Vol.2023-ARC-251 No.3
2023/1/10

サイクル・アキュレートなシミュレータ実行トレースの解析におけるメモリアクセスに着目した統計の活用

東郷 凛太朗[1]　野村 隼人[1]

概要：プロセッサの高速化を目的としたアーキテクチャ研究のプロセスのひとつに，プログラム実行時の動作を観察し，ボトルネックとなる動作を見つけ，それを改善可能な機構を考案，開発するという流れがある．サイクル・アキュレートなプロセッサシミュレータはそのために利用されるツールのひとつである．これにより，プログラムの実行をサイクル精度でシミュレートし，どの命令実行でどれだけストールが発生しているかを調べることができる．しかし，その情報だけでは，どのような命令やアドレス帯でストールを起こしやすいのか，どのような構造体でストールが起きているのかを十分に把握することは難しい．本研究では，命令パイプラインビジュアライザ Konata を改良し，操作やデータとストール発生の関係を明快にし，それに基づいて支援されたプロファイリングの作業手順を提案する．

1. 背景・目的

プロセッサの高速化を目指すにあたって，サイクル・アキュレートなシミュレータを用いてプログラムをサイクル精度でシミュレートし，プロセッサの動作を解析するアプローチがある [1]．そこではどの命令実行でストールが発生しているかなどの情報を得られるが，どのような処理をした時，どのような構造体にアクセスした時にストールが発生するかといった情報は直接得られず，実行速度低下の原因箇所を見つけ出すことは難しい．

本研究では，メモリアクセス単位で命令実行の情報を統計し提示することで，それに基づいて支援されたプロファイリングの作業手順を提案する．

2. 先行研究

現在オープンソースで公開されているサイクル・アキュレートなプロセッサシミュレータに鬼斬弐 [2] があり，鬼斬弐が出力する実行トレースは命令パイプラインビジュアライザ Konata によって可視化できる．Konata には統計機能も備わっており，シミュレーション全体の実行命令数や分岐予測ミスの回数，IPC などを見ることができる．

3. メモリアクセスに着目した統計の提案

プロセッサ高速化のアプローチのひとつに，キャッシュシステムの改善がある．これに試みる場合，キャッシュミスの発生頻度が高い状況を特定することで効果的にプロセッサ性能を改善できる．

Konata ではシミュレーション全体の統計を見ることができるが，本研究の提案システムではさらに細かくメモリアクセス単位で統計する．鬼斬弐によるシミュレーションで出力される実行トレースからメモリアクセスした命令実行とそこでアクセスしたアドレスの情報を得られるため，それにより命令実行を分類していくつかのグループに集約して統計する．ここで同じまたは近いアドレスにアクセスした命令実行を集約することで，どのアドレス群にアクセスした時に実行の様子がどのようになっているかが分かる．例えばあるアドレス群へのアクセスで実行時間が長くなっていれば，そのアクセス群でキャッシュミスが多く発生していると考えることができる．

4. 実装

Konata に機能を追加する形で実装する．

シミュレートするプログラムにデバッグ情報を埋め込んでコンパイルしたバイナリを objdump に渡し，`-dlS` オプションを付けて実行した時の出力を読む機能を追加し，読み込むと命令実行の情報と同時にその命令に対応するソースコードの断片と行番号を表示する機能を追加する．

アクセスしたアドレスによって命令を集約し，それぞれのアドレス群に対応する命令の実行回数の合計を**実行回数**，ステージ数の合計を**ステージ数**，ストールと committable を除いたステージ数の合計を**実質のステージ数**，ストールしたステージ数の合計を**ストールしたステージ数**，ステー

[1] 明石工業高等専門学校
National Institute of Technology, Akashi College

アドレス	実行回数	ステージ数	1実行当たりのステージ数	実質のステージ数	1実行当たりの実質のステージ数	ストールしたステージ数	1実行当たりのストールしたステージ数	ストール率	ソースコード
87e78-1086e78	197616	63291937	320.28	47146280	238.58	16132786	81.64	25.49%	++long_array[j];
1087e78-108be74	202345	7988385	39.48	5609229	27.72	407239	2.01	5.098%	++short_array[j];

図 2 メモリアクセスに着目した統計の結果（Konata 0.38.0 改のスクリーンショット）

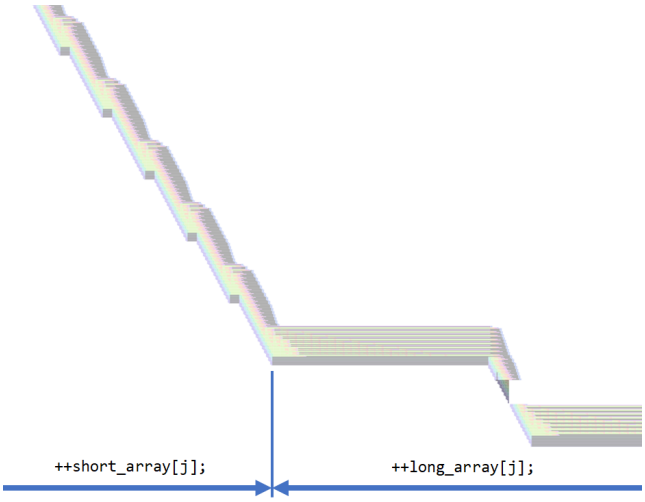

図 1 実行の様子が変化した時のパイプラインビジュアライザの出力

ジ数に対するストールしたステージ数の割合を**ストール率**，さらに，ステージ数，実質のステージ数，ストールしたステージ数をそれぞれ実行回数で割った**1 実行当たりの平均値**を計算し，表で出力する．

5. 評価環境

使用したソフトウェアのバージョンは以下の通りである．

- 鬼斬弐: v2.9
- Konata: 0.38.0 改
- RISC-V GNU Compiler Toolchain: 2022.06.10

評価に使用したソースコードは，キャッシュミスを起こすケースと起こさないケースが混ざって観測できるようなものを書いた（**図 1**）．

シミュレーションでは，短いシミュレーションでもキャッシュミスの様子が見やすくなるようにキャッシュ容量を小さく設定した．シミュレーションに用いたパラメータは以下の通りである．

- スキップする命令数: 10 k 命令
- シミュレートする命令数: 1 M 命令
- L1D キャッシュ: 容量 1 KB
- L1I キャッシュ: 容量 32 KB
- L2 キャッシュ: 容量 64 KB, ストリームプリフェッチャ有, ストライドプリフェッチャ有

6. 結果

作成した機能でメモリアクセスに着目した統計を出力した様子を**図 2** に示す．これを見ると，`long_array` へのアクセスは `short_array` へのアクセスに比べて 1 実行当たりのステージ数が約 8.1 倍，実質のステージ数が約 8.6 倍，ストールしたステージ数が約 40.6 倍，ストール率が 5 倍であることが分かる．このことから，配列 `long_array` へのアクセスでキャッシュミスが多発していることが分かり，そのようなアクセスでのキャッシュミス率を下げることができれば，このプログラムの実行について高速化できる．

このように，対処できれば高速化に繋がるメモリアクセスを起こしているアドレス群の範囲や構造体を見つけ，そのアクセス回数やストール率を計算する作業は従来人がしていたものであるが，それが自動化されている．また，デバッグ情報を読み込ませることによりメモリアクセスに対応するソースコードを表示することができるため，アセンブリコードを見ながらソースコードを辿る手間も省ける．これにより，アーキテクチャ上のボトルネックを見つける作業が支援される．

7. 展望

キャッシュミスの発生箇所の特定を自動化することができたため，その情報を用いて実際にプロセッサの改善に取り組む．それに当たって，集約されたアドレス群の各群に対する改善により見込まれる性能向上を，IPC などの具体的な数値で新たに統計情報として Konata 上に表示する．

また，現状は表として出力しているが，これに指定した命令群についてそのメモリアクセスの様子をプロットする仕組みを追加することで，より視覚的にメモリアクセスの特性の把握を支援する．

参考文献

[1] Butko, A., Garibotti, R., Ost, L. and Sassatelli, G.: Accuracy evaluation of GEM5 simulator system, *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–7 (2012).
[2] 塩谷亮太: プロセッサ シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS2009, Vol. 2009, No. 4, pp. 120–121 (2009).