4K-5

CUDAプログラムのロギング・再演機構

松浦　広明　丸山　真佐夫†
木更津工業高等専門学校†

1　はじめに

近年，GPGPU が注目を集めており，代表的な開発環境として CUDA が広く知られている．

CUDA によるプログラムのデバッギングは逐次プログラムと比べると，より困難である．この原因として，CPU と GPU の非同期な動作，GPU のメニーコアによる非決定的な並列計算，想定される大規模計算でのデバッグ実行時間があげられる．我々はホストプログラムおよびカーネルの単独実行により，この問題が解決できると考えた．

並列プログラムの非決定性に対する解決手段として，再演法が提案されている．再演法には順序再演法[1]とデータ再演法がある．GPU はイベント順序の保存および再現の制御が困難であるため，順序再演法は不適である．データ再演法による並列計算のデバッギング手法が提案されている[2]．この手法では，1 回の実行によりデバッグ対象の処理の実行に必要なデータを収集し，以後の実行ではそのデータを元に実行を再現する．

本稿では，ホストプログラムおよびカーネルの単独実行をデータ再演法により実現し，CUDA プログラムのデバッギングを支援するロギング・再演機構について提案する．

2　データ再演法によるデバッギング

2.1　デバッギングにおける問題

CUDA によるプログラムは CPU と GPU によって非同期で実行される．計算順序が定まらないプログラムでは，毎回異なる結果を出力する可能性があり，デバッギングが難しい．

GPU は大量のコアによって計算を行う．コアは並列に動作するため，複数のコアが同一メモリ領域を操作すると，実行ごとに異なる計算結果が出力される可能性があり，デバッグが困難となる．

GPGPU の対象となる計算は，その特性のために大規模であることが想定される．計算順序が後方の処理をデバッグするとき，必要のない前処理も実行しなければならない．前処理に長時間かかる場合，必要でない処理によってデバッグ実行時間が増え，デバッギングの効率が低下する．

2.2　データ再演法による再演実行

これらの問題の解決手段として，ホストプログラムおよびカーネルの単独実行が考えられる．ここでいう単独実行とは，あらかじめ実行に必要な入力データを収集しておき，そのデータを元に単独実行するというものである．

入力データを固定化することで，非同期な動作による計算順序の変化を気にせずに済む．メニーコアによる非決定的な計算については，実行する処理自体の非決定性は解消できないが，決定的な入力を受け取ることができるのでデバッグしやすくなる．また，単独実行する処理以外の部分を省略できるため，デバッグ実行時間を短縮できる．

ホストプログラムおよびカーネルの単独実行をデータ再演法によって実現する．これにより，
・ホストプログラムの再演実行
・任意のカーネルの再演実行
・再演に必要なデータを収集するロギング実行
が可能となる．一度のロギング実行で再演実行に必要なデータを収集し，以後の実行では収集したデータを元に再演する．

通常，ホストとデバイスのメモリ空間は独立しており，メモリコピーは基本的に CUDA API を通して行われる．再演機構はその API を横取りし，再演に必要なログデータの保存・再現を行う．ホストプログラムの再演実行ではデバイスからホストにコピーされるデータを保存・再現する．カーネルの再演実行では逆にホストからデバイスへのメモリコピーを保存・再現する．

ロギング実行では通常の処理に加えてログデータの保存も行うため，オーバーヘッドが発生する．ログデータを保存する領域も必要となる．再演実行では，ログデータによる再現のみを行うため，前処理にかかる時間が大きいほど実行時間の短縮が見込める．デバッグ実行は繰り返

"Logging and Replay System for CUDA Programs"
†Hiroaki MATSUURA and Masao MARUYAMA (Kisarazu National College of Technology)

し行われることが想定されるため，ロギング実行のオーバーヘッドより，再演実行時の時間短縮が重要である．

3 ロギング・再演機構の実装

3.1 ロギング・再演機構の構成

ロギング・再演機構は，CUDA API へのラッパとして提供され，アプリケーションコードによる CUDA API の呼び出しを横取りする．このとき，アプリケーションコードに影響がないように API をシミュレートし，必要ならばログデータを収集する．同様にアプリケーションコードの main 関数も横取りし，代わりに再演機構の main 関数を呼び出し，そこで初期化処理を行う．再演機構を API へのラッパとすることで，アプリケーションコードを変更せずに再演機構を組み込むことができる．

3.2 ログデータの保存と再現

ログデータの保存はホスト・デバイス間のメモリコピーの API を横取りして行う．このとき，シミュレート対象から再演対象に向かって行われるメモリコピーをロギングする．この対応表を表 1 に示す．

カーネルの再演実行を行うためには，カーネル実行直前のデバイスメモリの状態を再現する必要がある．現在の実装では，そのカーネル実行に影響する API を時系列順にすべて再演することで実現している．

デバイスメモリはカーネル終了後も保持される．したがって，メモリコピーの API を横取りするだけでなく，影響を受ける他のカーネルも実行する必要があり，この再現方法では効率が悪い．この問題はカーネル実行の直前で必要となるデータをすべて保存しておくことで改善できる．最終的には再演実行時にデバイスメモリの再現方法を選択実行することを考えている．

4 性能評価

本システムの基礎的なオーバーヘッドを測定するために，1 MB のデータをホスト・デバイス間で 100 往復分コピーするプログラムの実行時間を計測した．このプログラムを 10 回実行し，

表 1：ロギングするメモリコピーの方向

再演対象	メモリコピーの方向
ホストプログラム	デバイス→ホスト
カーネル	ホスト→デバイス

表 2：各種実行の計測結果

実行内容	実行時間[s]	倍率
通常	0.342	1.00
ロギング	4.06	11.9
ホスト再演	0.104	0.53
カーネル再演	0.181	0.30

その平均時間を実行時間として採用した．通常実行，ロギング実行，ホストプログラムの再演実行，カーネルの再演実行の 4 通りについて計測し，実行時間を比較した．ロギング実行では，ホストプログラムとカーネルの両方で再演実行できるように，双方向のメモリコピーに対して，必要なログデータを保存した．計測環境は以下の通りである．CPU は Core2Duo(2.53GHz)，メモリは 4GB，GPU は GeForce 9400M，CUDA のドライバおよびランタイムのバージョンはそれぞれ 4.0 である．計測結果を表 2 に示す．

計測の結果，実行時間が通常実行と比べて，ロギング実行時で約 11.9 倍，ホストプログラムの再演実行で約 0.53 倍，カーネルの再演実行で約 0.30 倍となった．1 回のロギング実行により保存されるログデータの容量は 200MB（1MB×2 回×100 回）である．

このプログラムは GPU での計算時間を含まない．よって実プログラムでは相対的にオーバーヘッドが小さくなると予想される．

5 まとめ

本研究では，データ再演法による CUDA プログラムのデバッギングを支援するロギング・再演機構について述べた．

通常の実行時間と比べ，ロギング実行では約 11.9 倍，ホストプログラムの再演実行では約 0.53 倍，カーネルの再演実行では約 0.30 倍という良好な結果を得た．今後は提案したロギング・再演機構を用いたデバッギング機構を実現する予定である．

参考文献

[1] LeBlanc，T.J．and Mellor-Crummey，J.M，“Debugging Parallel Programs with Instant Replay”，IEEE Trans．Comp，Vol．C-36 No．4，pp．471-482(1987).

[2] 丸山真佐夫，津邑公暁，中島浩，“データ再演法による並列プログラムのデバッギング”，情報処理学会論文誌：コンピューティングシステム，Vol.46，No．SIG 12 (ACS11)，pp.214-224(2005)