through page coloring*, *I/O throttling*, and *reliability management*. We describe these mechanisms below.

3.1 Scheduling

Within the ODA loop shown in Fig. 3, VM scheduling is performed based on a holistic combination of factors, including a platform description, information about the category of each active VM, sensing information about power, thermal, and performance, and feedback from the predictive models. From the platform description, the scheduler is able to perform mappings based on the multicore CPU topology considering the different performance/power tradeoffs of each CPU (e.g. ARM's big.LITTLE). We follow the scheduler design proposed by [9] and the mapping approach proposed by [19] to achieve energy efficient core allocation while simultaneously delivering the predictability needed to support critical and sensitive VMs.

CHIPS-AHOy uses a multilevel queue with dynamic priority as the primary policy and round-robin as the secondary policy to schedule the virtual CPUs (VCPU) that are allocated to VMs. Each VM can run in one or more virtual CPUs and virtual CPUs are not shared by distinct VMs. VCPUs associated to critical and sensitive VMs are handled as periodic real-time tasks. When they are created, additional information about period, deadline, and utilization is provided to the scheduler, which applies the *Clustered Earliest Deadline First* (CEDF) [25] algorithm to dynamically determine their priorities at run-time. Best-effort VMs are not periodic and their VCPUs share the same, lower priority that is never reached by the periodic ones. They are therefore scheduled using the round-robin policy. An entity conceptually similar to the *idle thread* in ordinary systems is used to speculatively perform functions pertaining the ODA loop described earlier. Since the scheduling of these lowest priority flows only occur when a core has no VPCU to run, they present a strategic opportunity to perform sensing, prediction, and actuation of the ODA loop.

In the envisioned scenario of a few critical and sensitive VMs and many best-effort ones, energy efficiency is attained by carefully mapping best-effort VCPUs to cores as illustrated in Fig. 4. In adaptive periodic epochs (typically every 200ms), information from sensors and performance counters (Fig. 4(a)) is used as input for performance/power predictive models (Fig. 4(b)). Bin-based prediction models [19] are used to predict the performance and power of individual VCPUs across the heterogeneous cores in the system (e.g., predicting performance of a VCPU on a *little* core, given measurements done on a *big* core). These prediction models are similar to regression-based models that have been used in previous works for the same purpose [24, 26] and are trained offline. The performance/power predictions are then used as input to the dynamic predictive models (Fig. 4(c)) and the core allocation algorithm (Fig. 4(d)). The dynamic models are interactively queried by the scheduler to predict the effective performance and power consumption of the system given a new actuation decision (e.g. a new VCPU -> core mapping). These models account for the behavior of other actuators in the system (DVFS policy and memory partitioning). The goal of the core allocation algorithm is to find the most energy efficient allocation that meets the QoS or the performance constraints for every VCPU in the system. Since finding the optimal solution is NP-hard, we employ the list scheduling based heuristic described by [19] augmented with reliability management (Section 3.4) within the *idle* flows described earlier. Indeed, surges of such flows directly drives DVFS and VCPU migration towards less energetic levels, while their absence signals the scheduler to expedite more resources.

3.2 Page Coloring

Static partitioning and reservation of resources have been exploited as effective countermeasures to false sharing in the memory hierarchy. CHIPS-AHOy relies on a platform description to partition the memory hierarchy in order to isolate critical and sensitive VMs from interference caused by best-effort ones. The description includes the multicore CPU topology, the size and associativity of the caches, the size of buffers, and the topology of the DRAM. It is used to map page-colored memory to VMs according to their categories: critical VMs are assigned private colors, while sensitive and best-effort VMs share colors that are not used by the critical ones [9]. This scenario is depicted in Fig. 3.

Besides partitioning the memory hierarchy, CHIPS-AHOy uses available sensing mechanisms to collect run-time data in order to improve predictability. By observing cache coherence activities, for instance, the scheduler can detect best-effort tasks interfering with sensitive ones (that use the same color) and thus improve predictability and response time by limiting the amount of time a best-effort VM can run. Another possible action is migration: two interfering VMs running on different cores (or clusters) can be migrated so they reside on the same core, thus mitigating the impact of the coherence protocol on the interconnect and consequently reducing memory access time. If interference continues to grow and reaches a certain threshold, the hypervisor temporarily suspends a best-effort VM in favor of a sensitive one. Being fully isolated, critical VMs perceive very little interference in terms of memory access time as will be demonstrated in Section 4. Moreover, the limitation in terms of partition size is in tune with our envisioned scenario of many best-effort VMs and just a few critical ones.

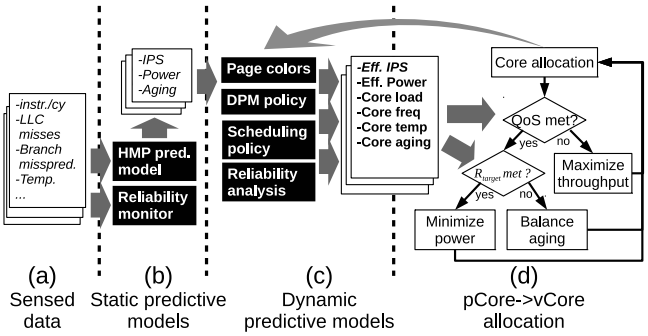


Figure 4: Holistic task mapping approach.

3.3 I/O Throttling

Some modern I/O interconnects support per-device QoS [16], which we fully exploit in CHIPS-AHOy. Critical VMs are mapped to cores that get a fixed share of the I/O bandwidth, while sensitive and best-effort ones run on cores that share the remaining fraction. I/O devices capable of overwhelming the interconnect, such as network adapters and disk controllers, are also given fixed shares. In addition, the interference of best-effort VMs on sensitive ones is constantly monitored through the PMU that is usually available in platforms featuring I/O QoS. If a sensitive VM starts to receive too much interference (configurable at VM creation time), a QoS reservation is activated for it just like if it were a critical VM. If no more bandwidth is available to be allocated, then the devices causing the interference in the name of best-effort VMs are put under a *Dynamic Power Management* (DPM) policy, eventually even being put in sleep mode. CHIPS-AHOy exposed I/O operations are intrinsically blocking, so VMs trying to interact with a suspended device will temporally block, immediately alleviating the interference on the sensitive ones.

For platforms that do not support I/O QoS nor feature I/O performance counters, a software solution is deployed. At machine initialization time, I/O devices capable of performing DMA are profiled to determine the typical duration of various size transfers (machine initialization is an interference-free scenario, so the obtained times are roughly best-case ones). During regular operation, I/O requests are time-stamped to implement a run-time monitor, which is used by the hypervisor just like a PMU. The main difference being that, without reservation, critical VMs are subjected to the same scenario as sensitive ones: they share the I/O interconnect with best-effort VMs until interference is detected. When this happens, before applying DPM to I/O devices associated with best-effort VMs, CHIPS-AHOy first tackles the DMA controller capabilities to adjust burst lengths according to VM categories. Critical VMs continue to use the optimized performance/energy values, while best-effort ones are limited to very short bursts, with sensitive ones falling in between (operating the I/O subsystem always on small bursts is not an option, since this results in poor performance and energy efficiency). The effectiveness of this approach is demonstrated in Section 4.

3.4 Reliability Management

In addition to considering heterogeneity, cache, and I/O interference as described previously, the performance and power models also take into account the life-time reliability requirements. We therefore incorporate reliability monitors and analysis units in the hypervisor to enrich the core allocation algorithm (Fig. 4(c)). Typically, systems are expected to work for a planned service life, expressed in a number of years. Thus, we may use the classical stochastic reliability model [27, 28] to estimate the probability that the system will survive until the specified lifetime, which is modeled by means of the Weibull distribution: $R(t) = e^{-\left(\frac{t}{\alpha(T)}\right)^{\beta}}$
We augment the scheduler to consider aging effects due to temperature variations using the extended model presented in [28]. We then customize and incorporate the technique presented in [18] to provide the scheduler with reliability awareness. To indicate the expected lifetime and to measure system reliability during its operational life, we express a minimum reliability level R_{target} the system must fulfill at the end of the service life t_{target} (as in [29]).

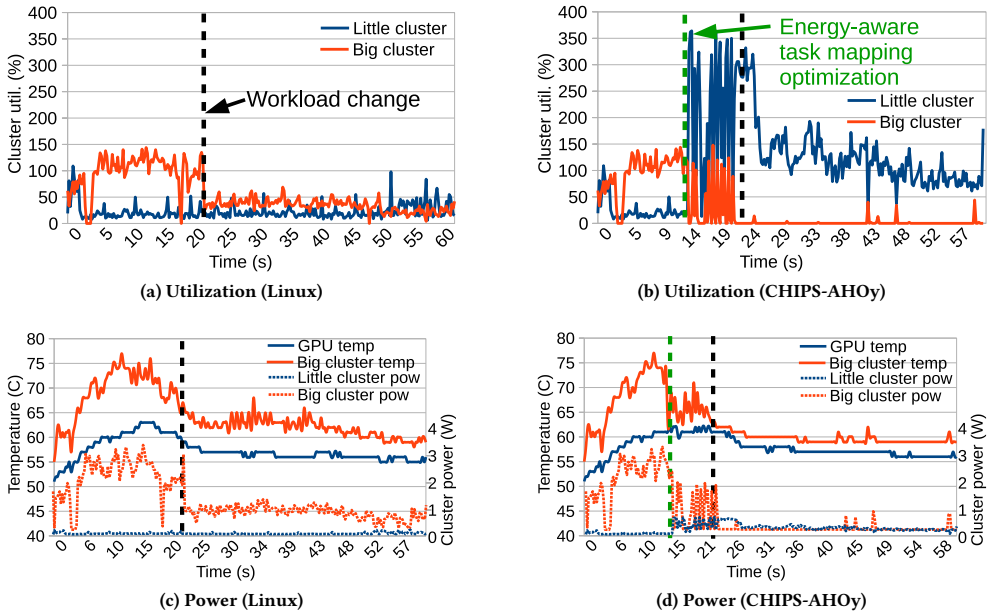
CHIPS-AHOy monitors the cores' aging status and provides reliability metrics to its scheduler. It defines a target reliability curve $R_{target}(t)$, called *aging reference*, on the basis of the required reliability target $R_{lifetime}$ at the given lifetime $t_{lifetime}$. Then, every long-term epoch, the unit computes for each core $n_{w,h}$, a metric $\Delta R_{w,h}(t)$ measuring the divergence of the current reliability, received by the reliability analysis unit, and its target value.

The aim of incorporating life-time reliability awareness in CHIPS-AHOy is to optimize system performance, while guaranteeing reliability requirements. In fact, when considering only performance and energy requirements, a subset of the cores would be generally preferred for applications execution, thus leading to a non-balanced cores' aging and poor predictability. On the other hand, when considering only reliability requirements, we would cause a high dispersion in application mapping, and, consequently, a considerable on-chip interconnect congestion and low system performance.

We integrate the reliability monitor with core allocation to applications as described in Fig. 4. The reliability monitor and analysis units compute $\Delta R_{w,h}(t)$ as a function of the measured core temperature and the core allocation. The first order optimization goal is to meet the performance constraints of the application. The second order goal is to minimize power while keeping the aging profile balanced. We use $\Delta R_{w,h}(t)$ to compute a scaling factor $\Delta R_{w,h}(t)/\Delta R_{min}(t)$, where R_{min} is the smallest $\Delta R_{w,h}$ across all cores in the platform, applied to the core power estimated by the predictive models; thus cores with unbalanced aging are perceived by the core allocation algorithm as less power efficient and therefore less likely to be allocated.

4 EXPERIMENTAL RESULTS

We present a comparison between CHIPS-AHOy mechanisms for managing hybrid workloads in a holistic and deterministic fashion against traditional approaches in 4 scenarios: 1) HMP core allocation, 2) I/O monitoring and throttling, 3) Isolation of critical and sensitive VMs via page coloring, and 4) Lifetime reliability management. We compare a baseline Linux kernel with a proof-of-concept CHIPS-AHOy prototype on three real platforms and a long-term aging simulation.



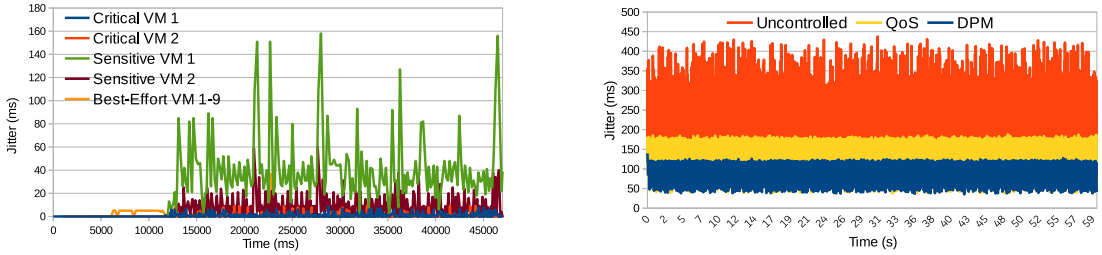
(a) Utilization (Linux)

(b) Utilization (CHIPS-AHOy)

(c) Power (Linux)

(d) Power (CHIPS-AHOy)

Figure 5: Traditional Linux (using GTS) vs CHIPS-AHOy core allocation for HMPs.



(a)

(b)

Figure 6: (a) Impact of I/O overutilization on a task-set with a few critical/sensitive tasks and several BEs. (b) under different execution regimens: uncontrolled, QoS and DPM .

4.1 HMP Core Allocation

Fig. 5 shows that the core allocation approach employed by CHIPS-AHOy adapts at run time to improve energy efficiency in the context of an HMP through a real implementation on 8-core ARM's big.LITTLE platform (4×big+4×Little OdroidXU3). In this experiment, we run 4 instances of PARSEC's x264 application with different QoS requirements, 2 apps at 5 *fps* and the other 2 at 15 *fps* input rate. The black dashed line in Figs. 5a-d shows when the 15 *fps* app finish and the load on the system changes. Figs. 5b and 5d show how the system behavior changes when CHIPS-AHOy's core allocation algorithm is employed (denoted by the green dashed line). By using perf/power models, CHIPS-AHOy identifies that the little cores are able to run x264 applications without performance degradation, thus reducing power consumption and improving energy efficiency. Figs. 5c and 5d show how these optimizations impact the thermal profile of the system. On average, CHIPS-AHOy improves energy consumption by 65% without performance degradation for this case study.

4.2 I/O Monitoring and Throttling

CHIPS-AHOy is capable of sensing the jitter increase in I/O-bound tasks running on VMs, identifing potential damage to the critical VMs timing requirements, and evaluating hypervisor actions to maintain the predictability of the system. In this experiment, we run 13 tasks (2 critical, 9 BE, and 2 sensitive), with a total utilization of 128% on CPU operations and of 130% on I/O operations, on a Zynq-7000 (dual-core ARM Cortex-A9) with HPAXI traffic generators. Critical tasks run on VMs under a RTOS, while sensitive ones mimic the x264 Linux workload used previously, and BEs mimic Linux background tasks. The task-set runs for 47 s, with a 1 s activation delay for each task. Fig. 6(a) shows the jitter in the execution time of each task caused by an uncontrolled (i.e., FIFO) I/O regimen. As

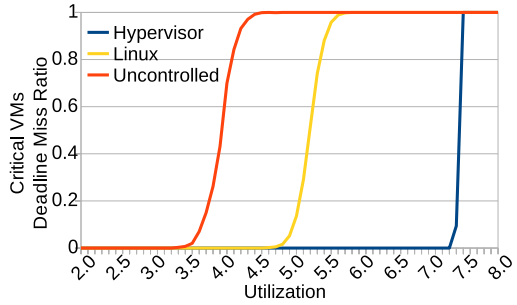


Figure 7: Critical VMs scheduled naively, by Linux, and by CHIPS-AHOy.

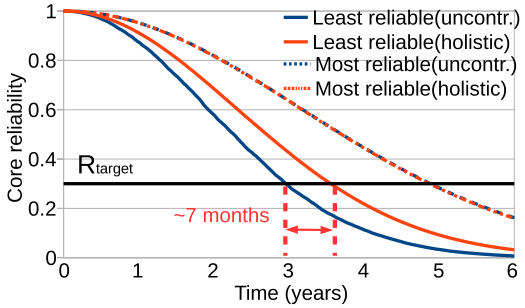


Figure 8: Linux HMP mapping aging vs CHIPS-AHOy life-time reliability-aware HMP mapping.

more tasks performing I/O are released, the total jitter increases to unpredictable levels.

Fig. 6b shows the jitter imposed on a similar task-set scheduled under three different regimens: uncontrolled (i.e., Linux-like), CHIPS-AHOy doing QoS by limiting the burst length of peripherals associated to non-critical tasks, and CHIPS-AHOy powering off peripherals associated to non-critical tasks (DPM). The task-set used has lower utilization than the previous one, since the goal here was not to exacerbate the problem, but to demonstrate two potential solutions in a more realistic scenario. The total I/O utilization was adjusted to 90%. As can be observed, running multiple I/O-intensive tasks on VMs without monitoring and throttling the interconnect to protect critical and sensitive VMs can cause operations that in the best case would finish in 50 ms to be delayed for up to 400 ms, possibly disrupting their requirements.

4.3 C-EDF with Page Coloring

In CHIPS-AHOy, individual cache partitions (i.e., colors) are exclusively assigned to critical tasks which are scheduled using a C-EDF policy. We compared this mapping with two others on an Intel i7-2600 with 8 cores: a global scheduling policy, in which tasks run on the first available core, the ordinary Linux scheduler, in which tasks are grouped in clusters (cgroups). We randomly generated task-sets similar to [9], with each task allocating memory for its working set (WSS) and subsequently reading and writing to it randomly. Fig. 7 shows the ratio of missed deadlines as we increase the utilization cap of the task-sets. A ratio of 0.6, for instance, means that 60% of the total generated task-sets missed at least one deadline. In contrast with the two less controlled scenarios, CHIPS-AHOy can take the utilization close to the machine's capacity without disrupting critical VMs' requirements.

4.4 Life-time Reliability Management

Fig. 8 demonstrates the efficacy of CHIPS-AHOy on the aging and reliability profile of a heterogeneous multiprocessor system. We use the same workload and experimental setup from Section 4.1 integrated with the aging simulation method presented in [28]. The simulator uses the reliability model presented in [28] and calculates the reliability of each core from the thermal traces collected at runtime from the physical on-chip thermal sensors on ARM's big.LITTLE platform (OdroidXU3). In other words, as aging analysis is a long-term process, we use an analytical simulation method where the thermal sensors' data is collected from a real hardware, but analyzed using an aging simulator. We assume the required lifetime $t_{target} = 6$ years, and, in order to have a reasonable reliability at the end of the operational life, we set a per-core target reliability $R(t_{target}) = 30\%$ (similar to [30]). The chip floorplan has been defined according to available specifications on the ARM's big.LITTLE platform: a squared grid floorplan, and the chip area of $122mm^2$ ($22nm$ technology). The simulator uses the reliability model presented in [28] and calculates the reliability of each core from the thermal traces collected at runtime from the physical on-chip thermal sensors on ARM's big.LITTLE platform. Fig. 8 compares the results when an unmodified Linux kernel (GTS task mapping [21], interactive DVFS governor) is used with the case when GTS task mapping is controlled by CHIPS-AHOy, which uses inputs from on-chip performance counters and models for performance, power, and reliability to perform life-time reliability aware scheduling. Each graph reports the minimum and the maximum reliability values of the cores within the architecture. Fig. 8 shows that CHIPS-AHOy is able to prolong the system life-time by 20%. CHIPS-AHOy minimizes the variance in the reliability values thus maximizing the overall lifetime of the system. In contrast, since the unmodified Linux kernel is reliability-agnostic, it distributes the applications without considering the aging values, and therefore leads to an unbalanced distribution of the workload and, consequently, aging of the cores. This leads to a lower reliability of some cores that probabilistically will fail earlier.

5 CONCLUSIONS

Many emerging CPS and IoT applications such as autonomous cars pose unique constraints based on different classes of application tasks (requiring hard, soft and best-effort deadlines) that demand strict temporal predictability, while delivering high performance in the face of multi-dimensional design constraints. To address this challenge, we presented CHIPS-AHOy, a predictable holistic cyber-physical hypervisor for heterogeneous MPSoCs that enables synergistic virtualization of critical, sensitive, and best-effort tasks, while managing disparate constraints such as energy minimization, thermal bounds, and system lifetime management. CHIPS-AHOy holistically integrates several novel features, including cross-layer sensing and actuation, predictive models, and self-awareness embodied in the observe-decide-act paradigm. We presented four specific self-aware actuation mechanisms: scheduling,

resource partitioning via page coloring, I/O throttling, and reliability management. Our experimental results on three real multi-core platforms (ODROID, Zynq and Intel I7) demonstrate the ability of CHIPS-AHOy to manage these hybrid workloads in a holistic and predictable manner in the face of variability imposed by the shared memory hierarchy, the shared I/O subsystem, and the hardware substrate. Our future work will extend our platform i) to manage unpredictability for Networks-on-Chip based MPSoCs, ii) to analyze the scalability of the hypervisor in the many-core system era, iii) to manage the power consumption in a holistic manner to be leveraged as another knob to enhance predictability, and iv) to perform sensitivity analysis against variations in workload volume and distribution.

ACKNOWLEDGMENTS

We acknowledge financial support from the following: NSF Grant CCF-1704859; and the Marie Curie Actions of the European Union's H2020 Programme.

REFERENCES

[1] M. B. Taylor, "Is dark silicon useful?" in *Proc. of DAC*, 2012.
[2] "Fiasco micro-kernel." http://os.inf.tu-dresden.de/fiasco/
[3] "INTEGRITY Multivisor." http://www.ghs.com/products/rtos/integrity_virtualization.html
[4] S. Xi *et al.*, "Real-time multi-core virtual machine scheduling in xen," in *Proc. of EMSOFT*, 2014.
[5] J. Henkel *et al.*, "New trends in dark silicon," in *Proc. of DAC*, 2015.
[6] C. Imes *et al.*, "POET: a portable approach to minimizing energy under soft real-time constraints," in *Proc. of RTAS*, 2015.
[7] S. Sarma *et al.*, "CyberPhysical-System-On-Chip (CPSoC): A Self-Aware MPSoC Paradigm with Cross-Layer Virtual Sensing and Actuation," in *Proc. of DATE*, 2015.
[8] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *Proc. of ISORC*, 2008.
[9] G. Gracioli *et al.*, "On the design and evaluation of a real-time operating system for cache-coherent multicore architectures," *SIGOPS Oper. Syst. Rev.*, vol. 49, 2016.
[10] G. Gracioli *et al.*, "A survey on cache management mechanisms for real-time embedded systems," *ACM Comput. Surv.*, vol. 48, 2015.
[11] B. Akesson *et al.*, "Architectures and modeling of predictable memory controllers for improved system integration," in *Proc. of DATE*, 2011.
[12] H. Yun *et al.*, "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Trans. Comput.*, vol. 65, 2016.
[13] *PCI Express Base Specification (Rev. 3.1)*, PCI-SIG, 2014.
[14] S. Bak *et al.*, "Real-time control of I/O COTS peripherals for embedded systems," in *Proc. of RTSS*, 2009.
[15] R. Pellizzoni *et al.*, "Impact of peripheral-processor interference on WCET analysis of real-time embedded systems," *IEEE Trans. on Comp.*, vol. 59, 2010.
[16] *AMBA® Network Interconnect (NIC-301) Technical Reference Manual (Revision: r2p0)*, ARM, 2009.
[17] ITRS, "Process Integration, Devices, and Structures (PIDS)," ITRS, Tech. Rep., 2011.
[18] M. H. Haghbayan *et al.*, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *Proc. of DATE*, 2016.
[19] B. Donyanavard *et al.*, "SPARTA: Runtime Task Allocation for Energy Efficient Heterogeneous Many-cores," in *Proc. of CODES*, 2016.
[20] Y. Zu *et al.*, "Adaptive guardband scheduling to improve system-level efficiency of the POWER7+," in *Proc. of MICRO*, 2015.
[21] ARM, "big. LITTLE Technology : The Future of Mobile," ARM, Tech. Rep., 2013.
[22] P. Barham *et al.*, "Xen and the art of virtualization," in *Proc. of SOSP*, 2003.
[23] A. Kivity *et al.*, "KVM: the linux virtual machine monitor," in *Proc. of OLS*, 2007.
[24] A. Annamalai *et al.*, "An opportunistic prediction-based thread scheduling to maximize throughput/watt in AMPs," in *Proc. of PACT*, 2013.
[25] T. P. Baker *et al.*, "Schedulability analysis of multiprocessor sporadic task systems," in *Handbook of Realtime and Embedded Systems*. CRC Press, 2007.
[26] T. Muck *et al.*, "Run-DMC: Runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency," in *Proc. of CODES+ISSS*, 2015.
[27] JEDEC Solid State Tech. Association, "Failure mechanisms and models for semiconductor devices," *JEP122G*, 2010.
[28] C. Bolchini *et al.*, "A lightweight and open-source framework for the lifetime estimation of multicore systems," in *Proc. of ICCD*, 2014.
[29] P. Mercati *et al.*, "Dynamic variability management in mobile multicore processors under lifetime constraints," in *Prof. of ICCD*, 2014.
[30] L. Huang *et al.*, "Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint," in *Proc. of DATE*, 2010.