E-06

CUDA プログラムにおけるオーバヘッドの小さいタイムスタンプ記録手法の検討

A Low Overhead Method for Timestamp Logging in CUDA Programs

神田裕士† 奥山倫弘† 伊野文彦† 萩原兼一†
Hiroto Kanda Tomohiro Okuyama Fumihiko Ino Kenichi Hagihara

1. はじめに

GPU（Graphics Processing Unit）はグラフィクス処理用の並列プロセッサであり，CPU を上回る演算性能を持つ．近年，GPU 向け開発環境 CUDA（Compute Unified Device Architecture）を用い，数千個ものスレッドにより汎用計算を高速化する試みが盛んである．

並列プログラムの性能を解析するための手段として，実行履歴に基づく手法がある．ここで実行履歴とは，プログラム実行時に記録したタイムスタンプを時系列に並べたものである．実行履歴はプログラム実行時の挙動を提示でき，開発者を支援できる．例えば，スレッドごとの計算負荷や進捗のばらつきなどをガントチャート上に可視化できる．

我々は，CUDA プログラムを対象として実行履歴を生成するための手法[1] を開発している．実行履歴をプログラム実行時に保存するために，この既存手法はタイムスタンプを取得しメモリに書き込む処理（記録処理）を対象プログラムに挿入する．この処理は挿入前のプログラムに存在しないため，挿入により実行時間が 30%ほど増加している．つまり，実行履歴より得られる実行状況，および実行履歴を生成することなく実行した場合の（元の）実行状況の間にずれがある．ずれの大きい実行履歴はチューニングすべき箇所を誤認識させる可能性がある．したがって，正確な実行履歴を得るために，生成オーバヘッドの小さな記録手法が必要である．

そこで本稿では，正確な実行履歴を得ることを目的として，生成オーバヘッドの小さいタイムスタンプ記録手法を提案する．生成オーバヘッドを削減するために，提案手法は対象プログラムにおける一部の処理を記録処理に置換する．なお，置換の結果，コンパイラが一連の命令を削除したコードを生成する可能性がある．本来実行すべき命令が削除されないように，提案手法はダミー処理をプログラムに追加しコンパイラの過度な最適化を防ぐ．

2. CUDA プログラムにおける実行履歴生成

実行履歴生成において使用する GPU のメモリ階層について述べる．GPU にはグローバルメモリおよび共有メモリの 2 種類のメモリ領域がある．グローバルメモリは GPU 上の全てのスレッドから参照でき，CPU とのデータ受け渡しができるメモリ領域である．アクセス時には 400 クロック程度の遅延が発生する．

一方，GPU 上のスレッドは複数のブロックに分割されており，同一ブロック内のスレッド間でのみ共有できるメモリ領域が共有メモリである．共有メモリはレジスタと同程度の短いアクセス遅延を持つが，その容量は小さい．共有メモリ上のデータは CPU から参照できないため，GPU 上で計算した結果などを CPU 側で参照したい場合は，グローバルメモリに記録する必要がある．

実行履歴に記録する情報は，各スレッドで取得したタイムスタンプおよびプロセッサの ID である．GPU は異なるプロセッサ上のスレッドを独立して実行するため，どのスレッドを同一プロセッサ上で実行しているかが重要となる．プロセッサ ID は同一プロセッサに割り当てられているスレッドを把握するために必要である．これらの情報をグローバルメモリに記録し，CPU 側で実行履歴のファイルを出力する．

記録処理において，共有メモリを利用してグローバルメモリへのアクセス回数を削減するために，記録処理を 2 段階に分割する．タイムスタンプの取得時には，そのタイムスタンプを共有メモリに保持しておく．その後，GPU プログラムの終了時にまとめてグローバルメモリに書き込む．

なお，タイムスタンプの取得には CUDA が提供する clock 関数を使用する．clock 関数は 1 クロックごとにインクリメントされるレジスタの値を返す．

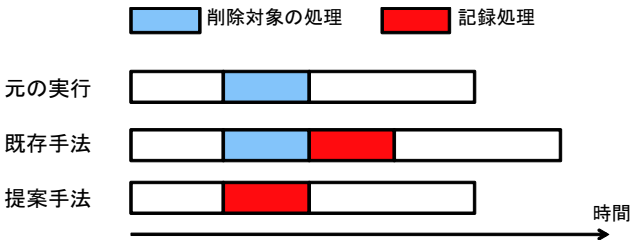

図 1　置換によるオーバヘッドの削減

3. 提案手法

提案手法は，解析対象のプログラムに記録処理を挿入し，対象プログラムにおける一部の処理を削除することで，生成オーバヘッドの削減を図る（図 1）．挿入する処理の内容および挿

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

入箇所が与えられたときに，生成オーバヘッドを小さく抑えられるように削除対象を選択する．選択の方針は以下の 2 つである．これらの方針に従い，選択できる処理が挿入箇所に存在する場合のみ置換を適用する．

方針 1　発生する遅延の長さが，挿入する処理の遅延と同等である処理を選ぶ

方針 2　削除により副作用が発生しないか，あるいはその発生を防止できる処理を選ぶ

方針 1 に関して，置換前後の処理がいずれも計算命令，あるいはメモリアクセス命令であるなど，同種の命令であれば，発生する遅延の長さが同等であると考える．置換前後の遅延の長さが同等でなければ，生成オーバヘッドを削減できないか，あるいは本来発生すべき遅延が発生しなくなり，正確な実行履歴が得られない．したがって，方針 1 に従う必要がある

方針 2 における副作用とは，処理の挿入あるいは削除をしていない箇所における命令数あるいは処理時間の増減を指す．図 2 に示すコードにおいて，副作用の例を説明する．このコードを実行すると，`condition` の値は −1 となる．その後の条件式で `condition` の値を参照しており，結果として関数 `function2` が実行される．しかし，ここで 3 行目の処理を削除したとすると，最終的な `condition` の値は 0 となる．条件式の評価結果も真に変わることから，実行する関数は `function1` となる．実行フローおよび処理内容の変化により，実行命令数が増減する．これが処理の削除により発生する副作用の例である．方針 1 に従って削除対象を選択しても，副作用の発生により，置換前後で本来実行されない処理が実行されるか，あるいは逆に本来実行すべき処理を削除してしまう場合がある．したがって副作用の発生は避けるべきである．

方針 1 および 2 に従い，タイムスタンプをグローバルメモリに書き込む処理に置換を適用する．挿入箇所が GPU プログラムの終了時であることから，方針 1 に従い，対象プログラムにおける計算結果のグローバルメモリへの書き込み処理を削除対象の候補として選択する．

次に方針 2 に従い，置換時の副作用について検討する．計算結果の書き込みを削除するときに発生しうる副作用として，コンパイラの最適化による命令の削除がある．図 3 に単純に置換した場合の擬似コードを示す．関数 `computation` で計算した結果 `result` をメモリ領域 `output` に書き込む．これが元のプログラムの処理であるとする．この例では，`output` への書き込みを実行しないようコメントアウトし，履歴記録用のメモリ領域 `log` へタイムスタンプを書き込む処理を追加することで置換を実現している．

この例では置換の結果，変数 `result` を参照する処理が存在しない．このときコンパイラは，`result` への値の代入および代入値を求める関数 `computation` の実行を不要と判断し，これらを削除する．結果として，処理の挿入あるいは削除をした箇所とは異なる箇所において，実行命令数が減少する．

この副作用の発生は，ダミー処理をプログラムに追加することで防げる．図 4 に例を示す．if 文を用いて，変数 `tmp` に `result` あるいは `timestamp` のいずれかを代入し，最後に `tmp`

```
1: condition = 0;
2: a = 1; b = 2;
3: condition = a - b;
4: if(condition == 0)
5:   function1();
6: else
7:   function2();
8: ...
```

図 2　副作用の例を示すためのサンプルコード

```
1: result = computation(input);
       /* 計算結果を求める関数 */
2: //output[index] = result;
       /* 計算結果の書き込みを削除 */
3: log[index] = timestamp;
       /* タイムスタンプの書き込み */
```

図 3　単純な置換の例

```
1: result = computation(input);
2: if(always false)
3:   tmp = result; /* 計算結果の書き込み */
4: else
5:   tmp = timestamp; /* タイムスタンプの書き込み */
6: log[index] = tmp;
```

図 4　ダミー処理を追加した置換の例

をメモリ領域 `log` に書き込む，というコードである．分岐次第で `result` をメモリに書き込む可能性があるとコンパイラが判断するため，`result` への値の代入および関数 `computation` を実行する処理の削除を防げる．if 文の条件式として常に偽と評価される式を与えているため，実際には `result` は参照せず，常にタイムスタンプのみを `log` に書き込む．

以上の方法で，置換時に発生する副作用を防げるため，方針 1 および 2 の両方から，グローバルメモリへの書き込みを置換対象として選択することに問題はないと判断できる．

一方，共有メモリへの書き込み処理は，削除対象の選択が難しいため，置換対象として選択しない．まず，処理を挿入する箇所はタイムスタンプを記録する箇所であり，開発者が任意に選択する．その箇所に方針 1 に従う，すなわち遅延の大きさが同等となる処理が存在せず，置換できない場合がある．また，副作用の観点からも制約が厳しい．計算結果の書き込みを置換する場合に限らず，置換の結果として不要になる処理があれば，コンパイラの最適化による命令の削除は発生する．共有メモリへの書き込みの場合，1 箇所の置換により削減できるオーバヘッドが小さいため，ダミー処理における分岐命令の増加が無視できない．そのため，コンパイラの最適化が発生する置換を避ける必要がある．

さらに，図 2 の例で示したような，削除する処理を参照していた処理に対する副作用の発生も避けなければいけない．例で示した実行フローの変化だけではなく，メモリアクセスの規則性を乱してしまうなどの副作用も発生しうる．グローバルメモリへの書き込みを置換する場合は，計算結果を GPU プログラム内で再度参照することは考えにくいため，この副作用は無視できる．

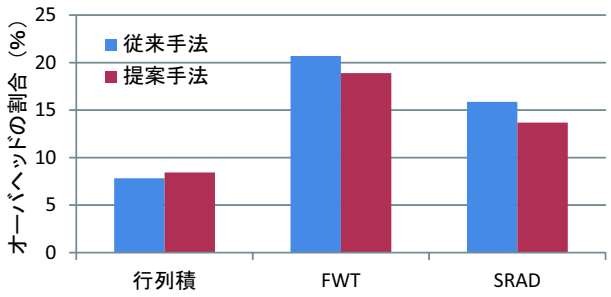

図 5 タイムスタンプ記録時のオーバヘッド

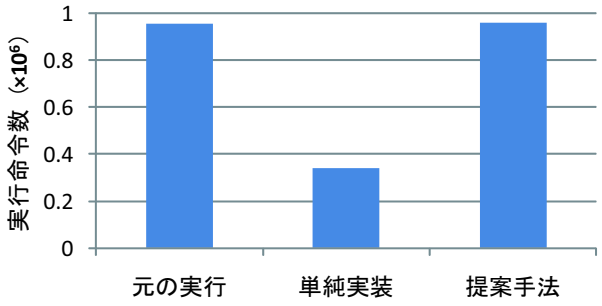

図 6 置換時の実行命令数

4. 評価実験

実験対象のアプリケーションは行列積[2]，FWT（Fast Walsh Transform）[2]，および SRAD（Speckle Reducing Anisotoropic Diffusion）[3] である．スレッドあたりのタイムスタンプ記録回数は行列積，FWT および SRAD のそれぞれにおいて 32 回，8 回および 9 回である．置換の前後で，アクセスするグローバルメモリのデータ量およびアクセスパターンは同じである．

本来の実行時間に対するオーバヘッドの割合 $R = 100(T_2 - T_1)/T_1$ を評価する．ここで T_1 および T_2 はそれぞれ本来の実行時間およびタイムスタンプ記録時の実行時間を表す．

行列積においては従来手法の R が提案手法の R よりも小さい（図 5）．行列積においては，メモリアクセスのデータ量の観点から比較して，書き込みデータ量は全体の 1%未満である．アクセス時間の比率も同様になることから，置換によるオーバヘッド削減の効果が小さい．FWT および SRAD においては，従来手法の結果と比較して，提案手法の R は約 2%程度減少した．以上の結果より，記録処理のオーバヘッド全体に対し，グローバルメモリへの書き込みを原因とするオーバヘッドが占める割合は小さい．したがって，グローバルメモリへの書き込みのみの置換ではオーバヘッド削減効果は不十分である．

次に，ダミー処理をプログラムに追加した場合の，コンパイラによる最適化の防止効果について確認する．実験対象のアプリケーションは行列積である．元のプログラムを実行した場合（元の実行），ダミー処理を追加せずに置換を適用した場合（単純実装）およびダミー処理を追加した場合（提案手法）のそれぞれについて，CUDA プロファイラ[4] を用いてプロセッサあたりの実行命令数を計測した．

図 6 に計測結果を示す．単純実装では，元の実行と比較して命令数が約 3 分の 1 に減少しており，最適化により命令が削除されていることがわかる．それに対して提案手法を適用した場合には，ほぼ元の実行と命令数が等しい．このことから，ダミー処理の追加によりコンパイラの最適化を防げていることが確認できた．

5. まとめ

本研究では CUDA プログラムの正確な実行履歴を生成することを目的として，生成オーバヘッドの小さいタイムスタンプ記録手法を提案した．提案手法は対象プログラムにおける計算結果の書き込みをタイムスタンプの書き込みに置換することにより，グローバルメモリへのアクセスが原因で生じるオーバヘッドを削減する．また，置換の結果本来実行すべき命令がコンパイラの最適化により削除される問題を，ダミー処理をプログラムに追加することで解決した．

3 つのアプリケーションに提案手法を適用した結果，削減できたオーバヘッドは約 2%であった．また，全体の処理のうちグローバルメモリへの書き込みが占める割合が小さい場合，提案手法によるオーバヘッド削減の効果は小さい．

今後の課題としては，記録処理のうち，共有メモリへの書き込みについても置換を適用することが挙げられる．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（23300007）および大阪大学グローバル COE プログラム「予測医学基盤」の補助による．

参考文献

1) 神田他:CUDA カーネルの性能を解析するための実行履歴生成ツール，情処研報，2010-HPC-126，7 pages (2010).
2) NVIDIA Corporation: NVIDIA CUDA SDK Code Samples (2010). `http://developer.nvidia.com/object/cuda_sdk_samples.html`.
3) Che, S. et al: Rodinia: A Benchmark Suite for Heterogeneous Computing, *Proc.IISWC 2009*, pp. 44–54 (2009).
4) NVIDIA Corporation: CUDA Visual Profiler User Manual (2011).