rebellions_

RBLNコンパイラを理解しよう。

10月 01, 2024

AI推論におけるコンパイラ

コンパイラは高水準言語を低水準言語のマシンコードに変換する役割を果たしますが、ディープラーニングモデルの場合は、その重要性がさらに高まります。RBLNコンパイラがAI推論を加速化する過程で、どのような役割をするかを理解するためには、まずAIコンパイラ独自の特性を知っておく必要があります。

グローバルデータに基づく最適化

AIコンパイラは推論過程でモデル全体の計算グラフとデータ依存関係を分析し、全体的な観点で実行効率を最大化する最適化を行います。ここには、スケジューリングやメモリ割り当て、並列実行といった戦略が含まれます。

その結果として、コンパイラはレイテンシ（latency）を最小化し、スループット（throughput）を最大化できます。また、リベリオンのATOM™のようなハードウェアのリソースを、最適なレベルで活用できるよう高度に最適化されたコードを生成します。

コンパイル時のメモリおよび依存関係の管理

AI推論過程でメモリ割り当てやキャッシュ（SRAM）の活用、演算間の依存関係の管理は、**コンパイルの段階で**行います。これにより、リソースの分配やデータフローを密に制御できます。

既存のコンパイラがメモリを**ランタイムで動的に割り当て**、キャッシュおよび依存関係の管理を**ハードウェアに任せる**のに対し、AIコンパイラは**モデル全体の計算グラフを基にメモリを事前に割り当てSRAM内のデータ配置を最適化**します。この方式は、全てのメモリおよび依存関係の要素をコンパイルで高度に統合するため、**レイテンシ（latency）の最小化**と効率性の最大化が期待できます。また、個別のAIモデルを実行する際にその特徴に合わせて最適化した性能を提供します。

RBLNコンパイラ

最適化の目標

AI専用コンパイラの主な目標は以下の2点です。

1. **メモリへの直接転送（DMA）を最小化しつつ、演算を迅速に行う。**
2. **演算とメモリの作業を並列化・最大化する。**

DMAはDRAMとSRAM間でデータを高速に転送し、演算ユニットを最大で活用できるようにデータフローを安定化します。これと同時に、ATOM™のニューラルエンジン（Neural Engine）のようなマルチコア上で演算作業を並列で行うため、ニューラルネットワークのそれぞれの区間を同時に処理できます。

DMAと演算作業を賢くスケジューリングすれば、アイドル時間が最小化できるうえレイテンシが短縮でき、スループットが向上します。これにより、データがボトルネックにならずに持続的に供給できます。特に、リアルタイム・ローレイテンシの応用環境でAIモデルの性能を最大化できます。

コンパイルの段階

1. グラフの最適化およびオペレーション融合（Op Fusion）

AIモデルはノードとエッジで構成された計算グラフ（computational graph）で表します。コンパイラは、**共通部分式の削除（CSE）、不要なコードの削除（DCE）**などの方法でグラフを簡素化し実行時間を短縮します。

また、**Op Fusion**を通じて複数の演算を一つの演算に統合し、**テンソルおよびベクトル演算の並列実行ができます。**さらに、共有メモリやニューラルエンジンのスクラッチパッド（Scratch Pad）へのアクセス回数を減らします。

2. 演算分割（Op Splitting）およびグループ化（Grouping）

一部の演算は単体SRAMの容量を超えるほど大きいため、効率よく処理するには**小さい単位で分割（Splitting）**する必要があります。

この際に、隣接する演算間の関係を考慮して**最適な実行順序に再配置（Grouping）**し、不要なデータの移動は最小限に抑えます。

これにより、ハードウェアの稼働率を最大化し、全体の実行効率を高めます。

3. 演算タイリング（Op Tiling）

ニューラルエンジンの内部演算は**タイリング**を通じてさらに最適化します。このタイリン

グとは、演算を多数のニューラルエンジンに適切に分割し、並列性を強化する過程を指します。リベリオンの**演算ライブラリ（Compute Library**）は、演算タイプ（例えば、行列乗算、活性化関数）とテンソルの形に沿って**RBLN RISC ISA基盤のカスタマイズプログラム**を生成します。これにより演算ライブラリは、**タイリングの方法や必要なSRAMのサイズ、演算の予想時間**など細かい演算パラメータを決めて、精密かつ効率的な計算ができます。この演算ライブラリから算出された情報は、その後の**コンパイラのパス**（例えば、SRAMの割り当て、コマンドのスケジューリングなど）に利用され、全体的な実行効率をさらに高めます。

4. 演算スケジューリング（Op Scheduling）

コンパイラは**SRAMの活用効率を最大化**する一方、**ハードウェアの並列性**を最大限に引き上げる方向へと演算をスケジューリングします。これは、演算の全体の流れを決める重要な段階であり、各演算がいつどんなリソースを使うかを細かく調整します。

5. バッファー化（Bufferization）

初期のコンパイル段階でモデルは、抽象的なテンソル演算で表示されており、メモリアドレス（DRAM/SRAM）は明示されていません。バッファ化（**Bufferization**）は、このような抽象的な演算を**実際のメモリバッファ（連続的なメモリブロック）**に変換し、データが物理的にアクセス・保存・再利用される過程を最適化します。その結果として、**SRAMの重複利用を減らし**、ハードウェアに適したコードを生成することができます。

6. メモリの割り当て（Memory Allocation）

コンパイラは各バッファの**寿命（lifetime**）に従ってSRAMを動的に割り当てます。つまり、必要なタイミングだけメモリをシェアするように管理し、**演算とメモリ作業を並列化**します。これで一部のハードウェアが演算を行う間に、別のエリアでは次の演算に必要なデータを同時に取り込んだり、保存したりします。結果として、全体の処理速度が向上しハードウェアのリソースが効率よく使われるため、不要な遅延を防ぐことができます。

7. 依存関係の分析（Dependency Analysis）

コンパイラは、各演算がアクセスする**DRAM/SRAMの空間と演算間の依存関係**を分析します。この過程は正しい実行順序を保証し、可能な限り多くの演算を並列化できるように設計されています。依存関係を明確に把握すれば、**データが衝突せずに並列実行を最大化できます。**

8. コマンドスケジューリング（Command Scheduling）

RBLN Compilerは、**コマンドの実行順序**を決める重要な役割を果たします。この段階の目

標は、**メモリへのアクセスと演算の並列性を最大化しながら、メモリ依存関係を保つ**ことです。これにより、システムは最大の実行効率を確保できます。

9. コードの生成（Code Generation）

最後に、コンパイラは**コマンドプロセッサ（Command Processor）向けに最適化したマシンコード**を生成します。このプロセッサは、全体のワークロードの実行を管理します。リベリオンのNPUは、**依存関係の処理およびメモリの管理に最適化したアーキテクチャ**を備えており、コンパイラが生成したコードはハードウェアの性能を最大限に活用できます。また、演算ライブラリはニューラルエンジン向けのプログラムバイナリを生成し、**実行可能な形式のファイル**を完成します。

結論

RBLNのコンパイラは、リベリオンのATOM™と同じく**ニューラルネットワークの処理専用のハードウェアの特性**を最大限に生かし、AIモデルの推論を最適化するように設計された高性能なコンパイラです。グラフの最適化、オペレーション融合（Op Fusion）、タイリング、メモリの割り当てなどの技術を用いて、モデルの実行における全段階を細かく調整します。また、コンパイルタイムにおける依存関係およびメモリの管理機能を通じて、**ハードウェアの性能を最大限に活用できるマシンコード**を生成します。これにより**迅速かつ効率的で安定したAI推論**を実現しています。