論文の内容の要旨

論文題目：分散命令発行マイクロプロセッサの命令ステアリング方式に関する研究

服部直也

半導体実装技術の微細化とアーキテクチャ設計技術の進化により、マイクロプロセッサの性能は指数関数的な向上を続けている。しかし時代と共に高度な情報サービスや研究分野が現れ続けるため、マイクロプロセッサには常に高性能化が要求され続けている。その一方で、半導体実装技術の微細化速度は鈍化し、ムーアの法則の終焉が囁かれ始めている。そのため、アーキテクチャ設計技術の重要度は相対的に増加しており、近年の高性能マイクロプロセッサはパイプライン段数を増やして一段あたりの処理量を減らし、動作クロックを向上させている。一般にプログラム中の命令には複雑な依存関係が存在するが、パイプライン段数を増やすと各処理に要するサイクル数が増加するため、IPC が低下する。マイクロプロセッサの性能はクロック× IPC で与えられるため、パイプラインを設計する際は IPC の低下を考慮する必要がある。IPC に与える影響は一様ではなく、ALU-to-ALU レイテンシ、Load-to-Use レイテンシ、分岐予測ミスペナルティの順に影響が大きいことが知られている。そのため高クロック設計を考える際には、データ依存のある 2 命令の発行間隔である ALU-to-ALU レイテンシの増加抑制を第一に考える必要がある。

一般にトランジスタの応答速度は大きさに比例するため、半導体実装技術はマイクロプロセッサの速度向上に寄与してきた。しかし配線遅延はその長さに比例しないことにより、徐々に配線遅延がマイクロプロセッサの主要な遅延になりつつある。そのため長い配線を要する構造に対しては、特に工夫が必要になる。長い配線を用いる構造は多々存在するが、ALU-to-ALU レイテンシに影響する構造として、演算器間でデータを受け渡す データフォワーディング と、発行した命令の情報を待機中の命令に渡す 命令タグフォワーディング が存在し、どちらも問題視されている。

これらに対する工夫として、本研究ではクラスタアーキテクチャに着目した。この方式では少数の演算器間、Issue Queue エントリ間を短い配線で接続した塊(クラスタ)を形成する。このクラスタの内部では高速な情報伝達が可能であり、ALU-to-ALU のレイテンシを低く保つことができる。個々のクラスタは演算資源に乏しいために処理並列度が低くなるが、これに関しては、クラスタ間を別の経路で接続することで高スループットを確保する。

一般にプログラム中に存在するデータ依存関係は一様ではないことが知られている。これを利用して、データ依存が厳しく並列度に乏しい部分を 1 クラスタ内で、並列度が豊富な部分を複数クラスタを用いて実行できれば、この不均質なハードウェアを活かすことができ、高い性能を発揮できる。つまり、クラスタ化されたアーキテクチャでは、命令をクラスタに割り当てる Steering 次第で、性能が良くも悪くもなる。本研究ではこの命令 Steering に関して、高クロック動作を妨げないことを前提に、高い IPC を引き出す方式を提案する。

本研究ではまず、クラスタアーキテクチャから最大の IPC を引き出す Steering 方式を検討した。仮にプログラム中の全ての命令が等しい重要度を持っているとすると、全ての命令を等しく速やかに処理しなければならない。一般に最高 IPC が得られる方式を議論することは難しいが、命令発行機構等はこの仮定の下で設計されることが多い。この仮定の下では全ての命令は、最も早く発行可能なクラスタに Steering されるべきであり、その場合に最大の IPC が得られる。この Steering を Ultimate Steering と呼ぶことにする。Ultimate Steering は任意の命令の演算レイテンシが既知であれば、命令をそれぞれのクラスタに割り当てた場合の発行時刻を計算することで実現可能である。

クラスタアーキテクチャは、非クラスタアーキテクチャの ALU-to-ALU レイテンシを部分的に高速化したアーキテクチャである。命令 Steering の品質次第で部分高速化の恩恵を受ける量が決定し、クラスタアーキテクチャの見かけの ALU-to-ALU レイテンシをという形で現れる。本研究ではクラスタ内の ALU-to-ALU レイテンシを 1、クラスタ間の ALU-to-ALU レイテンシを 3、クラスタ並列度を 1、クラスタ数を 8 と設定したため、ランダムな命令 Steering を行うと平均レイテンシは 2.75 になる。これに対して Ultimate Steering を用いることで、平均レイテンシを 1.60 まで軽減できることが確認できた。

しかしながら、Ultimate Steering を実現するためのハードウェアを検討したところ、(1) 各命令の Steering が直前の命令の Steering 結果に依存するため、N 命令／サイクルの Steering スループットを確保するためには N 倍の遅延が生じる。(2) Select 時刻を推定するために用いる履歴テーブルが大きいため、配線遅延の影響を受けて長い遅延が発生する。という 2 つの問題点のために、高クロック動作が期待できない。ここで仮に、2 つの問題を解決した近似 Steering を用いたとしても ALU-to-ALU レイテンシの上限は 3 であるため、IPC が半分になることはない。そこで以降の議論では可能な限り品質を劣化させずに Ultimate Steering の問題を克服する方式を検討する。

命令 Steering に関する関連研究を調査した結果、Steering 先の選択肢として、(1) デ

ータを生成する命令が存在するクラスタと (2) 命令数が最低のクラスタ、のみに着目している。そこで Ultimate Steering にもこの近似を適用したところ IPC の低下は 1% 程度であり、この近似が非常に良いことが分かった。

Steering のスループットを確保するためには、同サイクルに処理する N 命令の Steering 処理間に依存関係があってはならないが、実際には先行命令とのデータ依存や、最低命令数クラスタに関する依存が存在する。これらの依存を無視すると Steering の品質が顕著に低下する。これを回避するために、Steering の前にデータ依存のある命令 (Producer) を更にその Producer に差し替える方式を、また最低命令数クラスタを複数選択することで、1 クラスタへの命令集中を避ける方式を検討した。

また、データ生成クラスタと最低命令数クラスタの選択基準について、関連研究 Parcerisa 方式と Palacharla 方式をベースにそれぞれ議論した。その際、Steering が高クロックで動作することを保障するために (a) 特に条件を加えない場合、(b) 直接 Steering に関与する回路の遅延を、関連研究 Dependence Based Steering と比べて増加させない(高クロック動作条件)場合、の 2 つのシナリオに関して議論した。

Parcerisa 方式では、各クラスタの命令数の均衡状態に基づいて、特定クラスタへの命令集中が認められれば最低負荷クラスタへ、認められなければデータ生成クラスタへ Steering する。この選択基準を簡略化された Ultimate Steering に適用した Uni-Policy Steering の性能を詳細に調査したところ、ALU-to-ALU レイテンシの大部分は、最低命令数クラスタに Steering された命令に対するクラスタ間のフォワーディング遅延であることが判明した。これは特定クラスタへの命令集中が起きている場合でも、データ生成クラスタに Steering すべき命令が存在することを意味する。これを受けて本研究では、2 つの方式を提案した。Multi-Policy Steering では、より多くの命令をデータ生成クラスタへ割り当てるために、各クラスタごとに命令数の均衡状態を判断する。Criticality Based Steering では、プログラム実行時間に対して重要度の高い命令を選別して、それらを常にデータ生成クラスタへ割り当て、重要度の低い命令を負荷の低いクラスタで実行する。また、両手法を Ultimate Steering に搭載して評価を行い、IPC が向上することを確認し、両手法を併用することで更に性能が向上することを確認した。Multi-Policy Steering は「高クロック動作条件」を満たさないため、この条件下では Criticality Based Steering しか適用できないが、それでも併用した場合と比べて大きな IPC 低下は起こらないことを確認した。

Palacharla 方式は、In-Order Issue Queue を搭載する非クラスタアーキテクチャ向けの命令 Steering 方式である。この方式は Issue Queue のエントリに着目しており、デー

タ生成命令とデータ使用命令のエントリ間距離が 1 になる場合に限り、データ生成クラスタへ命令を Steering する。この条件を満たすクラスタが存在しない場合は、Issue Queue 内の命令数が 0 であるクラスタへの Steering を試み、それが無い場合は Stall する。本研究ではまず、この選択基準を他のアーキテクチャに適用させるために 2 つの調査を行った。その調査の結果、Out-of-Order Issue Queue に対しては Stall という判断が適切でないこと、閾値とすべきエントリ間距離は本来 ALU-to-ALU レイテンシ差であるべきことが確認できた。また、これらの変更を加えた Local Distance Steering を用いると、高い IPC が得られることが分かった。しかし、この方式は「高クロック動作条件」を満たしていない。そこで Local Distance 計算の代わりに、データ生成命令とデータ使用命令のフェッチ順に於ける距離（Global Distance）で近似する Global Distance Steering を提案した。評価の結果、この近似による大幅な性能低下は見られないことを確認した。

以上の調査・議論の結果、最も高い IPC が得られるのは Ultimate Steering であることが確認された。また、高スループット条件を満たす方式の中では Local Distance Steering が最も優れていることが分かった。更に、高クロック条件を要求する場合には Criticality Based Steering が最も優れていることが分かった。