PCCS: Processor-Centric Contention Slowdown Model for Heterogeneous System-on-chips

***Abstract*—Many performance models have been proposed to characterize the memory interference of workloads co-running on heterogeneous Systems on Chip (SoC). But they are mostly for post silicon time. How to effectively consider memory interference in the SoC design stage yet remains an open problem. This paper presents a new approach to the problem, consisting of a novel *processor-centric performance slowdown modeling* methodology and a new *three region interference-conscious performance model*. The modeling process needs no measurements of co-runs of various combinations of applications, but the produced performance models can be used to estimate the co-run performance of arbitrary workloads on various SoC designs that embed newer generation of accelerators, such as deep learning accelerators (DLA), in addition to CPUs and GPUs. The new method reduces average prediction errors of the state-of-art model from 24.8% to 8.7% on GPU, and from 13.0% to 3.3% on CPU, and demonstrates much improved efficacy in guiding SoC designs.**

I. Introduction

As domain specialization is proven to be a promising path to achieve high performance at low energy [15], integrated shared memory heterogeneous architectures have coupled CPUs with other accelerators on the same die to serve the demanding needs of autonomous, mobile and edge computing. System on chips (SoC), such as NVIDIA's Jetson AGX Xavier [2], Qualcomm's SnapDragon [14], and Apple's A1X Bionic [34], embedded specialized processing units, such as vision processors (PVA), deep learning accelerators (DLA) and digital signal processors (DSP), under the same memory bus to efficiently run a variety of computations with distinct characteristics.

The diversity of processing units (PU) amplifies the complexity in SoC design, an issue studied in this work. Figure 1 illustrates the specific problem. An SoC design team needs to build an SoC to support the execution of some important workloads, such as an autonomous vehicle workload that consist of a set of related modules (e.g., object recognition, trajectory prediction, etc.). The team has access to a set of PUs

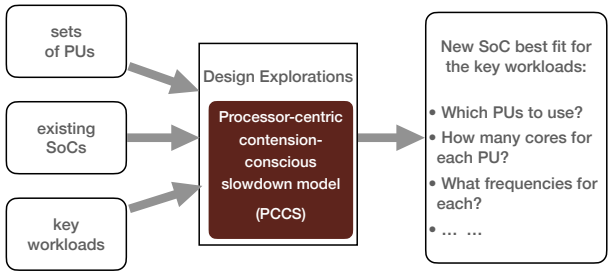

Fig. 1. The SoC design problem focused in this study.

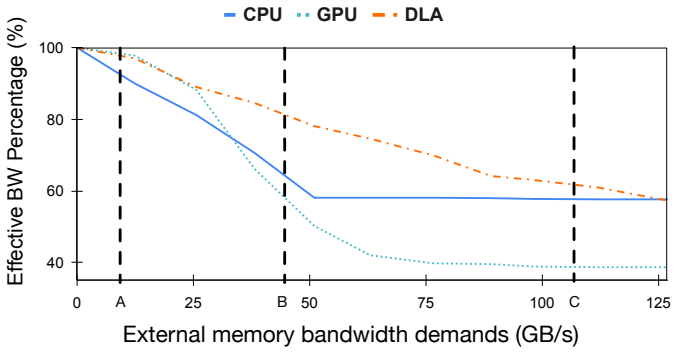

Fig. 2. The percentage of the requested memory bandwidth that is met for a processor under various degrees of external memory pressure. The requested memory bandwidths are 30GB/s, 93GB/s, and 127GB/s for DLA, CPU, and GPU respectively. The peak memory bandwidth of the SoC (NVIDIA Xavier AGX) is 137GB/s. A, B, C mark the points where requested BW + external BW = DRAM peak BW, for GPU, CPU, DLA respectively. The effects of contention is being observed even when the sum of requested bandwidth and the external memory pressure is less than the total available DRAM bandwidth capacity.

as well as some existing SoCs, which each are equipped with some of the PUs. The team needs to decide on the design of the new SoC that best fits the workload of interest. Specifically, they need to determine (1) what PUs should be put onto the SoC, (2) how many cores of each PU, and (3) what frequencies or other configurations each PU should use[1].

The team may be able to run every module (or kernel) of the workload on the PUs on existing SoCs to measure the performance of the module's standalone executions on different types of PUs. The challenge is on figuring out how the modules and the workload would perform if multiple apps are co-run on a new SoC. As PUs on a SoC typically share the memory and bus, the co-location would incur memory interference and hence performance degradations of various degrees as Figure 2 illustrates.

Many prior performance models have included memory interference into considerations [7], [8], [10], [12], [13], [18], [22], [24]–[26], [33], [36]–[38]. They are however for post-silicon runtime optimizations rather than uses in the SoC design stage. Among efforts in creating performance models to guide hardware designs [5], [16], [35], [39], the state of the art is Gables [16]. The work is valuable as the first attempt to integrate memory interference into Roofline models. Its proposed performance model is however remarkably rough,

[1]Memory subsystem design tends to stay stable across SoCs of a certain period of time; this work focuses on PUs, assuming the reuse of memory subsystems in existing SoCs and leaving memory design to future research.

with many oversimplifications. It, for instance, assumes available memory bandwidth is proportionally distributed among the heterogeneous PUs. Evidence has shown that large errors result from those simplifications. The model for example suggests zero slowdowns for co-running applications if the sum of their standalone-run memory bandwidth consumption is less than the total memory bandwidth of the SoC, which contradicts the empirical results shown in Figure 2. How to effectively handle memory interference in SoC design remains a problem yet to solve.

This paper proposes a new approach, named processor-centric contention-slowdown modeling (PCCS), to address this important problem. PCCS is based on a systematic analysis of the impact of co-run memory contention, and comprises a new methodology and a new performance model.

- *Analysis:* We conduct a series of experiments to observe the influence of co-run memory contention. The observations contradicts the proportional distribution of memory bandwidth assumption the prior SoC co-run model [16] builds on. Through an in-depth analysis, we validate that fairness control in memory controller is an important reason for the observed co-run performance—a factor neglected by prior SoC co-run modeling [16].
- *Methodology:* The new approach leverages a *source-obliviousness insight*, that is, the influence external memory interference has on the performance of an application is determined by the degree of interference, and is largely oblivious to what the sources of the external traffic are. Led by the insight, the new approach employs a processor-centric modeling scheme, which uses a set of *calibrators* (controllable memory traffic generators) to assist the empirical measurements in determining the model parameters. The modeling process needs no measurements of co-runs of various combinations of applications, but the produced performance models can be applied to arbitrary applications.
- *Slowdown model:* The new performance slowdown model is a *three region interference-conscious model*, which classifies an application, based on its algebraic computation intensity, into one of three categories, each of which features a distinctive class of slowdown models. The model is processor-centric, characterizing the architecture behavior in the presence of external memory bandwidth demands. Piecewise formulation is used in model formulation to ensure accuracy.

We evaluate the general applicability of our performance model and demonstrate that it is precise enough for guiding SoC design explorations. We verify our model via a set of Rodinia and deep-learning operations on the three diversely different PUs, i.e., 8-core ARM CPU, Volta GPU and deep learning accelerator (DLA), embedded in NVIDIA's Jetson AGX Xavier autonomous SoC [2], and also on the CPU and the GPU of Qualcomm's SnapDragon 855 mobile SoC [4]. The new method reduces average prediction errors from 24.8% of the state-of-the-art models to 8.7% on GPU, from 13.0% to 3.3% on CPU. The results, in use case studies, help avoid over provisioning PUs or their frequencies, saving up to 50% area (with reduced cores) or 51% power budget (with reduced frequencies) over the suggested configurations by prior models, while maintaining the same level of actual co-running workload performance. The results confirm the effectiveness of the proposed method and models in bridging the gap in the current support of heterogeneous SoC designs.

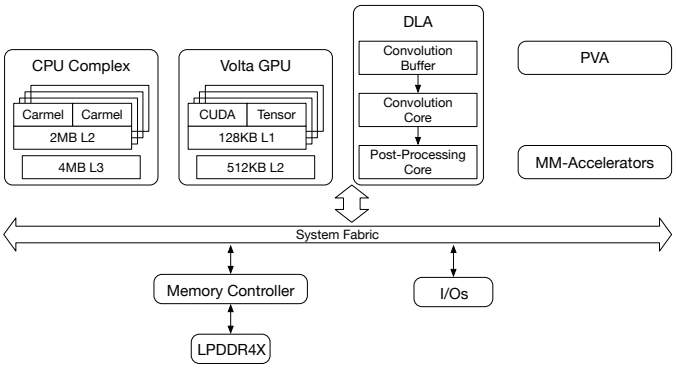

Fig. 3. The NVIDIA Jetson AGX Xavier Architecture

II. Memory Interference Characterization

The first step in building our proposed slowdown model is to understand and characterize the contention occurring when kernels with different memory access behaviors are co-located on different PUs.

A. Target Architecture

Our slowdown model targets the heterogeneous shared memory (HSM) SoC, which has multiple types of processors, a memory controller interface to shared DRAM memory, and high-bandwidth on-chip interconnect. An example is NVIDIA's Jetson AGX Xavier architecture [2], shown in Figure 3. It is a recent HSM-SoC targeting autonomous computing applications. Xavier integrates 8-core ARM v8.2 CPU, 512-core NVIDIA Volta with 64 TensorCores, an NVIDIA Deep Learning Accelerator (DLA), a Programmable Vision Accelerator (PVA) and multimedia accelerators on the same die, and they share the same system memory.

This work assumes the following about the target HSM-SoC: 1) Each PU operates concurrently and independently. 2) An execution of a kernel spans only one type of PU and a PU can run only one kernel at a given time. 3) The available memory bandwidth is shared among PUs.

B. Observations

Our work identifies three main factors that affect the memory access slowdown, i.e., *perceived throughput*, of a kernel K_i running on PU P_j :

- The maximum memory throughput that a specific PU P_j that the kernel K_i is running on can achieve.
- The memory throughput requested by the kernel K_i, when it was run standalone on that specific PU P_j.
- The external bandwidth demand issued by kernels running on other PUs ($\{P_0, P_1, ..., P_n\} - P_j$).

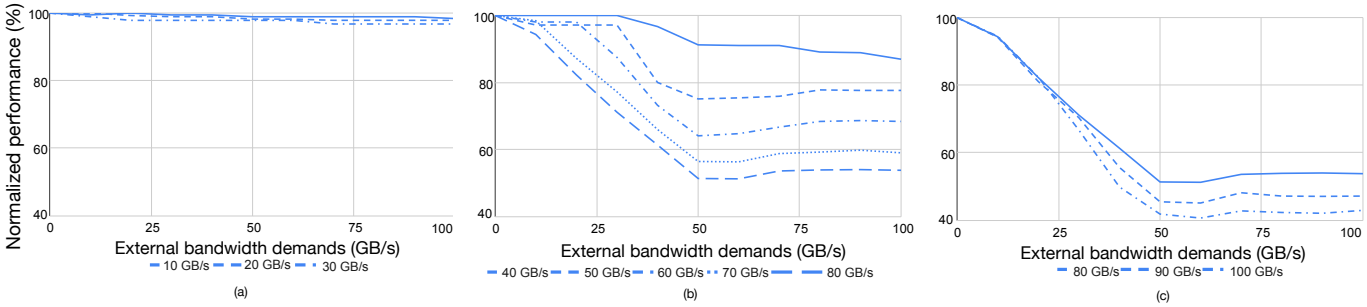

Fig. 4. The performance of synthetic programs over different memory pressures. The standalone requested memory BW varies from 10 GB/s to 30 GB/s in (a), from 40 GB/s to 80 GB/s in (b) and from 80 GB/s to 100 GB/s in (c).

To understand how an HSM-SoC behaves under these three factors, we run the ERT suite [1], which bundles several synthetic vector add and multiplication kernels with different memory bandwidth demands, on the GPU and CPU of NVIDIA's Xavier AGX system. We vary the requested standalone memory bandwidth of the kernels from 10GB/s to 100GB/s with a 10GB/s increase. For each experiment, we also create a synthetic external bandwidth demand that varies between 0 and 100GB/s and co-run it with the experiment.

The results are shown in Figure 4. The y-axis represents the percentage ratio of the perceived performance of the slowed-down kernel to its standalone performance. In all these experiments, the external pressure is created by a kernel running on another PU. The x-axis represents the external memory bandwidth demands, defined as the sum of the memory bandwidth demanded by all co-running kernels. (The bandwidth demand of a kernel is its memory bandwidth consumption when it runs alone.) The actual external bandwidth pressure is equal to or lower than the demands.

The workloads fall into three categories.

1) Figure 4 (a) shows the observations on the kernels that request only a small amount of memory bandwidth. Their perceived performance drops only slightly as external bandwidth demand increases.
2) Figure 4 (b) shows the kernels that demand a more significant amount of bandwidth (i.e., 40-80 GBps), and the memory performance curves exhibit a three-stage trend:
   a) The curves start with a relatively flat segment, showing the little influence of external pressure on the perceived throughput.
   b) The curves enter a fast near-linear dropping region when the external bandwidth demand increases beyond a certain level.
   c) The curves then flatten out as the external bandwidth demand exceeds a certain level.
3) Figure 4 (c) shows the observations of the kernels that request a large amount of bandwidth. Even when there is just a small amount of external bandwidth demand, the perceived throughput reduction is significant. But when the external bandwidth demand goes beyond a certain level, the curves flatten out.

The observations contradicts the proportional distribution of memory bandwidth assumption the prior SoC co-run model [16] builds on. For instance, according to that model, there should be no slowdowns at the beginning part of the curves in Figure 4 (b,c) as the total requested bandwidth has not yet reached the peak bandwidth of the system, and there should not be a flat part at the end of the curves in Figure 4 (b,c). We further conducts an in-depth analysis of the observed co-run performance trends to explain those observations, especially the flat part at the end of those curves. We report the analysis next.

C. Validation: Fairness Control in Memory Controllers

Our study shows that the observed co-run performance trends are the results of the prioritization of row-hit requests in memory controllers (MC) and the fairness control employed in MC. MCs typically prioritizes row-hit requests in the MC queues to maximize standalone bandwidth [30]. But when more PUs simultaneously access the memory, high row-buffer hits can no longer be maintained. Hence, even when the cumulative bandwidth is less than what the memory is capable of, a significant amount of memory access slowdown is observed. The reason for the flattened segments at the end of the trends is that the memory controller (MC) employs fairness control [12], [28], [29] to improve overall system throughput and avoid starving any processor on the HSM-SoCs.

As prior studies show [6], [19], [20], [27]–[29], adopting fairness-aware scheduling policies in memory controllers are essential for multi-core processors and HSM-SoCs. Without fairness control, the shared memory multi-core processors and HSM-SoCs will experience three major problems: 1) Low system throughput; 2) Vulnerability to denial-of-service; 3) Unpredictable performance and uncontrollable quality-of-services (QoS). The main reason is that memory-intensive programs would hog most bandwidth in uncontrolled memory interference.

To validate that fairness control is the reason for the observed performance trend, we conduct a series of measurements. Because commercial systems do not disclose detailed MC designs, we conduct the study via a cycle-accurate x86 CMP DRAM simulator, Ramulator [21], by using scheduling policies with and without fairness controls. The front end of the simulator is based on Pin [23]. We model the memory system in detail to faithfully capture bandwidth limitation

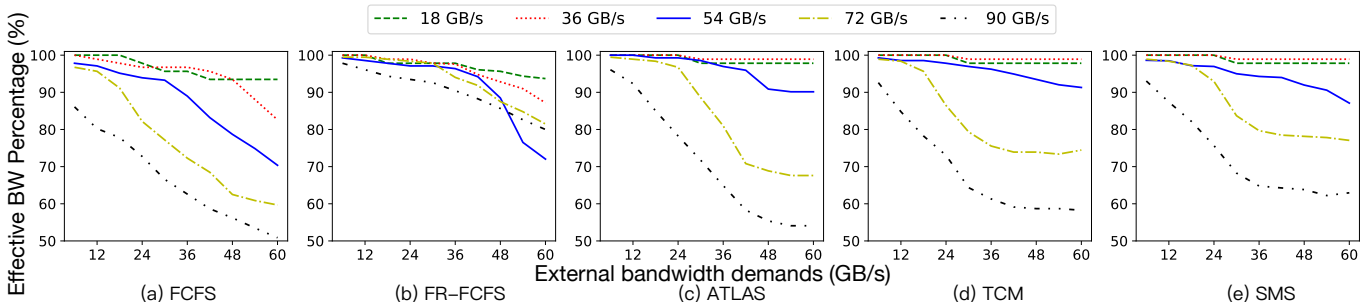

Fig. 5. The performance of synthetic programs running on high BW group over different memory pressures on different scheduling policies.

and contention, and enforce bank/port/channel/bus conflicts. Table I shows the major DRAM and processor parameters.

We construct several synthetic vector add and multiplication kernels built from the roofline toolkit [1] with different memory throughput demands. To simulate heterogeneous scenarios, we regard the 16 cores as two classes, with cores 0-7 as the low-bandwidth group, and cores 8-15 as the high-bandwidth group. On the low BW group, we vary the requested standalone memory BW of the kernels from 6GB/s to 60GB/s with a 6GB/s increase, and from 9GB/s to 90GB/s with a 9GB/s increase on the high BW group.

We evaluate five memory scheduling policies: first-come-first-serve (FCFS), first-read-FCFS (FR-FCFS) [30], Adaptive per-Thread Least-Attained-Service (ATLAS) [19], Thread Cluster Memory Scheduling (TCM) [20], and Stage Memory Scheduling (SMS) [6]. Only the last three policies adopt fairness control. The brief description of these policies are shown in Table II. The default parameters of these policies are used.

The results of these five policies are shown in Fig. 5 where the x-axis represents the external bandwidth demand and the y-axis is the percentage of the standalone performance retained in co-run. Table III reports the average row buffer hit rates (RBH) and effective BW percentage over theoretical peak BW when the sum of co-located programs' standalone BW is equal or larger than the theoretical peak BW of these 5 policies.

In the FCFS results, as shown in Fig. 5 (a), the effective BW percentage is reduced proportionally with external bandwidth demand. Since MC with FCFS deals with memory requests chronologically, each row buffer deals with requests without locality awareness, leading to low RBH and small effective BW (Table III) in co-location scenarios. FR-FCRS improves BW usage but does not adopt fairness control. In Fig. 5 (b), the programs suffer from large slow down when they are co-located with memory intensive programs, resulting in low system throughput.

On the other hand, the results on all three scheduling policies with fairness control exhibit trends similar to our observed trends on Xavier (Fig. 4). ATLAS, for instance, tries to maintain fairness according to attained service. In HSM-SoCs, the processor attained least services will be prioritized, leading similar attained services in different processors. As shown in Fig. 5 (c), when a program needs a small BW and it is co-located with a high BW demanding program, its actual attained BW remains nearly unchanged, as its bandwidth demand gets satisfied by the MC prioritization scheme. A medium BW demanding program however sees some drop of its attained BW at the beginning as its demands exceed what the prioritization scheme of MC offers. When the external BW demand grows, the slowdown becomes larger, as other co-located programs send more requests in their own time slots. When the external BW demand keeps growing, they eventually reach a stable state under the MC scheduling policy, hence the flat segments in the performance curves. For a high BW demanding program, it is similar except that they get into the second phase from the very beginning. Similar effects are shown in the curves of TCM and SMS as Figures 5 (d,e)

TABLE I
MEMORY CONTROLLER SIMULATION CONFIGURATION

Processor	16-core, each 2.2 GHz, 128 entries reorder buffer;
Cache	Private L1D cache, 8-way, 64KB, 4 cycles; Two core shared L2 cache, 8-way, 1MB, 9 cycles; Shared L3 cache, 16-way, 4MB, 26 cycles.
DRAM Controller	256-entry request buffer, XOR-based address-to-bank mapping
DRAM Chip Parameters	DDR4-3200 timing parameter [21] 8 banks, 4K-byte row buffer per bank Single rank, 4 channels, 64-bit wide channel 102.4 GB/s theoretical bandwidth

TABLE II
THE DESCRIPTION OF 5 SCHEDULING POLICIES

Policy	Description
FCFS	MC schedules memory requests chronologically.
FR-FCFS [30]	MC prioritizes row-hit requests.
ATLAS [19]	Prioritization order: 1) Over threshold request. 2) Requests from the thread that attained least service. 3) Row-hit requests. 4) Oldest requests.
TCM [20]	Prioritization order: 1) Non-memory-intensive programs 2) Periodically rank shuffle memory-intensive programs 3) Row-hit requests. 4) Oldest requests.
SMS [6]	Steps:1) Group requests to the same row into batches 2) Schedule batches with p probability shortest first and (1-p) probability round-robin

TABLE III
RBH AND EFFECTIVE BW PERCENTAGE OF 5 POLICES AND XAVIER

Policies	FCFS	FR-FCFS	ATLAS	TCM	SMS	Xavier
RBH (%)	47.7	91.6	74.2	79.6	84.7	-
Effective BW Percentage over Peak BW (%)	65.6	89.7	78.4	80.8	84.3	79.1

show, although there are some detailed differences due to the differences in the specifics of the schedulers; detailed discussions are omitted for the sake of space.

III. Memory Interference Slowdown Model

Based on the observations in Fig. 4, we propose a *three-region memory interference slowdown model* for a processor in an HSM-SoC. Our approach is statistical via regression-based curve fitting. An alternative is to construct the model analytically on the detailed memory controller designs. That approach requires detailed knowledge on the memory controller of each target device. As memory controllers in commercial systems are usually undisclosed, the approach is infeasible in general.

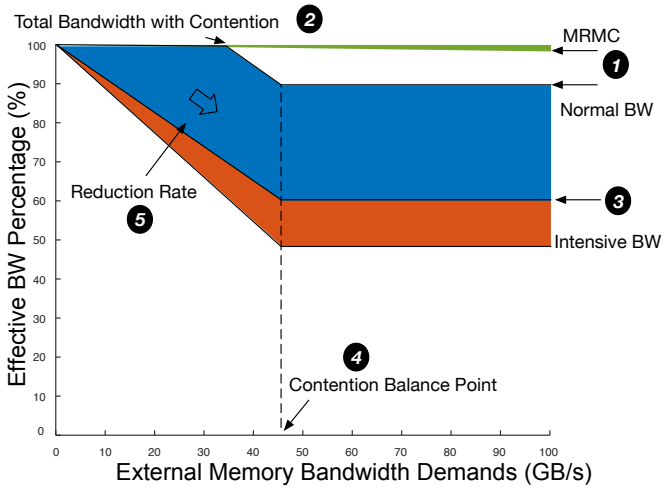

Fig. 6. The three-region interference classification model.

In the three-region model, we create a linear function for each of the three regions demonstrated in Fig. 4 (a), (b), and (c), and present them in the unified chart given in Fig. 6:

1) *Minor Contention Region* is the top-most region in the figure where the workload on the current PU requests memory bandwidth that is low enough so that the effects of the external memory bandwidth demand is minimal
2) *Normal Contention Region* is where the current PU's requested memory bandwidth is at a middle level such that the interference performance curve follows the pattern in Fig. 4 (b). The pattern has two points determining the beginning and end of its linear region and they are marked as *Total Bandwidth with Contention* point and *Contention Balance Point*, respectively.
3) *Intensive Contention Region* is the bottom-most region in the figure where the current PU's requested bandwidth is so high that it is significantly affected by the external bandwidth demand.

An important feature of this classification is that it is *processor-centric* since it characterizes each PU in the system separately rather than creating peer-to-peer contention analysis as many CPU+GPU based studies do. The different PUs on the same HSM-SoC would have different values for contention region boundary, contention balance point and reduction rate.

TABLE IV
Notations and Model Parameters

Term	Description
Normal BW	The amount of BW requested by the current processor that separates the minor and normal contention regions.
Intensive BW	The amount of BW requested by the current processor that separates the normal and intensive contention regions.
Maximum Reduction of Minor Contention (MRMC)	The BW reduction rate in the minor contention region at the largest external memory pressure
Contention Balance Point (CBP)	The value of external memory pressure demand where the perceived memory BW starts going flat.
Total Bandwidth with Contention (TBWC)	The sum of requested BW by a processor and the total external BW demand where the perceived memory BW of the current processor starts going down in the normal contention region
$rate^N$	The reduction rate in the perceived BW in the normal contention region
$rate^I$	The reduction rate in the perceived BW in the intensive contention region
Peak Bandwidth (PBW)	The peak bandwidth of the entire HSM-SoC
Achieved Performance (P)	The achieved performance (i.e. perceived or effective BW) under memory interference

For example, the GPU has a large number of threads and its standalone performance can hide the memory latency, hence using more bandwidth of the entire HSM-SoC. However, when the GPU is under contention from an external traffic caused by another processor in the HSM-SoC, the point of *total bandwidth with contention* would be larger. On the other hand, the reduction rate in both normal and intensive contention regions is larger. The normalized performance after the contention balance point is also smaller for GPUs. We next explain the exact formulations of the slowdown curves as well as how the model parameters can be determined for a PU.

A. Model Formulation

Table IV summarizes the parameters and concepts used in our model. We begin building our model by defining *region* partitions as shown in Equation 1. The x represents the bandwidth requested by the kernel running on the current PU. The normal bandwidth ($normalBW$) and intensive bandwidth ($intensiveBW$) are PU-specific values that can be obtained using the method explained in Section III-B.

$$Region = \begin{cases} Minor & 0 \leq x \leq normal\ BW \\ Normal & normal\ BW \leq x \leq intensive\ BW \\ Intensive & intensive\ BW \leq x \end{cases} \quad (1)$$

To model the contention in the minor contention region, we use the $MRMC$, the constant rate of reduction observed for this region. The achieved performance in the minor contention region, P^M, is as follows:

$$P^M = 100\% - \frac{MRMC * x}{PBW} \quad (2)$$

For the normal contention region, the achieved performance, P^N, is a piece-wise function, as shown in Equation 3; y represents the total demanded external memory throughput in this equation.

- The first piece of Equation 3 represents the case when the summation of requested BW and demanded external pressure ($x+y$) is smaller than the processor-specific value of $TBWC$ and y is smaller than CBP. In this case, the

achieved performance of the current processor is the same with the minor contention part.

- When $x+y$ is larger than $TBWC$ and the y is smaller than CBP, the achieved bandwidth of the current processor is reduced by a constant $rate^N$ multiplied by $x+y-TBWC$.
- When y is larger than CBP, the reduction of achieved bandwidth of the current processor remains constant, which is defined as $100\% - (x + CBP - TBWC) * rate^N$.

$$P^N = \begin{cases} 100\% - \frac{MRMC*x}{PBW} & 0 \leq x+y \leq TBWC \\ & and\ y \leq CBP \\ 100\% - (x+y-TBWC) * rate^N & TBWC \leq x+y \\ & and\ y \leq CBP \\ 100\% - (x+CBP-TBWC) * rate^N & CBP \leq y \leq PBW \end{cases} \tag{3}$$

For the intensive contention region, since the requested BW already exceeds the $TBWC$, the perceived memory BW reduction starts with minimal external pressure demand and the reduction rate R^I is larger. We obtain the rate $rate^I$, shown in Equation 4, by extending the performance reduction curve (dotted lines in Fig. 6) by multiplying with rate $rate^N$ in the normal region and then by dividing it by CBP.

$$rate^I = \frac{rate^N * (x + CBP - TBWC)}{CBP} \tag{4}$$

The perceived BW of an intensive region, P^I, has a reduction stage and a flat stage. The piece-wise equation is shown in Equation 5. The two pieces of this equation are identical to the second and third pieces of Equation 3 except that $rate^N$ is replaced with $rate^I$.

$$P^I = \begin{cases} 100\% - (x_i + y - TBWC) * rate^I & TBWC \leq y \leq CBP \\ 100\% - (x_i + CBP - TBWC) * rate^I & CBP \leq y \leq PBW \end{cases} \tag{5}$$

B. Model Construction

The slowdown model relies on several PU- and SoC-specific parameters that need to be determined via either standalone or collocated runs. To achieve varying amounts of memory bandwidth requests, we create synthetic kernels with different compute/memory (i.e., operational) intensities, similar to those used in the Roofline model [35]. They serve as calibrators of our models. For each variation of operational intensity, we record and report the resulting standalone requested memory pressure on the architecture it runs on.

We obtain the slowdown model parameters of a target processor in Fig. 6 by following these steps.

[1] To find the $normalBW$ and $MRMC$, ❶ in Fig. 6, we run synthetic kernels with an increasing amount of memory BW requests on the target PU and record their achieved performance while running an external synthetic kernel on a non-target PU to exert the maximum amount of memory pressure. As the requested memory BW of kernels gradually increases, the first kernel which suffers from a notable reduction on the achieved performance under the external memory pressure determines our boundary, $k^{boundary}$, for normal region, and the requested BW of this program is the *normal BW*. The throughput value of the kernel that comes right before entering the normal BW, belongs to the minor contention region and the achieved performance reduction of that kernel gives the $MRMC$ value.

[2] We run the boundary kernel $k^{boundary}$ found in the first step and co-locate it with contention generating kernels that generates increasing external memory pressure. The bandwidth of the first contention generating kernel that makes the boundary program $k^{boundary}$ to have a notable performance reduction gives the total bandwidth with contention, $TBWC$, as shown with ❷ in Fig. 6.

[3] We run a sequence of versions of the synthetic kernel with increasing memory requests where each run is put under a small external memory pressure. The first kernel that shows no flat stage in its perceived memory bandwidth curve defines the intensive bandwidth boundary ❸ in Fig. 6.

[4] We run the kernels in the normal region and the intensive region on the target processor and vary the external memory pressure. The average of demanded external memory pressure that makes the target processor exhibit a flat trend in the perceived bandwidth gives the contention balance point ❹ in Fig. 6.

[5] The average achieved performance reduction rate is the reduction rate of the normal contention region ❺.

After these steps, the memory interference performance model of a processor is constructed for the HSM-SoC. The construction process of our model uses the calibrators (synthetic traffic generators) based on the *source-oblivious assumption*, that is, the amount rather than the source of external traffic matters. It makes the construction processor-centric, avoiding running many combinations of co-running scenarios.

TABLE V
OUR EXPERIMENT PLATFORM

NVIDIA Jetson AGX Xavier	
CPU	8-core Carmel 64-bit ARMv8.2 @ 2265MHz
GPU	512-core NVIDIA Volta@1377MHz with 64 Tensor Cores
DLA	NVIDIA DLA @1395.2 MHz, 512KB Convolutional buffer
Memory	16GB 256-bit LPDDR4x @ 2133MHz — 137GB/s
Qualcomm Snapdragon 855	
CPU	8-core Kryo 485 64-bit ARMv8.2 @1.8GHz
GPU	Qualcomm® Adreno™ 640 GPU
Memory	16GB 64-bit LPDDR4x @ 2133MHz — 34GB/s

IV. EMPIRICAL VALIDATION OF THE SLOW-DOWN MODEL

Our slowdown model explained in Section III is built via synthetic benchmarks. In this section, we validate the prediction accuracy of our model using several widely used benchmark kernels and neural networks.

A. Setup and Methodology

Target architectures: We use two models of real heterogeneous SoC for experiments. One is NVIDIA Jetson Xavier autonomous system [2] which consists of CPU, GPU and DLA. The other is Qualcomm Snapdragon 855 [3] mobile platform consisting of CPU and GPU. Table V shows the architecture details.

Model construction: We follow the methodology detailed in Section III-B to build the models. For CPU and GPU, we

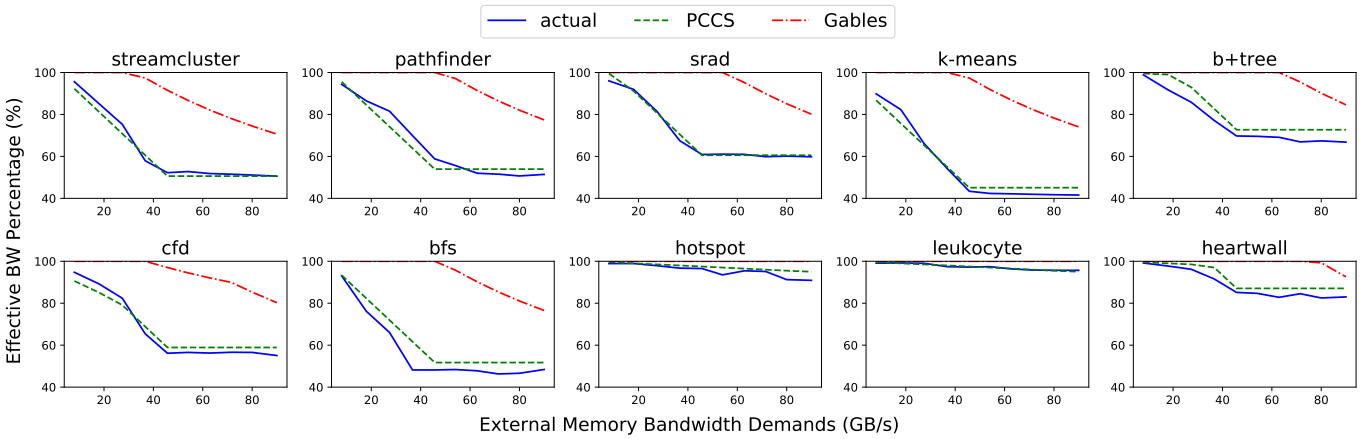

Fig. 7. The predicted and actual slowdowns of 10 Rodinia benchmarks on Xavier GPU

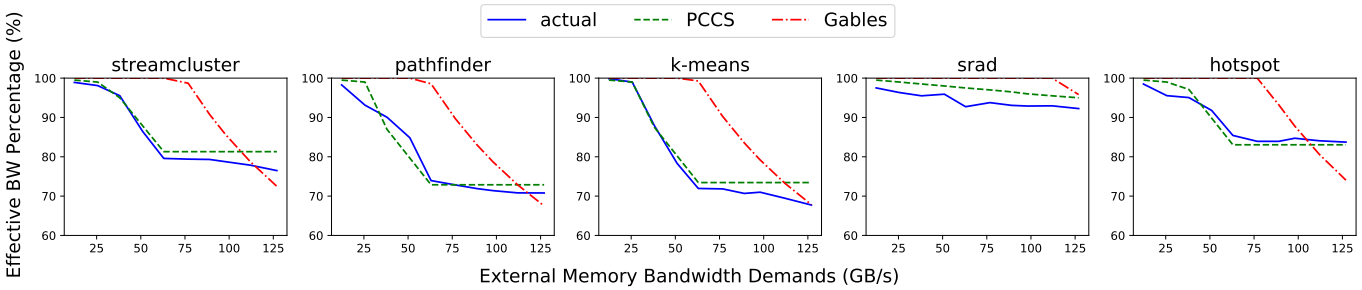

Fig. 8. The predicted and actual slowdowns of 5 Rodinia benchmarks on Xavier CPU

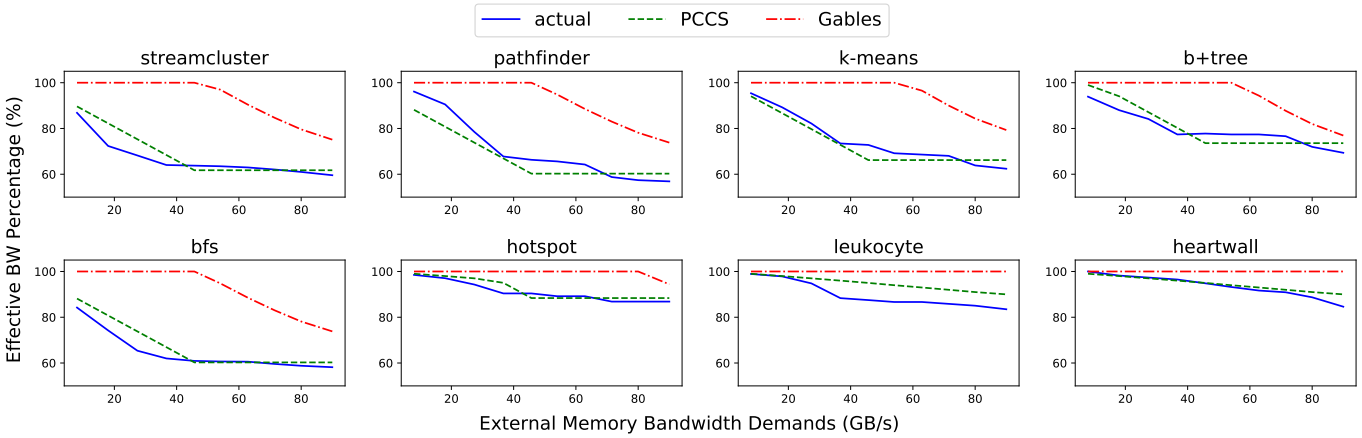

Fig. 9. The predicted and actual slowdowns of 8 Rodinia benchmarks on Snapdragon 855 GPU

employ vector-add kernels with different operational intensities; for DLA, we use MNIST neural network and control its operational intensities by varying convolution filter sizes.

Application selection: We evaluate our CPU and GPU slowdown model on Rodinia benchmarks [9], and our DLA model on ImageNet [11] with ResNet-50 and VGG19 models. We select 10 Rodinia benchmarks: three of them, `hotspot` (HS), `leukocyte` (LC) and `heartwall` (HW), are compute intensive and 7 of them, `streamcluster` (SC), `pathfinder` (PF), `srad`, `k-means` (KM), `b+tree` (BT), `CFD` and `BFS`, are memory intensive.

Bandwidth characterization: To find requested memory bandwidths of applications and kernels, we need only the standalone BW rates which can be obtained through nvprof, perf or Valgrind.

External memory pressure: To create contention, we run synthetic kernels on other PUs. For the CPU model, we create the external pressure using the GPU; for the GPU and DLA models, we create the external pressure using the CPU. For every benchmark, we vary the external pressure by changing the BW request of the other processor from 10% to 100% of the peak DRAM bandwidth with a 10% peak BW as the stride.

Baseline: Gables [16] is the closest and most recent work that proposes a sharing-aware analytical model for different types of processors that run on HSM-SoCs. The memory

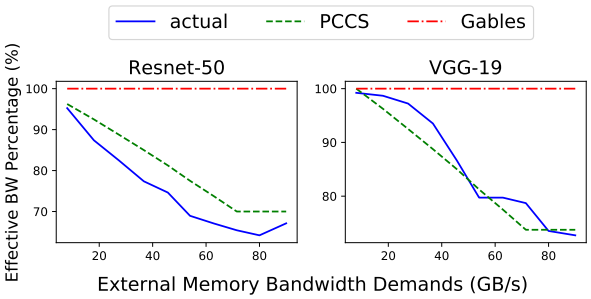

Fig. 10. The predicted and actual slowdowns of VGG19 and Resnet50 on the DLA

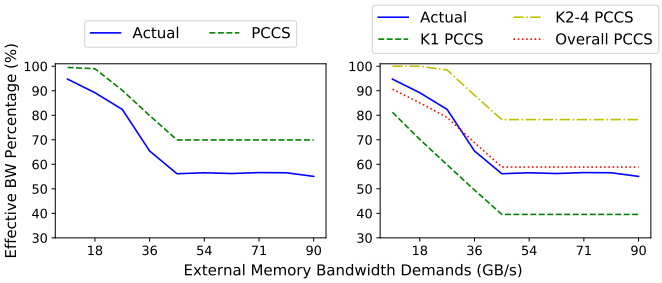

Fig. 11. The predicted slowdowns of CFD with (a) average BW and (b) piece-wise BW

contention model proposed by *Gables* assumes that the effective bandwidth of a processor under contention is not reduced as long as the total BW requested is smaller than SoC peak BW. Otherwise, the effective BW is calculated by prorating the requested BW to the available BW. This assumption constitutes the primary difference between our model, *PCCS*, and *Gables*.

B. Validation Results

Fig. 7 shows actual slowdown of 10 Rodinia benchmarks running on the Xavier GPU as well as the predicted slowdowns by *PCCS* and *Gables*, under varying amounts of external memory contention. The average error of our model for GPU is 6.3%. For the benchmarks that have small requested BWs, their effective BW percentage is close to the standalone one under memory contention. For other benchmarks that need medium BW in standalone runs, the slowdown for lower external memory pressure is minimal. However, as the pressure increases to exceed a certain level, their performance drops significantly with the external pressure, and eventually the slowdown flattens, as predicted by our three-region contention categorization. Our model consistently and accurately classifies and predicts these trends. The highest prediction error occurs with the `BFS` kernel, which has a poor locality that is affecting the row buffer hit rates significantly.

Fig. 8 compares the predicted (*PCCS* and *Gables*) and actual throughput percentages of 5 Rodinia benchmarks that are run on the Xavier CPU under external memory contention. Overall, the average error of our model for CPU runs is 2.6%. Since `hotspot` is computation intensive, it belongs to the minor contention region. Other four benchmarks belong to the normal contention region. Once again, our model predicts the contention regions for all kernels accurately. The accuracy slightly decreases during the flat regions of `BFS`, `k-means` and `streamcluster`, however the upper bounds for those regions still hold and overall has significantly less errors when compared to Gables.

Fig. 9 compares the predicted (*PCCS* and *Gables*) and actual slowdowns of 8 Rodinia benchmarks running on the Snapdragon 855 GPU, under varying amounts of external memory contention. In this experiment, the average error of our model for GPU is 5.6%. The results on Snapdragon CPU are even more accurate, with only 3.1% average error in the predicted slowdown. As Snapdragon architecture uses different memory controllers and different PU designs from Xavier, programs show different standalone bandwidth demands. Despite the differences of the two architectures, our PCCS model gives accurate prediction of co-run slowdown on both of them.

The effective BW percentages for DLA runs on Xavier are shown in Fig. 10. The average error of *PCCS* slowdown prediction for this PU is 5.3%. Since the DLA is a specialized processor for inference, we observe that the DLA can only achieve 20-30GB/s bandwidth in most standalone runs. Therefore, the DLA's slowdown under external pressure only falls under the normal contention region, while the kernels running on DLA are still sensitive to external memory pressure. As shown in the Fig. 10, the effective BW keeps reducing until ~70 GBpS of external pressure and there is only a small flat region at the higher end of external pressure demand.

In comparison, the average prediction errors of the *Gables* model on the GPU, CPU and DLA on Xavier are 39%, 10.3% and 26.7%, and on the CPU and GPU on Snapdragon are 8.1% on and 37.6% respectively. The reasons for the low accuracy are (i) it assumes the peak bandwidth is always achievable, and ignores the effects of contention; (ii) it assumes that the effective bandwidth can be proportionally divided down when the requested BW exceeds the DRAM capacity.

Programs with phase shifts. We also test the prediction accuracy of our model with applications that involve shifts of phases and exhibit obvious changes in memory bandwidth demands. `CFD` is such a program, which embeds 4 different kernels where one of them (K1) is a high BW kernel and the other three (K2-4) are medium BW kernels. When we use the average BW of the 4 kernels to characterize the interference behavior of CFD, as shown in the results given Fig. 11 (a), our model's prediction has an error rate of 19.4%. It is because high BW demanding kernels suffer a larger slowdown, but using the average BW as the input causes our model to underestimate the slowdown. When we use the BW demand of each kernel as the input to our model and combine the predictions of different kernels by using the execution time percentage of each kernel as the weight, as shown in Fig. 11 (b), the prediction error drops to 4.6%. It indicates the usefulness of our model on code with phase shifts; phase detection [17], [31], [32] is a well studied topic and is orthogonal to this work.

V. Design Space Exploration via PCCS

In this section, we explore how PCCS helps with the early-design stages of HW design. We first present a simplified,

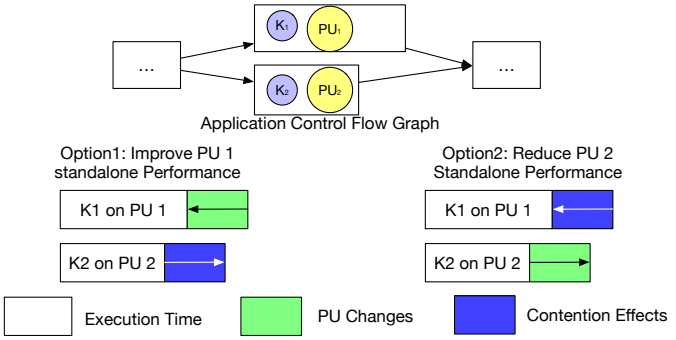

Fig. 12. Design options under collocated execution scenarios.

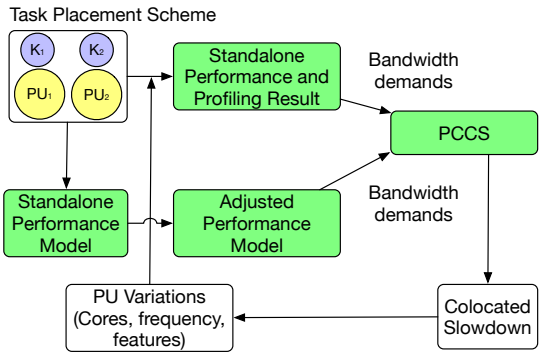

Fig. 13. The workflows of using PCCS.

two-step recipe to determine architectural features of PUs for a future SoC. Then, to evaluate our models and Soc-roofline models, we present region-based guidance and quantitative guidance for both models.

A. Architectural Feature Determination under Contention

The primary benefit of using PCCS in early stages of SoC design is *to factor shared memory contention effects into the design and prevent integration of unnecessary features by predicting the true performance of PUs under contention*. We simplify this process as follows, as depicted in Figure 12. The sample application control flow graph (CFG) has kernel 1 (K1) and kernel 2 (K2) can be placed on PU 1 and PU 2 respectively and they can be executed in parallel. K1 on PU 1 is in the critical path and we want to reduce the total execution time of co-located kernels K1 and K2.

Assuming there will be memory contention between K1 and K2, there are two options to reduce the execution time of this application by reducing the execution time of the critical path:

1) One option is to improve PU 1's standalone performance directly. Due to the memory contention, when we improve PU 1 standalone performance by $x\%$, the achieved performance may only improve $y\%$ where $y \leq x$. On the other hand, due to the contention, the execution time of k2 on PU 2 will likely increase, as PU 1 requests more bandwidth. So, we also need to ensure that the negative effect on K2 is not large enough to turn it into the critical-path kernel.
2) The other option is to reduce the standalone performance K2 on PU 2, hence generating less memory pressure. Due to the memory contention, when we reduce PU 2's standalone performance by $x\%$, the achieved performance may only reduce to $y\%$ where $y \leq x$. On the other hand, K1 on PU 1 will be able to obtain more bandwidth from memory contention. Similarly, we also need to ensure that the changes to the execution time of K2 does not deteriorate the overall performance.

In a real-life execution scenario, the actual memory contention effects from standalone performance changes will be more complicated than the above example. We need accurate memory contention models to be able to predict and reflect the effects of these changes to the design and optimization of HSM-SoCs. Our proposed model, PCCS is capable of helping in more complicated cases. In the rest of this section, we give a step-by-step walk-through over several use cases.

B. Workflow of using PCCS

The workflow of using PCCS is shown in Figure 13. For a given task placement scheme from the application CFG, we can obtain the standalone memory BW requirements of K1,K2, profiling results, and bandwidth model of each PU. We use the total external BW and the requested BW as inputs for our slowdown model to obtain collocated execution slowdown so that designers can determine architectural features, including changing the number of cores, changing the frequency, and adding hardware features (e.g., SIMD). While the designers can use an adjusted architecture to obtain new profiling results or they can use an adjusted performance model to do so. Then, these BW demands values can be fed into our slowdown model to obtain a slowdown prediction that can be used to decide whether this PU variation is effective. We will use the above steps as part of our use-case scenarios in the rest of this section.

C. Region-based Hardware Design Guidance

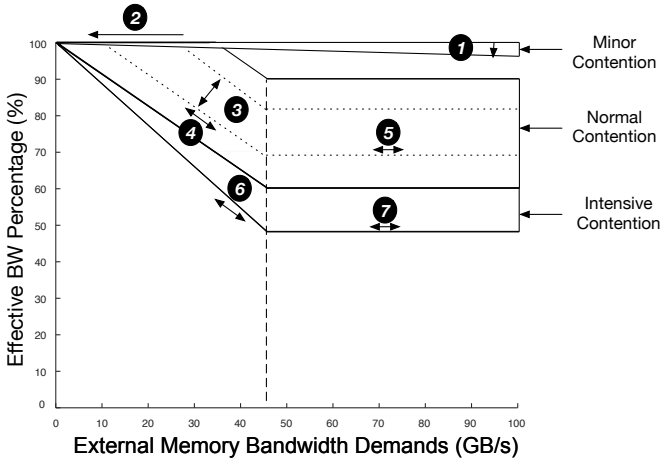

Fig. 14. Region-based guidance for our slowdown model, PCCS

PCCS provides region-based architecture guidelines, which indicate the type of different cases (program characteristic and total external memory pressure) and the consequences of various architectural modifications for each type. We use Figure 14 to explain it as follows:

- When the requested BW from a PU belongs to the minor contention region, its effective BW has a very small slowdown. When we modify the hardware design and ensure that requested BW belongs to the minor contention

region, the effective performance is almost the same as the standalone performance, ❶ in Figure 14.

- When the requested BW from a PU belongs to the normal contention region, if we modify the hardware to change the standalone performance and ensure that the new requested BW belongs to the minor contention region, its perceived performance will be almost the same as its standalone performance when the external memory pressure is small, as marked by ❷. When the external memory pressure is medium or large, if we improve its standalone performance, the perceived slowdown is more. The slowdown curve moves from the higher dotted line to the lower dotted line. If we reduce its standalone performance, its perceived slowdown becomes less. In this case, the slowdown curve moves from the lower dotted line to the higher dotted line, as marked by ❸.
- When the requested BW from a PU belongs to the intensive contention region, if we modify the hardware to change the standalone performance and ensure that the new requested BW belongs to the intensive contention region, similar to how the normal contention region dotted curve moves, the perceived slowdown is more if we improve the standalone performance, and less if we reduce its standalone performance, as marked by ❻.
- When the external memory BW changes, perceived slowdown is very small for the minor contention region. The perceived slowdown curves in the normal contention region have three stages for different external BW: flat, reducing and flat, as shown by ❷, ❹ and ❺, respectively. The perceived slowdown curves in the intensive contention region have two stages for different external BW: reducing and flat, as marked by ❻ and ❼. Its new slowdown can be obtained in the corresponding curve.

D. Quantitative Hardware Design Guidance

Methodology: To evaluate whether our memory interference slowdown model provides accurate predictions for architecture design space explorations, we evaluate our model by changing the architecture power settings of the HSM-SoC. Since building new devices to measure actual slowdown on new devices is not realistic, we use the following method to enumerate the steps to obtain the actual slowdown on future SoC using existing PU variations. We build our model with the maximum frequency, active cores and memory frequency setting by running various synthetic programs. We use the same slowdown model from the maximum power setting to predict memory interference slowdown under different architectural power settings, as Xavier allows users to write their own power profiles, like frequency, number of active cores and etc. After that, we change the power profiles and re-run the benchmarks with co-located programs to obtain actual slowdown. For every benchmark and every power profile, we use 10 different sums of other processors' standalone bandwidth from 10% peak bandwidth to 100% peak bandwidth with a 10% peak bandwidth stride. We use the same device, and use the same methodology in Section IV to construct our model and obtain the prediction from Gables (i.e. SoC-Roofline).

Predicting standalone performance on new devices (PU variations) is orthogonal to our work. Even though we proposed to use adjusted BW roofline to obtain standalone performance on PU variations, to avoid introducing another level of inaccuracy from standalone performance prediction to conceal insights provided by our work, we use measured standalone bandwidths of different programs on different power settings as the inputs to our model and Gables model.

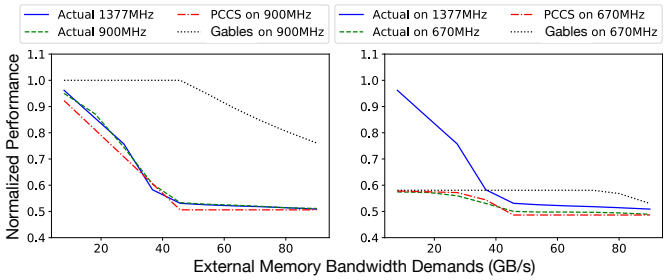

Fig. 15. Architectural feature exploration for GPU using streamcluster.

Use cases: Figure 15 shows the actual and prediction of a memory intensive benchmark, `streamcluster`, on a GPU over different frequency settings. The y-axis is the normalized performance. The solid line is the actual performance under maximum frequency. When we change the frequency from 1377MHz to 900MHz, its actual performance is close to that at 1377MHz, as reducing computation power of PU only affects a little memory intensive program performance. When we change the frequency from 900MHz to 670MHz, its standalone performance is affected a lot, and the requested BW decreases accordingly. However, under a certain external memory pressure, its perceived is similar to that in the 1377MHz setting, since high BW demanding programs experience a larger slowdown percentage. Therefore, under a range of external memory pressure in collocated scenarios, our PU can save 51.2% computation unit power budget from PU frequency while providing similar performance. Our slowdown model accurately predicts this phenomenon. We also observe a similar phenomenon when we halve active cores for benchmark `streamcluster`, indicating that our model can save 50% computation unit area space from the number of cores while providing similar performance in collocated scenarios.

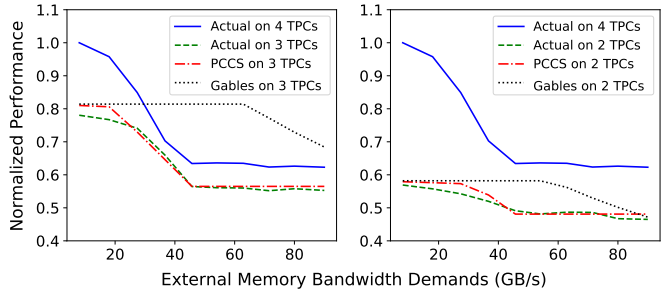

Fig. 16. Architectural feature exploration for GPU using srad.

Figure 16 shows the actual and predicted performance of `srad`, a computation intensive benchmark with high requested

BW, running on a GPU over a different number of texture processor clusters (TPC). One TPC consists of a number of SMs. The y-axis is the normalized performance. The solid line is the actual performance in the maximum frequency. When we change the number of active TPCs from 4 to 3, its actual standalone performance becomes about 20% lower, as it is a computation intensive program. However, under a certain external memory pressure, the perceived performance of 4 TPCs is only 8% better than the perceived performance of 3 TPCs. Therefore, under a range of external memory pressure in collocated scenarios, the configurations based on our model can save 25% computation unit area space from the number of cores while keeping similar performance. When we change the number of active TPCs from 4 to 2, its perceived performance reduces 23%, which is less than its standalone reduction, 40%. This reduction might be acceptable if this task is not in the critical path in the entire application.

TABLE VI
PREDICTION ERRORS (%) OF ARCH. DESIGN EXPLORATION ON GPU

	Our method					Gables (SoC-Roofline)				
	TPC		Feq. MHz		Avg.	TPC		Feq. MHz		Avg.
	3	2	900	600		3	2	900	600	
PF	9.8	8.9	11.7	7.7	9.6	45.1	37.0	31.6	32.0	36.4
SC	6.0	4.8	5.9	3.1	5.0	52.5	50.9	52.7	10.5	41.6
HS	3.6	3.2	2.0	1.0	2.4	4.9	4.7	4.0	3.0	4.2
LC	4.7	1.7	0.7	1.1	2.0	5.5	2.3	2.5	3.1	3.4
KM	26.9	53.7	9.1	5.5	23.8	69.2	61.8	75.7	31.9	59.7
Srad	3.6	4.6	4.7	2.4	3.8	27.3	9.9	14.9	9.9	15.5
BFS	16.4	28.7	6.6	3.6	13.8	61.9	54.7	19.8	12.5	37.2
BT	19.7	24.5	6.6	2.9	13.4	23.7	18.5	19.8	8.1	17.5
CFD	8.2	8.4	11.8	5.8	8.5	37.9	29.0	21.0	14.9	25.7
HW	3.5	3.2	7.3	3.9	4.5	9.2	4.5	7.0	8.3	7.3
Avg.	10.2	14.2	6.6	3.7	8.7	33.7	27.3	24.9	13.4	24.8

TABLE VII
PREDICTION ERRORS (%) OF ARCH. DESIGN EXPLORATION ON CPU

	Our method					Gables (SoC-Roofline)				
	Cores		Feq. MHz		Avg.	Cores		Feq. MHz		Avg.
	6	4	1780	1450		6	4	1780	1450	
PF	3.2	3.6	2.9	2.9	3.1	16.5	13.7	19.1	11.6	15.2
SC	3.5	5.7	4.2	4.5	4.5	15.2	13.0	13.8	11.2	13.3
HS	2.7	5.9	1.7	5.5	3.9	12.7	9.1	11.0	13.4	11.5
KM	4.4	3.9	3.8	2.4	3.6	21.2	12.3	21.3	11.9	16.7
Srad	1.9	1.8	0.8	1.5	1.5	13.9	5.9	4.8	8.1	8.2
Avg.	3.1	4.2	2.7	3.4	3.3	15.9	10.8	14.0	11.2	13.0

Detailed Results: Table VI and Table VII show detailed prediction errors for different benchmarks and different power settings. Overall, our method reduces average prediction errors from 24.8% to 8.7% on GPU, from 13.0% to 3.3% on CPU. The average prediction error for computation intensive programs with small bandwidth (`Leukocyte`, `hotspot`, `heartwall`) is small, since low BW demanding programs are less sensitive to external memory pressure. Our slowdown models are able to predict memory intensive programs or computation intensive programs with high bandwidth accurately, while Gables model suffers large errors. The prediction is more accurate when power varies than when the number of cores varies. Changing the frequency does not affect the program data partition, but changing the number of cores does. As a result, `K-means`, `BFS` and `B+tree` have a relatively lower prediction accuracy due to task imbalance from data reparationing.

VI. RELATED WORK

There is a rich set of work on performance modeling with memory interference awareness. Table VIII lists some studies on memory interference modeling.

TABLE VIII
RELATED WORK COMPARISON

Related Work	Memory Interference Model	Accuracy	Architecture Design Exploration
Bubble-up [24]	Empirical	High	×
ASM [33]	Monitoring	High	×
Co-run [38]	Lookup Table	High	×
ESP [26]	Linear Regression	Medium	×
Gables [16]	Analytical	Low	✓
Ours	Empirical & Analytical	High	✓

Bubble-up [24] proposes to empirically measure the memory interference sensitive curve over different other processor memory pressures for each application. Each application has its own sensitive curve which has a high accuracy to predict the memory interference over memory pressure. Application slowdown model (ASM) [33] proposes to use a runtime monitor to observe the memory interference performance. Then, their monitor periodically gives the highest priority to one processor's memory requests. They assume the performance during the priority stage is the standalone performance. Then the memory interference slowdown percentage is derived without requiring a clean environment. Another work on co-run scheduling [38] proposes a memory interference performance lookup table to predict co-run performance. This lookup table is measured by profiling different co-run combinations. Another study [26] uses different co-run combinations to train a linear regression model.During the prediction, they use this linear regression model to match maximum likelihood co-run application characteristics. Then the memory interference performance is derived. Through a black-box method, their accuracy is lower than the previous three studies. The above four studies provide an accurate memory interference performance for runtime usage purposes, but is for post-silicon runtime uses rather than SoC designs.

VII. CONCLUSION

Heterogeneous SoC design has been facing the dilemma of accuracy and coverage. Accurate simulations are time-consuming, difficult to cover a large design space; rough estimations by experience or analytical models face high risks of inaccuracy. The new approach proposed in this work strives to bridge the gap. It provides a novel *processor-centric performance modeling* methodology, and a new *three region interference-conscious performance model*. The modeling process needs no measurements of co-runs of various combinations of applications, but the produced performance models can be used to estimate the co-run performance of arbitrary workloads on various SoC designs. The new method reduces average prediction errors of the state-of-art model from 24.8% to 8.7% on GPU, and from 13.0% to 3.3% on CPU, and has demonstrated much improved efficacy in guiding SoC designs in several use-case studies.

REFERENCES

[1] "Cs roofline toolkit," https://bitbucket.org/berkeleylab/cs-roofline-toolkit/src/master/, accessed July, 2020.

[2] "Nvidia tensor cores," https://devblogs.nvidia.com/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/, accessed Nov, 2020.

[3] "Qualcomm snapdragon 855 mobile platform," https://www.qualcomm.com/products/snapdragon-855-mobile-platform/, accessed Nov, 2020.

[4] "Snapdragon 855 mobile platform," https://www.qualcomm.com/products/snapdragon-855-mobile-platform, accessed Sep, 2019.

[5] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[6] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2012, pp. 416–427.

[7] R. Barik, N. Farooqui, B. T. Lewis, C. Hu, and T. Shpeisman, "A black-box approach to energy-aware scheduling on integrated cpu-gpu systems," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*. ACM, 2016, pp. 70–81.

[8] D. Black-Schaffer, N. Nikoleris, E. Hagersten, and D. Eklov, "Bandwidth bandit: Quantitative characterization of memory contention," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Computer Society, 2013, pp. 1–10.

[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[10] Y. Cho, F. Negele, S. Park, B. Egger, and T. R. Gross, "On-the-fly workload partitioning for integrated cpu/gpu architectures." in *PACT*, 2018, pp. 21–1.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[12] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems," *ACM Sigplan Notices*, vol. 45, no. 3, pp. 335–346, 2010.

[13] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multicore memory systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 2, p. 7, 2012.

[14] L. Gwennap, "Two-headed snapdragon takes flight," *Microprocessor Report*, vol. 323, pp. 1–6, 2010.

[15] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 37–47, 2010.

[16] M. Hill and V. J. Reddi, "Gables: A roofline model for mobile socs," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 317–330.

[17] M. Hind, V. T. Rajan, and P. F. Sweeney, "Phase shift detection: a problem classification," IBM Research, Tech. Rep. Report 22887, August 2003.

[18] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver, "A qos-aware memory controller for dynamically balancing gpu and cpu bandwidth use in an mpsoc," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 850–855.

[19] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–12.

[20] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 65–76.

[21] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[22] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 450–462.

[23] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, vol. 40, no. 6, pp. 190–200, 2005.

[24] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 248–259.

[25] J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "Contention aware execution: online contention detection and response," in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*. ACM, 2010, pp. 257–265.

[26] N. Mishra, J. D. Lafferty, and H. Hoffmann, "Esp: A machine learning approach to predicting application interference," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2017, pp. 125–134.

[27] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 146–160.

[28] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in *2008 International Symposium on Computer Architecture*. IEEE, 2008, pp. 63–74.

[29] T. M. O. Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," in *USENIX security*, 2007.

[30] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 128–138, 2000.

[31] X. Shen, Y. Zhong, and C. Ding, "Locality phase prediction," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004, pp. 165–176.

[32] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," in *Proceedings of International Symposium on Computer Architecture*, San Diego, CA, June 2003, pp. 336–349.

[33] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 62–75.

[34] Wikichip, "Apple a13 bionic," https://en.wikichip.org/wiki/apple/ax/a13, accessed Jan. 2020.

[35] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[36] Y. Xie and G. Loh, "Dynamic classification of program memory behaviors in cmps," in *the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2008.

[37] H. Zhu and M. Erez, "Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 33–47, 2016.

[38] Q. Zhu, B. Wu, X. Shen, L. Shen, and Z. Wang, "Co-run scheduling with power cap on integrated cpu-gpu systems," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 967–977.

[39] T. Zidenberg, I. Keslassy, and U. Weiser, "Multiamdahl: How should i divide my heterogenous chip?" *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 65–68, 2012.