2012 年 4 月 3 日 情報システム基盤学修士論文中間発表会発表資料

マルチコア間でのパイプラインリソース貸与を用いた信頼性向上の研究

所属： 高性能コンピューティング学講座 本多・近藤研究室
発表者： 1153009 齋藤翔太
主任指導教員： 近藤正章

1 はじめに

半導体微細化技術の進歩により CPU 1 チップあたりのトランジスタの積載数は年々増加し続け、CPU の高性能化が進んでいる。それとともに微細化によってチップ内配線の断線や熱によるトランジスタ劣化の確率が増加し、永久的なハードウェア故障が増加する問題が深刻化している。現在の予測では、今後プロセッサに含まれる数十億ものトランジスタの多くが製造時に使用不能となるとも言われている [1]。トランジスタが故障すると、それに関連するリソースを利用することができなくなり、処理を続けられなくなる。そこで、故障が発生した場合においても処理を続けられる仕組みが重要となる。

既存の研究ではコア単位の冗長化や、故障したコアを利用しないなどの対応が主としてとられてきた。しかし、コア単位の粒度の場合、コアの内部でまだ使用可能なリソースまでも利用できなくなってしまう。そこで本研究では、まだ利用できるリソースを有効に活用し、性能を維持しつつ信頼性を向上させるために、より細粒度なパイプラインステージ単位で故障に対応可能なアーキテクチャを提案する。これにより、今まで故障発生時には利用できなかったリソースも活用でき、スループットの向上が期待できる。

2 背景

パイプラインステージ単位で冗長化を行った研究として StageNet[2] がある。これは、あるパイプラインステージが利用できなくなった場合に他のコアのパイプラインステージを利用できるようにし、利用できるステージを組み合わせることで新しくコアを形成し、命令を処理できるようにするものである。StageNet は静的に利用可能なステージを組み合わせることを主としており、一つのステージを共有することについてあまり考えられてない。そのため、各ステージのさらなる有効利用には改善の余地がある。

本研究でも同じくパイプラインステージ単位の冗長化を検討するが、パイプライン処理時におけるステージの故障や命令のストールに対しパイプラインステージを動的に貸与しながら対処することを目標とする点が StageNet とは異なる。パイプラインステージをプロセッサ間で融通しながら命令を処理することで、チップ内のリソースを有効活用しつつ、信頼性の向上を狙う点が本研究の特徴である。

3 提案手法

3.1 概要

本研究では、パイプラインリソースの貸与を行いつつ命令処理を行うために、図 1 のようなアーキテクチャを考える。

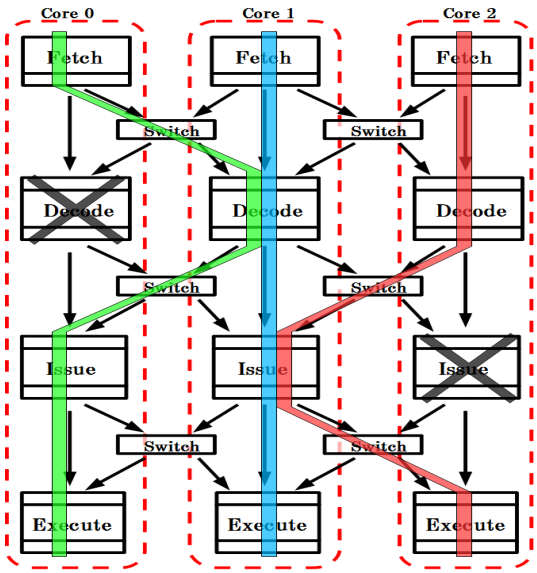

図 1: 提案手法の概要図

図 1 では 3 コアを持つマルチプロセッサを例に、それぞれのコアのパイプラインのうち Fetch から Execute ステージまでを示す。この時、Core 0 の Decode および Core 2 の Issue ステージが故障で使えない場合を考える。本来であれば、Core 0 および Core 2 ではパイプライン処理を行うことができない。

そこで、本研究では Core 0 および Core 2 の故障したステージの処理を他のコアのパイプラインリソースを用いることで処理を続けられるようにする。

既存研究ではそれぞれの利用可能なパイプラインステージを組み合わせることで命令を処理することに主眼を置いていた。本研究では、図 1 で Core 1 がプログラムを実行中でも、Core 0 が Core 1 の Decode ステージを利用

するなど、他のコアでプログラムが実行している場合でもリソースを貸与できるようにすることで、効率的な資源利用を目指す。

特に、あるコアのパイプラインがストールしてしまった場合に発生する各ステージのアイドル時間を利用して、故障したステージの処理を代替することなどを考えている。

3.2 ハードウェアの拡張

提案手法を実現するために必要なハードウェアとして新たにスイッチ、キュー、ストール検知器を追加する。

- **スイッチ：** 各コアのステージ間の接続を行う。
- **キュー** (Buffer)**：** ステージ間で命令を一時的に保持する。スイッチの一部として実装。
- **ストール検知：** ストール発生を検知し、スイッチに検知内容を伝達する。

3.3 リソーススケジューリング

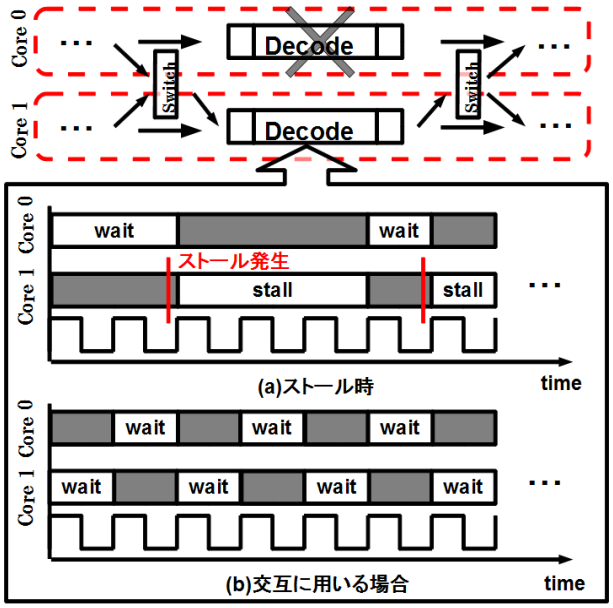

図 2: Decode リソースを利用する命令

本手法では、1 つのパイプラインステージが複数のコアで共有されるため、どちらのコアの命令をいつ実行するかのスケジューリングを行う必要がある。この際のスケジューリングとして次の二つを考える。なお、例として図 2 のような Core 0、1 のどちらも Core 1 の Decode ステージを利用する場合を考える。

(a) **ストール時**

Core 1 がステージを利用していた時にストールが発生した場合はすぐさま処理する命令を Core 0 のものに切り替える。その後、Core 1 のストール原因が解消するとまた Core 1 の命令を処理し始める。

(b) **交互に利用**

Core 0, 1 両方のプログラムに対し、交互に処理を行っていく。命令処理の公平性を保ちつつ、スループットを向上できると考えられる。

4 進捗状況

これまで、プロセッサのアーキテクチャの理解やパイプライン処理など、CPU の動作に関して知識を獲得してきた。それとともに、CPU シミュレータを用いてアーキテクチャにおける命令の動きについて調査した。また、信頼性に関する研究論文を精読し、ハードウェアの様々なレベルで技術の調査を行ってきた。

現在はパイプラインに対する信頼性に関する論文を読むほか、実際に研究を進めていくうえで用いるシミュレータを決定するための調査も引き続き行っている。

5 今後の方針

本手法を実現するために、アーキテクチャの詳細を詰めていく。また、実験および評価を行っていくうえで必要となるシミュレータを決定し、提案手法の実装を行っていく。

パイプライン中に命令がストールするような場合以外にも本手法を有効に活用できる条件の調査も行っていく。

6 おわりに

本研究ではパイプラインリソースを融通しあうことで全体の性能を維持しつつ、故障にも対応可能なアーキテクチャを検討していく。現在の課題としてハードウェアをどのように拡張するか、またステージの利用を各コアにどう分配するのかについて考慮していく必要がある。また、今度の展望としてハードエラーのような永久的な故障だけでなく、トランジスタの生産時における性能のばらつきを考慮する。たとえば、プログラムが動いていないコアの空きリソースを使用した方がスループットが高くなれば、そちらを使うようにすることも考えている。

参考文献

[1] S. Borkar. : “Designing reliable systems from unreliable components: The challenges of transistor variability and degradation”, IEEE Micro , 25(6):pp.10-16 , 2005.

[2] Gupta, S. , Shuguang Feng , Ansari, A. , Blome,J. , Mahlke, S. : “The StageNet fabric for constructing resilient multicore systems ”, MICRO-41 2008 41st IEEE/ACM International Symposium on Microarchitecture: pp.141-151, 2008.