HAL
open science

Design Space Exploration of Heterogeneous-Accelerator SoCs with Hyperparameter Optimization

Thanh Cong, François Charot

To cite this version:

Thanh Cong, François Charot. Design Space Exploration of Heterogeneous-Accelerator SoCs with Hyperparameter Optimization. ASP-DAC 2021 - 26th Asia and South Pacific Design Automation Conference, Jan 2021, Virtual Conference, Japan. pp.1-6. hal-03119732

HAL Id: hal-03119732

https://inria.hal.science/hal-03119732v1

Submitted on 26 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Design Space Exploration of Heterogeneous-Accelerator SoCs with Hyperparameter Optimization

Thanh Cong
minh-thanh.cong@irisa.fr
Univ Rennes, INRIA, CNRS, IRISA
Rennes, France

François Charot
francois.charot@inria.fr
INRIA, Univ Rennes, CNRS, IRISA
Rennes, France

ABSTRACT

Modern SoC systems consist of general-purpose processor cores augmented with large numbers of specialized accelerators. Building such systems requires a design flow allowing the design space to be explored at the system level with an appropriate strategy. In this paper, we describe a methodology allowing to explore the design space of power-performance heterogeneous SoCs by combining an architecture simulator (gem5-Aladdin) and a hyperparameter optimization method (Hyperopt). This methodology allows different types of parallelism with loop unrolling strategies and memory coherency interfaces to be swept. The flow has been applied to a convolutional neural network algorithm. We show that the most energy efficient architecture achieves a 2x to 4x improvement in energy-delay-product compared to an architecture without parallelism. Furthermore, the obtained solution is more efficient than commonly implemented architectures (Systolic, 2D-mapping, and Tiling). We also applied the methodology to find the optimal architecture including its coherency interface for a complex SoC made up of six accelerated-workloads. We show that a hybrid interface appears to be the most efficient; it reaches 22% and 12% improvement in energy-delay-product compared to just only using non-coherent and only LLC-coherent models, respectively.

KEYWORDS

Heterogeneous architecture design, System-on-chip, Hardware accelerators, Hyperparmeter optimization, Simulation

1 INTRODUCTION

The energy efficiency gap between application-specific integrated circuits (ASICs) and general-purpose processors motivates the design of heterogeneous-accelerator system-on-chip (SoC) architectures, the latter have received increasing interest in recent years [22]. To support several heavy demanding workloads simultaneously, and reduce unpowered silicon area, computer architects design many special-purpose on-chip accelerators implemented in ASIC and share them among multiple processor cores. Such architectures offer much better performance and lower power compared to performing the same task on a general-purpose CPU. Designing heterogeneous-accelerator SoCs is extremely expensive and time-consuming. The designer has to face many design issues such as the choice of the parallelism degree and the resource utilization of accelerators, their interfaces with the memory hierarchy, etc. Design space exploration methodologies are of major importance.

In this paper, we present a methodology for designing modern SoC architectures which combine many specialized hardware accelerators and processor cores. We explore the design space of power-performance accelerator-based systems with a SoC simulator and determine the optimal configuration using a hyperparameter optimization algorithm. The proposed simulation infrastructure is based on the use of two tools: gem5-Aladdin [24] and Hyperopt [1]. gem5-Aladdin is an architectural simulator that supports the modeling of complex systems made up of heterogeneous accelerators. Hyperopt is a library implementing different hyperparameter optimization algorithms for solving optimization problems with an unknown objective function [21], such as architecture simulation in our case. The main contributions of this work are as follows.

- A framework for determining, at the system-level, the microarchitecture with the best efficiency, in terms of performance-power ratio.
- A case study allowed us identifying the most energy efficient architecture for a convolutional neural network (CNN). We showed that the solution obtained achieves a 2x to 4x improvement in energy-delay-product (EDP) compared to an architecture without parallelism. Furthermore this solution is more efficient than commonly implemented architectures (Systolic, 2D-mapping, and Tiling).
- To demonstrate the efficiency of the heterogeneous-accelerator SoC design approach, we determined the optimal architecture including its coherency interface for a complex SoC made up of six common accelerated-workloads. Three possible coherency models are considered: a software-managed direct memory access (DMA), a shared last level cache (LLC-coherent) and a fully-coherent cache. Our framework allowed to determine that a hybrid interface appears to be the most efficient; it reaches 22% and 12% improvement in EDP compared to just only using non-coherent and only LLC-coherent models, respectively.

2 RELATED WORK

Simulation platforms adapted to accelerator-centric architectures are proposed in PARADE [7] and gem5-Aladdin [24]. Both provide simulation platforms enabling the exploration of many-accelerator designs. PARADE is a simulation platform that can be automatically generated through a high-level synthesis (HLS) description of the accelerator. Unlike PARADE, gem5-Aladdin models accelerators based on a dataflow representation extracted from the profiling of the dynamic execution of the program, enabling fast design space exploration. The gem5 simulator [4] is then used for the simulation. With such approaches, the designer is faced with the problem of compromise between accuracy and speed of the simulation. To speed up the simulation, there exist approaches where performance models are deployed on FPGA-based platforms [8].

There are several projects whose goal is to be able to rapidly evaluate accelerator-based hardware design. Embedded Scalable

Platforms (ESP) [19] uses HLS to design accelerator SoCs. Cosmos [18] leverages both HLS and memory optimization tools to improve exploration of the accelerator design space. Centrifuge [12] is able to generate and evaluate heterogeneous accelerators SoCs by combining HLS with FireSim [13], a FPGA-accelerated simulation platform. All these works provide design frameworks for evaluating accelerators and it is up to the user to select the optimal one. Unlike these projects, we aim to provide a unified framework for the design, simulation and optimization of the architecture of accelerator-based SoCs.

Machine learning quickly became a powerful tool in computer architecture, with established applicability to design, optimization, simulation, etc., as attested by the survey proposed by Penny et al. [17]. These techniques offer interesting opportunities for architecture simulation, especially in the early stages of the design process. Bhardwaj et al. present in [3] a Bayesian optimization-based framework for determining the optimal hybrid coherency interface for many-accelerator SoCs in terms of performance.

3 DESIGN APPROACH OVERVIEW

3.1 Heterogeneous-Accelerator SoC

Figure 1 shows the organization of a typical heterogeneous-accelerator SoC architecture. It includes a number of processor cores and many specialized accelerators. Each accelerator is made up of several dedicated datapaths that implement parts or all of an algorithm of a specific application domain. Each accelerator has a local memory (scratchpad memory or private cache) to speed up data transfer. The architecture also includes DMA, last level cache controller, and coherent memory controllers shared by both processor cores and accelerators. At the system-level, a customized network-on-chip enables the communications between these different components.

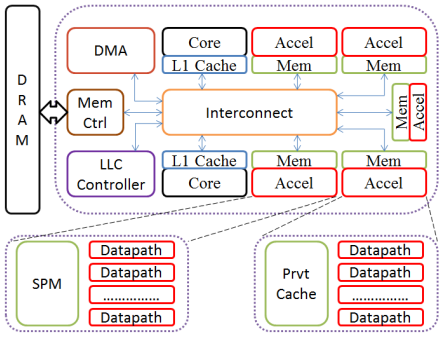

Figure 1: View of a heterogeneous-accelerator SoC.

3.2 Accelerator Modeling and SoC Simulation

The Aladdin trace-based accelerator simulator [23] profiles the dynamic execution trace of an accelerated workload initially expressed in C and estimates its performance, power, and area. The process mainly consists in building a dynamic data dependence graph (DDDG) of the workload which can be viewed as a data flow representation of the accelerator. This graph is then scheduled taking into account the resource constraints by the way of user-defined hardware parameters such as loop unrolling, loop pipelining, and number of memory ports. The underlying model of the Aladdin simulator is a standalone datapath and its local memories.

The interactions at the SoC view level, that is to say between the accelerators and the other components of the system, are managed by gem5-Aladdin [24], which realizes the coupling of Aladdin with the gem5 architectural simulator [4]. gem5-Aladdin is capable of evaluating interactions between accelerators and processor cores, DMAs, caches, virtual memory, in SoC architectures such as the one illustrated in Figure 1. gem5-Aladdin supports three coherency models for accelerators. (i) non-coherent: using software-managed DMAs (ii) LLC-coherent: by directly accessing the coherent data in the last-level-cache (LLC) without having a private cache; (iii) fully-coherent caches: each accelerator can use its private cache to access the main memory.

3.3 Hyperparameter Optimization Method

Performing design space exploration manually leads to inefficient and time-consuming processes. Approaches based on hyperparameter optimization prove to be very effective in optimizing unknown objective functions as stated in works presented in [3, 21]; they are more powerful than heuristic optimization in terms of convergence and quality of obtained solutions.

There are several approaches to hyperparameter optimization. Bayesian hyperparameter optimization (also known as sequential model-based optimization, SMBO) is used here. The approach is to build a probability model of the objective function which is used to select the most promising hyperparameters to evaluate in the true objective function. Several variants of SMBO methods exist in practice, based on Gaussian Processes, Random Forests Regressions and Tree Parzen Estimators (TPE). Hyperopt package implements TPE [2]. One of the main advances of TPE over other probabilistic methods is that it is able to retain dependencies between parameters as it models the density of a parameter for "good" experiments and compares this to its density for "bad" experiments. It can then use these models to determine an expected improvement of the objective function for any values a parameter can take. $P(x \mid y)$, which is the probability of the hyperparameters given the score on the objective function, is expressed as:

$$P(x \mid y) = \begin{cases} l(x) & \text{if } y < y* \\ g(x) & \text{if } y \geq y* \end{cases} \tag{1}$$

where $y < y*$ represents a lower value of the objective function than the threshold. As equation 1 shows, two different distributions for the hyperparameters are calculated: one where the value of the objective function is less than the threshold, $l(x)$, and one where the value of the objective function is greater than the threshold, $g(x)$. Once $l(x)$ and $g(x)$ have been expressed, TPE is able to identify the next parameter x_{next} considering that $x_{next} = argmin_x \frac{g(x)}{l(x)}$.

TPE builds a search history and predicts at each iteration the best trial to try next. TPE itself has many parameters that can be tuned to improve the effectiveness of the TPE algorithm. Adaptive-TPE (ATPE), a modified version of TPE, uses a pre-trained machine-learning model to help optimize faster and more accurately.

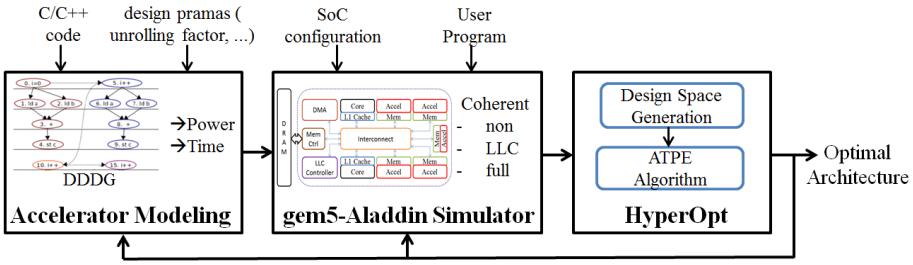

Figure 2: Overview of our generic design flow using the hyperparameter optimization-based method.

4 DESIGNING SOC WITH THIS FLOW

4.1 Parallel Accelerator Exploration

To improve the performance of applications, computation-intensive parts, typically loops, are mapped to hardware accelerators. Loop nests of the considered workloads define the exploration space, as illustrated in Figure 3, with the convolutional layer of a Convolutional Neural Network (CNN) application. This layer exhibits intensive parallelism at the feature map, neuron, and kernel levels. There are four parameters: M (number of output feature maps), N (number of input feature maps), S (output feature map size, or number of neurons), and K (kernel size). These 6 nested-loops offer an interesting exploration space as it is possible to play with different loop unrolling factors.

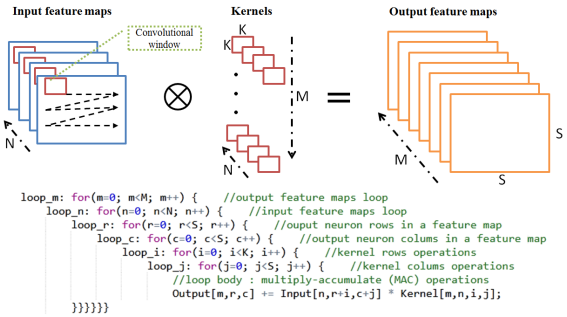

Figure 3: Convolutional layer operation of a CNN.

4.2 Memory Coherency Models Exploration

Giri et al. [11] identified three common coherency interfaces used to integrate accelerators with the memory hierarchy in a loosely-coupled architecture.

- In a non-coherent interfacing model, the accelerator has a scratchpad memory (SPM) for local storage and uses DMA to load data from DRAM, as illustrated in Figure 1.
- LLC-coherent accelerators send DMA requests to the LLC. Their implementation is similar to non-coherent accelerators, but the LLC-coherent DMA requests/responses are routed to the cache-coherent module instead of the DMA.
- In a fully-coherent model, each accelerator has its private cache which implements a cache coherence protocol such as MESI or MOESI, similar to a processor's cache.

Each of these three coherency models offers interesting power-performance trade-offs. In a SoC integrating several accelerators to support a versatile application, a single coherency interface used by all accelerators may not be the most optimal in terms of power and performance as shown by Giri et al. [11]. Each accelerator may have its own coherency model, and this is what we explore here.

4.3 Hyperopt-gem5-Aladdin Framework

As shown in Figure 2, the parallelism of an accelerator is set by design pragma directives such as the loop unrolling factor during the accelerator modeling phase. This phase is used to evaluate and update the power-performance of accelerators.

The gem5-Aladdin simulator is able to model SoCs including several accelerators that can use different coherency models, and run various workloads concurrently. The complete SoC is specified using a SoC configuration file which describes the configuration of processors, accelerators, memories/caches, and interconnect. The gem5-Aladdin simulator is an objective function of the optimization method; it provides performance, power, delay time of SoC architectures that we want to optimize.

Algorithm 1: Hyperopt-based method Pseudo-Code

1 Define architecture search spaces (coherent or non-coherent cache, loop_m, loop_n, loop_r, loop_c, loop_i, loop_j);
2 Randomly simulate k architecture configurations;
3 Define initial *search history* with k pairs (x, EDP_x);
4 **while** $n \leq (N - k)$ **do**
5 Construct models density functions $g(x), l(x)$;
6 $x_{next} = argmin_x \frac{g(x)}{l(x)}$;
7 Simulate x_{next};
8 Update *search history* $\leftarrow (x_{next}, EDP_{next})$;
9 $n++$;
10 Return the best $(configuration, minimumEDP)$;

Algorithm 1 presents the pseudo-code of the method used in Hyperopt. The algorithm starts by randomly selecting k architecture configurations and simulates them with the gem5-Aladdin simulator (line 2). The *search history* is initiated (line 3), it consists of k pairs of configuration and its associated EDP. The next steps of the method are iterative and are performed $N - k$ times, where N is the budget on the number of architecture simulations. The search space

is narrowed down from the search history and a new configuration for the next simulation step is suggested using equations presented in Section 3.3 (lines 4, 5, 6, 7, 8, 9). Once all the iterations have been completed, the optimal architecture configuration set which reaches the minimum EDP is selected (line 10).

5 EXPERIMENTS

To show the effectiveness of our design approach, we present two experiments: the first one concerns the design of CNN accelerators and the second one the design of a SoC including six accelerator tiles. The SoC configuration set up of the gem5-Aladdin simulator of the two experiments is given in Table 1.

Table 1: gem5-Aladdin SoC Architecture Configuration

Component	Description
CPU Type	Out-of-order X86
System Clock	100MHz
Cache Line Size	64 bits
L2 Cache (LLC)	2 MB, 16-way, LRU
Memory	DDR3_1600_8x8, 4 GB
Hardware Prefetchers	Strided
Data Transfer Mechanism	DMA/Cache

5.1 CNN Accelerator in a SoC

As discussed in Section 4.1, CNN layers are highly computation intensive and exhibit fine-grained parallelism at feature map (FP), neuron (NP), and synapse (SP) levels. This potential parallelism offers many opportunities to speed up the calculations. However, most of existing CONV accelerators exploit the parallelism only at one level [16]. Systolic architectures can only exploit synapse parallelism [5], 2D-Mapping architectures neuron parallelism [10], and Tiling architectures feature map parallelism [6]. There is a lack of architectural studies trying to exploit these different types of fine-grained parallelism simultaneously. By exploring all possible types of parallelism, and depending on user constraints, greater efficiency can be expected.

The calculations of a CONV layer, as shown with the code in Figure 3, can be unrolled in different ways. The labels in the code ($loop_m, loop_n, loop_r, loop_c, loop_i, loop_j$) are used to set the unrolling factors and quantify the parallelism degree of each loop. According to the different unrolling strategies of the loops, there are three types of parallelism.

- Feature map Parallelism (FP), $loop_m$ output feature maps, and $loop_n$ input feature maps are processed at a time (maximum factors are M and N respectively).
- Neuron Parallelism (NP), $loop_r$, and $loop_c$ neurons of one output feature map are processed at a time (maximum factor is S).
- Synapse Parallelism (SP), $loop_i$, and $loop_j$ synapses of one kernel are computed at a time (maximum factor is K).

The design space is built by combining these three types of parallelism. As an example, an architecture may handle a single input feature map and a single output feature map ($loop_m = 1$ and $loop_n = 1$), one neuron of each output feature map ($loop_r = 1$ and $loop_c = 1$), but multiple synapses of each kernel at a time ($loop_i > 1$ or $loop_j > 1$). This style of parallel computing is named Single Feature map, Single Neuron, Multiple Synapses (SFSNMS). It is obviously possible to define other processing styles: SFSNSS, SFMNSS, SFMNMS, MFSNSS, MFSNMS, MFMNSS and MFMNMS [16].

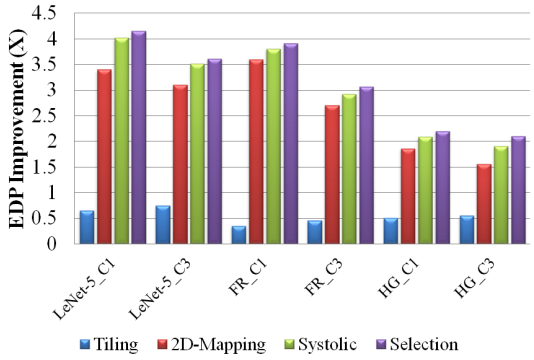

Figure 4: EDP improvement for CNN workloads.

Table 2: Unrolling factors for CNN-Workloads(M,N,K,S) (loop_m,loop_n,loop_r,loop_c,loop_i,loop_j)

Workloads	Systolic	2Dmapping	Tiling	Selection
LN5_C1(6,1,5,28)	1,1,1,1,5,5	1,1,28,28,1,1	6,1,1,1,1,1	1,1,15,15,5,5
LN5_C3(16,6,5,10)	1,1,1,1,5,5	1,1,10,10,1,1	16,6,1,1,1,1	2,2,7,7,5,5
FR_C1(4,1,5,28)	1,1,1,1,5,5	1,1,28,28,1,1	4,1,1,1,1,1	1,1,15,15,5,5
FR_C3(16,4,4,10)	1,1,1,1,4,4	1,1,10,10,1,1	16,4,1,1,1,1	1,1,10,10,4,4
HG_C1(6,1,5,24)	1,1,1,1,5,5	1,1,24,24,1,1	6,1,1,1,1,1	1,1,16,16,5,5
HG_C3(12,6,4,8)	1,1,1,1,4,4	1,1,8,8,1,1	12,6,1,1,1,1	1,1,7,7,4,4

We evaluated three common workloads. LeNet-5 [14], the most famous handwriting recognition model, FR[9] implementing a face recognition model and HG [15] used to recognize hand gestures of human. In this experiment, we used a non-coherent interface model, it has a private scratchpad memory for local storage and uses DMA to request data from the main memory.

Table 2 gives the configuration of three well-known parallel architectures (tiling, 2D-mapping and systolic) for each of the considered workloads. The fourth architecture, called *selection*, corresponds to that resulting from our exploration.

Figure 4 shows the EDP results for the six workloads. The horizontal axis denotes the workloads and the vertical axis denotes EDP value normalized by EDP of a baseline architecture without any parallelism whose parameters are (1,1,1,1,1,1). The columns in one benchmark represent the normalized EDP of the different architectures.

The different EDP improvements of these architectures, illustrated in Figure 4, can be explained by two main reasons: data reuse and use of computing resources. Systolic and 2D-Mapping architectures have a comparable improvement in terms of energy. Systolic has a higher latency than 2D-Mapping because of the long initialization phase to fill the chain of processing elements. But systolic has a higher data reuse factor than 2D-Mapping, therefore systolic consumes less energy than 2D-Mapping for most workloads. At the SoC level, most of the energy is consumed by the data

movement, so if data reuse increases, EDP also increases. In the case of tiling, the EDP improvement is very low because of low computing resource utilization. It has the poorest energy efficiency due to high latency and poor data reuse. Our selected configuration combines systolic and 2D-Mapping. This corresponds to configurations having maximal synapse parallelism to increases data reuse, and a high neuron parallelism to balance computing resource utilization and local memory load/store power consumption. In summary, the configuration proposed with our flow allows obtaining a better EDP than usual architectures (Systolic, 2D-mapping and Tiling) for accelerator-based SoCs. This results in an improvement of the EDP by a factor between 2 and 4 compared to a sequential architecture.

5.2 Hyperopt Convergence Study

We studied the convergence of the hyperparameter optimization algorithm and compared three implementations: random search, conventional TPE and ATPE. LeNet-5 workload is used as a case study. The three implementations are executed in the same search space and the convergence results are illustrated in Figure 5. The total number of possible configuration is 840. The simulation of all possible configurations confirmed the solution obtained with our optimization method. As illustrated in Figure 5, the optimal solution is obtained after a small number of iterations, since 40 are sufficient. The EDP improvement values are distributed into four groups. This distribution can be explained by the complexity of the exploration space, since we try to mix three types of parallelism, as mentioned in 5.1. The ATPE algorithm requires around

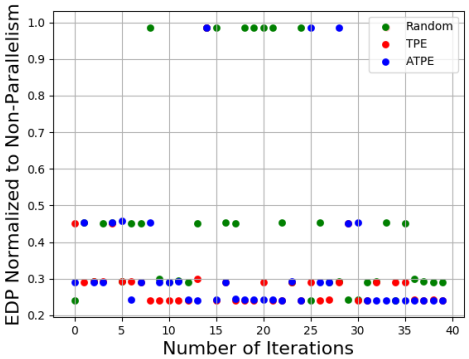

Figure 5: Convergence of the hyperparameter optimization.

30 iterations to converge in the lowest group and achieves the best EDP improvement after 40 iterations. Each iteration requires 30 minutes of CPU time (Intel Xeon E5-2609 at 1.9GHz), considering that gem5-Aladdin represents most of the CPU time. Using this hyperparameter optimization method, and as shown here, we can get a solution faster, which is really useful in the presence of large design spaces.

5.3 Coherency Interface Choice Study

The SoC configuration used for evaluating heterogeneous-accelerator architectures is a tiled architecture consisting of one CPU and six accelerator tiles, along with L2 cache controller and main memory controller tiles. The processing units all perform a different task, which means that all the accelerators operate in parallel. Table 3 gives the features of the accelerated-workloads used for the experiment. Two LeNet-5 convolutional neural network layers perform an image classification task. The others correspond to four benchmarks from MachSuite[20]: AES-256, GEMM-nCubed, FFT-Transpose, and Stencil-3D.

Table 3: Accelerated-workloads in a SoC

Workloads	Description
LeNet5_C1	Convolutional layer (5x5), 32x32 input, 6x28x28 ouput
LeNet5_C3	Convolutional layer (5x5), 28x28 input, 16x10x10 ouput
AES-256	AES encryption 256 bits
GEMM_nCubed	Matrix multiplication, 64x64 input
FFT-Transpose	Fast Fourier transform (512-point)
Stencil-3D	Stencil computation, 32x32x16 input

The goal of this experiment is to determine the best coherency interface for each accelerator separately and for the SoC made up of these six accelerators. The performance of each accelerator is affected not only by its computation time and memory access patterns but also by possible conflicts when accessing shared resources. Consequently, the coherency models adapted to each accelerator are difficult to predict at design time. The input space of hyperparameter is six dimensional due to the six accelerators. Each accelerator interface can be either non-coherent, LLC-coherent or fully-coherent, this results in a total of 729 possible configurations. Figure 6 shows the EDP results for each accelerator and for the six-accelerator version. The horizontal axis denotes the different accelerators and the vertical axis denotes EDP normalized with respect to the non-coherent configuration. The columns in one benchmark represent the normalized EDP of the different interfaces.

In most cases, with the exception of FFT-transpose, the full-coherent interface performs worst due to its significant hardware and performance overheads. In particular, for CONV-accelerators such as LeNet-5_C1 and LeNet-5_C3. They access a large amount of data (kernels, inputs, and outputs), which cannot fit in the L1 caches and can therefore lead to significant cache misses, penalizing the overall latency. FFT-transpose performs better with fully-coherent than with non-coherent because only eight bytes per 512 bytes of data are read per iteration whereas with the DMA system almost all the data must be available before the computation starts. Furthermore, LLC-coherent shows a better EDP than non-coherent since the memory requests are first sent to the LLC, and when the LLC hits, this results in much shorter access latency.

For the six-accelerator version, hybrid selection offers better EDP than systems using a single coherency interface. The solution obtained is the following: LeNet-5_C1 and LeNet-5_C3 use non-coherent while the other accelerators use LLC-coherent interface. This hybrid solution results in an improvement in EDP of 22% and 12% respectively, compared to only non-coherent and LLC-coherent. Although the average (geometric mean) of the EDP improvement over the six accelerators gives the benefit to the only LLC-coherent model, it appears that in the global system view, the hybrid coherent model achieves better EDP improvement. There

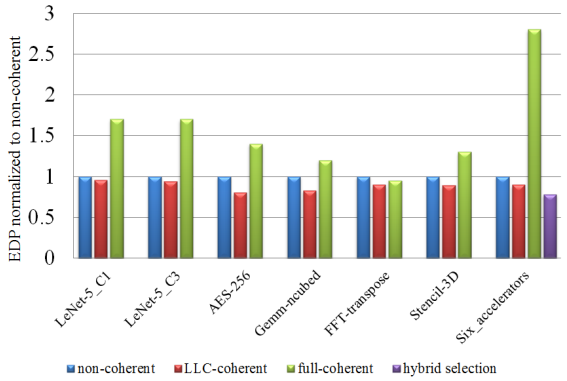

Figure 6: EDP improvement for coherency interface.

are many reasons that explain the EDP improvement brought by the hybrid solution. Having a subset of accelerators with non-coherent interfaces reduces pressure at the LLC level. Indeed, if only four accelerators share the last-level cache, the time spent in data movement compared to an all LLC-coherent solution is reduced. In addition, CONV accelerators benefit from the use of a non-coherent interface with streaming data access patterns applications.

6 CONCLUSION AND FUTURE WORK

In this paper, we described a flow helping in the design of heterogeneous-accelerator SoCs. The flow combines the gem5-Aladdin simulator and a hyperparameter optimization method. It identifies automatically the optimal architecture for heterogeneous-accelerator SoCs. To evaluate our approach, we explored the design space of accelerators for convolutional neural networks including their memory coherency interfaces.

In the near future, we plan to add more parameters to expand our design space (scratchpad partitioning, system bus width, cache size, network on chip, etc.). In addition, we are also interested in measuring the efficiency of the optimization algorithms, and are looking to integrate new algorithms into our framework, in order to compare them.

REFERENCES

[1] J. Bergstra, D. Yamins, and D. D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning* (Atlanta, GA, USA) *(ICML'13)*. 115–123.

[2] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2546–2554.

[3] K. Bhardwaj, M. Havasi, Y. Yao, D. M. Brooks, J. M. H. Lobato, and G. Wei. 2019. Determining Optimal Coherency Interface for Many-Accelerator SoCs Using Bayesian Optimization. *IEEE Computer Architecture Letters* 18, 2 (2019), 119–123.

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[5] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. 2010. A Dynamically Configurable Coprocessor for Convolutional Neural Networks. *SIGARCH Comput. Archit. News* 38, 3 (June 2010), 247–257.

[6] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS'14)*. 269–284.

[7] J. Cong, Z. Fang, M. Gill, and G. Reinman. 2015. PARADE: A cycle-accurate full-system simulation Platform for Accelerator-Rich Architectural Design and Exploration. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 380–387.

[8] T. Cong and F. Charot. 2019. Designing Application-Specific Heterogeneous Architectures from Performance Models. In *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 265–272.

[9] S. A. Dawwd and B. S. Mahmood. 2009. A reconfigurable interconnected filter for face recognition based on convolution neural network. In *2009 4th International Design and Test Workshop (IDT)*. 1–6.

[10] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 92–104.

[11] D. Giri,, P. Mantovani, and L. P. Carloni. 2018. Accelerators and Coherence: A SoC Perspective. *IEEE Micro* 38, 6 (2018), 36–45.

[12] Q. Huang, C. Yarp, S. Karandikar, N. Pemberton, B. Brock, L. Ma, G. Dai, R. Quitt, K. Asanovic, and J. Wawrzynek. 2019. Centrifuge: Evaluating full-system HLS-generated heterogenous-accelerator SoCs using FPGA-Acceleration. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[13] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanovic. 2018. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 29–42.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[15] H. Lin, M. Hsu, and W. Chen. 2014. Human hand gesture recognition using a convolution neural network. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. 1038–1043.

[16] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li. 2017. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 553–564.

[17] Drew Penney and Lizhong Chen. 2019. A Survey of Machine Learning Applied to Computer Architecture Design. *ArXiv* abs/1909.12373 (2019).

[18] Luca Piccolboni, Paolo Mantovani, Giuseppe Di Guglielmo, and Luca Carloni. 2017. COSMOS: Coordination of High-Level Synthesis and Memory Optimization for Hardware Accelerators. *ACM Transactions on Embedded Computing Systems* 16 (09 2017), 1–22.

[19] L. Piccolboni, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. 2017. Broadening the exploration of the accelerator design space in embedded scalable platforms. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7.

[20] B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. Brooks. 2014. MachSuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 110–119.

[21] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (2016), 148–175.

[22] Yakun Sophia Shao and David M. Brooks. 2015. *Research Infrastructures for Hardware Accelerators.* Morgan & Claypool Publishers.

[23] Y. S. Shao, B. Reagen, G. Wei, and D. Brooks. 2015. The Aladdin Approach to Accelerator Design and Modeling. *IEEE Micro* 35, 3 (May 2015), 58–70.

[24] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks. 2016. Co-designing accelerators and SoC interfaces using gem5-Aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12.