LLM Attention의 효율적인 연산을 위한 Vector Engine Chain과 Dataflow

장재혁[0], 신동군
성균관대학교 전자전기컴퓨터공학과
jjh4777@g.skku.edu, dongkun@skku.edu

Vector Engine Chain and Dataflow for Efficient Computation of LLM Attention

Jaehyuk Jang[0], Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

최근 몇 년 동안 트랜스포머 모델의 발전이 빠르게 이루어졌다. 특히, 트랜스포머 모델을 기반으로 자연어 처리 작업을 수행하는 LLM(Large Language Model)은 여러 분야에서 혁신을 주도하였다. 그러나 LLM의 성능 향상을 위해 모델의 크기는 계속해서 증가하고 있다. 이러한 LLM을 실행하기 위해서는 매우 높은 계산 및 메모리 요구사항이 필요하게 되었으며 빠른 응답시간을 요구하는 LLM 추론의 지연시간이 계속해서 증가하는 문제가 발생하고 있다. 이러한 문제를 해결하기 위해 본 논문에서는 기존 머신 러닝 가속기와 적합하지 않은 LLM 추론 연산 특징들을 분석하고 동시에 LLM 연산을 효율적으로 처리할 수 있는 가속기 구조를 제안하여 더 빠른 LLM 추론을 가능하게 한다. 실험 결과 본 논문에서 제안한 새로운 가속기 구조는 기준선과 비교하여 약 3배 수행 시간 감소를 보여준다.

1. 서 론

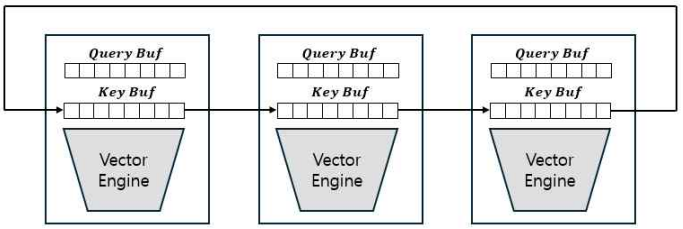
최근 GPT[1]와 같은 트랜스포머 모델 기반의 대형 언어 모델인 LLM(Large Language Model)이 광범위한 작업에서 뛰어난 성과를 보이며 많은 주목을 받고 있다. LLM은 보다 강력한 성능을 발휘하기 위해 수십억 개의 매개변수를 보유하게 되었고 그 규모는 계속해서 증가하고 있다. 그러나 이러한 거대한 모델은 실행에 매우 높은 연산량과 메모리 요구 사항을 동반한다. 따라서 제한된 리소스에서 LLM 추론을 진행할 때 추론 지연시간이 증가하는 문제가 발생하게 된다.

LLM 추론은 문장을 입력 시퀀스로 받고 트랜스포머의 디코딩 과정을 거쳐 새로운 토큰을 생성한다. 생성된 토큰은 입력 시퀀스 이후의 문장을 완성하거나 질문에 대답하는 데 사용할 수 있다. LLM 추론 과정은 다양한 텐서 연산들로 이루어져 있으며 이때 두 가지 주요 과정을 거치게 된다. 먼저 Prefill 단계에서 초기 입력을 받아 생성 작업을 한 번 수행한다. 그리고 Generation 단계에서는 이전에 캐시 된 Key와 Value를 활용하여 여러 번의 토큰 생성 작업을 수행한다. Prefill 단계는 행렬과 행렬의 곱셈인 GEMM(GEneral Matrix-Matrix multiplication) 연산이 진행되는 반면 Generation 단계는 행렬과 벡터의 곱셈을 수행하는 GEMV(GEneral Matrix-Vector multiplication) 연산이 진행되는 특징을 가지고 있다. 따라서 전반적으로 LLM 추론은 대부분 GEMM 연산과 GEMV 연산으로 이루어져 있으며 두 가지 연산 특징을 모두 수용하며 가속할 수 있는 연산장치를 설계하는 것은 현대 LLM 가속에서 필수적인 과제이다.

기존 TPU나 NPU와 같은 Systolic Array 기반의 머신 러닝 가속기들은 GEMM 연산을 처리하는 데 능숙하지만 GEMV 연산의 경우 산술 밀도가 낮기 때문에 GEMM 연산의 특화된 기존 구조에서 연산 리소스를 충분히 활용하지 못하는 특징을 가지고 있다. Systolic Array 구조가 GEMM 연산에 효율적인 이유는 여러 Processing Unit들이 연속적으로 데이터를 전달하며 병렬적으로 연산을 진행하기 때문이다. 반면에 벡터 엔진의 경우 GEMV 연산을 진행할 때 연산 리소스를 충분히 활용하며 연산을 진행할 수 있지만 GEMM 연산의 경우 메모리 액세스 패턴이 Systolic Array와 비교하였을 때 효율적이지 않다. 따라서 본 논문에서는 여러 벡터 엔진을 Systolic Array와 같이 데이터 이동을 진행할 수 있도록 상호 연결된 벡터 엔진을 제안하여 GEMM 연산과 GEMV 연산을 모두 효율적으로 처리할 수 있도록 하는 벡터 엔진을 제안한다.

또한 LLM은 Attention 연산을 진행할 때 독특한 연산 특징을 가지고 있다. LLM은 시퀀스에서 이전에 생성되었던 토큰들을 기반으로 인과적인 관계를 보며 다음 토큰을 생성하기 때문에 Prefill 단계에서는 Attention Score 행렬에 마스크를 적용한다. 마스크 되는 부분의 값은 attention 연산 중에 softmax 과정을 거쳐 추후 0의 값으로 수렴한다. 따라서 마스크가 적용된 부분의 연산을 건너뛰더라도 결과의 영향을 미치지 않으며 연산을 건너뛰게 될 때 절반가량의 연산량을 줄일 수 있다. 하지만 여러 개의 상호 연결된 벡터 엔진을 병렬로 실행하더라도 마스크 되는 부분의 연산들을 연산 리소스를 완전히 활용하며 Skip 할 수 없다. 이러한 문제를 해결하기 위해 본 논문에서는 상호 연결된 벡터 엔진을 완전히 활용하면서 마스크 되는 부분의 연산을 모두 건너뛸 수 있는 dataflow를 제안하며 그에 따른 메모리 접근 패턴을 고려한 구조 최적화를 제안한다.

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.IITP-2017-0-00914, 지능형 IoT 장치용 소프트웨어 프레임워크)



[그림 1] Chained Vector Engine 구조

2. Chained Vector Engine

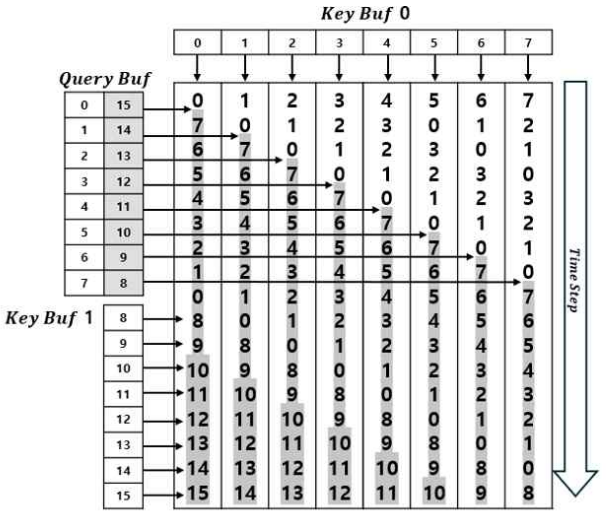
기존에 DRAM 내부에서 연산을 진행하는 Processing-In-Memory 연구 중에는 BANK마다 벡터 엔진을 두고 병렬로 실행하며 Bank 간 데이터 이동이 가능하도록 연구된 TransPIM[2]이 제안되었다. TransPIM의 경우 Encoder 기반 트랜스포머 모델을 PIM에서 처리하기 위한 연구로 각 BANK마다 벡터를 분산시킨 후 연산 중 벡터를 이동시키며 GEMM 연산이 가능하도록 하였다. 이러한 기법의 영감을 받아 가속기로서 이러한 구조를 그림 1과 같이 설계하였다. 제안된 벡터 엔진은 단순히 여러 개의 벡터 엔진을 병렬로 동작시킬 때와 비교하여 연산량은 동일하지만 더 효율적인 메모리 접근이 가능하다. 이에 대한 분석은 5장에서 진행한다.

3. LLM Mask Skip

Masked GEMM Attention에서 마스크 되는 부분을 Chained 벡터 엔진으로 Skip 하기 위해서는 시퀀스 길이를 벡터 엔진 개수의 크기로 Tiling을 진행하여 처리하고 완전히 마스크 된 Tile의 연산은 Skip 하는 방식으로 진행할 수 있다. 하지만 대각에 있는 Tile과 같이 일부분만 마스크 되어 있는 Tile의 경우는 Skip 하지 못하고 기존에 GEMM 연산과 동일하게 진행되어야 한다. 이러한 문제를 해결하기 위해 Chained 벡터 엔진에 새로운 데이터 흐름을 적용하고 구조를 최적화하여 마스크 된 부분을 완전히 Skip 할 수 있도록 설계하였다.

3-1. Dataflow

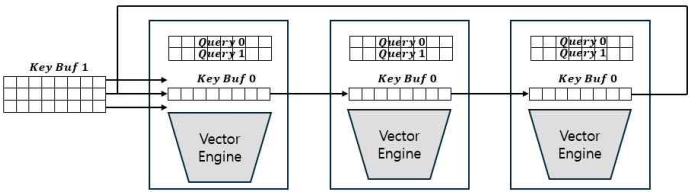
완전한 Mask Skip을 위해 제안하는 dataflow는 벡터 엔진 개수의 크기였던 Tile Size를 2배로 늘려서 연산한다. 예를 들어 엔진 개수가 8개라고 한다면 가장 처음 Query, Key 0~7번 Index를 load 하여 연산을 진행한다. 이때 Key Index가 Query Index보다 큰 경우 Masking이 되는 부분이고 이 부분을 Skip 하면서 동시에 8~15번 Index의 Query를 입력으로 읽어서 모든 벡터 엔진이 idle 상태를 거치지 않고 연산 리소스를 완전히 활용하며 Masked GEMM 연산을 진행할 수 있도록 한다. 연산 과정은 그림 2에서와 같이 초기 Query, Key(0~7) 연산을 순방향으로 진행하도록 한다. 이후 Mask Skip으로 유휴 상태가 되는 벡터 엔진으로 나머지 역방향의 회색 Query(15~8)들을 차례로 전환하여 현재 Key들이 올바른 Index의 Query들과 연산을 진행할 수 있도록 한다. 또한 전환된 Query가 현재 Key들과 모든 연산을 마치면 Key Buffer 1에서 첫 번째 벡터 엔진으로 Key를 차례로 넣어주며 남은 연산을 위해 데이터가 흘러갈 수 있도록 한다. 이러한 방식으로 진행할 경우 벡터 엔진을 완전히 활용하며 Masked 된 연산을 모두 Skip할 수 있다.



[그림 2] Mask Skip Dataflow

3-2. 구조 최적화

그림 2과 같은 Dataflow를 적용하기 위해서는 벡터 엔진의 개수의 2배 Data를 저장할 Buffer가 필요하다. Buffer가 없을 경우 역방향으로 전환되는 데이터들은 차례대로 한 Cycle에 하나씩 읽어야 하고 그렇게 될 경우 데이터를 병렬로 access 할 수 있는 기회를 잃게 된다. 따라서 각 벡터 엔진들에 그림 3과 같이 Query와 Key Buffer를 추가하여 메모리 접근의 효율을 높일 수 있도록 하였다.



[그림 3] Chained Vector Engine with Double Buffer

4. 연산 Cycle 분석

이 장에서는 Mask Skip을 진행하였을 때 Chained 벡터 엔진과 Dataflow 및 구조 최적화를 적용한 엔진 간의 연산량을 확인한다.

Chained 벡터 엔진의 연산량은 전체 시퀀스 길이를 벡터 엔진 개수만큼의 Tile Size로 나누어 처리를 진행할 수 있으며, 연산 Cycle을 수식화하기 위해 Tile Size를 T라 하고 Tile의 개수를 N으로 정의한다. 마스크 되는 부분의 연산을 Skip 하기 위해 완전히 마스크 되는 Tile의 개수를 제외한 Tile 개수는 1부터 Tile 개수의 합으로 나타내고 각 Tile에서의 연산량은 Tile Size 만큼의 Cycle을 소요하기 때문에 Chained 벡터 엔진의 연산량은 다음과 같이 나타낼 수 있다.

$$Chained\ \ Engine\ \ Total\ \ Cycle = \frac{TN^2 + TN}{2}$$

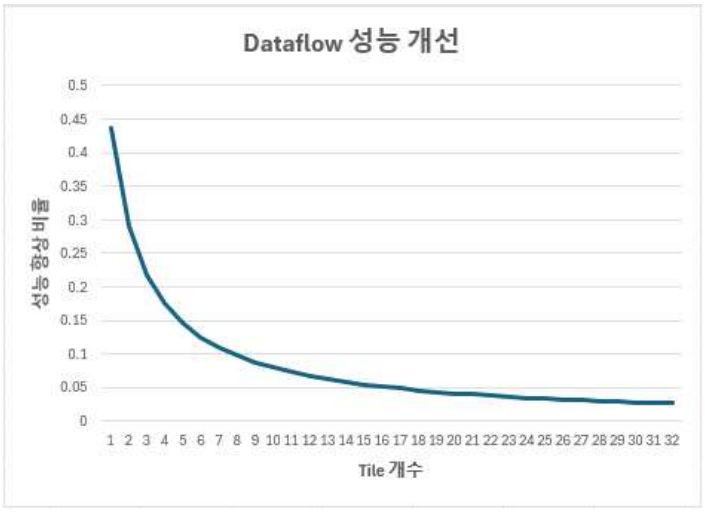
마스크 Skip을 위한 Dataflow를 적용하였을 때 Tile Size는 2배가 되기 때문에 마스크 되지 않은 Tile에서의 연산 Cycle은 4개 Tile을 처리하는 Cycle이므로 $4T$가 된다. 부분적으로 마스크 되는 Tile의 경우 그림 2를 참고해 본다면 알 수 있듯이 연산량

은 $2T+1$이다. 대각 Tile 수는 Tile Size가 2배가 되었기 때문에 $N/2$이고 나머지 Tile의 수는 대각 Tile을 제외한 1부터 $N/2-1$까지의 합이 되므로 총 연산량은 다음과 같이 나타낼 수 있다.

$$Dataflow\ \ Total\ \ Cycle = \frac{TN^2+N}{2}$$

위와 같은 Cycle 분석 수식을 기반으로 연산 Cycle 수를 비교해 본다면 벡터 엔진의 개수가 8일 때 그림 4와 같은 결과를 보여주게 된다. 그림 4의 결과는 Input으로 들어오는 Sequence Length의 길이가 늘어날수록 성능 향상이 낮아지는 추세를 보여준다. 이유는 Dataflow를 사용하더라도 대각 요소의 있는 Tile에서만 성능 향상이 이루어져 Sequence Length의 길이가 커지면 마스크 되지 않은 Tile의 개수의 비율이 높아지기 때문이다.

분석 결과를 바탕으로 Mask Skip의 성능 향상의 이점을 더욱 높이기 위해서는 벡터 엔진의 개수를 늘려 Tile 크기를 더 높이거나 대각 요소의 Tile에서만 연산을 진행하는 Sparse Pattern [3]을 적용한다면 성능이 더 향상될 수 있을 것이다.



[그림 4] Dataflow 적용에 따른 성능 향상 비율

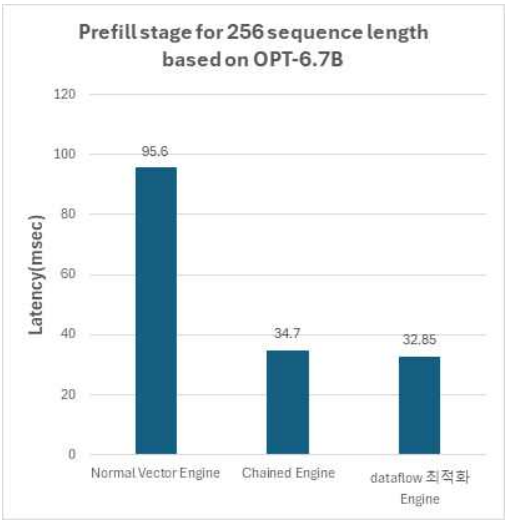
5. 메모리 접근 분석

이 장에서는 Engine 간 연결이 되어 있지 않은 일반적인 벡터 엔진과 Chained 벡터 엔진의 메모리 접근 횟수를 확인한다. 2가지 방식 모두 Query를 읽는 방식은 동일하므로 Query에 대한 접근은 제외한다.

일반적인 벡터 엔진으로 1개 Tile을 처리할 때 여러 개의 벡터 엔진에 서로 다른 Query를 고정시키고 단일 Key를 메모리로부터 읽어 모든 Engine에 복사하여 연산을 진행하게 된다. 이 과정을 벡터 엔진 개수만큼 진행하며 메모리의 다중 채널이 있어도 한 번의 접근 당 하나의 Key를 읽어오는 비효율적인 동작을 거치게 된다. 반면에 Chained 벡터 엔진은 여러 Key를 여러 채널에서 한 번에 읽어와 Engine 간의 데이터를 이동시키며 사용하기 때문에 매번 메모리 접근을 할 필요가 없으며 메모리 Channel 활용도가 올라가 더 효율적인 GEMM 연산이 가능해진다.

6. 실험 및 결론

제안된 벡터 엔진의 성능 향상을 검증하기 위해 OPT-6.7B 모델[4]을 기반으로 시퀀스 길이 256인 Prefill $Q \cdot K^T$연산을 실험하였다. 3가지 Engine을 각각 Vitis HLS tool을 활용하여 Xilinx Alveo U280 FPGA 보드 위에 합성 및 구현하여 실험을 진행하였으며 엔진은 8개로 각 Read/Write Channel 1개씩 총 16 Channel로 구현하였다.



[그림 5] 실험 결과

그림 5에서의 실험 결과는 각 구조에서의 Latency를 나타내며 Chained 벡터 엔진은 일반적인 벡터 엔진과 비교하여 약 3배의 성능 향상을 보여주었다. 이러한 결과는 이전에 분석하였던 Memory 접근에서의 성능 차이를 반영한다. 또한 dataflow 최적화로 인한 성능 향상은 약 5%이며 4장에서 진행하였던 연산량 분석에서의 결과와 어느 정도 일치하는 결과를 보여주며 추후 Sparse한 Attention Pattern 적용 가능성을 확인시켜준다.

결론적으로 기존 GEMV 연산에 특화되었던 LLM 전용 벡터 엔진[5]과 비교하여 GEMM 연산과 GEMV 연산을 모두 수용하는 Chained 벡터 엔진은 보다 다양한 연산 요구를 충족시킨다. 이는 기존의 제한된 용도를 넘어, 향후 LLM 가속 연구의 기반이 될 것이다.

참 고 문 헌

[1] ACHIAM, Josh, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[2] Zhou, Minxuan, et al. "Transpim: A memory-based acceleration via software-hardware co-design for transformer." 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2022.

[3] Child, Rewon, et al. "Generating long sequences with sparse transformers." arXiv preprint arXiv:1904.10509 (2019).

[4] Zhang, Susan, et al. "Opt: Open pre-trained transformer language models." arXiv preprint arXiv:2205.01068 (2022).

[5] Hong, Seongmin, et al. "DFX: A low-latency multi-FPGA appliance for accelerating transformer-based text generation." 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2022.