A Design Space Exploration Method of SoC Architecture for CNN-based AI Platform

Salita Sombatsiri[1,2], Jaehoon Yu[2], Masanori Hashimoto[2], Yoshinori Takeuchi[3]
[1] Data Science Research Laboratories, NEC Corporation
[2] Graduate School of Information Science and Technology, Osaka University
[3] Graduate school/Electronic Engineering, Kindai University
s-sombatsiri@bp.jp.nec.com, {yu.jaehoon, hasimoto}@ist.osaka-u.ac.jp, takeuchi@ele.kindai.ac.jp

Abstract— This paper proposes a design space exploration (DSE) method for CNN-based AI platform to find SoC architectures that optimally parallelize massive data computation and data transfer. First, the proposed DSE explores both functional blocks, which undertake a process execution, and their parameters, i.e. the number of instances and PEs, to parallelize CNN's intensive intra-process computation with the ease of system modeling and exploration. Second, a multi-layer bus architecture and configuration are optimized to parallelize data transfer by performing master-slave clustering with three-step channel mapping. Experimental result shows that the proposed DSE with pruning technique found 17 Pareto-optimal architectures from the design space of 2 million architectures within 11.5 hours, which is 21% time reduction compared to the exhaustive exploration.

I. Introduction

Designing an edge device for real-time convolutional neural network (CNN)-based application is complicated because of strict requirements. Due to the fact that CNN is computation-intensive, the edge device must be high performance, yet compact and low power. System-on-a-chip (SoC) is a good candidate because it provides power and area efficiency. However, designing an SoC takes long design period to find an optimal architecture.

To shorten the design period, design space exploration (DSE) is studied to search a design space for optimal architectures in the early design stage. Generally, the DSE maps system-level data processing to a pre-designed circuit, called intellectual property (IP), maps data transfer to a bus, optimizes architecture's parameters, and evaluates design qualities [1, 2]. However, for CNN, two problems exist: (1) complexity in finding architectures that parallelize CNN's intra-layer computation; (2) insufficiency in exploring high-performance bus, i.e. multi-layer bus.

The first problem relates to granularity of CNN modeling in system-level. It affects the DSE in the ability to leverage the parallelism within one layer, aka intra-layer parallelism, and the complexity of DSE, which implies the effort for modeling and exploring the design space. A CNN is composed of multiple layers, such as convolutional, pooling and activation layers. In coarse-grained granularity, each process models each CNN layer and is mapped onto an IP. This incurs low complexity, but fails to leverage intra-layer parallelism, which leads to workload imbalance when some layers, e.g. convolutional layer, include more operations than the others, e.g. pooling layer. On the other hand, modeling the convolutional layer in finer granularity, such as one operation as one process (fine-grained), allows intra-layer parallelization at the cost of higher modeling effort and larger design space.

The second problem discloses the difficulty in discovering high-performance communication architecture, aka bus architecture, between IPs. Standard multi-layer bus specification, such as AMBA's multi-layer AHB [3], provides high-performance bus architecture, communication protocol and several configurations for optimization. It is important to model multi-layer bus and find its optimal configuration because bus architecture, configuration and protocol affect the design quality significantly.

This paper proposes a DSE method for designing an SoC for CNN-based application using system-level and IP-based design methodology. It solves multi-objective DSE optimization with traversal through parameter trees and tackles both of the above-mentioned problems with two key contributions: (1) an IP is parameterized in terms of the number of instances and processing elements (PEs). Then, the DSE maps each process to an IP and explores the parameters to exploit intra-layer parallelism by distributing workload with data tiling, while keeping the complexity of DSE low with a coarse-grained CNN model; (2) a multi-layer bus is modeled in terms of bus matrix configuration, and its parameters. Then, the DSE explores multi-layer bus architecture through three-step channel mapping, bus matrix mapping and bus parameter mapping to parallelize data transfer.

II. Related Studies

Heuristic DSE searches the design space for optimal architectures with design space pruning. Matai et al. reuses the designed components for data processing and optimizes parameters like operating frequency and bus width [1]. A DSE based on system-level modeling, maps data processing to an IP, maps data transfer to a bus, and selects their parameters [2]. They are good for optimizing architectures and parameters, but intra-layer computation cannot be parallelized with coarse-grained modeling.

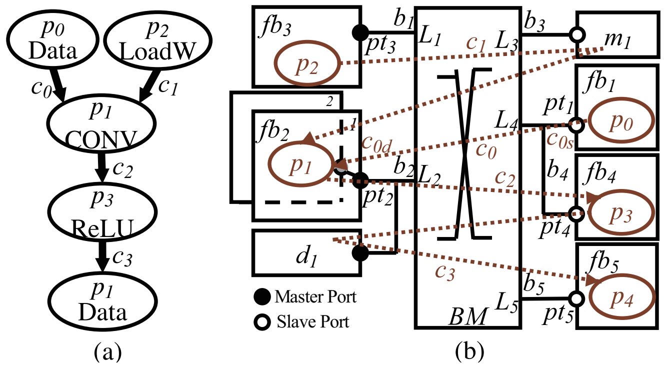

Fig. 1.: Example of models: (a) SLM; (b) ALM.

The DSE for deep learning should consider intra-layer parallelism. DeepBurning analyzes NN model and customizes hardware building blocks in its library to parallelize NN's computation [4]. Tsimpourlas et al. optimize the parameters, such as the number of PEs, of the existing devices for CNN-based applications in terms of performance and power[5]. Both methods can exploit CNN parallelisms, but they do not consider concurrent data transfer using, for example, multi-layer bus.

The multi-layer bus is optimized by master and slave clustering using traffic traces. The method in [6] determines the shared and multi-layer bus architecture by partitioning masters and slaves into clusters. However, it does not consider multiple-master cluster, bus protocol and parameters. This paper parallelizes both intra-layer computation and data transfer by optimizing the number of functional block instances and PEs, and exploring multi-layer bus configuration, direct memory access controller (DMAC), memory and parameters.

III. Model Definitions

This paper employs two models: a model of computation (MoC) describes the application in system-level and an architectural model describes the SoC architecture.

A. *Model of computation (MoC)*

This paper employs a **system-level model (SLM)** $M_{sl} = (P, C)$ in [7] as an MoC. A process set $P = \{p_i | i = 0, 1, 2, ...\}$ and a channel set $C = \{c_j | j = 0, 1, 2, ...\}$ describes data processing and data transfer, respectively. To describe a CNN, this paper extends the SLM by describing a process with $p_i = (S_{p_i})$, where S_{p_i} is the CNN's layer specification including the process type, such as weight loading (**LoadW**), convolutional layer (**CONV**), ReLU layer (**ReLU**) and data storing (**Data**), and layer description, such as the size of input, output and weight filters. An channel $c_j = (p_m, p_n, s_j)$ transfers the data of size s_j from the source process p_m to the destination process p_n. Fig. 1(a) shows an example of an SLM of a CNN consisting of five processes written together with process type and four channels represented by the arrows, which shows data flow direction. It is a coarse-grained SLM, so intra-layer and intra-process parallelisms are equivalent.

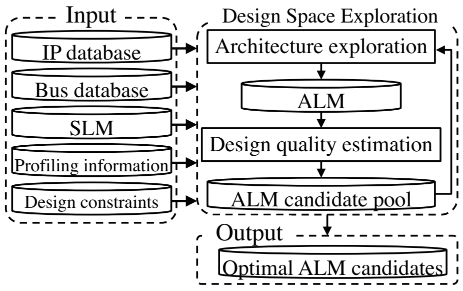

Fig. 2.: Overview of the proposed DSE.

B. *Architectural model*

An **architecture-level model (ALM)** describes the hardware components of an SoC and the mapping with the SLM. To efficiently explore design space for CNN-based AI platform, this paper extends the ALM in [7] by parameterizing the IPs in terms of the number of instances and PEs, which enables the intra-process computation parallelization. An ALM is represented by $M_{al} = (\boldsymbol{F}, \boldsymbol{PT}, \boldsymbol{B}, \boldsymbol{BM}, \boldsymbol{D}, \boldsymbol{M})$ as follows:

- $\boldsymbol{F}$ is a set of functional blocks $fb_i = (I_k, n_{fb_i}, n_{pe_i}, f_{fb_i}, E_{fb_i})$, which is implemented by n_{fb_i} instances of IP I_k and each instance consists of n_{pe_i} PEs. $E_{fb_i} = \{e_{(p_j, fb_i)}\}$ is execution cycle set, where $e_{(p_j, fb_i)}$ represents the execution cycles of the mapped process p_j on fb_i, and f_{fb_i} is the operating frequency.
- $\boldsymbol{PT}$ is a set of master or slave ports $pt_i = (fb_j, b_k, n_q, n_r, pr_{pt_i}, y_{pt_i}, C_{pt_i})$ connects fb_j to a shared bus b_k. The number of receive and transmit buffers are n_q and n_r, respectively. The pr_{pt_i} and y_{pt_i} represents the protocol and cluster of port pt_i. C_{pt_i} refers to a set of channels that port pt_i conducts data transfer.

The set of shared bus, $\boldsymbol{B}$, bus matrix, $\boldsymbol{BM}$, DMAC, $\boldsymbol{D}$, and memories, $\boldsymbol{M}$, are defined as in [7].

Fig. 1(b) shows an example of an ALM, where the processes of the SLM in Fig. 1(a) are mapped onto fb_i. The fb_2 has two instances ($n_{fb_2} = 2$). The source and destination of the channels are mapped to the ports of fb_i that undertake their source and destination processes, e.g. c_0's source and destination (c_{0s} and c_{0d}) are mapped to pt_1 of fb_1 and pt_2 of fb_2, respectively. Bus matrix's master layers (L_1 and L_2) are connected to single- and multiple-master clusters, and slave layers (L_3, L_4 and L_5) are connected to single- and multiple-slave clusters. The type of cluster determines the bus matrix configuration. Memory m_1 and DMAC d_1 are inserted into c_1's and c_3's communication path, respectively, to fulfill the master-to-slave communication regulation of the standard bus [3].

IV. Design Space Exploration Method

Fig. 2 illustrates the proposed DSE. The architecture exploration decides hardware components, e.g. functional blocks and bus, to construct ALMs. Then, the design quality of each ALM is estimated. The ALM is kept in the ALM candidate pool if it is a Pareto-optimal architecture.

The proposed DSE method enables low DSE complexity with high parallelization ability with two points. First, it maps a process (one layer) to a functional block and optimizes the number of instances and PEs for intra-process computation parallelization. Second, it optimizes the multi-layer bus configurations by partitioning masters and slaves into clusters, each of which connects to a bus matrix's layer, through three-step channel mapping.

A. DSE problem formulation

The proposed DSE is formulated as a multi-objective DSE problem. There are five inputs: (1) **IP database** includes gate count, frequency candidates, executable process, execution cycle, and available IP's ports; (2) **Bus database** includes bus protocols, bus width and frequency candidates; (3) **SLM** M_{sl}; (4) **Profiling information** includes processing timings, transfer timings and the amount of transferred data obtained as described in [8]; (5) **Design constraints** include design quality and architectural constraints, such as mapping constraints and maximum resources. The objective functions are (1) **performance function** $T(M_{al})$ is the estimated execution time; (2) **hardware area function** $A(M_{al})$ is the estimated hardware area. The output of the DSE is **Pareto-optimal architectures**, and it includes the ALM (M_{al}), the execution time and the hardware area.

B. Design quality estimation

B.1. Performance estimation

This paper follows the method in [7] to estimate the performance in terms of execution time with four procedures. First, **system-level profiling** gathers profiling information using system-level simulation. Then, **system-level execution dependency graph (SL-EDG) construction** makes an execution dependency graph from the profiling information. Next, **architecture-level execution dependency graph (AL-EDG) construction** makes a graph representing ALM-dependent execution orders. Finally, **AL-EDG analysis** estimates the execution time by analyzing system execution of AL-EDG. The AL-EDG construction and analysis are iterated to evaluate the execution time of various ALMs. In this paper, they consider data tiling to parallelize intra-process computation using multiple functional block instances and PEs.

In **AL-EDG construction**, the vertices of process p_j that executes on multiple instances of functional block fb_i are partitioned into tiles according to n_{fb_i}, n_{pe_i} and S_{p_j}, and then, divided into groups of each functional block instance. The vertices and edges of the processes and channels related to p_j are also partitioned accordingly.

The **AL-EDG analysis** determines a vertex of a data tile to be analyzed on each instance of fb_i, so that vertices of the process undertaken by multiple instances can be analyzed simultaneously, which conforms with intra-process parallelism. Vertex's processing time, t_p, is calculated according to data tiling as follows:

$$t_p = \alpha \times \frac{e_{(p_j, fb_i)}}{f_{fb_i}}, \tag{1}$$

where α is the processing time factor, and it depends on IP and data tiling. The α of the vertices that do not involve data tiling is 1.

B.2. Hardware area estimation

Area of an architecture, $A(M_{al})$, is the summation of areas of all the hardware components in M_{al} as in Eq. 2.

$$A(M_{al}) = \sum_{fb_i \in F} (n_{fb_i} \times A(fb_i)) + \sum_{pt_i \in PT} A(pt_i) + \sum_{d_i \in D} A(d_i) + \sum_{m_i \in M} A(m_i) + \sum_{b_i \in B} A(b_i) + A(BM). \tag{2}$$

The area of functional block fb_i, $A(fb_i)$, which is an instance of IP_j, is estimated as in Eq. 3.

$$A(fb_i) = (g_{ip_j} + g_{pe_j} \times n_{pe_i}) \times A_{nand}, \tag{3}$$

where g_{ip_j} and g_{pe_j} are the gate counts of IP I_j and PE of IP I_j, respectively, and A_{nand} is the NAND gate area. The area of port, $A(pt_i)$, DMAC, $A(d_i)$, and memory, $A(m_i)$, are estimated from the product of the number of buffers/storage blocks and the area of SRAM that can store all the data for one process execution. The area of shared bus, $A(b_i)$, and bus matrix, $A(BM)$, are estimated with the wire area and the bus logic area from the bus area library.

C. Architecture exploration

The proposed DSE explores the design space to construct ALMs using a depth-first traversal through a **parameter set search tree**, which is composed of architecture selection trees and parameter mapping trees by concatenating the root of a tree to the leaves of the preceding tree as shown in Fig. 3. The architecture exploration traverses the architecture selection trees first because the parameters depend on the components. The proposed DSE prunes the tree branches to shorten exploration time.

C.1. Architecture selection trees

The architecture selection trees determine the hardware components and their organization. The proposed DSE explores bus matrix configurations [3] of the bus architecture through channel mapping, and DMAC and memory placement. The order of architecture selection trees comes from the dependencies in organizing the architecture, e.g. channel mapping depends on the functional blocks, so process mapping proceeds before channel mapping.

First, **process mapping tree** maps each process to a functional block of an IP. In Fig. 3, $p_j \rightarrow fb_i(I_k)$ denotes that the process p_j is mapped to functional block fb_i, which is the implementation of IP I_k. Assuming that IP I_0 to I_k can undertake p_0 and p_1 of the SLM in Fig. 1(a). First, p_0 is mapped onto a new functional block fb_1 of IP I_0 (node A) to I_k (node B). Then, from node A, p_1 can be mapped onto the existing fb_1 (node C) or a new functional block fb_2 of I_0 to I_k (node D).

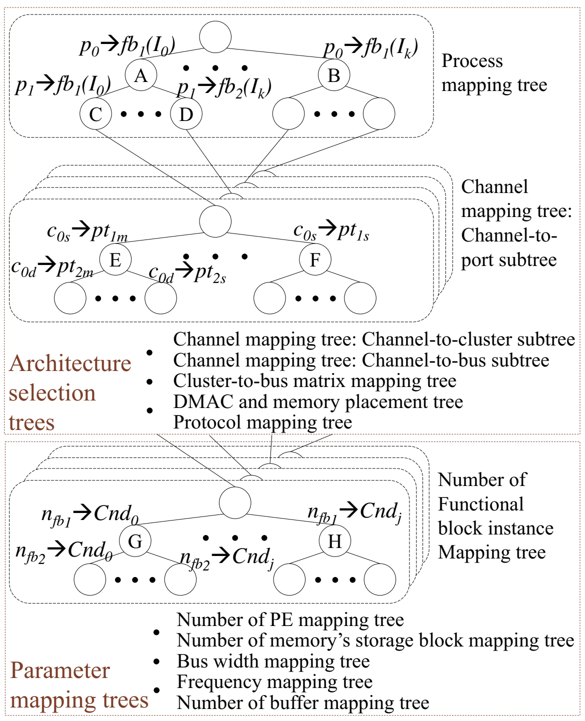
$p_0 \rightarrow fb_1(I_0)$
$p_0 \rightarrow fb_1(I_k)$
$p_1 \rightarrow fb_1(I_0)$
$p_1 \rightarrow fb_2(I_k)$
A
B
C
D
Process mapping tree
$c_{0s} \rightarrow pt_{1m}$
$c_{0s} \rightarrow pt_{1s}$
$c_{0d} \rightarrow pt_{2m}$
$c_{0d} \rightarrow pt_{2s}$
E
F
Channel mapping tree: Channel-to-port subtree
Architecture selection trees
Channel mapping tree: Channel-to-cluster subtree
Channel mapping tree: Channel-to-bus subtree
Cluster-to-bus matrix mapping tree
DMAC and memory placement tree
Protocol mapping tree
$n_{fb1} \rightarrow Cnd_0$
$n_{fb1} \rightarrow Cnd_j$
$n_{fb2} \rightarrow Cnd_0$
$n_{fb2} \rightarrow Cnd_j$
G
H
Number of Functional block instance Mapping tree
Parameter mapping trees
Number of PE mapping tree
Number of memory's storage block mapping tree
Bus width mapping tree
Frequency mapping tree
Number of buffer mapping tree

Fig. 3.: An example of the parameter set search tree.

Second, **channel mapping tree** maps the source and destination of each channel to ports and buses in three steps. First, the **channel-to-port subtree** maps each channel to a port of the mapped functional block. For example, in Fig. 3, the source of channel c_0, represented by c_{0s}, can be mapped to either a master port, pt_{1m} (node E), or slave port, pt_{1s} (node F). Then, the **channel-to-cluster subtree** maps each channel to a cluster, which implies the cluster of each port. Finally, the **channel-to-bus subtree** maps each channel to a shared bus, which implies the bus connected to each port. A channel can be mapped only to a bus in the same cluster or a new bus. The structure of the latter two trees is similar to the channel-to-port subtree.

Third, **cluster-to-bus matrix mapping tree** selects a shared bus from each cluster to connect to bus matrix's master and slave layer. Fourth, **DMAC and memory placement tree** inserts a DMAC or memory into the communication path of the channels to fullfil the master-to-slave communication regulation. Finally, **protocol mapping tree** selects a protocol of the bus matrix, shared buses, and ports.

C.2. Parameter mapping trees

The parameter mapping trees determine the parameters of functional blocks, buses, and memories. The proposed DSE parallelizes intra-process computation by optimizing the number of functional block instances and PEs. To take advantage of pruning the search trees earlier based on hardware area, the DSE traverses the trees that affect area first, and then, the tree of frequency. The number of buffers is explored last because its tree tends to be the highest, which leads to a larger number of nodes.

First, **number of functional block instance mapping tree** selects the number of functional block instances, n_{fb_i}, of each fb_i. In Fig. 3, this tree maps n_{fb_1} to the candidates of each IP, which are Cnd_0 (node G) to Cnd_j (node H). The structure of the other parameter mapping trees are similar to this tree.

Second, **number of PE mapping tree** selects the number of PEs, n_{pe_i}, within each functional block fb_i from the candidates of each IP. Noted that sometimes, n_{pe_i} is configured as different parameters that indicate n_{pe_i} according to the PE organization of the target IPs.

The other parameter mapping trees are as follows: **number of memory's storage block mapping tree** selects the number of storage blocks for each memory inserted during the DMAC and memory placement; **bus width mapping tree** selects a data and address bus width of the bus matrix and shared buses; **frequency mapping tree** selects an operating frequency of each functional block, bus matrix and shared bus; **number of buffer mapping tree** selects the number of buffer, n_q and n_r, in each port in the same way as in [2].

D. *Pruning the parameter set search tree*

Pruning non-optimal architecture earlier results in a shorter exploration time. The pruning eliminates tree branches and all descendants in the parameter mapping trees that do not produce Pareto solutions using the branch and bound algorithm. It takes place when one of the following conditions is met.

- Lower bounds of either execution time or hardware area of the current search node exceed the design constraints or the explored optimal architecture.
- The child nodes that do not yield the smallest hardware area when the lower and upper bound of the execution time are equal.
- A deadlock incurs in the performance estimation.

V. Case Study

The case study shows an IP parameterization and the effectiveness of the proposed DSE in terms of the discovered Pareto-optimal architectures and time for exploration. Here, it should be noted that the design space of the proposed DSE is significantly extended for CNN-based AI platform with parameterized IP and multi-layer bus, and it is incompatible with other methods such as [2]. Therefore, quantitative comparisons are not performed.

A. *Modeling a parameterized IP*

As a case study, we parameterize only the IP for convolutional layers using the parallelism-flexible convolution core [9]. It specifies the PE organization with the number of PEs per group, n_{N_i}, the number of group per PE bank, n_{G_i}, and the number of PE banks, n_{M_i}, so we define $n_{pe_i} = n_{N_i} \times n_{G_i} \times n_{M_i}$ and explore n_{pe_i} of fb_i as these three parameters. The g_{pe_k} is defined as the gate count of IP I_k's basic components including the gate count of a PE bank, g_{M_k}, a PE group, g_{G_k}, and a PE, g_{M_k}.

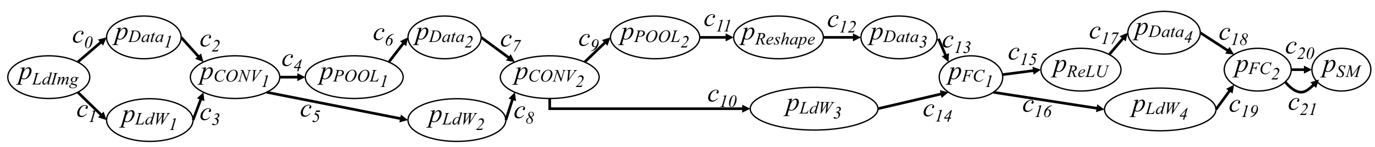

Fig. 4.: An SLM of Lenet-5.

TABLE I: IP database specifies IPs and design constraint specifies (1) $fb_i \in \mathbf{F}$ that are implemented from each IP and mapped processes; (2) $pt_i \in \mathbf{PT}$ (Type M : AHB master, S : AHB slave) of each fb_i together with cluster ID, y_{pt_i}, and mapped channels.

IP database			Design constraint				
IP	Area (gate)	IP Param. (Candidate)	$\boldsymbol{F}$	Mapped process (cycle)	$\boldsymbol{PT}$(Type, n_q,n_r)	y_{pt_i}	Mapped channel
IP_1	3.3k	-	fb_1	$p_{LdImg}(20)$	pt_1(M,0,1)	1	c_{0s}
IP_2	45.5k	-	fb_2	$p_{Data1}(1)$, $p_{DataFC1}(1)$	pt_2(S,1,1)	2	c_{0d}, c_{2s}, c_{12d}, c_{13s}
			fb_3	$p_{Data2}(1)$, $p_{DataFC2}(1)$	pt_3(S,1,1)	3	c_{6d}, c_{7s}, c_{17d}, c_{18s}
IP_3	300	-	fb_4	$p_{LdW1}(20)$, $p_{LdW2}(20)$, $p_{LdW3}(20)$, $p_{LdW4}(20)$	pt_4(S,1,1)	4	c_{3s}, c_{8s}, c_{14s}, c_{19s}
IP_4	g_{ip_4} : 10k	n_{fb_i} (1,2)	fb_5	$p_{CONV1}(330)$,	pt_5(M,2,0)	5	c_{2d}, c_{7d}
	g_{M_4} : 3k	n_{M_i} (8,16)		$p_{CONV2}(150)$	pt_6(M,2,0)	6	c_{3d}, c_{8d}
	g_{G_4} : 500	n_{G_i} (4)			pt_7(M,0,2)	7	c_{4s}, c_{9s}
	g_{N_4} : 30	n_{N_i} (8,16)					
IP_5	2k	-	fb_6	$p_{POOL1}(3)$, $p_{POOL2}(1)$	pt_8(M,1,1)	8	c_{4d}, c_{6s}, c_{9d}, c_{11s}
IP_6	500	-	fb_7	$p_{Reshape}(1)$	pt_9(S,1,1)	9	c_{11d}, c_{12s}
IP_7	3.3k	-	fb_8	$p_{FC1}(16)$, $p_{FC2}(10)$	pt_{10}(M,1,1)	-	c_{13d}, c_{14d}, c_{15s}, c_{18d}, c_{19d}, c_{20s}
IP_8	1.5k	-	fb_9	$p_{RELU}(16)$	pt_{11}(S,1,1)	-	c_{15d}, c_{17s}
IP_9	3k	-	fb_{10}	$p_{SM}(10)$	pt_{12}(S,1,0)	-	c_{20d}

One convolutional layer is modeled as one SLM's process. The AL-EDG construction partitions the AL-EDG vertices of the convolutional layer to handle data tiling according to n_{fb_i}, n_{N_i}, n_{G_i}, n_{M_i}, and S_{p_j}. The processing time of each AL-EDG's process vertex is calculated using Eq. 1 with $\alpha = \frac{1}{P}$, where P is the degree of inter-output parallelism according to [9] and $e_{(p_j, fb_i)}$ is the time for computing all output channels from one input channel of one tile. The candidates of P are $1, 2, 4, 8, 16$.

B. *Target system*

The experiment performs the DSE for LeNet-5 [10], shown in Fig. 4. The process' name also specifies its type. Process p_{LdImg} loads a 28×28-pixel image from an external memory. Process p_{Data_k} writes data to a storage. Process p_{LdW_k} loads the weights of a convolutional and a fully-connected layer. Process p_{CONV_k} is a convolutional layer, and p_{POOL_k} is a max-pooling layer. Process $p_{reshape}$ reshapes the input into a vector. Process p_{FC_k} computes a fully-connected layer, and p_{ReLU} applies ReLU function. Process p_{SM} applies softmax function and outputs ten probability values.

The channels perform inter-process data transfers. Noted that c_1, c_5, c_{10}, c_{16}, and c_{21} are dedicated one-bit signals, each of which indicates that the preceding layer is done, and are omitted in the exploration.

C. *Experimental setting*

The target library is the CMOS 0.18 μm process technology. The A_{nand} is $9.8 \mu m^2$.

The bus database includes a AHB-shared bus and a AHB-multi-layer bus. Table I shows the IP database and design constraint during architecture exploration. The design constraint specifies $fb_i \in \boldsymbol{F}$, $pt_i \in \boldsymbol{PT}$, 256-bit bus width, 200-MHz operating frequency. Here, the experiment focuses on exploring the parameters of IP_4 (fb_5), which is the IP in [9], and y_{pt_i} of pt_{10}, pt_{11}, and pt_{12}.

The experiment was conducted on a 2.3GHz Intel Xeon, 1TB RAM and 64-bit CentOS 6.1 machine. The SLM was implemented with SystemC 2.3.1a. The proposed DSE was implemented with C and compiled with gcc 4.4.7.

D. *Varieties of Pareto-optimal architectures*

Fig. 5 shows a trade-off relationship between execution time and area of 17 Pareto-optimal architectures with different n_{fb_5}, n_{M_5}, n_{G_5}, and n_{N_5} of fb_5. The architectures which contain more PEs (Total PEs $=$ $n_{fb_5} \times n_{M_5} \times n_{G_5} \times n_{N_5}$) consume more area, but execute the LeNet-5 faster because multiple instances and PEs of fb_5 parallelize intra-process computation of the convolutional layers. The architectures containing the same fb_5's parameters have different area and execution time because pt_{10}, pt_{11}, and pt_{12} are in different clusters. Fig. 6 shows two examples of the Pareto-optimal architectures. The master and slave layers of their multi-layer bus are connected to single-master, subsystem, single-slave and multiple-slave clusters, which shows heterogeneous multi-layer bus configuration. The two architectures have different multi-layer bus configurations for parallelizing data. The result shows that the proposed DSE customizes

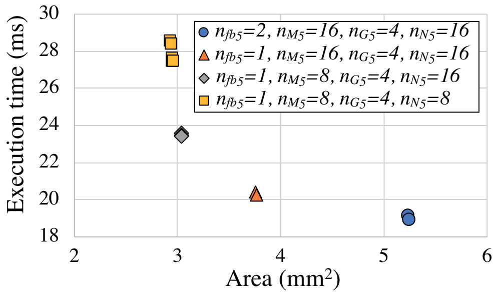

Fig. 5.: Pareto-optimal architectures from the experiment.

the parameters of functional blocks and explores several multi-layer bus configurations, and hence, provides varieties of Pareto-optimal architectures.

E. *Time for exploration*

Table II shows the number of leaves, the number of design quality estimation, and the total runtime of the proposed DSE for exploring the architectures of LeNet-5. The exhaustive exploration traverses every leave in the parameter set search tree and estimates their design qualities, so the number of leaves implies the size of the design space. The exploration with pruning takes shorter time because it discards the branches of the tree that do not yield Pareto-optimal solutions. The number of estimation is more than the number of leaves because additional estimation is required for tree pruning. The result shows 21% (3 hours) of time reduction thanks to pruning. The time reduction will be more significant when exploring a large number of candidates and less design constraints.

The design space becomes larger and the DSE consumes longer time when the design constraints are removed. For example, when the ports are not constrained to any cluster, the design space becomes about 15.7 times larger and the exploration with pruning may take up to 7.6 days. Furthermore, when the AI system grows larger, it may consume weeks or months to explore a larger number of functional blocks and ports, which is barely acceptable in the early design stage. Therefore, the proposed DSE still needs improvements in terms of the time to explore a large design space. The improvements can be achieved by introducing incremental computation for successive architectures with a small difference, more aggressive pruning, and parallel traversal of the parameter set search tree.

VI. Conclusions and Future work

The proposed DSE is suitable for discovering high-performance SoC architecture for CNN-based AI platform with two features: (1) it explores optimal parameters of IPs to leverage intra-process parallelism; (2) it explores optimal configurable multi-layer bus for parallelizing data transfer. The result shows that the proposed DSE discovers varieties of architecture including varieties of functional blocks and multi-layer bus configuration. The future works include energy consumption estimation and additional acceleration techniques of the DSE.

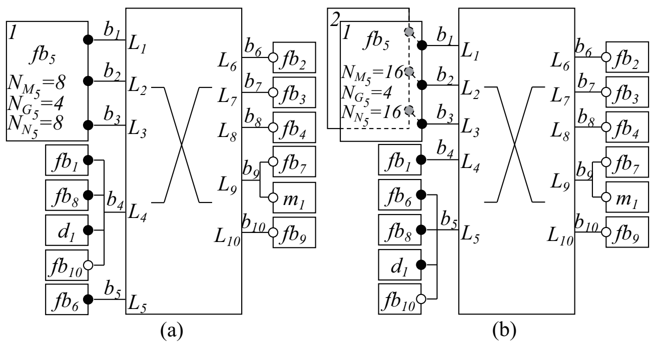

Fig. 6.: Examples of the Pareto-optimal architectures.

TABLE II: Runtime of the proposed DSE

Method	# of leaves	# of est.	Total time
Exhaustive	2,140,577	2,139,768	14.5 hours
Pruning	917,893	1,657,035	11.5 hours

References

[1] J. Matai, D. Lee, A. Althoff, and R. Kastner, "Composable, parameterizable templates for high-level synthesis," *Proc. of DATE2016*, pp. 744–749, 2016.

[2] K. Ueda, K. Sakanushi, N. Yoneoka, Y. Takeuchi, and M. Imai, "Optimal bus architecture exploration for IP-based design," *IPSJ Journal*, vol. 46, no. 6, pp. 1374–1382, 2005.

[3] ARM, "Multi-layer AHB overview," [Online] http://infocenter.arm.com, 2004.

[4] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family," *Proc. of DAC2016*, pp. 1–6, 2016.

[5] F. Tsimpourlas, L. Papadopoulos, A. Bartsokas, and D. Soudris, "A design space exploration framework for convolutional neural networks implemented on edge devices," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2212–2221, 2018.

[6] A. Cilardo, E. Fusella, L. Gallo, and A. Mazzeo, "Exploiting concurrency for the automated synthesis of MPSoC interconnects," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 3, pp. 57:1–57:24, 2015.

[7] S. Sombatsiri, Y. Takeuchi, and M. Imai, "An efficient performance estimation method for configurable multi-layer bus-based SoC," *IPSJ T-SLDM*, vol. 8, pp. 26–37, 2015.

[8] K. Ueda, K. Sakanushi, Y. Takeuchi, and M. Imai, "Architecture-level performance estimation method based on system-level profiling," *IEE P-COMPUT DIG T*, vol. 152, no. 1, pp. 12–19, 2005.

[9] S. Sombatsiri et al., "Parallelism-flexible convolution core for sparse convolutional neural networks on FPGA," *IPSJ T-SLDM*, vol. 12, pp. 22–37, 2019.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.