대규모 언어모델 추론의 성능 향상을 위한 GPU/FPGA 기반 하이브리드 플랫폼 개발

연세대학교 대학원

전기전자공학과

박 현 준

대규모 언어모델 추론의 성능 향상을 위한 GPU/FPGA 기반 하이브리드 플랫폼 개발

지도교수 김 한 준

이 논문을 석사 학위논문으로 제출함

2023년 12월 26일

연세대학교 대학원

전기전자공학과

박 현 준

박현준의 석사 학위논문을 인준함

심사위원 김 한 준 인

심사위원 노 원 우 인

심사위원 송 진 호 인

연세대학교 대학원

2023년 12월 26일

감 사 의 글

인생을 살면서 수없이 많은 사람을 만나지만, 본인의 삶에 긍정적인 방향으로 전폭적인 지지와 힘을 보태주는 의인은 평생에 있어서 한 사람을 만나기도 쉽지 않다고 생각합니다. 누군가 저에게 이러한 의인이 있냐고 물으신다면 저는 감사하게도 두 분을 만났다고 말씀드릴 것 같습니다. 초등학교 때부터 재수 시절까지 과외를 해주셨던 김양우 선생님, 과외라는 매개로 만나 공부를 넘어 인생에 대한 큰 가르침을 주셨고 지금은 비록 하늘에 계시지만 제 마음속에 항상 남아있습니다. 학부 4학년부터 석사과정까지 지도해주셨던 김한준 교수님, 교수님으로서 제가 잘 성장할 수 있도록 논문을 읽는 방법, 연구 주제를 정하는 방법, 연구를 구현하는 방법 등의 가르침을 주셨습니다. 멘토로서 저의 의견을 존중해주심과 동시에 아낌없는 조언을 해주셨고, 방황할 때도 항상 같은 곳에서 묵묵히 손을 뻗어주셨습니다. 그리고 연구실 내외에서 많은 대화를 나누며 인생의 선배로서 세상을 바라보는 방식, 생각하는 방식, 사람을 대하는 방식 등의 깊은 깨달음도 주셨습니다. 두 분께 받은 무한한 사랑에 이 자리를 빌려 가장 먼저 감사드리고 싶습니다.

학위논문의 심사위원으로 자리해주신 노원우 교수님과 송진호 교수님께도 감사드립니다. 학부 시절 들은 수많은 수업 중 노원우 교수님의 수업을 듣고 저의 진로를 결정하였으며, 덕분에 지금도 재미있게 공부하고 있습니다. 군입대 전에 해주신 개인 면담 때의 가르침도 마음 깊이 새겨두고 있습니다. 대학원 진학 후 들었던 송진호 교수님의 수업 역시 저에게 소중한 자산이 되어 남아있습니다.

연구실 생활을 함께하며 많은 도움과 행복을 준 승빈이 형, 용우, 신넝, 재호 형, 동관이 형, 근우, 희림, 선영, 성우, 주민에게 감사드립니다. 특히 항상 곁에 있으며 큰 힘이 되어준 희림, 선영, 성우, 주민에게 다시 한번 깊은 감사를 드립니다.

부를 때마다 한 걸음에 나와주고 함께 이야기 나눠주는 군희, 영준, 정민, 대균, 대학교 1학년 때부터 모르는 게 없던 공부 도라에몽 상진이에게도 감사드립니다.

마지막으로 항상 열정적인 지지와 무조건적인 사랑을 보내주시는 어머니 아버지, 하늘에 계시는 할아버지와 할머니께 감사드립니다. 잘 키워주신대로 앞으로도 잘 살아갈게요. 어머니 아버지 사랑합니다.

차　　　례

그 림 차 례

표 차 례

국 문 요 약

대규모 언어모델 추론의 성능 향상을 위한 GPU/FPGA 기반 하이브리드 플랫폼 개발

본 논문은 Transformer 기반 대규모 언어모델(Large Language Model, LLM) [1] 연산의 특성과 GPU와 FPGA의 특성을 바탕으로 LLM의 추론을 최적화한 하이브리드 플랫폼을 제안한다. 대규모 언어모델은 입력 토큰을 받아 핵심 정보를 추출하는 요약 스테이지(Summarization Stage)와 핵심 정보를 받아 출력 토큰을 연산하는 출력 스테이지(Generation Stage)로 나눌 수 있다. 요약 스테이지의 경우 입력 토큰에 대해 병렬적으로 연산할 수 있다는 특성이 있고, 생성 스테이지의 경우 하나의 출력 토큰을 연산할 때 이전 출력 토큰을 필요로 하기에 순차적으로 연산해야 한다는 특성이 있다.

현재 생성형 AI을 위한 반도체로 대부분 NVIDIA의 GPU가 사용되고 있고, 기존 인공지능 모델과 같이 대부분의 지표에서 높은 성능을 보이고 있다. 하지만 높은 병렬성으로 연산량이 병목이 되는 요약 스테이지는 병렬 연산을 효율적으로 수행할 수 있는 GPU가 잘 연산할 수 있는 것에 반해, 높은 메모리 사용량으로 메모리가 병목이 되는 생성 스테이지는 GPU의 연산기를 아직 충분히 사용하지 못하고 있는 문제가 있다.

이러한 부분을 개선하기 위해 하드웨어 레벨에서 최적화하고자 하는 FPGA, ASIC 연구도 많이 이루어지고 있다. ASIC 연구의 예로는 Google이 개발한 TPU [2] 가 있고, FPGA 연구의 예로는 LLM 연산에 최적화된 아키텍처를 설계한 DFX [6] 가 있다. 하지만 디바이스들은 생성 스테이지를 수행하는 많은 경우에 대해 GPU보다 앞선 성능을 보이지만, GPU보다 적은 연산기 개수를 가지고 있어 병렬화가 잘 이루어지는 요약 스테이지에서는 앞서지 못하고 있다.

본 논문의 아이디어는 여기서 시작한다. GPU와 FPGA가 모두 포함된 하이브리드 환경에서 각 디바이스에 각각이 잘 수행할 수 있는 연산을 할당한다면 기존의 방식보다 더 빠르게 연산할 수 있을 것이라고 생각하여 연구를 시작하였다.

해당 플랫폼을 구축하기 위해 크게 5단계로 연구를 진행하였다. 첫째, 하드웨어로는 NVIDIA의 A10과 XILINX의 U55C를 사용하였고, 각각의 디바이스를 컨트롤하기 위하여 FasterTransformer [7] 와 DFX [6] 코드를 선택하였다. 둘째, LLM을 GPU에서 실행할 연산과 FPGA에서 실행할 연산으로 분할하였고, 이종 하드웨어로 작동하기 위하여 어떤 정보를 넘겨주어야 하는지 분석하였다. 셋째, 재배열(reshape) 커널과 통신(communication) 커널을 구현하여 GPU에서 요약 스테이지를 실행하며 생성된 중간 데이터를 FPGA로 전송하였다. 넷째, Latency Hiding 기법을 이용하여 통신 오버헤드를 획기적으로 감소시켰다. 마지막으로 확장성(Scalability)를 고려한 API화를 진행하여 하이브리드 플랫폼(Hybrid Platform)을 구현하였다.

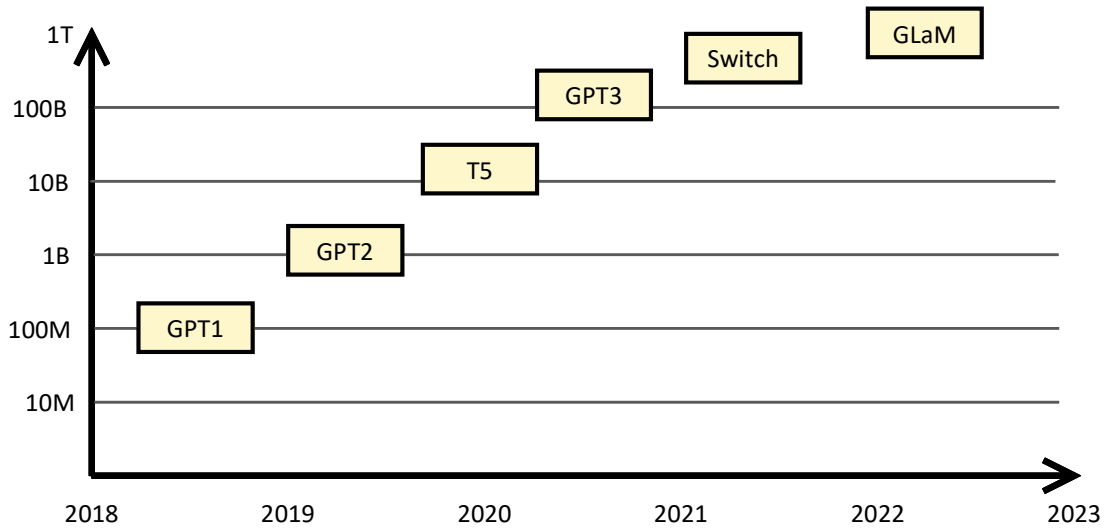
해당 플랫폼의 성능을 검증하기 위해 여러 크기의 입력 토큰과 출력 토큰에 대해 실험을 진행하여 분석했다. 단일 디바이스로 수행한 연산을 비교군으로 설정하여 실험한 결과, 256 이상의 입력 토큰과 출력 토큰에 대해 하이브리드 플랫폼이 최대 1.56배 빠르게 연산을 수행하였다. 선행연구로는 Hype-training [21] 와 FARNN [22] 가 CNN [8]과 Transformer [1] 계열 모델의 훈련(Training)을, Walther [24] 와 FleetRec [25] 이 CNN과 추천 시스템의 추론(Inference)를 이종 하드웨어를 사용하여 최적화하였다. 그러나 LLM 추론을 가속하기 위하여 이종 하드웨어를 사용한 선행 연구는 찾아볼 수 없었다는 점에서도 본 연구의 의의가 있다.

또한 확장성을 고려하여 구현하였기에 미시적으로는 새로운 모델과 옵션에 대해 능동적으로 대처할 수 있을 뿐만 아니라, 거시적으로 다중 하드웨어와 다중 서버로 확장하여 연구를 진행할 수 있다. 추후 데이터 센터 규모의 클러스터 플랫폼으로 확장하여 연구를 진행한다면 더 큰 가치를 창출할 수 있을 것으로 기대한다.

핵심 단어 : 대규모 언어모델, LLM 추론, 하이브리드 플랫폼, 이종 하드웨어

제 1장 서론

대규모 언어모델(LLM)의 개념은 1966년에 발표된 Eliza [3] 에서 처음 등장했으나, 본격적인 시작은 2017년에 발표된 Google의 Transformer [1] 이후라고 볼 수 있다. LLM이란 인공 신경망을 이용하여 자연스러운 언어 문장이 존재할 확률 분포를 연산하는 모델이다. LLM의 개념은 이를 기반으로 2018년 OpenAI의 GPT [4]가 발표되었는데, 2022년 초 GPT 기반 어플리케이션인 ChatGPT가 흥행함에 따라 LLM의 가치가 전세계적으로 인정받게 되었다. ChatGPT의 흥행은 LLM 시장의 기폭제가 되었고, 현재 전 세계 수많은 엔지니어들이 LLM 관련 연구 [16, 17, 18, 19, 20, 21, 22, 23, 24, 25] 를 활발히 진행하고 있다.

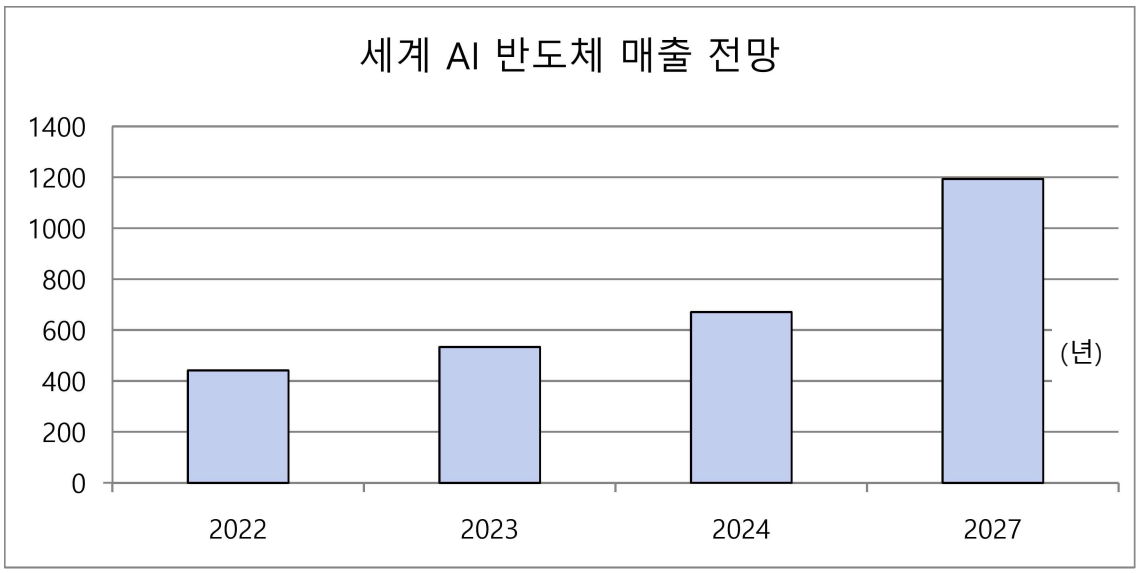


[그림 1] LLM의 성장 [9]

[그림 1]은 GPT의 발표 이후 출시되고 있는 LLM의 크기를 나타낸 그래프이다. 모델의 파라미터 수와 모델의 정확도가 비례하는 경향이 있기에 파라미터 수를 키우는 방향의 연구가 진행되고 있다. 2018년에 발표된 GPT1의 경우 파라미터 수가 1억 1700만 개였지만 [9], 4년도 채 지나지 않아 GLaM의 파라미터 수는 1조 2000

억 개에 달하였고 [9], 지금도 더 높은 정확도를 위해 더 큰 모델들이 꾸준히 연구되고 있다.

점점 더 높은 정확도의 LLM이 등장함에 따라 여러 도메인에서 활용하고자 하는 시도가 늘어나고 있다. 이전에는 질문에 대한 대답이나 단순 번역 등의 일을 위주로 수행했다면, 이제는 논문을 요약하거나 소설을 작성하는 등 긴 입출력문장을 다루고자 하는 수요도 증가하고 있다. 특히, 현재 챗봇 형식으로 지원되는 대부분의 LLM은 유저와의 대화를 수행할 때 문맥을 가지고 답변하기 위해 이전에 했던 질문과 답변을 모두 쌓아 한 번에 입력 문장으로 넣는 방식으로 서비스를 제공하고 있기 때문에 이를 지원하기 위해 현재 수천 개를 넘어 수만 개의 입력 토큰을 빠르게 연산하는 연구도 진행되고 있다.



[그림 2] 세계 AI 반도체 매출 전망 [10]

이처럼 LLM 자체의 크기뿐만 아니라 요구되는 입출력 문장의 길이도 길어짐에 따라 LLM을 효율적으로 연산할 수 있는 가속기에 대한 필요성이 증가하고 있다. 그 결과 AI 반도체 시장의 규모가 기하급수적으로 커지고 있으며, 관련 전문가들은 앞으로 AI 반도체 시장이 더 커질 것으로 예상하고 있다. 글로벌 연구 조사 기업 가트너(Gartner)에 따르면, 매년 AI 반도체 시장은 급속도로 증가하고 있으며, 생성

AI의 발전에 따른 고성능 반도체의 수요가 이를 가속해 2027년에는 1194억 달러에 이를 것이라고 전망하고 있다. 현재 전 세계적으로 AI를 가동하기 위한 하드웨어로 대부분 NVIDIA의 GPU를 사용하고 있다. [10] 이러한 NVIDIA의 독주를 막기 위해 수많은 대기업 및 스타트업이 AI 반도체 개발에 주력을 다하고 있지만, 시장조사기관 TOP500에 따르면 2022년 기준 슈퍼컴퓨터 가속기 점유율은 NVIDIA가 92%로 사실상 독주가 지속되고 있다. [10]

하지만, LLM 추론을 실행하는 데 있어서 GPU는 크게 두 가지 문제점을 가지고 있다. 첫째, 매우 빠른 속도로 커지고 있는 모델 사이즈에 비해, 각 디바이스에 내장된 메모리는 크기 및 메모리 대역폭이 그 속도를 따라가지 못해 어려움을 겪고 있다. [그림 1]에서 언급했듯 LLM의 크기는 기하급수적으로 커지고 있고, 이때 디바이스에 사용하고자 하는 모델이 모두 올라가야 이를 실행할 수 있기 때문에 해당 문제는 반드시 해결되어야 하는 문제이다. 이를 해결하기 위하여 다중 디바이스로 확장하는 방향으로 주로 연구가 진행되고 있다. 다중 GPU로 병렬처리를 진행하기 위하여 NVLink 등 다양한 라이브러리와 하드웨어를 출시하여 GPU 간 데이터 동기화를 최적화하고 있다. 이 과정에서 발생하는 통신 오버헤드를 많이 최적화하고 있지만, 현재 기준으로 아직까지는 이 오버헤드가 전체 성능에 적지 않은 영향을 주고 있다. 또한 LLM은 높은 메모리 대역폭을 필요로 하는데, GPU와 CPU의 통신을 수행하는 PCIe의 대역폭이 이를 따라가지 못해 메모리 병목도 발생하고 있다.

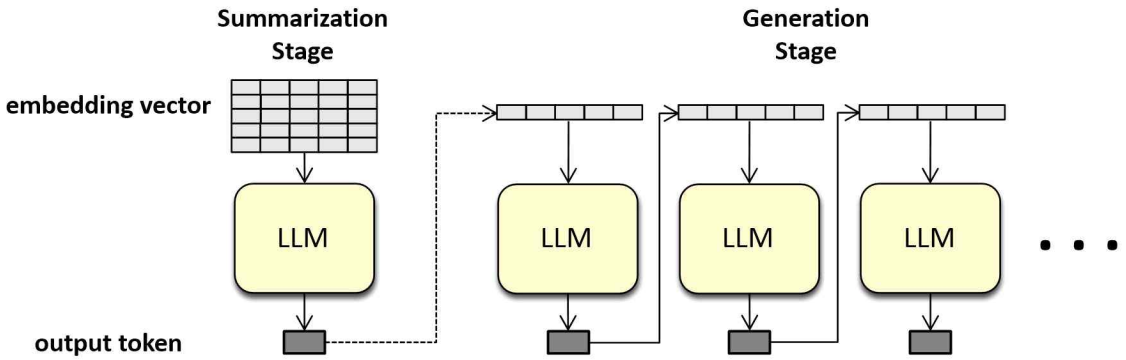
둘째로, LLM의 기본 신경망 구조인 Transformer는 그 연산 구조가 복잡하여 병렬화가 쉽지 않다. 인공지능은 다양한 신경망 구조를 개발하며 발전해왔고, 대표적인 예시인 CNN(Convolution Neural Networks) [8] 의 경우 단순 행렬곱과 유사한 형태로 GPU가 가장 잘 수행할 수 있는 연산이다. 하지만 LLM의 경우 대부분 Transformer 신경망 구조를 기반으로 구현되어 있으며, Transformer 연산의 많은 부분을 차지하는 어텐션(Attention) 계열의 연산은 그 구조가 복잡하여 GPU의 연산기를 충분히 사용하기 쉽지 않다. 이를 해결하기 위하여 FasterTransformer [7] 등 특정 연산에 최적화된 라이브러리를 통해 지속적인 성능 개선을 시도하고 있다.

이러한 GPU의 한계를 극복하기 위한 새로운 디바이스를 개발하는 연구도 활발히 진행되고 있다. 그중에서는 FPGA와 ASIC이 가장 조명받고 있으며, 여러 회사에서 아키텍처를 설계하고 있다. Google은 새로운 디바이스인 TPU을 사용해 자체 개발한 모델인 PaLM을 효율적으로 학습하여 사용하고 있다.[10] 국내에서도 새로운 AI 반도체를 개발하기 위해 퓨리오사, 리벨리온, 사피온 등 국내에서도 많은 스타트업이 연구개발을 진행하고 있다. 특히 리벨리온은 인공지능 벤치마크 중 가장 공신력 있다고 평가받고 있는 MLPerf의 특정 부분에서 NVIDIA GPU를 앞서는 등 유의미한 성과도 이뤄내고 있다.

본 논문에서는 LLM 추론의 가속을 목적으로 GPU와 FPGA가 LLM 추론을 어떻게 수행할 수 있는지 주목하였다. 분석 결과 GPU는 입력 문장이 긴 경우에 연산을 빠르게 수행할 수 있고 FPGA는 출력 문장이 긴 경우에 연산을 빠르게 수행할 수 있음을 파악했으며, 이를 바탕으로 이종 디바이스를 활용하여 LLM 추론을 가속하는 하이브리드 플랫폼을 제안한다. 2장에서는 LLM의 구조 및 LLM을 GPU와 FPGA로 실행했을 때의 현상을 알아보았고, 3장에서는 이를 기반으로 하이브리드 플랫폼의 구조 및 구현 과정에 대해 설명하였다. 4장에서는 실험 설계 및 결과를 분석하였다. 5장에서는 관련 선행 연구에 대해, 6장에서는 결론과 향후 연구 방향에 대해 언급하였다.

제 2장 연구 동기

2.1 LLM 연산구조



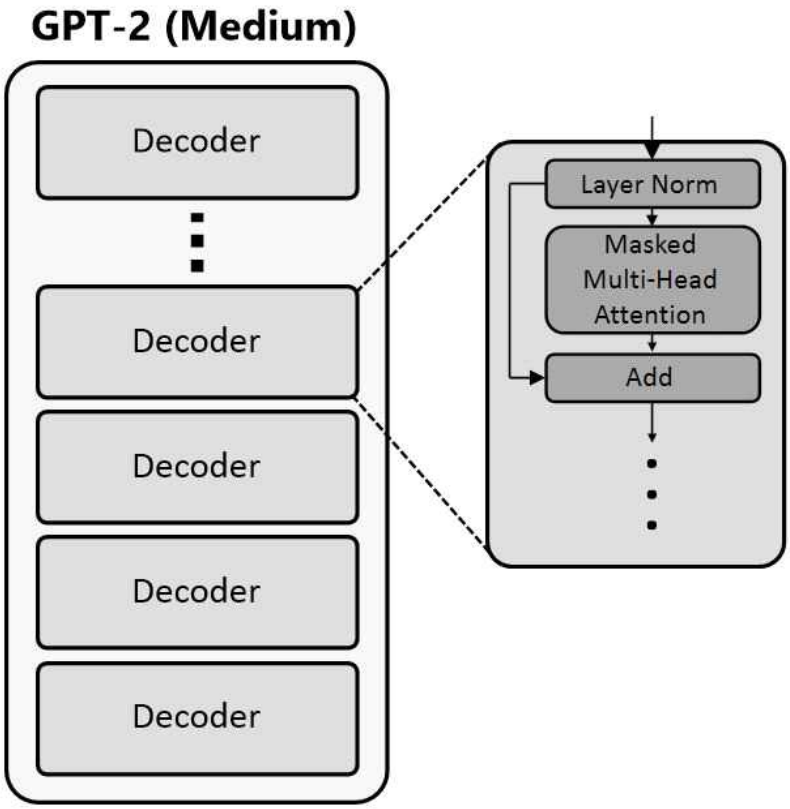
[그림 3] LLM의 연산 구조

[그림 3]는 LLM의 연산 구조를 시각화한 그림이다. LLM을 연산하기 전, 유저의 입력 문장은 전처리 과정을 거친다. 유저가 문장을 입력하면, 입력 문장은 토크나이저(Tokenizer)를 통해 토큰 리스트로 변환된다. 각 토큰들은 임베딩 벡터로 변환된 후 합쳐져(Concat) Matrix 형태로 LLM에서 연산된다.

LLM 모델은 크게 유저의 입력 문장을 연산하여 핵심 정보를 추출하는 요약 스테이지(Summarization Stage)와 핵심 정보를 바탕으로 새로운 텍스트를 출력하는 생성 스테이지(Generation Stage)로 나뉜다. 전처리 과정을 거친 데이터는 요약 스테이지 연산을 거쳐 문맥에 대한 핵심 정보를 담고 있는 중간 데이터와 하나의 출력 토큰을 생성한다. 즉, 요약 스테이지의 경우 여러 개의 입력 토큰을 Matrix 형태로 묶어 한 번에 연산하기 때문에 병렬적으로 연산을 진행할 수 있다. 생성 스테이지는 요약 스테이지에서 생성된 정보를 기반으로 출력 토큰을 연산하게 되는데, 하나의 출력 토큰을 연산하기 위해서는 바로 전에 출력된 토큰이 필요하기 때문에 순

차적으로 연산을 진행해야 한다. 생성 스테이지가 완료되고 나면 출력 토큰들은 인코딩(Encoding) 과정을 거쳐 유저가 읽을 수 있는 문장으로 변환된 후 출력된다.

일반적으로 DNN은 하나의 태스크를 수행할 때 DNN 모델이 한 차례 실행한다. 그러나 LLM은 한 차례 실행될 때마다 하나의 출력 토큰을 생성한다. 따라서 언어 모델은 하나의 태스크를 수행할 때 LLM이 출력 토큰의 개수만큼 반복 실행된다. 예를 들어 출력 토큰을 100으로 설정한 태스크가 주어지면, LLM을 총 100번 실행하여 출력 토큰 100개를 생성한다.



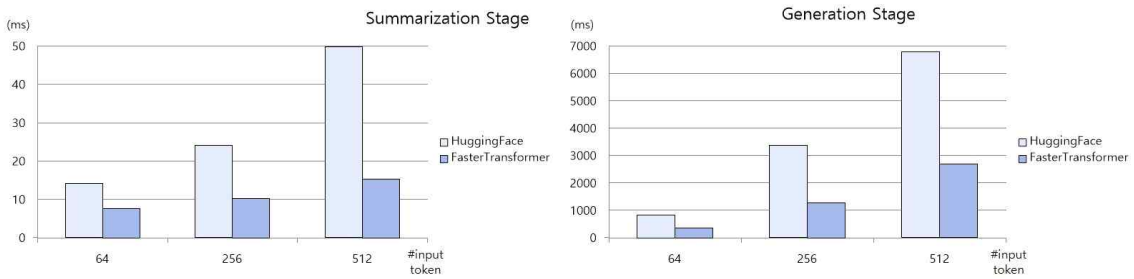
[그림 4] GPT2의 구조와 디코더를 구성하는 레이어

LLM의 연산에 대해 조금 더 자세히 알아보면, 모델의 종류마다 다르지만 LLM은 일반적으로 디코더(Decoder)가 여러 개 쌓여있는 형태로 구현되어 있다. LLM에는 다양한 레이어가 존재하지만, 일반적으로 LLM 내부에 있는 디코더는 [그림 4] Layer Norm, Attention Layer, Add, Feed Forward Network(FFN) 로 이루어져 있다. 이 중 비교적 메모리 접근이 단순하고 병렬성이 높은 DNN의 FFN Layer와는 달리, Layer Norm과 Attention Layer의 경우 메모리 접근 패턴이 복잡하여 병렬화하기 쉽지 않다. 따라서 GPU로 이 연산을 효율적으로 수행하기 위해 관련 연구 [16, 17, 18] 가 활발히 진행되고 있다.

2.2 GPU 기반 LLM 연산

LLM 연산을 지원하는 많은 프레임워크가 있지만, 가장 널리 쓰이는 프레임워크로는 크게 HuggingFace [5] 와 FasterTransformer [7] 가 있다. HuggingFace는 LLM 추론을 수행할 수 있는 가장 널리 알려진 프레임워크(Framework) 중 하나로, Python을 기반으로 구현되어 있기에 높은 접근성과 호환성을 가지고 있다. 최근에는 이러한 장점을 기반으로 LLM의 여러 지표에 관련된 밴치마크들을 지원하고 있고, 세계적인 연구가 이 밴치마크를 기준으로 진행되고 있기에 공신력이 있다고 평가받고 있기도 하다. 그러나 Python 기반이라는 한계로 인해 단순한 속도 측면에서는 다소 느린 모습을 보여주고 있다.

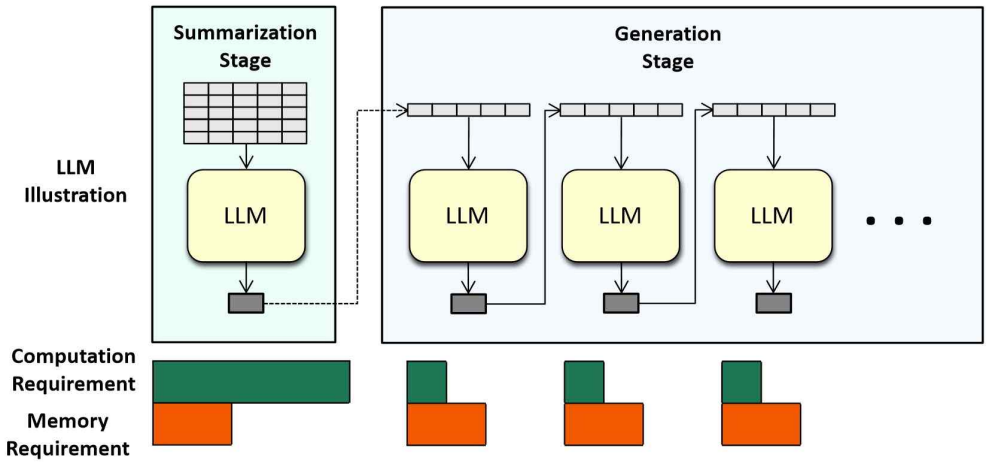
FasterTransformer는 NVIDIA에서 LLM 추론을 가속하기 위해 만든 C++/CUDA 기반 library이다. FasterTransformer는 기존에 있는 라이브러리인 cuBLAs, cuBLASLt, cuDNN도 사용하고, 양자화(Quantization), 다중 디바이스 지원 Kernel Fuse 기법 등을 사용하여 LLM 추론을 가속하였다.



[그림 5] HuggingFace와 FasterTransformer의 성능 그래프

[그림 5]은 Python 기반 HuggingFace와 C++/CUDA 기반 FasterTransformer의 성능을 비교한 표이다. 좌측 그래프의 x축은 입력 토큰의 개수, 우측 그래프의 x축은 출력 토큰의 개수이며, 두 그래프의 y축은 모두 수행 시간(ms)이다. FasterTransformer는 HuggingFace와 비교했을 때 요약 스테이지와 생성 스테이지 모두에서 1.85배~3.25배의 속도 개선을 이루어내었음을 확인할 수 있다.

2.1에서 언급했듯 요약 스테이지는 병렬적으로 연산을 수행할 수 있기 때문에 일반적으로 병목의 원인이 연산량에 해당한다. 따라서 FasterTransformer는 요약 스테이지를 효율적으로 가속하였다. 물론 FasterTransformer는 [그림 5]에서 알 수 있듯 생성 스테이지도 효율적으로 가속하였다.



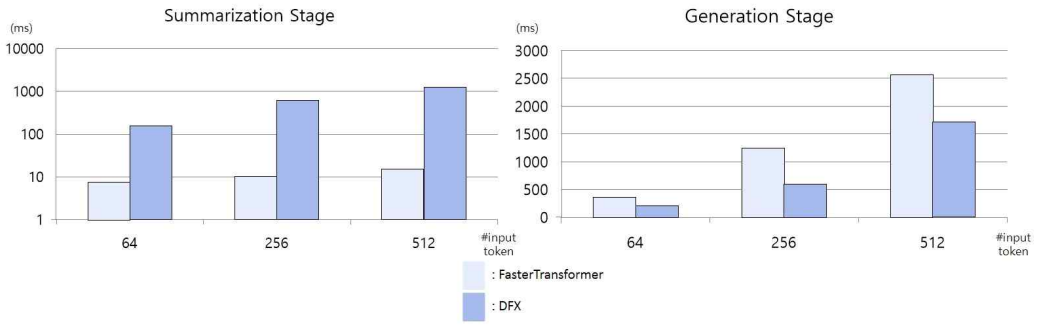
[그림 6] LLM의 스테이지 별 병목의 원인

그러나 절대적인 값으로 봤을 때 [그림 5]에서 요약 스테이지의 경우 입력 토큰이 늘어나도 10ms대를 유지했지만, 생성 스테이지의 경우 출력 토큰이 512인 경우 3000ms에 달하는 수행시간을 보여주었다.

2.1에서 언급했듯 생성 스테이지의 경우 출력 토큰의 개수만큼 LLM을 순차적으로 실행해야 하고, 실행할 때마다 매번 LLM의 웨이트가 모두 로드되어야 하므로 [그림 6] 많은 양의 메모리 접근이 필요하다. 뿐만 아니라 특정 레이어의 경우 메모리 패턴까지 복잡하기에 메모리가 병목이 되고 있다. 따라서 더 좋은 성능을 위해서는 메모리 대역폭 활용의 극대화 및 메모리 할당에 관한 최적화를 진행해야 한다. GPU의 경우 충분히 많은 연산기를 보유하고 있다는 장점이 있지만, 각 연산기가 단순하다는 단점을 가지고 있다. 즉, 순차적인 특성을 가지고 있으면서, FFN Layer 등 비교적 단순한 레이어로 구성된 DNN과 달리 Multi-Head Attention Layer 등 병렬화하기 쉽지 않은 레이어를 가지고 있기에 GPU로 가속하는 데 있어서 Challenge가 있고, 이를 개선하기 위한 연구개발이 꾸준히 이루어지고 있다.

2.3 FPGA 기반 LLM 연산

2.2에서 언급한 메모리 병목현상을 개선하기 위해서 DFX [6] 는 HBM이 내장된 FPGA를 사용하여 가속하고자 하였다. DFX는 HBM의 모든 채널을 효율적으로 사용할 수 있는 VPU(Vector Processing Unit)과 MPU(Matrix Processing Unit)을 설계하여 메모리 대역폭을 최대로 사용할 수 있는 아키텍처를 제안했다.



[그림 7] FasterTransformer와 DFX의 성능 그래프

[그림 7]는 FasterTransformer와 DFX의 요약 스테이지 및 생성 스테이지에서 성능을 비교한 표이다. 우측 그래프는 생성 스테이지 측정 결과이다. 좌측 그래프의 x축은 입력 토큰의 개수, 우측 그래프의 x축은 출력 토큰의 개수이며, 두 그래프의 y축은 모두 수행 시간(ms)이다. DFX는 메모리 대역폭 사용량을 극대화하는 방법을 이용하여 메모리가 병목이 되는 생성 스테이지를 최적화하였고, 실험 결과 FPGA로 연산한 DFX가 GPU로 연산한 FasterTransformer보다 더 좋은 성능을 보여주었음을 확인할 수 있다. 하지만 FPGA는 절대적인 연산기의 개수가 부족하기에 연산량이 병목이 되는 요약 스테이지에서는 GPU가 더 좋은 성능을 보여주었다.

즉, GPU는 많은 연산기를 기반으로 요약 스테이지를 더 빠르게 연산할 수 있고, FPGA는 HBM과 LLM에 최적화하여 설계된 Processing Unit들을 기반으로 생성 스테이지를 더 빠르게 연산할 수 있다. 즉, FPGA와 GPU의 상호보완적인 특정을 기반으로 하이브리드 플랫폼을 구성하면 단일 디바이스보다 빠르게 연산할 수 있을 것이라고 예상하여 연구를 진행하였다.

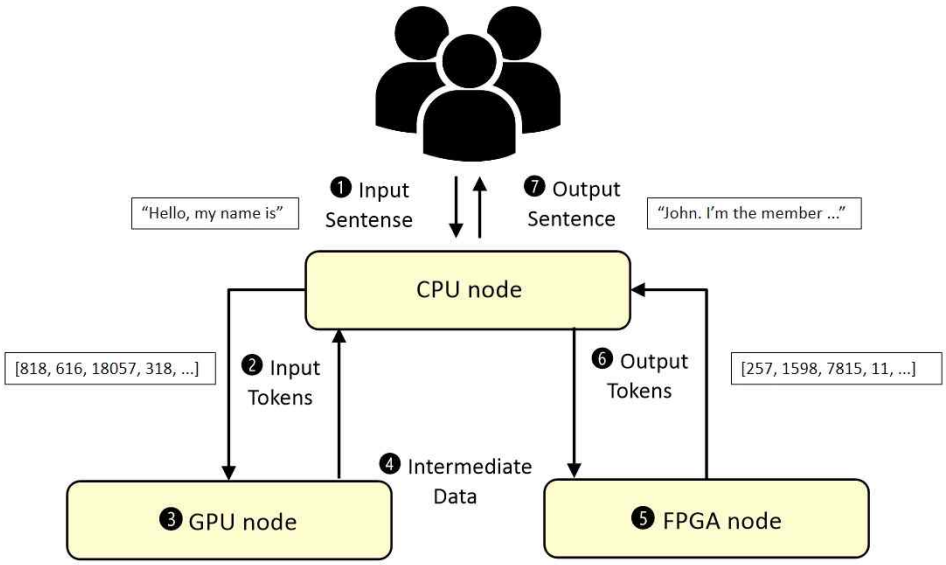
제 3장 디자인 및 구현

3.1 하이브리드 플랫폼

본 논문은 FPGA와 GPU가 LLM의 서로 다른 스테이지에서 좋은 성능을 보여주고 있는 점을 이용하여, 두 디바이스를 모두 이용한 하이브리드 플랫폼을 제안한다. 하이브리드 플랫폼은 크게 GPU 파트, p2p 파트, FPGA 파트로 나눌 수 있고, 특히 GPU 파트와 FPGA 파트는 LLM 추론을 가장 빠르게 수행할 수 있는 선행 연구를 기반으로 구현하였다. GPU 기반 선행 연구로는 NVIDIA가 자체 개발한 LLM에 특화된 최적화 라이브러리인 FasterTransformer [6] 를 기반으로 구현하였다.

FPGA 기반 선행 연구로는 Brainwave [19], FARNN [20], DFX [6]가 있다. Brainwave는 메모리 대역폭보다는 연산 처리량에 집중하였기에 CNN 계열은 잘 처리할 수 있지만 LLM에는 적합하지 않았다. FTRAN은 모델 수행 시간보다 모델 경량화에 초점을 맞춘 연구이다. 마지막으로 DFX는 LLM 추론의 속도 개선을 초점으로 아키텍처를 구현한 논문이기에 FPGA를 가동하기 위한 선행연구로 DFX를 선택하였다.

마지막으로 p2p 파트는 GPU 메모리에 있는 데이터를 FPGA에서 사용할 수 있도록 재배열한 후 전송하는 파트로, CUDA API와 OpenCL API를 사용하여 host를 경유하여 이동하도록 구현하였다. 구현한 커널을 GPU 코드 내에 삽입하는 방식으로 구성하였다.

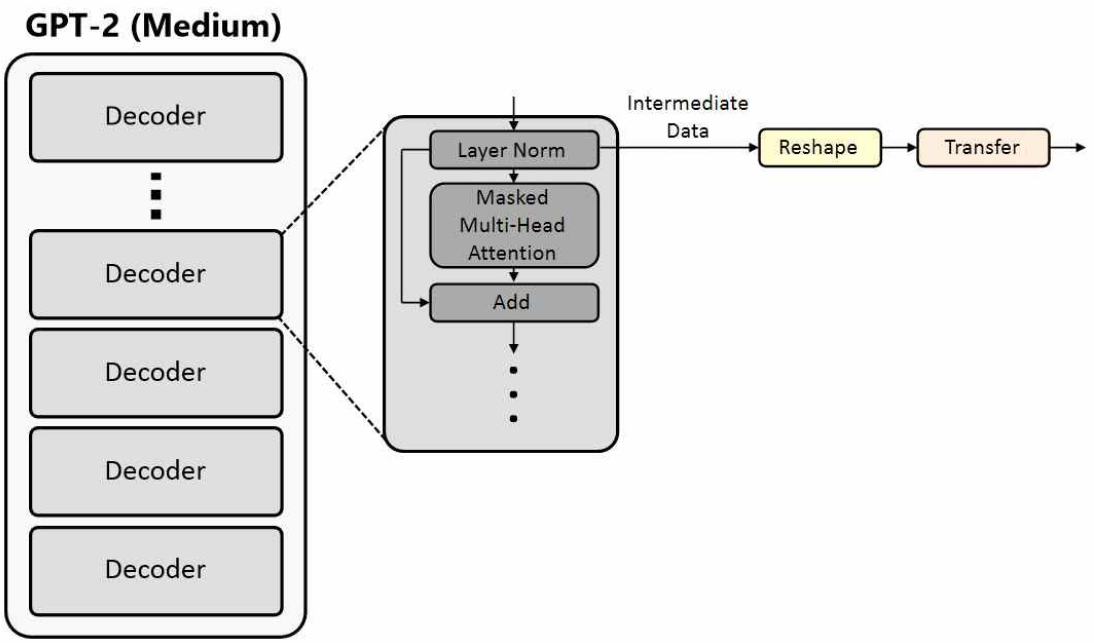


[그림 8] 하이브리드 플랫폼 구조

[그림 8]는 하이브리드 플랫폼을 도식화한 그림이다. ❶ 유저가 사람이 읽을 수 있는 문장을 입력하면 ❷ 전처리 과정을 지나며 단어들로 쪼개지고, 각 단어는 컴퓨터가 이해할 수 있는 자연수인 토큰 형태로 변환된다. ❸ GPU는 입력 토큰을 받아 요약 스테이지를 연산하여 핵심 정보를 담은 중간 데이터(Intermediate Data)를 생성한다. ❹ 중간 데이터는 p2p 통신을 이용하여 FPGA에 내장된 HBM으로 전송된다. ❺ FPGA는 전송된 중간 데이터를 받아 생성 스테이지를 연산하여 ❻ 출력 토큰을 생성한다. ❼ 출력 토큰은 후처리 과정을 거쳐 유저가 읽을 수 있는 출력 문장으로 변환되어 출력된다.

3.2에서는 LLM 중 하나인 GPT2 모델의 구조를 기반으로 중간 데이터에 대해 설명하였고, 3.3과 3.4에서는 통신을 수행하기 위해 구현한 재배열 커널과 통신 커널에 대해 설명하였다. 3.5에서는 구현 과정에서 적용한 최적화 기법인 Latency Hiding에 대해, 마지막으로 3.6에서는 GPU 코드와 FPGA 코드를 합치는 과정에 대해 설명하였다.

3.2 중간 데이터의 이해

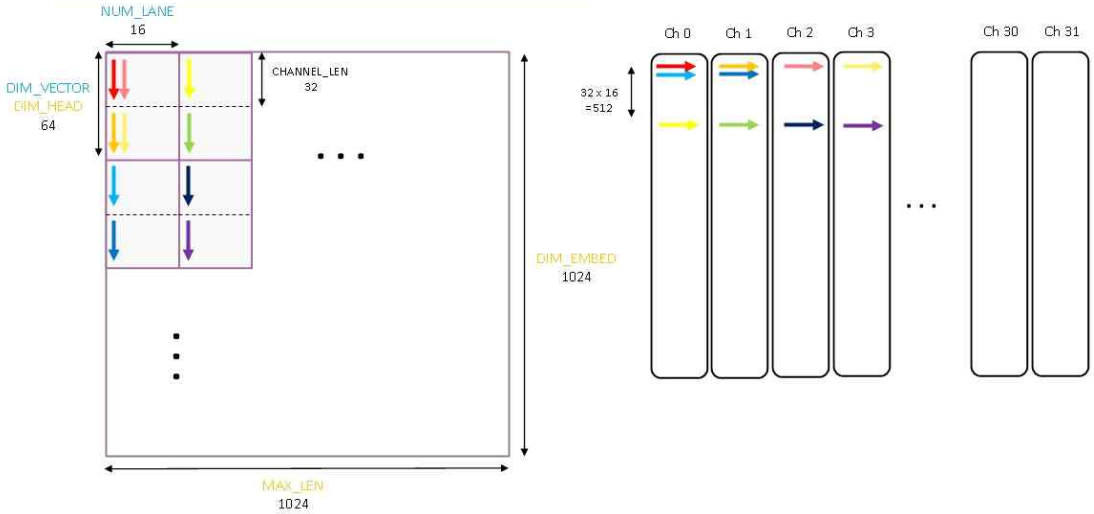


[그림 9] GPT2의 구조 및 중간 데이터

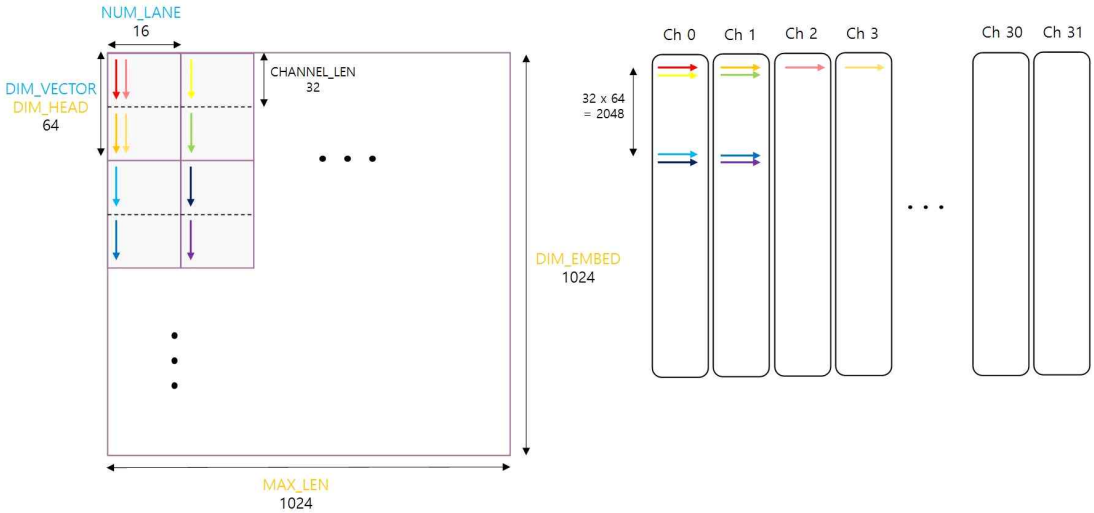
LLM마다 개수의 차이는 있지만, 기본적으로 LLM은 여러 개의 디코더(Decoder)로 이루어져 있다. 이해를 돕기 위해 gpt2-medium [4] 모델을 예시로 들면[그림 9], GPT2의 경우 총 24개의 디코더로 이루어져 있다. 유저에 의해 입력 토큰과 출력 토큰의 길이가 입력되고 나면, LLM이 출력 토큰 개수만큼 순차적으로 연산된다. 이때, 현재 연산되고 있는 LLM은 직전에 연산된 LLM의 Multi-Head Attention Layer 연산 결과를 입력으로 받아 연산하게 된다. 따라서 요약 스테이지와 생성 스테이지를 각각 다른 디바이스에서 연산하기 위해서는 이 부분을 전달해야 한다. 또한 생성 스테이지에서 n번째 출력 토큰을 연산할 때 n-1번째 출력 토큰을 입력값으로 받기 때문에 요약 스테이지에서 출력된 첫 번째 출력 토큰도 전달해야 한다. 본 논문에서는 Multi-Head Attention Layer의 결과물을 중간 데이터(Intermediate Data)라고 정의하였고, 요약 스테이지에서 중간 데이터와 첫 번째 출력 토큰을 생성 스테이지로 전달하였다.

3.3 재배열 커널

FasterTransformer의 요약 스테이지에서 생성되는 중간 데이터와 DFX의 생성 스테이지에서 사용되는 중간 데이터의 값은 일치한다. 그러나 그 모양은 각 프로젝트에 최적화된 형태로 저장이 되어있었기에 약간의 차이가 있었다. 이를 구현하기 위하여 FasterTransformer에서 생성되는 중간 데이터와 DFX에서 사용하는 중간 데이터의 모양을 비교하여 이를 맞춰주는 재배열 커널(Reshape Kernel)을 추가하였다. 화살표 하나는 32개의 데이터를 의미하며, 양쪽 화살표의 색이 같으면 같은 데이터임을 의미한다.



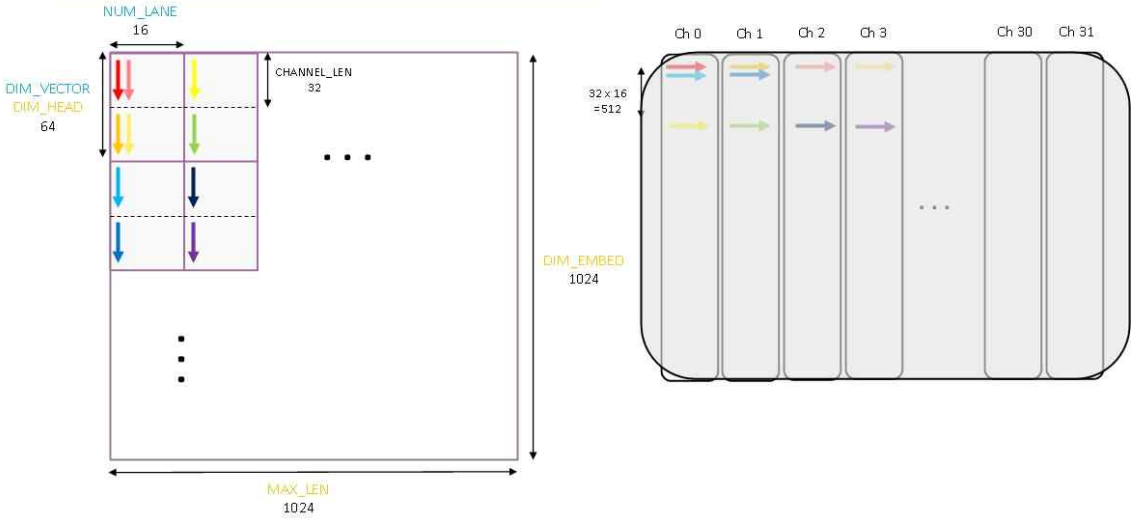
[그림 10] 중간 데이터 1의 재배열을 시각화한 모식도



[그림 11] 중간 데이터 2의 재배열을 시각화한 모식도

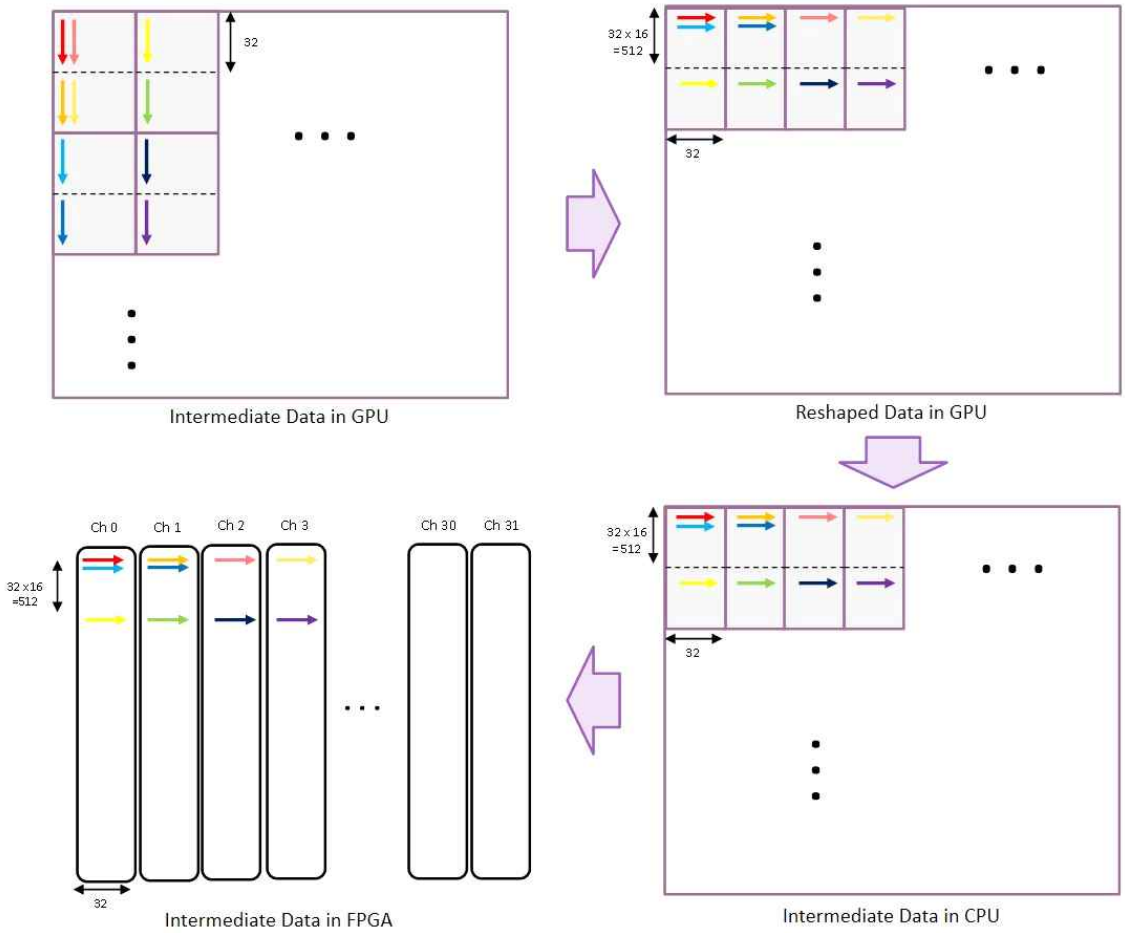
[그림 10]과 [그림 11]은 FasterTransformer에서 생성된 중간 데이터와 FPGA의 HBM가 어떻게 대응되는지에 대해 시각화한 그림이다. 좌측의 Matrix는 단일 디코더에서 생성되는 중간 데이터이다. 이 데이터는 그림과 같이 재배열된 후 32개의 배열로 나누어지고, 각 배열은 32개의 HBM 채널에 전송된다. 그림이 두 개인 이유는 중간 데이터가 모양이 다른 두 개로 나누어져 있기 때문이다. 두 데이터를 모두 전달하기 위해서 각 데이터에 맞는 두 개의 재배열 커널을 구현하였다.

각 모델은 고유의 파라미터 값을 가지고 있다. 예를 들어 GPT2는 DIM_HEAD 값이 64지만, LLAMA-7B의 경우 128이다. 재배열 커널을 파라미터 기반으로 구현하였기에 다른 모델이 들어오더라도 해당 값만 바꿔주면 정상적으로 재배열을 진행할 수 있다.



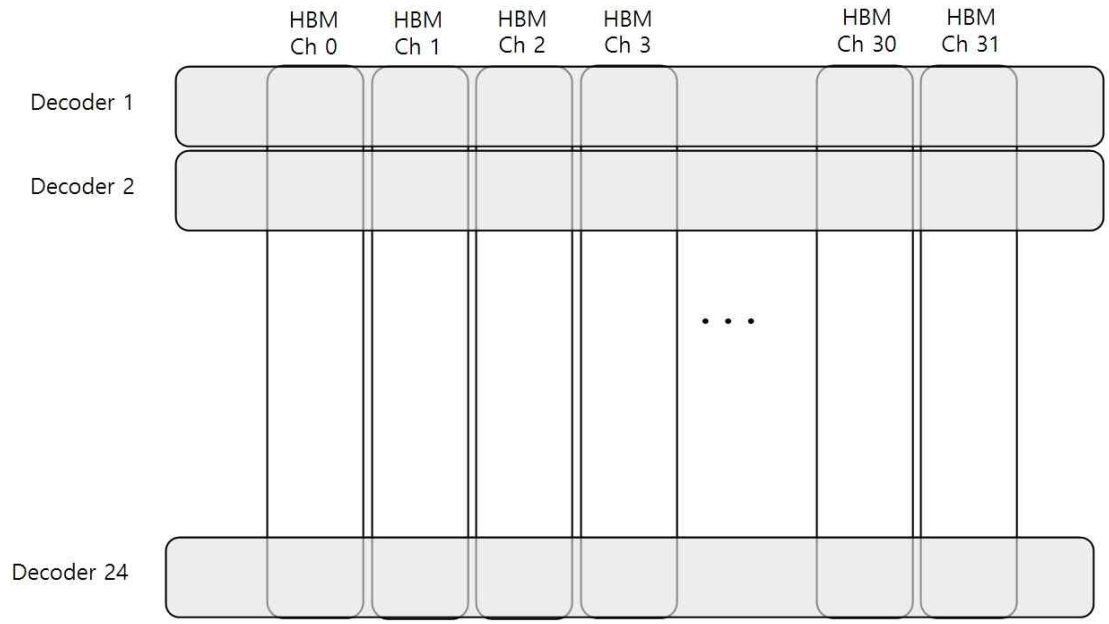
[그림 12] 단일 디코더의 중간 데이터와 HBM의 대응 관계

[그림 12]은 중간 데이터와 HBM이 어떻게 대응되는지에 대한 이해를 돕기 위한 자료이다. 단일 디코더에서 전송되는 중간 데이터를 회색 사각형으로 표시하였고, 후술되는 [그림 13]과 [그림 14]에 있는 회색 사각형의 경우 [그림 12]에 있는 회색 사각형과 같은 데이터를 의미한다.



[그림 13] 중간 데이터가 이동하는 과정

[그림 13]은 단일 디코더에서 생성된 중간 데이터가 물리적으로 어떤 메모리를 거쳐 HBM으로 전달되는지 나타낸 그림이다. 우선 GPU에 있는 중간 데이터는 재배열 커널을 통해 GPU 내부에서 재배열된다. 이 데이터는 CUDA API를 통해 Host로 전달된다. 이때 GPU코드는 실행하기 전 HBM의 주소가 포함된 구조체를 입력으로 받게 되는데, OpenCL API를 통해 해당 주소값으로 복사를 진행하여 최종적으로 FPGA에 복사된다.

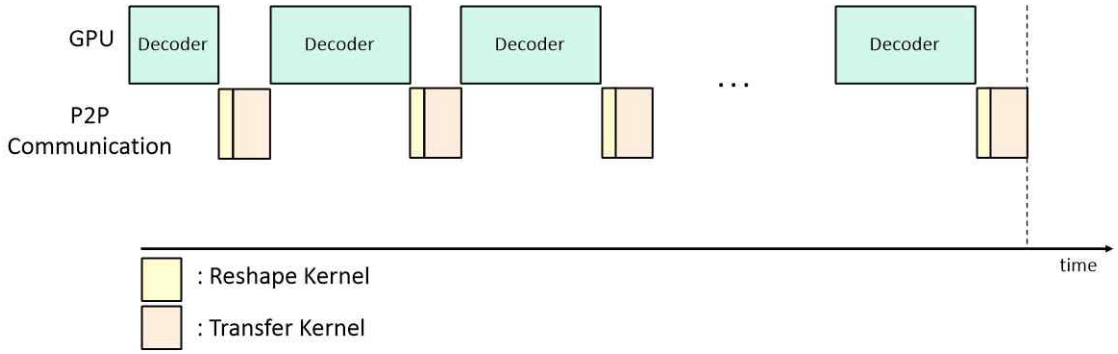


[그림 14] 여러 디코더의 중간 데이터와 HBM의 대응 관계

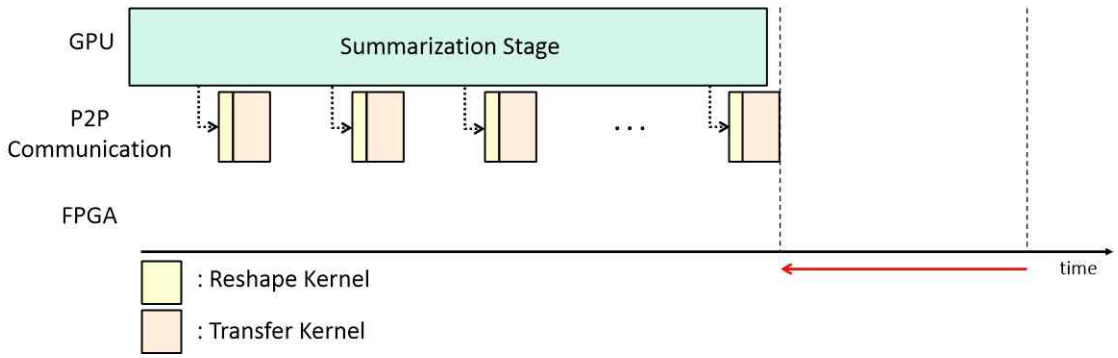
[그림 14]은 24개의 디코더를 거치며 [그림 13]이 묘사한 통신을 총 24번 진행했을 때 여러 중간 데이터들과 HBM 사이의 대응 관계를 나타낸 모식도이다. 각 회색 사각형은 단일 디코더에서 연산된 중간 데이터이며, 각 데이터는 재배열된 후 32개의 HBM 채널로 나눠져서 전송된다. 다음 중간 데이터는 이전 중간 데이터의 바로 다음 주소에 이어서 전송된다. GPT2에서 24개의 디코더로부터 중간 데이터의 통신이 완료되고 나면 FPGA는 HBM에 누적된 중간 데이터들을 토대로 생성 스테이지 연산을 수행한다.

3.4 통신 커널과 최적화

재배열 커널을 통해 재배열된 배열들은 통신 커널(Communication Kernel)을 통해 GPU 메모리에서 HBM으로 복사된다. GPT2의 경우 디코더가 총 24개로, [그림 15]와 같이 한 번의 요약 스테이지에서 통신 커널이 총 24번 수행된다.



[그림 15] 요약 스테이지에서 통신이 동기적으로 수행되는 시퀀스

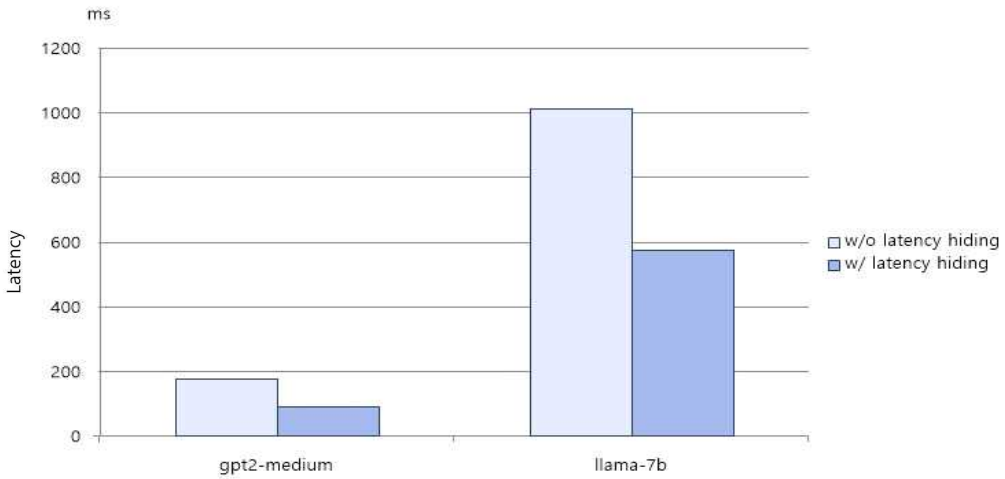


[그림 16] 요약 스테이지에서 통신이 비동기적으로 수행되는 시퀀스

추가적인 최적화 없이 구현하게 되면 [그림 15]와 같이 통신에 소요되는 시간만큼 전체 수행 시간이 늘어나게 된다. 중간 데이터의 경우 PCIe 통신에 의해 수행되므로 이론상 중간 데이터 크기를 PCIe 통신 대역폭으로 나눈 값만큼의 시간이 걸리게 된다. 또한 중간 데이터의 크기는 모델이 가질 수 있는 최대 토큰 길이의 제곱

에 비례하는데, 작은 모델의 경우 최대 토큰 길이가 1024로 MB 단위지만, 큰 모델의 경우 32768 혹은 그 이상의 최대 토큰 길이를 가지므로 중간 데이터의 크기가 매우 커지게 된다.

따라서 확장성을 고려한다면 통신에 소요되는 시간을 최적화하는 작업이 필요하다. 본 논문에서는 통신 커널과 GPU 커널을 비동기적으로 수행하는 레이턴시 하이딩(latency hiding) 기법을 이용하여 통신 수행 시간을 최적화하였다. [그림 16]은 비동기적으로 통신 커널을 실행했을 때 시간에 따라 실행되는 커널을 나타낸 그림이다. 이때 입력 토큰이 지나치게 짧다면 GPU로 연산한 요약 스테이지의 수행시간보다 통신 커널의 수행시간이 더 길어져 수행시간을 효과적으로 숨길 수 없다. 하지만 일반적인 경우에 있어서 통신 API를 GPU 연산과 비동기적으로 수행함으로써 다음과 같이 연산 수행시간을 줄일 수 있다.

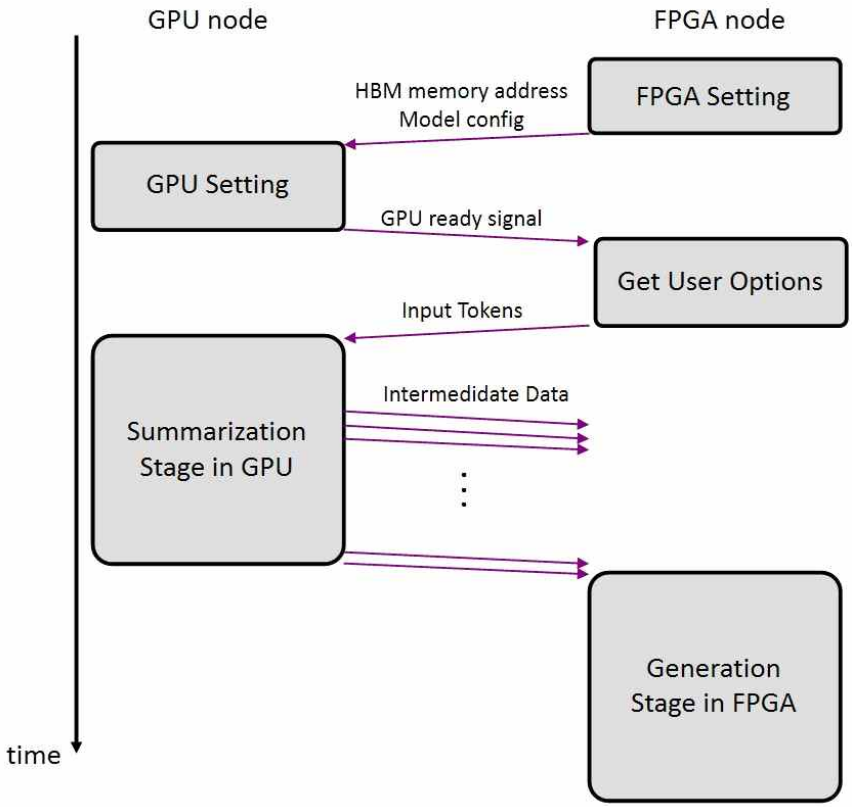


[그림 17] 동기/비동기적 통신에 따른 요약 스테이지 수행 시간 비교

[그림 17]은 두 가지 모델에 대하여 통신 커널을 동기적으로 실행했을 때와 비동기적으로 실행했을 때의 성능을 비교한 차트이다. llama-7b의 최대 토큰 길이는 2048로 gpt2-medium보다 2배 더 크므로 수행 시간이 약 4배 차이 나는 것을 확인할 수 있다. 또한 두 모델 모두에 대해 비동기적으로 통신을 수행함으로써 전체 수행시간에 드러나는 통신 수행시간이 절반으로 줄였음을 확인할 수 있다.

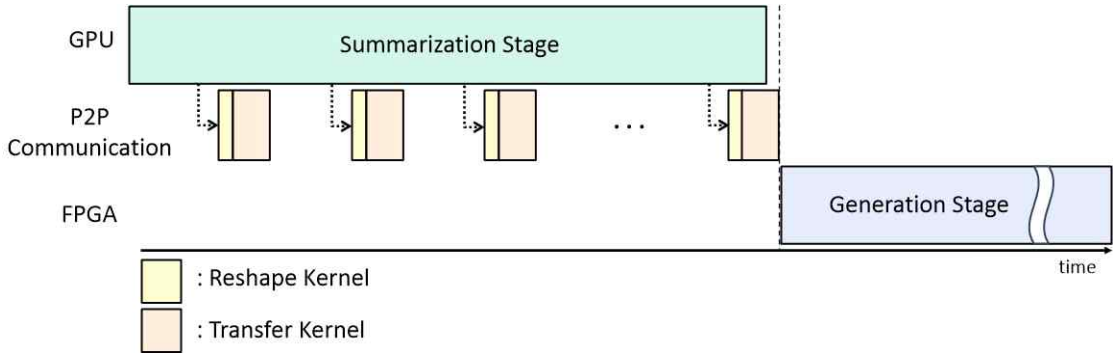
3.5 하이브리드 플랫폼

앞서 3.2-3.4에서 중간 데이터를 GPU 메모리에서 FPGA HBM으로 전송하기 위하여 구현한 재배열 커널(Reshape Kernel)과 통신 커널(Communication Kernel)에 대해 설명하였다. 이 커널들을 GPU 코드 내에 삽입하는 방식으로 구현되어있고, 커널이 통신을 수행하기 위해서는 GPU에 대한 정보 뿐만 아니라 FPGA에 대한 정보까지 모두 알아야 한다. 따라서 GPU 코드를 공유 라이브러리로 컴파일한 후, FPGA를 실행하는 DFX 코드에서 GPU 코드를 실행할 때 FPGA에 대한 정보를 입력값으로 받도록 하였다. 해당 방식으로 구현하면 통신이 종료되는 시점에 바이너리로 되어있는 GPU 코드도 종료되므로, 바로 FPGA에서 생성 스테이지를 연산할 수 있다. [그림 18]



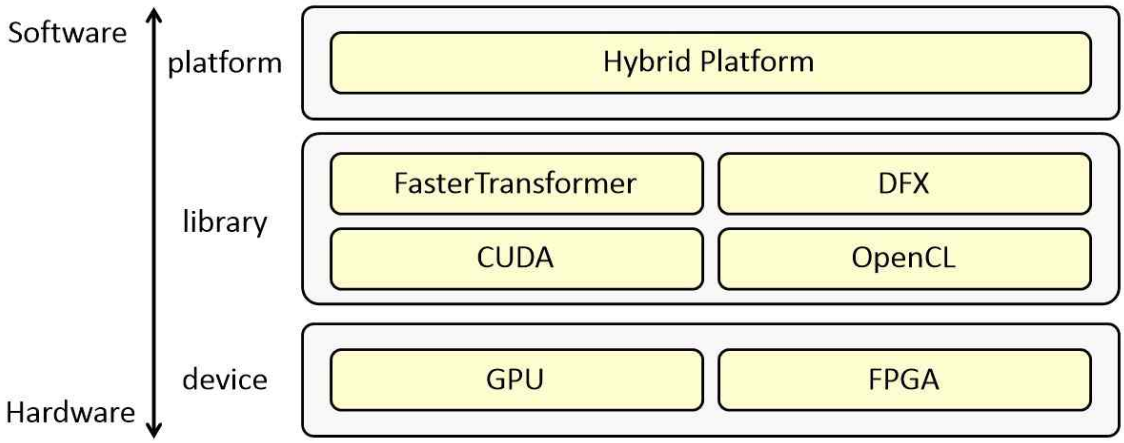
[그림 18] 각 디바이스가 수행하는 기능과 통신되는 데이터의 종류

호스트 파일을 실행하면 유저는 사용할 모델을 선택과 디바이스를 선택하게 된다. 선택된 정보를 토대로 호스트 파일은 선택된 모델에 대한 정보와 FPGA 관련 데이터가 담긴 구조체를 넘겨주며 GPU 실행을 준비하는 함수를 실행한다. 준비가 완료되면 유저는 입력 문장과 원하는 출력 문장의 길이를 선택한다. 호스트 파일은 해당 값을 넘겨주며 GPU로 요약 스테이지를 수행하는 함수를 실행한다. 해당 함수는 통신 커널이 포함되어 있는 함수로, 실행하면 자동으로 처음에 삽입했던 FPGA HBM 메모리로 중간 데이터를 저장한다. 통신이 완료되고 나면 모든 GPU 코드가 종료되며, 넘겨진 HBM 메모리 기반 FPGA로 생성 스테이지를 수행한다.



[그림 19] 하이브리드 플랫폼의 전체 시퀀스

FPGA에서 요약 스테이지와 생성 스테이지는 둘 다 모델을 불러온다는 점에서 큰 틀은 같지만, 입력값에서 차이가 있다. 해당 입력값을 외부에서 받을 수 있도록 수정하여 DFX 코드에서 생성 스테이지만 연산할 수 있도록 구현하였다. 구현된 하이브리드 플랫폼의 전체 흐름은 [그림 19]과 같다.



[그림 20] 하이브리드 플랫폼 구조

더 많은 모델과 옵션에 대해 능동적으로 처리할 수 있도록 확장성(Scalability)를 고려한다면 프로그래머블(Programmable)하도록 API화를 해야한다. 앞서 언급하였듯 FasterTransformer는 GPU에서 LLM을 효율적으로 가속할 수 있는 CUDA 기반 라이브러리이고, DFX는 FPGA에서 LLM을 효율적으로 가속할 수 있는 구조로 OpenCL을 기반으로 작동한다. 본 시스템을 위하여 수정된 FasterTransformer 라이브러리를 다시 API화하여 공유 라이브러리로 컴파일하였고, 이를 기존에 FPGA를 컨트롤하는 OpenCL 기반 호스트 파일에서 호출하는 방식으로 구현하였다. 결과적으로 하이브리드 플랫폼을 사용하는 유저 입장에서는 GPU와 FPGA를 모두 컨트롤할 수 있다.

제 4장 실험 결과

4.1 실험 방법

본 연구는 하이브리드 플랫폼에 있는 GPU와 FPGA에 각각 LLM의 다른 스테이지를 할당하는 방법을 사용하여 LLM 추론을 가속했다. LLM의 경우 GPU, FPGA 등 단일 하드웨어를 이용한 가속기 [11, 12, 13, 14, 15, 16] 는 제안되었으나, GPU, FPGA를 둘 다 사용하는 연구는 아직 발표된 사례가 없다. 따라서 비교군으로 단일 종류의 디바이스를 사용한 연구를 설정하였고, GPU 환경에서 LLM 연산을 최적화한 라이브러리인 FasterTransformer [5] 와 FPGA 환경에서 LLM 연산을 잘 수행할 수 있는 아키텍처를 디자인한 DFX 코드 [6] 와 비교를 진행하였다.

Paper	Device	Application	Optimization Scheme
FasterTransformer[5]	GPU	LLM	Custom Kernel Kernel Fusion
A3 [7]	GPU	LLM	Approximation
GODO [8]	GPU	LLM	Quantization
SpAttn [9]	GPU	LLM	Pruning
DFX[6]	FPGA	LLM	VPU, MPU for LLM
Brainwave [10]	FPGA	LLM	Novel SIMD ISA
FTRAN [11]	FPGA	LLM	Model Compression
Hype-Training [12]	hybrid	CNN Training	Kernel-based Runtime Scheduling
FARNN [13]	hybrid	RNN Training	Allocate memory bound task in FPGA
Cong. [14]	hybrid	Autonomous driving	Precision-aware Operation Partitioning
Walther. [15]	hybrid	CNN Inference	Decompose Conv. Layer
FleecRec [16]	hybrid	Recommandation System Inference	Allocate LUT in FPGA

[표 1] 관련 선행 연구 정리

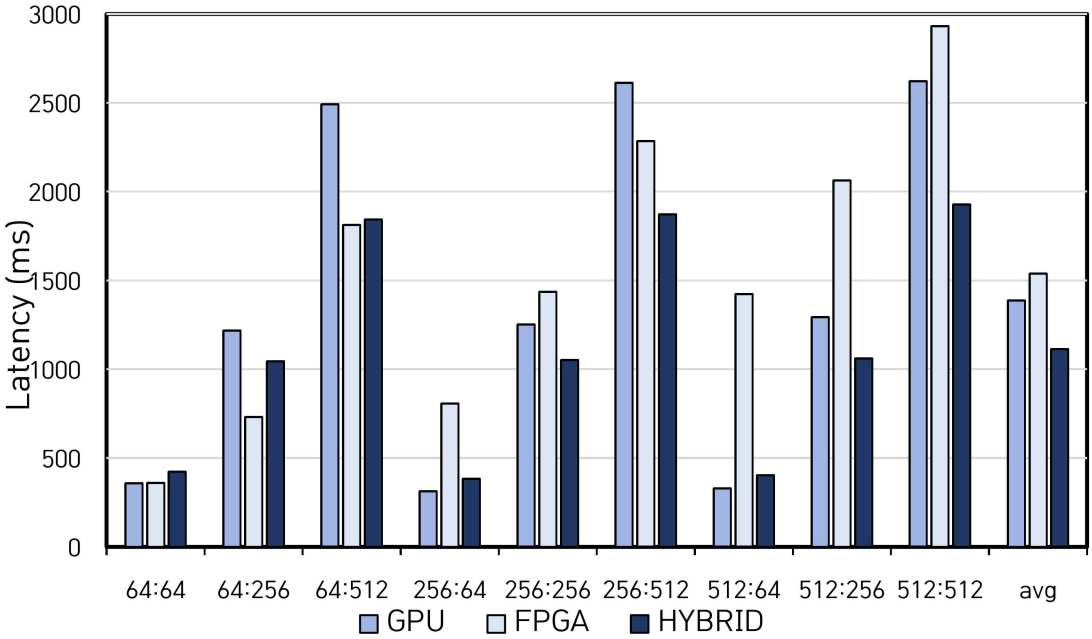
LAMBADA 등 LLM에 관련하여 널리 알려진 벤치마크의 경우 주로 LLM의 정확도와 관련된 지표(Metric)를 측정하는 벤치마크이다. 본 논문에서 제안하는 하이브리드 플랫폼은 없던 연산을 새로 만들거나 변형한 것이 아닌, 같은 연산을 수행하는 두 프로그램을 합친 형태이므로 결과값이 단일 하드웨어로 연산한 결과값과 완전히 일치한다. 따라서 매트릭(Metric)으로 수행 시간(Latency)를 설정하였고, 세부적으로는 입력 토큰과 출력 토큰의 길이를 다양하게 설정하는 방법으로 실험을 구성하였다.

Device	HBM Memory	HBM Bandwidth	Power
A10 (GPU)	24GB	600GB/s	150W
U55C (FPGA)	16GB	460GB/s	150W

[표 2] A10과 U55C의 성능 비교표

[표 2]은 NVIDIA의 A10과 XILINX의 U55C 디바이스의 스펙을 비교한 표이다. A10과 U55C는 서로 메모리나 에너지 효율 측면에서 비슷한 스펙을 가졌기에 하드웨어는 NVIDIA의 A10 GPU와 AMD의 U55C FPGA를 각 1개씩 사용하였다. LLM은 gpt2-medium 모델을 사용하여 실험을 진행하였다. 본 논문에서 사용한 gpt2-medium 모델의 경우 입력 토큰과 출력 토큰의 합이 1024를 넘지 못하게 설계되어 있으므로, 입력 토큰과 출력 토큰의 최댓값을 512로 설정하였고, 중간값을 256, 최솟값을 64로 설정하였다. 즉, 총 9가지 경우의 수에 대해 실험을 진행하였다.

4.2 결과 요약



[그림 21] GPU, FPGA, HYBID의 LLM 성능 그래프 (input token: output token)

[그림 21]은 하이브리드와 타 디바이스의 성능을 비교한 그래프이다. 하단의 숫자 A:B에서 A는 입력 토큰의 길이, B는 출력 토큰의 길이를 의미한다. 입력 토큰과 출력 토큰의 길이를 64, 256, 512로 설정하여 총 9가지 경우의 수에 대하여 실험을 진행했다. y축은 해당 경우의 수를 실행하는데 걸린 시간을 의미하므로 그 값이 작을수록 빠르게 연산을 수행함을 의미한다. 우선 입력 토큰과 출력 토큰이 모두 작은 64:64의 경우 GPU와 FPGA가 비슷하게 앞서는 성능을 보여주었다. 입력 토큰만 작은 64:256과 64:512의 경우 GPU가 가장 좋은 성능을 보여주었다. 반대로 출력 토큰만 작은 256:64와 512:64의 경우 FPGA가 가장 좋은 성능을 보여주었다. 마지막으로 입력 토큰과 출력 토큰 둘 다 작지 않은 나머지 경우의 수에 대해서는 하이브리드가 가장 좋은 성능을 보여주었다.

4.3 결과 분석

5.1의 내용을 토대로 [표 3]을 [표 4]와 같이 정리할 수 있다.

GPU:FPGA:HYBRID		output size		
		64	256	512
input size	64	369.00 : 371.99 : 436.68	1257.6 : 755.44 : 1079.12	2574.02 : 1872.02 : 1905.14
	256	322.42 : 833.00 : 395.71	1292.48 : 1484.21 : 1087.51	2699.78 : 2359.7 : 1934.68
	512	339.09 : 1469.47 : 416.06	1336.04 : 2132.07 : 1096.35	2708.12 : 3029.38 : 1992.02

[표 3] GPU, FPGA, HYBID의 LLM 추론 성능 지표

하이브리드는 통신으로 인한 수행시간이 추가되지만, GPU와 FPGA가 가지고 있는 상호보완적인 특성을 통해 시간적 이득을 보는 특징을 가지고 있다. 따라서 하이브리드가 어떤 경우에 가장 빠르게 연산하는지 분석하기 위해서는 디바이스 별 요약 스테이지 수행시간, 생성 스테이지 수행시간, 통신 수행시간을 알아야 한다. 우선 요약 스테이지의 수행시간은 별도의 계산 없이 출력 토큰을 1로 설정하여 구할 수 있다. 요약 스테이지 수행시간은 앞서 [그림 7]에서 측정하였다.

요약 스테이지 수행시간 = 출력 토큰이 1인 언어모델 수행시간 (1)

언어모델의 수행시간은 요약 스테이지 수행시간과 생성 스테이지 수행시간의 합이다. 따라서 요약 스테이지의 수행시간과 언어모델 수행시간의 차이로 생성 스테이지의 수행시간을 구할 수 있다.

생성 스테이지 수행시간 = 언어모델 수행시간 - 요약 스테이지 수행시간 (2)

하이브리드에서의 언어모델 수행시간은 GPU의 요약 스테이지 수행시간과 FPGA의 생성 스테이지 수행시간, 그리고 통신 수행시간의 합이다. 이를 이용해 통신 수행시간도 구할 수 있다. 식으로 표현하면 다음과 같다.

통신 수행시간 = 하이브리드 언어모델 수행시간
- 요약 스테이지 수행시간 - 생성 스테이지 수행시간 (3)

하이브리드가 GPU보다 빠른 성능을 내기 위해서는 FPGA 생성 스테이지의 수행시간과 통신 수행시간의 합이 GPU 생성 스테이지의 수행시간보다 작아야 한다.

GPU 생성 스테이지 수행시간 - FPGA 생성 스테이지 수행시간 > 통신 수행시간 (4)

마찬가지로, 하이브리드가 FPGA보다 빠른 성능을 내기 위해서는 GPU 요약 스테이지의 수행시간과 통신 수행시간의 합이 FPGA 요약 스테이지의 수행시간보다 작아야 한다.

FPGA 요약 스테이지 수행시간 - GPU 요악 스테이지 수행시간 > 통신 수행시간 (5)

두 가지 조건 (4, 5) 이 모두 충족되는 입출력 토큰 길이에 대해서 하이브리드가 GPU, FPGA보다 빠르게 연산을 수행한다. 다음은 수식 (1, 2, 3)을 기반으로 [그림 7]과 [표 4]의 데이터를 가공한 데이터이다.

input size	output size	GPU		FPGA		통신 (ms)	Hybrid
		요약 스테이지(ms)	생성 스테이지(ms)	요약 스테이지(ms)	생성 스테이지(ms)		
64	64	7.61	361.39	156.457	215.533	213.537	436.68
256		10.25	312.17	614.437	218.563	166.897	395.71
512		15.34	323.75	1247.52	221.95	178.770	416.06

[표 4] small output에서 디바이스 별 성능 비교표

[표 4]는 출력 토큰이 작은 경우에 대해 정리한 표이다. [그림 5]에서 요약 스테이지 수행시간과 [표 3]에서 언어모델의 수행시간을 연산하여 순차적으로 생성 스테이지와 통신 수행시간을 계산하였다.

FPGA와 비교했을 때, 입력 토큰의 개수가 커질수록 FPGA의 요약 스테이지 수행시간이 크게 상승하는 점을 확인할 수 있다. 따라서 (5)에 의해 입력 토큰이 커질수록 하이브리드가 FPGA보다 더 빠르게 연산을 수행하였다.

그러나 GPU와 비교했을 때, GPU와 FPGA의 생성 스테이지 수행시간은 입력 토큰의 길이와 무관하게 거의 변함이 없었는데, 이는 생성 스테이지 수행시간이 출력 토큰의 길이에 영향을 받기 때문이다. 통신 수행시간은 입력 토큰의 길이와 무관하게 모두 비슷한 수치를 보였다. 이는 입력 토큰의 길이와 출력 토큰의 길이에 무관하게 항상 같은 크기의 중간 데이터를 전송하기 때문이다. 따라서 출력 토큰의 길이가 작다면 입력 토큰의 길이에 무관하게 비슷한 차이로 (4)에 의해 GPU가 더 빠르게 연산을 수행하였다.

input size	output size	GPU		FPGA		통신 (ms)	Hybrid
		요약 스테이지(ms)	생성 스테이지(ms)	요약 스테이지(ms)	생성 스테이지(ms)		
64	64	7.61	361.39	156.457	215.533	213.537	436.68
	256	7.61	1249.99	156.457	598.983	148.847	755.35
	512	7.61	2566.41	156.457	1715.56	181.97	1905.14

[표 5] small input에서 디바이스 별 성능 비교표

[표 5]는 입력 토큰이 작은 경우에 대해 정리한 표이다. 마찬가지로 [그림 5]와 [그림 7]의 데이터를 순차적으로 연산하여 생성 스테이지와 통신 수행시간을 계산하였다.

GPU와 비교했을 때 출력 토큰이 길어짐에 따라 GPU의 생성 스테이지와 FPGA의 생성 스테이지의 수행시간이 모두 크게 증가하였다. 그러나 GPU 생성 스테이지 수행시간의 증가폭이 FPGA 생성 스테이지 수행시간의 증가폭보다 더 크게 나타났고, 통신 수행시간은 출력 토큰 길이와 무관하게 비슷한 값을 보였다. 따라서 (4)에 의해 출력 토큰의 길이가 길어짐에 따라 점점 하이브리드가 빠른 성능을 보여주었다. 추가로 통신 수행시간에 대해 앞에서 분석한 결과와 종합하면 통신 수행시간은 입력 토큰과 출력 토큰의 길이와 무관하게 항상 비슷한 값을 가짐을 확인할 수 있다.

그러나 FPGA와 연산했을 때, FPGA와 GPU의 요약 스테이지는 입력 토큰의 길이에 영향을 받기에 같은 값을 가지고 있었고, 통신의 수행시간 역시 큰 변화가 없었다. 따라서 (5)에 의해 출력 토큰의 길이와 무관하게 비슷한 차이로 FPGA가 더 빠르게 연산을 수행하였다.

input size	output size	GPU		FPGA		통신
		요약 스테이지	생성 스테이지	요약 스테이지	생성 스테이지	
256	256	10.25	1282.23	614.437	869.773	207.487
256	512	10.25	2689.53	614.437	1745.265	179.165
512	256	15.34	1320.70	1247.52	884.55	196.46
512	512	15.34	2692.78	1247.52	1779.86	196.82

[표 6] non-trivial token에서 디바이스 별 성능 비교

마지막으로 [표 6]은 입력 토큰과 출력 토큰이 모두 작지 않은 경우에 대해 정리한 표이다. [그림 5]와 [그림 7]의 데이터를 순차적으로 연산하여 생성 스테이지와 통신 수행시간을 계산하였다.

입력 토큰과 출력 토큰이 길어짐에 따라 GPU의 생성 스테이지와 FPGA의 요약 스테이지의 수행시간이 큰 폭으로 상승하였다. 따라서 앞장과 동일한 방법으로 비교를 진행했을 때 (4)와 (5)에 의해 토큰들의 길이가 길어질수록 하이브리드가 GPU 및 FPGA보다 더 빠르게 연산을 수행하였다.

앞에서 언급했듯 통신 수행시간의 경우 gpt2-medium 모델에서 모든 경우의 수에 대해 유사한 값을 보였다. 이는 모델에 따라 중간 데이터의 크기가 입력 토큰과 출력 토큰의 길이와 무관하게 항상 같은 크기를 가지고 있기 때문이다. 즉, 모델이 지원하는 입력 토큰과 출력 토큰의 합의 제곱에 비례해서 통신 수행시간이 길어질 것이라고 예상할 수 있다.

그에 반해 현재 출시된 모델의 크기는 [그림 1]에서 확인할 수 있듯 기하급수적으로 상승하고 있다. 예를 들어 LLM 모델 중 하나인 GPT3의 경우 모델 크기는 최대 175억 개에 달하지만, 해당 모델이 지원하는 토큰의 개수는 최대 4096개이다. 이러한 점을 보았을 때 큰 모델일수록 하이브리드가 더 좋은 성능을 보여줄 것이라고 예상할 수 있고, 추가적인 연구를 이어나갈 계획이다.

제 5장 선행 연구

5.1 LLM의 경량화

우선 어플리케이션 레벨에서는 LLM을 경량화하는 연구가 많이 이루어지고 있다. 특히 DNN 등의 기존에 있는 모델을 경량화한 선행 연구가 많이 있는 만큼 해당 아이디어들이 LLM에 적용되고 있다. 현재 출시된 대부분의 LLM은 Transformer [1] 를 기반으로 구성되어 있고, 따라서 트랜스포머 모델을 경량화하는 방향으로 연구가 많이 진행되었다.

A3 [16] 은 트랜스포머 모델에는 기존 CNN 및 RNN에 없던 Attention Layer가 추가되어있고, 해당 레이어가 연산 시간에 많은 부분을 차지하고 있다는 점에 주목하였다. Attention Layer 내부에 있는 Softmax 함수는 값들 사이의 차이를 증폭시키는 역할을 수행하는데, weight의 많은 부분이 0에 수렴하고 이 값들을 0으로 근사하여 연산을 수행하면 연산량은 크게 줄인 채 정확도를 유지할 수 있다고 주장하였다. 실험 결과 Attention Layer를 연산할 때 정확도를 크게 잃지 않으면서 에너지 효율과 연산 속도를 개선하였다.

GOBO [17] 는 Attention-based 모델에서 파라미터의 0.1%만 큰 차이를 보이고 (outlier) 나머지 99.9%는 유사한 값이었다는 점을 기반으로 32-bit인 데이터 타입을 8-bit로 양자화하여 모델 크기를 줄이고자 하였다. 양자화 결과 메모리 사용량과 메모리 트래픽을 개선하여 연산 속도와 에너지 효율을 개선하였다.

SpAttn [18] 는 Attention Layer가 데이터 이동이 복잡하고 연산량이 병목이 되지 않아 GPU에서 사용하기 어렵다는 점을 지적한다. 이를 해결하기 위해 중요하지 않은 토큰과 헤드를 제거하는 가지치기(Pruning)기법을 사용하였고, 중요도 순서대로 토큰과 헤드를 처리하는 Top-K 옵션을 구현하였다. 다음과 같은 방법으로 메모리 사용량을 줄여 연산 속도와 에너지 효율을 개선하였다.

5.2 FPGA 기반 가속기

하드웨어 레벨에서도 많은 연구가 이루어지고 있다. Brainwave [19] 은 실시간 시스템을 목적으로 빠른 속도와 높은 에너지 효율을 가지는 FPGA 기반 NPU를 설계하였다. 해당 아키텍처는 벡터와 행렬을 단위 하는 96,000개의 MAC 유닛을 SIMD 기반 명령어로 프로그래밍이 가능하도록 구현되었다. 그러나 해당 아키텍처의 경우 메모리 대역폭보다는 연산 처리량에 주목하여 구현하였기에 연산량이 병목이 되는 DNN 연산은 효율적으로 수행할 수 있었지만, 높은 메모리 대역폭이 필요한 LLM 추론에서는 좋은 성능을 보여주지 못하였다.

FTRAN [20] 은 FPGA가 AI를 가속하기에 적합한 디바이스지만, 큰 사이즈의 대규모 언어모델을 저장하기 힘들다는 점에서 착안해 대규모 언어모델을 압축하는 연구를 진행하였다. BCM(Block-Circulant Matrix) 기반 데이터 표현방식(Weight Representation)과 이를 사용하는 하드웨어 아키텍처를 개발하여 대규모 언어모델을 최대 16배까지 압축하여 연산함에 성공하였고, 그 결과 최소한의 정확도 손실로 CPU보다 최대 81배, GPU보다 최대 8.8배 에너지 효율 향상을 보여주었다.

DFX [6] 는 큰 메모리 대역폭을 필요로 하는 LLM 추론을 효율적으로 수행하기 위해 HBM이 내장된 FPGA를 사용하였다. 또한 HBM의 여러 채널을 최대한으로 활용하기 위하여 HBM의 스펙을 기반으로 VPU(Vector Processing Unit)과 MPU(Matrix Processing Unit)을 설계하였다. 결과적으로 HBM의 이론상의 최대 대역폭을 거의 모두 사용하며 LLM의 생성 스테이지를 GPU보다 빠르게 처리하는데 성공하였다. 다만 2.3에서 언급했듯 FPGA는 GPU에 비해 절대적인 연산기 개수가 부족하기에 연산량이 병목의 원인인 요약 스테이지에서는 GPU가 더 좋은 성능을 보여주었다.

5.3 훈련을 위한 하이브리드 플랫폼

GPU와 FPGA 기반 하이브리드 시스템을 이용하여 인공지능 모델의 트레이닝을 최적화하는 연구가 다양한 분야에서 진행되고 있다. 모델의 트레이닝은 매우 오랜 시간 가동되어야 하기에 전력이 많이 소모되는데, FPGA에 특정 연산에 최적화된 아키텍처를 디자인하면 전력 소모량을 획기적으로 줄일 수 있기 때문이다.

Hype-training [21] 는 CNN과 Transformer 모델의 훈련(Training)을 하이브리드 플랫폼으로 실행하여 소비전력을 절약하는 연구를 진행하였다. GPU와 FPGA가 서로 상호보완적인 특성이 있다는 점을 인지하였고, 모델 훈련 연산을 디바이스 특성에 맞게 할당하여 전력 소모를 줄이도록 연구를 진행하였다. GPU를 사용하기 위해 Tensorflow 프레임워크를 사용했고, 에너지 절약을 목적으로 FPGA 아키텍처를 설계했다. 실험 결과 모델 훈련 과정에서 GPU보다 평균 44.3%의 에너지 절약에 성공했다.

FARNN [22] 는 RNN 훈련(Training) 연산을 GPU에 효율적인 연산과 비효율적인 연산으로 분리하여 GPU와 FPGA에 할당하는 하이브리드 플랫폼을 제안하였다. GPU를 사용하기 위해 cuDNN library를 사용하였고, GPU에 비효율적인 연산을 잘 수행할 수 있는 FPGA 아키텍처를 설계하여 구현하였다. 실험 결과 FARNN은 GPU보다 최대 4.2배의 속도 향상, 34%의 에너지 효율 향상을 이루어냈다.

5.4 추론을 위한 하이브리드 플랫폼

GPU와 FPGA를 사용하여 인공지능 모델의 추론을 가속화하는 연구도 진행되고 있다. 인공지능 모델의 추론은 수행 시간(Latency)가 중요한데, 복잡한 연산의 경우 단순히 연산기만 많은 GPU보다 특정 연산에 최적화된 아키텍처를 통해 더 빠르게 연산할 수 있기 때문이다.

Cong Hao [23] 는 자율주행 분야에서 하이브리드 시스템을 적용하는 연구를 진행하였다. 신뢰성이 높은 FPGA의 특성을 이용하여 시스템의 검증 및 오류 감지는 FPGA로 수행하고, 강력한 컴퓨팅 파워를 필요로 하는 실시간 이미지 인식 등의 테스크는 GPU로 수행하도록 구성하여 자율주행 시스템을 이종 하드웨어로 구성하였다.

Walther [24] 는 Convolution 연산을 수행할 때 텐서 및 필터의 모양에 따라 GPU, FPGA가 수행할 때 필요한 시간과 에너지가 다름에 주목하였다. 이를 기반으로 DNN 추론을 최적화하기 위하여 GPU와 FPGA로 구성된 플랫폼을 구성하였고, Convolution 연산의 각 부분을 더 잘 수행할 수 있는 디바이스로 할당하는 기법을 사용하였다.

FleetRec [25] 는 추천 시스템의 연산 중 일부인 LUT(Look Up Embedding Table)이 단기간에 많은 메모리에 접근해야 한다는 점을 주목하여 이에 최적화된 아키텍처를 개발하였다. 메모리가 병목이 되는 LUT 연산은 FPGA로, MIP 등 연산이 병목이 되는 연산은 GPU로 수행하는 이종 하드웨어 클러스터를 구성해 추천 시스템의 추론을 가속하였다.

제 6장 결론

본 논문에서는 LLM의 크기와 LLM의 시장이 기하급수적으로 커짐에 따라 더 좋은 성능의 가속기를 만들고자 하였고, 하드웨어의 특성 및 LLM 연산에 대한 이해를 기반으로 상호보완적인 성격을 가지고 있는 두 디바이스(GPU, FPGA)에 각각 잘 수행할 수 있는 연산을 할당하여 LLM 추론을 더 빠르게 수행할 수 있는 GPU-FPGA 하이브리드 플랫폼을 제안했다. 서로 다른 두 프로젝트를 연결하기 위해 중간 데이터를 재배열하는 커널을 추가하였고, 통신 커널을 구현하는 과정에서 Latency Hiding 기법을 사용하여 최적화를 수행하였다. 실험에서는 입력 토큰과 출력 토큰의 개수에 변화를 주며 기존의 플랫폼과 성능 비교를 실시하였고, 실험 결과 작지 않은 입력 토큰과 출력 토큰에 대해 기존의 플랫폼보다 빠르게 연산을 수행하였다.

본 논문의 경우 하이브리드 플랫폼의 발전 가능성을 보여주는 이정표로서의 의의가 있다. 단기적으로는 더 큰 모델과 더 큰 입출력 토큰에 대한 실험을 진행한다면 더 높은 정확도를 가진 모델에서 더 큰 성능 향상을 보여줄 것이라고 예상하고, 장기적으로는 다중 디바이스 및 다중 서버로 연구를 확장하여 데이터 센터 레벨에서의 연구를 진행한다면 현실에도 적용 가능한 더 큰 가치를 창출할 수 있을 것으로 기대한다.

참 고 문 헌

[1] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[2] Jouppi, Norman P., et al. "In-datacenter performance analysis of a tensor processing unit." Proceedings of the 44th annual international symposium on computer architecture. 2017.

[3] Weizenbaum, Joseph. "ELIZA—a computer program for the study of natural language communication between man and machine." Communications of the ACM 9.1 (1966): 36-45.

[4] Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI blog 1.8 (2019): 9.

[5] Wolf, Thomas, et al. "Huggingface's transformers: State-of-the-art natural language processing." arXiv preprint arXiv:1910.03771 (2019).

[6] 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.

[7] GitHub - NVIDIA/FasterTransformer: Transformer related optimization, including BERT, GPT

[8] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.

[9] Amatriain, Xavier. "Transformer models: an introduction and catalog." arXiv preprint arXiv:2302.07730 (2023).

[10] https://www.top500.org/lists/top500/2022/11/highs/

[11] https://www.furiosa.ai/

[12] https://rebellions.ai/

[13] https://www.sapeon.com/

[14] Chowdhery, Aakanksha, et al. "Palm: Scaling language modeling with pathways." arXiv preprint arXiv:2204.02311 (2022).

[15] Jouppi, Norman P., et al. "A domain-specific supercomputer for training deep neural networks." Communications of the ACM 63.7 (2020): 67-78.

[16] Ham, Tae Jun, et al. "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation."2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2020.

[17] Zadeh, Ali Hadi, et al. "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference."2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020.

[18] Wang, Hanrui, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning."2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021.

[19] Fowers, Jeremy, et al. "A configurable cloud-scale DNN processor for real-time AI."2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2018.

[20] Li, Bingbing, et al. "Ftrans: energy-efficient acceleration of transformers using fpga." Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design. 2020.

[21] He, Xin, et al. "Enabling energy-efficient DNN training on hybrid GPU-FPGA accelerators." Proceedings of the ACM International Conference on Supercomputing. 2021.

[22] Cho, Hyungmin, Jeesoo Lee, and Jaejin Lee. "FARNN: FPGA-GPU hybrid acceleration platform for recurrent neural networks." IEEE Transactions on Parallel and Distributed Systems 33.7 (2021): 1725-1738.

[23] Hao, Cong, et al. "A hybrid GPU+ FPGA system design for autonomous driving cars." 2019 IEEE International Workshop on Signal Processing Systems (SiPS). IEEE, 2019.

[24] Carballo-Hernández, Walther, Maxime Pelcat, and François Berry. "Why is FPGA-GPU heterogeneity the best option for embedded deep neural networks?." arXiv preprint arXiv:2102.01343 (2021).

[25] Jiang, Wenqi, et al. "Fleetrec: Large-scale recommendation inference on hybrid gpu-fpga clusters." Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021.

ABSTRACT

GPU/FPGA-based Hybrid Platform for Accelerating LLM Inference

Park, Hyun Jun
Dept. of Electrical and Electronic Engineering
The Graduate School
Yonsei University

This paper proposes a hybrid platform for accelerating the LLM inference based on the characteristics of LLM operations and hardware platforms (GPU, FPGA). The LLM can be divided into two stages, the Summarization Stage and the Generation Stage. The Summarization Stage extracts context information from input tokens, and the Generation Stage computes the output tokens based on the context information. The Summarization Stage has a high parallel opportunity because it can process input tokens in parallel. On the contrary, the Generation Stage is consists of sequential operations because the current output token has dependencies on the previous output token.

Most current AI models uses NVIDIA's GPUs because they shows great performance in various metrics. While the summary stage, bottlenecked by computational load, benefits greatly from the high parallelism of GPUs, but the generation stage, bottlenecked by memory bandwidth, does not fully utilize the computational capabilities of GPUs.

Some researches are attmpting to optimize them at the hardware level. Notably, DFX [6] designed an architecture optimized for LLM operations, achieving faster processing in generation stages under certain conditions using FPGA, compared to GPU.

This paper begins from this point. We thought the hybrid environment that uses both GPU and FPGA can operate faster than the conventional homogeneous systems if the summary stage is assigned to GPU and the generation stage is assigned to FPGA.

To construct this platform, we followed n steps. First, we established using NVIDIA's A10 and XILINX's U55C, and selected the FasterTransformer [5] and DFX [6] codes to control each device. Second, we divided LLM operations into tasks to be executed on the GPU and FPGA, while analyzing what data need to be transferred between heterogeneous hardwares. Third, we implemented reshaping and communication kernels to transfer intermediate data from GPU to FPGA. Fourth, we uses Latency Hiding technique to reduce communication overhead. Finally, for scalability, we developed an API with c++ and python wrapper to implement a Hybrid Platform.

To verify the performance of this platform, we implemented experiments with various sizes of input and output tokens. Compared to operations performed on a single device, the Hybrid Platform demonstrated up to 1.56 times faster processing for substantial input and output tokens.

We can find prior researches related to GPU-FPGA hybrid systems. Hype-training [12] and FARNN [13] are aimed to optimize the training of CNN and Transformer models, and Walther [15] and FleetRec [16] are aimed to optimized inference for CNN and recommendation systems. All of them uses GPU-FPGA heterogeneous hardware, but there was no prior research to accelerate LLM inference using heterogeneous hardware. Our research holds significance in this point.

Additionally, we designed hybrid platforms with scalability. So it not only allows for adaptation to new models and options on a micro level, but also enables research expansion to multiple hardware and servers on a macro level. We anticipate significant value will be created from extending this research to data center-scale cluster platforms.

Key words : Large Language Model(LLM), LLM Inference, Text Generation, Hybrid System, Heterogeneous Hardware