コア融合アーキテクチャのためのプログラムの振舞いに着目した融合コア数の制御

坂口　嘉一 †　若杉　祐太 †　三好　健文 †,‡　吉瀬　謙二 †
東京工業大学 †　科学技術振興機構 CREST ‡

1　はじめに

現在1つのチップに多数のコアを集積するCMP(Chip Multi Processor)の研究が盛んに行われている．CMPの設計には逐次性能と並列性能のバランスという従来のシングルコアプロセッサには存在しなかった問題が存在する．逐次性能を重視した発行幅の広いコアは発行幅の狭いものに比べ広い面積を占め，集積コア数を制限する．一方で発行幅の狭い小さなコアを多数集積した場合には，逐次処理性能の低下が問題となる．

このような問題に対する解として，コア融合アーキテクチャ[1][2]が提案されている．これらのアーキテクチャでは必要に応じて複数のコアが融合することで発行幅の広い一つ のコアとして動作することができる．そのため並列性能と逐次性能を調整することで幅広いプログラムに対応することができる．しかし融合は実行スレッド数の減少を伴い，性能を引き出すためにはその適切な制御が必要となる[3]．

本稿ではプログラムの振る舞いが一様ではなく，融合により性能が大きく向上する部分，そうでない部分が存在することに着目する．融合の効果が大きい部分を実行しているプログラムに優先してコアを配分することで有効にコアを活用する手法について評価を行う．

2　コア融合の制御

2.1　対象アーキテクチャ

本稿では，CoreSymphonyアーキテクチャ[1]上での評価を行う．CoreSymphonyアーキテクチャは2命令発行のアウトオブオーダコアをベースに4つまでのコアが協調して動作することができる．高い逐次性能が求められる場合には4つのコアを融合することで8命令発行のコアを構成し，逐次部分の高速な実行を可能にする．

2.2　プログラムの振る舞い

図1にSPEC2006の403.gccをCoreSymphony上で実行したときのIPCの10M命令毎の変化を示す．横軸は命令数で単位は1M命令，縦軸は10M命令区間のIPC

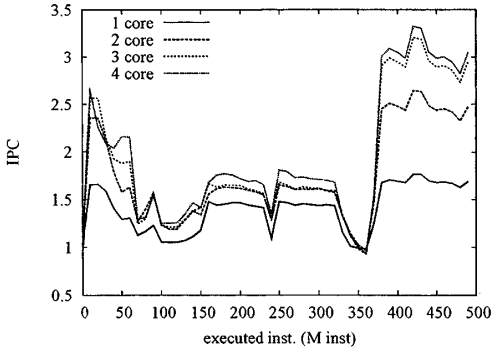

図1: 融合コア数別にgccを実行した場合のIPCの推移．

である．各グラフは融合コア数別の結果を表わしている．先頭1G命令をスキップした後の500M命令を実行した．

図のように150M命令目から350M命令目の間では1コアと4コアとのIPCの差は0.3以内であり，また2コアと3コアの性能差は極めて小さい．一方400M命令目以降では1コアと2コアの差でも約0.8であり，1コアと4コアの差は約1.4にまで広がる．このようにプログラムの実行部分によってコア融合の効果には差が有ることが分かる．

2.3　プログラムの振る舞いの利用

gccにおいて全体を2コアで実行する場合と3コアで実行する場合とではIPCは3コアで実行した方が高い．しかし150M命令目から350M命令目までの区間では2コアで実行しても性能に大きな差が無い．そのためコアを有効に活用できているとは言えない．他のプログラムが走っていれば，そちらに多くコアを割り当てることでプロセッサ全体でのスループットが向上する可能性がある．

以上の理由から，プログラム実行中に最適なコアの割り当てを動的に求め融合を制御することでプロセッサ全体での性能向上が期待できる．

本稿では，動的な割当てによって改善可能な幅を見積もることを目的とする．そのため，コアの割当て決定に事前情報を用いる．あらかじめ各プログラムの10M命令毎のIPCを融合コア数毎に収集しておく．この情報に基づきプロセッサ全体でのスループットが最大になるようコアの融合を制御する．コア割り当ての変更は実行しているプログラムの内の1つが10M命令実行する度

Dynamic core allocation for fusible core architecture focusing on program behavior
Yoshito Sakaguchi†, Yuhta Wakasugi†, Takefumi Miyoshi†‡, and Kenji Kise†
†Tokyo Institute of Technology
‡CREST, Japan Science and Technology Agency

表 1: プロセッサの構成.

Conventional I-cache	8KB, 2way, 1cycle
Local I-cache	8KB, 4way, 2cycle
D-cache	16KB, 2way, 1cycle
L2-cache	2MB, 4way, 10cycle
Memory Latency	200 cycle
Branch predictor	Bimode, 1k entry
Instruction window	INT 24, FP 24
ROB	80 entry
Rename register	INT 32, FP 32

にスループットが最大となる組み合わせを求める.

3 評価

3.1 評価方法

CoreSymphony を実装した 4 コア，L2 キャッシュ共有型の CMP を想定してシミュレーションを行う．各コアの詳細な構成は表 1 の通りとする．ベンチマークには SPEC2006 の内から 401.bzip2，403.gcc，429.mcf の 3 つを用いる．2 スレッド実行のワークロード 3 種類 (bzip2-gcc，bzip2-gcc，gcc-mcf) および 3 スレッド実行のワークロード 1 種類を構成する．データセットは train を使用し各プログラムの先頭 1G 命令をスキップした後に 500Mcycle のシミュレーションを実行し，各スレッドが実行した命令数を求める.

3.2 評価結果

実行中に構成を変更しない場合（静的割当て）と比較を行う．静的割当てに対する動的割当ての相対性能を図 2 に示す．bzip2 と gcc の組み合わせで 1.6%，bzip2 と gcc と mcf の 3 つを同時に実行したケースでは 3.8%の性能向上が得られた．一方，gcc と mcf の組み合わせでは 0.65%の性能低下がみられた.

3.3 性能低下に関する考察

gcc と mcf の組み合わせにおける 0.65%の性能低下について考察する．図 3 に gcc と mcf の組み合わせにおける 10Mcycle 毎のスループットをプロットしたものを示す．250Mcycle 以前では gcc と mcf をそれぞれ 2 コアで実行してるが，そこから 340Mcycle 付近までの間は gcc に 3 コアが割り当てられる．この間 mcf は 1 コアで実行するため IPC が低下するが gcc の IPC 向上がその量を上回っているため割当てが変更される.

この区間での変更が最終的な実行命令数に与える影響は以下のようになる．gcc が 3 コアで実行していたサイクル数を k_{gcc3}，この間に実行した命令列を I として，この命令列 I を 2 コアで実行するのに必要サイクル数を k_{gcc2} とすると，動的割当てを行った場合 $n = (k_{gcc3} - k_{gcc2})$ cycle 先行することになる．その為，あるサイクルで区切り，それまでに実行した命令数を比較すると，先行しているものが最後の n cycle に実行した命令数が実行命令数の差と

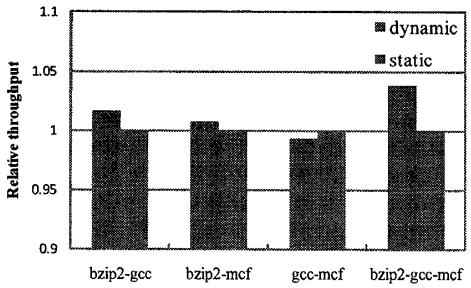

図 2: 動的割当ての静的割当てに対する相対スループット.

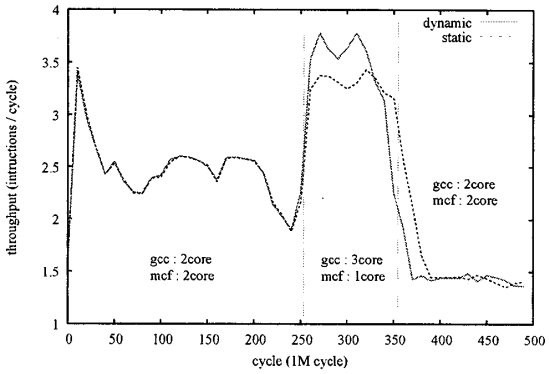

図 3: 静的割当てと動的割当てのスループットの比較.

して現れる．同様に mcf において $m = (k_{mcf2} - k_{mcf1})$ cycle だけ遅れ，先行している静的な割当てが最後の m cycle に実行する命令数が，動的な割当てを行った場合の命令数の減少となる．このことから局所的なスループットを最大化する方法では，単位時間に処理できる命令数を増やすことが困難な場合が存在することが分かる.

4 まとめ

本稿では，コア融合アーキテクチャのチップ全体のスループットを最大化することを目的とし，動的なコア割り当ての検討をおこなった．ある区間ごとのスループットを最大化する局所的なスケジューリングポリシーを用い，動的にコア割り当てを変更したところ，静的にコア割り当てを決定する場合に比べ最大で 3.8 能が向上することを確認した．しかし，局所的なスケジューリングポリシーでは有効な結果が得られない場合があることを確認し，さらなる検討の必要があることが分かった.

参考文献

[1] 若杉他. CoreSymphony アーキテクチャの高効率化. 情報処理学会研究報告 2009-ARC-184 (2009).

[2] Engin .Ipek, et al. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors In *Proc. of ISCA-34*,pp.186-197(2007).

[3] Divya P.Gulati, et al. Multitasking Workload Scheduling on Flexible-Core Chip Multiprocessors *PACT-2008*