2011 年 6 月 27 日 情報システム基盤学合同輪講発表資料

Architectural Core Salvaging in a Multi-Core Processor for Hard-Error Tolerance

著者： Michael D. Powell, Arijit Biswas, Shantanu Gupta, and Shubhendu S. Mukherjee
出典： *Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09) , pp.93-104.*
発表者： 本多・近藤研究室 1153009 齋藤翔太

1 はじめに

現在のマルチコア CPU はプロセッサのダイサイズを大きくすることで多数のコアを集積している。マルチコア CPU のチップ面積のうち大部分を費やしているのがキャッシュであり、既存の技術を用いて欠陥から保護できる。しかし、それら技術ではチップの残りの大部分である CPU コアを欠陥から保護することはできない。

この論文では、ハードウェアを用いて欠陥のあるコアも利用しスレッドを処理させ、当該コアを停止するよりもスループットを高める。また、スレッドがコア内の欠陥のある部分を使う場合は他のコアにマイグレートさせて実行させる Architectural Core Salvaging を提案している。

2 関連研究

CPU コアを欠陥から保護する手法としてハードエラーがあっても冗長性のある構造 (例えばスーパースカラプロセッサの複数演算器など) を利用し、コアの機能を維持する手法 [1] があり、ここでは Microarchitectural Core Salvaging と呼ぶ。しかし、これは導入が複雑であり冗長性のある構造自体が少ないためにハードエラーに対してコア領域の限られた範囲しか保護できないという問題がある。

実際に冗長化できる領域は、Intel Core-2 のようなコアではキャッシュおよび TLB array を除いたコアエリアのうち 10.4%のみである。

3 提案手法

この論文で提案する手法は Architectural Core Salvaging および Hybrid Core Salvaging である。前者はコア間の冗長化を利用する手法であり、後者は Architectural Core Salvaging と Microarchitectural Core Salvaging のハイブリッド型である。

3.1 Architectural Core Salvaging

Architectural Core Salvaging はコアの一部に欠陥が生じた場合、あるスレッドがその欠陥部分を使おうとした時、そのスレッドを別のコアで動いているスレッドとスワップし、他のコアへとマイグレートする。マイグレートしたスレッドは転送先で処理を実行し、欠陥のあるコアに転送されたスレッドは欠陥部分を使わない限り欠陥のあるコアで処理を実行する。

提案手法を実現するために欠陥部を利用する命令の検出およびコア間のスレッドマイグレーションが必要となる。

3.2 実行できない命令の検出

欠陥のあるコアで命令がデコードされたときにその命令が実行できる命令かどうかをチェックする。もしここで実行できない命令だとわかれば、命令がコミットされる前にスレッドを止め、他のコアへとマイグレートする。

3.3 スレッドマイグレーション

欠陥のあるコアから他のコアへのスレッドの転送は既存の技術を用いて可能である。スレッドを他のコアへとマイグレートするには数 10〜100 サイクルかかってしまう。よって、スレッドマイグレーションの発生をうまく制御し、パフォーマンスの低下を防ぐ必要がある。

マイグレートが多く発生した場合は Core Salvaging を止め、欠陥のあるコアを無効にする。ただし処理する命令が変わる可能性を考え、一定サイクルの後、再び当該コアを使い Core Salvaging する。

コアが有効なときに一定サイクルの間でマイグレート発生を 1 回まで許容する。このサイクル数を X とし、この間隔でマイグレートが 2 回発生した場合はコアを無効にする。コアをある一定サイクルの間無効にした後、コアを有効にする。このときのサイクル数を Y とする。以降ではこのインターバルサイズを Y/X として示す。

また、スレッドをマイグレートするときに他のすべてのコアでスレッドが実行中であった場合、どのスレッドとスワップするかを選ぶ必要がある。以前欠陥のあるコアで実行できない命令をデコードし、マイグレートされたスレッドを記憶しておき他のスレッドがマイグレートするときにそのスレッド以外を選びスワップする。これはマイグレートされたスレッドは再び実行できない命令が含まれている可能性があるためである。

3.4 OS に対する透過性

ハードウェアによりスレッドが他のコアにマイグレートされると OS から見えるスレッド ID が変わってしまう。

そこで、OS から ID が変わることなくアクセスできるようにする。

OS が CPU に割り込み通知を送る場合は Advanced Programmable Interrupt Controller (APIC) を通す。APIC ID number は固定された番号である。これをプログラム可能にすることで実際は違うコアでスレッドが処理されたとしても、同じ ID でスレッドにアクセスできる。

3.5 Hybrid Core Salvaging

Hybrid Core Salvaging は Architectural Core Salvaging の実装に加え、コア内に冗長化を施す。もともとコア内にいくつも存在する小容量のメモリと同様の機能をもち、それよりも少し小さいメモリを新たに追加することで小容量のメモリを冗長化し、機能を維持できるようにする。

4 性能評価

Asim シミュレーション環境で SPEC CPU 2000 (int および fp)、SPEC CPU 2006、multimedia、server を用いて 8 つ同じスレッドを動作させた。Core Salvaging を使い欠陥のある 1 コアと正常な 7 コアの計 8 コアの場合に対し、正常な 8 コア、正常な 7 コアの場合とでスループットの比較を行った。

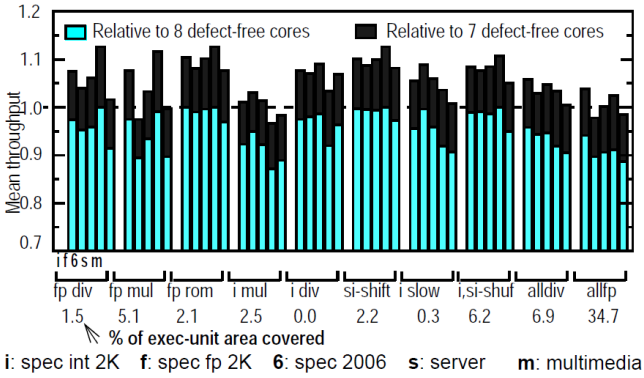

図 1: 正常なコアが 7、8 コアのときと比較したときの Core Salvaging のスループット

図 1 は正常なコアが 8 コア、7 コアのときのスループットをそれぞれ 1.0 としたときの Architectural Core Salvaging を用いて欠陥のあるコアを機能させたものとの比較である (fp div などは欠陥のある演算器を示している)。8 コアのものと比べると、欠陥演算器の場合でスループットは 1.0 よりも低い。しかし、7 コアのものと比べるとほとんどの場合でスループットは高い。しかし、i mul の場合のみ 1.0 程度もしくは低くなっている。よって i mul が使えない場合は Core Salvaging を用いるべきではない。i mul でスループットの低下が見られたのは他の演算よりも多く利用される演算であり、そのぶんマイグレートが起きたからである。

図 2 は alldiv の場合に、インターバルのサイズを変えたときのスループットの比較である。図から特に multimedia の場合、無効状態のコアを有効にし、Core Salvaging 可

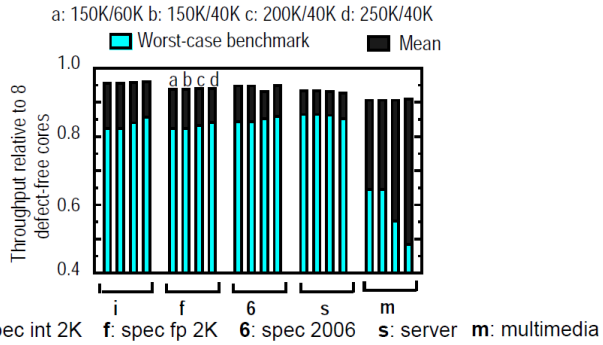

図 2: インターバル変更時のスループット比較 (alldiv)

能な状態に戻すサイクル数が 200K および 250K のときに最悪ケースのスループットが低い。このことから、サイクル数は 150K がよい。

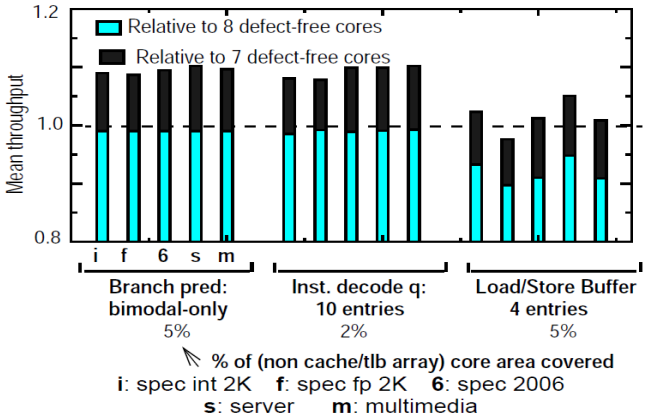

図 3: Hybrid Core salvaging のスループット

分岐予測、デコードキュー、ロードストアバッファの小容量のメモリに対して冗長化を行い Hybrid Core salvaging での結果を図 3 に示す。正常な 8 コアと比較した場合では分岐予測、デコードキューのスループットは 1.0 に近い。また、7 コアのときと比べると 1.0 を超える。また、ロードストアバッファも 1.0 を超えるものが多い。それぞれの小容量のメモリに欠陥があっても機能を維持することができ、パフォーマンスの低下が抑えられたと言える。

5 まとめ

コア間での冗長性を利用した Architectural Core Salvaging を提案した。Architectural Core Salvaging により実行ユニットの 46%をカバーできる。これはコア全体からすると 9%の範囲になる。Hybrid Core Salvaging によりカバーできるコアエリアは 12%であり、合計で 21%の範囲をカバーできる。関連研究による保護範囲も含めると、コアエリアの 30%程度をカバーすることができる。

参考文献

[1] Ethan Schuchman and T. N. Vijaykumar, Rescue: A microarchitecture for testability and defect tolerance, Proceedings of the 32th annual international symposium on Computer architecture, pp. 160-171, 2005