rebellions_

RBLN 컴파일러 이해하기

Oct 01, 2024

AI 추론에서의 컴파일러

컴파일러는 고수준 코드를 저수준 머신 코드로 변환하는 역할을 하지만, 딥러닝 모델의 경우 그 중요성이 특히 커집니다. RBLN 컴파일러가 AI 추론을 가속화하는 과정에서 어떤 역할을 하는지를 이해하려면, 먼저 AI 컴파일러 특유의 특성을 살펴볼 필요가 있습니다.

전역 데이터 기반의 최적화

AI 컴파일러는 추론 과정에서 모델 전체의 연산 그래프와 데이터 의존성을 분석해, 전역적 관점에서 실행 효율을 극대화하는 최적화를 수행합니다. 이 과정에는 스케줄링, 메모리 할당, 병렬 실행과 같은 전략이 포함됩니다.

이러한 과정을 통해 컴파일러는 지연(latency)을 최소화하고 처리량(throughput)을 극대화하며, 리벨리온 ATOM™과 같은 하드웨어 리소스를 최적 수준으로 활용할 수 있는 고도로 최적화된 코드를 생성합니다.

컴파일 타임 메모리 및 의존성 관리

AI 추론 과정에서 메모리 할당, 캐시(SRAM) 활용, 그리고 연산 간 의존성 관리는 **컴파일 단계**에서 수행됩니다. 이를 통해 리소스 할당과 데이터 흐름을 정밀하게 제어할 수 있습니다.기존 컴파일러가 메모리를 **런타임에서 동적으로 할당하고**, 캐시 및 의존성 관리를 **하드웨어에 위임**하는 것과 달리, AI 컴파일러는 **모델 전체의 연산 그래프를 기반으로 메모리를 사전에 할당하고 SRAM 내 데이터 배치를 최적화**합니다. 이러한 접근 방식은 모든 메모리 및 의존성 요소를 컴파일 과정에 긴밀히 통합함으로써, **지연을 최소화하고 효율성을 극대화**하며, 개별 AI 모델의 실행 특성에 맞게 최적화된 성능을 제공합니다.

RBLN 컴파일러

최적화 목표

AI 전용 컴파일러의 핵심 목표는 두 가지로 요약됩니다:

- **직접 메모리 접근(DMA)을 최소화하면서 빠른 연산 달성**
- **연산과 메모리 작업의 병렬화 극대화**

DMA는 DRAM과 SRAM 간 데이터를 빠르게 전송하여, 연산 유닛이 최대 활용도로 동작할 수 있도록 데

이터 흐름을 안정화합니다. 동시에, ATOM™의 뉴럴 엔진(Neural Engine)과 같은 다중 코어에서 연산 작업을 병렬로 수행함으로써 신경망의 서로 다른 구간을 동시에 처리할 수 있습니다.

DMA와 연산 작업을 지능적으로 스케줄링하면, 유휴 시간이 최소화되고, 지연이 줄어들며, 처리량이 향상됩니다. 이를 통해 데이터가 병목 없이 지속적으로 공급되어, 특히 **실시간·저지연 응용 환경에서 AI 모델의 성능을 극대화**할 수 있습니다.

컴파일 단계

1. 그래프 최적화 및 연산 융합 (Op Fusion)

AI 모델은 노드와 엣지로 구성된 연산 그래프(computational graph)로 표현됩니다. 컴파일러는 **공통 부분식 제거(CSE), 불필요한 코드 제거(DCE)** 등의 기법을 통해 그래프를 간소화하여 실행 시간을 줄입니다.

또한, **Op Fusion**을 통해 여러 연산을 하나의 연산으로 융합해 **텐서 및 벡터 연산의 병렬 실행**을 가능하게 하고, 공유 메모리나 뉴럴 엔진의 스크래치 패드(Scratch Pad) 접근 횟수를 줄입니다.

2. 연산 분할(Op Splitting) 및 그룹화(Grouping)

일부 연산은 단일 SRAM 용량을 초과할 만큼 크기 때문에, 효율적으로 처리하기 위해 **작은 단위로 분할(Splitting)** 해야 합니다. 이때 인접한 연산 간의 관계를 고려해 **최적의 실행 순서로 재배치(Grouping)**하며, 불필요한 데이터 이동을 최소화합니다. 이 과정은 하드웨어 활용도를 극대화하고 전체 실행 효율을 높입니다.

3. 연산 타일링(Op Tiling)

뉴럴 엔진 내부 연산은 **타일링(Tiling)** 기법을 통해 더욱 최적화됩니다. 타일링은 연산을 여러 뉴럴 엔진에 **균등하게 분할**하여 병렬성을 강화하는 과정입니다. 리벨리온의 **연산 라이브러리**(Compute Library)는 연산 타입(예: 행렬 곱, 활성화 함수)과 텐서 형태에 따라 **RBLN RISC ISA 기반의 맞춤형 프로그램**을 생성합니다. 이 과정에서 Compute Library는 **타일링 방식, 필요한 SRAM 크기, 예상 연산 시간** 등 세부 연산 파라미터를 결정하여 정밀하고 효율적인 계산을 보장합니다. 연산 라이브러리에서 산출된 이 정보는 이후 **컴파일러의 후속 패스**(예: SRAM 할당, 명령 스케줄링 등)에 활용되어, 전반적인 실행 효율을 더욱 향상시킵니다.

4. 연산 스케줄링(Op Scheduling)

컴파일러는 **SRAM 활용 효율을 극대화**하면서 **하드웨어의 병렬성**을 최대로 끌어올리는 방향으로 연산을 스케줄링합니다. 이 과정은 전체 연산의 흐름을 결정짓는 핵심 단계로, 각 연산이 언제 어떤 리소스를 사용할지 세밀하게 조정됩니다.

5. 버퍼화(Bufferization)

초기 컴파일 단계에서 모델은 추상적인 텐서 연산으로 표현되어 있어 메모리 주소(DRAM/SRAM)가 명시되지 않습니다. **버퍼화(Bufferization)** 는 이러한 추상적 연산을 **실제 메모리 버퍼(연속적인 메모리 블록)** 로 변환하여, 데이터가 물리적으로 저장·접근·재활용되는 과정을 최적화합니다. 이를 통해 **SRAM 중복 사용을 줄이고**, 하드웨어 친화적인 코드 생성을 가능하게 합니다.

6. 메모리 할당(Memory Allocation)

컴파일러는 각 버퍼의 **수명(lifetime)** 에 따라 SRAM을 동적으로 할당합니다. 즉, 필요한 시점에만 메모리를 점유하도록 관리하여 **연산과 메모리 작업을 병렬화**할 수 있습니다. 이로써 일부 하드웨어가 연산을 수행하는 동안, 다른 영역에서는 다음 연산에 필요한 데이터를 동시에 불러오거나 저장할 수 있습니다. 결과적으로 전체 처리 속도가 향상되고 하드웨어 자원이 효율적으로 사용되어 불필요한 지연이 방지됩니다.

7. 의존성 분석(Dependency Analysis)

컴파일러는 각 연산이 접근하는 **DRAM/SRAM 공간과 연산 간의 의존 관계**를 분석합니다. 이 과정은 올바른 실행 순서를 보장하고, 가능한 한 많은 연산을 병렬화할 수 있도록 설계됩니다. 의존성을 명확히 파악하면 **데이터 충돌 없이 병렬 실행을 극대화**할 수 있습니다.

8. 명령 스케줄링(Command Scheduling)

RBLN Compiler는 **명령어 실행 순서**를 결정하는 중요한 역할을 합니다. 이 단계의 목표는 **메모리 접근과 연산의 병렬성을 최대화하면서도 메모리 의존성을 유지**하는 것입니다. 이로써 시스템은 최대 실행 효율을 확보합니다.

9. 코드 생성

마지막으로, 컴파일러는 커맨드 프로세서**(Command Processor)**용 **최적화된 머신 코드**를 생성합니다. 이 프로세서는 전체 워크로드의 실행을 관리합니다. 리벨리온 NPU는 **의존성 처리 및 메모리 관리에 최적화된 아키텍처**를 갖추고 있어, 컴파일러가 생성한 코드가 하드웨어 성능을 최대한 활용합니다. 또한 연산 라이브러리는 뉴럴 엔진용 프로그램 바이너리를 생성하여, **실행 가능한 형태의 파일**을 완성합니다.

결론

RBLN 컴파일러는 리벨리온 ATOM™과 같은 **신경망 처리 전용 하드웨어의 특성**을 최대한 활용하여 AI 모델 추론을 최적화하도록 설계된 고성능 컴파일러입니다. 그래프 최적화, 연산 융합, 타일링, 메모리 할당 등의 기법을 통해 모델 실행의 모든 단계를 세밀하게 조율합니다. 또한 컴파일 타임에서의 의존성 및 메모리 관리 기능을 통해 **하드웨어의 성능을 완전히 활용하는 머신 코드**를 생성함으로써, **빠르고 효율적이며 안정적인 AI 추론**을 실현합니다.