E-WarP: System-wide Framework for Memory Bandwidth Profiling and Management

Parul Sohal , Rohan Tabish†, Ulrich Drepper‡ and Renato Mancuso
Boston University, US , {psohal, rmancuso}@bu.edu
†University of Illinois at Urbana-Champaign, US , rtabish@illinois.edu
‡Red Hat, drepper@redhat.com

2020 IEEE Real-Time Systems Symposium (RTSS) | 978-1-7281-8324-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/RTSS49844.2020.00039

***bstract*—The proliferation of multi-core, accelerator-enabled embedded systems has introduced new opportunities to consolidate real-time systems of increasing complexity. But the road to build confidence on the temporal behavior of co-running applications has presented formidable challenges. Most prominently, the main memory subsystem represents a performance bottleneck for both CPUs and accelerators. nd industry-viable frameworks for full-system main memory management and performance analysis are past due.**

In this paper, we propose our *Envelope-aWare Predictive model*, or E-WarP for short. E-WarP is a methodology and technological framework to: (1) analyze the memory demand of applications following a profile-driven approach; (2) make realistic predictions on the temporal behavior of workload deployed on CPUs and accelerators; and (3) perform saturation-aware system consolidation. This work aims at providing the technological foundations as well as the theoretical grassroots for truly workload-aware analysis of real-time systems. We provide a full implementation of our techniques on a commercial platform (NXP S32V234) and make two key observations. First, we achieve, on average, a 6% overprediction on the runtime of bandwidth-regulated applications. Second, we experimentally validate that the calculated bounds hold if the main memory subsystem operates below saturation.

I. Introduction

The proliferation of inexpensive and high-performance multi-core embedded platforms have been enthusiastically embraced by the industry. These are seen as an opportunity to migrate away from system designs with many interconnected single-core chips; to consolidate all the application workload onto a few systems-on-chip (SoC) with multiple CPUs and accelerators. nd while the transition has been smooth for general purpose workload, the same cannot be stated for safety-critical systems.

It is well known that contention over shared hardware resources leads to substantial violation of temporal properties when workload developed and tested in isolation is consolidated on the same multi-core platform. Effects like shared cache contention [1], [2], DR M bank conflicts [3], [4], contention at the DDR controller [5] have significantly slowed down the adoption of multi-core solutions in the safety-critical domain. The presence of performance *interference channels* has been acknowledged by certification authorities [6], that have mandated methodologies to "account and bound" the temporal effect of interference channels for the certification of avionic systems.

The last decade has produced seminal results [7] on techniques to manage contention at the different levels of the memory hierarchy. But unfortunately, there is a substantial lack of frameworks and methodologies that can be applied system-wise to: (1) take into account realistic applications, (2) consider that processing workload does not occur only on CPUs; accelerators (e.g. DM s, video-encoders, GPUs) are also fundamental components in real systems, and (3) that can be deployed on existing platforms while ensuring that the models assumed to derive analytical guarantees are in match with the true behavior of the hardware.

In concurrent activity of CPUs and accelerators, main memory is the performance bottleneck. Thus we set our focus on the problem of contention in the DR M subsystem. DR M bandwidth management [8], [9] is a promising grassroots technique to exert control over main memory contention. Many works have studied the behavior of applications in multi-core systems under main memory bandwidth regulation [4], [10], [11]. But the overwhelming focus of these works has been put on formulating an increasingly more refined model of the DR M subsystem [4], [11] to reduce the pessimism in the timing analysis. On the other hand, the behavior of applications is abstracted away with only a few parameters, for instance, to summarize the worst-case end-to-end number of cache misses [4], [5], [12].

In this paper, we propose a focus-shift. We introduce a comprehensive framework of techniques called *Envelope-aWare Predictive model*, or E-WarP for short. In E-WarP, accurate predictions on the worst-case execution time (WCET) of co-running applications are made following a profile-driven approach. Profiling represents a substantial refinement of measurement-driven approaches, where fine-grained knowledge of the interaction between applications and the platform is collected and leveraged. Conversely, we treat the DR M subsystem, as much as possible, as a black-box. By shifting our emphasis on a more precise representation of memory bandwidth requirements of applications and by ensuring that the DR M subsystem operates below its saturation threshold, we demonstrate that highly accurate predictions on the behavior of tasks operating on CPUs and accelerators can be made.

We stress upfront that we do not construct a formal model of the DR M subsystem, nor formulate provable guarantees. The correctness of our approach is corroborated by a full-system evaluation, which provides evidence that the work presented is practical for industrial applications. Furthermore, our profile-driven approach enables a better understanding of the important aspects that have traditionally received little attention. Precise regulation overheads, impact of burst size on DR M utilization, and the unexpected presence of memory instructions that bypass regulation are some examples. The proposed E-WarP framework can be used to integrate multi-core, accelerators-enabled real-time systems in all those domains where a measurement-based approach was deemed acceptable for single-core systems.

In summary, this paper makes the following contributions. (1) It introduces the E-WarP model where the time-varying demand for main memory resources is characterized via envelopes. (2) It introduces key requirements and design principles for profile-driven approaches. (3) It considers the integration of broadly implementable techniques for DR M

2576-3172/20/$31.00 ©2020 IEEE
DOI 10.1109/RTSS49844.2020.00039

bandwidth regulation of CPUs and accelerators. (4) It describes how to leverage memory enveloping to perform accurate WCET predictions under regulation for both CPU and accelerator workload. (5) It provides a technique to reason on the saturation level of the DR M subsystem. (6) Lastly, it proposes a full-system implementation and evaluation that includes a low-overhead profiler and an augmented partitioning hypervisor.

II. Rel ted Work

There has been a plethora of research works [4], [5], [10], [13] that aimed at providing hard real-time guarantees for tasks running on multi-core systems. common denominator in these works is that they consider the worst-case number of main memory transactions (LLC misses) for tasks in isolation [5], [10]–[12]; then compute an upper-bound on memory interference when multiple applications run in parallel. This type of analysis has been proposed with various degrees of refinement on different DR M/CPU models. For instance, in [4] the authors assume that there is only one outstanding request per CPU; while [5] focuses on the First-Ready First-Come First-Served (FR-FCFS) DR M scheduling policy. Compared to this line of work, E-WarP is substantially different because its premise is to rely on high-accuracy observations of the memory demands of applications, while treating the DR M subsystem mostly as a black-box.

Other works such as [8], [9], [14] focus on implementable mechanisms to regulate/throttle the bandwidth of other low criticality tasks with the goal of reducing contention and improving performance isolation. The first work in this sense was [8], where budget-based bandwidth enforcement is proposed. The work in [9] builds on this technique by allowing high-priority tasks to acquire a "bandwidth lock" on the memory controller. These techniques have also been shown to be implementable at the hypervisor level [14], [15]. Recently, there have been important efforts to control, account, and ultimately integrate the behavior of accelerators into real-time systems. The work in [16] lays the groundwork for managing hardware accelerator defined in FPG , while [17] touches on the topic of non-CPU components regulated via platform-specific throttling mechanisms. In many ways, E-WarP builds on top of the seminal results achieved in this context and complements the CPU-centric management by integrating traditional accelerators (e.g., DM s, GPUs) in the picture.

Finally, the need for a DR M controller capable of enforcing bandwidth partitioning and traffic prioritization has been expressed in multiple papers [18]–[21]. We acknowledge the important design principles proposed in said works. However, as we strive for immediate industrial applicability, we restrict ourselves to commercial-off-the-shelf platforms.

In summary, our work sets itself apart because it proposes a novel profile-driven methodology to characterize the behavior of applications that execute on CPUs and accelerators. It then combines (1) CPU-centric bandwidth regulation techniques with (2) broadly available hardware support for regulation of non-CPU masters. In doing so, key relationships between extracted bandwidth and saturation of the DR M subsystem are derived. Finally, a full-system integration is proposed where we demonstrate that E-WarP is practical in real systems.

III. System Model nd ssumptions

We consider a heterogeneous multi-core system with accelerators and traditional CPUs. hardware accelerator can be any module capable of initiating transactions to the main memory. DM engines, GPUs, video encoders/decoders, audio sequencers, network interfaces, are some notable examples. We use m to indicate the number of CPUs present in the system and the index $k \in \{1, \ldots, m\}$ to refer to a given CPU_k. The system also features a accelerators indexed using l, with $l \in \{1, \ldots, a\}$. The l-th accelerator is indicated with ACC_l. We use *processing element* (PE) whenever what stated applies to both CPU and accelerator.

We make a restriction, namely the *single driver assumption*, on how accelerators are used in our system. We assume that there exists a single CPU task that acts as the driver for a given accelerator. I.e., it must hold that for ACC_l there exists at most one CPU task acting as the driver. The assumption allows us to abstract away the differences in the preemption model of accelerators. The single driver assumption is accurate only in a subset of possible system designs, but it allows us to keep our focus on how accelerators interact with the main memory. For the same reason, we make the assumption that caches [15], [22] and DR M banks [3], [4] are statically partitioned on a per-core basis to ensure that the load generated by each application toward main memory does not change when multiple applications execute in parallel.

We assume that only one main memory controller is used by all the tasks under analysis. This is referred to as the "DDR controller", or the "DR M controller". If more than one controller exists, the techniques presented in this work can be extended by partitioning tasks to memory controllers, and then considering each sub-system independently. We assume that the traffic originated by CPU and accelerators towards main memory can be regulated. We use budget-based periodic regulation (MemGuard [8]) to manage traffic from the CPU; we leverage standard RM QoS support that is broadly available in modern RM-based SoCs to regulate traffic from accelerators (see Section VI-B). Lastly, the bandwidth at the interconnect should be greater than the bandwidth of both memory controllers.

IV. E-W rP T sk Model

The E-WarP task model incorporates the relationship between a task's progress and its demand for main memory. This relationship, expressed via *cumulative memory envelopes*, is captured for each task in isolation. It is leveraged to derive precise predictions on the behavior of the task under regulation. Section VIII is dedicated to constructing memory envelopes following a profile-driven approach.

We consider a set of n sporadic, deadline-constrained real-time tasks scheduled according to fixed-priority. The generic task τ_i is statically assigned to execute on a given CPU_k — partitioned fixed-priority scheduling. task τ_i is a tuple of the following form: $\tau_i = \{T_i, D_i, C_i, \mathcal{M}_i\}$. T_i represents the minimum inter-arrival time between two jobs of the same task, D_i is the relative deadline of task τ_i, and C_i captures the worst-case execution time (WCET) of τ_i in isolation and without memory bandwidth regulation. The $\mathcal{M}_i$ parameter is a super-set of memory envelopes, one per each PE that the task uses. Each memory envelope $M_j \in \mathcal{M}_i$ is of the form $\{R_j, \sigma_j\ 1), \ldots, \sigma_j\ L_i)\}$. Here, L_i is simply the number of $\sigma_j\ h)$ elements that compose the envelope, while each

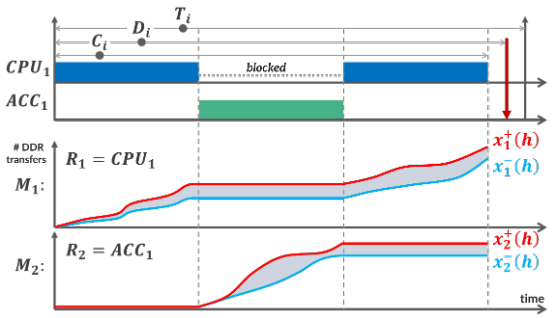

Fig. 1. Overview of main parameters in the E-WarP model for a generic task executing on CPU and ACC .

$\sigma_j\ h)$ captures the activity of the task over a fixed small time interval δ. It follows that $L_i = \lceil C_i/\delta \rceil$. The generic $\sigma_j\ h)$ has the structure $\{x_j^+\ h), x_j\ \ h)\}$, where $x_j^+\ h)$ (resp., $x_j\ \ h)$) captures the upper-bound (resp., lower-bound) on the cumulative number of memory transactions at time $h \cdot \delta$ for the task in isolation and without regulation. Lastly, R_j keeps track of the PE (CPU_k or ACC_l) on which the transactions are executed. Figure 1 provides a visualization of the main parameters in the E-WarP model.

Note that in principle the E-WarP model is capable of encapsulating the behavior of input-dependent applications. That is, as long as profiling is performed over a set of input vectors that exercise those execution paths leading to the worst-case memory envelope. To identify a set of representative input vectors, one could use approaches like symcretic execution [23].

V. Tr nsp rent Profiling

In the E-WarP methodology, the profiler plays a key role to (1) define the memory envelopes of applications, (2) study the saturation point of the DDR subsystem, and (3) differentiate between the behavior in main memory of different PEs. To precisely measure these quantities, the profiler must be designed to satisfy the ***transparency requirement***. Under this requirement, the task under observation is not (or minimally) impacted by the activity of the profiler. On the other hand, the higher the granularity of the profiler, the smaller will be the pessimism on WCET estimations. This corresponds to the ***fine-granularity requirement***. Indeed, an extremely coarse profiler degenerates into a model where the entire activity of the task is summarized by a single value for the worst-case number of transactions.

The definition of a good profiler is challenging because the two requirements are opposing objectives. Indeed, to achieve fine granularity, the profiler needs to frequently sample DDR performance and to keep in memory a complete history of the acquired samples. Thus, a fine-grained profiler acts as a *memory bomb*. We briefly outline the principles according to which a profiler that satisfies both requirements can be designed and implemented. We describe our software-only implementation in Section IX.

To satisfy the fine-granularity requirement, the target platform must provide a performance monitoring interface to sample key metrics of the DDR activity. The closer the interface is to where the transactions are served, the higher the accuracy of the resulting profile. Modern embedded platforms include extensive facilities for performance monitoring [24]. Some of these interfaces, such as the RM Performance Monitoring Unit (PMU), are broadly known and supported in software. Often times, however, there exist better interfaces that operate much closer to main memory. few notable examples are discussed in the following paragraph.

The P- and T-series of NXP embedded platforms [25], [26] have been extensively studied in the literature [3], [10], [12], [27], [28]. These platforms include an Event Processing Unit (EPU) and a DDR debug subsystem. The DDR debug subsystem can be configured to generate a trace of events at the DDR controller(s) that includes performed reads, writes, DR M refreshes, DR M row hit/miss events, and so on. The trace can be processed on chip to create custom event counters.

The DDR trace can also be stored in memory for later retrieval [29], [30]. The Xilinx Zynq UltraScale+ family platforms [31] that are surging in popularity in the recent years [32], [33] include an XI Performance Monitor (PM) that is interposed between the interconnect and the DDR and that well fits our requirement. The PM can measure the exact number of bytes read/written, as well as their max/min latency over a user-specified sampling interval [31]. Unsurprisingly, support for fine-grained monitoring close to memory resources is not limited to embedded platforms. Intel has recently introduced its family of memory monitoring and management techniques under the name of Resource Director Technology (RDT) [34]. RDT includes support to monitor the memory bandwidth extracted by CPUs via the Memory Bandwidth Monitoring (MBM) interface [35]. On top of the families of platforms mentioned above, yet another example is the NXP S32V234 (NXP S32V family) platform [36] targeted in our implementation.

To ensure transparency, the platform must allow storing the profiled samples without introducing spurious DR M traffic. Fortunately, modern embedded platforms feature heterogeneous memory subsystems, with two common features that can be leveraged. (1) The presence of fast on-chip scratchpad memories (SPM); and (2) the existence of multiple DDR controllers. Both are valid alternatives. But the limited size of SPMs restricts the length of profiled application, and/or the granularity of the profile. The NXP P- and T-series family of platforms, the Xilinx Zynq UltraScale+ SoCs, the NXP S32V and S32G family of platforms all define both multiple DDR controllers and on-chip SPMs. key takeaway is that fine-grained transparent profiling is possible today in a range of modern platforms. sound implementation requires careful consideration of platforms-specific features and the flow of data within the memory hierarchy. Nonetheless, this level of knowledge of the underlying hardware is not uncommon in the development cycle of safety-critical systems.

VI. System-wide B ndwidth Regul tion

To achieve system-wide control over memory bandwidth allocation, with the goal of keeping the DR M subsystem below its saturation threshold, we combine two different mechanisms for CPUs and accelerators, respectively.

. *Regulating CPU Memory Traffic*

The first mechanism is budget-based periodic regulation following the MemGuard [8] approach to regulate CPU memory traffic. MemGuard defines a global *regulation period* P and a per-core budget of last-level cache line refills Q_k. The performance measurement unit (PMU) is used to keep track of per-core cache line refills since the beginning of the current P.

local interrupt is delivered by the PMU to stop the core until