バイナリレベルでの実行時最適化を支援するランタイムシステム*

5ZB−07　青木 政人　小野 喬史　大津 金光　横田 隆史　馬場 敬信 †
宇都宮大学工学部情報工学科‡

1　はじめに

我々は，シングルスレッドコードからマルチスレッドコードへのバイナリトランスレーションシステムを提案し，実現に向けての研究を進めている[1, 2]．その際，実行前に静的に行う最適化には限度があり，実行時に行う最適化が必要となるが，実行時最適化のためには，どのようにして低コストで，必要な実行履歴を記したプロファイルデータを取得するか，取得したデータをどのように最適化へ適用するかといった問題がある[4]．

本稿では，この一環として，詳細な実行履歴を記したプロファイルデータを自動生成するために，サンプリングによる自動プロファイリングシステムを試作し，その評価をした結果について述べる．

2　システムの概要

2.1　システム全体図

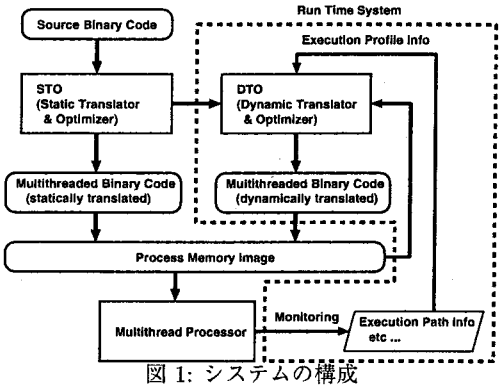

図 1: システムの構成

図1にシステムの構成図を示す．本システムは大きくわけて，プログラム実行時に動作する，ランタイムシステムに相当する **DTO (Dynamic Translator and Optimizer)**(点線内)と，プログラム実行前に動作する **STO(Static Translator and Optimizer)**，及び，実際にプログラムを実行するマルチスレッドプロセッサの3つの部分から構成される．アプリケーションプログラムは **STO** によりマルチスレッド化されたコードにバイナリ変換され，プロセスイメージを生成する．これをマルチスレッドプロセッサ上でマルチスレッド実行することで，実行性能の向上を図る．

* Run-time System for Binary Level Optimization
† Masato Aoki, Takafumi Ono, Kanemitsu Ootsu, Takashi Yokota, and Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

実行前にマルチスレッド化処理ができなかった部分のコードに関しては，**DTO**が実行時にこれらの処理を行なうことになる．マルチスレッドプロセッサは実行と同時にプロファイル情報を収集しており，定期的あるいは，特定のイベントの発生などのタイミングで **DTO** に実行を遷移させる．**DTO** に制御が移されると，**DTO** は実行時に収集されたプロファイル情報からプログラムのホットスポットを選別し，該当部分のコードに最適化の余地があれば最適化を行なう．

2.2　プロファイリングシステム

本研究では，クロック割り込み毎にPCをサンプリングするシステムを実装した．図2に，本システムの動作の流れを示す．

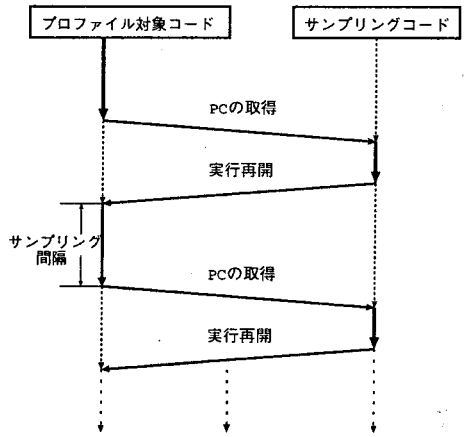

図 2: サンプリングシステムの動作

処理の流れは，始めに基本ブロック分割プログラムによって，入力バイナリを基本ブロックへと分割し，取得した基本ブロック情報をファイルへ出力し，そのファイルをサンプリングコードに受渡すことで，基本ブロック情報を取得し，サンプリングコード内のアドレステーブルへ登録する．次に割り込みをかけるインターバルを設定し，ユーザー側のバイナリの実行へと移る．そして設定したインターバル経過の後，割り込みがかかり，その時のPCの値を取得し，取得したPCの値をアドレステーブルに登録，再度，インターバルを設定し，ユーザー側のバイナリの実行へと移る．これらの繰り返しにより，サンプリングを行う．その後，ユーザーアプリの終了と同時にプロファイルコードも終了となる．

3 評価

評価にはスレッドパイプライニングモデルのシミュレータ **SIMCA**[3] を用いた．また，対象となるバイナリはSIMCA用gccクロスコンパイラ(version2.7.2.3)を用いて生成したコードを用いた．評価用プログラムとして，SPECint95のm88ksimを用いた．データセットはtestである．図3にm88ksimのサンプリングの実行結果を示す．横軸が基本ブロックのアドレスに対応し，縦軸がサンプリングした際に，対応する基本ブロックが実行されていた回数を示している．サンプリング間隔は200000サイクルである．また，サンプリング間隔10000, 50000, 100000, 500000, 1000000, 2000000においても同様の結果を得ている．

図3の結果を見ると，200000サイクル毎のサンプリングであっても，頻繁に実行される基本ブロックと，そうでない基本ブロックとの実行回数の差は顕著であり，最適化を行うべき部分が明示的に分かる．ゆえにこのプロファイリング結果は最適化を行うために十分な情報が得られていることが分かる．

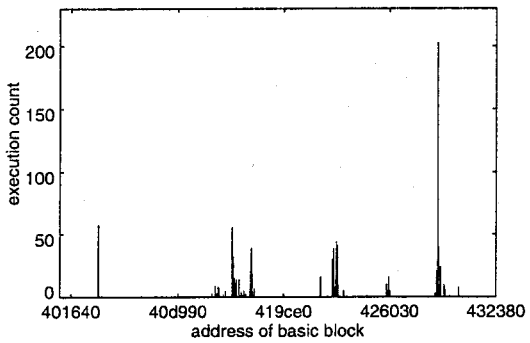

図 3: m88ksimにおけるプロファイリング結果

次に，図4にはプロファイルを取得するためのオーバーヘッドを示す．横軸がサンプリングを行う間隔であり，縦軸がサンプリングによるオーバーヘッドの割合を示す．m88ksimにおいては，一回のサンプリングにかかるサイクル数が約300サイクルを要する．しかし図4を見ると，サンプリング間隔を広げていくことにより，総実行サイクル数の1％以下という低オーバーヘッドに抑えている．しかし，大幅にサンプリング間隔を広げた場合には，サンプリングを行う回数が減少し，ホットスポットの検出が困難になる．

4 おわりに

本稿では，ハードウェアクロックを用いて，一定周期毎に割り込みをかけ，その際にPCの値をサンプリングするシステムを実装し，評価を行った．

評価結果より，ある程度サンプリングの間隔が広くなっても，基本ブロック毎の実行回数の違いは顕著であり，最適化を行う際に対象とすべき基本ブロックの検出が可能であることが示された．

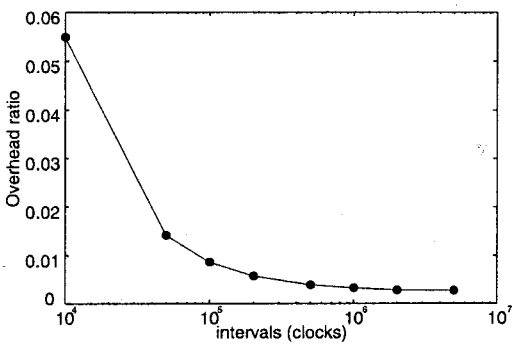

図 4: サンプリングによるオーバーヘッド

さらに，サンプリングによって生じるオーバーヘッドも間隔を調整することで，1%以下という，実行性能に影響を与えない実用的な値に抑えることが可能であることを示した．

今後の課題として，取得したサンプリング結果から，マルチスレッド化の対象とする部分を見つける方法．マルチスレッド化を用いて最適化するか，用いないで最適化を行うかの，その判断基準の確立．詳細で質の高いプロファイリング結果の取得と，オーバーヘッドのトレードオフの問題や，どのような実行回数をもって，ホットパス，ホットスポットとするのかといった，閾値の問題も課題として上げられる．

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)課題番号12680328)の援助による．

参考文献

[1] 大津金光， 小野喬史， 馬場敬信．バイナリレベルにおけるマルチスレッド化コード生成手法，情報処理学会研究報告(ARC141)，Vol.2001，No.10，pp.41-46，2001.

[2] K.Ootsu, T.Yokota, T.Ono, and T.Baba. A Binary Translation System for Multithreaded Processors and its Preliminary Evaluation, Proc. 5th Workshop on Multithreaded Execution Architecture and Compilation, pp13-21(2001).

[3] J. Huang. The SImulator for Multi-threaded Computer Architecture(SIMCA), Release 1.2. http://www-mount.cs.umm.edu/Research/Ag-assiz/simca.html.

[4] Nick Gloy, Zheng Wang, Catherine Zhang, Brad Chen, and Mike Smith. Profile-based Optimization with Statistical Profiles, TR-02-97, Center for Research in Computing Technology, Harvard University.