Pipeline Configuration Methodology for Optimizing Neural Network Accelerators Utilization under a Throughput Constraint

Ali Oudrhiri, Alix Munier

Pipeline Configuration Methodology for Optimizing Neural Network Accelerators Utilization under a Throughput Constraint

Ali Oudrhiri†, Alix Munier†
†Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

***Abstract*—Neural Networks (NNs) have gained widespread popularity, leading to the development of dedicated NN accelerators. They typically optimize key performance indicators (KPIs) such as throughput, power, and chip area, involving trade-offs between these factors. They usually consist of a Neural Processing Unit (NPU) comprising parallel Processing Elements for computations, along with on-chip memories. This paper investigates the feasibility of enhancing processing throughput and achieving improved trade-offs with other KPIs by deploying a pipeline configuration with multiple instances of the NPUs.**

We develop a generic methodology applicable to a wide class of NN accelerators for determining an optimal pipeline architecture and a layers mapping to NPUs to optimize a fixed KPI under a throughput constraint. Our methodology is validated using an industrial accelerator prototype for which close functions for the evaluation of the considered KPIs are available. Our experiments show that the pipeline significantly increases the throughput of the system without degrading the other considered KPIs.

***Index Terms*—Neural networks accelerators, pipeline configuration, throughput enhancement, key performance indicators**

I. Introduction and related work

Neural networks (NNs) have become incredibly popular [1]. Using a NN requires a large amount of calculation; an important research community is working on NN accelerators chips (see the survey of Sze et al. [2]). These chips are designed to optimize multiple key performance indicators (KPIs), with the most significant ones being the throughput, the power, and the chip area. Several projects are focused on designing efficient NN accelerators [2]–[6].

In this paper, we discuss accelerators that are composed of a Neural Processing Unit (NPU) dedicated to computations, along with on-chip memories (SRAMs). The accelerator NPU comprises an array of Processing Elements (PEs) responsible for executing multiplications and accumulations, which have a substantial impact on the overall performance of the NPU.

By increasing parallelization through the addition of more PEs, the processing latency of the NPU (i.e. the number of cycles required to process one image) is improved until a point where further parallelization on PEs does not yield more efficient processing. Moreover, when using a unique NPU, the images are usually processed one by one, which limits the throughput (i.e. the number of images inferred per second). This limitation becomes especially significant for stream applications, where throughput plays a crucial role.

In this paper, we investigate the feasibility of enhancing the throughput by considering several NPUs in a pipeline configuration. We consider as input a feed-forward NN, the description of a configurable NPU and a minimum throughput. We also consider that close functions for the evaluation of the considered KPI and processing time of a layer are available. The problem is to find, for a given throughput, an optimal pipeline architecture (number of NPUs, number of PEs per NPU, size of RAMs) and the mapping of the NN layers optimizing a specific KPI (it could be antagonistic to throughput).

We observe that these multi-criteria optimization problems are particular versions of the basic assembly line balancing problem [7] proved to be NP-hard [8]. We show that our particular case can be polynomially solved using a dynamic programming algorithm inspired by Held et al. [9]. Solving exactly this problem can thus be done without simulations.

Many researchers have explored the pipelining of multiple accelerators to accelerate NNs evaluation and optimize various KPIs [10]–[18]. However, our methodology distinguishes itself by offering several advantages. First, it allows the adaptation of each NPU to the computation needs of layers; this capability is not achievable using fixed heterogeneous machines as in [11]–[13]. Second, our approach enables mapping multiple layers onto a same NPUs resulting in optimized solutions compared to methods assigning each layer to a separate NPU [14], [18]. Furthermore, our approach is compatible with various types of accelerators. Indeed, layers are processed sequentially, leaving the schedule of instructions within the NPUs unmodified and flexible. Our approach thus differs from [10], [15], [16] where specific accelerators are employed, allowing the computation of successive layers to potentially overlap. We illustrate our methodology using Gemini-1 [19], for which closed-formulas for the KPIs are available. However, it can be applied to any other accelerator if the necessary information is accessible. Finally, the methodology for determining the pipeline architecture and layer mapping is both generic and exact. Recently, Cai et al. [17] proved that grouping layers onto different configurable NPUs considering the given KPI can be expressed using Mixed Integer Linear Programming (MIP) and they developed heuristics to solve approximately this optimization problem. In contrast, our paper demonstrates that this problem can be exactly solved using a polynomial time-efficient algorithm, ensuring easy scalability for NNs with numerous layers.

The goal of this study is not a standard comparison between state-of-the-art accelerators but to highlight a pipeline methodology, applicable to a wide range of accelerators, that optimizes different KPIs for a given throughput. The objective is then to elaborate on the potential of the accelerator's pipeline methodology to improve KPIs compared to a single instance of this accelerator with an equivalent number of PEs. This observation is made in [17], where they experimentally demonstrate a speedup of $1.2\times$ to $6.3\times$ compared to benchmarked ASICs. Additionally, thanks to Gemini-1 [19], we observe that the pipeline significantly improves the maximum throughput of the application, and the computing resources are better utilized. For example, the implemented pipeline achieves a throughput surpassing what can be achieved by a single NPU, up to $3.2\times$ for MobileNet [20]. Moreover, we also notice that, even when SRAMs are taken into consideration, the KPIs of optimal solutions using the pipeline architecture are often better. For instance, we observe a 13% throughput improvement for the same area or a 14% throughput improvement within the same power budget for a VGG-like network [21]. Furthermore, low-frequency utilization in the pipeline leads to substantial power savings without sacrificing throughput, achieving a 43% power gain at 38.4 inferences per second. Finally, it enables the best energy efficiency, achieving twice the energy reduction compared to the best energy obtained with a unique NPU. Another key objective of this paper is to simplify the process of identifying the optimal pipeline architecture and layer mapping. To this end, we propose a dynamic programming solution that is exact and scalable.

This paper is organized as follows. Section II describes our optimization problem. Section III presents our dynamic programming algorithm to solve it. Section IV is dedicated to our experiments. Section V is our conclusion.

II. Description of the problem

This section introduces the problem and important properties of feasible solutions. Subsections II-A, II-B and II-C provide brief descriptions of the NPU, the considered NNs, and the pipeline architecture, respectively. Subsection II-D describes a feasible mapping; subsections II-E and II-F show the computation of the minimum SRAM capacities and the minimum number of PE for each NPU of the pipeline for a fixed mapping and a minimum throughput. Lastly, subsections II-G and II-H present the considered KPIs and address the identified problem.

A. NPU accelerator working principle

The NN accelerator consists of a configurable NPU engine, with N adjustable parallelized PEs, and on-chip RAMs for storing NN weights and *fmaps* as illustrated by Figure 1. The partitioning of RAMs into multiple cuts or considering them as multi-bank instances does not affect the problem at hand. The NPU reads and writes *fmaps* in the FMAPS RAM. The weights are read from the Weights RAM. To streamline the study, we consider that the input *fmaps* and the NN weights are preloaded into their corresponding RAMs. It is assumed that the NPU process the NN layers sequentially.

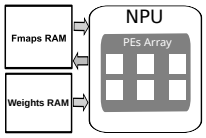

Fig. 1: *Single NPU accelerator general architecture*

B. Description of the NN

The fixed feed-forward NN to be processed is composed by a set of $\ell > 0$ successive layers $\mathcal{L} = [L_0, L_{\ell-1}]$, as presented by Figure 2. The intermediary *fmaps* are denoted by $\mathcal{I} = [I_1, I_{\ell-1}]$; for any $j \in [1, \ell-1]$, I_j is the *fmap* of size s_j produced by the layer L_{j-1} and input of L_j.

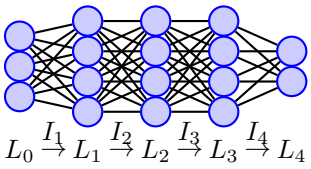

Fig. 2: *A feed-forward NN of $\ell = 5$ layers and the corresponding intermediary fmaps*

C. Description of the pipeline architecture

The pipeline is composed of n NPUs instances $\mathcal{G} = [G_0, G_{n-1}]$ and $n+2$ SRAMs $\mathcal{R} \cup \{\text{IFMAP}, \text{OFMAP}\}$ with $\mathcal{R} = [R_0, R_{n-1}]$. The components are linked following Figure 3 all located on the same chip. The two SRAMs IFMAP and OFMAP contain respectively the successive input and output *fmaps*; the other SRAMs are dedicated to intermediary *fmaps*. Each NPU $G_i \in \mathcal{G}$ contains N_i processors, while each SRAM $R_i \in \mathcal{R}$ has a capacity denoted as K_i (in KB). If single-port RAMs are used, it is assumed that there is no simultaneous access to the SRAM R_i from the two NPUs G_i and G_{i+1} Our study does not consider Weights RAMs since they do not impact the optimization: indeed, the quantity of NN weights remains constant and are distributed into the multiple RAMs.

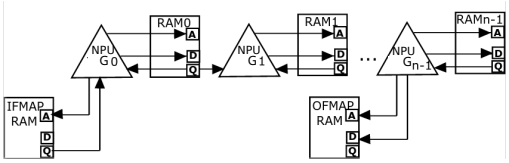

Fig. 3: *Overview of a pipeline architecture of n NPU*

D. Layers mapping on NPUs

As the NN layers are consecutive, the natural way to distribute them to $\mathcal{G}$ is using a pipeline scheme. A mapping of the layers to a pipeline architecture of n NPU $\pi : [0, \ell-1] \mapsto [0, n-1]$ fulfills the following conditions: a layer is processed by exactly one NPU, layers L_0 and $L_{\ell-1}$ are processed respectively by G_0 and G_{n-1}, each NPU processes at least one layer, and the mapping π is non-decreasing, i.e. if the NPU G_i processes the layer L_j,

then any subsequent layer $L_{j'}$ with $j' > j$ is assigned to an NPU $G_{i'}$ with $i' \geq i$.

E. Evaluation of the minimum intermediary SRAMs capacity

First, we do not consider the two SRAMs IFMAP and OFMAP since they only store respectively the input and output *fmap* only fixed by the NN. We show in the following that a minimum size $\widehat{K}_i(\pi)$ of the intermediate memories R_i for $i \in [0, n-1]$ can be defined from any mapping π.

Lemma 1. *For any couple $(i,j) \in [0, n-1] \times [1, \ell-1]$ and any layers mapping π, the intermediary* fmap *I_j is stored by R_i if and only if $\pi(j-1) = i$.*

Proof. We first observe that the intermediary *fmap* I_j is the output of the layer L_{j-1}.

Let suppose first that I_j is stored by R_i. Then, following the communication scheme of the pipeline architecture, L_{j-1} is mapped to G_i and thus $\pi(j-1) = i$. Conversely, if $\pi(j-1) = i$, then the output I_j of L_{j-1} is stored to the output R_i of G_i, which completes the proof. □

Theorem 1. *Let us consider a layers mapping π and $i \in [0, n-1]$. Let us also consider the value $\bar{j}$ (resp. $\underline{j}$) as the maximum (resp. minimum) value $j \in [1, \ell-1]$ such that $\pi(j-1) = i$. Then, $\widehat{K}_i(\pi) = \max(s_{\underline{j}}, \max_{j \in [\underline{j}, \bar{j}-1]}(s_j + s_{j+1}))$ is the minimum feasible size of the SRAM R_i.*

Proof. By Lemma 1, all the layers L_{j-1} for $j \in [\underline{j}, \bar{j}]$ are performed by G_i and their output *fmap* I_j are stored by R_i.

Now, the output $I_{\underline{j}}$ is first solely stored by R_i. Then, for $j \in [\underline{j}, \bar{j}-1]$ the *fmap* I_j and I_{j+1} need to be stored in R_i simultaneously to evaluate the layer L_j. Thus, lower bound on the memory size of R_i is proved. □

F. Execution time, throughput and a lower bound of the number of PE for a fixed mapping

We assume that the execution time of a layer L_j by the NPU G_i follows $p_{i,j} = y(L_j, N_i) + c$ where y is a non-increasing function of N_i and c a constant value. The execution time of a set of consecutive layers $[L_g, L_h]$ for $0 \leq g \leq h \leq \ell - 1$ by G_i is then expressed as:

$$p_{i,[g,h]} = \sum_{j=g}^{h} y(L_j, N_i) + (h-g).c \tag{1}$$

For any mapping π, we also note $p_{i,\pi}$ the total execution time of the successive layers mapped to G_i.

In a pipeline system, n *fmaps* are processed simultaneously (when the pipeline is filled). The **throughput** T of a mapping π is then given by the execution time of the slowest NPU, i.e. $T = \min_{i \in [0,n-1]} \frac{1}{p_{i,\pi}}$. The associated **period** P is then defined as $P = \frac{1}{T} = \max_{i \in [0,n-1]} p_{i,\pi}$ and has to be minimized. In the following, $P^\star$ denotes a fixed upper bound of the period.

Theorem 2. *Let us consider that for $i \in [0, n-1]$, the layers L_j mapped to G_i follow $q \leq j \leq h$. Let also suppose that a maximum period $P^\star$ is fixed. Then, the minimum number of PEs of G_i, denoted by $\widehat{N}_i(\pi, P^\star)$, can be computed in time complexity $\mathcal{O}(\log N_{\max})$ where $N_{\max}$ is an upper bound of the number of PEs.*

Proof. Since y is a non-increasing function of N_i, $\widehat{N}_i(\pi, P^\star)$ can be computed by simply using a binary search [22] on N_i. □

G. Objective functions considered (or KPIs)

Several objectives $\varphi(\pi, P^\star)$ can be taken into consideration to evaluate the couple $(\pi, P^\star)$ and the associated pipeline architecture. However, we suppose that these objective can be evaluated (by a close formula or a polynomial time algorithm). If $\varphi_i(\pi, P^\star)$ is the restriction of φ to the layers allocated to the NPU G_i, $\varphi(\pi, P^\star) = \sum_{i=0}^{n-1} \varphi_i(\pi, P^\star)$.

The **latency** corresponds to the total execution time of the NN. If several successive executions of the NN are launched, the execution time of the treatment for each NPU is fixed to the maximum period $P^\star$ and thus the total latency is $Lat(\pi, P^\star) = n.P^\star$. The **area** of the pipeline system is the sum of the area of each NPU G_i with the corresponding SRAM. For $i \in [1, n-1]$, they depend on the size $\widehat{K}_i(\pi)$ of the SRAM R_i and the number of PEs $\widehat{N}_i(\pi, P\star)$ of G_i. We thus note $a(\pi, P^\star) = a_{NPU}(\pi, P^\star) + a_{RAM}(\pi)$. Another commonly considered objective is the NPUs **power consumption** $pw(\pi, P^\star)$; it computed summing up the static and dynamic power of each NPU in the pipeline system. The NPUs' power can be aggregated as they operate simultaneously. Lastly, the **energy** consumed by frame is $e(\pi, P^\star) = P^\star.pw(\pi, P^\star)$. Any other convex linear combination of KPIs can be considered.

H. Formal description of the problem

An instance of our problem is defined by a fixed NN with the number of layers and the size of the intermediary *fmaps*, the description of the configurable NPU engine, a maximum period $P^\star$ and an objective function φ to minimize. The problem consists then to compute a mapping π that minimizes φ. As stated by Theorems 1 and 2, the pipeline architecture is deduced from π and $P^\star$.

The described pipeline approach is generic and applicable to any accelerator processing NN layers sequentially.

III. Description of the Dynamic Programming algorithm

The problem tackled in this paper can be seen as a particular instance of the assembly line balancing [7]. We develop in the following an intuitive Dynamic Programming algorithm inspired from [9] to solve it exactly.

The main idea of our algorithm is to build a valued directed state graph $\mathcal{H} = (V, E, w)$ defined as follows: the set of vertices is $V = \{s, p\} \cup V_1$ with $V_1 = \{[g,d], 0 \leq g \leq d \leq \ell - 1\}$. Each vertex $u = [g,d] \in V_1$ models the successive layers $[L_g, L_d]$ mapped to a same NPU. The set of arcs $E = E_s \cup E_p \cup E_1$ is defined as:

- $E_1 = \{a = (u, u') \in V_1^2, u = [g,d]$ and $u' = [d+1, m]\}$ models that u' can be mapped to a NPU just after u;

- $E_s = \{(s, u), u = [0, d] \in V_1\}$;
- $E_p = \{(u, p), u = [g, \ell - 1] \in V_1\}$.

Lastly, the valuation $w : E \mapsto \mathbb{N}$ of the arcs is defined following the objective φ as follows:

- For each arc $a = (u, p) \in E_p$, $w(a) = 0$;
- Each arc $a = (u, v) \in E_p \cup E_1$ with $u = [g, d]$ is valued by the restriction of φ to the layers $[L_g, L_d]$ assuming that these layers are mapped to a same NPU.

We observe that, for some objectives as the area or the power, the minimum number of PEs requires for each node to achieve the requested throughput, or the minimum memory size must be evaluated for each vertex in order to compute the valuations w. Moreover, the throughput constraint is fulfilled by fixing a minimum number of PEs, as stated by Theorem 2.

Figure 4 presents the state graph $\mathcal{H}$ for $\ell = 4$ without the valuation of the arcs. One can observe that paths of the state

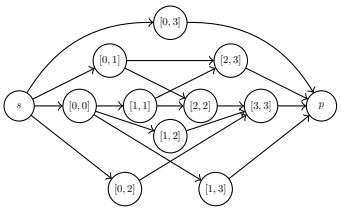

Fig. 4: *A state graph $\mathcal{H}$ for $\ell = 4$. The valuations are not presented.*

graph $\mathcal{H}$ from s to p model all the feasible mapping π. For our example, the path $s \to [0, 1] \to [2, 2] \to [3, 3] \to p$ is associated to the mapping $\pi : [0, 3] \to [0, 2]$ with $\pi(0) = \pi(1) = 0$, $\pi(2) = 1$ and $\pi(3) = 2$. The minimum number of PE's for each NPU and the minimum memory size are evaluated using Theorems 2 and 1. Since the arcs are valued from the criteria restriction on each NPU, the path of minimum value from s to p models a feasible mapping minimizing φ. Thus, our algorithm simply builds the state graph $\mathcal{H}$ and finds its shortest path using Dijkstra algorithm [22].

Theorem 3. *The time complexity of the DP algorithm belongs to $\mathcal{O}(\ell^4 \log \ell + \ell^2 \log N_{max})$ where N_{max} is an upper bound on the number of PEs.*

Proof. The number of vertices (resp. arcs) of the state graph belong to $\mathcal{O}(\ell^2)$ (resp. $\mathcal{O}(\ell^4)$). Moreover, for each vertex $u \in V_1$, determining the minimum number of PEs required to reach a given throughput takes $\mathcal{O}(\log N_{max})$ instructions (see Theorem 2), while determining the minimum memory size requires $\mathcal{O}(\ell)$ instructions (see Theorem 1). So, the computation of $\mathcal{H}$ belongs to $\mathcal{O}(\ell^4 + \ell^2(\ell + \log N_{max}))$, which is equivalent to $\mathcal{O}(\ell^4 + \ell^2 \log N_{max})$. Now, the complexity of Dijkstra algorithm [22] is bounded by $\mathcal{O}(\ell^4 \log \ell)$. □

IV. Experimental results

This section aims to present our experiments using the DP algorithm. Subsection IV-A presents briefly the evaluation of the KPIs for the industrial prototype Gemini-1. The NN considered are detailed in subsection IV-B. Subsection IV-C shows that the architecture pipeline allows reducing drastically the period, allowing the increasing of the latency. In Subsection IV-D, it is proved that the pipeline architecture allows adjusting as possible the whole number of PEs. Lastly, Subsection IV-E and IV-F studies respectively the power and the energy minimization, and shows that for small values of the period, the pipeline architecture has better performance even when the SRAMs are taken into account.

A. KPI of Gemini-1

We consider for our experiments the industrial prototype Gemini-1 for which analytical close formulas for execution times and KPIs are available [19]. Additionally, the number of PEs in Gemini-1 is not fixed and can be adjusted before logic synthesis. The execution time function of a layer L_j on a NPU of N_i PEs can be set here to $y(L_j, N_i) = \lceil \frac{f(L_j)}{N_i} \rceil$ where f is a function depending on the layer L_j parameters.

The models of KPIs were obtained using gates level simulations for the CMOSC40 technology at 25°C using the typical corner for a 200MHz sign-off frequency. The RAMs considered are SPREGHD SRAMs developed by STMicroelectronics.

The area and power of the NPU G_i follow $a_{i,NPU}(\pi, P^\star) = c_0 + c_1 \times \widehat{N}_i(\pi, P^\star)$ and $pw_{i,NPU}(\pi, P^\star) = g_0(\pi) + g_1(\pi) \times \widehat{N}_i(\pi, P^\star)$. The two values c_0 and c_1 are constants obtained through linear regression on various hardware setups. The functions g_0 and g_1 are determined through regression on diverse hardware configurations for different NN executions, and their values are dependent on the layer parameters.

Lastly, the synchronization between two consecutive NPUs is ensured by temporarily interrupting them when their input *fmaps* are not yet completed. Single port RAMs are considered; the NPUs do not read data in every cycle, which allows for write operations to be performed when the read operations are not required (readings and writings never occur simultaneously).

B. Description of the NNs considered

Gemini-1 is designed for edge applications. It is tailored for NNs with low RAM requirements. Our experiments do not serve demonstrations, they highlight the practicality of our generic approach. Importantly, while Gemini-1 is limited to NNs used on the edge, there should be no expected issues when using larger ones with other accelerators. This is attributed to our allocation algorithm, which maintains polynomial complexity. Three NNs were taken into consideration: MobileNet x0.25 [20] with 27 layers predominantly comprising depthwise separable convolutions, VGG-like (shown in Figure 5), which is inspired by VGG-16 [21] and consists of 11 layers combining convolution, maxpools, and fully connected layers (that are relatively large), and P-Net with 7 layers [23], containing the same layers as the previous ones. These networks have 8-bit quantized weights with weight sizes of 850,000, 526,000, and 7,900, respectively, and input feature map (*fmaps*) sizes of 224x224x3, 128x128x1, and 32x32x1, respectively. In cases where the objective φ involves the use of RAMs, we only consider the VGG-like network, as the

other two networks were either too large, requiring progressive loading, or too small to impact RAM performance.

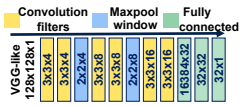

Fig. 5: *VGG-like network structure*

C. Minimizing the throughput and the latency

We start by considering the latency minimization, i.e. $\varphi = Lat$. We observe that smaller values of the period $P^\star$ are unattainable with a single NPU and that a pipeline allows to decrease drastically the period. In the following, we restrict solutions to fewer than 700 PEs to remain realistic.

Table I compares the minimum period obtained for a single NPU (in this case the latency and the period are equal as the images are processed sequentially) vs. the minimum period for a pipeline with its corresponding minimum latency. Throughput can be optimized 3.2 times, 3.5 times, and 1.85 times respectively for MobileNet, VGG-like, and P-Net using a pipeline architecture. However, the pipeline's latency is significantly higher; for MobileNet, using 21 NPUs results in a latency 6.6 times higher than the best single NPU latency. Consequently, the pipeline significantly reduces the system's period but increases latency.

TABLE I: *Minimal reachable period P^* and the corresponding latency for a pipeline architecture vs. a single NPU (in cycles)*

		Pipeline Solution	
NN	Single NPU	Min. P^*	Corresp. Lat.
Mobilenet	26810	8400	176400
VGG-like	117890	34000	102000
P-Net	2720	1470	5880

D. Minimization of Processing Elements number

We tackle in this section the minimization of the whole number of PEs ($\widehat{N}$). Figure 6 presents $\widehat{N}$ depending on the period for the three NNs. The green curves correspond to the optimal pipeline solution obtained using our DP algorithm; the red ones correspond to the solution using a unique NPU.

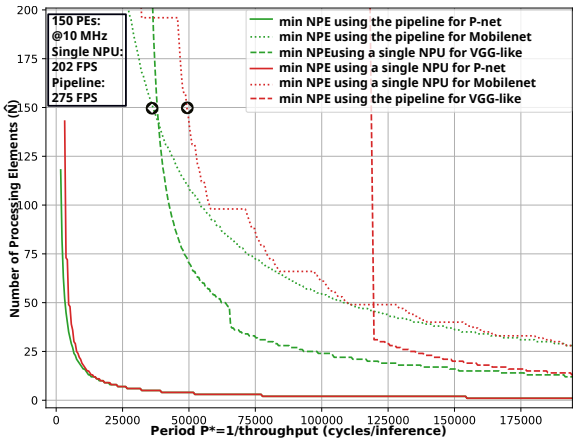

Fig. 6: *PEs number under throughput constraints for 3 NNs*

As mentioned earlier, smaller values of the period cannot be achieved considering only one single NPU. Furthermore, we observe that the green curves are always under their corresponding red ones. The consequence is that PEs are underutilized for many layers for the single NPU, while the pipeline optimal solution adjusts as possible the PEs number.

Let us consider that a frame corresponds to the total execution of the NN for one image. For example, we observe that for MobileNet, with 150 PEs operating at 10MHz, a single NPU can process 202 frames per second (FPS). However, the optimal pipeline solution for 150 PEs obtained by our algorithm is given by $n = 5$, $N_i = [20, 56, 18, 49, 7]$, and the grouping of the layers $[0]$, $[1, 11]$, $[12, 14]$, $[15, 22]$ and $[23, 26]$; the images processing rate is then 275FPS for the same frequency, resulting in a significant acceleration of 36%.

E. Minimizing the power or the area

Let us consider first the power minimization. Two objective functions were studied: $pw_{NPU}(\pi, P^\star)$ and $pw(\pi, P^\star) = pw_{NPU}(\pi, P^\star) + pw_{RAM}(\pi)$ to take the SRAMs into consideration. Figure 7 presents the minimum values pw_{NPU} (dashed lines) and pw (solid lines) depending on the period $P^\star$ at 1MHz for the optimum pipeline (green curves) or a single NPU approach (red curves).

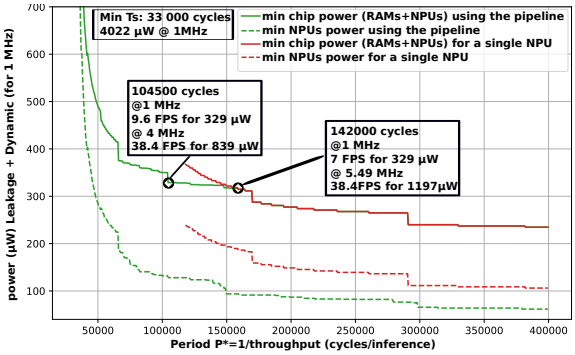

Fig. 7: *Power under throughput constraints for VGG-like*

The main observation is that the pipeline remains interesting for higher throughput. As example, the pipeline architecture given by $n = 2$, $N_i = [16, 6]$ along with the layers mapping $[0, 7]$, $[8, 10]$ can achieve a processing rate of 9.6FPS at 1MHz and consuming $pw = 329\mu$W; the single NPU configuration can only process 7FPS under the same power budget.

As aforementioned, the pipeline method reduces the whole number of PEs, or the power dedicated to NPUs. The downside is that the pipeline requires much more memory than a single NPU. Therefore, the pipeline solution is usually interesting when there is a significant enough reduction in the NPU's power to compensate for the increase due to the added RAMs.

However, the range of applications where the pipeline architecture is beneficial extends beyond high-throughput applications. Indeed, since dynamic power is linear with respect to frequency (as shown in [24]), it is possible to lower the frequency (voltage fixed) to decrease the power: when targeting a throughput of 38.4FPS, the optimum pipeline architecture consumes 839μW at 4MHz. The single NPU would need to

operate at 5.49MHz to achieve the same throughput, resulting in a power of 1197μW (43% higher than the pipeline).

Lastly, our experiments on the area minimization (not presented here due to a lack of space) are leading to a conclusion similar to the power (13% higher throughput can be achieved by the pipeline for 4.12mm², for example).

F. Minimizing the energy

Figure 8 presents our experiments on the minimization of the energy depending on a fixed period $P^\star$. Overall,

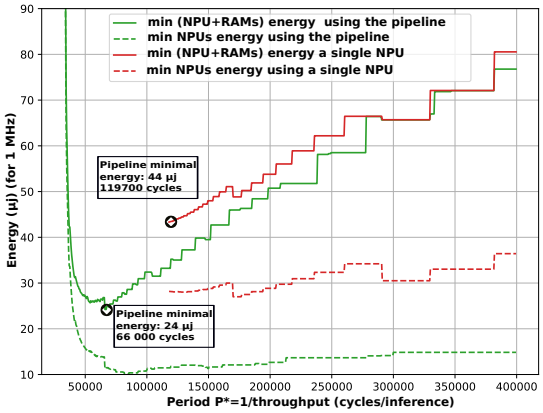

Fig. 8: *Energy under throughput constraint for VGG-like*

we observe that optimal solutions pipe-lining multiple NPUs consume less energy than single NPU ones even when the SRAMs are taken into account, particularly when facing high throughput constraints. This is attributed to the fact that the power of a single NPU is high under such constraints. The minimum energy of 24µJ is achieved for $P^\star = 66000$ (the pipeline configuration is $n = 3$, $N_i = [26, 11, 1]$ along with the mapping $[0, 1, 2, 3, 4, 5, 6, 7]$, $[8]$, $[9, 10]$), which cannot be attained with a single NPU. It is nearly two times lower than the best energy achieved with one NPU (44µJ). Additionally, since the green curve is consistently below the red one, it is possible to optimize energy for a specific throughput. Finally, the optimization can be further improved by reducing the frequency, as described in the previous paragraph.

V. Conclusion

We developed in this paper a generic methodology for optimizing various KPIs using NN accelerators, while adhering to throughput constraints. The methodology provides the pipeline architecture and layers mapping on NPUs. The algorithms developed are exact and efficient with low time complexity. Through experimentation with an industrial prototype, we demonstrated that the pipeline structure enables optimizing the throughput beyond what a single NPU can achieve. Additionally, it offers important optimization for hardware resources, area, power, and energy. The most significant advantage is low-frequency operation, resulting in substantial power gains.

This works also offers numerous theoretical perspectives. For example, is it possible to extend our approach to multiples NNs with different utilization rates, or to give the Pareto-front for different KPIs simultaneously. Another interesting question is the determination of an optimal mapping for a fixed pipeline architecture. Finally, with appropriate adjustments, this method could target IMC accelerator tiles and also support NNs with residual layer connections.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553), 2015.

[2] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12), 2017.

[3] Linyan Mei, Huichu Liu, Tony Wu, H. Ekin Sumbul, Marian Verhelst, and Edith Beigne. A uniform latency model for dnn accelerators with diverse architectures and dataflows. In *2022 Design, Automation,Test in Europe Conference, Exhibition (DATE)*, 2022.

[4] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[5] 2017. http://nvdla.org/ 69 76 92 94 96 97 113 114 Nvidia, NVDLA Open Source Project.

[6] Rastislav Struharik, Bogdan Vukobratović, Andrea Erdeljan, and Damjan Rakanović. Conna – compressed cnn hardware accelerator. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 365–372, 2018.

[7] Nils Boysen, Philipp Schulze, and Armin Scholl. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3), 2022.

[8] Eduardo Álvarez-Miranda and Jordi Pereira. On the complexity of assembly line balancing problems. *Computers & Operations Research*, 108, 2019.

[9] Michael Held, Richard M Karp, and Richard Shareshian. Assembly-line balancing—dynamic programming with precedence constraints. *Operations research*, 11(3), 1963.

[10] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *2016 49th Annual IEEE/ACM MICRO*, 2016.

[11] Marcos Lupión Lorente, N.C. Cruz, Juan Sanjuan, Ben Paechter, and Pilar Ortigosa. Accelerating neural network architecture search using multi-gpu high-performance computing. *The Journal of Supercomputing*, 79, 12 2022.

[12] Guanwen Zhong, Akshat Dubey, Cheng Tan, and Tulika Mitra. Synergy: A HW/SW framework for high throughput cnns on embedded heterogeneous soc. *CoRR*, abs/1804.00706, 2018.

[13] Bogil Kim, Sungjae Lee, Amit Ranjan Trivedi, and William J. Song. Energy-efficient acceleration of deep neural networks on realtime-constrained embedded edge devices. *IEEE Access*, 8, 2020.

[14] Qingyang Yi, Heming Sun, and Masahiro Fujita. FPGA based accelerator for neural networks computation with flexible pipelining. *CoRR*, abs/2112.15443, 2021.

[15] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[16] Yangyang Zhao, Qi Yu, Xuda Zhou, Xuehai Zhou, Xi Li, and Chao Wang. Pie: A pipeline energy-efficient accelerator for inference process in deep neural networks. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, 2016.

[17] Xuyi Cai, Ying Wang, Xiaohan Ma, Yinhe Han, and Lei Zhang. Deepburning-seg: Generating dnn accelerators of segment-grained pipeline architecture. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

[18] Xiaochen Peng Shanshi Huang, Shimeng Yu. Neurosim v1. 2019.

[19] Ali Oudrhiri, Emilien Taly, Nathan Bain, Alix Munier, Roberto Guizzetti, and Pascal Urard. Performance modeling and estimation of a configurable output stationary neural network accelerator. In *2023 IEEE 35th SBAC-PAD*.

[20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
[22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.
[23] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10), 2016.
[24] Tarek Darwish and Magdy Bayoumi. 5 - trends in low-power vlsi design. In WAI-KAI CHEN, editor, *The Electrical Engineering Handbook*. Academic Press, Burlington, 2005.