A Comprehensive Methodology to Determine Optimal Coherence Interfaces for Many-Accelerator SoCs

Kshitij Bhardwaj
Harvard University, USA

Marton Havasi
University of Cambridge, UK

Yuan Yao
Harvard University, USA

David M. Brooks
Harvard University, USA

José Miguel Hernández-Lobato
University of Cambridge, UK

Gu-Yeon Wei
Harvard University, USA

ABSTRACT

Modern systems-on-chip (SoCs) include not only general-purpose CPUs but also specialized hardware accelerators. Typically, there are three coherence model choices to integrate an accelerator with the memory hierarchy: *no coherence, coherent with the last-level cache (LLC),* and *private cache based full coherence.* However, there has been very limited research on finding which coherence models are optimal for the accelerators of a complex many-accelerator SoC. This paper focuses on determining a *cost-aware coherence interface* for an SoC and its target application: find the best coherence models for the accelerators that optimize their power and performance, considering both workload characteristics and system-level contention. A novel comprehensive methodology is proposed that uses Bayesian optimization to efficiently find the cost-aware coherence interfaces for SoCs that are modeled using the gem5-Aladdin architectural simulator. For a complete analysis, gem5-Aladdin is extended to support LLC coherence in addition to already-supported no coherence and full coherence. For a heterogeneous SoC targeting applications with varying amount of accelerator-level parallelism, the proposed framework rapidly finds cost-aware coherence interfaces that show significant performance and power benefits over the other commonly-used coherence interfaces.

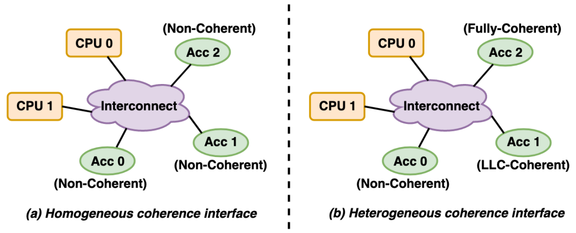
CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**.

ACM Reference Format:
Kshitij Bhardwaj, Marton Havasi, Yuan Yao, David M. Brooks, José Miguel Hernández-Lobato, and Gu-Yeon Wei. 2020. A Comprehensive Methodology to Determine Optimal Coherence Interfaces for Many-Accelerator SoCs. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '20), August 10–12, 2020, Boston, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3370748.3406564

1 INTRODUCTION

Modern SoCs not only consist of general-purpose processors but also application-specific accelerators, which efficiently execute specific tasks such as image processing or speech recognition. Recent SoCs utilize several accelerators, for example, *Tesla's Full Self Driving* ASIC integrates Arm A72 CPUs, a GPU, two neural network accelerators, and a video encoder [1]. Such integration of different compute units has given rise to the *heterogeneous SoC paradigm* [2].



(a) Homogeneous coherence interface

(b) Heterogeneous coherence interface

Figure 1: Accelerator coherence interfaces for an SoC

An accelerator can be integrated with the system's memory hierarchy using one of the three common coherence models: *non-coherent, coherent with the last-level cache (LLC),* or *fully-coherent* [3, 4]. In the non-coherent model, an accelerator typically uses software-managed direct memory access (DMA) to request data from the main memory, and stores it in its local scratchpad. In the LLC-coherent model, data is directly retrieved from the LLC into the local scratchpads. Finally, in the fully-coherent model, a private cache is used instead of a scratchpad, implementing a coherence protocol such as MESI, similar to the processor's cache.

These coherence models exhibit interesting cost trade-offs [4]. The non-coherent model is simple and involves minimal hardware overhead, but it suffers from performance/power overheads due to expensive main memory accesses. LLC-coherent can be more efficient than non-coherent for workloads with high LLC hit rates. However, its performance can suffer when several accelerators are operating concurrently, leading to significant LLC conflict misses. The fully-coherent model can be power-efficient due to the use of small private caches instead of larger scratchpads but can incur performance overheads when operating on a large streaming dataset, unable to fit in a small cache.

We define an *accelerator coherence interface* for a many-accelerator SoC as a set of coherence models selected for its various accelerators (Figure 1). There are two possible accelerator coherence interfaces: homogeneous, where all accelerators use the same coherence model (for example all fully-coherent accelerators), and heterogeneous with different models for the accelerators (some non-coherent, others LLC-coherent or fully-coherent). However, most of the existing works use homogeneous interfaces for many-accelerator SoCs [2, 5–9].

There has been very limited research on determining the optimal accelerator coherence interfaces for many-accelerator SoCs. This selection is challenging as it should consider a *twofold criterion* for best solutions: (i) the characteristics of the kernels being accelerated, e.g., their data access patterns, and (ii) the system-level properties, such as contention in the shared resources (e.g., shared LLC). Contention is a critical bottleneck for accelerator performance

ISLPED '20, August 10–12, 2020, Boston, MA, USA

ACM ISBN 978-1-4503-7053-0/20/08…$15.00
https://doi.org/10.1145/3370748.3406564

as modern SoCs utilize increasing number of accelerators in parallel (also called accelerator-level parallelism (ALP) [10]). To this end, this paper determines a *cost-aware accelerator coherence interface* for an SoC and its target application at design time, where the coherence models are selected based on the above twofold criterion in order to optimize the power and performance of the accelerators.

Contributions. We introduce a novel methodology, called *CHIME,* to automatically and efficiently find **C**ost-aware co**H**erence **I**nterfaces for **M**any-acc**E**lerator SoCs. CHIME uses gem5-Aladdin architectural simulator [11] to comprehensively model and evaluate the impact of different coherence interfaces on the power and performance of the accelerators in an SoC, and employs Bayesian optimization to rapidly converge to a cost-aware coherence interface for a target application. For a complete analysis, we extended gem5-Aladdin to support LLC coherence in addition to the already-supported no coherence and full coherence. *To the best of our knowledge, this is the first comprehensive methodology for determining optimal coherence interfaces for many-accelerator SoCs.*

CHIME can be utilized not only for finding the cost-aware coherence interface for an application-specific SoC but also for a general-purpose SoC that runs multiple applications. A coherence interface optimal for one application may not be optimal for others. In this case, CHIME is first used to determine a cost-aware coherence interface for each application separately, and then the performance/power overheads are evaluated using the chosen cost-aware coherence interface of each application for all the others. The final interface selected for the SoC is the one that leads to minimal overheads for all the target applications.

The effectiveness of CHIME is demonstrated by finding the cost-aware coherence interfaces for a many-accelerator SoC with 7 accelerators, targeting 9 different applications synthetically created to mimic real workloads such as image processing. These applications have varying degrees of ALP: from only one accelerator operating to all 7 running in parallel. For these applications, CHIME rapidly converges to cost-aware coherence interfaces that show up to 32.7% improvements in average accelerator performance and 51.3% lower total accelerator power compared to several baseline coherence interfaces. Moreover, if the optimization target is a low power, then we find that the same cost-aware coherence interface is obtained for each of the 9 applications, which then becomes the final interface for the SoC. Alternatively, if the target is high performance, then we select the cost-aware coherence interface corresponding to the application that uses all 7 accelerators concurrently as the final one. It incurs an average performance overhead of only 2.4% for the other 8 applications.

2 RELATED WORK

Most of the prior research have used homogeneous coherence interfaces for many-accelerator SoCs, while only a limited works use heterogeneous interfaces. All-non-coherent interface is used not only in early SoCs [8] but also very recently for integrating several accelerators for wearable electronics [12]. All-LLC-coherent has been used both in academia [2] as well as in industrial products such as in ARM Cortex-A72 processor [5]. Similarly, all-fully-coherent interface has been common in IBM's SoCs, for example, the wire-speed processor [9] and the CAPI interface [6]. Recently, there has been research that highlights the need for a heterogeneous coherence interface for many-accelerator SoCs [13], [4]. A hybrid coherence interface, called Spandex, correctly integrates CPUs and GPUs that use different coherence models [14]. However, none of these works provides a methodology to determine cost-aware coherence models for accelerators of an SoC.

A couple of recent works proposed methods to determine coherence models for the accelerators of an SoC, but have several limitations and are not comprehensive [15, 16]. The first work does not optimize for power, and it only chooses between non-coherent and LLC-coherent models while ignoring the commonly-used fully-coherent model [15]. The second proposed selecting the appropriate coherence models for the accelerators at runtime [16]. However, this approach is also limited in the following ways: (i) the impact of various coherence models on the accelerator's power is ignored, while our approach optimizes both power and performance; (ii) their coherence selection algorithm may not lead to optimal solutions in terms of performance. This algorithm selects models only on the basis of dynamic memory footprint while ignoring other factors, such as data access pattern. In contrast, our framework exhaustively considers these accelerator characteristics; and (iii) their runtime approach requires support for all of the coherence models for every accelerator, which is not scalable and can lead to significant power/area costs. In contrast, our technique only needs support for a single coherence model per accelerator as the cost-aware models are determined during design time.

3 BACKGROUND

Background on the two main threads of this paper is presented: gem5-Aladdin simulator and Bayesian optimization (BayesOpt).

gem5-Aladdin overview. This tool has the capability to model a complex SoC and all the interactions between its different components. It integrates gem5 architectural simulator [17] with the Aladdin accelerator simulator [18]. In gem5-Aladdin, a CPU invokes an accelerator using *ioctl* system calls to begin simulation using Aladdin. Aladdin is a trace-based simulator that profiles the dynamic execution of a kernel, and constructs its data dependence graph. Various design optimizations, such as loop unrolling, can then be applied followed by scheduling and execution of the graph, and returning of the control to the CPU.

Bayesian optimization. Bayesian optimization has been shown to be highly-effective for optimizing black-box functions [19], [20]. These functions are expensive to evaluate and cannot be expressed as closed-form expressions. BayesOpt intelligently and rapidly reduces the search space such that only minimal objective function evaluations are performed.

BayesOpt first evaluates the objective functions at random inputs. This initial sampling is followed by intelligently selecting those inputs that tend to optimize the objectives. In more details, the algorithm builds a Bayesian statistical model of each of the objective functions using the initial random sampling. Usually, a Gaussian process (GP) is used for this modeling, which is a probabilistic model that describes the distribution of the objective function ($f(x)$) at some candidate input x [21] based on the past observations. These GP models are then updated as the algorithm proceeds and samples new input values. The selection of these inputs is determined by the evaluation of an *acquisition function,* which is computed using the GP-predicted objective values [19]. The algorithm keeps sampling those inputs that maximize the acquisition function until all the optimal solutions are found.

Choice of GP parameters. A GP distribution is defined by a *mean* and a *covariance function.* The mean reflects the expected value of a function at some input before any observations. The covariance function, called the *kernel,* models the dependence between the function values at two different inputs. In this paper, the widely-used *squared exponential (SE)* kernel [21] is used.

Choice of acquisition function. In this paper, the *S-Metric-Selection-based Efficient Global Optimization (SMS-EGO)* is used as the acquisition function, which has been shown to be highly-effective for multi-objective optimization and handling a large design space [22]. SMS-EGO uses a *hypervolume metric* to determine the degree to which a candidate point is optimal. Hypervolume is defined as the volume enclosed between a candidate point and a fixed reference point in the Pareto space. The reference point is usually chosen to be the maximal value in each objective function. The hypervolume of a Pareto-optimal point is higher than a non-optimal point. The idea is to maximize the hypervolume until all the points with the highest hypervolume (i.e, Pareto-optimal) are found.

4 MODELING COHERENCE INTERFACES

In this paper, gem5-Aladdin is extended to support LLC-coherence in addition to the already-supported no-coherence and full-coherence models. Details on this extension and on the correctness of a system using a heterogeneous accelerator coherence interface are now provided.

Extension to LLC coherence. An accelerator coherency port (ACP) interface is implemented in gem5-Aladdin for LLC-coherence. ACP is a 64-bit bus interface between an accelerator scratchpad memory and the LLC. Only limited modifications are made to gem5's Ruby memory simulator to support ACP with minimal overheads. First, a new ACP controller is added for the accelerators to issue ACP-read and ACP-write requests from the scratchpads to the LLC. Second, the LLC controller is augmented to receive the new ACP requests. Finally, gem5's MESI protocol is modified to add two new transient states in the LLC controller, while keeping the stable states (M,E,S,I) unmodified. The two states are needed to handle the ACP read and write misses in the LLC: we call these states *ACP-R* and *ACP-W,* respectively. No new states are needed for ACP hits as the LLC is simply accessed for data. In cases when ACP read/write arrives at the LLC and the requested cacheline is still owned by a private L1, the request will be forwarded to the exclusive owner. If it is a read request, the owner will directly send the data to the accelerator, or if it is a write request, the owner will write the data back to the LLC. After receiving the data, the LLC performs the actual modification. No extra transient states are needed in these cases as the existing ones in the MESI protocol (for similar cache-to-cache transfers) can be re-used.

In more detail, when in *ACP-R,* a read request is sent to the DRAM, and a stall is issued to the requesting unit to wait for the response from the memory. In *ACP-W,* in addition to the issuing of requests to the DRAM and stalling, all the other copies of the target cache line are also invalidated. Further, during the ACP misses, these states also merge a new main memory request with any previously pending requests, if these are for the same cache line. Hence, avoiding extra requests and improving power-performance efficiency. Moreover, no additional cache storage is required as the transient states are stored in the miss status holding registers.

Correctness. An SoC using a mix of no coherence, LLC coherence, and full coherence for its different accelerators operates correctly. In case of the non-coherent model, the CPU flushes its caches both before an accelerator is invoked, and also after the accelerator completes processing and writes the data back to the main memory so as not to read any stale data. In both full coherence and LLC coherence, the accelerators follow the same coherence protocol as the CPUs (MESI) for correctness.

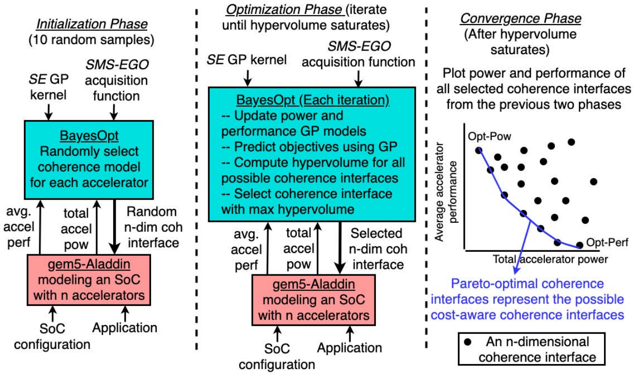

Initialization Phase
(10 random samples)
SE GP kernel
SMS-EGO acquisition function
BayesOpt
Randomly select coherence model for each accelerator
avg. accel perf
total accel pow
Random n-dim coh interface
gem5-Aladdin
modeling an SoC with n accelerators
SoC configuration
Application
Optimization Phase (iterate until hypervolume saturates)
SE GP kernel
SMS-EGO acquisition function
BayesOpt (Each iteration)
-- Update power and performance GP models
-- Predict objectives using GP
-- Compute hypervolume for all possible coherence interfaces
-- Select coherence interface with max hypervolume
avg. accel perf
total accel pow
Selected n-dim coh interface
gem5-Aladdin
modeling an SoC with n accelerators
SoC configuration
Application
Convergence Phase
(After hypervolume saturates)
Plot power and performance of all selected coherence interfaces from the previous two phases
Opt-Pow
Opt-Perf
Average accelerator performance
Total accelerator power
Pareto-optimal coherence interfaces represent the possible cost-aware coherence interfaces
● An n-dimensional coherence interface

Figure 2: CHIME operation for a given application

5 PROPOSED CHIME METHODOLOGY

This paper determines a *cost-aware accelerator coherence interface* for an SoC and its target application at design time. In such an interface, those coherence models are selected for the accelerators that optimize their power and performance, considering an exhaustive twofold criterion for best solutions: (i) the characteristics of the accelerators' target kernels (data access pattern, memory footprint, etc.), and (ii) the interaction of the accelerators with the SoC, such as the impact on contention in shared resources (e.g., LLC/NoC).

We propose a comprehensive methodology, called *CHIME,* to efficiently determine **C**ost-aware co**H**erence **I**nterfaces for **M**any-acc**E**lerator SoCs. In CHIME, the extended gem5-Aladdin (Section 4) is used to model the above twofold criterion and evaluate the effects of different coherence models (no coherence, LLC coherence, and full coherence) on the power and performance of the accelerators for a given application. BayesOpt is used in CHIME to intelligently tune the accelerators' coherence models and rapidly converge to the cost-aware coherence models. Two objective functions are minimized: (i) average (geo-mean) of the cycles taken by the accelerators to execute their respective kernels, and (ii) total accelerator power dissipated while running these kernels, which includes both static and dynamic power (memory and functional unit) of all the accelerators. The approach, however, is general and can use other coherence models as well as optimize different cost functions. CHIME can also be extended to GPUs, which however, will require a coherence wrapper such as Spandex [14] for correctness. GPU extension is left as future work.

This section first presents how CHIME selects a cost-aware coherence interface for an SoC and its target application. It then describes how CHIME can be utilized for general-purpose SoCs in order to select a single cost-aware coherence interface at design time that can be used by multiple applications.

CHIME operation for a given application of an SoC. There are three phases of CHIME (Figure 2): initialization, optimization, and convergence. In the initialization phase, BayesOpt performs random sampling, where it generates 10 random coherence interfaces. Each of these interfaces is *n*-dimensional with a coherence model randomly selected for each of the *n* accelerators of the SoC. gem5-Aladdin evaluates the power and performance metrics for these interfaces, which are fed back to BayesOpt.

After initialization, the framework enters the iterative optimization phase. During each iteration, BayesOpt first updates the GP

Component	Parameters
CPU Cores	Out-of-order X86 @1 GHz 8-μop issue width, 192-entry ROB
L1 Caches	64 KB, 4-way associative, 64 B cacheline, LRU
L2 Cache (LLC)	2 MB, 16-way associative, LRU, 1 LLC controller
DRAM	LP-DDR4, @1600 MHz, 4 GB, 25.6 GB/s, 1 memory controller
Accelerators	2 FFT Transpose, 3 Sparse matrix vector multiply (SPMV), and 2 Stencil-3Ds
NoC	4 × 4 2D-mesh topology, link bandwidth: 128 bits, single-cycle routers/links

Table 1: SoC micro-architecture configuration

models of the two objectives based on the available data (for the first iteration, this data are the initial random samples). Next, these GP models are used to predict the values of the objective functions for all possible n-dimensional coherence interfaces. These predicted values are then used to compute the SMS-EGO acquisition function (i.e. hypervolume) for all possible coherence interfaces. The interface with the maximum hypervolume is then selected. For this coherence interface, the power and performance metrics are then evaluated using gem5-Aladdin. These values are fed back to BayesOpt to update the GP models in the next iteration, followed by selecting a new interface that maximizes the hypervolume. The iterations continue until no further improvement in hypervolume is seen and the convergence is achieved.

In the final convergence phase, the values of the two objectives, obtained from gem5-Aladdin, are plotted for all the n-dimensional coherence interfaces that were selected by BayesOpt in the previous two phases. The Pareto-optimal interfaces are extracted from this plot, which represent the possible cost-aware coherence interfaces for the SoC and its given application. A final interface can then be selected from these cost-aware coherence interfaces based on the SoC's optimization target: performance or power. For example, if the system is targeted for low power, the Pareto-optimal coherence interface with the lowest total accelerator power is chosen (point Opt-Pow in Figure 2), else if the target is high performance, the Pareto-optimal interface with the lowest average accelerator execution time is selected (point Opt-Perf).

Extension to general-purpose SoC. If an SoC is targeted for multiple applications, CHIME can be used to find the cost-aware coherence interface for each of these applications at design time, and then select one of these interfaces to be used for all the target applications. The selected interface, while optimal for one of the applications, may lead to overheads for the others as different applications can have varying amount of ALP. The idea is to choose that interface which leads to minimum performance/power overheads for all the applications. To this end, the following steps are used if the SoC's optimization target is high performance: (i) using CHIME, find Opt-Perf interfaces for each of the SoC's applications; (ii) using gem5-Aladdin, determine the performance and power overheads of using Opt-Perf of each application for all the other applications; (iii) select the Opt-Perf interface with the least average performance overheads for all the applications; and (iv) if there are more than one such interfaces then select the one with lower power overheads. Similar approach is used when the target is low power, but Opt-Pow interfaces are considered and priority is given to power overheads.

6 EXPERIMENTAL RESULTS

This section presents the experimental setup (SoC configuration and the workloads), followed by detailed results to show the effectiveness of CHIME for different applications.

6.1 Experimental Setup

Table 1 shows the SoC configuration modeled in gem5-Aladdin. The modeled SoC consists of 7 accelerators: 2 FFT-transpose accelerators (FFT0/1), 2 Stencil-3Ds (Stencil0/1), and 3 SPMV accelerators (SPMV0/1/2). These kernels are taken from the *Machsuite benchmarks* [23]. The accelerators, along with 7 CPUs, and LLC/memory controllers are connected to different routers of a 4 × 4 NoC. This SoC is modeled to mimic support for several read-world applications, e.g., SPMVs can be used for linear algebra solvers, data/graph analytics, and partial differential equation solvers; Stencil-3Ds can be used for FIR filtering and 1D convolutions; and the FFTs can be used for image feature extraction and DSP for speech compression.

Application configurations	Accelerator(s) running
App1	FFT0
App2	SPMV0
App3	Stencil0
App4	SPMV0 and Stencil0
App5	SPMV0, SPMV1, and Stencil1
App6	SPMV0, SPMV1, Stencil0, and Stencil1
App7	SPMV0, SPMV1, SPMV2, Stencil0, and Stencil1
App8	FFT0, FFT1, SPMV0, SPMV1, Stencil0, and Stencil1
App9	All 7 accelerators running concurrently

Table 2: Applications with varying amount of ALP

To thoroughly demonstrate the effectiveness of CHIME, we created 9 synthetic applications that use different accelerators with varying amount of accelerator-level parallelism (Table 2). These synthetic applications are based on real workloads that can run several accelerators in parallel, such as for image and speech processing [10, 24]. While App1-App3 use only a single accelerator, App4-App9 use 2-7 accelerators concurrently. Furthermore, when used as non-coherent/LLC-coherent, the following scratchpad memory sizes are used for these accelerators to fit the entire operating dataset of the kernels: 8 KB for FFT, 512 KB for SPMV, and 256 KB for Stencil-3D, On the other hand, when used as fully-coherent, these accelerators use 64 KB L1 caches.

6.2 Results

This section presents how the performance and power of the cost-aware coherence interfaces, obtained for each of the above 9 applications using CHIME, compares with several baseline coherence interfaces. More detailed analysis is then presented for App9 that uses all the accelerators concurrently. Finally, a single cost-aware coherence interface is selected for the targeted SoC's 9 applications.

Cost-aware coherence interface for each application. Figures 3 and 4 show the performance and power of the cost-aware coherence interfaces obtained using CHIME for the 9 applications. For each application, two cost-aware coherence interfaces are selected: *Opt-Perf* that achieves the best performance among the Pareto-optimal interfaces (as shown in Figure 2), and *Opt-Pow* that achieves the lowest power among these points. Four other baseline interfaces are used for comparisons: three homogeneous coherence interfaces, and a heterogeneous interface obtained at design time based on the coherence model selection algorithm of [16]. The homogeneous baselines are highly common [2, 5–9]: (i) *All-NC (All-Non-Coherent),* (ii) *All-LLC (All-LLC-Coherent),* and (iii) *All-FC (All-Fully-Coherent).* The heterogeneous baseline (called *Mem-Footprint-Only*) does not optimize for power, and selects the coherence models based only on the accelerators' memory footprints while ignoring workload attributes such as data access pattern. For the target application, the Mem-Footprint-Only will choose FFT0 and FFT1 as fully-coherent as their datasets (8 KB) can fit in private caches (64 KB), while the remaining as LLC-coherent, which are unable to fit in private caches but their combined footprint is still less than or equal to the LLC size (2 MB).

Low amount of ALP (Apps1-4). These applications only have a single accelerator running (Apps1-3) or two operating concurrently (App4). For these cases, the cost-aware coherence interface selected

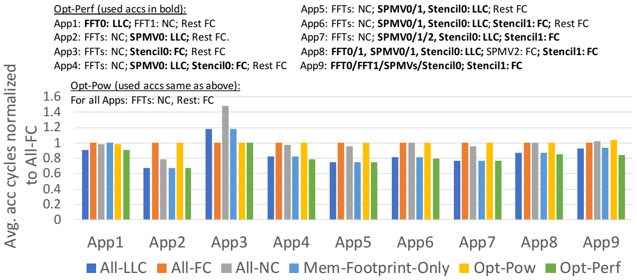


Figure 3: Performance comparison of coherence interfaces

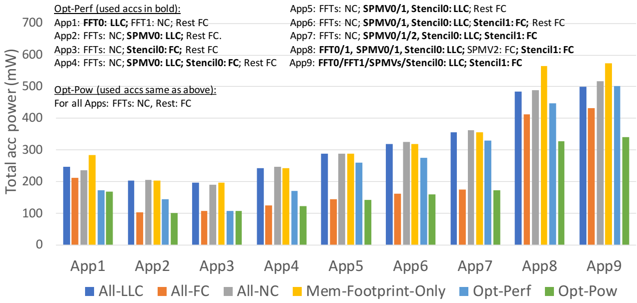


Figure 4: Power comparison of coherence interfaces

by CHIME is only based on the kernel properties as there is no (or minimal) system-level contention for shared resources. The selected Opt-Perf interfaces use the best performing coherence models for the operating accelerator(s) and for the remaining ones, those coherence models are chosen that lead to minimal accelerator memory leakage power (non-coherent for FFTs with only 8 KB scratchpads, and fully-coherent for SPMVs/Stencils with 64 KB caches). Interestingly, Opt-Pow for all applications is the same that minimizes the total accelerator memory power (functional unit power is independent of the coherence models): FFT0/1 non-coherent, SPMV0/1/2 and Stencil0/1 as fully-coherent.

As shown in Figure 3, for Apps1-4, Opt-Perf achieves significant improvements over the four baselines in terms of average accelerator performance: up to 32.5% over All-NC, 32.7% over All-FC, 15.1% over All-LLC, and 15.1% over Mem-Footprint-Only. As expected, Opt-Pow incurs significant performance overheads except for App3, where it performs the same as Opt-Perf and All-FC. In App1, the uniform accesses of the FFT kernel [11] are more efficiently handled using scratchpads than caches; hence LLC coherence is preferred (in All-LLC and Opt-Perf) which also minimizes accesses to the main memory compared to no coherence. For App2, the large dataset for SPMV is unable to fit in small private caches, hence again larger scratchpad-based LLC-coherence performs better (in All-LLC, Mem-Footprint-Only, and Opt-Perf). For App3 with Stencil-3D accelerator operating, full-coherence achieves better performance (in All-FC, Opt-Pow, and Opt-Perf) as it is more suited to the kernel's non-uniform strided access pattern [11]. Following the above reasoning, Opt-Perf for App4 chooses LLC-coherence for SPMV0 and full-coherence for Stencil0, showing the best performance compared to the others.

Figure 4 shows significant improvements in total accelerator power for Opt-Pow: up to 49.9% over All-LLC, 20.6% over All-FC, 50.5% over All-NC, and 49.8% over Mem-Footprint-Only. Interestingly, Opt-Perf also achieves up to 45.8% improvements over All-NC, All-FC, and Mem-Footprint-Only, while showing overheads over All-FC and Opt-Pow. Opt-Pow selects those coherence models that lead to minimal accelerator memory power (NC for FFTs, and FC for the rest). Opt-Perf also uses the low memory power coherence models for the idle accelerators, and since a majority are not being used, it achieves considerable improvements.

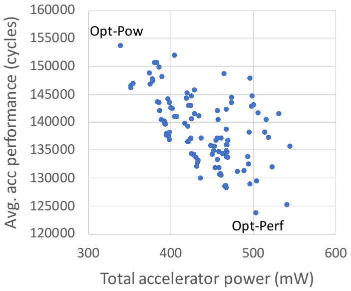


Figure 5: Two cost-aware coherence interfaces for App9

High amount of ALP (Apps5-9). Figure 3 shows the average accelerator performance of different coherence interfaces for these applications that have 3-7 accelerators running concurrently. Opt-Perf interfaces for these applications are selected taking into account both the kernel characteristics (data access patterns/memory footprint) and also system properties (e.g., contention in LLC/NoC). Accelerators that are not operating simply use the coherence models that lead to minimal memory leakage power. For Apps 5-9, Opt-Perf achieves significant performance improvements over the four baselines: up to 9.4% over All-LLC, 23.2% over All-FC, 21.6% over All-NC, and 10.4% over Mem-Footprint-Only. Overall, Opt-Perfs use LLC coherence for SPMVs/FFTs, and a mix of LLC and full coherence for Stencil-3Ds to achieve the best performance for these applications. Detailed analysis for App9 in the next section shows that such a mix of coherence models leads to smaller data transfer cycles for the accelerators under contention.

Figure 4 shows the total accelerator power of different coherence interfaces for Apps 5-9. Opt-Pow, which is the same as for Apps 1-4, achieves the lowest power with significant improvements over the baselines: up 51.3% over All-LLC, 21.2% over All-FC, 51% over All-NC, and 51.2% over Mem-Footprint-Only. Opt-Perf also shows up to 15.9% lower power than All-LLC, All-NC, and Mem-Footprint-Only but incurs overheads compared to All-FC.

Detailed analysis for App9. Details on the convergence of CHIME to Pareto-optimal coherence interfaces are now presented for the application with all 7 accelerators running concurrently. Additionally, while the above results showed performance gains obtained by Opt-Perf only in terms of average accelerator performance, this section now presents improvements achieved by Opt-Perf in terms of individual accelerator performance for App9.

Pareto-optimal analysis for App9. Figure 5 shows the Pareto-optimal coherence interfaces obtained for App9 using the CHIME framework, from which we selected the Opt-Perf and Opt-Pow interfaces. In this case, CHIME started with 10 random samples in the initialization phase, followed by 60 iterations of the optimization phase to achieve convergence. The framework is found to be highly-efficient as only a total of 70 gem5-Aladdin evaluations were performed to achieve these Pareto-optimal solutions, while the complete design space comprises of 2187 configurations (3 possible coherence choices for each of the 7 accelerators). Each iteration of the framework took 70 minutes: mostly dominated by the gem5-Aladdin run, while BayesOpt only took 20 seconds to select the next coherence interface for evaluation.

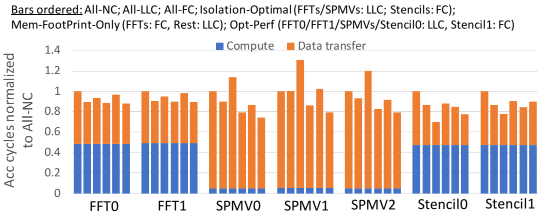


Figure 6: Performance breakdown for App9

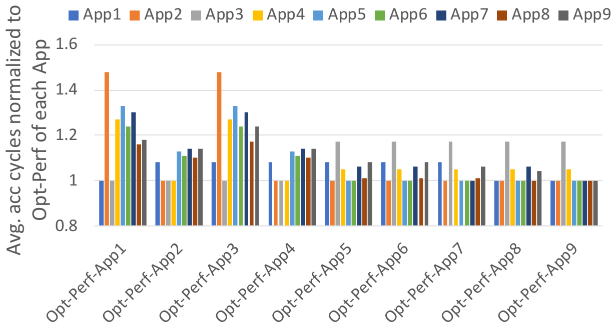


Figure 7: Overhead analysis when using Opt-Perf of each application for all the other applications

Performance breakdown for App9. Figure 6 shows the performance for each of the accelerators, comparing the Opt-Perf cost-aware coherence interface with the previously considered 4 baseline interfaces as well as one additional baseline called *Isolation-Optimal*. The Isolation-Optimal interface assigns those coherence models to the accelerators that lead to best performance when these accelerators are run in isolation without any contention from the others (e.g., Apps 1-3). Based on the analysis of Apps 1-3 in Figure 3, Isolation-Optimal will use LLC coherence for the FFTs and SPMVs, and full coherence for the Stencils. The comparison of Opt-Perf with this baseline shows that the impact of contention can be significant on the performance of the accelerators.

Compared to the 5 baselines, Opt-Perf showed significant performance improvements for most of the accelerators. It achieved up to 25.6% better performance than All-NC (for SPMV0), up to 21.2% better than All-LLC (for SPMV1), up to 39.3% better than All-FC (for SPMV1 but with small overheads for the Stencils), up to 11.9% better than Isolation-Optimal (for Stencil0), and up to 14.3% better than Mem-Footprint-Only (for SPMV0 but with minor overhead for Stencil1). On average also, Opt-Perf achieved considerable performance benefits. The reason is while it selects those coherence models for most of the accelerators that favor their kernel properties (LLC coherence for SPMVs and FFTs, full coherence for Stencil1), for Stencil0, it uses a model that is not suited to its characteristics but eases overall system contention. The net effect is significant reductions in data transfer times, especially for SPMVs and FFTs.

Selecting a single coherence interface for the SoC. If the optimization target is high performance, Figure 7 shows the overheads in average accelerator performance when using Opt-Perf of each application for all the other applications. The Opt-Perf interface corresponding to App9 (Stencil1: FC, Rest: LLC) is selected as the final interface for the SoC as it has the least average performance overhead of 2.4%. Alternatively, if the target is low power then the overhead analysis is not required as all of the 9 applications have the same Opt-Pow interface (FFTs: NC, and the rest FC), which is then selected as the final coherence interface.

7 CONCLUSION

This paper proposes a comprehensive methodology, called CHIME, to efficiently determine cost-aware coherence interfaces for many-accelerator SoCs. The approach uses Bayesian optimization with a new version of gem5-Aladdin that now supports LLC coherence for accelerators. For a complex SoC and its target 9 different applications, CHIME finds cost-aware coherence interfaces that show significant performance and power benefits over several baseline interfaces. CHIME will be made public and can be used to determine optimal coherence models for different SoCs and applications, and in combination with tuning other architectural parameters.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. Government, under the DARPA DSSoC program. This work was also supported in part by the Semiconductor Research Corporation, NSF grant # CNS-1718160, and Intel. We are also thankful to Michael Kishinevsky, Pietro Mercati, and Raid Ayoub at Intel Labs, Hillsboro, OR, for their helpful feedback on this research.

REFERENCES

[1] D. D. Sarma and G. Venkataramanan. Compute and redundancy solution for Tesla's full self driving computer. *Hot Chips*, 2019.

[2] Y. Chen et al. Accelerator-rich CMPs: from concept to real hardware. In *ICCD*, pages 169–176, 2013.

[3] E. G. Cota et al. An analysis of accelerator coupling in heterogeneous architectures. In *DAC*, pages 202:1–202:6, 2015.

[4] D. Giri et al. Accelerators and coherence: an SoC perspective. *IEEE Micro*, 38(6): 36–45, 2018.

[5] Arm. ARM Cortex-A72 MPCore processor reference manual.

[6] B. Wile. Coherent accelerator processor interface (CAPI) for POWER8 systems. *https://www.alpha-data.com/pdfs/capi_wp_29sept2014_pub.pdf*, 2014.

[7] Ashley Stevens. Introduction to AMBA® 4 ACE and big.little processing technology. *ARM White Paper, CoreLink Intelligent System IP*, 2011.

[8] J. Cong et al. On-chip interconnection network for accelerator-rich architectures. In *DAC*, pages 8:1–8:6, 2015.

[9] H. Framke et al. Introduction to the wire-speed processor and architecture. In *IBM J. of Research and Development*, 2010.

[10] V. J. Reddi and M. Hill. Accelerator-level parallelism (ALP). *ACM SiGARCH Computer Architecture Today*, 2019.

[11] Y. S. Shao et al. Co-designing accelerators and SoC interfaces using gem5-Aladdin. In *MICRO*, pages 48:1–48:12, 2016.

[12] C. Tan et al. Stitch: fusible heterogeneous accelerators enmeshed with many-core architecture for wearables. In *ISCA*, pages 575–587, 2018.

[13] D. Giri et al. NoC-based support of heterogeneous cache-coherence models for accelerators. In *NOCS*, pages 1:1–1:8, 2018.

[14] J. Alsop, M. D. Sinclair, and S. V. Adve. Spandex: a flexible interface for efficient heterogeneous coherence. In *ISCA*, pages 261–274, 2018.

[15] K. Bhardwaj et al. Determining optimal coherency interface for many-accelerator SoCs using Bayesian optimization. *IEEE CAL*, 18:119–123, 2019.

[16] D. Giri et al. Runtime reconfigurable memory hierarchy in embedded scalable platforms (invited). In *ASP-DAC*, pages 719–726, 2019.

[17] N. L. Binkert et al. The gem5 simulator. *SIGARCH Computer Architecture News*, pages 1–7, 2011.

[18] Y. S. Shao et al. Aladdin: a pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ISCA*, pages 97–108, 2014.

[19] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *NIPS*, pages 2960–2968, 2012.

[20] B. Shahriari et al. Taking the human out of the loop: a review of Bayesian optimization. *Proceedings of the IEEE*, pages 148–175, 2016.

[21] C. Rasmussen and C. Williams. Gaussian processes for machine learning. *MIT press, Cambridge, MA*, 2005.

[22] W. Ponweiser et al. Multiobjective optimization on a limited budget of evaluations using model-assisted S-Metric selection. In *PPSN*, pages 784–794, 2008.

[23] B. Reagen et al. Machsuite: benchmarks for accelerator design and customized architectures. In *IISWC*, pages 110–119, 2014.

[24] J. Ragan-Kelley et al. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *PLDI*, pages 519–530, 2013.