SuperNPU: An Extremely Fast Neural Processing Unit Using Superconducting Logic Devices

Koki Ishida[1*], Ilkwon Byun[2*], Ikki Nagaoka[3], Kosuke Fukumitsu[1], Masamitsu Tanaka[3], Satoshi Kawakami[1], Teruo Tanimoto[1], Takatsugu Ono[1], Jangwoo Kim[2], and Koji Inoue[1†]

[1]Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University
{koki.ishida, kosuke.fukumitsu, satoshi.kawakami, teruo.tanimoto, takatsugu.ono, koji.inoue}@cpc.ait.kyushu-u.ac.jp
[2]Department of Electrical and Computer Engineering, Seoul National University
{ik.byun, jangwoo}@snu.ac.kr
[3]Department of Electronics, Nagoya University
nagaoka@super.nuee.nagoya-u.ac.jp, masami_t@nagoya-u.jp

***Abstract*—Superconductor single-flux-quantum (SFQ) logic family has been recognized as a highly promising solution for the post-Moore's era, thanks to its ultra-fast and low-power switching characteristics. Therefore, researchers have made a tremendous amount of effort in various aspects to promote the technology and automate its circuit design process (e.g., low-cost fabrication, design tool development). However, there has been no progress in designing a convincing SFQ-based architectural unit due to the architects' lack of understanding of the technology's potentials and limitations at the architecture level.**

In this paper, we present how to architect an SFQ-based architectural unit by providing design principles with an extreme-performance neural processing unit (NPU). To achieve the goal, we first implement an architecture-level simulator to model an SFQ-based NPU accurately. We validate this model using our die-level prototypes, design tools, and logic cell library. This simulator accurately measures the NPU's performance, power consumption, area, and cooling overheads. Next, driven by the modeling, we identify key architectural challenges for designing a performance-effective SFQ-based NPU (e.g., expensive on-chip data movements and buffering). Lastly, we present *SuperNPU*, our example SFQ-based NPU architecture, which effectively resolves the challenges. Our evaluation shows that the proposed design outperforms a conventional state-of-the-art NPU by 23 times. With free cooling provided as done in quantum computing, the performance per chip power increases up to 490 times. Our methodology can also be applied to other architecture designs with SFQ-friendly characteristics.

***Index Terms*—Single flux quantum (SFQ), Cryogenic computing, Modeling, Simulation**

I. Introduction

We are now facing the era where both Moore's Law [1] and Dennard scaling [2] do not hold anymore. In this era, we are running out of a convincing option to improve the performance of the computer system, while maintaining its power and temperature budget. Therefore, we believe that it is the right time to actively exploit emerging device technologies with significant potentials and make a serious effort to improve their feasibility by resolving their limitations.

Among several candidates, superconductor SFQ logic family [3], [4] is a highly promising solution thanks to its ultra-fast speed and low-power consumption at 4 K. The SFQ technology enables a low-level voltage impulse-driven switching which allows both extremely-fast switching (~10^{-12}s) and low-energy consumption (~10^{-19} J per switching) [3], [4]. That is, with this technology, it is feasible to improve the device's clock frequency (and thus performance) by order of magnitude (i.e., several tens of GHz [5], [6]).

By focusing on these high potentials, many serious SFQ-related research efforts have been made in various aspects to promote the technology and automate its device and circuit-level design process (e.g., technology hardening, low-cost fabrication, design tool development) [7], [8]. As a result, the SFQ logic is now considered for an extreme-performance computing and a promising post-Moore solution.

However, due to its unique pulse-driven nature, SFQ logic requires completely different architecture designs from conventional CMOS technology. Therefore, with the architectural trade-offs considered, the following questions must be clearly addressed to computer architects: (1) what architecture is promising for this technology, (2) how to implement various microarchitectural units with the voltage pulse-driven logics, (3) how to maximize its potential at the architecture level while minimizing its limitations, and (4) how to simulate and validate a proposed architecture design.

In this paper, we resolve the fundamental challenges by (1) providing straightforward answers to the questions above, and (2) presenting *SuperNPU*, our example SFQ-based neural processing unit (NPU) design. First, as our case-study architecture in this work, we choose to architect a conventional NPU and present the basic structure with carefully designed microarchitectural units. For instance, we design our baseline NPU architecture consists of processing elements (PEs) with the weight-stationary dataflow, systolic array network, and data alignment unit. This baseline NPU architecture well satisfies the requirements of SFQ-based logics such as fast computation, dataflow-like data movements, and shift-register-based memory implementation, respectively.

*Both authors contributed equally to this research.
†Corresponding author.

Next, we implement an architecture-level simulation framework to model an SFQ-based NPU architecture accurately. Our simulator can accurately estimate the under-the-design NPU's performance, power consumption, area at various levels (i.e., gate, microarchitecture, architecture). For the purpose, the simulator constructs a target SFQ-based NPU architecture by integrating SFQ-based microarchitecture and gate modules using AIST 1.0 μm fabrication process technology [9]. We carefully validate the simulator by comparing the results obtained from our die-level microarchitecture prototypes and post-layout simulations against our modeling results.

Third, based on the validated model, we identify key performance bottlenecks in a naively designed SFQ-based NPU. First, the data movement among different units and within a single unit takes too long, mainly due to the shifting register-based operation. Next, fast computing units often become idle due to the workload's low computational intensity and relatively slow memory access. Also, the on-chip memory underutilization can make the above overheads much worse.

Lastly, we present *SuperNPU*, our example SFQ-based NPU design, which effectively resolves the performance bottlenecks at the architecture level. First, it merges the partial-sum and output memories to avoid unnecessary inter-memory data movements. Second, it partitions a larger on-chip buffer to multiple small chunks to reduce the length of intra-memory shifting as well as the underutilization. Third, it increases the computational intensity by balancing hardware resources for a larger-batch purpose. Fourth, it further increases each PE's utilization by assigning more registers to each PE for enabling multi-kernel execution.

Our evaluation shows that SuperNPU significantly outperforms a conventional NPU design by 23 times when running various CNN workloads. However, without the SFQ-aware architectural optimizations, the SFQ-based NPU design's performance drastically drops to the point even below the conventional design. Therefore, it is extremely essential to identify the SFQ-unfriendly bottlenecks and architect an optimized design to resolve them. With the cooling cost considered, SuperNPU's performance per watt is slightly higher than the conventional design. But, with free cooling cost assumed, SuperNPU's performance per watt becomes significantly higher than the conventional design by 490 times.

In summary, our work makes the following contributions:

- **Architecting an SFQ-based NPU:** To the best of our knowledge, this is the first work to design an NPU which addresses the SFQ technology's architectural trade-offs.
- **Simulation framework:** It is also the first work to model and validate a simulator for SFQ-based architectures.
- **SFQ-specific architectural optimizations:** We identify critical architectural bottlenecks and optimizations which can cause a performance variance around 60 times.
- **Significant results:** SuperNPU provides extreme performance and power efficiency by outperforming a conventional design by 23 times and 490 times, respectively.
- **Applicability:** Our modeling-driven methodology can be applied to other architectures favoring the SFQ logic.

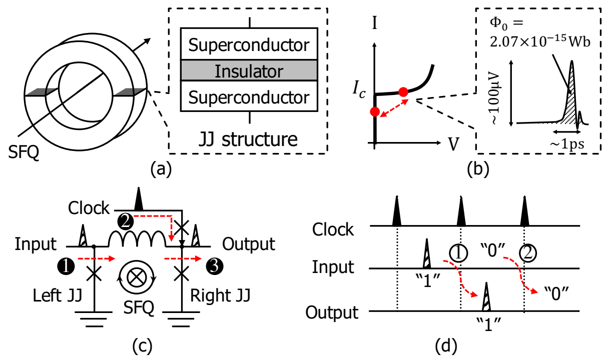

Fig. 1. (a) Superconductor ring with SFQ (b) Electrical characteristics of JJ (c) Circuit diagram of an SFQ-based DFF with (d) its operating example

II. Background & Motivation

A. Superconductor SFQ logic technology

Superconductor SFQ logic [3], [10] and its energy-efficient families [11]–[15] are representative ultra-fast and low-power VLSI technologies using superconducting devices. Fig. 1(a) shows a basic circuit element of SFQ technology, a superconductor ring. SFQ circuits utilize the existence of a single magnetic flux quantum (SFQ) in the superconductor ring as an information carrier, as similar to the voltage level in conventional CMOS circuits. The superconductor ring can store and transfer the SFQ by using a superconducting device called Josephson junction (JJ), which consists of a thin insulator sandwiched by the superconductors.

Fig. 1(c) shows the working principle of SFQ logic gates, which operate based on the superconductor rings. We take an SFQ-based delay-flip-flop (DFF) as our example due to its simplest structure consisting of only a single superconductor ring and a clock line. First, when the input pulse comes to the ring, it makes the current flowing through the left JJ higher than its critical current, I_c. With the electrical characteristic shown in Fig. 1(b), the left JJ generates a voltage pulse and it is stored to the ring as an SFQ (Fig. 1(c) ❶). Next, by taking a clock pulse (Fig. 1(c) ❷), the right JJ is activated and the stored SFQ is transferred to the output as a voltage pulse (Fig. 1(c) ❸). In this manner, SFQ gates can define the logical value '1' as the existence of stored SFQ between the clock pulses (Fig. 1(d) ①). On the other hand, if no input pulse comes during a clock period, no voltage pulse is generated on the output, and it indicates the logical value '0' (Fig. 1(d) ②).

The SFQ technology's voltage pulse-driven nature enables the extremely low-latency (~10^{-12} s) and low-energy (~10^{-19} J) JJ switching [3], [11]. With this promising aspects, serious SFQ-related research efforts have been made in various aspects to promote the technology and automate its design process [7], [8], [11], [12], [16]–[19]. Moreover, several physical implementations have been successfully demonstrated at outstanding frequencies, a few tens of GHz [5], [6]. As a result, the SFQ logic is now considered as a highly promising solution for the post-Moore era and extreme-performance computing.

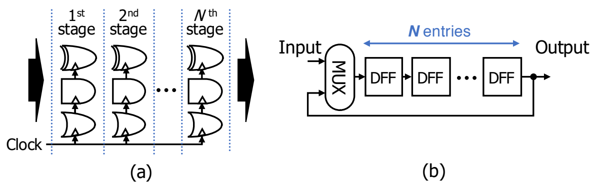

Fig. 2. Example of the SFQ technology's architectural characteristics (a) Gate-level-pipelined datapath (b) 1-bit N-entry shift register

B. SFQ technology in the architect's perspective

In the architect's perspective, it is essential to understand SFQ logic's architectural characteristics originated from the pulse-driven nature. Therefore, we summarize some notable features as follows.

1) Deeply pipelined datapath: In the SFQ logic, architects can naturally apply the gate-level pipelining without any overhead. All SFQ logic gates are synchronized with the clock because they need a clock pulse to transfer the stored SFQ to the adjacent gates (Section II-A). In other words, every SFQ gate has the latch functionality and thus can be pipelined without additional DFFs (Fig. 2(a)). With this property, several chips have been successfully demonstrated at several tens of GHz [5], [6]. However, the deep pipeline structure can suffer from performance degradation because it is difficult to avoid data (or control) hazards and the huge pipeline stalls. Therefore, the SFQ technology favors streaming execution rather than applications with complex control flows.

2) Frequency determination with pulse-driven clocking: Unlike conventional CMOS technology, SFQ circuits' frequency is determined by the timing difference between the data and clock pulse arrival. In the CMOS technology, the clock frequency is bounded by the longest datapath delay because it only can put single digital information (i.e., voltage level) in a wire. On the other hand, SFQ logic can put several data into a single wire because its data is encoded as a voltage pulse. That is, SFQ circuits can achieve high frequency by flowing many data pulses through a single wire, simultaneously. However, if there is a large difference between the data and clock arrival timing for an SFQ gate, its frequency can be significantly reduced. This is because the next clock pulse should wait for the slow data pulse propagation, and thus the time interval between two adjacent clock pulses increases. Therefore, it is crucial to match the data and clock pulse arrival timing for maximizing the SFQ circuits' frequency.

3) Shift-register-based on-chip memory: For the SFQ logic's on-chip memory, the shift-register-based memory is much more practical than the random access memory (RAM). Even though we can implement RAM with SFQ technology, it severely suffers from low driving capability and scalability. Such limitations mainly result from the difficulty of driving the word lines and bit lines with the small pulses [3], [20]. On the other hand, a shift-register-based memory does not have those problems because it just consists of the serially connected DFFs and the feedback loop (Fig. 2(b)). However, it is difficult for the shift-register-based memory to support the random memory access due to the complex control logic and the variable access latency [21]. Therefore, the SFQ technology favors applications with sequential memory access when its on-chip memory implementation is considered.

4) Lack of off-chip memory technology: It has been a long-standing challenge to implement a large-scale and high-speed off-chip memory operating at the 4K environment. There has been a few research about JJ-based memories [22]–[24], and one of them is Vortex Transition Memory (VTM) [22]. The VTM is the largest Josephson memory whose 4-kbit prototype has been demonstrated. Despite the demonstration, it has been difficult to practically use the VTM mainly due to the scaling and speed problems with the AC-biasing and the large superconductor-ring-based memory cells. Even though several off-chip memory technologies (e.g., hybrid Josephson-CMOS memory [23], [24], Josephson magnetic memory [25]) are currently being developed, these technologies also have not been put to practical use yet. For these reasons, it is currently practical to use CMOS memory technology, which is slower than the 4 K JJ-based memory but large and reliable. Therefore, SFQ technology favors computation-oriented applications with a minimal number of off-chip memory access.

C. Challenges for designing SFQ-based architectural unit

Even though there have been several studies regarding the SFQ architecture's features [5], [16], [26]–[28], there still exist critical challenges for designing SFQ-based architectural units.

SFQ-optimal architecture design: First, for the target architecture application, architects should carefully design each microarchitectural unit because the novel circuit-level trade-offs occur in SFQ logic (e.g., frequency trade-off with the applied clocking scheme). Furthermore, architects must carefully analyze the performance bottlenecks and propose the best SFQ architecture design based on the analyses. However, as far as we know, there does not exist either such SFQ-friendly microarchitecture implementation or the SFQ-optimal architecture proposed with the bottleneck analysis.

Absence of an SFQ-based architecture modeling tool: Architects are in dire need of high-level architecture modeling tools to design and evaluate their architectural innovations, especially for emerging technologies such as SFQ logic devices. Even though researchers recently have made an effort to develop several SFQ design automation tools [7], [8], to the best of our knowledge, a reliable SFQ-based architecture modeling tool is currently absent.

D. Research goal: Provide SFQ design principles with NPU

In this paper, we resolve the challenges and provide the guidelines for designing an SFQ-optimal architectural unit by presenting an extreme-performance SFQ-based NPU. We first conduct thorough analyses and introduce the baseline SFQ-based NPU architecture by designing all microarchitectural units in the SFQ-friendly manner (Section III). Next, on top of the baseline NPU architecture, we develop *SFQ-NPU*, a validated SFQ-based architecture modeling tool (Section IV). Finally, we use the tool to identify critical performance bottlenecks and propose our SFQ-optimal NPU, which successfully resolves the bottlenecks at the architecture level (Section V).

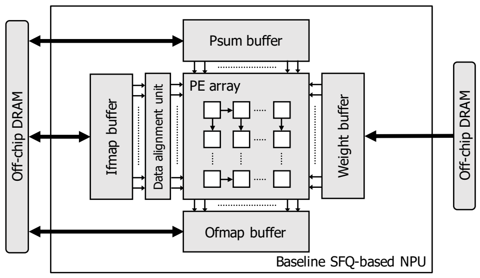

Fig. 3. Overview of our baseline SFQ-based NPU design

In this work, we choose NPU as one of the promising examples to apply our design principles for the following reasons. First, there are no complex control flows in Deep Neural Network (DNN) applications, and therefore we can fully exploit the SFQ's gate-level pipelining nature without control hazard. Second, we can take the best advantage of shift-register-based memory and avoid its disadvantage thanks to the static memory access pattern of DNN algorithms. Finally, NPUs can reduce off-chip memory access by utilizing the data-reuse pattern in DNN applications. Note that the underlying SFQ circuits, such as multipliers and adders, have already been demonstrated with around 50 GHz frequency [5], [6]. Besides, we currently target the DNN inference as the first case study to show SFQ-based NPU's potential.

III. Baseline SFQ-based NPU design

In this section, we design the baseline SFQ-based NPU architecture by identifying the SFQ-friendly implementation for key microarchitectural units. Fig. 3 shows the overview of our baseline SFQ-based NPU which mainly consists of four microarchitectural units: on-chip network unit (NW unit), processing element (PE), data alignment unit (DAU), and on-chip buffers. We perform detailed circuit-level analyses to describe our design choice for each unit, except for the shift-register-based on-chip buffers explained in Section II-B.

A. On-chip network unit design

To design the SFQ-friendly on-chip network, we compare two representative network unit (NW unit) designs: fan-out network and store-and-forward chain. The fan-out network multicasts the data to several PEs simultaneously by using the bus or tree structure. On the other hand, the store-and-forward chain provides the data and subsequently forwards it from a PE to the adjacent PE. Note that a network branch consists of a DFF (D in Fig. 4) and a wire component called splitter (S in Fig. 4), which splits a pulse into two identical pulses.

Among these two network designs, we adopt the store-and-forward chain because it is superior to the fan-out network in terms of both clock frequency and area. Fig. 4 shows the structures of three network design candidates: two splitter tree (2D and 1D) designs (fan-out network) and a 2D systolic array (store-and-forward chain). In our analysis, we include both 2D and 1D splitter tree designs which can be applied to the output stationary (OS) and weight stationary (WS) dataflow,

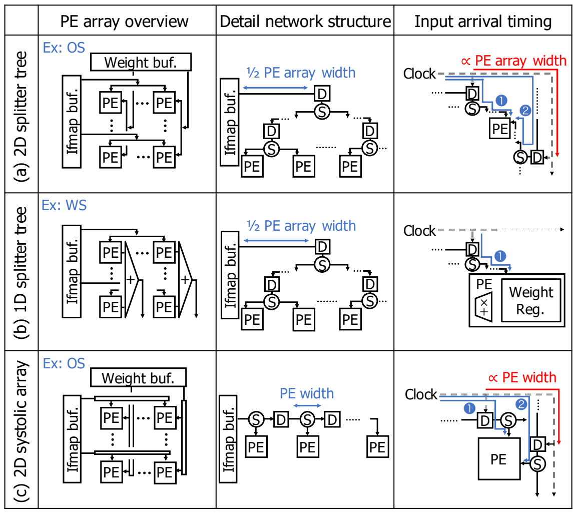

Fig. 4. On-chip network structure for three alternative designs

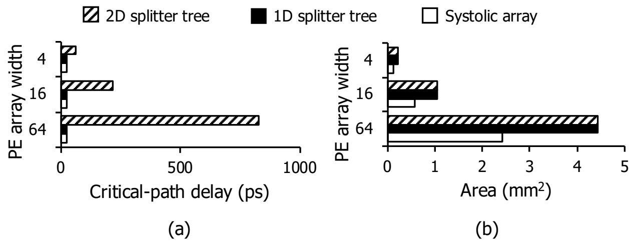

Fig. 5. Network unit designs' (a) critical-path delay and (b) area comparison

respectively. Also, we assume that all network designs target 2D square-shaped PE array.

Fig. 5 shows the critical-path delay (i.e., the inverse of maximum frequency) and the area comparison for the three network designs, obtained with JSIM [29]. First, the 2D splitter tree significantly suffers from the long critical-path delay due to the increasing timing difference of two PE inputs. As shown in Fig. 4(a), a single PE requires two inputs from each splitter tree. As both splitter trees share a global clock line, the critical-path delay increases in proportion to the PE array width (Input arrival timing in Fig. 4(a)). As a result, the critical-path delay of the 2D splitter tree keeps increasing with the PE width and reaches above 800 ps in 64×64 PE array. Even though we can mitigate this problem with the aggresive clock skewing (i.e., intentionally increase the clock propagation delay in path ❶), it incurs much more area overhead and lowers the yield of fabrication [30]. Next, even if there is no such a timing issue in the 1D splitter tree, its area overhead is high as the same with the 2D tree. The large area overhead is mainly due to the large number of wire cells for the tree construction.

On the other hand, the 2D systolic network has the shortest critical-path delay and the smallest area, as shown in Fig. 5. Even though the 2D systolic network also provides two different inputs to a single PE as same with the 2D splitter tree, their timing difference is negligible (Fig. 4(c)). Besides, its simple structure does not require much wire cells. For these reasons, we conclude that the systolic array is more suitable and adopt it as our on-chip network design.

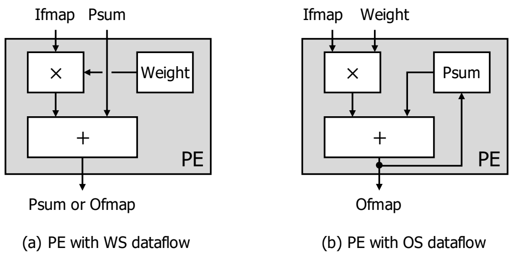

Fig. 6. PE designs with two different dataflows

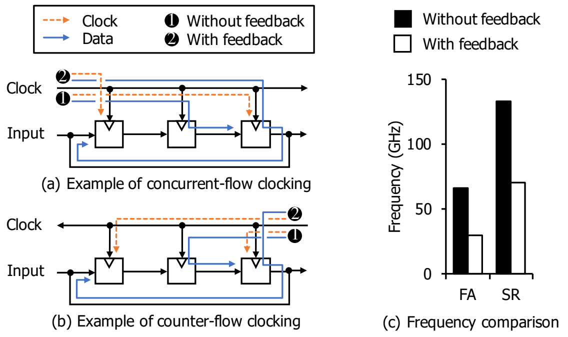

Fig. 7. Feedback loop's impact on the frequency of SFQ circuits

B. PE design

For the SFQ-friendly PE design, we identify the most suitable dataflow by carefully considering the SFQ logic's circuit-level characteristics. Among three major dataflows in a 2D systolic network, *Weight Stationary* (WS), *Output Stationary* (OS), and *Input Stationary* (IS) [31], [32], we focus on WS and OS because the PE with IS has almost the same hardware structure as the PE with WS. Fig. 6(a) shows the PE with WS dataflow, where PE holds a weight in its register, multiplies it with the input feature map data (ifmap), and adds the result to the partial sum input (psum). On the other hand, the PE with OS dataflow has a feedback loop consisting of the adder and its register, and continuously accumulates the partial sums to generate final output feature map data (ofmap) (Fig. 6(b)).

Among these two PE designs, we choose the PE with WS to maximize the clock frequency because it does not include any feedback loop. Unlike the CMOS technology, the existence of the loop significantly degrades the SFQ circuit's frequency as the loop enforest the slower clocking scheme.

Fig. 7 provides the example with two representative clocking schemes: (a) concurrent-flow clocking and (b) counter-flow clocking. In our example, we show how the SFQ circuit's frequency is affected by the feedback loop. As Fig. 7(a)❶ shows, when there is no feedback loop, SFQ circuits can hide the data propagation delay by flowing the clock pulse along with the data. However, such clocking cannot be utilized when the circuit includes the feedback loop. In fact, the circuit's frequency is significantly reduced because the next clock pulse should wait for a very long data transfer through the feedback path (Fig. 7(a)❷). On the other hand, we can resolve this problem with the counter-flow clocking, which can perfectly hide the data feedback delay (Fig. 7(b)❷). However, the frequency of the counter-flow clocked circuit is much lower than that of the concurrent-clocked circuit without feedback. Such difference is due to the unhidden feed-forward delay of the counter-flow clocking (Fig. 7(b)❶).

Fig. 7(c) shows the feedback loop's impact on the SFQ circuit's clock frequency by running JSIM [29] simulations with simple example circuits, a full adder (FA), and a shift register (SR). For the circuits without the feedback loop and with the loop, we apply the concurrent-flow clocking and counter-flow clocking, respectively. As Fig. 7(c) clearly shows, the existence of the feedback loop significantly degrades the clock frequency, from 66 GHz to 30 GHz in FA and from 133 GHz to 71 GHz in SR. Thus, we conclude that the PE design without a feedback loop, PE with WS, is a more SFQ-friendly choice and adopt it as our PE design.

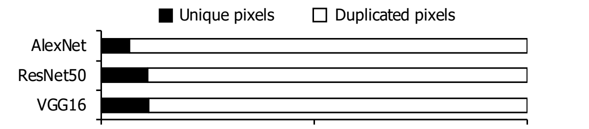

Fig. 8. Data ratio breakdown for unique and duplicated ifmap pixels

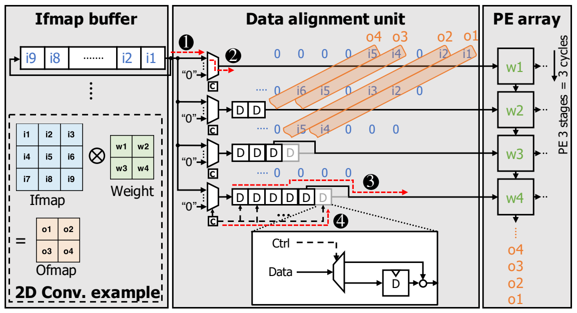

Fig. 9. Data alignment unit's structure with the working example

C. Data alignment unit design

In the SFQ-based NPU adopting systolic network and WS dataflow, the ifmap buffer can suffer from a large amount of duplicated data. As the ifmap buffer is the shift-register-based memory, each ifmap buffer row dedicatedly feeds data to the corresponding PE array row. However, in CNN execution, weights mapped to the adjacent PE array rows require partly the same ifmap data due to the weight sharing property of CNN. Therefore, the duplicated data significantly wastes the buffer capacity if adjacent ifamp buffer rows hold all ifmap data shared across the different weights. Fig. 8 clearly shows that the amount of duplicated data can be over 90% for three CNN networks. Note that such a massive waste of on-chip buffer capacity incurs a severe off-chip memory pressure.

To resolve the problem, we design a data alignment unit (DAU), which replicates and forwards data to the appropriate PE rows at exact timing. Fig. 9 shows the DAU's structure with a working example that runs a simple 2D-convolutional operation. Our DAU consists of sets of a selector, a controller, and cascaded special DFFs for each PE row. The DAU operates in two steps: 1) data selection and 2) timing adjustment.

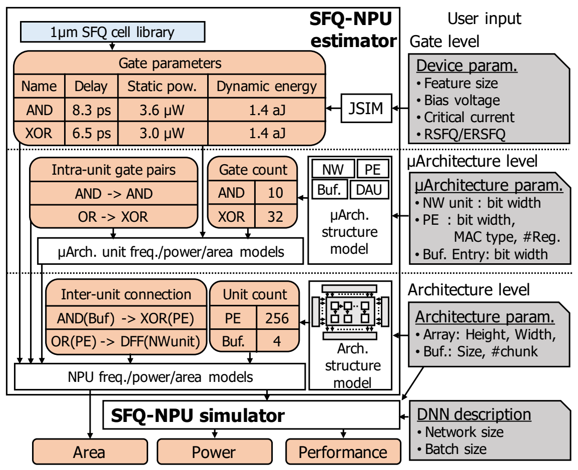

Fig. 10. SFQ-NPU overview

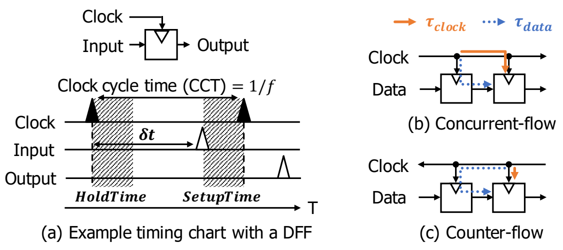

Fig. 11. Frequency model illustration

Data selection: Before starting computation, each ifmap buffer row dedicatedly holds data for a given ifmap channel. First, each ifmap buffer row provides its data to all DAU rows through a splitter tree, where each DAU row is dedicated to a single PE row's weight. For example, nine ifmap pixels are transferred to all four DAU rows in Fig. 9 (❶). Next, the selector in each DAU row selectively takes the required input for the weight mapped in the corresponding PE array row. The first row in Fig. 9 takes only i1, i2, i4, and i5, and 0 for others as a bubble to avoid the computation stall (❷). The bubbles are filtered at the end of computation by using a valid bit. For such data selection, the controller in each DAU row dynamically generates control signals. Note that the controllers can identify whether the given input is required or not based on the DNN layer configuration and current weight mapping information (e.g., current ifmap and weight pixel index).

Timing adjustment: To adjust the arrival timing of selected ifmap pixels, DAU utilizes the cascaded special DFFs with different lengths. By using the DFFs, each DAU row delays to feed the data because the computed psum in the above PE and the ifmap data should simultaneously arrive at the PE. For example, if the PE consists of three pipeline stages, the inputs from the second row should be delayed at most 2 ($= 3 - 1$) cycles. In fact, our 8-bit PE consists of 15 pipeline stages. Also, we bypass some DFFs when the adjacent PE rows map the weights with different row index. For example, as the weight's row index mapped to the third PE increases from that of the second PE (from 1 (w2) to 2 (w3)), we should bypass one DFF for the correct operation (❸). To support the bypassing, our special DFF has a bypassing line, whose control signal is statically determined by weight filter width, strides, and current weight index (❹).

IV. Simulation Framework

In this section, we describe our architectural simulation framework, *SFQ-NPU*, to explore and optimize the SFQ-based NPU architecture. Fig. 10 shows the overview of SFQ-NPU, which consists of two simulation engines: *SFQ-NPU estimator* and *SFQ-NPU simulator*. SFQ-NPU estimator takes device-level, microarchitecture-level, architecture-level information as inputs, and derives the frequency, power, and area of the target NPU design. Based on the obtained frequency and power information, the SFQ-NPU simulator reports the effective performance and power consumption by simulating target DNN applications. In the following sections, we explain the implementation details of each engine.

A. SFQ-NPU estimator

To carefully consider the SFQ logic's unique features ranging from the device to architecture, our SFQ-NPU estimator takes a strategy of three-layer abstraction: gate-level, microarchitecture-level, and architecture-level estimation.

1) Gate-level estimation: The gate-level estimation layer accurately provides the timing parameters (i.e., *SetupTime*, *HoldTime*, and delay), the power information (i.e., static power and access energy), and the area for all SFQ logic gates and wire cells with the given device parameters (e.g., bias voltage, critical current). The gate models are compatible with two SFQ technologies; rapid single-flux-quantum (RSFQ) [3], [13]–[15], and energy-efficient RSFQ (ERSFQ) [11]. RSFQ is the most practical and proven technology in the successful demonstrations, whereas ERSFQ is a promising technology that completely excludes the static power dissipation. The only difference is how to supply the DC bias current, i.e., RSFQ uses the bias resistors; on the other hand, ERSFQ uses the bias JJs.

For the RSFQ gates, we extract all gate parameters by running JSIM [29] simulations with RSFQ cell library for AIST 1.0μm fabrication process technology [9]. For example, the access energy is derived by taking an average of the dynamic energy for all the possible states. Besides, we calculate each gate's area based on its number of JJs.

On the other hand, we estimate gate parameters of ERSFQ based on those of RSFQ gates due to the lack of fabrication information (or cell library) about ERSFQ technology. Specifically, the timing parameters and area of ERSFQ gates are assumed to be the same as those of RSFQ because all their gate structures are the same, except for the bias current supply line. Meanwhile, we estimate the access energy and static power of ERSFQ gates as twice as that of RSFQ and zero, respectively. Note that the difference in both access energy and static power originates from the JJ-based DC biasing scheme [11].

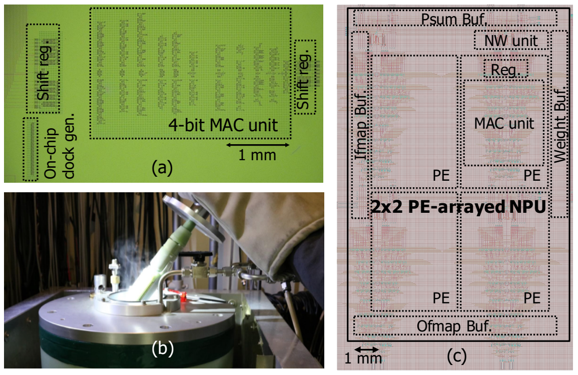

Fig. 12. Model validation setup (a) Chip microphotograph of 4-bit MAC unit (b) 4 K measurement setup (c) Layout of the 2 × 2 PE-arrayed NPU

2) Microarchitecture-level estimation: This abstraction layer estimates the frequency, static power, access energy, and area of each microarchitectural unit designed in Section III (i.e., NW units, PE, DAU, and on-chip buffers). For the accurate estimation, the microarchitecture-level layer first generates the intra-unit gate pair and the gate count information for each unit based on the gate-level circuit structure model. The intra-unit pair and the gate count are utilized to derive each unit's frequency and power/area, respectively.

$$f = 1/\text{CCT} = 1/(\mathit{SetupTime} + \text{Max}(\mathit{HoldTime}, \delta t)) \quad (1)$$

With the gate-level pipelining nature considered, the microarchitecture-level frequency model calculates the frequency of all gate pairs in the target unit and takes the minimum value as the unit's frequency. Fig. 11 and Eq. (1) illustrate the model to calculate the frequency of one gate pair, where δt is the difference between the data and clock propagation delay (i.e., $\tau_{data} - \tau_{clock}$). Note that Eq. (1) is the direct translation of the two timing constraints: 1) data should arrive after the HoldTime and 2) the next clock pulse should arrive after SetupTime elapsed from the data arrival.

To reflect the real-world SFQ circuit design practice, we model both representative clocking schemes, concurrent-flow clocking (Fig. 11(b)) and counter-flow clocking (Fig. 11(c)). As explained in Section III-B, to achieve high frequency, we apply the concurrent-flow clocking to all circuits without the feedback loop. Also, we include the frequency-enhancing technique called clock skewing, which minimizes δt by adjusting the length of data and clock line. On the other hand, we apply the counter-flow clocking to the circuits with the feedback loop, such as shift-register-based on-chip buffers.

Meanwhile, the microarchitecture-level power and area models calculate the power information (i.e., static power and access energy) and area of each unit based on the gate count information and the gate parameters.

3) Architecture-level estimation: The architecture-level layer reports the final estimation results regarding the area, static power, access energy, and clock frequency of the target NPU configuration. For the accurate prediction, this layer not only integrates the microarchitecture-level estimations based on the unit counts but also considers the inter-unit gate pair information. For instance, we calculate all the inter-unit communication latency based on the interfacing gates' timing parameters and include them to derive the highest frequency in NPU. Also, based on the estimated unit-to-unit distance, we calculate the area of wire cells required to connect each unit and include it to the final area estimation.

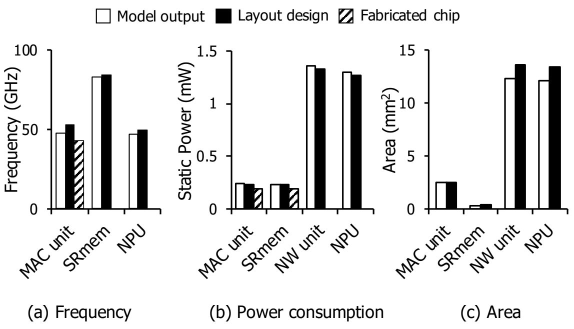

Fig. 13. Model validation result

4) Model validation: We carefully validate our SFQ-NPU estimator in terms of the frequency, power, and area by comparing it with a fabricated 4-bit MAC unit (Fig. 12(a)) measured in the 4 K environment (Fig. 12(b)). Also, we compare the model's output with the post-layout characterizations for 8-bit 8-entry shift-register-based memory (SRmem), 8-bit NW unit, and a 4-bit 2 × 2 PE-arrayed NPU (Fig. 12(c)). Note that our gate-level estimation is already validated in its accuracy because it is based on the validated cell library, which succeeded in fabricating real chips for many times.

First, we validate our microarchitecture-level estimation with MAC unit, SRmem, and NW unit. Note that there is no frequency result for a single NW unit because it only consists of DFF-splitter pairs. As Fig. 13 shows, SFQ-NPU estimator accurately predicts all the frequency, power, area for each unit with the average error of 5.6%, 1.2%, and 1.3%, respectively.

Next, we also validate the architecture-level estimation with the 4-bit 2 × 2 PE-arrayed NPU design (Fig. 12(c)). Even though it is a small NPU prototype, the layout design is enough to show the inter-unit connections' impact on the frequency due to the 2D-systolic network's scalable structure. As shown in Fig. 13's NPU, our model well matches the frequency, power, and area result of the post-layout simulation with the error of 4.7%, 2.3%, and 9.5%, respectively.

B. SFQ-NPU simulator

For the given SFQ-based NPU design running DNN applications, SFQ-NPU simulator reports the effective performance and power consumption based on the obtained frequency and power information from the SFQ-NPU estimator. As the first step for the simulation, SFQ-NPU simulator analyzes all required weight mappings by taking the DNN description file (i.e., ifmap window size, filter window size, the number of filters, the number of strides) and architecture description file as inputs. We use a batch of typical DNN input images (224 × 224 × 3) as inputs. Next, for each weight mapping,

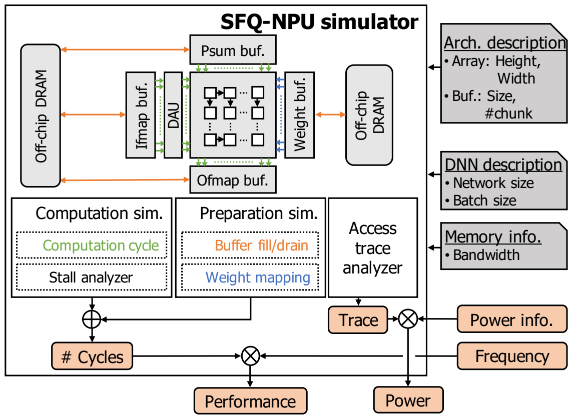

Fig. 14. SFQ-NPU simulator overview

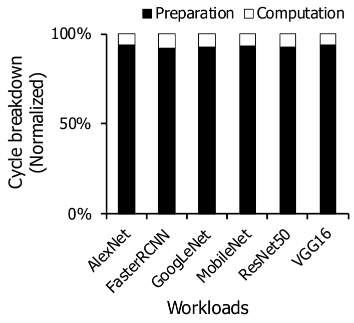

Fig. 15. Baseline's cycle breakdown normalized for each CNN workload

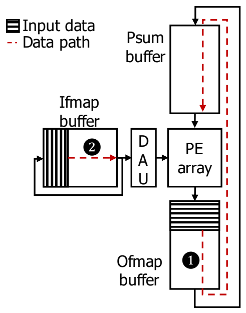

Fig. 16. Example data path of on-chip buffers

the simulator runs the cycle-based simulation to calculate the consumed cycles and the activated cycles for each hardware unit. During the simulation, the simulator also models the memory stall incurred by limited memory bandwidth by taking memory bandwidth as its input. Finally, the SFQ-NPU simulator aggregates the result of each mapping and reports the performance numbers (e.g., latency, throughput, PE utilization) and the power values (i.e., static power, dynamic power).

V. Optimizing SFQ-based NPU design

In this section, with our simulation framework, we architect an extreme-performance SFQ-based NPU by taking the best advantage of SFQ technology. Similar to other devices, SFQ-based NPUs have a large design space that cannot be easily explored even with the modeling tool. Therefore, we start from our baseline SFQ-based NPU architecture (Section III) and identify the major performance bottlenecks to be resolved (Section V-A). Next, we propose an optimal SFQ-based NPU architecture, *SuperNPU*, which resolves the identified bottlenecks with the architecture-level solutions (Section V-B).

In the following sections, we conduct performance analyses by running six CNN workloads (i.e., AlexNet [33], FasterR-CNN [34], GoogLeNet [35], MobileNet [36], ResNet 50 [37], VGG16 [38]) with our simulation framework. As the input fabrication process information, we take the currently available AIST 1.0 μm process to show the SFQ technology's performance potential conservatively[1]. Moreover, we assume the memory bandwidth of 300 GB/s, which is the typical value of HBM used by the recent TPUv2 [39].

A. Design implications for the SFQ-optimal NPU architecture

1) Baseline SFQ-based NPU setup: We first introduce performance-side design implications by conducting analyses with the baseline SFQ-based NPU design introduced in Section III (hereinafter called *Baseline*). To show the implications, we start from Baseline following the TPU core's [40] architectural specification for three reasons. First, we target the server-side NPU due to the need for cryogenic cooling support. Second, Baseline has a similar hardware structure with the TPU core (i.e., weight-stationary dataflow and systolic-array network). Third, its estimated area might be comparable to the TPU core (< 330 mm^2) if the SFQ circuits or JJs are equivalently scaled to 28 nm as CMOS technology used in the TPU design[2]. We summarize Baseline's specification including the architectural configurations in Table I.

As Table I shows, the peak performance of Baseline is significantly high, 3366 TMAC/s, with the clock frequency over 52 GHz. However, we find that the effective performance of Baseline is only about 6.45 TMAC/s on average, which is even lower than 0.2% of its peak performance (Fig. 17). In the following subsections, we identify the performance bottlenecks and set the design directions to resolve the identified challenges.

2) Bottleneck 1. Huge data movement overhead: We first emphasize the importance of reducing the overhead of data movement among different on-chip buffers and within a single buffer. Fig. 15 shows the Baseline's cycle breakdown normalized for each CNN workload. As the figure clearly indicates, the Baseline's performance is highly dominated by the preparation step (above 90%), which moves data to the appropriate location before starting computation. Based on this analysis, we identify the huge data movement overhead as the first performance bottleneck.

Fig. 16 shows the data movement overhead with the example showing the data location right after the end of computation for one weight mapping. First, the calculated partial sums in the ofmap buffer should move to the psum buffer when they need to be accumulated with the next computation result (Fig. 16 ❶). In this case, the Baseline should consume a huge amount of cycles corresponding to the sum of two buffers' length, 65,536 cycles (= 16 MB $\div$ 256 B/cycle), due

[1]An i-line stepper with a wavelength of 365 nm (introduced to the market in the mid-1990s) is used in the fabrication. The state-of-the-art steppers using KrF or ArF excimer lasers would allow the fabrication of ultrafine Josephson junctions and patterns.

[2]To the best of our knowledge, no study mentions the physical limit of JJ scaling. On the other hand, there is the scaling rule that the frequency increases in proportion to the reduction rate of JJ until 200 nm [41], and T-flip-flop (TFF) has successfully demonstrated at up to 770 GHz with the technology [42]. Moreover, there are several schemes to reduce the SFQ cell size without the JJ scaling, such as the introduction of shunt-resistor-free junctions [43], vertically-stacked junctions [44], multi-layer process technology with high-inductance layers [45] and new materials.

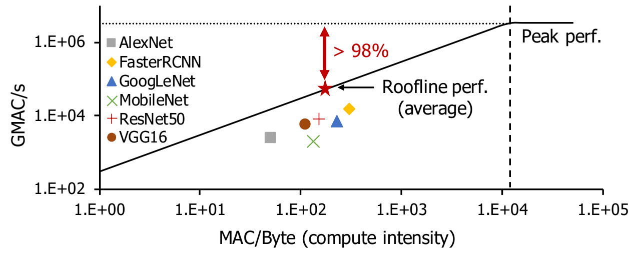

Fig. 17. Limited performance improvement in Baseline due to the low computational intensity with a single batch

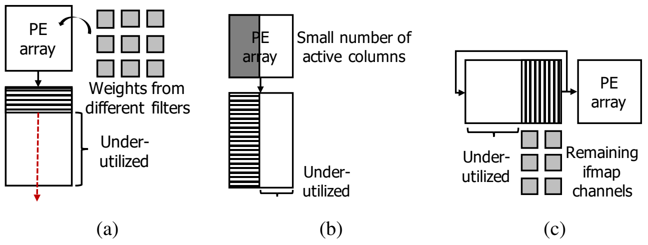

Fig. 18. On-chip buffer under-utilization in terms of (a) ofmap buffer's length, (b) ofmap buffer's width, and (c) ifmap buffer's length

to the shift-register-based memory implementation. Also, the ifmap buffer suffers from a similar situation to move the data from its tail to the head when the used ifmap data is required for the next computation again (Fig. 16 ❷). Therefore, we conclude that we should minimize the wasteful length of the data movement.

3) Bottleneck 2. Fast but idle computing units: Next, we emphasize the importance of improving the computing unit's (i.e., PE array) utilization. Fig. 17 shows the Baseline's roofline plot, which represents the highest achievable performance for a given computational intensity. In this work, we define computational intensity as the number of MAC operations executed with one weight data mapped on the PE. Note that it includes the impact of input batch size on the amount of data reuse.

With the roofline model, Fig. 17 shows the performance and computational intensity of each workload with a single input batch. Even though the Baseline's computing units are fast, they are mostly idle with the maximum PE utilization (= roofline performance ÷ peak performance) below 2% on average. The underutilization directly results from the workloads' low computational intensity and the relatively slow memory access (vs. 52 GHz computation speed). Therefore, we conclude that we should maximize the PE utilization by increasing the computational intensity.

4) Bottleneck 3. Waste of on-chip buffer capacity: Lastly, we highlight that it is crucial to resolve the on-chip buffer underutilization issue. To increase the computational intensity, it is required to increase the input batch size for DNN workloads. However, it is highly difficult for the Baseline to take larger batch sizes (i.e., more than one) without additional off-chip memory access because the on-chip buffer can be significantly underutilized.

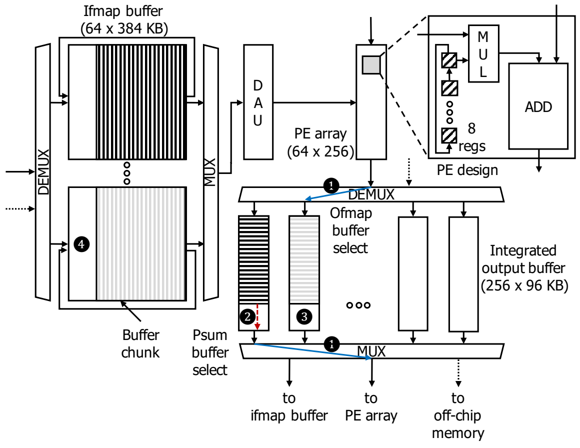

Fig. 19. SuperNPU overview

Fig. 18 shows the three scenarios to explain the buffer underutilization problem, which happens even with a single batch. First, even with the huge amount of empty space, the ofmap buffer should flush the data when the next computation is for the different set of output channels (Fig. 18(a)). Second, we waste the ofmap buffer's capacity when the number of active PE columns is smaller than the ofmap buffer's width (Fig. 18(b)). Third, we cannot map the remaining ifmap channels to the ifmap buffer because each buffer row is dedicated to a given ifmap channel (Fig. 18(c)). Therefore, we conclude that we should maximize the buffer utilization by resolving all the mentioned cases.

B. SuperNPU: SFQ-optimal NPU architecture

The design implications for optimizing SFQ-based NPU are summarized as follows. First, the optimal design should have a short path for the on-chip buffer data movement. Next, the optimal design should be able to take a large batch size without additional off-chip memory access. Lastly, to achieve the goal, the buffer underutilization problem also should be resolved.

Following all the implications, we design the optimal SFQ-based NPU architecture, *SuperNPU*, as shown in Fig. 19. First, SuperNPU has the optimized on-chip buffer architecture where each on-chip buffer is divided into small chunks and connected by the multiplexer and demultiplexer trees. Note that SuperNPU does not have a separated psum buffer but integrates it with the ofmap buffer. Next, the PE array width of SuperNPU is reduced to 1/4, and the on-chip buffer capacity becomes twice compared to the Baseline. Finally, each PE in SuperNPU has eight registers, so a PE can hold eight different weights simultaneously. In the following subsections, we explain how each design choice resolves the identified performance bottlenecks in detail.

1) Optimized on-chip buffer architecture: The optimized on-chip buffer architecture meets the design implications as follows. First, SuperNPU removes the unnecessary data movement and increases the effective buffer capacity by integrating the psum buffer and the ofmap buffer. For example, SuperNPU does not have to move the calculated psum to other buffer

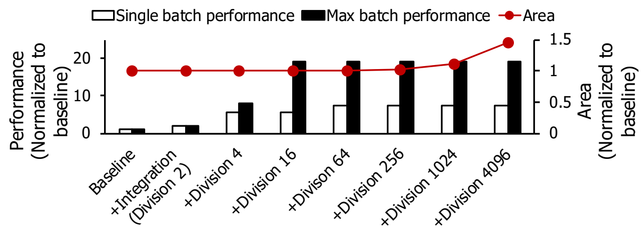

Fig. 20. Performance impact and area overhead of the buffer optimizations

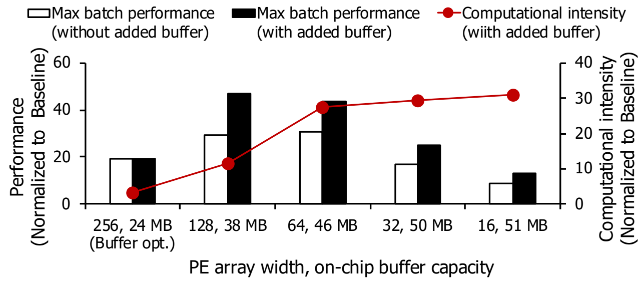

Fig. 21. Performance and computational intensity with resource balancing

chunks. Instead, it just selects the buffer chunk with the psum data as the psum buffer, and one of the empty buffer chunks as the ofmap buffer (Fig. 19 ❶). Moreover, by individually selecting the ofmap buffer and psum buffer through the separated multiplexer/decoder, we can flexibly utilize the integrated buffer. Fig. 20 shows the performance impact of buffer integration. To match the capacity of input and output buffer, we adjust each buffer's capacity to 12 MB.

Next, by dividing each buffer to several small buffer chunks, SuperNPU significantly reduces data movement overhead in the on-chip buffer (Fig. 19 ❷). Our simulation results (Fig. 20) show the performance impact of dividing buffer with the various degree of division. With the increasing division degree, the single batch performance continuously increases and achieves 6.26 times higher performance compared to Baseline, from the division degree of 64. Such performance improvement mainly originates from the shortened buffer length, which correspondingly reduces the data-shifting cycles. This result indicates that the buffer division successfully resolves the performance bottleneck resulting form data movement overhead.

The buffer division also mitigates the buffer underutilization issues. For example, the NPU does not need to flush the calculated data in ofmap buffer if there are remaining buffer chunks to hold it, as shown in Fig. 19 ❸ (i.e., resolves Fig. 18(a)). Moreover, divided ifmap buffer can hold many input channels, corresponding to the PE array height multiplied by the number of buffer chunks in maximum, as shown in Fig. 19 ❹ (i.e., resolves Fig. 18(c)). Fig. 20 shows the impact of improved buffer utilization on the performance with the maximum batch size for each workload. The performance continuously increases and achieves 20 times higher performance from the division degree of 64. Based on the result, we set the buffer division degree as 64 because the performance is saturated. Note that further division incurs the exponentially increasing area overhead of multiplexer/decoder (Fig. 20).

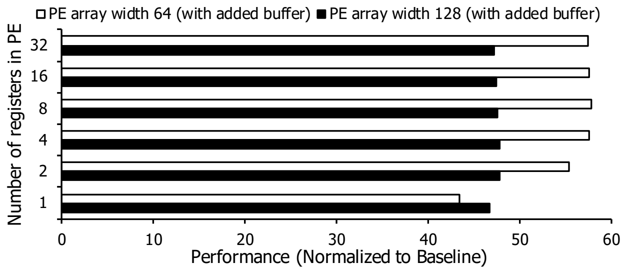

Fig. 22. Performance impact of number of registers in PE

2) Efficient resource balancing: Next, in SuperNPU, we increase the on-chip buffer capacity by reducing the number of PEs in the PE array. The insight for this design choice is that there is more room to increase the computational intensity by sacrificing the excessively-high peak performance. Note that we cannot utilize the current peak performance without increasing the computational intensity furthermore (Fig. 17).

When reducing the number of PEs, we do not reduce the height of the PE array but its width, to simultaneously resolve the remaining buffer underutilization issue. Even with the optimized on-chip buffer architecture, we still cannot fully utilize the output buffer due to the problem shown in Fig. 18b. However, this underutilization issue naturally disappears with the reduced PE array width because the width of the output buffer correspondingly decreases. While reducing the PE array width, we correspondingly divide the integrated output buffer further (i.e., division degree from 64 to 256) to maintain the length of each buffer chunk.

Fig. 21 shows the performance impact of resouce balancing which increases the on-chip buffer capacity while reducing the PE array width. In our analysis, we start from the design with the optimized on-chip buffer (256, 24 MB (Buffer opt.)). Max batch (without added buffer) indicates that the NPU with the given PE array width and fixed 24 MB on-chip buffer (i.e., no additional capacity). On the other hand, Max batch (with added buffer) has the increased buffer capacity corresponding to the values shown in the graph. We derive each on-chip buffer capacity based on the area occupancy of the PE array and the on-chip buffers.

First, Max batch performance (without added buffer) increases to around 30 times higher compared to Baseline, even though the peak performance decreases. This result indicates that the PE array width reduction itself increases the computational intensity by improving the buffer utilization. Next, with the increased buffer capacity, performance (Max batch performance (with added buffer)) is further improved to 47 times and 42 times higher compared to Baseline in the PE-array width of 128 and 64, respectively. Although the optimal PE array width is 128 in this analysis, the design with the PE array width of 64 has more room for the performance improvement with a much higher computational intensity. Therefore, we focus on these two NPU architectures in the following subsection.

TABLE I
EVALUATION SETUP

	TPU	Baseline	Buffer opt.	Resource opt.	Super-NPU
PE array width	256	256	256	64	64
PE array height	256	256	256	256	256
Ifmap buf.	24 MB	8 MB	12 MB	24 MB	24 MB
Ofmap buf.		8 MB	12 MB	24 MB	24 MB
Psum buf.		8 MB			
Weight buf.		64 KB	64 KB	16 KB	128 KB
# regs in PE	1	1	1	1	8
Frequency (GHz)	0.7	52.6	52.6	52.6	52.6
Peak perf. (TMAC/s)	45	3366	3366	842	842
Area (mm^2) (28nm)	<330	∼283	∼285	∼298	∼299

TABLE II
WORKLOAD SETUP (BATCH SIZE)

	TPU	Baseline	Buffer opt.	Resource opt.	Super-NPU
AlexNet	22	1	15	30	30
FasterRCNN	20	1	3	30	30
GoogLeNet	20	1	3	30	30
MobileNet	20	1	3	30	30
ResNet50	20	1	3	30	30
VGG16	3	1	1	7	7

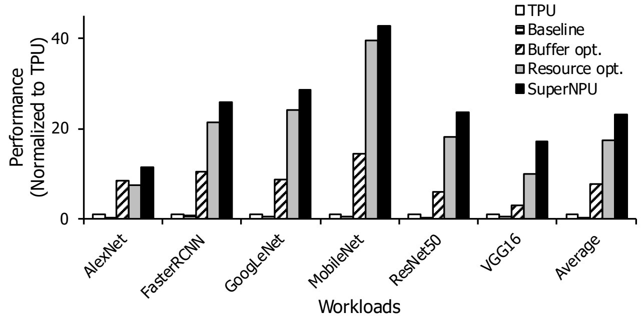

Fig. 23. Performance evaluation

3) Increasing the number of registers in PE: Finally, to further improve the performance, we increase the number of weight registers in PE. With the larger number of weight registers in each PE, SuperNPU increases the PE utilization by filling several PE pipeline stages with a single ifmap data. For example, if each PE holds four different weights from different weight filters, PE can compute four different MAC operations with one ifmap pixel.

Fig. 22 shows the performance impact of the number of registers in PE on two chosen designs (128-width and 64-width PE arrays). First, the PE array width of 128 (with added buffer) cannot improve its performance further due to its lower computational intensity, i.e., performance is bounded by the relatively slow memory access. On the other hand, in the PE array width of 64, we get much higher performance improvement thanks to its high computational intensity (Fig. 21). Based on this performance analysis, we take the PE array width of 64, the on-chip buffer capacity of 46 MB, and eight registers per PE in SuperNPU.

VI. EVALUATION

In this section, we show the system-level performance and power efficiency of SuperNPU by pointing out the impact of each optimization scheme step by step. We first introduce our evaluation methodology (Section VI-A). Next, we evaluate SuperNPU in terms of the performance (Section VI-B) and performance per Watt (Section VI-C).

A. Evaluation methodology

1) Evaluation setup: We evaluate SuperNPU by comparing it with the TPU core [40], one of the most representative server-side DNN accelerators. To estimate the TPU core's performance, we use SCALE-SIM [32], systolic-array-based cycle-accurate DNN accelerator simulator, with the hardware specification summarized in Table I. For the TPU's power consumption, we take 40 W as its average value based on [40]. Also, we set the memory bandwidth of TPU core as 300 GB/s following TPUv2 board specification [39].

In our performance evaluation, we explicitly show the performance impact of each optimization step by accumulatively evaluating three intermediate SFQ-based NPU architecture designs: architecture introduced in Section V-A (Baseline), with optimized on-chip buffer (Buffer opt.), and with reduced PE array and larger buffer capacity (Resource opt.). We also set the memory bandwidth for SFQ-based NPU designs as same as the TPU core, 300 GB/s. As the fabrication technology, we take AIST 1.0 μm process as same as in Section V. We summarize the setup for each architecture in Table I.

For the evaluation, we use six representative CNN workloads that have various application characteristics (e.g., computational intensity, layer configurations). We set each workload's batch size as the maximum value, which can be held by a given on-chip buffer capacity without additional off-chip memory access. For example, for TPU, we set the batch size of AlexNet as 22 because its largest layer's (second layer) input/output data size is 1.05 MB, where 22 input batches can be held within 24 MB in maximum. Our batch setup is conservative because there is room to increase the batch size while improving performance. We summarize each workload's batch size setup for all NPU designs in Table II.

B. Performance evaluation

Fig. 23 shows the speed-up of SFQ-based NPU designs. The speed-up is calculated by the throughput (i.e., TMAC/s) normalized to that of the TPU. In our performance evaluation, SuperNPU achieves the significant speed-up (23 times) as three architectural optimizations are applied one by one.

The baseline SFQ-based NPU design (Baseline) shows the poor performance, only 40% of TPU's performance on average. The low performance mainly results from the data movement overhead between the on-chip buffers, idle PEs, and low on-chip buffer utilization, as identified in Section V-A.

Note that we cannot put even one more input image without off-chip memory overhead in Baseline (Table I).

With the optimized buffer architecture, Buffer opt. achieves the speed-up of 7.7 times on average by resolving the performance bottlenecks in Baseline. The divided and integrated on-chip buffers significantly improve the performance of all workloads by removing the wasteful on-chip buffer data movement. Furthermore, by mitigating the buffer underutilization (Fig. 18(a), (c)), most workloads can take benefit of larger batch size (15 in AlexNet and 3 for others except for VGG16).

In Resource opt., average speed-up reaches 17.3 times, mainly thanks to the much higher computational intensity in all workloads. For example, FasterRCNN, GoogLeNet, and MobileNet show the drastically increasing performance in this optimization step with the 10 times larger batch size compared to Buffer opt. Among them, MobileNet shows the highest speed-up (around 40 times) because of its small number of weight filters, usually lower than 64. Note that even with the reduced PE array width, there is no performance degradation in layers consisting of few weight filters. On the other hand, the performance of AlexNet is reduced in Resource opt. due to the reduced peak performance. This is the exact opposite case compared to MobileNet. However, the performance degradation is almost mitigated by the doubled batch size in AlexNet.

Finally, with the increased number of registers in PE, SuperNPU boosts all workloads over 10 times, 23 times on average, and 42 times in MobileNet. In the previous step, the layers consisting of less than 64 filters suffer from performance degradation corresponding to the reduced PE array width. However, in SuperNPU, we mitigate performance degradation by increasing the number of weights in PE (i.e., improving the PE pipeline utilization). For example, in AlexNet, we compensate the performance reduction in Resource opt. by filling the PE pipeline several times with the single input data. As a result, SuperNPU not only successfully shows the performance potential of SFQ-based NPU design but also clearly indicates the importance of optimizing SFQ processors in the right direction.

C. *Power-efficiency evaluation*

We evaluate SuperNPU's power-efficiency for two different SFQ device technologies, RSFQ [3], and ERSFQ technology [11]. Table III shows the power consumption and performance per Watt (i.e., power-efficiency) for SuperNPUs and TPU core. Power-efficiency values are normalized to that of TPU core, which dissipates 40 W in its operation [40]. Also, to include the cooling cost for the 4 K, we set the cooling cost as the 400 times of NPU's power consumption following [46]. In our power-efficiency evaluation, SuperNPU shows 490 times higher power-efficiency provided the free cooling, with ERSFQ technology.

With RSFQ device technology, SuperNPU consumes 964 W, which is infeasible power consumption, due to its huge static power dissipation. Even though its low switching energy, RSFQ technology requires to supply DC-biased current (i.e., DC-biased voltage with bias resistor) for each JJ for the operation (2.5 mV and 70 μA, respectively). Recently, although several device-level optimizations are proposed to reduce the static power, 960 W is too high to make this technology feasible. As Table III shows, the power efficiency of RSFQ-SuperNPU is not much lower than TPU (95%) thanks to the 23 times of speed-up. However, with the cooling cost included, the normalized power efficiency value becomes 0.002.

TABLE III
POWER-EFFICIENCY EVALUATION

	Power (W)	Performance/W (Normalized to TPU)
TPU	40	1
RSFQ-SuperNPU (w/o cooling cost)	964	0.95
RSFQ-SuperNPU (w/ cooling cost)	3.8x10^5	0.002
ERSFQ-SuperNPU (w/o cooling cost)	1.9	490
ERSFQ-SuperNPU (w/ cooling cost)	751	1.23

On the other hand, ERSFQ-SuperNPU consumes only 1.9 W because there is no static power consumption in ERSFQ technology [4], [11]. As ERSFQ provides the bias current using JJ with inductors (i.e., bias resistors are replaced to bias JJ), it does not consume static power, but the number of JJs increases (i.e., twice higher dynamic energy per switching). However, thanks to the significantly low switching energy of JJs, ERSFQ-SuperNPU achieves 490 times higher power efficiency compared to the TPU with free cooling provided. Even including the 400 times of cooling cost, ERSFQ-SuperNPU attains 1.23 times higher power efficiency. That is, with ERSFQ-SuperNPU, architects can increase the server-side NPU's performance to 23 times with 490 times higher power-efficiency with assuming free cooling.

VII. RELATED WORK

Exploiting emerging devices is a critical challenge to design next-generation computer systems. Many researchers have so far been proposed and discussed such novel architectures. In this section, we discuss prior work from the viewpoint of neural network (NN) acceleration and superconducting computing to clarify the novelty of this paper.

A lot of researchers have proposed NN accelerators for power-efficient processing [47]–[51]. A representative approach exploiting an emerging device is to implement memristor-based dot-product operations [52]–[54]. Another direction is to introduce PIM (Processor-In-Memory) and die-stacking technologies [53], [55]–[57]. A more challenging attempt is to apply nanophotonic technology [58]–[62], or superconducting SQUIDs [63], [64] to NN operations. Unlike previous researches, this paper focuses on SFQ circuits and achieves better performance in both the computing power and energy efficiency than the conventional CMOS designs even with the cooling penalty.

Prior researches regarding SFQ demonstrated its significant potential from the viewpoint of circuit implementation. Regardless of its high-speed operations, unfortunately, their throughput was quite low due to the simple but bit-serial designs [65], [66]. Although a recent design successfully demonstrated high-throughout bit-parallel multiplier [5], [67], it is still not clear whether or not the SFQ technology can realize at the system level. Swamit et al. proposed an accelerator for SHA-256 for low latency operations [26]. Tzimpragos et al. introduced an interesting idea that attempts to apply the concept of the delay-based logic (race logic) [68] to SFQ [69]. Another relating proposal is to use not SFQ but AQFP [70] for stochastic computing [71]. Our target is to explore and optimize the architecture of the SFQ-based NPU and to clarify the system-wide potential. To achieve this goal, we have developed a simulation framework, including power/frequency/area models validated based on physical chip fabrication or post-layout characterizations. Also, we have deeply evaluated and presented the significant potential of SFQ devices at the architectural level.

VIII. Conclusion

Superconductor SFQ technology is a highly promising solution in post-Moore's era. However, SFQ computing has not yet been realized because of the lack of understanding of SFQ technologies' potentials and limitations. This paper resolves the challenge as follows. First, we implement and validate an SFQ-based NPU modeling framework. Next, by using the tool, we identify critical challenges in architecting an SFQ-based NPU. Finally, we present SuperNPU, our example SFQ-based NPU architecture, which effectively addresses the challenges at the architectural level. Our evaluation shows that the proposed design outperforms a conventional state-of-the-art NPU by 23 times with comparable power efficiency even including the extremely expensive cooling costs. We believe that our design methodology can also be applied to architect other SFQ-based architectural units.

Acknowledgments

This work was supported by JST-Mirai Program Grant Number JPMJMI18E1, JSPS KAKENHI Grant Numbers JP19H01105, JP18H05211, JP18J21274. The circuit is designed with the support by VDEC of the University of Tokyo in collaboration with Cadence Design Systems, Inc., and fabricated in the CRAVITY of AIST. We also appreciate the support from National Research Foundation of Korea (NRF) grant funded by the Korean Government (NRF-2019R1A5A1027055, NRF-2020M3H6A1084857).

References

[1] R. R. Schaller, "Moore's law: past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[2] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[3] K. K. Likharev and V. K. Semenov, "RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems," *IEEE Transactions on Applied Superconductivity*, vol. 1, no. 1, pp. 3–28, March 1991.

[4] O. A. Mukhanov, "Energy-efficient single flux quantum technology," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 760–769, 2011.

[5] I. Nagaoka, M. Tanaka, K. Inoue, and A. Fujimaki, "A 48GHz 5.6mW gate-level-pipelined multiplier using single-flux quantum logic," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 460–462.

[6] I. Nagaoka, M. Tanaka, K. Sano, T. Yamashita, A. Fujimaki, and K. Inoue, "Demonstration of an energy-efficient, gate-level-pipelined 100 TOPS/W arithmetic logic unit based on low-voltage rapid single-flux-quantum logic," in *2019 IEEE International Superconductive Electronics Conference (ISEC)*, 2019, pp. 1–3.

[7] G. Pasandi, A. Shafaei, and M. Pedram, "SFQmap: A Technology Mapping Tool for Single Flux Quantum Logic Circuits," *arXiv e-prints*, p. arXiv:1901.00894, Jan. 2019.

[8] M. Pedram and Y. Wang, "Design automation methodology and tools for superconductive electronics," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.

[9] S. Nagasawa, K. Hinode, T. Satoh, M. Hidaka, H. Akaike, A. Fujimaki, N. Yoshikawa, K. Takagi, and N. Takagi, "Nb 9-layer fabrication process for superconducting large-scale SFQ circuits and its process evaluation," *IEICE Transactions on Electronics*, vol. E97.C, no. 3, pp. 132–140, 2014.

[10] K. Nakajima, Y. Onodera, and Y. Ogawa, "Logic design of Josephson network," *Journal of Applied Physics*, vol. 47, no. 4, pp. 1620–1627, 1976. [Online]. Available: https://doi.org/10.1063/1.322782

[11] D. E. Kirichenko, S. Sarwana, and A. F. Kirichenko, "Zero static power dissipation biasing of RSFQ circuits," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 776–779, June 2011.

[12] M. H. Volkmann, A. Sahu, C. J. Fourie, and O. A. Mukhanov, "Implementation of energy efficient single flux quantum digital circuits with sub-aJ/bit operation," *Superconductor Science and Technology*, vol. 26, no. 1, p. 015002, 2013. [Online]. Available: http://stacks.iop.org/0953-2048/26/i=1/a=015002

[13] Y. Yamanashi, T. Nishigai, and N. Yoshikawa, "Study of LR-loading technique for low-power single flux quantum circuits," *IEEE Transactions on Applied Superconductivity*, vol. 17, no. 2, pp. 150–153, June 2007.

[14] N. Yoshikawa and Y. Kato, "Reduction of power consumption of RSFQ circuits by inductance-load biasing," *Superconductor Science and Technology*, vol. 12, no. 11, pp. 918–920, nov 1999. [Online]. Available: https://doi.org/10.1088%2F0953-2048%2F12%2F11%2F367

[15] M. Tanaka, M. Ito, A. Kitayama, T. Kouketsu, and A. Fujimaki, "18-GHz, 4.0-aJ/bit operation of ultra-low-energy rapid single-flux-quantum shift registers," *Japanese Journal of Applied Physics*, vol. 51, p. 053102, may 2012. [Online]. Available: https://doi.org/10.1143%2Fjjap.51.053102

[16] M. Dorojevets, P. Bunyk, and D. Zinoviev, "Flux chip: Design of a 20-GHz 16-bit ultrapipelined rsfq processor prototype based on 1.75-/spl mu/m lts technology," *IEEE Transactions on Applied Superconductivity*, vol. 11, no. 1, pp. 326–332, 2001.

[17] M. Dorojevets, Z. Chen, C. L. Ayala, and A. K. Kasperek, "Towards 32-bit energy-efficient superconductor RQL processors: The cell-level design and analysis of key processing and on-chip storage units," *IEEE Transactions on Applied Superconductivity*, vol. 25, no. 3, pp. 1–8, 2015.

[18] M. Bhushan, P. Bunyk, M. Cuthbert, E. P. DeBenedictis, M. Frank, and T. Humble, "Cryogenic electronics and quantum information processing," 6 2019.

[19] G.-M. Tang, P.-Y. Qu, X.-C. Ye, and D.-R. Fan, "Logic design of a 16-bit bit-slice arithmetic logic unit for 32-/64-bit RSFQ microprocessors," *IEEE Transactions on Applied Superconductivity*, vol. PP, pp. 1–1, 01 2018.

[20] Y. Yamanashi, T. Kainuma, N. Yoshikawa, I. Kataeva, H. Akaike, A. Fujimaki, M. Tanaka, N. Takagi, S. Nagasawa, and M. Hidaka, "100 GHz demonstrations based on the single-flux-quantum cell library for the 10 kA/cm^2 Nb multi-layer process," *IEICE Transactions on Electronics*, vol. 93, no. 4, pp. 440–444, apr 2010. [Online]. Available: https://ci.nii.ac.jp/naid/120006382470/

[21] K. Ishida, M. Tanaka, T. Ono, and K. Inoue, "Single-flux-quantum cache memory architecture," in *2016 International SoC Design Conference (ISOCC)*, 2016, pp. 105–106.

[22] S. Tahara, I. Ishida, Y. Ajisawa, and Y. Wada, "Experimental vortex transitional nondestructive read-out Josephson memory cell," *Journal of Applied Physics*, vol. 65, no. 2, pp. 851–856, Jan. 1989.

[23] G. Konno, Y. Yamanashi, and N. Yoshikawa, "Fully functional operation of low-power 64-kb Josephson-CMOS hybrid memories," *IEEE Transactions on Applied Superconductivity*, vol. 27, no. 4, pp. 1–7, 2017.

[24] M. Tanaka, M. Suzuki, G. Konno, Y. Ito, A. Fujimaki, and N. Yoshikawa, "Josephson-CMOS hybrid memory with nanocryotrons," *IEEE Transactions on Applied Superconductivity*, vol. 27, no. 4, pp. 1–4, 2017.

[25] I. M. Dayton, T. Sage, E. C. Gingrich, M. G. Loving, T. F. Ambrose, N. P. Siwak, S. Keebaugh, C. Kirby, D. L. Miller, A. Y. Herr, Q. P. Herr, and O. Naaman, "Experimental demonstration of a Josephson magnetic memory cell with a programmable π-junction," *IEEE Magnetics Letters*, vol. 9, pp. 1–5, 2018.

[26] S. S. Tannu, P. Das, M. L. Lewis, R. Krick, D. M. Carmean, and M. K. Qureshi, "A case for superconducting accelerators," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, ser. CF '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 67–75. [Online]. Available: https://doi.org/10.1145/3310273.3321561

[27] M. Tanaka, T. Kawamoto, Y. Yamanashi, Y. Kamiya, A. Akimoto, K. Fujiwara, A. Fujimaki, N. Yoshikawa, H. Terai, and S. Yorozu, "Design of a pipelined 8-bit-serial single-flux-quantum microprocessor with multiple ALUs," *Superconductor Science and Technology*, vol. 19, no. 5, p. S344, 2006.

[28] M. Tanaka, Y. Yamanashi, N. Irie, H. Park, S. Iwasaki, K. Takagi, K. Taketomi, A. Fujimaki, N. Yoshikawa, H. Terai *et al.*, "Design and implementation of a pipelined 8 bit-serial single-flux-quantum microprocessor with cache memories," *Superconductor Science and Technology*, vol. 20, no. 11, pp. S305–S309, 2007.

[29] E. Fang and T. V. Duzer, "A Josephson integrated circuit simulator (JSIM) for superconductive electronics application," *Extended Abstracts of 1989 International Superconductivity Electronics Conference*, pp. 407–410, 1989. [Online]. Available: https://ci.nii.ac.jp/naid/10008998489/

[30] K. Takagi, M. Tanaka, S. Iwasaki, R. Kasagi, I. Kataeva, S. Nagasawa, T. Satoh, H. Akaike, and A. Fujimaki, "SFQ propagation properties in passive transmission lines based on a 10-Nb-layer structure," *IEEE Transactions on Applied Superconductivity*, vol. 19, no. 3, pp. 617–620, 2009.

[31] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[32] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN Accelerator Simulator," *arXiv e-prints*, p. arXiv:1811.02883, Oct. 2018.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[34] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv e-prints*, p. arXiv:1506.01497, Jun. 2015.

[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv e-prints*, p. arXiv:1409.4842, Sep. 2014.

[36] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv e-prints*, p. arXiv:1704.04861, Apr. 2017.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[39] "Hot Chips 2017: A Closer Look At Google's TPU v2," https://www.tomshardware.com/news/tpu-v2-google-machine-learning,35370.html.

[40] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246

[41] A. M. Kadin, C. A. Mancini, M. J. Feldman, and D. K. Brock, "Can RSFQ logic circuits be scaled to deep submicron junctions?" *Applied Superconductivity, IEEE Transactions on*, vol. 11, no. 1, pp. 1050–1055, 2001.

[42] W. Chen, A. Rylyakov, V. Patel, J. Lukens, and K. Likharev, "Rapid single flux quantum t-flip flop operating up to 770 GHz," *Applied Superconductivity, IEEE Transactions on*, vol. 9, pp. 3212 – 3215, 07 1999.

[43] R. Kanada, Y. Nagai, H. Akaike, and A. Fujimaki, "Self-Shunted NbN Junctions With NbN_x/AlN Bilayered Barriers for 4 K Operation," *IEEE Transactions on Applied Superconductivity*, vol. 19, no. 3, pp. 249–252, Jun. 2009.

[44] M. A. Castellanos-Beltran, D. I. Olaya, A. J. Sirois, P. D. Dresselhaus, S. P. Benz, and P. F. Hopkins, "Stacked Josephson junctions as inductors for single flux quantum circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–5, 2019.

[45] S. K. Tolpygo, V. Bolkhovsky, T. J. Weir, A. Wynn, D. E. Oates, L. M. Johnson, and M. A. Gouker, "Advanced fabrication processes for superconducting very large-scale integrated circuits," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 3, pp. 1–10, 2016.

[46] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, "Energy-efficient superconducting computing—power budgets and requirements," *IEEE Transactions on Applied Superconductivity*, vol. 23, no. 3, pp. 1 701 610–1 701 610, 2013.

[47] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609–622.

[48] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.

[49] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[50] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 267–278.

[51] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: analog convnet image sensor architecture for continuous mobile vision," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 255–266, 2016.

[52] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-Situ analog arithmetic in crossbars," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, p. 14–26, Jun. 2016. [Online]. Available: https://doi.org/10.1145/3007787.3001139

[53] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural

network computation in ReRAM-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 27–39. [Online]. Available: https://doi.org/10.1109/ISCA.2016.13

[54] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 715–731. [Online]. Available: https://doi.org/10.1145/3297858.3304049

[55] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-Memory for energy-efficient neural network training: A heterogeneous approach," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51. IEEE Press, 2018, p. 655–668. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00059

[56] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, and Y. Xie, "SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory," in *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[57] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with High-Density 3D memory," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, p. 380–392, Jun. 2016. [Online]. Available: https://doi.org/10.1145/3007787.3001178

[58] M. Gruber, J. Jahns, and S. Sinzinger, "Planar-integrated optical vector-matrix multiplier," *Appl. Opt.*, vol. 39, no. 29, pp. 5367–5373, Oct 2000. [Online]. Available: http://ao.osa.org/abstract.cfm?URI=ao-39-29-5367

[59] K. Shiflett, D. Wright, A. Karanth, and A. Louri, "PIXEL: Photonic neural network accelerator," in *Proceedings of the 26th IEEE International Symposium on High-Performance Computer Architecture*, ser. HPCA '20, Feb. 2020.

[60] K. Kitayama, M. Notomi, M. Naruse, K. Inoue, S. Kawakami, and A. Uchida, "Novel frontier of photonics for data processing–photonic accelerator," *APL Photonics*, vol. 4, no. 9, p. 090901, 2019. [Online]. Available: https://doi.org/10.1063/1.5108912

[61] A. N. Tait, T. F. de Lima, E. Zhou, A. X. Wu, M. A. Nahmias, B. J. Shastri, and P. R. Prucnal, "Neuromorphic photonic networks using silicon photonic weight banks," *Scientific Reports*, vol. 7, no. 1, Aug 2017. [Online]. Available: http://dx.doi.org/10.1038/s41598-017-07754-z

[62] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature Photonics*, 2017.

[63] F. Chiarello, P. Carelli, M. G. Castellano, and G. Torrioli, "Artificial neural network based on SQUIDs: demonstration of network training and operation," *Superconductor Science and Technology*, vol. 26, no. 12, p. 125009, oct 2013.

[64] M. Altay Karamuftuoglu and A. Bozbey, "Single Flux Quantum Based Ultrahigh Speed Spiking Neuron," *arXiv e-prints*, p. arXiv:1812.10354, Dec. 2018.

[65] Y. Yamanashi, M. Tanaka, A. Akimoto, H. Park, Y. Kamiya, N. Irie, N. Yoshikawa, A. Fujimaki, H. Terai, and Y. Hashimoto, "Design and implementation of a pipelined bit-serial SFQ microprocessor, CORE 1β," *IEEE Transactions on Applied Superconductivity*, vol. 17, no. 2, pp. 474–477, 2007.

[66] Y. Ando, R. Sato, M. Tanaka, K. Takagi, N. Takagi, and A. Fujimaki, "Design and demonstration of an 8-bit bit-serial RSFQ microprocessor: CORE e4," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 5, pp. 1–5, 2016.

[67] K. Ishida, M. Tanaka, I. Nagaoka, T. Ono, S. Kawakami, T. Tanimoto, A. Fujimaki, and K. Inoue, "32 GHz 6.5 mW gate-level-pipelined 4-bit processor using superconductor single-flux-quantum logic," in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[68] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, p. 517–528, Jun. 2014. [Online]. Available: https://doi.org/10.1145/2678373.2665747

[69] G. Tzimpragos, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Volk, J. Shalf, and T. Sherwood, "A computational temporal logic for superconducting accelerators," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 435–448. [Online]. Available: https://doi.org/10.1145/3373376.3378517

[70] K. Loe and E. Goto, "Analysis of flux input and output josephson pair device," *IEEE Transactions on Magnetics*, vol. 21, no. 2, pp. 884–887, 1985.

[71] R. Cai, A. Ren, O. Chen, N. Liu, C. Ding, X. Qian, J. Han, W. Luo, N. Yoshikawa, and Y. Wang, "A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 567–578. [Online]. Available: https://doi.org/10.1145/3307650.3322270