5L-8

コード差分実行に基づいた時短シミュレーションの有効性

椎名 敦之 † 大津 金光 † 大川 猛 † 横田 隆史 †
† 宇都宮大学大学院工学研究科情報システム科学専攻

1 はじめに

並列実行をはじめとした高性能な計算機アーキテクチャの登場に伴い，その上で実行するプログラムコードを改善しプロセッサの性能を最大限活用するためのソフトウェア最適化手法の需要が高まってきている．新規アーキテクチャにおけるソフトウェア最適化の開発手法に，サイクルベースの詳細なシミュレータにより複数の最適化手法を適用したプログラムコードを順次実行しその結果を比較評価するものがある．しかしサイクルベースの詳細なシミュレーションは実際のプロセッサで実行した場合に比べて時間がかかり，繰り返しのシミュレーションによりソフトウェア最適化手法の評価時間が膨大になるという問題が生じる．

そこで我々はこの問題を解決するために，シミュレーション時間を短縮する評価手法である ISS (Incremental Software Simulation)[1] を提案した．提案手法では，過去に実行されたシミュレーションの情報を再利用し，評価に必要な部分のコードのみの部分的なシミュレーションを行う．評価に必要のない部分のシミュレーションを省略することで総評価時間の短縮を図る．

本稿ではまず，ISS の概要と既存の問題点を示し，その改善案について述べる．続いて，実際の評価に提案手法を適用した場合の効果について論じる．

2 ISS

図 1 は，SPEC CPU2000 ベンチマークプログラムをスレッドパイプライニングモデルのシミュレータ SIMCA で実行した際のシミュレーション時間をまとめたものである．大半のプログラムが 1 回のシミュレーションに数時間以上のシミュレーション時間を要し，長いものでは 10 時間以上かかるものがある．入力データセット等によっては，1 回の実行に数日を要するものも少なくない．

ソフトウェア最適化手法の開発評価では，実行割合が高い特定のループ等の特定のプログラムコード部分に対して複数の候補となる最適化手法を適用し，それぞれの実行結果を比較していく．ソフトウェア最適化手法にはループアンローリングや命令のスケジューリング等のそれらの組み合わせも含めて様々なものが存在し，その中から最も実行性能が向上するものを探求する．本稿では以後，この最適化され比較評価の対象となる部分を評価対象区間と呼ぶ．

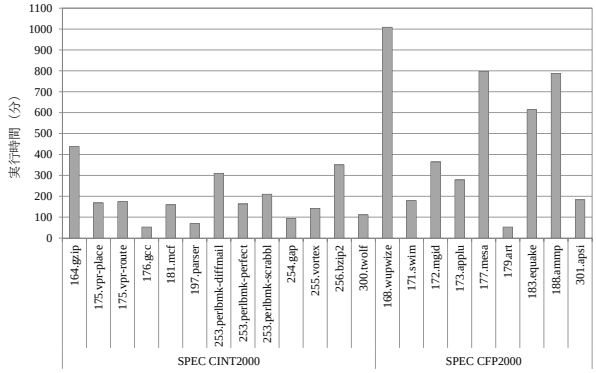

図 1: SPEC CPU2000 のシミュレーション時間 (train データセット)

我々が提案したシミュレーション時間短縮手法 ISS は，評価対象区間のみを実行することで評価時間の短縮を図る．ソフトウェア最適化手法の評価に必要な情報は，プログラム全体の実行結果ではなく，評価対象区間の結果のみである．したがって，評価対象区間のみを実行することができれば，それ以外のコードを実行する必要がなくなり，シミュレーション時間の大幅な短縮が可能になる．

ISS では，過去のシミュレーションプロセスをチェックポインティングにより記録しておき，2 回目以降のシミュレーションでは過去に記録されたシミュレータの状態を再利用することで部分的なシミュレーションを実現する．ISS では初回のシミュレーションか 2 回目以降のシミュレーションかにより評価手順が異なる．

- 初回のシミュレーション
  通常のシミュレーションと同様に，プログラムコード全体を実行する．評価対象区間が実行される直前にチェックポインティングを行いシミュレーション状態をチェックポイントファイルに保存する．
- 2 回目以降のシミュレーション
  チェックポイントファイル内の処理を改変後，初回シミュレーションのプロセスを復元し部分的なシミュレーションを行う（コード差分実行）．

ISS では，初回シミュレーション時にシミュレータのプロセス状態がチェックポインティング処理によってファイルとして記録される．1 回あたりのチェックポインティング処理の時間的オーバーヘッドは全体の時間と比較して極めて小さいため，評価対象区間の数が少ない場合はチェックポインティング回数も少なくなり，ISS の適用によりシミュレーション時間を大き

Effectiveness of Reduction Method of Simulation Time by Using Code Substitution
†Atsushi Shina, Kanemitsu Ootsu, Takeshi Ohkawa and Takashi Yokota
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

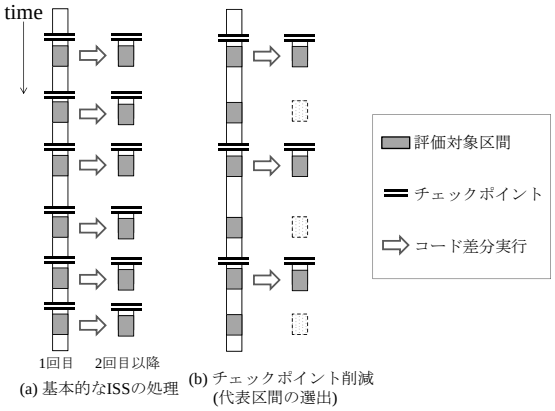

図 2: チェックポインティング数の削減

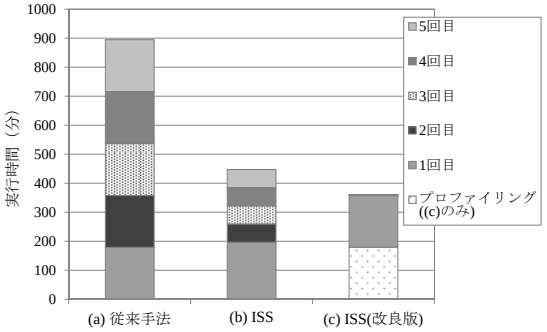

図 3: シミュレーション時間

く短縮可能であった [1]．一方で，評価対象区間の数が多い場合は時間的オーバーヘッドの総量が増加し，初回シミュレーション時間そのものが増大する．

本来であれば図 2(a) のように評価対象区間ごとにチェックポインティングおよびコード差分実行を行うが，チェックポインティング回数を削減することでその処理時間を削減することが可能になる．そこで新たに，図 2(b) に示すような，複数回実行される評価対象区間のうち一部を選出しコード差分実行を行う手法を提案する．それぞれの評価対象区間の実行結果に着目すると，サイクル数等は同様の数値が出現する場合があることがわかる．全ての区間をコード差分実行するのではなく，特定の区間のみを選び，その結果を基に全体の実行結果として評価を行う．SimPoint[2] のように，実行結果が類似している区間ごとに分類し，それぞれを基にして全体の結果を算出する．代表区間の選出の仕方によって本来の結果と若干異なる可能性が生じるが，大幅なオーバーヘッド削減が期待できる．

3 評価

SPEC CFP2000 のプログラムである 171.swim に対し，提案手法を適用した場合の効果について論じる．今回の評価対象区間は，実行頻度の高い関数である関数 calc2_とした．関数 calc2_のサイクル数に基づく実行割合は 31.2%であり，評価対象区間は 200 回実行される．

図 3 に実際の最適化手法の評価にかかった時間を示す．本実験では，評価対象区間に 5 種の最適化手法を適用したプログラムコードを用い，SIMCA で順次実行した．図 3(a) は従来手法による評価を行った場合の実行時間を示しており，5 回分のシミュレーションに 895 分かかった．それに対し，図 3(b) で示す ISS 適用の場合には 448 分となり，総実行時間が 0.50 倍に削減された．図 3(a) と (b) における初回のシミュレーション時間を比較すると，ISS 適用時の方が 1.1 倍程度時間がかかっている．ISS 適用時には評価対象区間ごとにチェックポインティングが行われるため，チェックポインティング 200 回分の処理によって実行時間が増加したと考えられる．一方で，ISS では 2 回目以降のシミュレーションでは評価対象区間のみをコード差分実行するため，従来手法と比較して大きく時間が短縮されていることがわかる．

図 3(c) に，図 2(b) のようにチェックポイント削減案を適用した場合のシミュレーション時間を示す．プロファイリングを行い，200 回実行される評価対象区間においてそれぞれのサイクル数を調査したところ，ほぼ全てが同じ値となっていることが分かった．そのため，改善案では 200 個の区間のうち初回に実行される 1 個をコード差分実行し，結果を 200 倍し算出したものを結果とした．実際の総サイクル数と算出したサイクル値を比較したところ，誤差は 3.15×10^6%であった．チェックポインティング数およびコード差分実行数を削減したことで，より短時間で従来とほぼ同様の結果が得られた．今回の実験では理想的な結果となったが，対象によっては各実行結果がばらつく可能性がある．そのため，実行結果の特徴ごとに区間を分類する必要が生じるが，実行時間の短縮が可能である．

4 おわりに

本稿では，シミュレーション時間短縮手法である ISS の概要と既存の問題点および改善点として性能向上手法を述べ，実際の評価に提案手法を適用した場合の有効性を示した．今後はプログラムの適用例を増やし，同様の解析と評価を進めていく．

謝辞

本研究は，一部日本学術振興会科学研究費補助金 (基盤研究 (C)24500055，同 (C)24500054，同 (C)25330055，若手研究 (B)25730026) の援助による．

参考文献

[1] Atsushi Shina et al.: “Proposal of Incremental Software Simulation for Reduction of Evaluation Time,” Proc. 3rd International Conference on Networking and Computing (ICNC), pp.311-315, 2012.

[2] G. Hamerly et al.: “SimPoint 3.0: Faster and More Flexible Program Analysis,” Journal on Instruction-Level Parallelism (JILP), 2005.