Bottleneck-Aware Bandwidth Regulation for Heterogeneous Memory Subsystems

Ivan Izhbirdeev*, Weifan Chen*, Heechul Yun†, and Renato Mancuso*
*Boston University †University of Kansas
{ivani, wfchen, rmancuso}@bu.edu heechul.yun@ku.edu

***Abstract*—Memory bandwidth regulation is critical for predictability on multi-core systems, where co-running tasks interfere through shared memory resources. Existing approaches typically regulate bandwidth at the core or system level. They often assume a single shared memory controller, without distinguishing which parts of a complex, potentially heterogeneous memory subsystem are being stressed. This paper introduces a bottleneck-aware bandwidth regulation approach for heterogeneous memory subsystems that combines spatial task separation, runtime bandwidth observation and regulation, and a general theoretical shared-resource model to reason about bandwidth guarantees. We implement our system on a Xilinx Zynq UltraScale+ ZCU102 platform and evaluate it on bank-partitioned workloads. Compared to conservative bank-oblivious regulation techniques, our approach preserves the guaranteed bandwidth of protected tasks while imposing less restrictive limits on co-runners, increasing total system bandwidth. Overall, this work introduces novel bottleneck-aware regulation that improves shared resource utilization without sacrificing the ability to reason about guarantees.**

***Index Terms*—bandwidth regulation, heterogeneous memory, DDR4 banks, multicore, coherence**

I. Introduction

The deployment of computationally demanding workloads on modern autonomous and embedded real-time systems has necessitated a shift toward high-performance multicore architectures. However, achieving stringent temporal predictability on these platforms remains fundamentally challenged by memory cross-core interference. When concurrent tasks simultaneously execute on separate processing cores, their memory requests inevitably contend for shared microarchitectural structures within the underlying memory hierarchy. This shared resource contention leads to high latency variations and execution time inflation, severely complicating worst-case execution time (WCET) reasoning and runtime Quality-of-Service (QoS) guarantees.

To mitigate memory-system interference, the real-time systems community has extensively investigated software- and hardware-based memory bandwidth regulation [1]–[5]. Traditional regulation mechanisms commonly rely on performance monitoring counters (PMCs) or dedicated hardware units to throttle overly aggressive cores once they exhaust a pre-assigned data budget over an accounting window. These mechanisms determine *how* traffic is observed and throttled; the budget-selection policy determines *how much* each actor may safely consume. Despite their practical efficacy, a major drawback of existing regulators is that they model the entire main memory subsystem as a monolithic, abstract bottleneck. By treating all memory transactions uniformly, these architectures operate in a space-oblivious manner. They inherently ignore the complex, parallel topologies of modern heterogeneous platforms, which often feature distinct system interconnect crossbars, multiple standalone memory controllers, and localized scratchpad or on-chip configurations. Consequently, baseline system-wide regulation must adopt overly pessimistic, conservative budgets that heavily degrade best-effort task throughput even when their requests traverse non-interfering or only partially interfering physical paths.

Concurrently, DRAM bank partitioning has emerged as an effective spatial isolation mechanism [6]–[10]. By applying page-coloring strategies and careful control over the physical-to-bank address mapping, systems can allocate dedicated, non-overlapping physical DRAM banks to individual cores or tasks. While this spatial separation completely eliminates direct bank-level conflicts, empirical evaluation reveals that bank isolation alone is insufficient for ideal performance predictability [6]. Even when workloads target entirely distinct banks, their requests must still compete within shared upstream segments of the memory hierarchy, including the shared last-level cache (LLC) snoop arbitration paths, central system interconnect, and the command scheduling queues inside the memory controller. Because spatial separation cannot dynamically scale the physical bandwidth bounds of these shared upstream nodes, an aggressive best-effort task can still saturate the controller's internal queues, imposing severe performance degradation on bank-isolated real-time workloads.

This paper bridges the gap by introducing *HeterCoRe*, a framework that combines spatial bank partitioning with a novel, bottleneck-aware runtime bandwidth regulation strategy. Rather than treating memory as a single point of failure, we present a general theoretical shared-resource model capable of more precisely tracking how bandwidth is extracted in a complex, potentially heterogeneous memory subsystem instead of treating it as a monolithic blackbox. We also propose scenario-based feasibility tests that incorporate the details of the memory sub-systems to assist in finding an improved yet safe memory bandwidth regulation budget assignment. When regulation is required, all co-runners receive the same bandwidth cap. The regulator chooses the highest configured cap that the feasibility test accepts for the measured co-runner traffic and protected setpoints.

We implement *HeterCoRe* on a Xilinx Zynq UltraScale+

Multiprocessor System-on-Chip (MPSoC) platform by building on MemCoRe [4] and extending it to support heterogeneous processing-system (PS) and programmable-logic (PL) memory subsystems and bank-set-level bandwidth attribution. Our evaluation shows that the proposed shared-resource model closely matches hardware behavior, with a mean absolute percentage error (MAPE) of approximately 1.1%, and enables higher co-runner throughput than conservative partition-oblivious regulation while preserving protected-task guarantees.

In summary, this work makes the following contributions:

- We propose a gray-box shared-resource model for reasoning about bandwidth contention in heterogeneous memory subsystems, capturing both spatial memory partitioning and shared bottlenecks.
- We introduce a scenario-based feasibility test and regulation strategy that determine whether requested bandwidth setpoints can be guaranteed and throttle co-runners only when necessary.
- We implement and evaluate *HeterCoRe* on a heterogeneous PS-PL Zynq UltraScale+ platform, demonstrating accurate bandwidth modeling and improved regulated system utilization compared to conservative regulation.

II. Background

A. PS-PL platform

A PS-PL platform is a highly heterogeneous system whose main chip is the combination of an MPSoC and a Field-Programmable Gate Array (FPGA). On the PS side, it typically features a cluster of processing elements (PEs), such as high-performance cores that cater to the increasing computational demands of modern applications. Besides their private caches, these performance cores have a shared last-level cache, and finally connect to the PS-side DRAM through a Cache Coherent Interconnect (CCI). On the PL-side, in addition to the FPGA, a variety of hard Intellectual Property (IP) blocks might also be present to facilitate the processing of specific tasks or serve as memory media such as Block RAM (BRAM) and PL-side DDR. High-speed interfaces exist to support PS to PL and PL to PS communication. Interrupts and cross-triggers might also exist. In Arm-based platforms, these ports usually employ the AXI protocol. Among various features, what pertains most to this work is the cache coherence between the PS and PL.

B. PS-PL Cache Coherence

To provide a consistent view of data to all the PEs, including those instantiated in the PL, a PS-PL cache coherence protocol is required. When a number of PEs are kept coherent, they are said to be in the same coherency domain. AXI Coherency Extension (ACE) is the protocol used for PS-PL coherency. The implementation of *HeterCoRe* relies on the ability to configure the PL-side to be in the same coherency domain as the PS-side. Under a snoop-based cache coherence protocol like the ones usually implemented in PS-PL platforms, when a memory transaction originated from a PS-side core results in an LLC miss, causing the subsequent refill request, the cache-coherent interconnect will broadcast a snoop request to all the participants in the coherency domain. Each participant must reply to indicate whether they have the most up-to-date cache line content and, if so, provide the data. Importantly, even if a participant does not, the snoops still carry metadata that identify the physical addresses of the cache lines being snooped. Thus, a custom design on the PL side can comply with the coherency protocol while using the snoop metadata to perform memory traffic profiling [11].

C. Memory Bandwidth Regulation with MemCoRe

MemCoRe [4] is a snoop-based PL-side memory bandwidth regulator that uses snoop metadata to perform regulation. It is comprised of three modules: an accounting module, a decision module, and an enacting module. The accounting module estimates the number of PEs' memory activities by counting cache refills from the observed snoops. The snoops do not include the information on which core caused the memory activity. To handle this, a kernel module accompanies MemCoRe to map the tasks on a specific core to a designated physical memory range. This way, the physical address included in the snoop can be used to identify the originating core. The decision module uses the current and past estimates of the accounting module to decide whether a halt or resume action needs to be taken. Typically, a user selects memory bandwidth budgets for each core. At runtime, the decision module uses a token bucket algorithm [4] to halt or resume a core, so that the memory bandwidth consumed by that core never exceeds its budget. The enacting module uses the CoreSight debug interface [12] to halt/resume the core by instructing the core to enter/leave the debug state.

D. DDR4 Organization and Memory Controller

a) Organization: A modern DDR4 memory subsystem is structured hierarchically to improve parallelism. The system is organized into channels, ranks, bank groups, and individual banks. Each channel has its own dedicated data buses, thus operating in parallel. Within a channel, memory is divided into ranks, which are blocks of DRAM chips that share the same buses but are selected independently via chip-select signals. DDR4 introduces the concept of Bank Groups to alleviate internal timing bottlenecks. Each rank contains multiple bank groups, and each bank group contains individual banks. A bank features its own 2D storage array consisting of rows and columns, alongside a dedicated row buffer that acts as a temporary cache for the currently selected (opened) row. If the access pattern causes excessive row buffer misses (row miss), pending requests incur additional latency.

b) Memory Controller: The memory controller acts as the gatekeeper to this physical hierarchy. It translates system-level physical addresses into the DRAM-specific coordinates: Row, Bank Group, Bank, and Column. The address mapping policy determines how physical address bits are assigned to row, bank group, bank, and column fields. This choice affects both locality and parallelism: some mappings favor row-buffer hits for sequential accesses, while others spread accesses

across banks to increase bank-level parallelism. The controller also buffers requests in internal queues and may reorder them to improve throughput. As a result, even tasks assigned to different banks can still interfere through shared queues, arbitration logic, command bandwidth, and data bus multiplexing. Thus, bank partitioning reduces direct bank conflicts, but does not eliminate memory-subsystem contention.

III. Related work

Memory bandwidth regulation is an especially well-studied approach that limits the bandwidth each core or task can consume to mitigate memory contention. To date, both software- and hardware-based mechanisms have been explored. Software-based approaches typically utilize PMCs and have attracted considerable attention due to their immediate applicability on commercial off-the-shelf (COTS) platforms. A canonical example is MemGuard [1], [2], which has been followed by a variety of alternative designs—all utilizing PMCs for monitoring but deploying different regulation and enforcement mechanisms [3], [13]–[18]. Parallel to these software methods, hardware-based memory bandwidth regulation mechanisms have been proposed in both academia (e.g., BRU [5]) and industry (e.g., Intel RDT [19], ARM MPAM [20], and RISC-V CBQRI [21]). However, a fundamental limitation of these approaches is that they treat the main memory as a monolithic entity, ignoring the banked and potentially heterogeneous organization of modern DRAM systems. Indeed, these works primarily provide mechanisms for observing and enforcing a selected limit, whereas this work explores bottleneck-aware policies to extend performance guarantees beyond what is possible with static limits.

Garcia-Esteban et al.'s QIQoS [22] configures hardware QoS controls on the same Zynq UltraScale+ family and is a complementary actuation mechanism. The Maximum-Contention Control Unit (MCCU) [23] proposes access-count and contention-time enforcement in new silicon, whereas *HeterCoRe* is deployable on a COTS platform. Resource-capacity analysis [24] uses access-count bounds for WCET analysis, whereas *HeterCoRe* uses measured scenario bandwidths for feasibility testing and budget selection.

Sullivan et al. [25] proposed a per-bank bandwidth regulation approach—implemented in an open-source RISC-V SoC using FireSim—which keeps banks spatially shared among the cores and enforces a per-bank bandwidth budget for each core to achieve strong isolation while maintaining high bank-level parallelism and throughput. *HeterCoRe* instead strictly partitions banks so that each actor can use its bank set without inter-actor bank conflicts, then regulates residual upstream contention. The approaches are orthogonal; Sullivan et al.'s controller changes are unavailable in the ZCU102's hardened controllers, so a same-platform quantitative comparison is not possible.

DRAM bank partitioning is another prominent isolation technique. By allocating dedicated, non-overlapping DRAM banks to each core or task, this approach completely eliminates the primary source of memory contention: direct bank-level conflicts (such as row-buffer conflicts). Software-level bank partitioning is typically achieved via page-coloring frameworks [6]–[10] that exploit the OS's ability to map virtual addresses to physical ones. However, bank partitioning alone does not provide perfect performance isolation.

In this work, we advocate for a holistic approach that leverages *both* bank partitioning and bandwidth regulation simultaneously. By combining these paradigms, our approach strives to achieve stronger isolation guarantees while maximizing total system throughput.

IV. Shared Resource Model

This section introduces the shared resource model used by *HeterCoRe* to reason about bandwidth guarantees and regulation. The goal is not to precisely model every internal mechanism of a memory subsystem, controller, or interconnect. Instead, we model the system at the level needed for regulation: actors generate memory requests, those requests traverse a hierarchy of resources, and different parts of this hierarchy may become bottlenecks depending on where the requests are directed.

A. System Abstraction

We consider a set of actors

$$A = \{a_0, a_1, \ldots, a_{n-1}\}, \tag{1}$$

where each actor may represent a core, a task, or another request-generating agent. Each actor generates requests to a memory subsystem. A heterogeneous platform may contain interconnects, arbiters, memory controllers, bank groups, and DRAM banks. Figure 1 shows an example. The four CPU nodes at the top are the actors in this general illustration.

Traditional memory bandwidth regulators [2], [3], [13] typically treat the entire memory subsystem as one entity and ignore differences among its constituent resources. To address this, we model the memory subsystem as a tree of resources as shown in Figure 2. Leaves correspond to spatial memory resources, such as banks, bank sets, or memories attached to distinct controllers. Internal nodes correspond to shared structures, such as memory controllers, interconnects, or arbiters. This abstraction captures a key property of modern memory systems: partitioning at one level does not necessarily eliminate interference at higher levels. For example, assigning tasks to distinct DRAM banks removes direct bank conflicts, but the corresponding requests may still contend in shared controller queues, command scheduling logic, data buses, or the system interconnect.

We assume a gray-box view of the platform. In particular, we do not require detailed knowledge of every internal arbitration policy. Instead, we require two pieces of information: (i) a coarse topology indicating which memory resources are shared or independent, and (ii) a way to determine the destination resource of a memory request, such as the target memory controller or bank set. This is sufficient for bottleneck-aware regulation: if two actors access distinct resources that do not share a bottleneck, they need not be regulated as if they fully

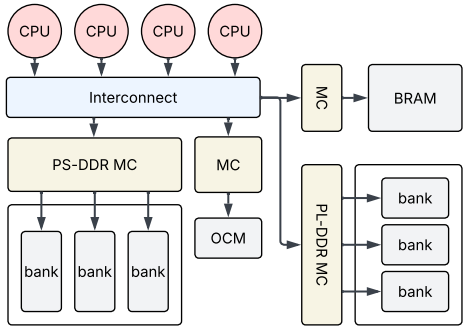

Fig. 1: A typical heterogeneous platform with multiple memory media and corresponding memory controllers.

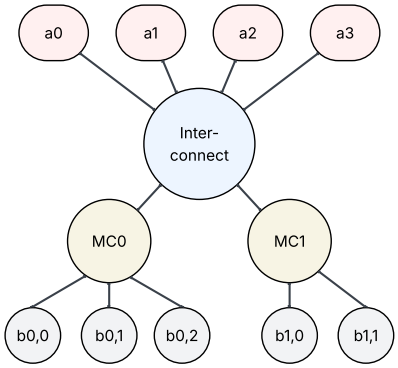

Fig. 2: Abstract resource-tree model. The top four nodes are actors; MC0 and MC1 are distinct memory controllers whose leaves represent their banks.

interfere; if they share a controller or another upstream bottleneck, the regulator must account for the resulting contention.

This motivates the model developed below. We first present the general heterogeneous case, where different memory subsystems may provide different bandwidth to their actors under the same concurrency level. We then show that the homogeneous formulation used later in the paper can be derived as a special case of the same model.

B. Heterogeneous Memory Subsystems

In a heterogeneous memory subsystem, actors may not receive equal bandwidth even when they execute concurrently. For example, if two actors access PS-side DDR and two actors access PL-side DDR, the former may observe a different per-actor bandwidth than the latter. Therefore, the achievable bandwidth depends not only on the number of active actors, but also on the memory subsystem targeted by each actor. We associate each actor a_i with a memory subsystem type

$$\tau_i \in \mathcal{T}, \tag{2}$$

where, for example, $\mathcal{T} = \{P, L\}$ denotes PS-side DDR and PL-side DDR. We assume that each actor accesses one memory subsystem during the interval being analyzed. Thus, the actor-to-subsystem mapping is known before applying the feasibility test. In our implementation, each actor is a task pinned to one PS core, and its memory is preallocated in one bank set of its assigned subsystem. Because this mapping is static, all deployed scenarios are known at configuration time.

For an active set $B \subseteq A$, its contention scenario $\Gamma(B)$ is the multiset of the actors' subsystem labels. For example, the scenario PPL denotes three active actors: two accessing PS-side DDR and one accessing PL-side DDR. For each scenario, we empirically measure the peak achievable per-actor bandwidth for every subsystem type present in the scenario. We denote this value as R_X^Γ, where Γ is the active scenario and $X \in \mathcal{T}$ is the subsystem type of the actor whose bandwidth is being measured.

For instance, R_P^{PPL} is the per-actor bandwidth of an actor accessing PS-side DDR when two actors access PS-side DDR and one accesses PL-side DDR. Similarly, R_L^{PPL} is the bandwidth of the actor accessing PL-side DDR in the same scenario. Actors mapped to the same subsystem type are assumed symmetric under the calibration workload. We take each R_X^Γ as the minimum of 100 steady-state measurements with the strongest IsolBench stressor, excluding startup and turn-off. A weaker stressor could yield an unsafe higher value. The test uses the matching read/write class when task behavior is known and the lower applicable rate otherwise.

For a system with up to two actors accessing PS-side DDR and two accessing PL-side DDR, the required constants are

$$\begin{aligned} &R_P^P, \quad R_L^L, \\ &R_P^{PP}, \quad R_L^{LL}, \quad R_P^{PL}, \quad R_L^{PL}, \\ &R_P^{PPL}, \quad R_L^{PPL}, \quad R_P^{PLL}, \quad R_L^{PLL}, \\ &R_P^{PPLL}, \quad R_L^{PPLL}. \end{aligned} \tag{3}$$

These constants capture heterogeneous bandwidth sharing. Unlike a homogeneous model, which assigns one bandwidth value to each concurrency level, this model assigns one bandwidth value per subsystem type and per active scenario.

Calibration produces the measured values R_X^Γ above. Before loading the feasibility-test table, we set $R_{X,\mathrm{cert}}^\Gamma$ to the lowest measured rate for type X over Γ and every reachable scenario with fewer actors where type X remains active. Thus, an actor completing earlier than predicted cannot expose the test to a lower calibrated rate. This one-time configuration step does not change the table size or the runtime lookup. The general test uses $R_{X,\mathrm{cert}}^\Gamma$, while Equation (16) and its validation retain the measured R values.

C. Feasibility Test for Heterogeneous Systems

The test considers one regulation interval: the period over which actors must receive their requested amounts of data. We express elapsed time as a fraction of this interval.

Assumption 1 (Conservative service)**.** Each $R_{X,\mathrm{cert}}^\Gamma$ is positive. While scenario Γ is active, a backlogged actor of type X receives service at a rate no lower than the measured R_X^Γ. Each actor's requested amount of data is available for service at the start of the interval, and enforcement prevents it from transferring more than that amount. The observation and enforcement requirements are stated in Section V-C.

We now define a feasibility test for a requested set of bandwidth setpoints. Let

$$S = \{s_0, s_1, \ldots, s_{n-1}\} \tag{4}$$

be the requested setpoints for n actors in A. The test determines whether all actors can receive their requested bandwidth within one normalized regulation interval. Thus, $b_i = s_i$. We initialize each actor's remaining bandwidth demand as $b_i \leftarrow s_i$, and initialize the accumulated execution fraction as $F \leftarrow 0$. Let A' be the set of actors with positive remaining demand. At each iteration, we construct the current scenario $\Gamma(A')$ from the subsystem labels of actors in A'. For every active actor $a_i \in A'$, the configured bandwidth used by the test is

$$r_i = R^{\Gamma(A')}_{\tau_i,\mathrm{cert}}. \tag{5}$$

We then compute the fraction of the regulation interval required for each actor to complete its remaining demand under the current scenario $f_i = b_i/r_i$. The first actor to complete its demand is determined by $f = \min_{a_i \in A'} f_i$. If $F + f > 1$ then the test rejects the vector. Otherwise, we update $F \leftarrow F + f$ and subtract the work completed during this fraction:

$$b_i \leftarrow b_i - f r_i, \quad \forall a_i \in A'. \tag{6}$$

All actors whose remaining demand reaches zero are removed from A', and the test repeats with the reduced active set. This corresponds to a new, less contended scenario. If all actors are removed before F exceeds one, the setpoint vector is feasible. Intuitively, the test simulates service over a normalized interval. The first iteration considers the most contended scenario, because all actors are active. Once an actor receives its requested bandwidth, it is removed, and the model transitions to the next scenario with fewer active actors. This process continues until either all setpoints are satisfied or the interval is exhausted. Rejection means only that the conservative model does not certify the vector; it does not prove physical infeasibility.

Theorem 1 (Soundness)**.** *Under Assumption 1, every setpoint vector accepted by the feasibility test and enforced on the actors is met on the real system within one regulation interval.*

Proof. The test simulates service in phases, one per iteration. We show by induction that, at the start of every phase, each actor's remaining demand on the real system is no greater than the remaining demand tracked by the test. This is true initially. During a phase, the real scenario can only be the scenario simulated by the test (fully backlogged interfering actors) or one in which at least some actor completes their allotted demand and becomes inactive earlier. By construction, $R^{\Gamma}_{X,\mathrm{cert}}$ is no greater than the measured rate in any such scenario. Assumption 1 therefore ensures that every unfinished actor receives at least the service credited by the test, and the claim follows for the next phase. Each phase satisfies at least one actor's remaining demand in the test, so there are at most n phases. If the test accepts with $F \leq 1$, all real demands also complete within the interval. □

Example: Heterogeneous Feasibility Test

Consider four actors: two have data in PS-side DDR and two in PL-side DDR. For these illustrative values, the conservative configuration step leaves the entries unchanged, so we omit the "cert" subscript in the table:

Γ	P	L	PP	LL	PL	PPL	PLL	$PPLL$
R^{Γ}_P	2600	–	2600	–	2550	2500	2550	2500
R^{Γ}_L	–	1450	–	850	1450	1400	850	850

Suppose the requested setpoints are

$$(s_0, s_1, s_2, s_3) = (2200, 2300, 700, 800) \text{ MB/s},$$

where a_0, a_1 access the PS-side subsystem and a_2, a_3 access the PL-side subsystem. The initial scenario is therefore $PPLL$, with rates $R^{PPLL}_P = 2500,\ R^{PPLL}_L = 850$.
The first iteration computes

$$\frac{2200}{2500} = 0.88,\ \frac{2300}{2500} = 0.92,\ \frac{700}{850} = 0.824,\ \frac{800}{850} = 0.941.$$

The minimum fraction is $f = 0.824$, so a_2 completes first. The remaining demand becomes approximately $(141.2,\ 241.2,\ 0,\ 100)$. The next active scenario is PPL, with

$$R^{PPL}_P = 2500, \qquad R^{PPL}_L = 1400.$$

The next minimum fraction is

$$f = \min\left(\frac{141.2}{2500}, \frac{241.2}{2500}, \frac{100}{1400}\right) = 0.056,$$

after which a_0 completes. Repeating the same procedure for the remaining scenarios PL and P gives a total accumulated fraction $F \approx 0.919 < 1$. Thus, all requested setpoints can be served within one normalized regulation interval, and the setpoint vector is feasible.

Complexity. A homogeneous system with n actors needs exactly n constants. With t subsystem types, the number of possible constants to measure is upper-bounded by $n t\mathrm{CR}(t, n)$, where $\mathrm{CR}(t, n)$ computes the number of combinations with replacement. Since t is small and fixed in practice, this reduces to a complexity of $O(n^{t+1})$. However, if the actor-to-subsystem mapping is fixed, only the constants for the deployed scenarios need to be measured. In the evaluated 2 PS + 2 PL only eight scenarios are possible and only 12 constants are needed, as listed in Equation (3). The test has at most n phases and $O(n)$ work per phase, and thus $O(n^2)$ time complexity.

D. Homogeneous Systems as a Special Case

The homogeneous model is a special case of the heterogeneous formulation. In a homogeneous subsystem, all active actors are equivalent under the calibration workload and receive the same per-actor bandwidth at a given concurrency level. This corresponds to a single subsystem type, denoted H, with scenarios

$$H, HH, HHH, \ldots, H^n. \tag{7}$$

The heterogeneous constants reduce to a simpler sequence

$$R_1, R_2, \ldots, R_n, \tag{8}$$

where

$$R_k \equiv R_H^{H^k}. \tag{9}$$

Thus, R_1 is the peak achievable bandwidth of one actor running alone, R_2 is the per-actor bandwidth when two identical bandwidth-intensive actors execute concurrently, and so on. Under symmetric contention, the total achievable bandwidth B_k at concurrency level k is modeled as

$$B_k = kR_k. \tag{10}$$

In an ideal linearly scaling system, we would expect $B_k \approx kR_1$. In practice, R_k often decreases as k increases, because the actors stress shared bottlenecks such as controller queues, command scheduling logic, or interconnect resources. The sequence $R_1, \ldots, R_n$ therefore provides an empirical characterization of concurrency-dependent maximum achievable bandwidth behavior for the homogeneous case.

E. Estimating Bandwidth in the Homogeneous Case

For homogeneous systems, the symmetry of the constants allows a simpler closed-form bandwidth estimate. Consider n actors and suppose we want to estimate the bandwidth available to actor a_{n-1} given the observed bandwidths, or assigned setpoints, of actors $a_0, \ldots, a_{n-2}$:

$$b_0, b_1, \ldots, b_{n-2}. \tag{11}$$

Sort these known bandwidth values in nondecreasing order:

$$b_{(0)} \leq b_{(1)} \leq \cdots \leq b_{(n-2)}, \tag{12}$$

and define

$$b_{(-1)} = 0. \tag{13}$$

We decompose the sorted values into increments:

$$\Delta_i = b_{(i)} - b_{(i-1)}, \quad i \in \{0, 1, \ldots, n-2\}. \tag{14}$$

The increment Δ_0 is consumed by all $n-1$ known actors. The next increment, Δ_1, is consumed by all known actors except the one with the smallest bandwidth. More generally, increment Δ_i is consumed by $n-1-i$ known actors. When estimating the bandwidth available to a_{n-1}, we include a_{n-1} as a potential co-runner, so increment Δ_i corresponds to a contention level of $n-i$ actors.

We define

$$x_i = \frac{\Delta_i}{R_{n-i}} = \frac{b_{(i)} - b_{(i-1)}}{R_{n-i}}, \quad i \in \{0, 1, \ldots, n-2\}. \tag{15}$$

The value x_i is the fraction of the normalized interval during which the system behaves as if it were operating at contention level $n-i$. The estimated bandwidth of actor a_{n-1} is then

$$\hat{b}_{n-1} = \left(1 - \sum_{i=0}^{n-2} x_i\right) R_1 + \sum_{i=0}^{n-2} x_i R_{n-i}. \tag{16}$$

The first term corresponds to the remaining fraction of the interval in which a_{n-1} effectively runs without contention from the known actors, and therefore receives bandwidth R_1. The second term accounts for the fractions in which a_{n-1} overlaps with different numbers of active actors and receives the corresponding concurrency-dependent bandwidth R_{n-i}.

If $\sum_{i=0}^{n-2} x_i > 1$, the known actors require more than one modeled interval, so Equation (16) gives no certified estimate. Actor a_{n-1} is intentionally absent from the sum: it has no setpoint here because the equation estimates its remaining bandwidth. When every actor has a setpoint, the general feasibility test applies instead. Section VII validates this estimator.

V. Regulation Strategy

The feasibility tests in Section IV can be used to guide bandwidth regulation. At a high level, the regulator has two objectives. First, it must ensure that protected actors receive their required bandwidth guarantees. Second, subject to those guarantees, it should avoid unnecessarily restricting co-running actors. This is achieved by using the feasibility test as an admission and runtime decision mechanism: a bandwidth assignment is accepted only if the model indicates that all protected guarantees can be satisfied under the corresponding contention scenario.

A. Model-Guided Regulation

Consider a set of actors A and a corresponding bandwidth setpoint vector S. The setpoints may represent required guarantees for protected actors or bandwidth budgets assigned to best-effort co-runners. A protected setpoint is a minimum entitlement given sufficient demand, whereas a co-runner budget is an enforced upper bound. A setpoint may exceed the initial fully contended rate because it is averaged over the interval, but it cannot exceed the actor's measured solo bandwidth. Before enforcing a finite vector, *HeterCoRe* applies the feasibility test from Section IV. If the vector is rejected, the allocation must be changed; rejection alone does not prove physical infeasibility.

This mechanism allows *HeterCoRe* to avoid the pessimism of partition-oblivious regulation. A conservative regulator may assume that all actors stress the same bottleneck and therefore impose strict bandwidth limits on every co-runner. In contrast, *HeterCoRe* uses the actor-to-resource mapping and calibrated bandwidth constants to determine whether the current contention pattern actually threatens a guarantee. If the protected actor's setpoint remains feasible, co-runners can continue using the available bandwidth. If the setpoint becomes infeasible, the regulator reduces the bandwidth assigned to one or more co-runners until the feasibility test passes.

The same logic applies both before execution and at runtime. Offline, the feasibility test can be used as an admission-control step for proposed bandwidth assignments. Online, it can be applied as the observed bandwidth consumption is measured and the contention conditions are computed to decide whether additional throttling is required. Thus, regulation is driven by model feasibility rather than by a fixed worst-case assumption that all actors interfere equally.

B. Selecting Co-Runner Budgets

When the feasibility test rejects a requested setpoint vector, *HeterCoRe* must reduce the bandwidth demand placed on the shared resource. This can be done by lowering the budgets of selected co-runners, prioritizing a subset of actors, or assigning explicit guarantees to protected actors while treating the remaining actors as best-effort. The evaluated policy assigns all co-runners an equal candidate budget and tests configured candidates from highest to lowest.

Once the feasibility test passes, the regulator has identified a set of budgets that is sufficient to preserve the protected actor's guarantee under the calibrated model, provided that co-runner demand during the regulated interval does not exceed the demand tested at that decision. Importantly, this process does not require all co-runners to be throttled to a fixed conservative budget. Instead, co-runner bandwidth is restricted only to the extent required to make the setpoint vector feasible. This allows *HeterCoRe* to preserve protected-task guarantees while giving every co-runner the highest configured cap accepted for the current measured demand.

Let T be MemCoRe's token period, bw_i actor a_i's measured bandwidth, and $budget_i$ its currently enforced rate (initially ∞). Let $\mathcal{B} = \{c_1 > \cdots > c_m = 0\}$ be the configured co-runner cap levels. The one-period demand formed at a runtime decision is $\widehat{d}_i = s_i T$ for a protected actor and $\widehat{d}_i = \min\{bw_i, budget_i\}T$ for a co-runner. For a candidate cap $c \in \mathcal{B}$, define

$$d_i(c) = \begin{cases} \widehat{d}_i, & a_i \text{ is protected,} \\ \min\{\widehat{d}_i, cT\}, & a_i \text{ is a co-runner.} \end{cases} \tag{17}$$

The regulator tests candidates from highest to lowest and selects the first accepted value. Protected setpoints are admitted offline, so the $c = 0$ case provides a valid fallback.

Theorem 2 (Regulated-Interval Safety)**.** *Suppose the descending search selects a finite $c^\star$. Under Assumption 1, in any complete regulated interval in which each co-runner's demand does not exceed $d_i(c^\star)$, every backlogged protected actor receives its setpoint. Moreover,*

$$c^\star = \max\{c \in \mathcal{B} \mid d(c) \textit{ is accepted}\},$$

so it is the largest configured equal cap accepted for the current measured demand vector.

Proof. The test accepts $d(c^\star)$. During finite regulation, protected actors are capped at their setpoints and co-runners at $c^\star$. Under the stated demand condition, the real demand vector is no greater than the accepted vector, so Theorem 1 supplies every protected setpoint. The protected caps also prevent one protected actor from exceeding its tested demand and starving another. Since candidates are examined in descending order, the first accepted value is the largest one in $\mathcal{B}$. □

This claim concerns equal caps in $\mathcal{B}$ and makes no claim about unequal allocations or values between configured levels. It applies only to regulation periods with finite budget assignments; when, due to a feasible measurement, the budgets are set to unlimited, later demand changes are handled reactively by the next decision.

Example. Using illustrative certification values, consider two protected actors whose data reside in PS-side DDR, each with a 600 MB/s setpoint, and two co-runners whose data reside in PL-side DDR and whose measured bandwidths are at least 900 MB/s. Let $R^{PPLL}_{P,\mathrm{cert}} = 2500$ MB/s, $R^{PPLL}_{L,\mathrm{cert}} = R^{LL}_{L,\mathrm{cert}} = 850$ MB/s, and $\mathcal{B} = \{900, 800, 0\}$ MB/s. For $c = 900$, the protected actors finish after $600/2500 = 0.24$ of the interval. Each co-runner then has $900 - 0.24(850)$ remaining, so $F = 0.24 + (900 - 0.24(850))/850 \approx 1.059$ and the vector is rejected. For $c = 800$, the same calculation gives $F \approx 0.941$, so the regulator enforces $(600, 600, 800, 800)$ MB/s.

C. Enforcement

The shared resource model determines whether a set of bandwidth setpoints is feasible, but it does not require a specific enforcement mechanism. Any mechanism can be used as long as it satisfies two requirements.

First, the system must be able to observe each actor's bandwidth consumption at runtime with sufficient granularity. The regulator must estimate how much bandwidth each actor consumes over a bounded accounting window and compare this value against the actor's assigned budget. In *HeterCoRe*, this observation must also preserve the actor-to-resource attribution required by the model, such as the memory subsystem and bank set targeted by each access.

Second, the system must provide a mechanism for reducing an actor's bandwidth consumption when it exceeds its assigned budget. This mechanism may take different forms, such as delaying memory requests, limiting request injection, changing scheduling priority, or temporarily suspending the actor. In our implementation, enforcement is performed through temporal suspension of the core.

These requirements separate the regulation strategy from the low-level implementation. The model identifies feasible bandwidth assignments and determines when throttling is necessary, while the enforcement mechanism ensures that actors remain within the accepted budgets at runtime. Therefore, the same strategy can be applied with different monitoring and throttling mechanisms. Here, *HeterCoRe* selects the limits and MemCoRe [4] observes and enforces them. With static actor-to-resource mapping, a PMC polling-based mechanism such as *MemPol* [13] could be used if it satisfies these requirements.

Algorithm 1[1] connects the offline and runtime uses of the test. Offline, proposed rates are converted to one-period demands and checked by FEASIBLE. At runtime, W is the configurable sliding measurement window (approximately 300 μs), independent of the token period T.

For each actor a, tx_a denotes the transactions attributed by MemCoRe during the latest window W. Here s_a, bw_a, $budget_a$, and c are rates in MB/s, whereas d_a is data demand

[1]The algorithm runs in FPGA logic using standard fixed-point integer arithmetic; divisions are implemented as multiplications by reciprocals precomputed at configuration. One online decision takes 10–20 cycles at 250+ MHz, negligible against the 15–80 μs enforcement periods.

Algorithm 1: Per-period budget selection.

```
function FEASIBLE(demands d, set active)
    E ← 0                                          ▷ elapsed simulated time
    while active ≠ ∅ and E ≤ T do
        t ← min_{a∈active} d_a / R_a[active]       ▷ first to finish
        d_a ← d_a − t R_a[active] for every a ∈ active
        move every actor with d_a = 0 out of active
        E ← E + t
    end
    return E ≤ T
end
function LARGESTBUDGET(demands d)
    d' ← d                                         ▷ copy demand vector
    foreach c ∈ B, highest to lowest do
        foreach co-runner a do
            d'_a ← min{d_a, cT}                    ▷ Equation (17)
        end
        if FEASIBLE(d', A) then return c
    end
end
procedure SELECTBUDGETS(window W, token period T)
    foreach actor a do
        tx_a ← MemCoRe transactions attributed to a in W
        bw_a ← bandwidth derived from tx_a and W
        d_a ← s_a T if a is protected; otherwise
          d_a ← min{bw_a, budget_a} T
          ▷ initial one-period demand
    end
    if FEASIBLE(d, A) then
        set every budget_a ← ∞                     ▷ no throttling
    else
        c ← LARGESTBUDGET(d)
        budget_a ← s_a for protected actors
        budget_a ← c for co-runners
    end
    forward the budget vector to MemCoRe for enforcement
end
```

over T. FEASIBLE operates on local copies of d and $active$, so the test does not alter its caller's inputs. In the listing, $R_a[active]$ is the configured conservative rate for actor a's subsystem under the current active set, E is elapsed simulated time, and t is the duration of one simulated phase.

MemCoRe's unmodified token buckets then halt an actor on exhaustion until the next-period replenishment. Protected actors are capped at their setpoints only during active regulation; Theorem 2 applies to those complete intervals with finite budgets. The unthrottled branch is reactive: later demand changes are assessed at the next decision.

VI. Instantiation and Implementation Details

A. Instantiation on Target Platform

We instantiate the proposed bottleneck-aware regulation approach on a heterogeneous PS-PL platform with two DRAM subsystems: PS-side DDR and PL-side DDR. Each subsystem has its own memory controller and a set of spatial memory resources. We will refer to these resources as bank sets.

Each actor is a task pinned to a designated PS core, and its memory accesses are assigned to a specific bank set. We distinguish between protected tasks, whose bandwidth guarantees must be preserved, and co-running memory-intensive tasks, which act as interference sources. The goal of regulation is to preserve the protected task's bandwidth setpoint while allowing co-runners to use as much bandwidth as possible.

On the PS-side DDR, each bank set contains two distinct DRAM banks. Bank sets assigned to different actors do not overlap. On the PL-side DDR, each bank set contains one distinct DRAM bank. This difference is caused by platform-specific address mapping and controllability constraints, which are discussed in the implementation section.

This instantiation allows us to evaluate three representative configurations. First, all actors may access bank sets in the PS-side DDR. Second, all actors may access bank sets in the PL-side DDR. Third, the protected task's data may reside in PS-side DDR while the co-runners' data reside in PL-side DDR. These configurations stress different parts of the heterogeneous memory subsystem and allow us to study how bandwidth scales under spatial separation.

The FPGA-side regulator is synthesized above 250 MHz in our experiments. At this frequency, coherence-side observation and response do not measurably reduce maximum memory bandwidth. At lower FPGA frequencies, however, we observed that the monitoring path itself can become a bottleneck, with the act of observing and responding to coherence traffic degrading memory throughput. All reported results therefore use a regulator frequency high enough to avoid this artifact.

B. Key Implementation Details

We implement our system on a Xilinx Zynq UltraScale+ ZCU102 platform. The implementation uses both available DRAM subsystems: the PS-side DDR [2] and a PL-side DDR [3].

1) Memory Mapping and Bank Sets: For the PS-side DDR, the default row-bank-column configuration uses bits 14 and 15 as DRAM bank bits, while bit 6 selects the bank group.[4] Since bit 6 is too low to be reliably constrained by our allocation mechanism, we do not explicitly control it. As a result, each PS-side spatial partition corresponds to a bank set containing two banks with fixed bits 14 and 15 and unconstrained bank-group bit.

For the PL-side DDR, the memory controller is configured using a bank-row-column mapping. Under this configuration, physical address bits 26, 27, and 28 select the DRAM bank. These bits can be controlled by our allocator, so each PL-side spatial partition corresponds to a single DRAM bank.

Thus, the implementation exposes two partitioning granularities: four two-bank partitions on the PS side and eight single-bank partitions on the PL side.

2) Task Placement: We use the *user-pool* (upool) interface from a customized version of the open-source MemScope framework [26] to allocate task memory from physical regions

[2]Micron MTA4ATF51264HZ-2G6E1

[3]Micron MT40A256M16LY-062E

[4]We reverse-engineered the mapping using DRAMA++ [25] and validated it against the memory controller documentation and control register values.

TABLE I: Measured peak achievable per-actor bandwidth on homogeneous PS/PL setups.

	1 CORE		2 CORES		3 CORES		4 CORES	
	read (MB/s)	write (MB/s)	read (MB/s)	write (MB/s)	read (MB/s)	write (MB/s)	read (MB/s)	write (MB/s)
PS-DRAM	2586	1348	2570	1136	2353	1101	1803	944
PL-DRAM	1345	446	857	304	539	207	403	167

TABLE II: Measured peak achievable per-actor bandwidth (MB/s) on heterogeneous PS+PL setups.

Setup	Γ	Read R_P^Γ	Read R_L^Γ	Write R_P^Γ	Write R_L^Γ
1 PS-DRAM + 1 PL-DRAM	PL	2550	1444	1250	473
1 PS-DRAM + 2 PL-DRAM	PLL	2557	614	834	195
1 PS-DRAM + 3 PL-DRAM	$PLLL$	1400	539	332	211
2 PS-DRAM + 1 PL-DRAM	PPL	2529	1435	1092	468
2 PS-DRAM + 2 PL-DRAM	$PPLL$	2488	861	890	306
3 PS-DRAM + 1 PL-DRAM	$PPPL$	1880	1412	636	425

matching selected address-bit patterns. Each actor is implemented as a task pinned to a core, with its memory allocated from a selected bank set. Protected tasks and interference tasks are assigned to non-overlapping bank sets when spatial separation is required.

On the PS side, we allocate pages with constrained bits 14 and 15. Constraining the bank-group bit (bit 6) is not possible as the finest page granularity is 4 KB on the considered platform. On the PL side, the relevant bank-selection bits are higher-order bits, allowing direct allocation to individual banks.

3) Bandwidth Monitoring and Regulation: Bandwidth monitoring is implemented by modifying MemCoRe's coherence-based accounting path. MemCoRe observes coherence snoop traffic from the PL side; each snoop carries the physical address of the requested cache line. We extend this mechanism to decode both the target memory subsystem and the corresponding bank set. No performance monitoring counters are used.

For every observed physical address, the regulator first determines whether the access targets PS-side or PL-side DDR. It then applies the corresponding bank-set decoding rule: bits 14 and 15 for the PS-side DDR, and bits 26–28 for the PL-side DDR. The regulator maintains bandwidth counters per actor and per spatial memory resource.

HeterCoRe adds the bank-aware decision layer summarized in Figure 3. Both *HeterCoRe* and MemCoRe have small, comparable FPGA footprints.

C. Runtime Operation

The feasibility test is practical for runtime use in our FPGA-based regulator, where one online decision takes 10–20 cycles at 250+ MHz, depending on the number of candidate co-runner budgets evaluated. This cost is negligible relative to the regulation periods, and Algorithm 1 gives the corresponding offline and online logic. In software, the same test can be used offline to validate candidate setpoints, but repeated runtime evaluation would add overhead and cause system slowdown.

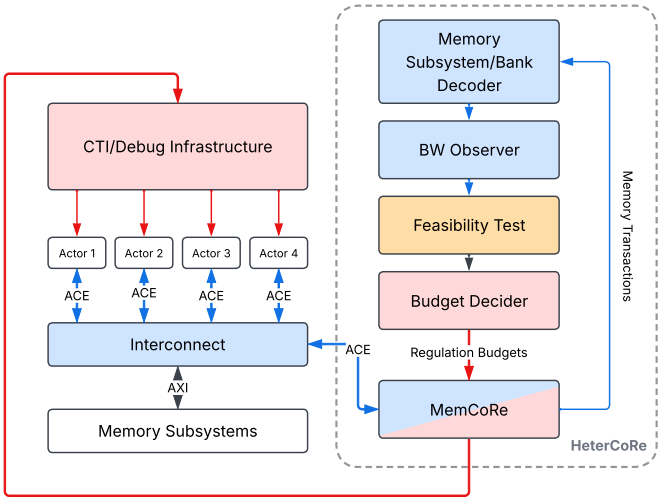

Fig. 3: Interaction between *HeterCoRe* and MemCoRe. MemCoRe provides coherence monitoring, token budgeting, and halt/resume enforcement through the debug interface; *HeterCoRe* adds memory-subsystem and bank-set decoding, per-resource accounting, scenario characterization, the feasibility test, and resource-aware budget selection. CTI: Cross-Trigger Interface.

TABLE III: FPGA resource utilization of the *HeterCoRe* regulator on the ZCU102 via Vivado post-implementation report.

Resource	Used	Utilization
LUTs	12,318	4.49%
Flip-Flops	13,606	2.48%
BRAM	25.5	2.80%
DSP	3	0.12%

We use a 15 μs period for PS-only homogeneous experiments, reducible to 1 μs as in MemCoRe, and an 80 μs period for PL and mixed PS+PL experiments due to longer halt-entry latency caused by draining PL-side DRAM queues.

VII. Evaluation

This section presents the evaluation of the implementation of *HeterCoRe* on ZCU102. The first two experiments serve as the validation of the formula proposed in Section IV, and the third experiment compares the regulation performance under different configurations.

A. Maximum Achievable Per-actor Bandwidth

The foundation of the bandwidth prediction model is the maximum achievable per-actor bandwidth R_i (Equation (8)). We first present the observed values on the target as summarized in Table I for homogeneous setups and Table II for

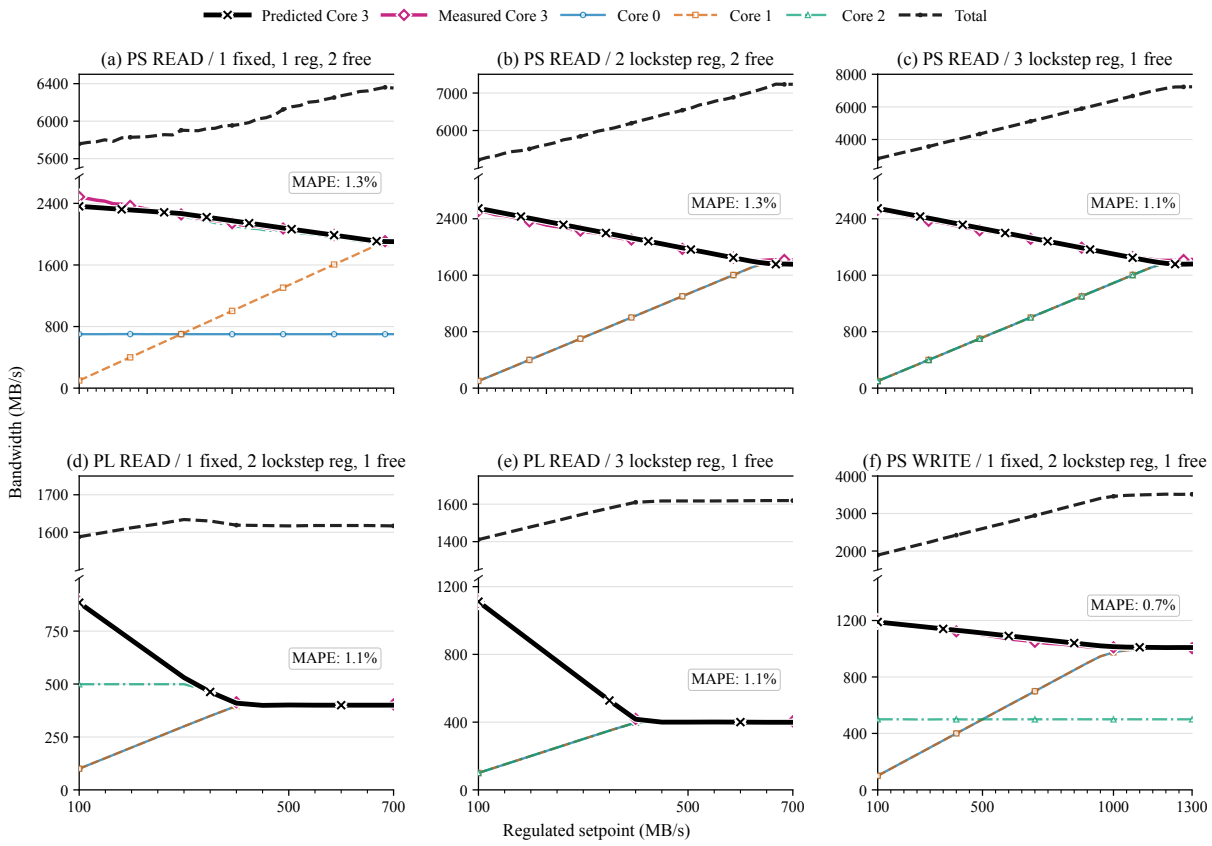

Fig. 4: The predicted bandwidth of Core 3 is compared with the observed bandwidth. All setups show satisfactory accuracy of the prediction model in Equation (16). Saturating IsolBench traffic tests the predicted ceiling; lighter traffic would trivially remain below it.

heterogeneous setups. The *bandwidth* benchmark from the RT-Bench IsolBench suite [27], [28] is used as the running task that can stress the memory through continuous read or write operations. The rows identify the memory subsystem stressed. The columns indicate the number of cores running *bandwidth*. The *bandwidth* instances on different cores will use different static bank bits. For example, the PS-DRAM 2 CORES entry indicates that there are two *bandwidth* instances running on two cores, respectively, using the PS-side DRAM. The bank bits of the two tasks are constant but different. Thus, these two tasks are partitioned at the bank level. The data show that, despite bank partitioning, the measured maximum achievable bandwidth of a *bandwidth* instance decreases as the number of co-running instances increases, suggesting bottlenecks in the upstream memory hierarchy. For practical regulator use, we slightly round down the tabulated bandwidths (*e.g.*, 2550 to 2500 MB/s and 473 to 450 MB/s) to account for a small margin of variability in the real system.

B. Bandwidth Prediction Curve Validation

From the previous set of experiments, we obtained the empirical maximum achievable per-actor bandwidths, namely $R_1, ..., R_4$ for PS-side and PL-side DDR respectively. This section presents the experiments that compare the observed bandwidth and that predicted by Equation (16). The results are shown in Figure 4. In the top-left plot, four *bandwidth* instances run on four cores, respectively. The task on Core 0 is configured to extract a constant bandwidth. The task on Core 1 is configured to extract different (increasing) bandwidth in each run. The remaining two tasks are configured to extract bandwidth at their maximum capacity. Then we compare the predicted bandwidth of Core 3 to its observed value. In the top-middle plot, the setup is similar to the former, except that Core 0 is configured to extract the same amount of bandwidth as Core 1. In the top-right plot, Core 0, Core 1, and Core 2 are configured to extract the same amount of bandwidth. All experiments show satisfactory prediction accuracy. Panels (d)–(e) report PL-side reads, while panel (f) reports PS-side writes; the per-panel MAPE values are shown in the figure.

C. Slowdown/Throughput Experiments

The following experiments use the PS-side DRAM. The benchmarks *disparity*, *mser*, *sift*, *stitch*, *tracking*, *localization*, *texture_synthesis* are chosen from RT-Bench San Diego Vision Benchmark Suite (SD-VBS) [28], [29]. The *bandwidth* benchmark from the RT-Bench IsolBench suite [27] is used as a memory stressor that will run on different core(s) (a.k.a., the co-runner). The co-runner stresses the memory by generating continuous sequential write transactions. The results are shown in Figure 5. We run each benchmark under various conditions:

(i) S-BS stands for solo bank set: the benchmark runs alone in one bank set (two banks in PS-side DDR or one in

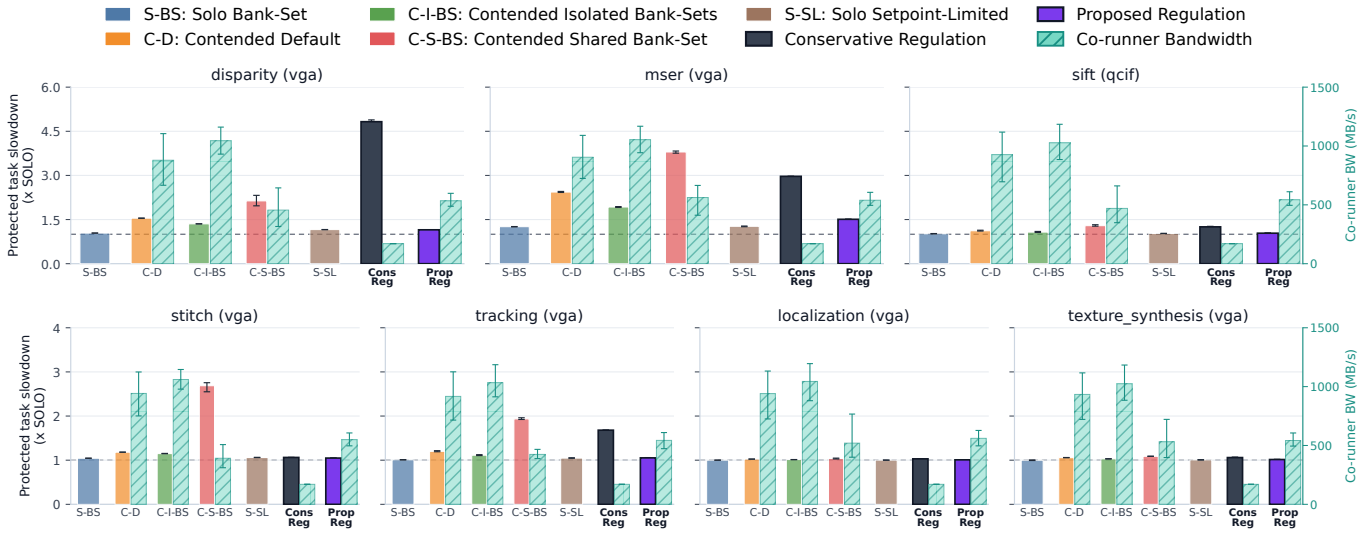

Fig. 5: Various setups are tested on the PS-DDR. The y-axis shows the slowdown of the protected task and average bandwidth of the co-runners. The x-axis labels represent different experiment setups. The slowdown is compared with the execution time of the benchmark in isolation without any regulation. The error bar is the max/min of the runs. The normalization baseline is a separate unrestricted solo run; S-BS is bank-set-constrained and therefore need not equal one.

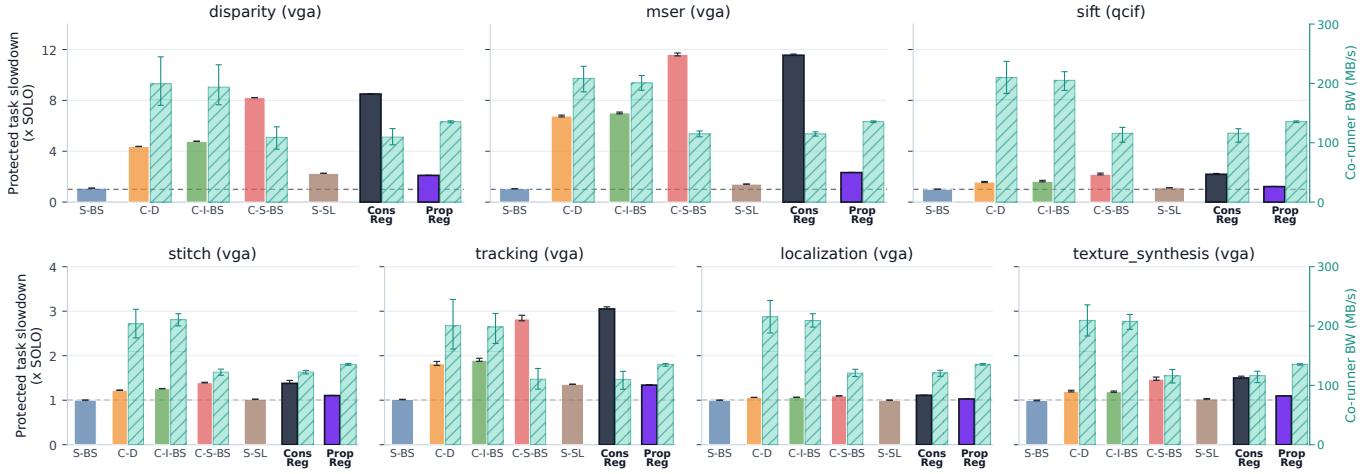

Fig. 6: Same setup but the target DRAM is the PL-side DDR. Normalization is as in Figure 5; S-BS remains bank-set-constrained and therefore need not equal one.

PL-side DDR);

(ii) C-D stands for contention default, in which the benchmark runs concurrently with three *bandwidth* instances running on three different cores. The system operates without partitioning—no bank partitioning, no regulation;

(iii) C-I-BS stands for contention with isolated bank sets: the benchmark's data and each co-runner's data reside in a distinct bank set, without regulation. It therefore isolates bank placement;

(iv) C-S-BS stands for contention with a shared bank set: the benchmark's data and all three co-runners' data reside in the same bank set;

(v) S-SL stands for solo setpoint limited, in which the benchmark is the sole task on the platform with exclusive bank set. It is further regulated at the exact setpoint (900 MB/s in this setup);

(vi) CONS. REG. uses MemCoRe's unmodified token-bucket regulation with bank-unaware worst-case equal caps: 345 MB/s for every actor in PS-side experiments, from $64/t_{RC} \approx 1.38$ GB/s with $t_{RC} = 46$ ns, and 250 MB/s in PL-side experiments, from the measured $\approx$ 1 GB/s all-on-one-bank bandwidth.[5]

(vii) PROP. REG. combines isolated bank-set placement with *HeterCoRe*'s scenario-aware regulation. It protects the benchmark at 900 MB/s, a demanding but feasible target

[5]The plots report read bandwidth only. Since each stressor write also induces a read/refill, the plotted value is approximately half of the total regulation budget.

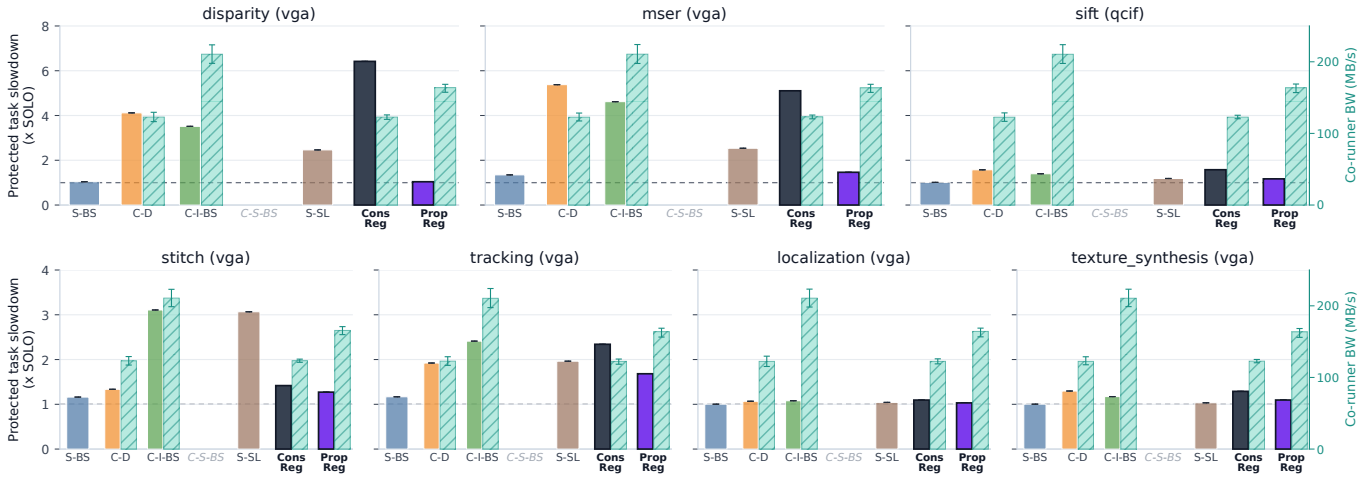

Fig. 7: Same setup but the protected task's data reside in PS-side DDR while the co-runners' data reside in PL-side DDR. Normalization is as in Figure 5; S-BS remains bank-set-constrained and therefore need not equal one.

chosen 20% below its average unrestricted solo bandwidth, leaving capacity for co-runners.

Let us focus on *disparity* which suffers from contention effect considerably. C-D shows the slowdown (1.54x) that *disparity* typically experiences when the memory subsystem is heavily contended. We can infer that in this setup some bank-level contention occurs because of the comparison against C-I-BS. In the latter setup, by simply assigning each task to an exclusive bank-set, not only does the benchmark experience a smaller slowdown (1.34x), but also the co-runners can achieve higher bandwidth (1013 MB/s vs 890 MB/s without bank partition). Another comparison group, CONS. REG. and PROP. REG., further illustrates the contention at bank level. The former represents a worst-case scenario in which all tasks access the same bank. This can occur under bank-unaware allocation, memory pressure or fragmentation, or adversarially when a co-runner intentionally targets the same bank to maximize interference. Contention worsens and the sustainable bandwidth is significantly lower (345 MB/s). The benchmark experiences a 4.63x slowdown and the co-runners run at 173 MB/s read traffic (around 345 MB/s total BW). The latter demonstrates two points: (1) by reducing bank-level contention via bank partitioning, the regulation can provide much more bandwidth to the benchmark; (2) more subtly, to guarantee just a little more bandwidth to the benchmark as compared to its natural allocation in C-I-BS, the bandwidth of the co-runners is severely throttled (524 MB/s vs 1013 MB/s). The little improvement to the benchmark (0.26 = 1.34 - 1.08 slowdown reduction) is at the expense of a large drop in co-runner bandwidth, because the actors still compete at shared upstream bottlenecks after bank partitioning.

Next, we conduct the same set of experiments but on the PL-side DRAM. The results are shown in Figure 6. One interesting observation is the comparison between C-S-BS and CONS. REG.: because the benchmark's data and the co-runners' data reside in the same bank, both suffer the highest slowdowns.

Finally, we run experiments in which the benchmark's data reside in PS-side DDR while the co-runners' data reside in PL-side DDR. The result is shown in Figure 7. The C-S-BS configuration is inapplicable because their data cannot share a bank across the two controllers. Comparing CONS. REG. and PROP. REG. illustrates again that once the bottleneck is identified and regulated accordingly, substantial performance gain can be extracted. Indeed, the consistent trend that emerges from all the experiments is that the slowdown of the benchmark under analysis is reduced while the bandwidth attained by the co-runners generally increases. Across the board, the performance improvements unlocked by *HeterCoRe* vary depending on how memory-intensive each benchmark is. Indeed, the largest gains are achieved in the case of *disparity*, *mser*, and *tracking*, which are known to be the most memory latency-sensitive benchmarks in the considered suite [13].

Notably, the proposed regulation strategy is remarkably effective at protecting the temporal characteristics of the benchmark under analysis in the case of heterogeneous memory controllers being utilized (Figure 7).

VIII. CONCLUSION

This paper introduces *HeterCoRe*, a bottleneck-aware memory bandwidth regulation approach for heterogeneous memory subsystems that combines spatial task separation with a general theoretical shared-resource model. Evaluation on a Xilinx Zynq UltraScale+ platform demonstrates that the proposed theoretical model achieves high bandwidth prediction accuracy with a mean absolute percentage error of around 1.1%. A bottleneck-aware adaptive regulation strategy is also proposed. The combination of bank partitioning and the proposed adaptive regulation also improves shared resource utilization and increases total system bandwidth.

ACKNOWLEDGMENT

We thank Golsana Ghaemi, Ph.D. and Prof. Kazem Taram for their contribution in the early stages of this project. We used generative AI tools for data visualization, plot formatting, and general textual/graphic refinements throughout the paper. This research was supported by the National Science Foundation (NSF) under grants numbers CSR-2238476, SHF-2403012, and CCF-240301.

REFERENCES

[1] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013.

[2] ——, "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Trans. Computers*, vol. 65, no. 2, pp. 562–576, 2016. [Online]. Available: https://doi.org/10.1109/TC.2015.2425889

[3] A. Zuepke, A. Bastoni, W. Chen, M. Caccamo, and R. Mancuso, "MemPol: Policing core memory bandwidth from outside of the cores," in *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2023, San Antonio, TX, USA, May 9-12, 2023*. IEEE, 2023, pp. 235–248. [Online]. Available: https://doi.org/10.1109/RTAS58335.2023.00026

[4] I. Izhbirdeev, D. Hoornaert, W. Chen, A. Zuepke, Y. Hammad, M. Caccamo, and R. Mancuso, "Coherence-aided memory bandwidth regulation," in *2024 IEEE Real-Time Systems Symposium (RTSS)*, 2024, pp. 322–335.

[5] F. Farshchi, Q. Huang, and H. Yun, "BRU: bandwidth regulation unit for real-time multicore processors," in *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2020, Sydney, Australia, April 21-24, 2020*. IEEE, 2020, pp. 364–375. [Online]. Available: https://doi.org/10.1109/RTAS48715.2020.00011

[6] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 155–166.

[7] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 367–375.

[8] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *2013 IEEE 16th International Conference on Computational Science and Engineering (CSE)*, Dec. 2013, pp. 685–692.

[9] X. Pan and F. Mueller, "NUMA-aware memory coloring for multicore real-time systems," *Journal of Systems Architecture*, vol. 118, p. 102188, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121001351

[10] H. Yang, R. Shao, Y. Cheng, Y. Chen, R. Zhou, G. Liu, G. Xie, and Q. Zhou, "REDB: Real-time enhancement of Docker containers via memory bank partitioning in multicore systems," *J. Syst. Archit.*, vol. 151, Jun. 2024, art. no. 103135. [Online]. Available: https://doi.org/10.1016/j.sysarc.2024.103135

[11] S. Roozkhosh, D. Hoornaert, and R. Mancuso, "CAESAR: coherence-aided elective and seamless alternative routing via on-chip FPGA," in *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*. IEEE, 2022, pp. 356–369. [Online]. Available: https://doi.org/10.1109/RTSS55097.2022.00038

[12] ARM, "ARM CoreSight architectural specification v3.0," 2022. [Online]. Available: https://developer.arm.com/documentation/ihi0029/latest/

[13] A. Zuepke, A. Bastoni, W. Chen, M. Caccamo, and R. Mancuso, "MemPol: polling-based microsecond-scale per-core memory bandwidth regulation," *Real-Time Syst.*, vol. 60, no. 3, pp. 369–412, Jun. 2024. [Online]. Available: https://doi.org/10.1007/s11241-024-09422-8

[14] N. Dagieu, A. Spyridakis, and D. Raho, "Memguard: A memory bandwidth management in mixed criticality virtualized systems—Memguard KVM scheduling," in *10th Int. Conf. on Mobile Ubiquitous Comput., Syst., Services and Technologies (UBICOMM)*, 2016, pp. 21–27. [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=ubicomm_2016_1_40_10072

[15] W. Ali and H. Yun, "Protecting Real-Time GPU Kernels on Integrated CPU-GPU SoC Platforms," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Altmeyer, Ed., vol. 106. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, pp. 19:1–19:22. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2018.19

[16] M. Xu, L. T. X. Phan, H.-Y. Choi, Y. Lin, H. Li, C. Lu, and I. Lee, "Holistic resource allocation for multicore real-time systems," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 345–356.

[17] A. Saeed, D. Dasari, D. Ziegenbein, V. Rajasekaran, F. Rehm, M. Pressler, A. Hamann, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Memory utilization-based dynamic bandwidth regulation for temporal isolation in multi-cores," in *28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2022, Milano, Italy, May 4-6, 2022*. IEEE, 2022, pp. 133–145. [Online]. Available: https://doi.org/10.1109/RTAS54340.2022.00019

[18] A. Zuepke, A. Pradhan, D. Ottaviano, A. Bastoni, and M. Caccamo, "ETM2: Empowering traditional memory bandwidth regulation using ETM," arXiv:2603.16490, Mar. 2026.

[19] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, "A closer look at Intel resource director technology (RDT)," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 127–139. [Online]. Available: https://doi.org/10.1145/3534879.3534882

[20] ARM, "Arm Architecture Reference Manual Supplement. Memory System Resource Partitioning and Monitoring (MPAM) for Armv8-A," Accessed: 2024-01-01, 2022. [Online]. Available: https://developer.arm.com/docs/ddi0598/db/

[21] RISC-V International, "RISC-V capacity and bandwidth controller QoS register interface (CBQRI)," https://github.com/riscv-non-isa/riscv-cbqri, 2026, accessed: 2026-05-22.

[22] S. Garcia-Esteban, A. Serrano-Cases, J. Abella, E. Mezzetti, and F. J. Cazorla, "Quasi isolation QoS setups to control MPSoC contention in integrated software architectures," in *35th Euromicro Conference on Real-Time Systems, ECRTS 2023, July 11-14, 2023, Vienna, Austria*, ser. LIPIcs, A. V. Papadopoulos, Ed., vol. 262. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 5:1–5:25. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2023.5

[23] J. Cardona, C. Hernández, J. Abella, and F. J. Cazorla, "Maximum-contention control unit (MCCU): resource access count and contention time enforcement," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, J. Teich and F. Fummi, Eds. IEEE, 2019, pp. 710–715. [Online]. Available: https://doi.org/10.23919/DATE.2019.8715155

[24] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement," in *26th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE Computer Society, 2014, pp. 109–118. [Online]. Available: https://doi.org/10.1109/ECRTS.2014.20

[25] C. Sullivan, A. Mamandipoor, C. R. Strickler, and H. Yun, "Per-Bank Memory Bandwidth Regulation for Predictable and Performant Real-Time Systems," in *32nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, May 2026, pp. 167–179. [Online]. Available: https://doi.org/10.1109/RTAS68450.2026.00025

[26] G. Ghaemi, G. Franco, K. Taram, and R. Mancuso, "MemScope: Open-source kernel-level framework for heterogeneous memory characterization," in *2025 IEEE Real-Time Systems Symposium (RTSS)*, 2025, pp. 500–513.

[27] P. K. Valsan, H. Yun, and F. Farshchi, "Taming non-blocking caches to improve isolation in multicore real-time systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*. IEEE Computer Society, 2016, pp. 161–172. [Online]. Available: https://doi.org/10.1109/RTAS.2016.7461361

[28] M. Nicolella, S. Roozkhosh, D. Hoornaert, A. Bastoni, and R. Mancuso, "RT-Bench: an extensible benchmark framework for the analysis and management of real-time applications," in *RTNS 2022: The 30th International Conference on Real-Time Networks and Systems, Paris, France, June 7 - 8, 2022*, Y. Abdeddaïm, L. Cucu-Grosjean, G. Nelissen, and L. Pautet, Eds. ACM, 2022, pp. 184–195. [Online]. Available: https://doi.org/10.1145/3534879.3534888

[29] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. M. Louie, S. Garcia, S. J. Belongie, and M. B. Taylor, "SD-VBS: the San Diego vision benchmark suite," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009, October 4-6, 2009, Austin, TX, USA*. IEEE Computer Society, 2009, pp. 55–64. [Online]. Available: https://doi.org/10.1109/IISWC.2009.5306794