5H-1

異なる OpenCL 実装を透過的に扱える Hybrid OpenCL の開発

青木　亮 †　　追川　修一 †　　土山　了士 ‡　　中村　孝史 ‡
筑波大学 コンピュータサイエンス専攻 †　　株式会社フィックスターズ ‡

1　はじめに

本研究では，異なる OpenCL[1, 2] 実装の接続を可能にする Hybrid OpenCL を開発した．Hybrid OpenCL は異なる OpenCL を抽象化するランタイムシステムと，複数のランタイムシステムを接続するブリッジプログラムから成る．OpenCL には次の 2 つの問題がある．1 つは同一マシンに異なる種類の OpenCL デバイスを接続しても同時に扱うことができない点，もう 1 つは同一マシンに搭載可能な GPU 等の計算デバイス数が内部バスの数に制限されてしまう点である．さらに，これらの問題によりアプリケーションの利用価値が下がってしまうという問題も浮上する．Hybrid OpenCL ではこれら 2 つの問題を解決することで，スケーラビリティの向上と多様な並列計算機環境を提供し，同時にアプリケーションの利用価値を向上させることを目的とする．

2　Hybrid OpenCL

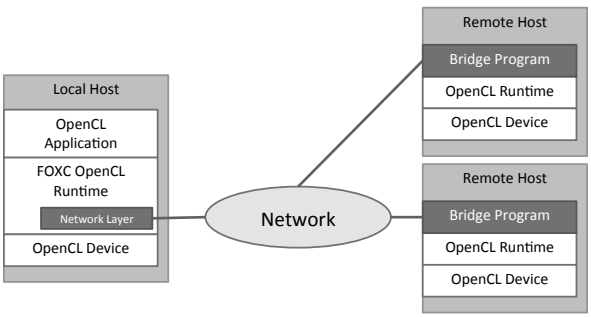

図 1　Hybrid OpenCL

Hybrid OpenCL は複数の OpenCL ランタイムを透過的に扱うために，図 1 のように，OpenCL アプリケーションにリンクする OpenCL ランタイムに，ネットワークレイヤを追加し，他の OpenCL ランタイムとブリッジする方法をとる．OpenCL ランタイム間にはブリッジプログラムを配置し，これが OpenCL アプリケーションからの API 呼び出しを RPC(Remote Procedure Call) を用いて再現し，リモートホストにおいての実際の OpenCL リソースを管理する．本章では，Hybrid OpenCL の設計とその実装について述べる．

2.1　設計

本研究では OpenCL ランタイムにネットワークレイヤを追加したものを Hybrid OpenCL ランタイムと呼ぶ．また，リモートホストの OpenCL デバイスを単にリモートデバイス，ローカルホストの OpenCL デバイスをローカルデバイスと呼ぶ．

OpenCL を透過的に扱うために，Hybrid OpenCL ランタイムは主に 2 つの処理を行う．1 つはリモートデバイスをリストアップし管理する処理，もう 1 つは，ブリッジプログラムの OpenCL ランタイムで取得したリソースハンドルと，実際に OpenCL アプリケーションに渡すリソースハンドルとの関連付けを行う処理である．

一方，ブリッジプログラムは，Hybrid OpenCL ランタイムからの API 呼び出しを，自身にリンクされている OpenCL ランタイムで実行し，結果を Hybrid OpenCL ランタイムに返す処理を行う．例えばバッファを作成する場合，cl_mem = clCreateBuffer(context, flag, size, host_ptr, errcode) という API を使用する．OpenCL アプリケーションによって clCreateBuffer() が呼び出された際，context がリモートデバイスを含む場合は，clCreateBuffer の API を表す整数値とリモートの context，引数である flag，size，host_ptr が Hybrid OpenCL ランタイムからブリッジプログラムへ渡される．errcode は出力引数であるのでブリッジプログラムへは送信されない．ブリッジプログラムはこららを受け取ると，ブリッジプログラム自身にリンクされた OpenCL ランタイムライブラリの API にそれらを渡し，実行する．そして，結果である cl_mem と errcode を呼び出し元のホストへ送信する．

2.2　実装

今回は Hybrid OpenCL ランタイムを開発するに当たって，Fixstars 製の FOXC[3](Fixstars OpenCL Cross Compiler) OpenCL ランタイムをベースとした．これは Linux で動作し Intel のマルチコア CPU をデバイスとみなす OpenCL 実装である．

実装量は，OpenCL API を 71 個中 42 個実装し，FOXC OpenCL ランタイムへの追加コードが C 言語で 約 5300 行，ブリッジプログラムは C++ 言語で約 4500 行となった．これは OpenCL ランタイム API 数が 71 個と多いことから，API の引数や戻り値をホスト間でやりとりするために必要なデータのシリアライズ・デシリアライズするコードが大半を占めているためである．

3　性能評価

今回, Hybrid OpenCL の性能評価として, NVIDIA NBody [4] アプリケーションによる実験を行った．NBody は多体問題を計算するプログラムで，N 体の質量を持つ粒子間の重力を計算し粒子の運動をシミュレートする．NBody では 1 ステップ計算を進めるごとに clSetKernelArg() clEnqueueNDRangeKernel() clEnqueueReadBuffer() という 3 つの OpenCL ランタイム API を呼び出す．clSetKernelArg() はカーネル (計算デバイス上で動作するプログラム) への引数の設定を行う API，clEnqueueNDRangeKernel() はカーネルの実行を指示する API，clEnqueueReadBuffer() はバッファの読み込みを行う API である．ただし，clSetKer-

---

Development of Hybrid OpenCL for Transparently Different OpenCL Implementations
†Ryo Aoki　†Shuichi Oikawa
†Department of Computer Science, University of Tsukuba
‡Ryoji Tsuchiyama　‡Takashi Nakamura
‡Fixstars Corporation

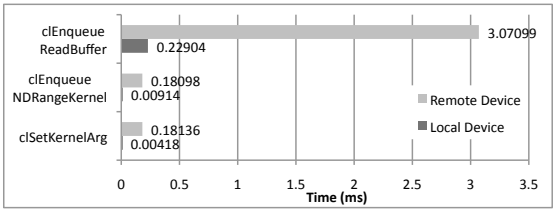

図 2　単一マシンでの API 処理時間

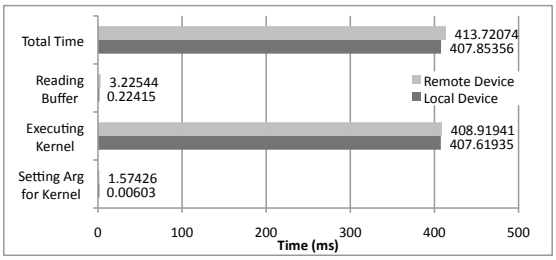

図 3　単一マシンでの NBody の計算時間（1 ステップ）

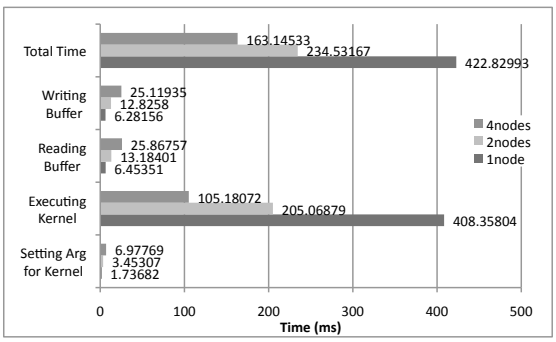

図 4　複数マシンでの NBody の計算時間（1 ステップ）

nelArg() は 1 つの引数しか一度に設定できない．これらの単一 API の処理時間とともに，各行程の全体の処理時間を計測した．これは，NBody におけるカーネルが引数を 9 つ持つため，clSetKernelArg() API は 9 回呼び出される点．また，clEnqueueNDRangeKernel() は単にカーネルの実行を指示するコマンドをコマンドキューに入れるだけで，実際の計算終了を OpenCL の同期機構を用いて待つ必要がある点から別途計測する必要がある．今回は Hybrid OpenCL の基本性能を計測するため，単一マシンでの評価と，スケーラビリティの確認を行うため，複数マシンでの評価を行った．

評価環境としては表 1 のスペックの PC を実験に必要な台数に合わせて使用した．

表 1　評価マシンスペック

Machine	Dell Precision 490
CPU	Xeon 5130 2.0GHz
Memory	2GB
NIC	Broadcom NetXtreme BCM5752 (rev 02) 1Gbps
OS	CentOS 5.5 (Linux 2.6.18)

評価結果を図 2，図 3，図 4 に示す．図 2，図 3 は単一マシンのみ使用した時の各 API 処理時間を計測したグラフと，NBody の 1 ステップごとの処理時間の平均を表したグラフである．また，図 4 は複数マシンで分散処理した際の NBody の 1 ステップごとの処理時間の平均を表したグラフである．

図 2 からネットワークを経由するリモートデバイスではローカルデバイスに比べ速度が低下している．しかし，今回の NBody ではカーネルの処理時間が比較的大きいため，全体のパフォーマンスにはさほど影響を与えていないことが図 3 からわかる．図 4 の複数マシンによる分散処理では，台数が 2，4 台と増えるに従い，カーネルの処理時間が約半分になっており，スケールアウトしていることが確認できた．しかし，粒子データの通信を行う Read/Write Buffer はマシンごとにシーケンシャルに読み書きを行っているため，台数が増えると処理時間が増加していおり，スケールアウトの効果を抑制してしまっている．

4　考察とまとめ

今後パフォーマンスを向上させるには API のレイテンシを押さえるか，Broadcast 通信や AllGather 通信等を実装する必要があると考えている．レイテンシは OpenCL ランタイム API の戻り値や出力引数を同期的に取得しているため起こる問題である．これは OpenCL API が全てがエラーコードを返す仕様になっているためである．改善案として，リモートホスト上の OpenCL ランタイムにエラーチェックをさせずに，ローカルの OpenCL ランタイムにエラーチェックを送信前に行わせることで，エラーコードの取得を待たずに非同期に API を実行していく方法が考えられる．これにより，clSetKernelArg() などのエラーコードのみが出力となっている API に関しては非同期に実行可能となる．また，Broadcast 通信や AllGather 通信は OpenCL API では仕様に含まれないため，独自に実装する必要がある．すでにこれらの通信が実装されている MPI(Massage Passing Interface) を組み合わせれば，OpenCL を拡張する必要はないが，MPI との比較実験の結果，速度は同程度であり，コードが非常に書きづらく，可読性や保守性が悪いため，Hybrid OpenCL の拡張として実装するメリットは十分にあると考えている．今後はこれらの問題を解決することで，低レイテンシ化や，高効率化を行い，Hybrid OpenCL システム全体のパフォーマンスを向上さつつ，異なる種類のデバイスを組み合わせた環境での実験を行っていきたい．

参考文献

[1] Aaftab Munshi: “The OpenCL Specification” , Khronos OpenCL Working Group, 2009.

[2] Aaftab Munshi: “OpenCL Parallel Computing on the GPU and CPU” , SIGGRAPH ’08, 2008.

[3] Copyright Fixstars Corporation, “FOXC (Fixstars OpenCL Cross Compiler),” http://www.fixstars.com/foxc/, accessed Aug. 10. 2010.

[4] Lars Nyland, Mark Harris, Jan Prins: “ Fast N-Body Simulation with CUDA” , GPU Gems 3 - Chapter 31, 2008.