2008 年 6 月 9 日 情報システム基盤学専攻合同輪講発表資料

Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-threaded Workloads on CMPs

著者： M. Aater Suleman, Moinuddin K. Qureshi, Yale N. Patt
出典： *Architectural Support for Programming Languages and Operating Systems-XIII, pp. 277-286, 2008*
発表者： 近藤研究室　0853009 久米正人

1 概要

1 つのコアを高性能化することは，設計の難しさや電力の問題などにより困難で，近年ではチップマルチコアプロセッサ (CMP) が主流になってきている．CMP とは，1 つのチップ上に複数のコアが組み込まれたプロセッサである．しかし電力の制約により，内部の 1 つ 1 つのコアは電力効率のよい簡素なものとなってしまうためそれ自体のパフォーマンスは低い．よって 1 つのコアのみでアプリケーションを動作させるよりは複数のコアで並列実行できるほうがパフォーマンスが高まる．そのためには，アプリケーションが複数のスレッドに分かれていることが必要となる．このようなアプリケーションをマルチスレッドアプリケーションという．

マルチスレッドアプリケーションを多くのスレッドに分割し，より多くのコアで並列実行することを考える．スレッドを増やすほど並列性が増し，パフォーマンス向上につながると考えられるが，パフォーマンスが最大となるようなスレッド数はアプリケーションにより異なる．実際には過剰なスレッド数の増加はさまざまな要因によりパフォーマンス向上をもたらさない．しかも，現在のシステムではスレッド数はコア数と静的に決定されてしまう．本論文では，パフォーマンス低下の主要な 2 つの原因を考慮したスレッド数の最適化を実現するために実行時情報を集めてスレッド数を動的に制御する手法である *Feedback-Driven Threading* (FDT) を提案する．

2 パフォーマンス低下の主な原因

パフォーマンス低下の主な原因の 1 つに，データ同期 (クリティカルセクション) に関する原因が挙げられる．各スレッドは，データの同期のために必ずクリティカルセクションで一定の時間を費やす．ここで，あるスレッド A がクリティカルセクション実行中に他のスレッド B もクリティカルセクションを実行する場合について考える．クリティカルセクションとは，複数のスレッドが同時に実行できない部分のことである．このときスレッド B はクリティカルセクションを実行できず，待つこととなる．この時間が長くなってしまうと，時間の浪費や電力の無駄にもつながる．そのため，スレッド数がいたずらに増えるとかえって実行時間が長くなり，パフォーマンスが悪化する．

外部バス帯域幅の限界もパフォーマンスを低下させる原因である．各スレッドは計算結果の更新のためのメモリアクセスなど，外部バスアクセスを行う．この外部バス帯域幅は無限ではなく，外部バスアクセスできるスレッド数には限界がある．それ以上アクセスできなくなったスレッドは，待つこととなる．この時間が長くなってしまうと，時間の浪費や電力の無駄にもつながる．

3 Feedback-Driven Threading

実際に FDT が行う手順は以下のようになる．

1. 対象となる並列実行可能部分をシングルスレッドで少し動かし，その間に情報を得る
2. その情報をもとに，スレッド数を決定する
3. 残りの部分を決定したスレッド数で並列実行する

4 原因を考慮したスレッド数の最適化

4.1 SAT, BAT

パフォーマンス低下の主な 2 つの原因に対して改善手法をそれぞれ考える．

Synchronisation-Aware Threading (SAT) は，データ同期 (クリティカルセクション) に関する手法である．クリティカルセクションでの実行時間を T_{CS}，それ以外での実行時間を T_{NoCS} とする．アプリケーションを P スレッドで並列実行したときにかかる実行時間 T_P は

$$T_P = \frac{T_{NoCS}}{P} + P \cdot T_{CS} \tag{1}$$

となり，T_P を最小化する $P(= P_{CS})$ は

$$P_{CS} = \sqrt{\frac{T_{NoCS}}{T_{CS}}} \tag{2}$$

となる．P_{CS} を求めるために FDT を用いて T_{NoCS}, T_{CS} の情報を得る．具体的には，クリティカルセクション前後でサイクルカウンタを読み T_{CS} を得る．次にイタレーション全体の前後でサイクルカウンタを読み，そこから T_{CS} を差し引くことにより T_{NoCS} を得る．この作業を Training という．Training は T_{CS} と T_{NoCS} の比が連続して 3 回のイタレーションで一定になるか，総イタレーション数の 1%を終えるまで続く．

Bandwidth-Aware Threading (BAT) は，外部バス帯域幅に関する手法である．シングルスレッドでのバスアクセス時間の割合を BU_1 とする．アプリケーションを P ス

レッドで並列実行したときに実行時間に対するバスアクセス時間の割合 BU_P は

$$BU_P = P \cdot BU_1 \tag{3}$$

となり，BU_P を 100 にする $P(= P_{BW})$ は

$$P_{BW} = \frac{100}{BU_1} \tag{4}$$

となる．P_{BW} を求めるために FDT を用いて BU_1 の情報を得る．具体的には，外部バスが使われたサイクルと全体の実行サイクルの商をとることにより BU_1 を得る．この Training は 10,000 サイクル計測して，各スレッドの平均バスアクセスと全コア数の積が 100 に満たないか，総イタレーション数の 1%を終えるまで続く．

それぞれにおいて最適スレッド数を求められ，その最適スレッド数で残りのループ部分を並列実行する．本研究においてスレッド数の変更は，`OpenMP`[1] の *num_threads* を用いて実現した．

4.2 SAT, BAT の複合

SAT, BAT は互いに複合させることもできる．そのためには，両者の実現に必要な情報を同時に FDT に収集させ，両者の Training が終わるまで Training を続ける．その後 SAT, BAT がそれぞれ予測したスレッド数とシステムのコア数の 3 種類の値の最小値をとり，実際に実行させるスレッド数として残りのループ部分を実行する．

5 評価

5.1 使用環境

評価に使用した CMP の仮定を表 1 に示す．

System	32-core CMP(shared 8-MB L3 cache)
Core	In-order, 2-wide, 5-stage pipeline, 4-KB gshare, 8-KB write-through private I and D cache, 64-KB 4-way associative inclusive private L2 cache

表 1: 評価に使用した CMP の仮定

5.2 SAT

データ同期に影響を受けるアプリケーションを 4 種類，評価に使用した．`PageMine`（テキストのデータマイニング）[2]，`ISort`（整数ソート）[3]，`GSearch`（有効グラフ探索）[4]，`EP`（線形合同法による疑似乱数生成）[3] のいずれにおいても最も高速になるスレッド数をほぼ正しく予測できた．

5.3 BAT

外部バス帯域幅に影響を受けるアプリケーションを 4 種類，評価に使用した．`ED`（ユークリッド距離計算），`convert`（画像処理）[5]，`Transpose`（転置行列計算）[5]，`MTwister`（メルセンヌツイスタ）[5] のいずれにおいても最も高速で，省電力になるスレッド数をほぼ正しく予測できた．

5.4 SAT+BAT

5.4.1 使用アプリケーション

SAT, BAT の評価で使用したアプリケーション 8 種に加え，どちらの影響もあまり受けないアプリケーションを `BT`（流体力学）[3]，`MG`（マルチグリッド法）[3]，`BScholes`（ブラック・ショールズ方程式）[5]，`SConv`（コンボリューション）[5] の 4 種類，評価に使用した．

5.4.2 32 スレッド固定での実行との比較

データ同期によって影響を受けるアプリケーションについてはスレッド数 32 と比べて非常に素晴らしい結果が実行時間，電力両方において得られた．外部バス帯域幅によって影響を受けるアプリケーションについては，電力において素晴らしい結果が得られた．どちらの影響もあまり受けないアプリケーションについては，32 スレッドと同じくらいの結果が得られた．これは，SAT, BAT があまり活躍できなかったとしてもそれら自体があまり負担にならないということを意味する．また平均すると，実行時間は 17%，電力は 59%も削減できることが示された．

6 まとめ

実行時情報によってスレッド数を動的に制御する手法である FDT を提案した．この FDT を用いてデータ同期 (クリティカルセクション) が実行時間に及ぼす影響を解析し，スレッド数を予測する SAT を実装した．また，外部バスが飽和したときに実行時間に及ぼされる影響を解析し，スレッド数を予測する BAT も FDT を用いて実装した．さらに，FDT を用いて SAT, BAT を複合し，12 のアプリケーションで平均して実行時間を 17%，電力を 59%削減できることを示した．

参考文献

[1] L. Dagum and R. Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 1998.

[2] R. Narayanen et al. MineBench: A Benchmark Suite for Data Mining Workloads. In *IISWC,* 2006.

[3] D. Bailey et al. NAS parallel benchmarks. Technical report, NASA, 1994.

[4] A. J. Dorta et al. The openmp source code repository. In *Euromicro Conference on Parallel, Distributed and Network-Based Processing,* 2005.

[5] Nvidia. CUDA SDK Code Samples. http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html, 2007.