JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 30 No. 12, pp. 15-24, December 2025
https://doi.org/10.9708/jksci.2025.30.12.015

Channel-Wise Thread Indexing for Performance Improvement of Decomposable Winograd Convolution

Wonho Lee*, Jong Wook Kwak**

*Researcher, Dept. of Computer Engineering, Yeungnam University, Gyeongsan, Korea
**Professor, Dept. of Computer Engineering, Yeungnam University, Gyeongsan, Korea

[Abstract]

Convolutional neural networks (CNNs) often require large receptive fields, making acceleration algorithms and GPU kernel configurations key factors in optimizing inference performance. In decomposable Winograd convolution algorithms, previous thread indexing methods lead to thread divergence, where threads within a warp are serialized due to differences in filter sizes. In this paper, we introduce a channel-wise thread indexing that eliminates thread divergence by mapping each convolutional filter channel to the same warp. This approach ensures uniform filter sizes across threads within a warp, significantly enhancing performance. Experiments show that the proposed method removes all potential thread divergence across diverse convolution configurations, including various filter sizes and input/output channel counts, reducing execution time up to 15% on state-of-the-art CNN models. These results demonstrate the potential for improving CNN computational efficiency on SIMT architectures.

▸ **Key words**: GPU, GPGPU(General Purpose computing on Graphics Processing Units), CUDA(Compute Unified Device Architecture), CNN(Convolutional Neural Network), thread divergence

[요 약]

최근 합성곱 신경망에서 넓은 Receptive Field를 요구하면서 합성곱 신경망의 가속 알고리즘과 GPU 커널 구성은 인공지능 모델의 추론 성능에 중요한 영향을 미친다. 분할 기반 위노그라드 합성곱 알고리즘에서 기존 스레드 인덱싱 방법은 워프 내 스레드가 처리해야 하는 필터 크기가 달라 스레드가 직렬화되는 스레드 다이버전스 현상이 발생한다. 본 논문에서는 Channel-wise thread indexing 기법을 제안해 스레드 다이버전스를 제거하고, 합성곱 알고리즘의 성능의 향상시키고자 한다. 이는 합성곱 필터의 각 채널을 같은 워프에 우선적으로 매핑해 워프 내 스레드가 처리해야 하는 필터 크기를 균일하게 맞춰 스레드 다이버전스를 제거한다. 기존 기법과 비교한 결과, 다양한 필터 크기, 입출력 채널 개수의 합성곱 구성에서 발생가능한 모든 스레드 다이버전스를 제거했고, 최신 합성곱 신경망 모델에서 실행 시간을 15%까지 단축했다. 본 연구 결과는 SIMT 구조에서 합성곱 신경망의 연산 효율성을 향상시키는 데 기여했다.

▸ **주제어**: GPU(Graphics Processing Units), GPGPU(General Purpose computing on GPU), CUDA(Compute Unified Device Architecture), CNN, thread divergence

• First Author: Wonho Lee, Corresponding Author: Jong Wook Kwak
*Wonho Lee (gh9908@gmail.com), Dept. of Computer Engineering, Yeungnam University
**Jong Wook Kwak (kwak@yu.ac.kr), Dept. of Computer Engineering, Yeungnam University
• Received: 2025. 10. 21, Revised: 2025. 11. 23, Accepted: 2025. 12. 01.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

합성곱 신경망(Convolutional Neural Network, CNN)은 딥러닝의 핵심 기술로, 이미지 분류, 객체 탐지, 자연어 처리 등 다양한 분야에서 뛰어난 성능을 발휘한다[1-3]. 특히, CNN의 핵심적인 합성곱 연산은 이미지나 영상의 복잡한 공간적 패턴을 효율적으로 이해하고 처리할 수 있다[4-5]. 그러나, 합성곱 연산의 높은 계산 복잡도는 대규모 모델을 GPU와 같은 병렬 연산 환경에서 효율적으로 실행하는 데 많은 제약 사항이 있다[6-8]. 따라서 효율적인 자원 활용을 통한 성능 향상을 위해, 합성곱 연산 최적화는 필수적이다.

위노그라드 합성곱 (Winograd Convolution) 알고리즘은 가장 널리 쓰이는 합성곱 가속화 알고리즘 중 하나로 수학적 변환을 활용해 연산 횟수를 줄여 합성곱 연산을 최적화한다[8]. 특히, 3x3 필터에서 위노그라드 합성곱은 이론적으로 기존 합성곱 연산보다 2.25배 적은 곱셈 연산으로 다른 가속 알고리즘들에 비해 높은 연산 효율성을 보인다[8]. 이는 AlexNet[9], ResNet[10], EfficientNet[11] 등과 같이 3x3 필터를 집중적으로 사용하는 기존 합성곱 신경망 모델에서 성공적으로 계산 복잡도를 감소시켰다[12-13].

최근 GPU와 같은 컴퓨팅 자원의 발전으로 인해, ConvNeXt[14], RepLKNet[15]와 같은 최신 합성곱 신경망 모델들은 기존 3x3 필터를 집중적으로 사용하던 합성곱 신경망 모델의 지역적 특징 파악과 더불어, 전역적인 특징 파악과 고해상도 이미지 처리에 대한 요구로 넓은 Receptive Field를 사용하기 위해, 기존보다 큰 크기의 필터를 사용한다. 하지만 위노그라드 합성곱은 출력과 필터의 크기에 따라 수학적 변환 과정이 달라져, 일관된 방식으로 다양한 크기의 필터에 적용하기 어렵다[16, 17]. 또한, 출력 또는 필터의 크기가 증가함에 따라, 기존 합성곱 연산과 연산 결과의 차이가 증가하는 수치 불안정성으로 인해, 3x3 필터에서 위노그라드 합성곱이 가지는 이점을 크기가 큰 필터로 이식하는 새로운 도전과제에 당면하게 됐다[16, 17].

이런 수치 불안정성을 해소하고, 위노그라드 합성곱의 효율성을 크기가 큰 필터로 이식하려는 도전과제를 해결하기 위해, 기존 연구들은 분할 기반 위노그라드 합성곱 방식을 제안했다[18-20]. 분할 기반 위노그라드 합성곱은 크기가 큰 필터를 작은 크기의 서브 필터들로 나눠 각 서브 필터의 크기에 맞는 수학적 변환 과정을 통해, 수치 불안정성을 해소하며, 기존 3x3과 같이 작은 크기의 필터에서의 이점인 연산 효율성을 크기가 큰 필터에 성공적으로 이식했다[18-20].

하지만, GPU(Graphic Processing Unit)와 같은 SIMT(Single Instruction, Multiple Threads) 구조의 특성을 충분히 반영하기 어렵다. GPU는 여러 스레드를 하나의 워프(warp)라는 그룹으로 묶어, 같은 워프에 속한 스레드들은 같은 명령어 흐름을 가진다[21-22]. 이런 GPU의 특성으로 인해, 조건에 따른 분기문을 실행해야 하는 경우, 같은 워프에 있는 스레드들이 서로 다른 조건을 만족할 때, 스레드들의 실행 흐름이 서로 달라, 다른 실행 흐름을 갖는 스레드들이 대기하게 되는 스레드 다이버전스 현상이 발생한다[23-25]. 이는 스레드들의 병렬 실행을 직렬화시켜, GPU의 병렬 처리 성능을 저해한다. 이를 방지하기 위해, CUDA(Compute Unified Device Architecture)와 같은 GPGPU(General Purpose computing on GPU) 커널 레벨에서 스레드의 구성을 조정할 필요가 있다[26].

하지만, 분할 기반 위노그라드 합성곱은 서브 필터의 크기에 따라 다른 수학적 변환 과정을 가지기 때문에, 기존 병합된 메모리 접근을 최대한 활용하기 위한 방식의 스레드 인덱싱을 적용한다면, 각 서브 필터의 수학적 변환을 수행하는 스레드들이 서로 다른 조건에 따라 실행해, 같은 워프의 스레드들이 서로 다른 실행 흐름을 갖는 스레드 다이버전스 현상을 야기한다.

본 논문에서는 분할 기반 위노그라드 합성곱에서 동일한 분기를 수행해 같은 실행 흐름을 갖는 스레드들을 같은 워프에서 실행할 수 있도록 하여, 스레드 다이버전스를 감소시키고, 합성곱의 실행 시간을 단축하는 새로운 스레드 매핑 기법인 Channel-wise thread indexing을 제안한다. 이는 하드웨어의 구성 변경 없이 GPGPU 커널 레벨의 소프트웨어적 매핑 기법으로 스레드 다이버전스를 효과적으로 제거해 GPU와 같은 SIMT 아키텍처에서 CNN 실행 시간을 감소시킨다.

본 논문의 나머지 구성은 다음과 같다. 2장에서는 배경 지식과 관련 연구들에 대해 논의하고, 3장에서는 제안된 Channel-wise thread indexing 방법에 대해 논의한다. 4장에서는 기존 분할 기반 위노그라드 합성곱 방법과 비교 실험에 대해 제시하고 분석하며, 마지막으로, 5장에서는 결론을 맺는다.

II. Backgrounds and Related Works

1. Backgrounds

1.1. Winograd Convolution

위노그라드 합성곱은 CNN에서 합성곱 연산의 효율성을 높이기 위해 도입된 알고리즘이다[8]. 이 알고리즘은 입력 데이터와 필터를 사전 변환하여 연산량을 줄이는 방법을 제공한다. 변환된 데이터는 요소별 곱셈으로 결합되고, 다시 역변환을 통해 최종 결과를 얻는다. 위 과정은 다음과 같이 정의된다.

$$V = B^T \cdot d \cdot B \tag{1}$$

$$U = G \cdot g \cdot G^T \tag{2}$$

$$M = U \odot V \tag{3}$$

$$Y = A^T \cdot M \cdot A \tag{4}$$

여기서, d는 입력 데이터 행렬, g는 필터 행렬, Y는 결과 행렬이고, B, G, A는 각각 입력 변환, 필터 변환, 역변환을 위한 변환 행렬이다. 이 변환 행렬을 필터의 크기와 출력 데이터의 크기에 따라 정해지며, 출력 데이터의 크기가 2×2이고, 필터의 크기가 3×3일 때 각 변환 행렬은 다음과 같이 정의된다.

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix},$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}, \tag{5}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

분할 기반 위노그라드 합성곱 알고리즘은 Fig. 1에서 보이는 것처럼, 기존 위노그라드 합성곱 알고리즘의 전단계에서 필터와 입력 데이터를 분할하는 과정을 포함하고, 각 서브 필터와 서브 입력 데이터에 대해 위노그라드 합성곱 연산을 수행한 후, 다시 결합하는 방식으로 진행된다.

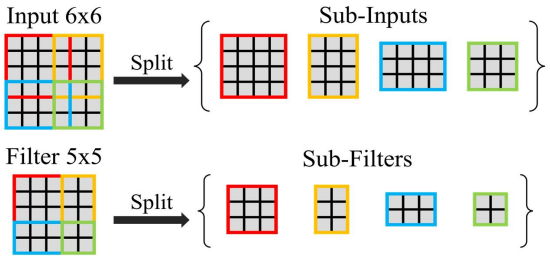


Fig. 1. Example of Decomposition-based Winograd Convolution

Fig. 1의 예시에서 5×5 크기의 필터는 4개의 서브 필터로 분할되며, 각 서브 필터의 크기는 3×3, 3×2, 2×3, 2×2로 서로 다르다. 위노그라드 합성곱의 변환 행렬은 필터 크기에 따라 결정되므로, 서로 다른 크기의 서브 필터는 각각 다른 변환 과정을 거쳐야 한다. 이는 GPU와 같은 SIMT 구조에서 동일한 워프 내의 스레드가 서로 다른 변환 과정을 수행하게 되어, 스레드 다이버전스를 유발하는 원인이 된다.

1.2. SIMT Architecture

SIMT 아키텍처는 현대 GPU에서 널리 사용되는 병렬 컴퓨팅 구조로, 하나의 명령어 흐름이 다수의 스레드를 동시에 제어하여 실행하는 방식이다[21-22]. 대표적으로 NVIDIA와 AMD의 GPU 구성은 모두 SIMT 아키텍처를 사용하지만, 세부 구현에는 차이가 있다. 이들간의 Thread/Block/Grid 계층 구조는 유사하나, NVIDIA의 GPU는 32개의 스레드를 워프로 그룹화하는 반면, AMD의 GPU는 64개의 스레드를 웨이브프론트로 그룹화한다. 본 논문에서는 NVIDIA의 CUDA 기반의 실험 및 구현을 수행하였으며, 이에 따라 워프 단위로 설명한다. 워프는 Fig. 2의 컴퓨팅 유닛에서 병렬 처리를 수행하며 모든 스레드는 동일한 명령어를 실행하므로, 동일한 작업을 병렬로 처리하는 데 매우 효율적이다. 그러나, 워프 내부의 스레드가 조건 분기 등으로 인해 다른 명령어 경로를 가지게 되면, 스레드 다이버전스가 발생한다. 스레드 다이버전스는 각 경로가 순차적으로 처리되도록 하여, 병렬 처리 효율을 저하시킨다.

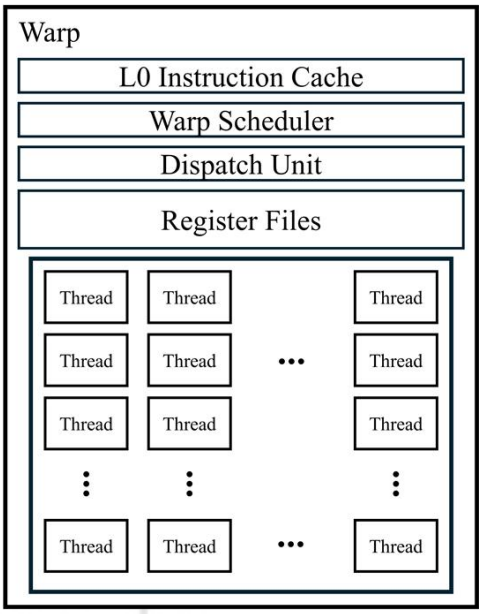


Fig. 2. Warp Structure in SIMT Architectures

SIMT 아키텍처에서 스레드는 계층적으로 구성된다. 가장 작은 단위는 개별 스레드이며, 여러 스레드가 모여 워프를 형성한다. 여러 워프는 다시 스레드 블록으로 묶이고, 이러한 스레드 블록들이 그리드 단위로 구성되어 전체 GPU의 연산 장치에서 병렬 실행된다. 이런 스레드 계층 구조 설계의 효율성에 따라 병렬 처리 효율, 메모리 접근 등 성능에 직접적인 영향을 미치기 때문에, GPU 프로그램에서는 효율적인 스레드 계층 구조 설계가 필요하다[26].

2. Related Works

2.1. Implementation of Winograd Convolution

위노그라드 합성곱은 3×3 과 같은 작은 크기의 필터를 사용하는 합성곱 신경망의 연산 효율을 높이는 기법이다. 하지만, 필터의 크기가 증가함에 따라 변환 행렬의 원소 값이 커지고, 이로 인한 부동소수점 연산 오차가 누적되어 수치 불안정성이 커진다.

이러한 수치 불안정성 문제를 해결하기 위해 분할 기반 위노그라드 합성곱 기법이 제안되었다. 이 방식은 큰 필터를 작은 크기의 서브 필터로 나누어 처리함으로써, 변환 행렬의 원소 값을 작게 유지하여 수치 불안정성을 완화한다. 그러나, 이 과정에서 서로 다른 크기의 서브 필터가 생성되며, 각 서브 필터마다 다른 변환 과정을 요구한다. GPU에서 이를 처리할 때, 같은 워프 내의 스레드가 서로 다른 크기의 서브 필터를 처리하게 되어 스레드 다이버전스가 발생한다. 위노그라드의 합성곱 구현 문제를 해결하기 위해 기존 연구들은 다양한 기법들을 제안했다.

A. Lavin and S. Gray 은 실수 보간법을 통해 변환 행렬을 직접 생성해 수치 불안정성을 해소했다[27]. 복잡한 데이터 접근 패턴을 해소하기 위해 S.A. Alam, et al 은 2차원 패턴의 행렬을 1차원 벡터의 형태로 계산하는 im2col 기법을 적용해 1차원 메모리 접근을 통해 메모리 접근 패턴을 향상시켰다[28]. 데이터 의존성을 해소하기 위해, Z. Zhang, et al은 기존 GPU 최적화 기법인 부분적 커널 합성 기법을 통해, 여러 커널에 걸친 데이터 의존성을 해결하는 방법을 제안했다[29]. 이러한 연구들은 주로 데이터 의존성 및 메모리 최적화에 중점을 두고 진행되었으며, 이들은 주로 컴파일러 수준의 최적화를 통해 스레드 다이버전스를 완화하려 했으며, 특정 하드웨어에 의존적이기 때문에 범용적으로 적용이 어렵다는 한계가 있다. 또한, SIMT 아키텍처의 근본적인 특성인 워프 단위 실행 구조를 고려한 스레드 매핑 전략은 충분히 다루지 못했다.

2.2. Optimization of Thread Divergence

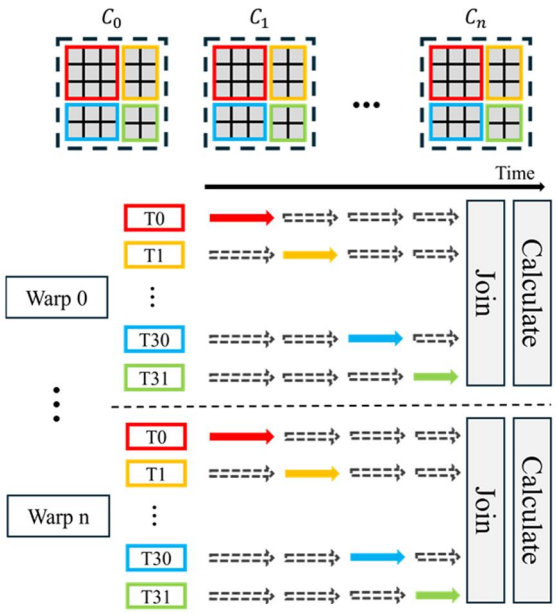
스레드 다이버전스는 병렬 처리 환경에서 동일한 워프 내의 스레드들이 서로 다른 경로를 따라갈 때 발생하여 성능 저하를 초래한다. 이를 완화하기 위해 기존 연구들에서는 컴파일러를 통한 최적화를 통해 이를 해결하려 했다.

C. Saumya, et al는 GPU에서 복잡한 제어 흐름 그래프를 분석하고, 구조적으로 유사한 제어 분기를 병합해 다이버전스를 감소시켰다[23]. C.A. Metz, et al는 CUDA 어플리케이션을 저수준 어셈블리 코드 기반으로 프로파일링해, 스레드 다이버전스를 선제적으로 파악할 수 있는 에뮬레이터를 개발했다[24]. 이런 기존 연구들은 스레드 다이버전스를 효과적으로 감소시키는 데 성공했지만, 컴파일러 최적화를 통한 방식으로 특정 GPU 마이크로아키텍처의 세부 하드웨어 특성에 의존적이며, 여러 환경에서 동일한 성능을 보장하기 어렵다.

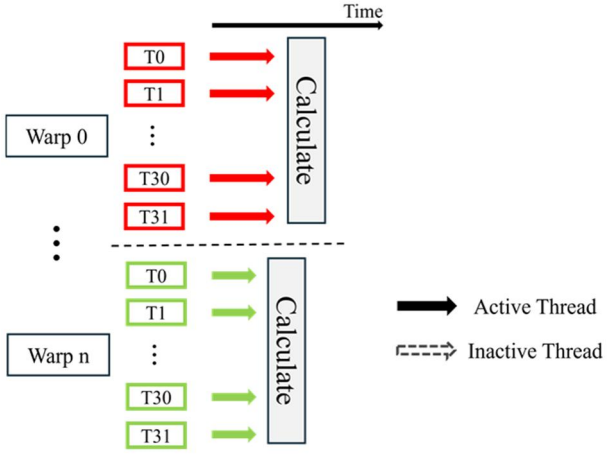
B. Cuneo and M. Bailey는 비동기 스케줄링을 통해, 스레드간 동기화를 줄여, 작업을 지연 처리하는 방식으로, 스레드 다이버전스를 감소시켰다[25]. 그러나, 런타임 기반 기법은 동적 스케줄링으로 인한 실행 시 오버헤드가 발생한다. 본 연구에서는 CUDA 커널 수준의 변환을 통해, 여러 GPU 마이크로아키텍처 환경에서도 적용이 가능하며, 런타임 오버헤드 없이 효과적으로 스레드 다이버전스를 감소시키는 기법을 제안해, 합성곱의 연산 효율성을 향상시킨다.

III. Channel-wise Thread Indexing

본 논문에서 제안하는 Channel-wise thread indexing은 스레드 계층 구조를 활용해, SIMT 아키텍처의 구조적 특성을 반영하여, 분할 기반 위노그라드 합성곱 연산에서 발생하는 스레드 다이버전스 문제를 해결한다. 합성곱 연산의 입력은 Width×Height×Channel 구조를 가지며, 분할 기반 위노그라드에서 각 채널의 필터는 동일한 방식으로 분할되므로, 서로 다른 채널의 동일한 위치의 서브 필터들은 동일한 크기를 갖는다. Channel-wise thread indexing은 이러한 특성을 활용하여 서로 다른 채널의 동일 위치 서브 필터들을 같은 워프에 배정함으로써 워프 내 모든 스레드가 동일한 크기의 서브 필터를 처리하도록 한다. 기존 기법인 Thread-wise thread indexing과, 본 연구에서 제안하는 Channel-wise thread indexing을 도식화하면 Fig. 3과 같다.



(a) Thread-wise thread indexing



(b) Channel-wise thread indexing

Fig. 3. Thread-wise and Channel-wise Thread Indexing Execution

기존 Thread-wise thread indexing 기법은 병합된 메모리 접근을 위해 스레드에 필터를 연속적으로 할당하는 방식을 사용한다. 하지만, 분할 기반 위노그라드 합성곱 알고리즘은 기존 위노그라드 합성곱 알고리즘에 추가로 분할과 결합의 단계를 포함하고 있어, 데이터의 접근 패턴이 불안정해져, 기존 방식의 이점이었던 병합된 메모리 접근의 이점을 활용할 수 없다. 또한, 분할 과정에서 생성되는 여러 크기의 서브 필터와 서브 입력 데이터들로 인해서, 각 스레드가 다른 위노그라드 변환 과정을 거쳐야한다. 이로 인해, 기존 스레드 인덱싱을 분할 기반 위노그라드 합성곱에서 채택할 경우, Fig. 3 (a)에서 보이는 것처럼, 스레드 다이버전스로 인해, 다른 실행 경로를 따르는 스레드는 비활성화되고, 추가로 분기된 실행 흐름을 다시 같은 실행 흐름으로 돌아올 수 있도록, join 과정이 추가돼야 한다. Fig. 3 (a)에서 한 워프 내의 32개 스레드(T0 ~ T31)는 한 채널 내의 서브 필터들을 순차적으로 할당받아, 같은 워프 내에서 서로 다른 크기의 서브 필터를 처리하게 된다. 이로 인해 Thread-wise thread indexing 기법은 한 크기의 서브 필터를 처리하는 동안 다른 크기를 담당하는 스레드가 대기하는 스레드 다이버전스 문제가 발생한다. 서브 필터의 크기가 4종류인 경우, 하나의 서브 필터를 처리하는 동안 다른 스레드는 대기하고 있으므로 해당 작업이 4단계로 이루어지게 된다.

반면, Fig. 3 (b)의 Channel-wise thread indexing 기법은 워프 내 모든 스레드가 동일한 크기를 처리하므로 1단계에 완료된다. 이러한 차이는 GPU의 병렬 처리 능력에 큰 영향을 미친다. 이 기법은 분할된 서브 필터와 서브 입력 데이터들을 순차적으로 스레드에 할당하지 않고, 각 스레드가 처리할 필터 채널(gid)와 서브 필터(sid)를 결정할 때 채널 인덱스를 우선으로 계산한다. 이를 통해 같은 워프 내 연속된 스레드가 서로 다른 채널의 동일 위치 서브 필터를 처리하게 된다. 해당 방식은 기존 워프 내의 스레드들이 서로 다른 크기의 서브 필터와 서브 입력 데이터에 대한 변환을 진행해 스레드 다이버전스를 발생시키던 것과는 달리, 여러 채널에 걸쳐 같은 크기의 서브 필터와 서브 입력 데이터를 우선적으로 같은 워프에 배정한다. 같은 크기의 서브 필터는 같은 위노그라드 변환 과정을 거치므로, 워프 내 모든 스레드가 동일한 실행 경로를 따르게 되어 스레드 다이버전스가 제거된다. Fig. 3 (b)에서 보이는

Algorithm 1 Parallel Filter Split (Channel-wise indexing)

Require: filter W (size: $K \times C \times H \times W$), sub-filter count SF
Ensure: Transformed Filter $\hat{W}$

procedure SPLIT WITH EXPANSION(W, SF)
  $BK \leftarrow 64$ ▷ Threads per block in K dimension
  $BC \leftarrow 8$ ▷ Threads per block in C dimension
  $BLOCK\_SIZE \leftarrow (BK, BC)$ ▷ Define thread block size
  Define threads block grid: $(\lceil K \cdot SF/(BK) \rceil, \lceil C/BC \rceil)$
  for each thread block (b_x, b_y) **in parallel do**
    Allocate shared memory array Gw[$BLOCK\_SIZE$][21]
    for each thread (t_x, t_y) **in threads block in parallel do**
      $k \leftarrow b_x$ / $Count_{subfilters}$
      $c \leftarrow b_y \cdot BC + t_y$
      $gid \leftarrow k \cdot BK + t_x$ ▷ Filter out channel index
      $sid \leftarrow b_x \bmod (\text{Count}_{subfilters})$ ▷ Sub-filter index
      if $k < K$ **and** $c < C$ **then**
        $Gw[t_x * BK + t_y][0:9] \leftarrow W[]$
        Compute $G \cdot w$ transformation
        Compute $(G \cdot w) \cdot G^T$ transformation
        Store result in $\hat{W}[c, gid, :, :, sid]$
    Synchronize threads in work-group
  return $\hat{W}$

Algorithm. 1. Parallel Filter Split (Channel-wise indexing)

것처럼, 워프 내 스레드가 일관된 명령어 흐름을 가질 수 있도록 구성하며, Warp 1 ~ Warp n-1 역시 각 워프 내에서 동일한 크기의 서브 필터만을 처리한다. 이로 인해 워프 내 모든 스레드가 동시에 활성화되어, 스레드 다이버전스로 인해 비활성화되는 스레드를 없애, 스레드의 대기 시간을 감소시킨다. 또한, 명령어 흐름이 동일하기 때문에, 실행 과정에서 명령어 흐름 제어를 위한 추가적인 join 연산이 필요하지 않다. 이는 GPU의 구조적 특성으로 인해 발생하는 스레드 다이버전스를 효과적으로 감소시킬 수 있다.

Algorithm 1은 제안된 Channel-wise thread indexing을 통한 분할 기반 위노그라드 합성곱 연산에서 필터 분할과 변환 과정에 대한 의사코드를 나타내고 있다. C와 K는 각각 합성곱의 입출력 채널을 나타내고, H와 W는 필터의 높이와 너비를 나타낸다. 2~4번 줄은 GPU 커널 함수 실행을 위한 스레드 구성 설정을 나타내고, 6번 줄에서 GPU 커널 함수를 실행한다. 11번 줄의 gid를 통해, 전체 필터 중 처리할 필터 채널을 선택하고, 12번 줄의 sid를 통해, 하나의 필터 내부에서 처리할 서브 필터를 선택한다. 특히, $Count_{subfilters}$ 변수는 분할된 서브 필터의 개수를 나타내며, 필터 크기에 따라 조정하여 다양한 필터 크기에 적용할 수 있다. 분할된 서브 필터의 개수는 필터 크기와 달리 제곱수로 결정되므로 mod 연산이 효율적으로 처리된다. 13~17번 줄은 각 크기에 맞는 필터 변환 변환을 수행하고, 저장하는 과정이다. t_x가 스레드의 가장 먼저 움직이는 방향이기 때문에, 해당 과정을 통해, 우선적으로 한 필터 내부의 서브 필터를 연속적으로 움직이는 게 아니라, 전체 필터 방향을 먼저 움직인다. 이로 인해, 각 스레드는 채널 내 서브 필터를 연속적으로 같은 워프에 할당하지 않고, 서로 다른 채널의 크기가 같은 서브 필터를 연속적으로 처리하게 된다. 이는 워프 내의 스레드가 처리하는 서브 필터가 각각 다른 크기가 아니라 같은 크기를 갖는 필터를 처리하도록 해, 스레드 다이버전스를 효과적으로 감소시킬 수 있다.

IV. Experiments

4.1. Experimental Setup

본 논문에서 제안된 Channel-wise thread indexing의 성능을 평가하기 위해 실험은 NVIDIA RTX 3080 GPU에서 CUDA 12.0[26]을 통해 수행했으며, Nsight Compute 2023.2.0.0[30]을 통해 실험 결과를 수집했다. 본 하드웨어는 다양한 합성곱의 계산 요구 사항을 다루기 위해, 널리 사용되는 GPU 구조를 반영한 기기로, 자세한 구성은 Table 1과 같다. 실험에서는 다양한 필터 크기와 입출력 채널에 따른 스레드 다이버전스 감소를 평가하기 위해, 워프 내의 활성된 스레드의 개수를 통해 스레드 다이버전스를 효과적으로 개선함을 보이려 한다.

Table 1. RTX 3080 Hardware Configuration

Configuration	Value
SMs	68
Thread / SM	128
Warp / SM	4
Threads / Warp	32

4.2. Experimental Results

본 연구에서는 실행 과정 중 워프 내 활성 스레드의 개수 변화를 사용해, 필터의 크기와 입출력 채널 개수에 따른 영향도를 분석했다. ResNet 기반의 ConvNeXt 구조에서 실제 큰 필터를 사용하는 합성곱 단계들에서 실행 시간을 분석해, 실제 합성곱 신경망에서 향상할 수 있음을 보인다.

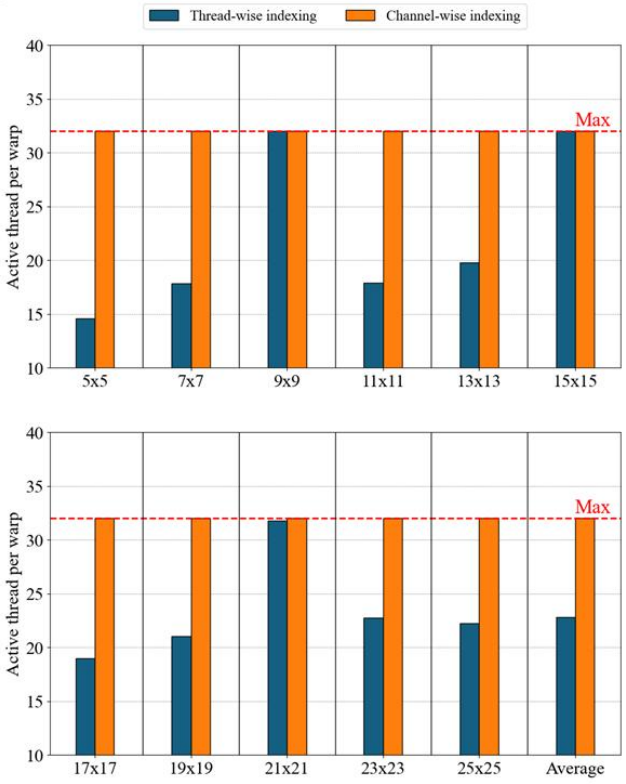


Fig. 4. Active Threads per Warp by Filter Size during Transformation

Fig. 4는 변환 과정 중 5×5에서 25×25의 크기를 가지는 필터들의 크기 변화 별 워프 당 활성 스레드의 개수를 나타내고 있다. 빨간 점선은 하드웨어가 가질 수 있는 최대 개수를 의미하며, 파란색으로 나타낸 기존 방식은 필터의 크기가 3의 배수가 되는 9×9, 15×15, 21×21는 분할 기반 위노그라드 합성곱에서 서브 필터의 크기가 모두 균일하기 때문에, 최대값을 가질 수 있지만, 이를 제외한 나머지 부분에서는 워프 내 활성 스레드 개수가 23 이하로 낮은 활성도를 보인다. 반면, 제안된 Channel-wise indexing 기법에서는 필터의 크기와 관계 없이 워프 내 활성 스레드 수가 32로 유지됨을 확인할 수 있다. 이는 워프 크기의 최댓값으로, 워프 내 모든 스레드가 활성화된 상태로 실행됨을 의미한다. 따라서 스레드 다이버전스로 인해 대기하는 스레드가 발생하지 않으며, 이는 제안된 기법이 스레드 다이버전스를 없앴다고 할 수 있다.

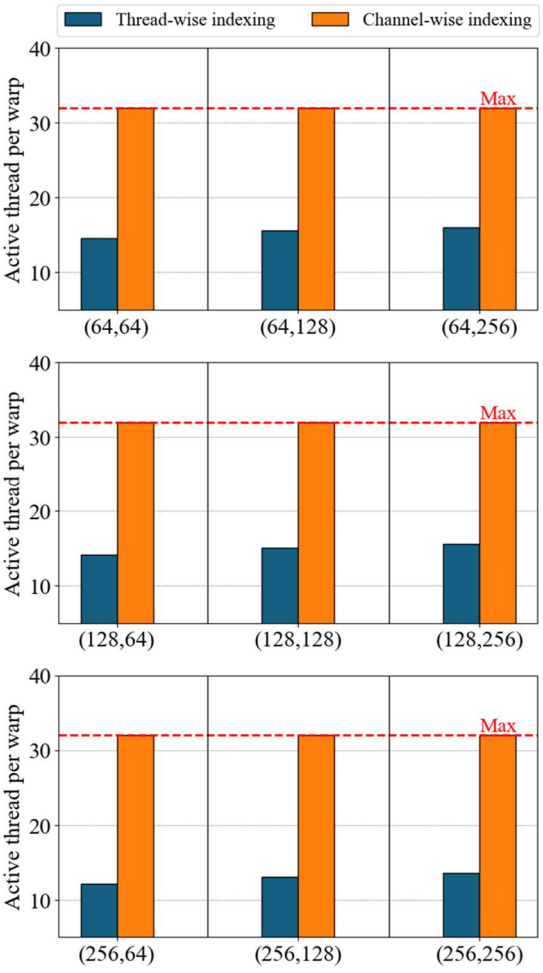


Fig. 5. Active Thread per Warp by Input/Output Channel during Transformation

Fig. 5는 변환 과정 중 합성곱의 입출력 채널이 각각 64부터 256까지 변화 하며, 그에 따른 워프 내 활성 스레드 개수를 나타내고 있다. 분할 기반 위노그라드 합성곱에서 발생하는 스레드 다이버전스는 서브 필터의 크기가 서로 다르므로 발생하는 현상이기 때문에, 필터 크기 변화와는 다르게, 입출력 채널의 변화를 통해서 기존 방식은 스레드 다이버전스가 감소하는 현상을 확인할 수 없고, 전체적으로 워프 내 활성 스레드 수가 15 이하로 낮은 활성도를 보인다. 반면, 제안된 Channel-wise indexing 기법은 입출력 채널에 상관없이 항상 워프 내 스레드가 모두 활성화 상태로 진행해, 스레드 다이버전스가 발생하지 않음을 확인할 수 있다.

Table 2. Convolution Layers with Filter Size Larger than 3 in ConvNeXt[14]

Layer	Output (H×W)	Filter(C,R×S,K)
Stage1	56×56	[96,7×7,96]
Stage2	28×28	[96,7×7,192]
Stage3	14×14	[192,7×7,384]
Stage4	7×7	[384,7×7,768]

Table 2는 실제 합성곱 신경망 모델에서의 성능 평가를 위해, 최근 넓은 Receptive Field를 위해 널리 사용되는 ConvNeXt 모델의 레이어 중 3×3보다 큰 필터를 사용하는 레이어들을 나타낸다.

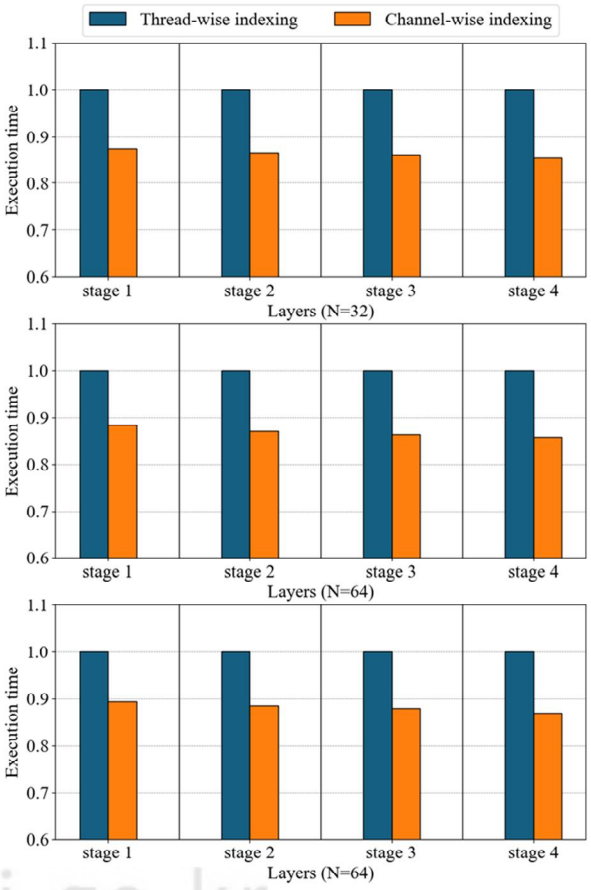


Fig. 6. Execution Time by ConvNeXt Layers

Fig. 6은 ConvNeXt 각 레이어와 배치 크기에 따른 실행 시간을 기존 방식을 기준으로 정규화하여 보인다. 그림 6에서 보이는 것처럼, 배치 크기의 변화에도 ConvNeXt의 모든 레이어에서, 실행 시간이 감소함을 확인할 수 있다. 제안된 Channel-wise indexing 기법은 분할 기반 위노그라드 합성곱의 서브 필터 크기가 서로 달라 변환 과정 중에 발생하는 스레드 다이버전스 현상을 감소시켜, 전체 실행 시간을 최소 11%에서 최대 15%까지 감소시켜 합성곱 신경망의 성능을 향상시킬 수 있었다.

한편, 제안된 기법은 채널을 우선적으로 스레드에 매핑하기 위해 기존의 단순 선형 인덱싱 방식과 달리, 필터 채널 인덱스와 서브 필터 인덱스를 별도로 계산해야 한다. 이로 인해 제안된 기법에서는 각 스레드에서 나눗셈 및 나머지 연산이 추가된다. 그러나, Fig. 6의 실험 결과에서 전체 실행 시간이 감소한 것은 인덱싱 연산에 따른 오버헤드가 전체 성능에 영향을 주지 않음을 보여준다.

V. Conclusions

본 논문에서는 분할 기반 위노그라드 합성곱 연산 시 발생하는 스레드 다이버전스 문제를 해결하는 Channel-wise thread indexing 방법을 제안했다. 이 방법은 분할되는 서브 필터와 서브 입력 데이터에 대해, 각 채널을 우선적으로 스레드에 순차적으로 배치해, 같은 명령어 실행 흐름을 갖는 스레드들이 같은 워프에 배치될 수 있도록 하는 방법이다. 이는 추가적인 하드웨어 변경이나 컴파일러 최적화 없이, 워프 내의 스레드들이 같은 명령어 흐름을 갖도록 한다. 실험 결과 Channel-wise thread indexing은 분할 기반 위노그라드 합성곱 연산의 스레드 다이버전스를 없애, 변환 과정에서의 활성 스레드 수를 최대화했다. 게다가, Channel-wise thread indexing은 분할 기반 위노그라드 합성곱의 실행 시간을 최대 15%까지 감소시켜, 추가적인 하드웨어 변경 없이 GPU와 같은 SIMT 구조에서 연산 효율성을 향상했다.

또한, 제안된 기법은 특정 CNN 구조에 종속되지 않고 범용적으로 적용이 가능하다. Channel-wise indexing 기법은 CNN의 네트워크 구조나 레이어 구성을 변경하지 않고, CUDA 커널 레벨에서 스레드와 데이터 간의 매핑 방식만을 조정한다. 본 논문의 실험에서 다양한 필터 크기(5×5, 25×25)와 입출력 채널 구성(64, 256)에서 일관된 성능 향상을 확인하였으며, 이는 제안 기법이 분할 기반 위노그라드 합성곱을 사용하는 다양한 CNN 구조에 동일하게 적용될 수 있음을 보여준다. 따라서 ConvNeXt 뿐만 아니라 RepLKNet, SLaK 등 넓은 Receptive Field를 위해 큰 필터를 사용하는 다양한 CNN 모델에서도 본 기법을 통해 연산 효율성을 향상할 수 있을 것으로 기대된다.

ACKNOWLEDGEMENT

This research was supported by the Gyeongsangbuk-do RISE (Regional Innovation System & Education) project [2025-RISE-B00805290023321].

REFERENCES

[1] F. Xu, S. Mei, G. Zhang, N. Wang, and Q. Du, “Bridging cnn and transformer with cross attention fusion network for hyperspectral image classification,” IEEE Transactions on Geoscience and Remote Sensing, 2024. DOI: 10.1109/TGRS.2024.3419266

[2] P. Sun, R. Zhang, Y. Jiang, T. Kong, et al., “Sparse r-cnn: An end-to-end framework for object detection,” IEEE transactions on pattern analysis and machine intelligence, 2023. DOI: 10.1109/TPAMI.2023.3292030

[3] K.S. Varshitha, C.G. Kumari, M. Hasvitha, et al., “Natural language processing using convolutional neural network,” 2023 7th International Conference on Computing Methodologies and Communication (ICCMC), pp.362-367, IEEE, 2023. DOI: 10.1109/ICCMC56507.2023.10083608

[4] M.M. Taye, “Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions,” Computation, vol.11, no.3, p.52, 2023. DOI: 10.3390/computation11030052

[5] S. Cong and Y. Zhou, “A review of convolutional neural network architectures and their optimizations,” Artificial Intelligence Review, vol.56, no.3, pp.1905-1969, 2023. DOI: 10.1007/s10462-022-10213-5

[6] A. Vasudevan, A. Anderson, and D. Gregg, “Parallel multi channel convolution using general matrix multiplication,” 2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP), pp.19-24, IEEE, 2017. DOI: 10.1109/ASAP.2017.7995254

[7] V. Podlozhnyuk, “Fft-based 2d convolution,” https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_64_website/projects/convolutionFFT2D/doc/convolutionFFT2D.pdf

[8] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” Proceedings of the IEEE conference on computer vision and pattern recognition, pp.4013-4021, 2016. DOI: 10.1109/CVPR.2016.435

[9] A. Krizhevsky, I. Sutskever, and G.E. Hinton, “Imagenet classification with deep convolutional neural networks,” Advances in neural information processing systems, vol.25, 2012. DOI: 10.1145/3065386

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Proceedings of the IEEE conference on computer vision and pattern recognition, pp.770-778, 2016. DOI: 10.1109/CVPR.2016.90

[11] M. Tan, “Efficientnet: Rethinking model scaling for convolutional neural networks,” arXiv preprint arXiv:1905.11946, pp.6105-6114, 2019. DOI: 10.48550/arXiv.1905.11946

[12] P. Mori, L. Frickenstein, S.B. Sampath, et al., “Wino vidi vici: Conquering numerical instability of 8-bit winograd convolution for accurate inference acceleration on edge,” Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp.53-62, 2024. DOI: 10.1109/WACV57701.2024.00013

[13] C. Yang, Y. Meng, J. Xi, et al., “Wra-ss: A high-performance accelerator integrating winograd with structured sparsity for convolutional neural networks,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2023. DOI: 10.1109/TVLSI.2023.3330993

[14] Z. Liu, H. Mao, C.Y. Wu, et al., “A convnet for the 2020s,” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp.11976-11986, 2022. DOI: 10.1109/CVPR52688.2022.01167

[15] X. Ding, X. Zhang, J. Han, and G. Ding, “Scaling up your kernels to 31x31: Revisiting large kernel design in cnns,” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp.11963-11975, 2022. DOI: 10.1109/CVPR52688.2022.01166

[16] X.Xue, H. Huang, C. Liu, et al., “Winograd convolution: A perspective from fault tolerance,” Proceedings of the 59thACM/IEEE Design Automation Conference, pp.853-858, 2022. DOI: 10.1145/3489517.3530531

[17] B. Barabasz, A. Anderson, K.M. Soodhalter, and D. Gregg, “Error analysis and improving the accuracy of winograd convolution for deep neural networks,”ACM Transactions on Mathematical Software (TOMS), vol.46, no.4, pp.1-33, 2020. DOI: 10.1145/3412380

[18] D. Huang, X. Zhang, R. Zhang, et al., “Dwm: A decomposable winograd method for convolution acceleration,” Proceedings of the AAAI Conference on Artificial Intelligence, pp.4174-4181, 2020. DOI: 10.1609/aaai.v34i04.5838

[19] J. Yepez and S.B. Ko, “Stride 2 1-d, 2-d, and 3-d winograd for convolutional neural networks,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.28, no.4, pp.853-863, 2020. DOI: 10.1109/TVLSI.2019.2961602

[20] D. Huang, R. Zhang, X. Zhang, et al., “A decomposable winograd method for n-d convolution acceleration in video analysis,” International Journal of Computer Vision, vol.129, no.10, pp.2806-2826, 2021. DOI: 10.1007/s11263-021-01500-9

[21] NVIDIA, “NVIDIA H100 Tensor Core GPU architecture”, https://resources.nvidia.com/en-us-tensor-core.

[22] AMD, “RDNA3 Instruction Set Architecture”, https://www.amd.com/content/dam/amd/en/documents/radeon-tech-docs/instruction-set-architectures/rdna3-shader-instruction-set-architecture-feb-2023_0.pdf.

[23] C. Saumya, K. Sundararajah, and M. Kulkarni, “Darm: control-flow melding for simt thread divergence reduction,” 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp.1-13, IEEE, 2022. DOI: 10.1109/CGO53902.2022.9741285

[24] C.A. Metz, C. Plump, B.J. Berger, and R. Drechsler, “Hybrid ptx analysis for gpu accelerated cnn inferencing aiding computer architecture design,” 2023 Forum on Specification & Design Languages (FDL), pp.1-8, IEEE, 2023. DOI: 10.1109/FDL59689.2023.10272088

[25] B. Cuneo and M. Bailey, “Divergence reduction in monte carlo neutron transport with on-gpu asynchronous scheduling,” ACM Transactions on Modeling and Computer Simulation, vol.34, no.1, pp.1-25, 2024. DOI: 10.1145/3626957

[26] NVIDIA, “CUDA C++ Programming Guide”, https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[27] S.A. Alam, A. Anderson, B. Barabasz, and D. Gregg, “Winograd convolution for deep neural networks: Efficient point selection,” ACM Transactions on Embedded Computing Systems, vol.21, no.6, pp.1-28, 2022. DOI: 10.1145/3524069

[28] Z. Zhang, P. Zhang, Z. Xu, et al., “Im2col-winograd: An efficient and flexible fused-winograd convolution for nhwc format on gpus,” Proceedings of the 53rd International Conference on Parallel Processing, pp.1072-1081, 2024. DOI: 10.1145/3673038.3673039

[29] G. Tong, R. Yan, L. Yang, et al., “Optimizing winograd convolution on gpus via partial kernel fusion,” IFIP International Conference on Network and Parallel Computing, pp.17-29, Springer, 2022. DOI: 10.1007/978-3-031-21395-3_2

[30] NVIDIA, “Nsight Compute Documentation,” https://docs.nvidia.com/nsight-compute/

Authors

Wonho Lee received B.S. and M.S. degrees in the Department of Computer Engineering from Yeungnam University, Korea, in 2023 and 2025, respectively. His current research interests include GPU architecture, accelerators, and computational optimization.

Jong Wook Kwak received a B.S. degree in Computer Engineering from Kyungpook National University, Daegu, Korea in 1998, a M.S. degree in Computer Engineering from Seoul National University, Seoul, Korea in 2001, and a Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea in 2006. From 2006 to 2007, he worked as a Senior Engineer in the SoC R&D Center, at Samsung Electronics Co., Ltd. During 2011~2012, he was a Guest Researcher at the Research Institute of Advanced Computer Technology, Seoul National University. During 2012~2013, he was a Visiting Scholar at the Georgia Institute of Technology, Atlanta, GA, USA. As a Head Director, he led DREAM Software Human Resource Training Center from 2014~2015. During 2018~2019, he was a Visiting Scholar at Arizona State University, Tempe, AZ, USA. He is currently a professor in the Department of Computer Engineering, Yeungnam University. His research interests include advanced processor architecture, low-power mobile embedded systems, high performance parallel and distributed computing, and AI-based intelligent computer systems.