공학석사 학위논문

Analytical Energy Modeling of a Vector Processing Unit

벡터 프로세싱 유닛의 해석적 전력 모델링

2023년 8월

서울대학교 대학원

전기 · 정보 공학부

김 규 리

Analytical Energy Modeling of a Vector Processing Unit

지도 교수 이 혁 재

이 논문을 공학석사 학위논문으로 제출함
2023 년 8 월

서울대학교 대학원
전기 · 정보 공학부
김 규 리

김규리의 공학석사 학위논문을 인준함
2023 년 8 월

위 원 장	김 태 환	(인)
부위원장	이 혁 재	(인)
위 원	심 재 웅	(인)

Abstract

Processor energy models have been extensively researched for a long time. Specifically, works on modeling deep learning processors have recently gained significant attention. Estimating energy consumption of neural processors plays a key role in various design decisions, across hardware architecture, software optimization, data and operation mapping space exploration, and neural architecture search. To achieve accurate energy prediction, it is necessary to consider the inter-instruction effects in addition to per-instruction energy.

In order to accurately model energy consumption with minimal overhead, we conducted an analysis of the target processor architecture's energy behavior at instruction level. Building upon this analysis, we developed simple analytical approaches to account for major power consumption factors, including inter-instruction effects. Our modeling method demonstrates an average kernel-level energy estimation accuracy of 95.52% with fast estimation time.

Keyword: Analytical Energy Consumption Model, Vector Processing Unit (VPU), Very Long Instruction Word (VLIW) Processor, Neural Processing Unit (NPU)

Student Number: 2021-26145

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Motivation

Processor energy consumption has been widely investigated in computer architecture field for decades. Recently, with the advent of deep learning applications and accelerators, there has been growing attention on power estimation of deep learning application execution on neural processing units (NPU).

Estimation on NPU energy consumption helps improving design on various levels. For instance, it can be used in architecture evaluation, data and schedule mapping space exploration, and hardware-aware neural architecture search. Especially in programmable processing units, energy estimation aids in software optimization for better algorithms or compile schemes. Therefore, estimating NPU energy consumption is important.

When operations are executed, energy consumption can be attributed to either the operation itself, or the switching of executed operations. Recent studies on NPU energy modeling have primarily focused on the former, neglecting the significant energy contribution from operation switching. This limitation leads to inaccurate estimation, particularly in scenarios with frequent operation switching. Previous researches conducted in 1990s and 2000s consider energy consumption from the switching of executed operations, but their modeling targets were processors at that time, which differ substantially from modern NPU architectures.

Therefore, we propose an accurate analytical energy modeling method that incorporates the energy consumed by operation switching. To achieve this, we conducted a detailed analysis of the processor architecture to simplify energy estimation. Specifically, our work targets a programmable vector processing unit within a modern NPU architecture.

1.2. Thesis Organization

This thesis is organized as follows. Chapter 2 gives a brief background on processor energy consumption and some previous energy estimation approaches. Chapter 3 explains the modeling target vector processor's hardware architecture and applications. Chapter 4 presents analysis on target processor's energy consumption behavior, and based on the analysis, Chapter 5 proposes an analytical energy modeling method to estimate processor kernel energy consumption. Chapter 6 presents experimental measurement and estimation results. Finally, Chapter 7 summarizes the proposed work and concludes this thesis.

Chapter 2

Background

2.1. Energy Consumption on Processing Units

Energy consumption in digital circuits is affected by several factors, such as leakage current, clock, control signal and data switching. To accurately model energy consumption, it is necessary to take these factors into account, which can introduce complexity into the modeling process. Besides, there are specific characteristics related to energy consumption in processing units that can simplify the energy modeling process.

Effect of Data Path Width In digital circuits, the width of data path plays a significant role in power consumption. A wider data path requires a larger number of transistors and interconnects, leading to increased capacitance and higher switching activity, thereby resulting in higher power consumption. Conversely, a narrower data path reduces the number of transistors and interconnects, resulting in lower power consumption. By considering this effect at the instruction level, we can simplify the energy model. Instructions with larger bit operands involve a wider data path in their execution path, thus consuming relatively more energy. Conversely, instructions with smaller bit operands consume less power and have a minimal impact on overall energy consumption. Therefore, they can be ignored for simplicity purposes.

Multi-Slot Structure In VLIW processors, the introduction of parallel instruction combinations adds complexity. However, decomposing power consumption can significantly simplify energy modeling.

When executing an instruction word with multiple slot instructions through pipeline stages, certain power consumption sources are shared among different slot instructions, while others are independent. The shared portion includes static power caused by leakage current and dynamic power dissipated in common hardware units such as the clock and instruction fetching and decoding control logic. The energy consumption from these shared components is not influenced by the instruction types, and therefore can be directly measured by executing a series of No Operation (NOP) instruction words.

The other portion of power consumption can be considered independent among different slots. Since the VLIW architecture is designed for parallel execution of an instruction word, different slot instructions typically do not share hardware units in the execution stages. Therefore, it is reasonable to assume that the mutual influence between different slot instructions can be disregarded. This property is commonly referred to as the spatial additive property [10].

2.2. Previous Works

2.2.1. Approaches to Processor Energy Estimation

Processor energy consumption has been extensively studied in the field of computer architecture, and there exists various classes of approaches on energy modeling [1]. A majority of works [2] [3] utilize performance counters to obtain computation activity factors and employ linear regression to determine power weights associated with each counter. These approaches have the advantage of not incurring additional overhead, but the main drawback is the inability to break down energy consumption on a per-process basis. Other models [8] [9] [11] obtain activity factors through simulation, providing detailed information at the hardware component and instruction level. However, these methods involve significant overhead. Some studies [10] [13] [14] focus on instruction-level energy estimation, by executing micro-benchmarks of instruction loops and profiling their execution.

In the case of NPUs, analytical energy modeling methods [5] [12] employ operation-based approaches by summing up the product of operation count and operation energy cost to estimate energy consumption. However, these approaches overlook the energy consumption resulting from operation switching, leading to inaccurate estimations, particularly for workloads with frequent operation switching.

2.2.2. Complexity in Energy Modeling

To achieve precise energy estimation, it is necessary to consider various factors from different abstraction levels at the hardware-software stack, including hardware micro-architecture and application algorithmic properties. Hence, the more we try to model accurately, the higher the modeling complexity becomes.

When targeting VLIW processors, the complexity becomes even more significant, since very long instructions can be created by combining multiple instructions. [10] introduces spatial additive property to reduce such complexity, which has been verified under extensive studies afterwards.

Unfortunately, accurate energy models require more than per-instruction estimation. Inter-instruction effects [16] can have a significant impact on power consumption, and incorporating these effects adds further complexity. In [7], the use of NOP instructions is proposed to model transitions between any two instructions, achieving an accuracy within 8% error while significantly reducing complexity. [6] suggests an instruction clustering method for modeling efficiency with average error of 1.9%. However, these works targeted general-purpose and digital signal processors with small data paths, thereby being applicable only to processors with similar architectures and not to vector processors designed for DNN applications.

Additionally, breaking down the target software into instructions is beneficial for accuracy, as in [14] [15] [6]. However, analyzing at instruction-level requires compile-able program codes, a compiler, and runtime instruction profiling, which can make the modeling process cumbersome. Alternatively, analyzing software using analytical methods can significantly improve efficiency.

Chapter 3

Target Processor Overview

3.1. Hardware Architecture

The target processor for energy modeling is a programmable vector processor within a systolic array-based neural processing unit (NPU) system. Figure 3.1 provides a schematic of the top-level NPU system architecture. The system comprises a systolic array, an on-chip scratchpad memory, vector processor cores, and off-chip DRAM. The on-chip scratchpad memory, implemented as a multi-bank SRAM, serves as a shared storage for tensor data that can be accessed by both the systolic array and the vector processor. The systolic array accelerates convolution and generalized matrix multiplication (GEMM), which are fundamental operations in deep neural networks (DNNs). The vector processor, composed of 16 core-tiles for parallel processing, handles a broader range of operations that the systolic array either cannot compute efficiently or cannot perform at all. Examples of such operations include depth-wise convolution, element-wise addition, pooling, up-sampling, and non-linear activation functions.

A vector processor core is a programmable very long instruction word (VLIW) processor consisting of an interface to on-chip scratchpad memory, local scratchpad memory components for program code, non-tensor scalar data, and lookup tables (LUTs), register files, and three issue slots. The register files are of eight distinct kinds with varying bit widths, and are accessible from multiple slots by separate read/write ports. More details about register files are provided in Chapter 4. The issue slots are scalar slot, memory slot, and vector slot with dedicated arithmetic units and orthogonal instruction set architectures (ISA) specific to each slot. During each cycle, a single VLIW instruction is fetched from the program scratchpad memory, decoded into instructions for each slot, and issued in an in-order manner.

The memory slot manages the memory access operations for tensor and scalar data between the memory components and the core registers. Vector slot instructions are designed to efficiently perform identical operations on multiple data elements, and they are dispatched to a single instruction multiple data (SIMD) unit for acceleration. Scalar slot instructions mainly use a scalar arithmetic logic unit (ALU) to handle auxiliary operations such as memory address computation, stack pointer and control flow operations.

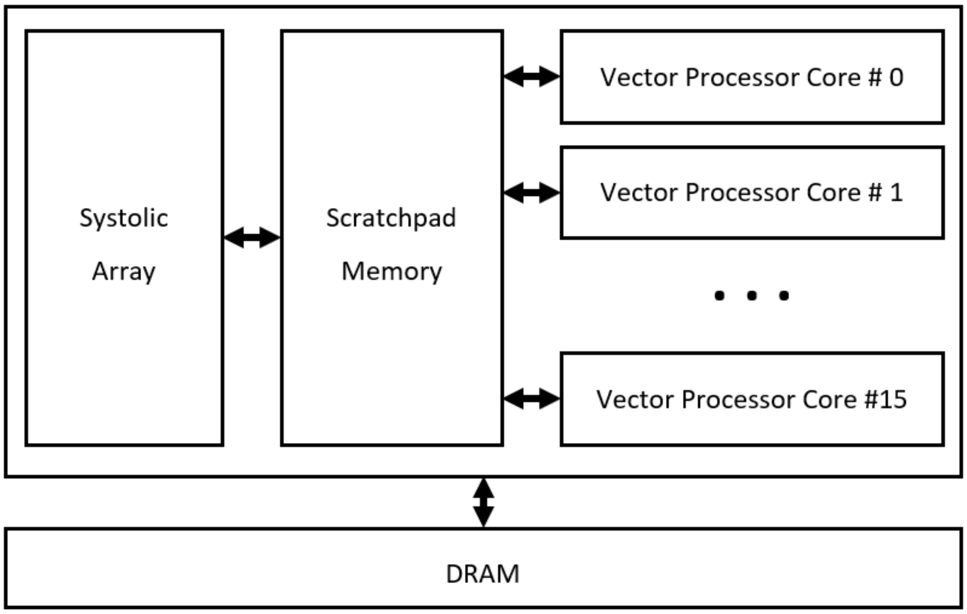

Systolic Array
Scratchpad Memory
Vector Processor Core # 0
Vector Processor Core # 1
• • •
Vector Processor Core #15
DRAM

Figure 3.1: Block diagram of a systolic array-based NPU system architecture

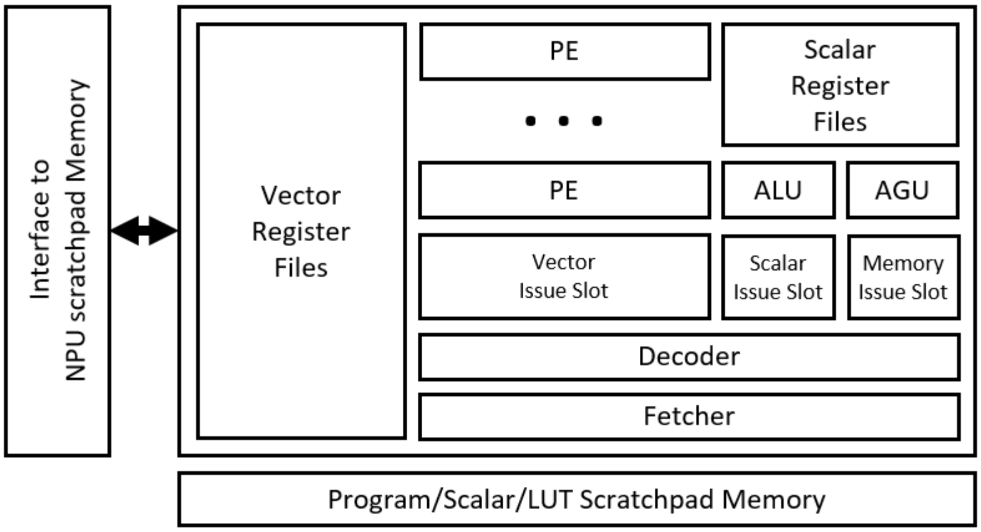


Figure 3.2: Block diagram of a vector processor core

3.2. Target Applications

Unlike general-purpose VLIW processors, the vector processor is specialized for DNN operations. More specifically, under the NPU system described in Section 3.1., it handles operations that are not suitable for the systolic array, such as depth-wise convolution, element-wise addition, max pooling, average pooling, and non-linear activations.

The execution of these target applications can be easily analyzed in a loop-based manner. The computation of DNN layers can be structured using simple nested loops, where the loop bounds are determined by feature map size, filter shape, and other configurations like padding and stride. Consequently, the control flow of these applications is relatively straightforward compared to other general-purpose processor kernels. This simplicity simplifies the estimation of runtime execution without the need for actual compilation or cycle-level simulation. Taking advantage of this characteristic, the proposed energy modeling method decomposes and analyzes DNN layer computations in a fast and analytical loop-based approach to model energy consumption.

To perform DNN layer computations on the vector processor, the layer computation algorithm is implemented as a processor kernel code. The vector processor ISA is exposed as a C primitive, allowing the implementation of target applications as processor kernels written in C code. The proposed energy modeling method has been validated on 8 different kernels, as presented in Table 3.1. Various other types of DNN operations can also be executed on the vector processor with appropriately written kernel codes. Algorithm 1 provides an example pseudocode for the processor kernel K5_DWCV.

The processor kernel program written in C code is compiled into VLIW instructions, and program binary code is generated. The compiler is provided with micro-architectural information, enabling it to explicitly leverage instruction-level parallelism (ILP) in the generated machine code. During the execution of a neural network on the NPU, when a layer needs to be processed by the vector processor, a runtime command scheduler on the NPU loads the generated program binary onto the instruction scratchpad of the vector processor and initiates kernel execution. Once the execution is completed, the vector processor core sends result signals to the command scheduler.

Kernel ID	Layer Type	Optimized Config
K0_ACTV	nonlinear activation	generic
K1_EADD	element-wise addition	generic
K2_MAXP_k3s2	max pooling	kernel = (3, 3), stride = (2, 2)
K3_GAP	global average pooling	generic
K4_DWCV_k3	depth-wise convolution	kernel = (3, 3)
K5_DWCV	depth-wise convolution	generic
K6_UPS_k2	nearest neighbor up-sample	kernel = (2, 2)
K7_UPS	nearest neighbor up-sample	generic

Table 3.1: Types of DNN layer processor kernels

Algorithm 1 K4_DWCV_k3 kernel pseudocode

```
 1: Initialize: in/out/kernel information, address
 2: for Oc: 0, 1, …, channel do
 3:     Load: SR_filter0, 1, 2, …, 8 ← filter data 0, 1, 2, …, 8
 4:     Broadcast: VR_filter0, 1, 2, …, 8 ← SR_filter0, 1, 2, …, 8
 5:     for Oh: 0, 1, …, out_height do
 6:         for Ow: 0, 1, …, out_width do
 7:             Initialize: ACCR
 8:             Load: VR_fmap0, 1, 2, …, 8 ← fmap data 0, 1, 2, …, 8
 9:             Compute mac: ACCR ← ACCR + VR_fmap0 * VR_filter0
10:             Compute mac: ACCR ← ACCR + VR_fmap1 * VR_filter1
11:             Compute mac: ACCR ← ACCR + VR_fmap2 * VR_filter2
12:             Compute mac: ACCR ← ACCR + VR_fmap3 * VR_filter3
13:             Compute mac: ACCR ← ACCR + VR_fmap4 * VR_filter4
14:             Compute mac: ACCR ← ACCR + VR_fmap5 * VR_filter5
15:             Compute mac: ACCR ← ACCR + VR_fmap6 * VR_filter6
16:             Compute mac: ACCR ← ACCR + VR_fmap7 * VR_filter7
17:             Compute mac: ACCR ← ACCR + VR_fmap8 * VR_filter8
18:             Compute shift & Move register: VR_result ← ACCR >> shifter
19:             Store: fmap data ← VR_result
20:         end for
21:     end for
22: end for
```

Chapter 4

Analysis on Processor Energy Consumption

4.1. Effect of Architectural Characteristics

4.1.1. Multi-Slot Structure

The spatial additive property explained in Section 2.1 can be applied to the vector, memory, and scalar slot instructions in our target vector processor. By considering the slot-independent energy measured from a NOP sequence and the additive property of each slot instruction, the total energy consumption (E_{total}) can be calculated using the NOP energy (E_{NOP}) and the energy of each slot instruction (E_{vector}, E_{memory}, E_{scalar}), as shown in Equation (4.1).

$$E_{total} = E_{NOP} + E_{vector} + E_{memory} + E_{scalar} \tag{4.1}$$

4.1.2. Data Path Width

Table 4.1 illustrates the vector processor core register files and their corresponding data widths. The table shows that different slots utilize different sets of register files. Scalar slot instructions utilize SCR, AR and SR, all of which are up to 32-bit wide. Since scalar slot instructions involve small bit operands, the wires and pipeline registers downstream of the scalar slot instruction data path also have narrow bit widths. On the other hand, vector slot instructions have access to VR, VCR, LUTR, ACCR, and BIASR, which have larger data widths, resulting in wider downstream execution data paths. The memory slot ISA consists of load and store instructions with optional address computation, with operands involving scalar data, stack data, and vector data. Scalar data and stack data memory instructions utilize SCR, AR, and SR, thus involving a small data path, while vector data memory instructions access VR, resulting in a wider downstream data path.

Based on these observations, vector processor instructions can be categorized into two groups: narrow data path (ND) instructions with a small data path involved, accessing data up to 32-bit wide, and wide data path (WD) instructions with larger data widths. All scalar slot instructions and scalar/stack data memory slot instructions belong to the ND instructions, while vector slot instructions and vector data memory instructions belong to the WD instructions. Considering the relationship between data path width and energy consumption discussed in Section 2.1, the energy consumption of WD instructions is dominant, whereas that of ND instructions is relatively insignificant. This was confirmed through experimental measurements where ND instructions consumed less than 0.3 W, while WD instructions consumed 1 W ~ 6W. Based on these findings, it can be concluded that ignoring ND instructions has minimal impact on energy modeling accuracy while significantly reducing model complexity. Therefore, the proposed energy modeling method focuses solely on the energy consumption of WD instructions for efficiency.

Register Name	Description	Bit Width (>: greater than)	Access from Slot		
			S	V	M
SCR	scalar condition	1	O	O	O
AR	memory address	16	O	O	O
SR	scalar data	32	O	O	O
VR	vector data	512		O	O
VCR	vector condition	512		O	
LUTR	LUT loaded data	512		O	
ACCR	accumulation data	> 512		O	
BIASR	bias data	> 512		O	

Table 4.1: Vector processor core register files

4.2. Instruction-Level Energy

In order to analytically model energy consumption of a processor kernel, we first measured and analyzed instruction-level energy consumption. As mentioned in Section 4.1.2 that we only consider WD instructions, we narrowed down measurement target to vector slot instructions and vector load and store instructions. To measure the base and inter-instruction energy of each instruction, we wrote short processor kernels composed of measurement target instructions. Furthermore, to differentiate the power consumption between the control path and the data path, measurement kernels were executed with both zero and random data. Details on experimental environments are described in Chapter 6.

Measurement of Instruction Base Energy To measure the instruction base energy, we wrote power measurement kernels as in Figure 4.1. These kernels consisted of a long repeated loop filled with fully-pipelined target instructions, vector MAC (Multiply-Accumulate) instructions in this case. During kernel execution, we collected samples of measured instantaneous power at regular intervals and averaged them to obtain base energy.

assembly code	description
kernel prologue	
register initialization	
loop 4 2500	// repeat succeeding 4-instr sequence 2500 times
vmac accr v0 v1	// accr += v0 * v1
vmac accr v2 v3	// accr += v2 * v3
vmac accr v0 v1	// accr += v0 * v1
vmac accr v2 v3	// accr += v2 * v3
kernel epilogue	

Figure 4.1: Example of vmac instruction base energy measurement kernel

To isolate the energy consumption of the target instruction sequence within a single slot, it is necessary to eliminate redundant energy factors. According to the energy decomposition described in Equation (4.1), NOP energy is considered redundant. Additionally, the overhead associated with loop control, which involves the utilization of the loop counter register, is also redundant. As a result, the measured power of target instruction sequence can be calculated using Equation (4.2).

In the equation, P_{instr_loop} and P_{NOP_loop} represent the average sampled power obtained from the instruction base energy measurement kernel in Figure 4.1, and the NOP loop measurement kernel in Figure 4.2, respectively.

$$P_{measure} = P_{instr\_loop} - P_{NOP\_loop} \tag{4.2}$$

assembly code	**description**
kernel prologue	
register initialization	
loop 4 2500	// repeat succeeding 4-instr sequence 2500 times
vnop	// vector slot nop instruction
vnop	// vector slot nop instruction
vnop	// vector slot nop instruction
vnop	// vector slot nop instruction
kernel epilogue	

Figure 4.2: NOP loop energy measurement kernel

The instruction base energy can be directly obtained from $P_{measure}$. Figure 4.3 illustrates the fully-pipelined execution of a 5-stage instruction. At the instant of power sampling, all pipeline stages are filled, and thus the measured power is the sum of single-cycle energy across the spatial axis, as shown in Equation (4.3) line 1 and 2 and the green-highlighted stages in Figure 4.3. Besides, line 2 of Equation (4.3) also corresponds to the blue-highlighted stages as well as green, so the summation can be considered equivalent to sum of each stage energy of a single instruction across the temporal axis. Therefore, the instruction base energy is equal to $P_{measure}$. In Equation (4.3), B_i denotes the base energy of instruction i.

$$P_{measure} * 1 \text{ cycle} = E_{measure}$$
$$= E_{stage0} + E_{stage1} + E_{stage2} + E_{stage3} + E_{stage4}$$
$$= B_i \tag{4.3}$$

In *register initialization* in Figure 4.1, if all operand registers are initialized with zero, no data switching occurs during kernel execution. Therefore, the measured energy can be considered as the energy consumption in the control path. On the other hand, when the registers are initialized with random data, both the control path and data path switching occur. By subtracting the measurement with zero data initialization from the measurement with random data initialization, we can obtain the energy consumption of the data path. To ensure data switching on all data paths, we took care to initialize each register with different data values and use different registers in every cycle. If this is not guaranteed, power would get much under-measured.

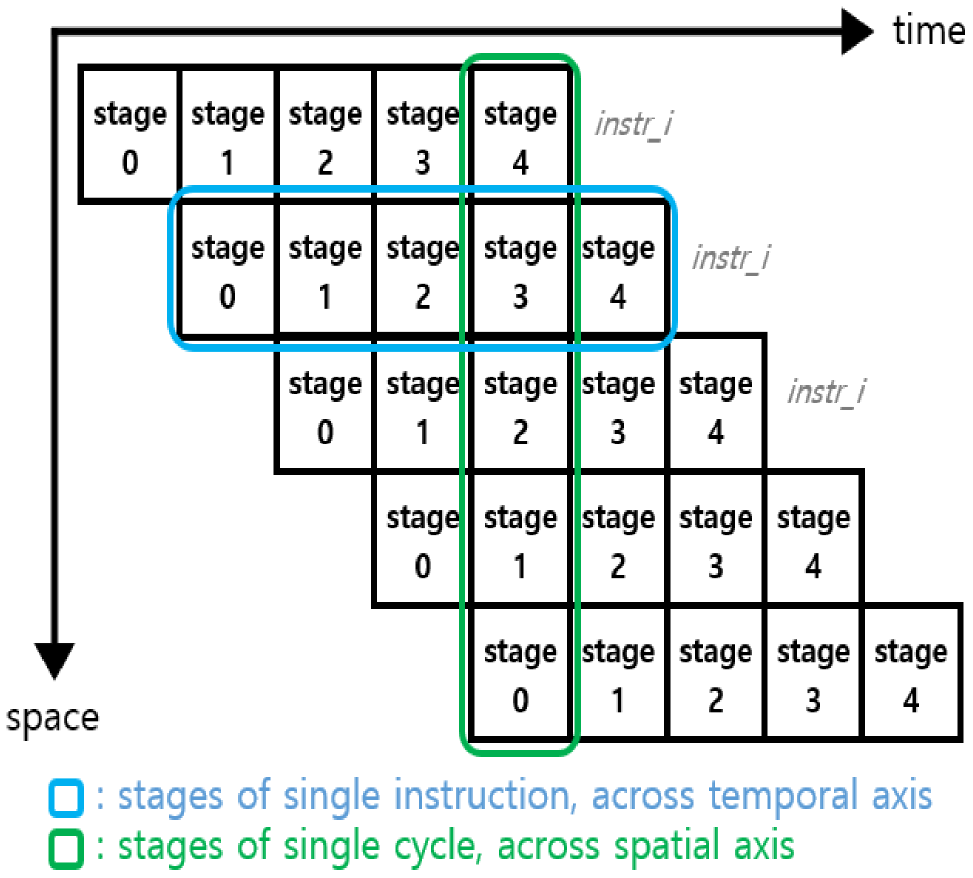

time
space
stage 0
stage 1
stage 2
stage 3
stage 4
instr_i
instr_i
instr_i
: stages of single instruction, across temporal axis
: stages of single cycle, across spatial axis

Figure 4.3: Pipeline diagram of a 5-stage instruction base energy measurement kernel

Measurement of Inter-Instruction Energy To measure inter-instruction energy, we used power measurement kernels as shown in Figure 4.4, similar to the base energy measurement kernel but with two types of instructions switching every cycle. As aforementioned, we only measured inter-instruction energy on pairs with NOP for model efficiency.

assembly code	description
register initialization	
loop 4 2500	// repeat succeeding 4-instr sequence 2500 times
vmac accr v0 v1	// accr += v0 * v1
vnop	// vector slot nop instruction
vmac accr v2 v3	// accr += v2 * v3
vnop	// vector slot nop instruction

Figure 4.4: Example of {vmac, vnop} instruction pair inter-instruction energy measurement kernel

Since two different instructions in the measurement kernel are time-sharing with equal contributions, the energy consumption can be considered as the average of each instruction's base energy. However, as mentioned in Section 2.2.2, the energy consumption of different instructions sequence is always larger than that of a single instruction sequence, and the corresponding term is regarded as the overhead of executing different instructions sequentially, namely inter-instruction energy. As a result, with $I_{i,j}$ denoting inter-instruction energy of instruction {i, j} pair, the relationship between different energy factors in the inter-instruction energy measurement kernel can be expressed as in Equation (4.4).

$$P_{measure} * 2 \text{ cycle} = E_{measure} * 2 = B_i + B_j + 2 * I_{i,j} \quad (4.4)$$

From this, the inter-instruction energy of instruction pair $\{i, j\}$ can be calculated as in Equation (4.5). Here, since NOP energy is already removed from $E_{measure}$, B_{NOP} can be regarded as zero in each slot energy consumption.

$$I_{i,j} = E_{measure} - \frac{1}{2} * (B_i + B_j) \quad (4.5)$$

As in base energy measurement, inter-instruction energy measurement was also done with both zero and random operand data, in order to distinguish between control path and data path energy.

4.2.1. Memory Slot Instructions

WD instructions of memory slot ISA consist of vector load and store instructions, with different addressing modes and optional address register update. On these instructions, instruction base energy, inter-instruction energy with NOP-pair, each on both control and data path are obtained from experimental measurement. The measured values are presented in Figure 4.5 and Figure 4.6. In the figures, the energy values on the y-axis are scaled with a mutual scaling factor for security reasons.

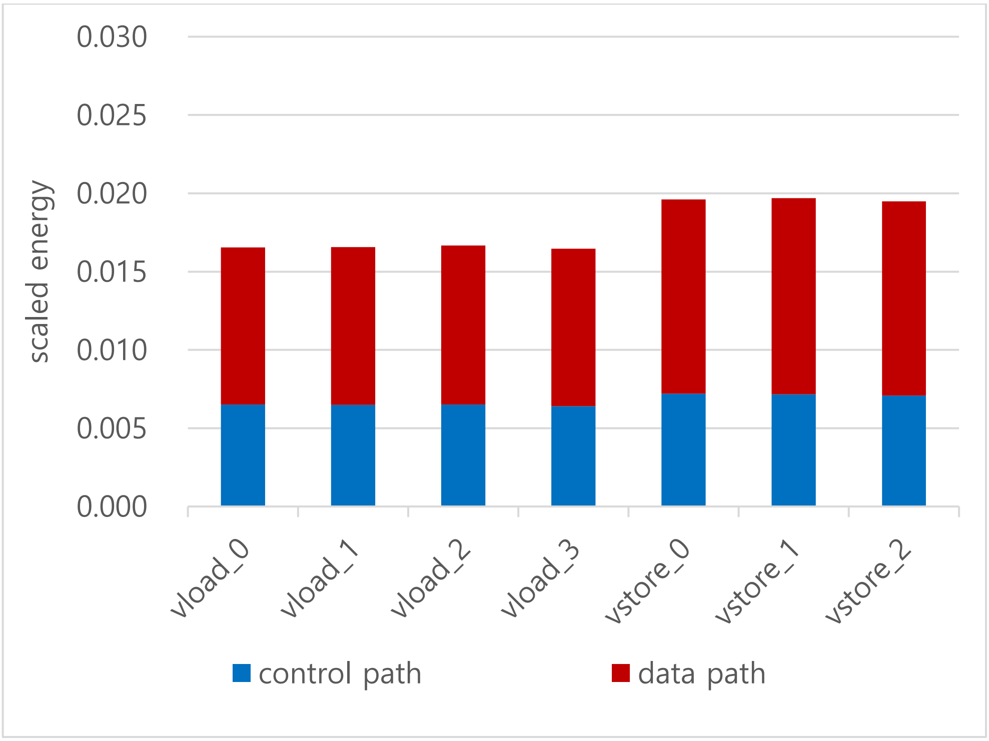


Figure 4.5: Measured base energy of memory slot instructions

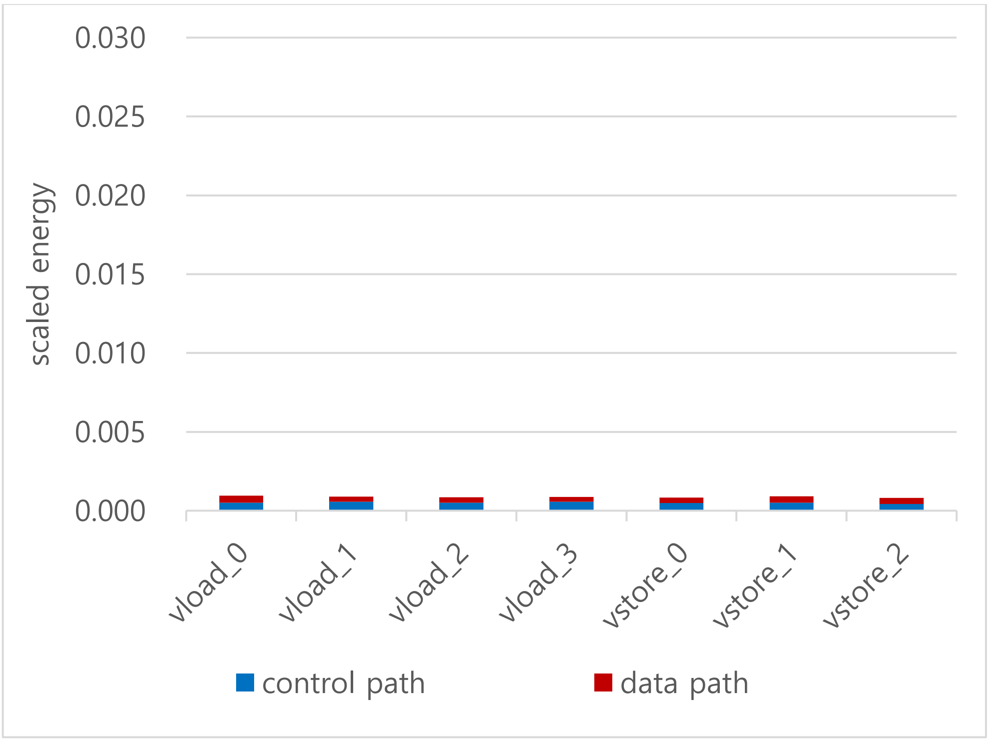


Figure 4.6: Measured inter-instruction energy of memory slot instructions

In Figure 4.5 and 4.6, different load and store instructions are depicted across the x-axis, varying in addressing modes and address register update options. It can be observed that both the data path and control path have an impact on the base and inter-instruction energy of memory slot instructions. However, the amount and variance of inter-instruction energy from Figure 4.6 are much smaller compared to those of the base energy. Basically, inter-instruction energy consumption occurs because different instructions utilize different parts of the circuit, leading to circuit state change [16]. Nonetheless, load and store instructions primarily involve only address calculation and memory access, resulting in simple control signals and pipeline data path structure. As a result, the overhead associated with switching instructions is relatively small and consistent. Consequently, we can assume a constant value for inter-instruction energy when executing different memory operations in sequence, regardless of specific instruction types.

Another observation is that addressing modes and address register update option do not significantly impact the energy behavior. The address-related logic in these instructions utilizes much smaller bits compared to the memory access operands, which is vector data. Since these operations have negligible energy consumption, it suffices to differentiate between load and store instructions for energy modeling purposes. Thus, we model memory slot instructions as either load or store instructions without considering the addressing modes and address register update option, thereby effectively reducing the complexity associated with handling these low-level functions, which would otherwise be challenging without the assistance of an actual compiler.

4.2.2. Vector Slot Instructions

Similar to memory slot instructions, base energy and inter-instruction energy with NOP-pair of vector slot instructions were measured on both the control and data paths, and the scaled results are presented in Figure 4.7 and Figure 4.8.

The measurements revealed substantial variations in energy consumption among instructions. Even within the same class of instructions, such as register move instructions (vinst_0 ~ vinst_4), notable differences were observed based on the type and number of source and destination registers. Moreover, the inclusion of a shift operation after vector mac operation (vinst_5, vinst_6) had a significant impact on energy consumption. Consequently, unlike memory slot instructions, precise instruction-level energy estimation is required for vector slot instructions, considering all subfunction information.

In particular, the large energy consumption and variance of vector slot instructions is mainly attributed to the data path rather than the control path. These instructions utilize vector operand data and SIMD units to accelerate vector arithmetic operations, which involve a large data path. Additionally, they handle the major computations of DNN layer operations, engaging complex control signals and a deep, intricate pipeline with many functional units. As a result, most of the switching activity occurs in the data path, highlighting the need to analyze and model energy consumption focusing on the data path.

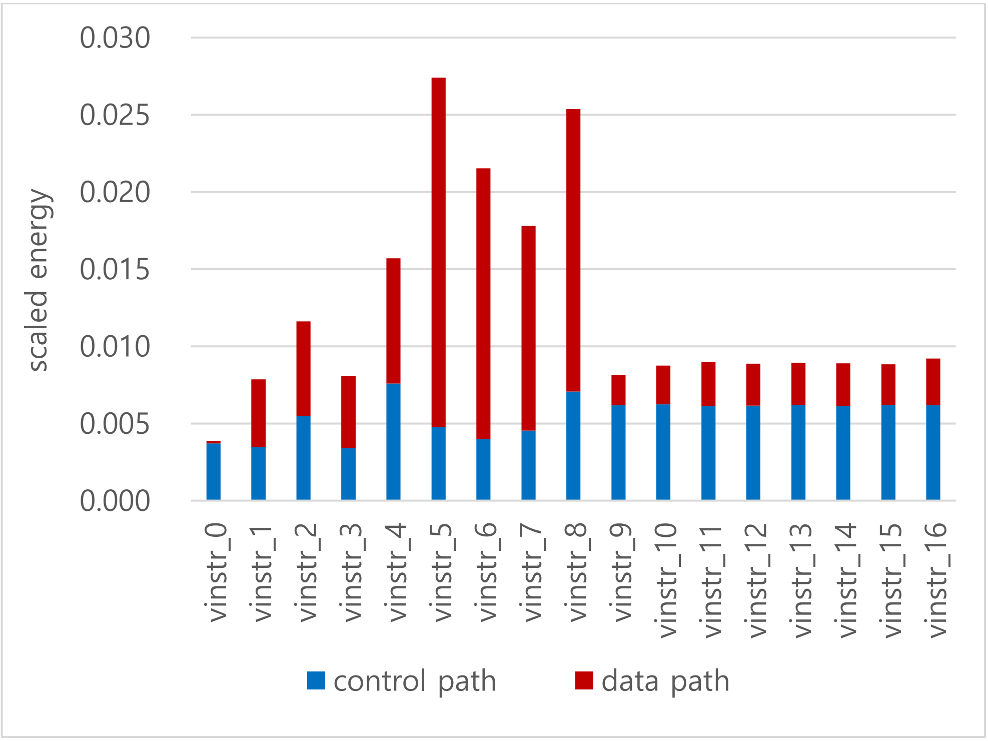


Figure 4.7: Measured base energy of vector slot instructions

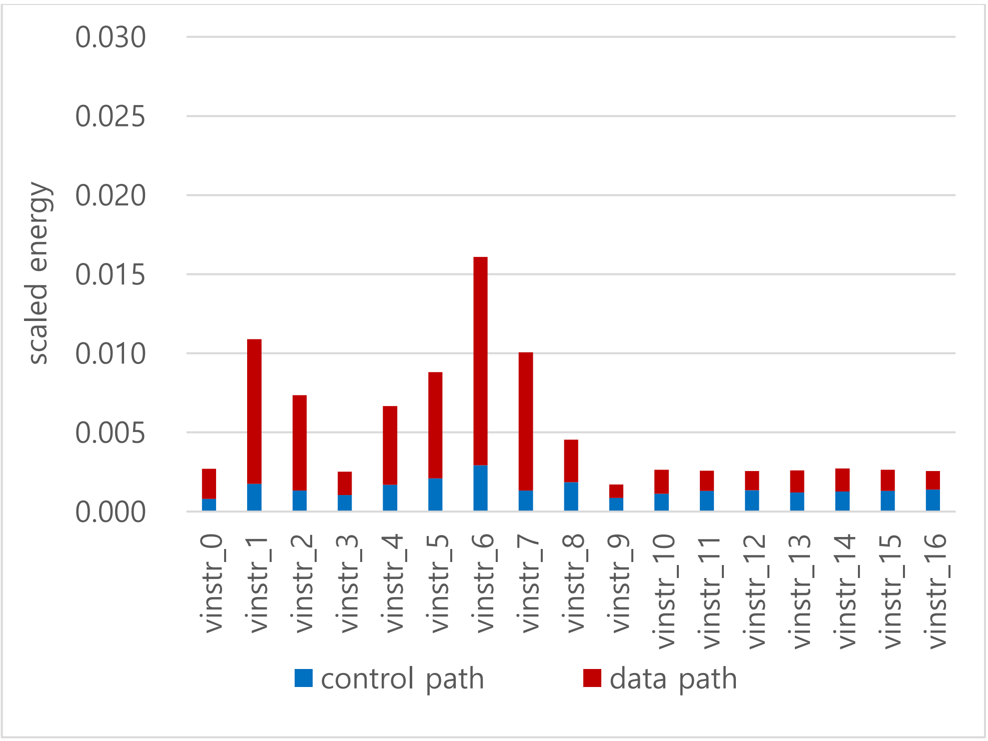


Figure 4.8: Measured inter-instruction energy of vector slot instructions

Chapter 5

Analytical Energy Modeling Methodology

5.1. Modeling Method Overview

Based on the observations made on the energy consumption of the vector processor in previous chapters, an analytical energy modeling methodology has been developed. The overall flow of the proposed method is depicted in Figure 5.1. It takes as input the layer configurations (e.g., layer type, data and kernel shape), as well as the corresponding layer processor kernel algorithm description. The output is the total energy consumption of each slot throughout the execution of single layer kernel.

To achieve analytical modeling without the need for kernel code compilation and simulation, the methodology consists of 4 analytical steps. Step 1, 2 will be covered in Section 5.2, and 2, 3 in Section 5.3 respectively. The entire framework is implemented in Python 3.8.

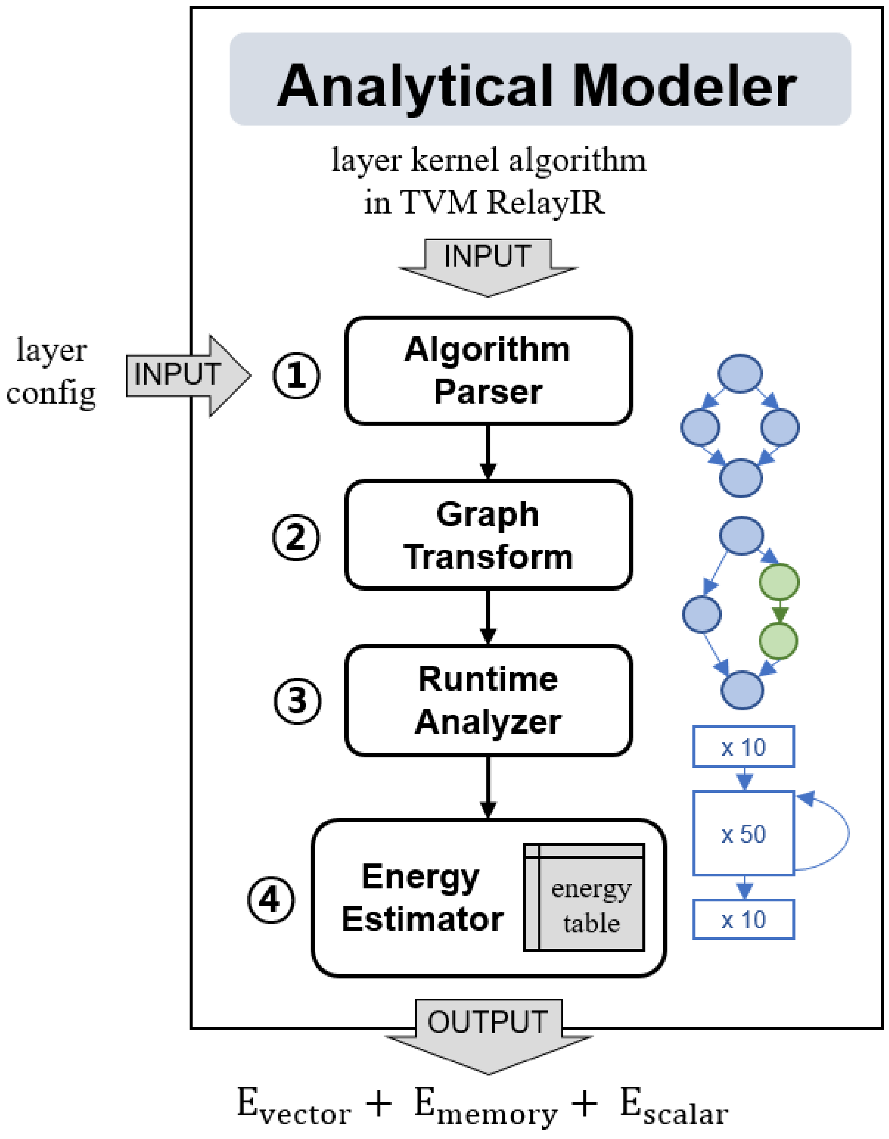


Figure 5.1: Overall flow of the proposed analytical energy modeling method

5.2. Input to Graph Conversion

5.2.1. Algorithm Parsing

In step 1, an in-house algorithm parser is utilized to generate a directed acyclic graph (DAG). The nodes of the graph represent general operations such as addition, multiplication, mac, max and so on. The algorithm parser leverages intrinsic functions provided by TVM, an open-source machine learning compiler framework [4]. Layer kernel algorithms are written in python code to describe computations in TVM's Relay Intermediate Representation (IR) level, and the algorithm parser lowers IR to TVM's nested Tensor-level IR (TIR) by employing a set of TVM intrinsic functions. Subsequently, the nested TIR is converted into an operation graph, as depicted in Figure 5.2. Control operations are colored in yellow, memory slot operations in green, and vector slot operations in blue. Moreover, iteration count information of loop control operations is also included in the graph.

5.2.2. Graph Transform

Step 2 transforms operation DAG from step 1 output into instruction DAG as illustrated in Figure 5.3, with all operation nodes except for control operations converted to instruction nodes. The conversion process involves removing, replacing, and adding certain nodes based on each slot ISA and hardware-specific features. Once the transformation is completed in step 2, the instruction nodes in the graph can be one-to-one mapped to energy table entries of energy estimator in step 4.

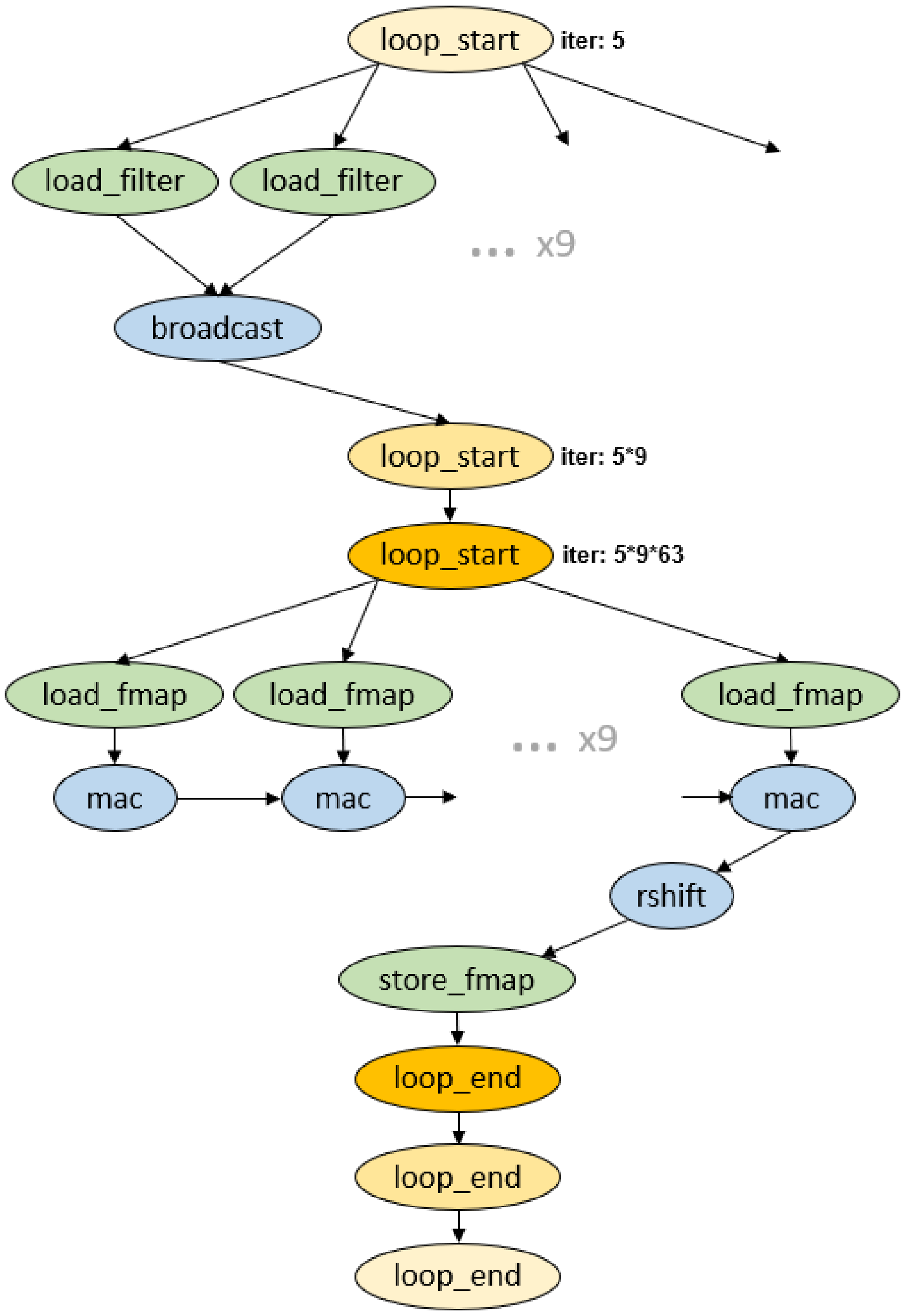


Figure 5.2: K4_DWCV_k3 kernel algorithm in operation DAG representation

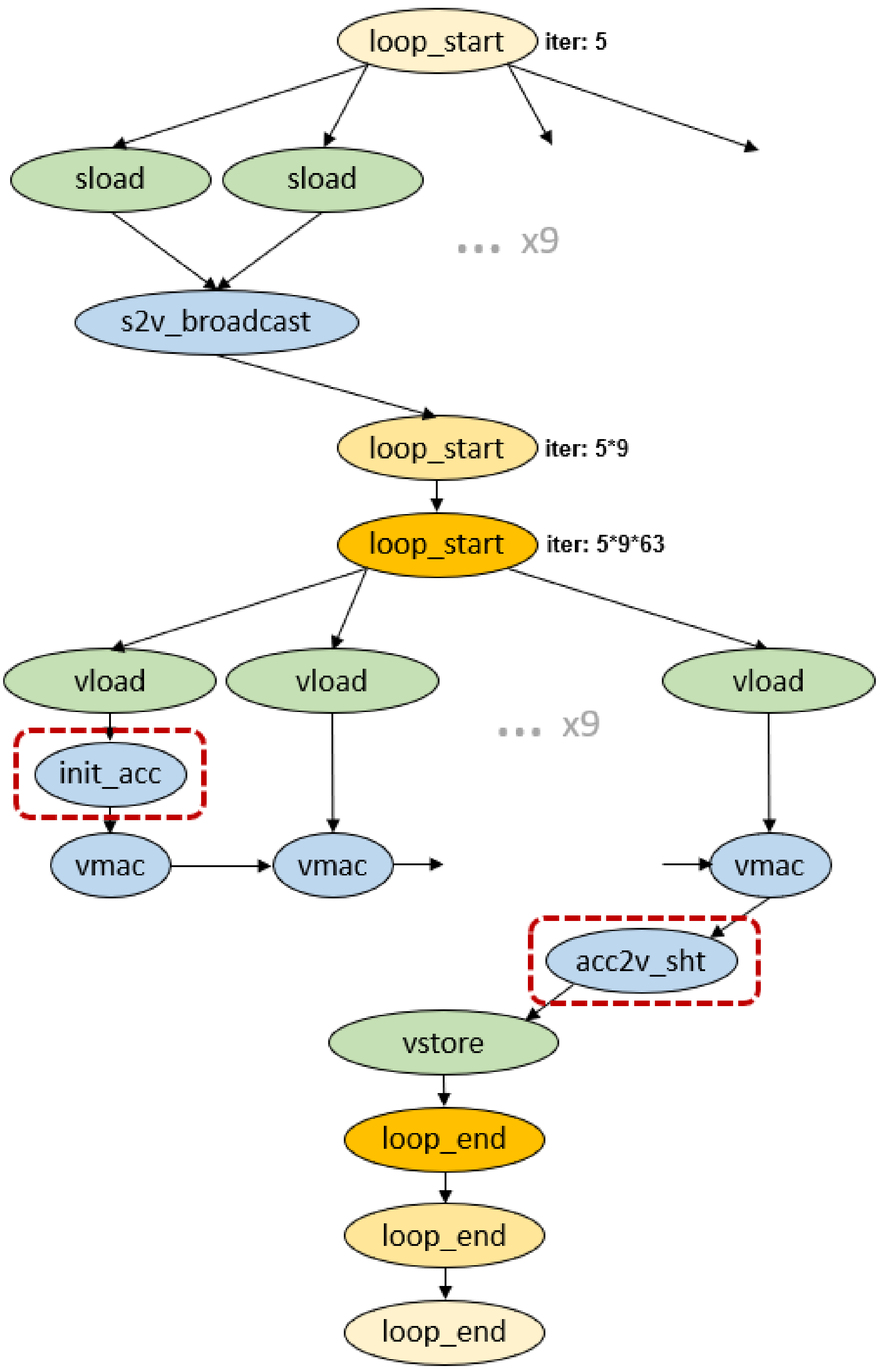


Figure 5.3: K4_DWCV_k3 kernel algorithm in instruction DAG representation

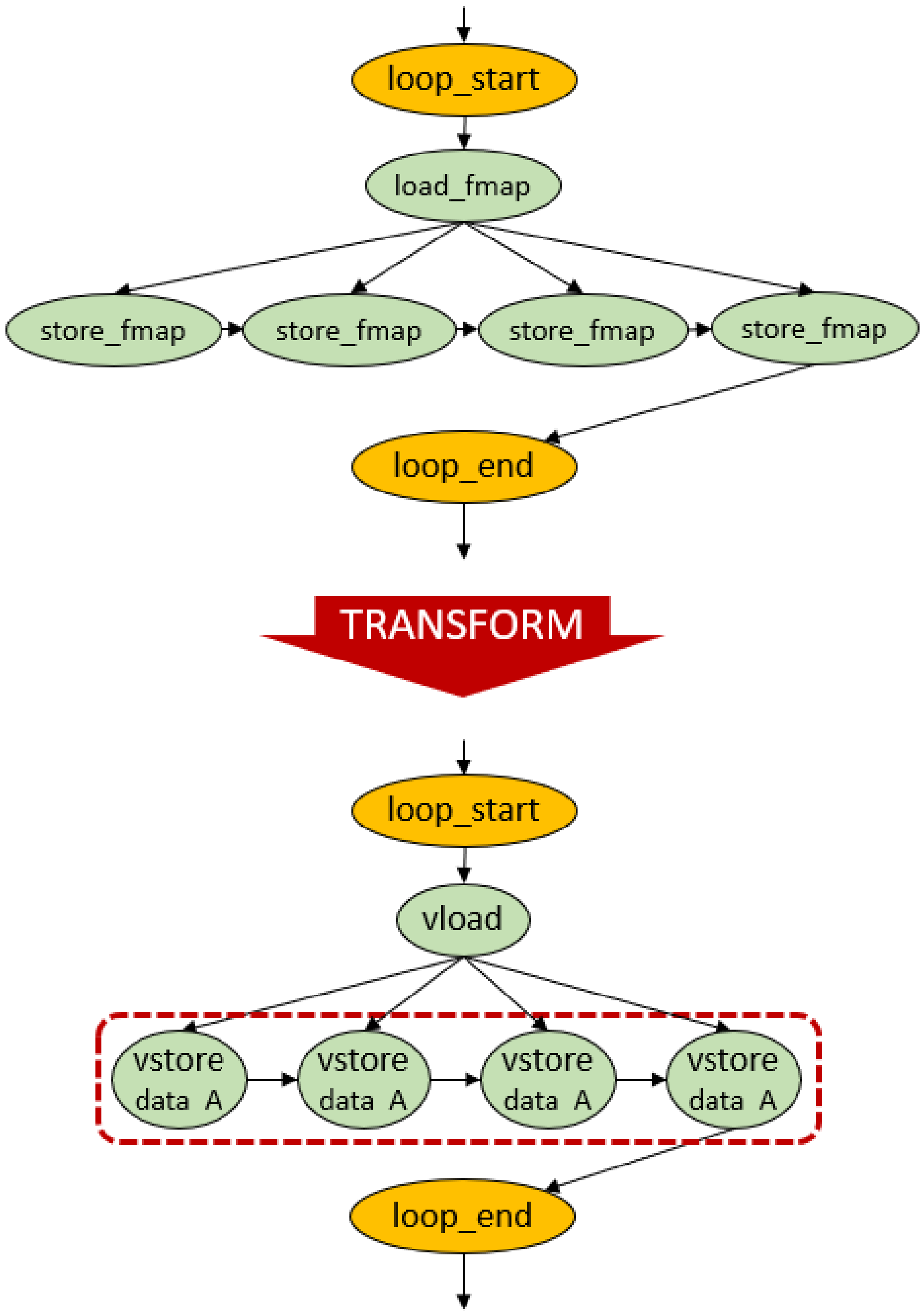


Figure 5.4: Same data store marking in K6_UPS_k2 kernel algorithm innermost loop

Memory Slot Operations In the operation DAG generated in step 1 (Figure 5.2), the green-colored nodes represent filter and feature map memory operations. These operations can be directly transformed into scalar and vector load and store instruction nodes, respectively, as they already align with the modeling level of load and store instructions without functional codes. However, it is important to note that scalar memory instructions consume relatively minor energy and will be ignored in the later steps of the methodology, as mentioned in Section 4.1.2.

In addition, there is a significant attribute of store operations that needs to be analyzed in this step. When consecutive store operations handle the same data, switching activity is reduced, leading to a significant decrease in energy consumption compared to cases where the data changes every cycle. This situation commonly occurs in algorithms such as nearest neighbor up-sampling, where a pixel is loaded once and then stored multiple times. To account for this scenario, store operation nodes that have a mutual predecessor are marked as using the same data, as shown in Figure 5.4. This information will be considered in the subsequent steps when these operations are executed in consecutive cycles.

Vector Slot Operations Since the vector processor ISA is designed to support DNN operations, many vector slot instructions align with single operation nodes. However, some operations function in a hardware-specific manner, requiring appropriate transformation to the instruction level.

First, the vector processor operates with INT8 data type. Consequently, tensor element values after computation undergo right-shifting for quantization before storage in memory. Thus, if the input graph does not explicitly include a shift operation, some kind of shift operation should be inserted during the transformation process. This does not apply to max pooling and nearest neighbor up-sampling layers, where tensor element values do not go through any computation operations, but only gets replaced.

Moreover, the vector processor employs an accumulation register (ACCR) with a larger bit capacity for accumulation operations, and only a single ACCR is available in the core. As a result, when a group of operation nodes involve value accumulation, an ACCR initialization instruction is inserted prior to the head node of the group. For instance, in Figure 5.3, an init_acc node is added as a predecessor to a sequence of consecutive vmac instructions.

Another consequence of using the ACCR is the insertion of register move instructions between different register types, based on the predecessor operation's destination register and successor operation's source register. Additionally, for efficiency purposes, quantization shift operations are combined with these register moves or other instructions whenever feasible. In Figure 5.2, the predecessor operation of the rshift operation has a destination register of ACCR, while the successor operation of the rshift operation has a source register of VR. This necessitates the insertion of an ACCR to VR register move, referred to as the acc2v instruction. Furthermore, the rshift operation is combined with the acc2v instruction, resulting in the final transformed node, acc2v_sht, as depicted in Figure 5.2.

5.3. Estimating Energy from Graph

5.3.1. Estimating Runtime Sequence

After step 2, we have all the information on which instructions are executed and how many times in the instruction DAG. However, to account for inter-instruction effects in energy estimation, information on the execution sequence of instructions is also required. To achieve this, we utilize an in-house runtime analyzer.

The runtime analyzer takes the instruction DAG from step 2 as input and considers the dependency information represented by graph edges. It utilizes pre-defined hardware execution latency information for each instruction to estimate the runtime execution sequence of nodes. As a result, step 3 produces the estimated control flow graph of the kernel, including the iteration information of each basic block and the internal execution sequence of instructions.

Figure 5.5 illustrates the result of step 3 for the K4_DWCV_k3 kernel, based on the input instruction DAG shown in Figure 5.3. The first, second and third columns in Figure 5.5 represents the scalar slot, memory slot, and vector slot respectively, while a hyphen denotes a NOP. As observed, the scalar slot sequence is empty, and scalar load and store instructions have been removed since they will be disregarded in the subsequent energy estimation step.

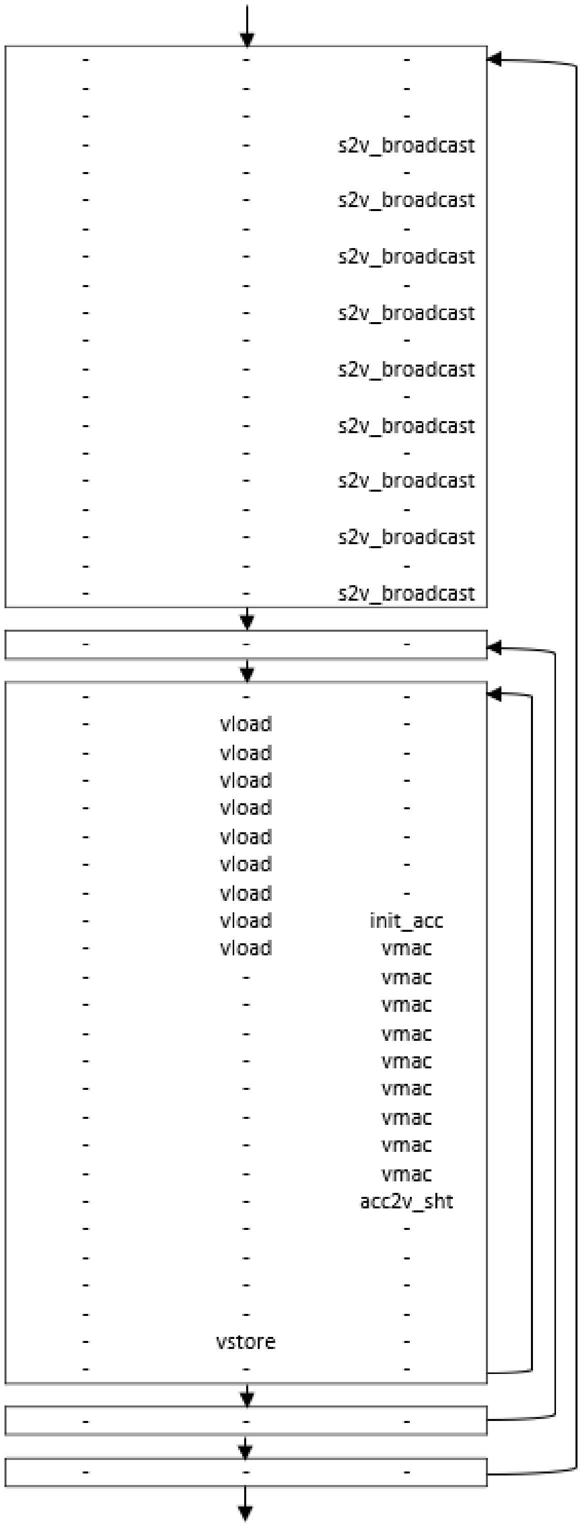
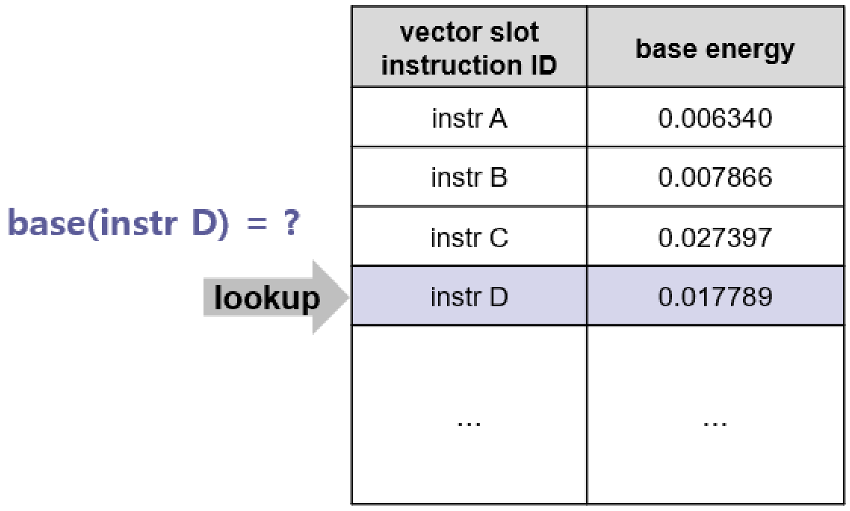


Figure 5.5: K4_DWCV_k3 kernel algorithm in estimated CFG representation

5.3.2. Kernel Energy Estimator

In this step, the energy estimator finally calculates the estimated energy of kernel execution by utilizing the estimated control flow graph (CFG) generated in the previous step and referencing a pre-built energy table. We have created two types of energy tables, which were obtained from experimental measurements as described in Section 4.2.

Estimating Base Energy The first energy table is the base energy table for memory and vector slots. Each entry in the table is one-to-one mapped to a specific slot instruction and contains the base energy value. As shown in Figure 5.6, the base energy of each instruction from the estimated CFG can be directly retrieved by looking up the base energy table.



vector slot instruction ID	base energy
instr A	0.006340
instr B	0.007866
instr C	0.027397
instr D	0.017789
…	…

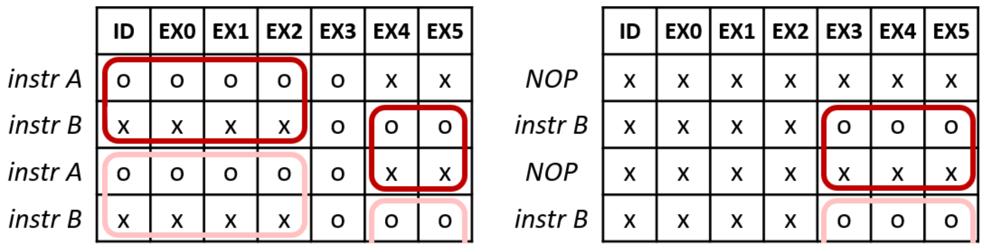
Figure 5.6 Example of base energy table lookup

Estimating Inter-Instruction Energy The other energy table is the inter-instruction energy table for NOP-pairs. In the case of the memory slot, a single constant value is sufficient to model inter-instruction energy as explained in Section 4.2.1. However, for the vector slot, we need to consider inter-instruction energy for each pair of instructions. To improve modeling efficiency, we propose a novel method where we only measure the inter-instruction energy for pairs with NOP and perform analytical conversion to estimate the inter-instruction energy for all pairs.

In Section 4.2.2, it was discovered that the inter-instruction energy consumption of vector slot instructions primarily originates from the data path of the vector processor, due to its application-specific ISA and SIMD architecture. When the type of executed instruction changes, different parts of the circuit are utilized, and during this process, various factors cause inter-instruction energy consumption. Among the factors, a primary consideration in the target vector processor microarchitecture is the multiplexing of input wires within the modules.

Input wire multiplexing in the vector processor is implemented as a means of operand isolation, a commonly employed power-saving technique. It serves to prevent redundant switching activity in idle combinational logics. As a consequence of this multiplexing, operand wires that were utilized in the previous instruction but are not required in the current instruction undergo a transition from their previous values to zero. That is to say, switching activity occurs due to the utilization of different parts of the circuit (e.g., pipeline registers, functional units and internal wires) by consecutive instructions, causing inter-instruction energy consumption, with the energy dissipation varying across different instruction pairs. To precisely evaluate this effect, we need to consider all sequential instruction pairs at the wire and bit-level through RTL synthesis and simulation, which demands significant engineering time and effort.

Instead, we propose a method to simply calculate the estimated inter-instruction energy for each instruction pair based on pipeline stage usage, as depicted in Figure 5.7. To obtain the inter-instruction energy of instruction A and B sequence, denoted as $inter(A, B, A, B, \ldots)$, we first determine the number of stage transitions from enabled to disabled stages, referred to as *E2D_stages*, in both *A*→*B*→*A*→... and *nop*→*B*→*nop*→*B*→... sequences. We then calculate the scale factor based on the ratio of *E2D_stages*, as shown in Figure 5.7. Finally, the estimated inter-instruction energy of instruction A and B sequence is obtained by scaling the inter-instruction energy with NOP using the scale factor, as in Equation (5.1). Here, the inter-instruction energy with NOP is directly retrieved by looking up the inter-instruction energy table, as depicted in Figure 5.8.



	ID	EX0	EX1	EX2	EX3	EX4	EX5
instr A	o	o	o	o	o	x	x
instr B	x	x	x	x	o	o	o
instr A	o	o	o	o	o	x	x
instr B	x	x	x	x	o	o	o

	ID	EX0	EX1	EX2	EX3	EX4	EX5
NOP	x	x	x	x	x	x	x
instr B	x	x	x	x	o	o	o
NOP	x	x	x	x	x	x	x
instr B	x	x	x	x	o	o	o

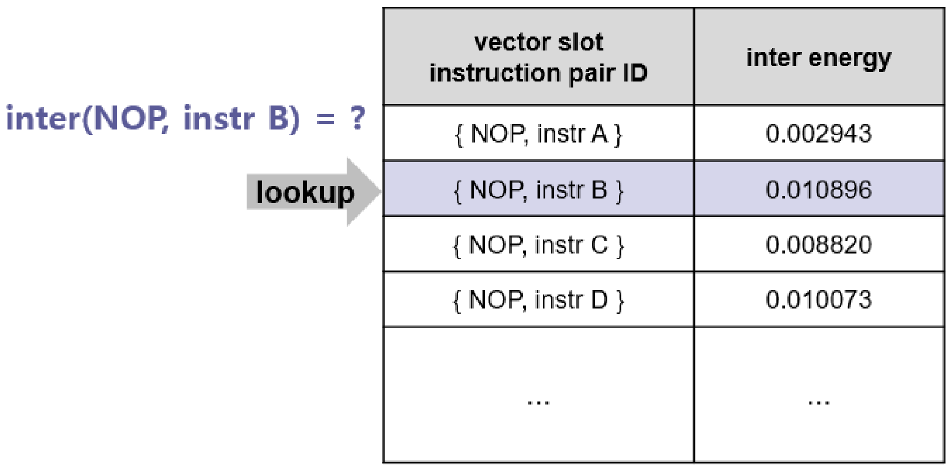
$E2D\_stages(A, B, A, B, \ldots) = 6$ $\quad$ $E2D\_stages(nop, B, nop, B, \ldots) = 3$

$$\therefore scale\_factor(A, B) = \frac{E2D\_stages(A,B,A,B,\ldots)}{E2D\_stages(nop,B,nop,B,\ldots)} = 2$$

Figure 5.7 Inter-instruction energy scale factor calculation

$$inter(A, B, A, B, \ldots)$$

$$= inter(nop, B, nop, B, \ldots) * scale\_factor(A, B) \qquad (5.1)$$



vector slot instruction pair ID	inter energy
{ NOP, instr A }	0.002943
{ NOP, instr B }	0.010896
{ NOP, instr C }	0.008820
{ NOP, instr D }	0.010073
...	...

Figure 5.8 Example of inter-instruction energy table lookup

By employing this method, it is no longer needed to collect measurements for all instruction pairs. Instead, only pairs consisting of an instruction and a NOP are considered, while still accounting for the circuit state changes in all instruction pairs. The inter-instruction energy values for all {vector slot instruction, NOP} pairs are contained in each entry of the inter-instruction energy table, with a table size of $O(n)$ instead of $O(n^2)$.

Estimating Kernel-Level Energy Consumption The energy estimator calculates the estimated energy of the kernel by combining the instruction base energy and inter-instruction energy over the estimated CFG. The algorithm for calculating the kernel-level estimated energy is described in Algorithm 2.

In phase 1 of the algorithm, the estimated energy of each node and edge in the CFG is calculated. Each CFG node corresponds to a basic block, which is a sequence of instructions. The node energy is calculated as the sum of the base energy of all instructions in the basic block, as well as the inter-instruction energy of all sequential instruction pairs within the basic block. A CFG edge represents the control flow path from the last instruction of the source node to the first instruction of the destination node. Thus, the edge energy is calculated as the inter-instruction energy between these two instructions.

In phase 2, the estimation of the entire kernel execution energy is calculated. While parsing the entire CFG, the energy of each node and edge is accumulated. The energy of a single node is multiplied by its basic block iteration count. The energy of a single edge is multiplied by the number of times the edge is taken, which is directly calculated from the iteration count of the connected nodes. Unlike in general processor kernels where control flow is complex, DNN layer kernels are structured in simple nested loops, with each basic block having no more than 2 incoming and outgoing edges, making it possible to determine how many times an edge is taken based on the basic block iteration count.

Repeating the above process for memory and vector slot, the kernel-level estimation of each slot's energy is obtained. Finally, from the estimation values of memory, vector slots and ignored scalar slot (assumed to have zero energy), the slot additive property gives the overall kernel-level vector processor energy estimation.

The entire estimation process, which involves 4 analytical steps of algorithm parsing, graph transformation, runtime analysis, and energy calculation, does not require any compile-able kernel C code, compiled assembly code, actual hardware execution, or low-level simulation. This approach enables fast evaluation of the energy consumption without the need for extensive hardware resources or time-consuming simulations. Instead, the estimation process relies on high-level analysis and modeling techniques to estimate the energy consumption of the kernel.

Algorithm 2 Estimated energy calculation algorithm pseudocode

```
1:  // phase 1-1: CFG node calculation
2:  for node: basic blocks of estimated CFG do
3:      for instr: instructions in node do
4:          node_energy[node] += base(instr)
5:      end for
6:      for {instr_1, instr_2}: sequential instruction pairs in node do
7:          node_energy[node] += inter({instr_1, instr_2})
8:      end for
9:  end for
10: // phase 1-2: CFG edge energy calculation
11: for edge: edges of estimated CFG do
12:     edge_energy[edge] += inter({edge_src_instr, edge_dst_instr})
13: end for
14: // phase 2: total kernel energy calculation
15: for node: basic blocks of estimated CFG do
16:     graph_energy += node_energy[node] * node_iteration[node]
17: end for
18: for edge: edges of estimated CFG do
19:     graph_energy += edge_energy[edge] * edge_taken_times[edge]
20: end for
```

Chapter 6

Experimental Results

6.1. Experimental Environments

To obtain the ground truth energy consumption, we conducted power measurements by capturing instantaneous voltage and current values from the on-chip system power management bus. Due to the limited speed of the I2C interface, we were only able to obtain instantaneous power readings every 20 ~ 30 milliseconds. To ensure reliability, we executed a single kernel repeatedly until 500 power measurement samples are collected. The average value of these samples was then multiplied by the cycle count obtained from the on-chip performance counter to calculate the energy consumption of the kernel execution.

Running the entire modeling process in Python 3.8 once to obtain the estimated energy value for a single layer execution takes approximately 300 milliseconds on an Intel Xeon Gold 6242R processor.

6.2. Kernel-Level Energy Estimation

In our evaluation, we applied our energy modeling method to eight target kernels listed in Table 3.1. We tested the method on 100 different layer configurations with random data and filter shapes for each kernel.

Figure 6.1 illustrates the average estimation accuracy of our proposed model, which is depicted in green bars, and two baseline models for comparison. The first baseline model, represented by the orange color, considers only the instruction base energy (*base-only* model). The other baseline model, depicted in yellow, incorporates both the base energy and inter-instruction energy by simply using the NOP-pair energy as the inter-instruction energy for all instruction pairs (*base + NOP* model).

The evaluation results demonstrate that ignoring inter-instruction energy leads to a considerable degradation of modeling accuracy. The worst-case accuracy is observed in the K5_DWCV kernel, where the accuracy drops to 50.39% when inter-instruction energy is neglected. In all the comparisons, the models that consider inter-instruction effects, namely the *base + NOP* model and the proposed model, outperform the *base-only* model. This indicates that incorporating inter-instruction energy improves the accuracy of the energy estimation.

The proposed model outperformed the *base + NOP* model in most cases. On average, the proposed model achieved estimation accuracies ranging from 93% to 99%. This indicates that the proposed scaling method significantly improves the accuracy of the energy estimation. In particular, for kernels with frequent instruction switching in vector slot, such as K1_EADD and K5_DWCV, the proposed model exhibited a substantial improvement in estimation accuracy compared to the *base* + *NOP* model. This suggests that the proposed method effectively captures the inter-instruction energy consumption patterns, leading to more accurate energy estimations.

The proposed model showed relatively poor performance in nearest neighbor up-sampling kernels, K6_UPS_k2 and K7_UPS. These kernels follow a simple algorithm of duplicating data pixels, primarily involving vector load and store operations without any vector slot instructions. Moreover, memory operations are executed sparsely due to long memory access latency, resulting low processor utilization and low power consumption. Consequently, the ignored scalar operations, such as address calculation and control flow management, become more prominent, leading to a degradation in estimation accuracy for these specific kernels. To address this weakness, a minor compensating term for up-sampling pixel address calculation energy is added at the dominant innermost loop.

Comparisons between measured and estimated energy consumption for all test cases are presented in Figure 6.2 to Figure 6.9, with energy values scaled as in Section 4.2. These figures clearly show that the estimated energy consumption accurately reflects the variation in energy consumption across different layer configurations.

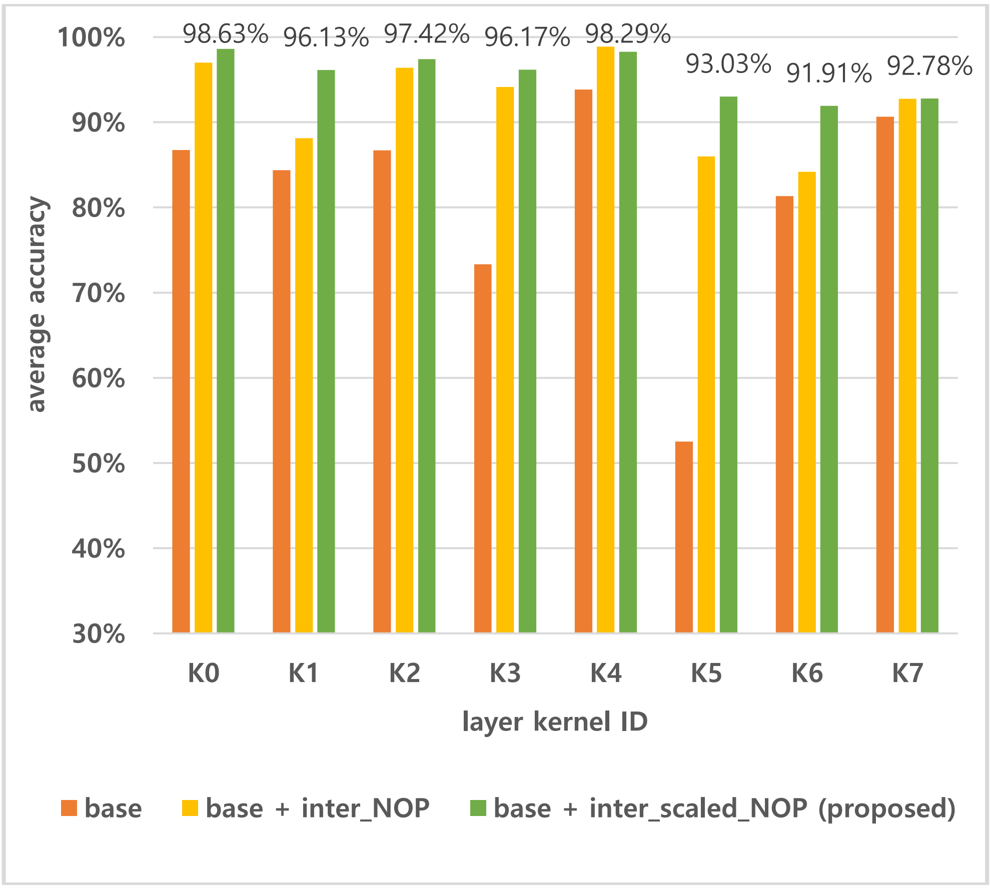


Figure 6.1: Average estimation accuracy on 100 test cases of 8 target layer kernels

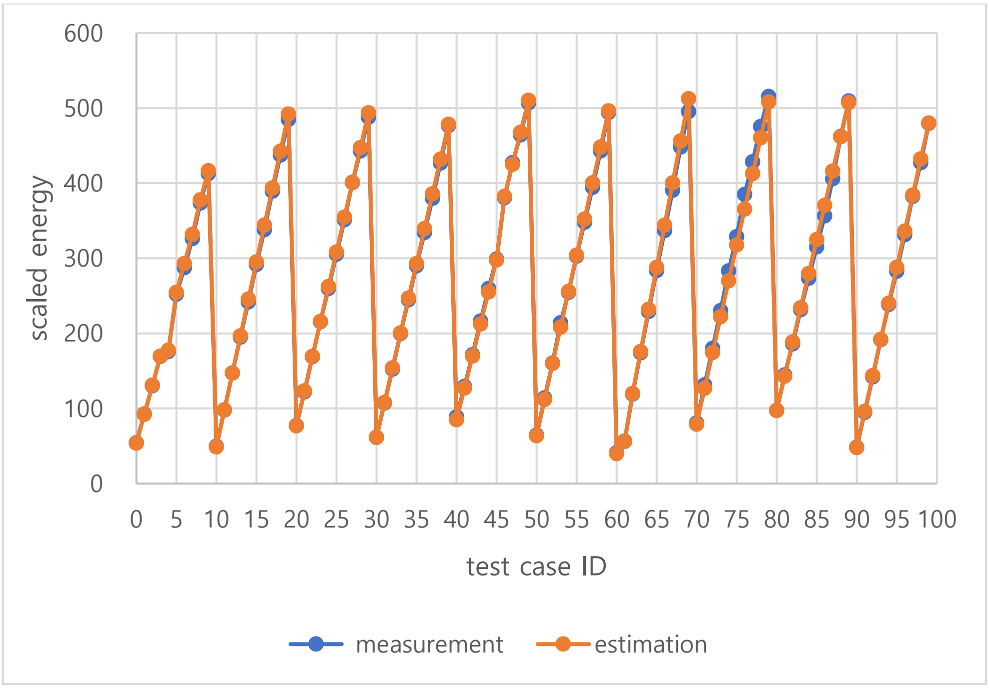


Figure 6.2: K0_ACTV energy measurement and estimation

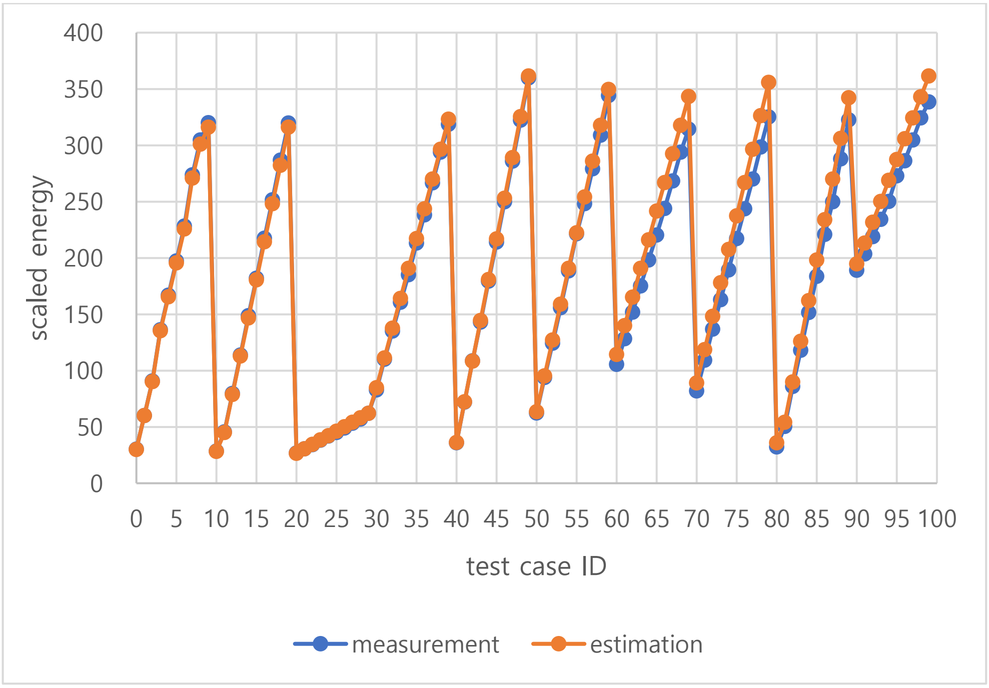


Figure 6.3: K1_EADD energy measurement and estimation

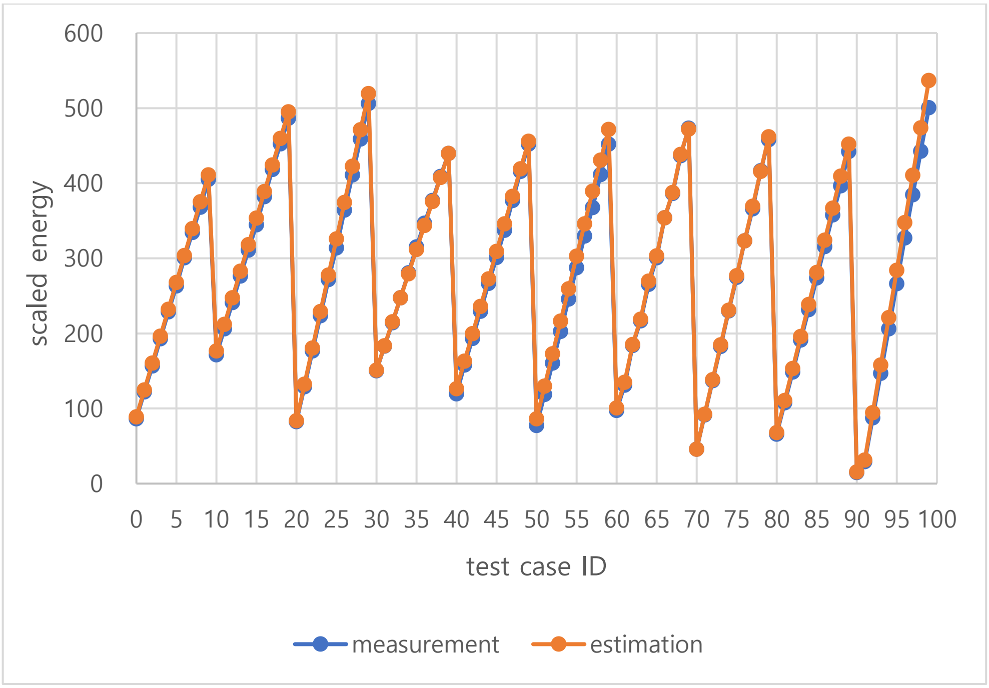


Figure 6.4: K2_MAXP_k3s2 energy measurement and estimation

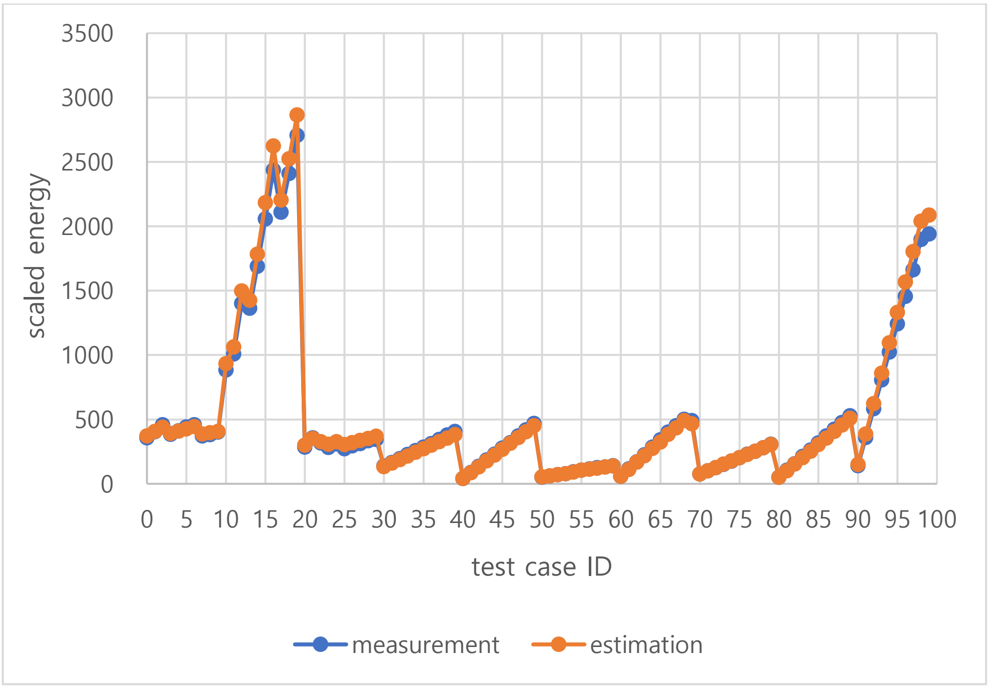


Figure 6.5: K3_GAP energy measurement and estimation

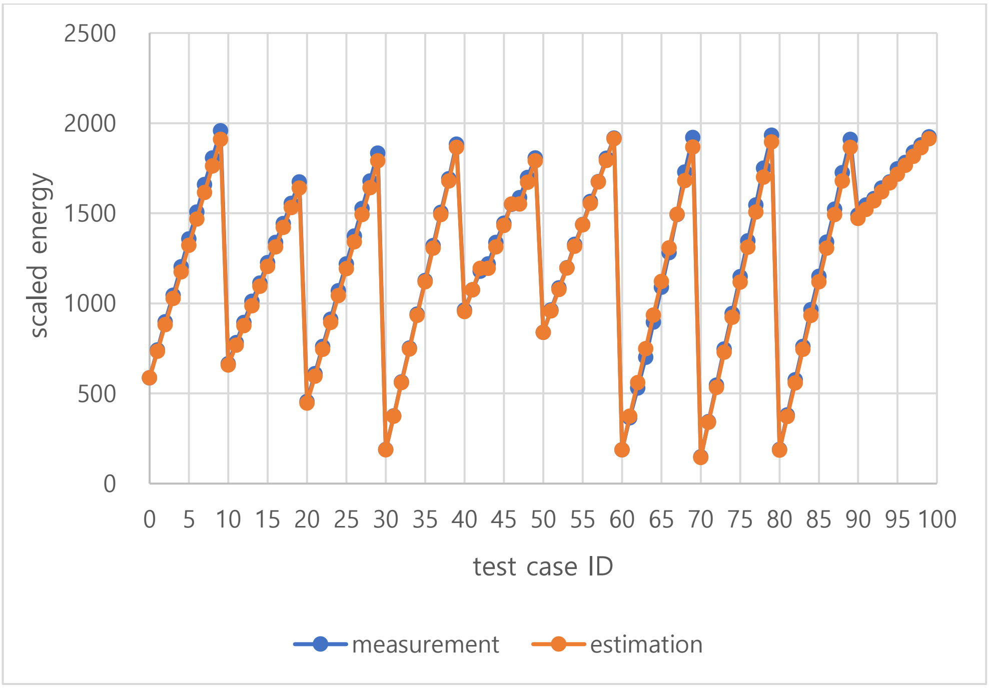


Figure 6.6: K4_DWCV_k3 energy measurement and estimation

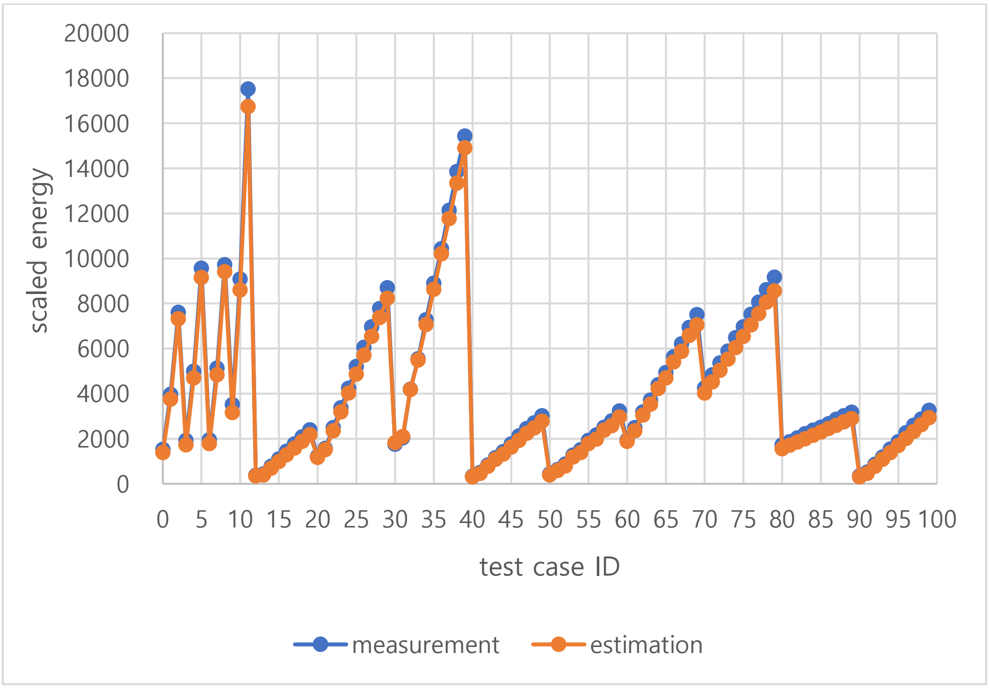


Figure 6.7: K5_DWCV energy measurement and estimation

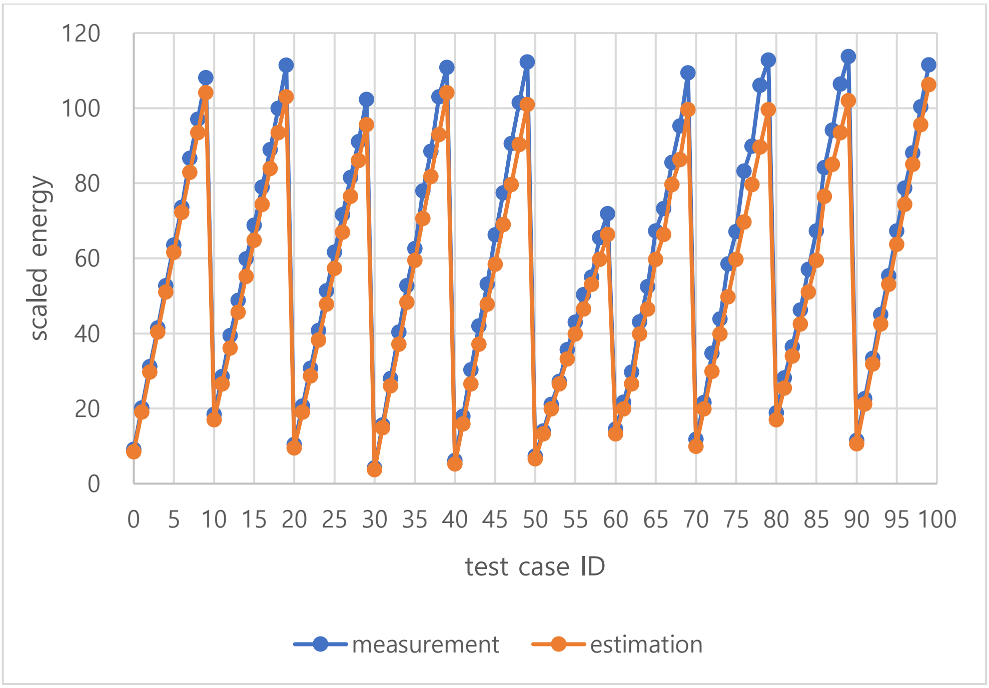


Figure 6.8: K6_UPS_k2 energy measurement and estimation

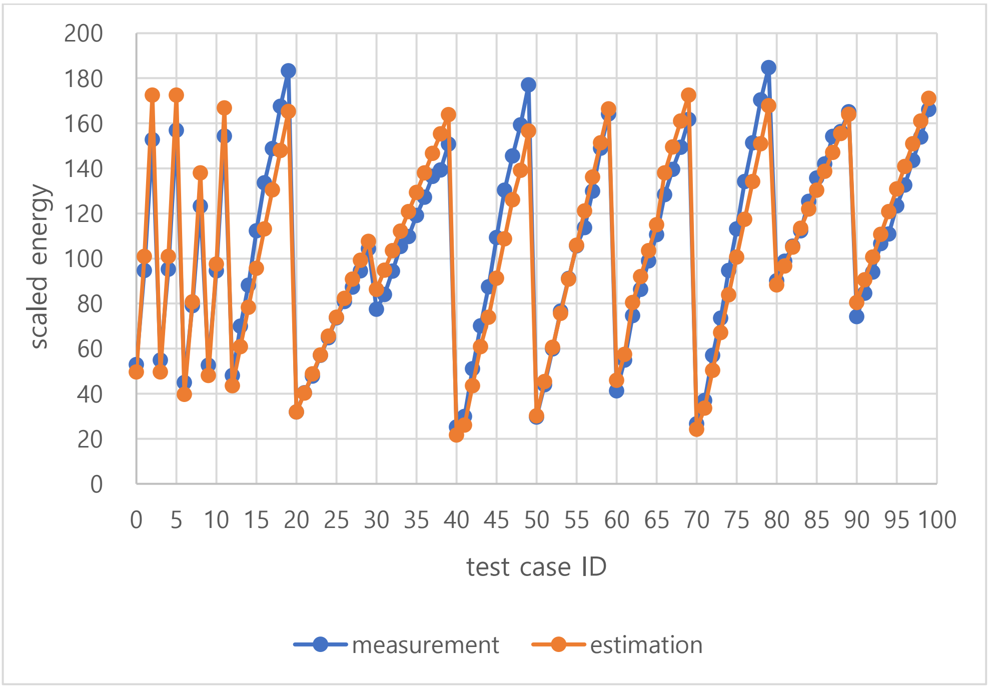


Figure 6.9: K7_UPS energy measurement and estimation

Chapter 7

Conclusion

This work proposes an analytical energy modeling method for estimating the energy consumption of a vector processor kernel execution. The method efficiently calculates the estimated energy by analyzing and focusing on energy-significant factors. It models energy consumption using novel analytical methods of lowering major operations down to the instruction level through graph transform and calculating inter-instruction energy consumption on the data path based on pipeline stage usage. Experimental results on 8 different DNN layer kernels, with 100 tests cases for each, demonstrate that the proposed method achieves an average estimation accuracy of 95.52%. Furthermore, the estimation process is fast, taking only 300 milliseconds on Intel Xeon Gold 6242R processor.

Bibliography

[1] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, “Estimation of energy consumption in machine learning,” *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, Dec. 2019.

[2] M. C. Walker et al., “Accurate and stable run-time power modeling for mobile and mmbedded CPUs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, Jan. 2017.

[3] R. Bertran et al., "Decomposable and responsive power models for multicore processors using performance counters,” in *Proceedings of the 24th ACM International Conference on Supercomputing*, Jun. 2010, pp. 147-158.

[4] T. Chen et al. (2018), “TVM: An automated end-to-end optimizing compiler for deep learning,” arXiv:1802.04799, [Online]. Available: https://arxiv.org/abs/1802.04799

[5] A. Parashar et al., “Timeloop: A systematic approach to DNN accelerator evaluation,” in *IEEE International Symposium on Performance Analysis of Systems and Software*, Mar. 2019, pp. 304-315.

[6] A. Bona et al., “Energy estimation and optimization of embedded VLIW processors based on instruction clustering,” in *Proceedings of the 39th annual Design Automation Conference*, Jan. 2002, pp. 886-891.

[7] B. Klass, D. E. Thomas, H. Schmit, and D. F. Nagle, “Modeling inter-instruction energy effects in a digital signal processor,” in *Proceedings of the Power-Driven Microarchitecture Workshop*, Jun. 1998.

[8] D. J. Brook, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *International Symposium on Computer Architecture*, May 2000, pp. 83-94.

[9] S. Li et al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009, pp. 469-480.

[10] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "An instruction-level energy model for embedded VLIW architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 9, pp. 998-1010, Sep 2002.

[11] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Proceedings of the 37th Annual Design Automation Conference*, Jun. 2000, pp. 340-345.

[12] H. Kwon et al., "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings." *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May 2020.

[13] Y. S. Shao, and D. J. Brooks. "Energy characterization and instruction-level energy model of Intel's Xeon Phi processor," in *International Symposium on Low Power Electronics and Design*, Sep. 2013, pp. 389-394.

[14] V. Tiwari, S. Malik, A. Wolfe, and M. T. C. Lee, "Instruction level power analysis and optimization of software." *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 13, no. 2–3, pp. 223–238, Aug. 1996.

[15] A. Bona et al., "Reducing the complexity of instruction-level power models for VLIW processors," *Design Automation for Embedded Systems*, vol. 10, no. 1, pp. 49–67, Mar. 2005.

[16] M. T. C. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and minimization techniques for embedded DSP software." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 1, pp. 123–135, Mar. 1997.

국문 초록

프로세서 에너지 모델은 오랜 기간동안 광범위하게 연구되어 왔다. 특히, 딥러닝 프로세서 모델링에 관한 연구는 최근에 큰 관심을 받고 있다. 딥러닝 프로세서의 에너지 소비량을 예측하는 것은 하드웨어 아키텍처, 소프트웨어 최적화, 데이터 및 연산 매핑 공간 탐색 및 신경망 구조 탐색(NAS)에 걸친 여러 레벨의 설계에서 중요한 역할을 한다. 정확한 에너지 예측을 위해서는 명령어 별 에너지에 더해서 명령어 간 영향까지 고려하는 것이 필요하다.

적은 오버헤드로 에너지 소비를 정확하게 모델링하기 위해, 우리는 대상 프로세서 아키텍처의 명령어 수준 에너지 동작 특성을 분석하였다. 이 분석을 바탕으로, 명령어 간 영향을 포함한 주요 전력 소비 요소를 고려해 에너지 소모를 예측하는 간단한 해석적 접근 방식을 고안하였다. 제안된 모델링 방법은 평균 95.52%의 커널 수준 에너지 예측 정확도와 빠른 예측 시간을 보여준다.

주요어: 해석적 전력 소모 모델, 벡터 처리 장치 (VPU), VLIW 프로세서, 신경망 처리 장치 (NPU)

학번: 2021-26145