1J-2

CoreSymphonyにおける命令ステアリングの検討

上野 貴廣 † 永塚 智之 ‡ 坂口 嘉一 ‡ 吉瀬 謙二 ‡

東京工業大学 工学部情報工学科 † 東京工業大学大学院 情報理工学研究科 ‡

1 はじめに

現在，広く普及しているCMP(Chip Multi-Processor)は，比較的小規模のコアを複数並べてスレッドレベル並列性により性能を向上させるアプローチである．しかし，依然として逐次処理性能を求められる場面もあり，各コアの逐次処理性能の限界が全体の性能向上の妨げになる場合があるという課題を抱えている．

それを克服する試みとして，CMPにおいて複数のコアを必要に応じて融合させ，逐次処理性能の向上を図るアプローチが提案されている．その一つとして「コア間の通信を極力少なくする」というコンセプトで提案されたCoreSymphonyアーキテクチャ[1]がある．本稿ではその中で，コアに命令を割り振る「命令ステアリング」に着目する．CoreSymphonyで用いられている「リーフノードステアリング(以下，LNSと表記)」と呼ばれるステアリングで無駄が生じてしまう箇所を是正し，より効率的で性能を向上させるステアリングを提案する．

2 従来のリーフノードステアリング

CoreSymphonyの命令ステアリングLNSは，コア間通信の抑制とコアに割り振られる命令の分散との両立を実現するアルゴリズムである．CoreSymphonyでは，コア数 × 4 命令または2つの基本ブロックを最大長とする命令トレース(フェッチブロック：以下，FBと表記)を1単位としてフェッチ/ステアリングを行う．コア間通信を発生させないために，単に命令を各コアに割り振るだけでなく，命令の複製すなわちひとつの命令を複数のコアに割り振る場合もあるのがLNSの特徴である．

LNSの動作について述べる．まず，FB内のすべての命令でデータフローグラフを作成する．その中でリーフノードになっている命令を，ラウンドロビンで各コアに割り当てる．次に，コアに割り振られたあるリーフノードが同一FB内で依存している命令全てを同じコアに割り振る．そのため，複数個の命令の依存元となっている命令は複数コアに割り当てられることがある．

図1の例では，FB内に番号の振られた12個の命令がある．この中では6，7，8，10，11，12がリーフノードである．まず最も若い番号のリーフノード6と，6が依存する命令である1，2，3，4，5を含む部分木aが生成される．次にリーフノード7に対して，7と7が依存する1，2，3，4の命令が部分木bに割り振られる．同様にすべてのリーフノードにこの操作を行い，図1から生成された部分木をまとめたものが図2である．また，これらの命令を3コアに割り振った例が図3である．

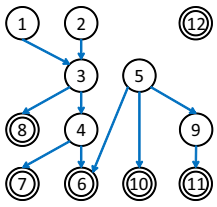

図 1: データフローグラフ.

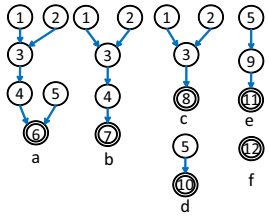

図 2: LNSによる部分木の作成.

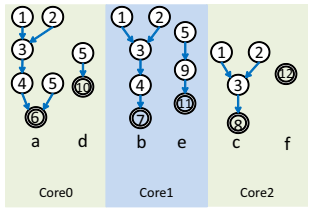

図 3: 3コアに割り振った例.

この例の場合，LNS前後のデータフローグラフを見比べてみると1，2，3，4，5が複数個のリーフノードの依存元になっているため実行する命令が複製されているが，全体としては同程度の負荷になっていることがわかる．

従来のLNSの問題点について述べる．従来のLNSでは，リーフノードを単純にラウンドロビンでコアに割り振っているが，その命令が依存する命令の数すなわち図2で示した部分木の大きさを一切考慮していないため，コアに割り振られる命令数に偏りが発生することがある．そのため，より多くの命令が割り振られるコアがボトルネックとなり，性能を低下させる可能性がある．

命令の割り振りによりどれだけ差が出るかを説明する．なお，説明の簡略化の為，重複する命令は考慮しない．図2の6つの部分木を4コアに割り振る場合を考える．従来手法では，リーフノードを番号が若い順に割り振るだけなのでa，b，c…の順に割り振られ，図4のようになる．LNSはFB単位で処理を行うため，一つのコアに負荷が集中し別のコアが処理を早く終えたとしても，その次のFBの命令は前のFB内の命令に依存している可

Decentralization of Branch Predictor on CoreSymphony Architecture.

Takahiro UENO†, Tomoyuki NAGATSUKA‡, Yoshito SAKAGUCHI‡ Kenji KISE‡

†Depart. of Computer Science, Tokyo Institute of Technology

‡Graduate School of Information Science and Engineering, Tokyo Institute of Technology

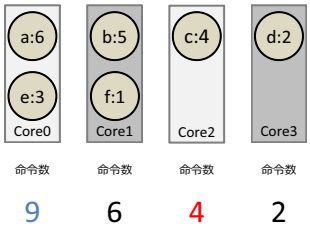

図 4: 従来手法．(部分木:命令数)

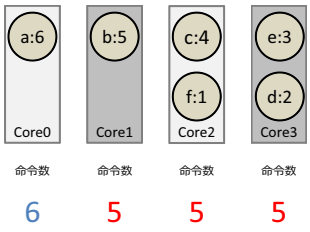

図 5: 最も効率が良い場合．(部分木:命令数)

能性が高く，大きな負荷を持ったコアがボトルネックになってしまう．よって，最も適した割り当てはコアに割り当てられた命令の最大値を最も小さくするようなものである．この例の場合では図 5 のような割り当てが，最も効率のいいパターンである．ただし，実際には同じ命令が 1 つのコアの中で 2 回以上割り振られても二重には実行されないので，厳密には異なる場合がある．

3 提案するリーフノードステアリング

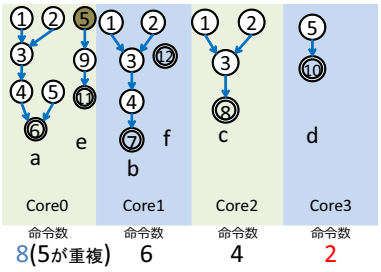

図 6: 重複を考慮した従来手法.

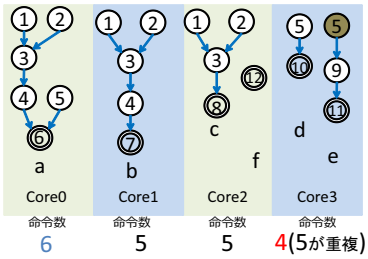

図 7: 重複を考慮した提案手法.

前章で述べたような最適な割り当ての探索は実現が難しい．そのため部分木は従来通り若い番号順に入れるが，「その時点で割り振られた命令数が最も少ないコア」を探索し，そのコアに割り振るよう改良する．その結果，図 2 を 4 コアに割り当て命令の重複を考慮し検討すると，従来手法では図 6 となり，大幅な偏りが見られる．しかし提案手法では図 7 となり，負荷が分散でき，割り振られた命令数の最大値も抑えることができている．

4 性能評価

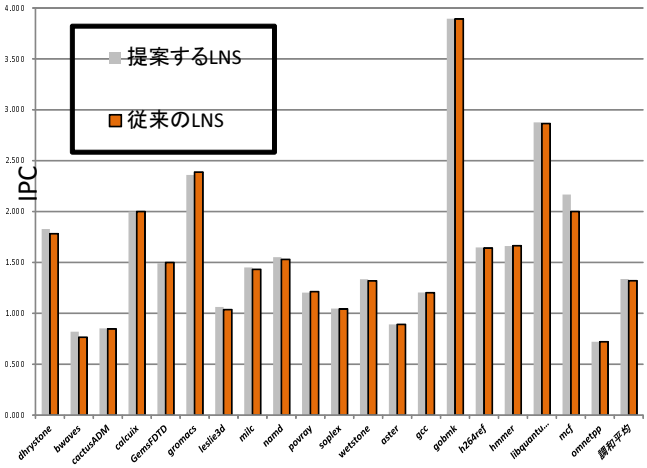

図 8: 評価結果

サイクルレベルシミュレータにより LNS を改良した 4 コア協調時の CoreSymphony アーキテクチャを評価する．評価に用いたベンチマークは dhrystone, wetstone, SPEC2006 FP から bwaves, cactusADM, calculix, GemsFDTD, gromacs, leslie3d, milc, namd, povray, soplex, INT から aster, gcc, gobmk, h264ref, hmmer, libquantum, mcf, omnetpp の 20 種である．基本的には 2G 命令をスキップしそこから 100M 命令を実行して評価しているが命令数が 2G に満たないものはそれよりも前の 100M 命令，dhrystone など命令数の少ないものは全てを実行し，各ベンチマークにおける CoreSymphony の IPC を測定した．その評価結果を図 8 に示す．

全体的に IPC は向上しており，平均では 1.1%，最も向上したもので 8.4%の性能向上となった．また，INT よりも FP を用いた計測の方が性能向上が若干顕著に見られる．これは，FP のベンチマークが INT に比べて 1FB 当たりの平均の命令数が多かったため，従来のステアリングにより生じるロスが多くなる可能性が高く，提案するステアリングにより改善されるところが大きいためと考えられる．

5 まとめ

本稿では，CoreSymphony アーキテクチャの命令ステアリングに着目し，より効率的なステアリングの構成を提案し，評価の結果性能が向上することを確認した．

今回の提案はハードウェアの実装を視野に入れた設計を行なっていないため，ハードウェア量を現実的な値に近づけるための構成が今後の課題として挙げられる．

謝辞

本研究の一部は科学研究費補助金若手研究 (B) 課題番号:22700046 「コア融合機能を持つメニーコアプロセッサに関する研究」による．

参考文献

[1] 若杉祐太, 他．協調可能スーパースカラ CoreSymphony. 情報処理学会論文誌 ACS, Vol.3, No.3, 2010.