Ruby を用いた分散 GPGPU フレームワーク『ParaRuby』

中村 涼
Ryo NAKAMURA

1　はじめに

Graphics Processing Units(GPU) を用いて汎用目的の並列処理を行う手法 (General-purpose computing on graphics processing units; GPGPU) が，近年広く利用されるようになっている [1) 2) 3)]．GPU は CPU と比べて単純ながらも多数のコアを持ち，並列性の高い問題に対して価格あたり，および消費電力あたりの性能が優れている．

しかし，GPU を用いてプログラミングを行うには，並列処理やデバイスのメモリ構造などのアーキテクチャに関する専門知識や，各環境独自のプログラミングモデルの学習など，実装に高度な技術を要する．

本研究では，GPGPU プログラミングを支援することを目的として，Ruby を用いた分散 GPGPU フレームワーク ParaRuby を提案する．このフレームワークにより，ネットワークを介した複数ノードの GPU による分散処理が容易に実行可能になる．サーバの GPU 上での処理は Ruby のメソッドとしてクライアント上から呼び出せるようになり，CPU/GPU 間やクライアント/サーバ間のデータ転送がフレームワークにより隠蔽される．

ParaRuby を評価するために，モンテカルロ法による多次元球の求積，およびマンデルブロ集合の計算の 2 つのアプリケーションを用い，性能を評価した．

2　GPGPU プログラミング

現在主流となっている GPGPU プログラミング環境としては，GPU ベンダーの NVIDIA が提供する CUDA[4)] や，Apple によって提唱された OpenCL[5)] がある．両環境とも，C 言語とよく似た構文で記述することができ，GPU のメモリ確保，解放，データ転送，その他各種 GPU の操作に関する機能が提供されている．

GPGPU プログラミングでは，CPU 側の処理と GPU 側の処理の区別を注意しておく必要があり，CPU 側をホスト，GPU 側をデバイスと呼ぶ．ホストによる逐次処理の中でデバイスの並列処理を呼び出し，処理結果を再びホストで利用する形となる．ホストとデバイスの処理は明確に分けて記述し，カーネル関数と呼ばれる特殊な関数に記述された処理がデバイスで処理され，それ以外はホストで処理される．

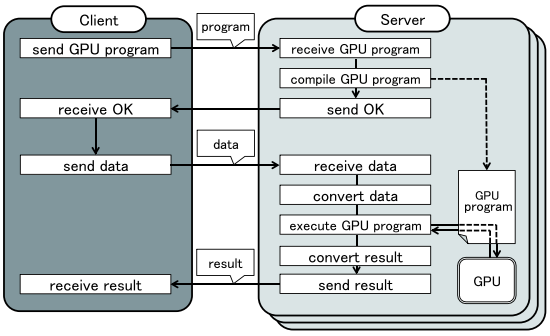

Fig. 1　フレームワークの構造

3　ParaRuby フレームワークの提案

ParaRuby は，Ruby を利用した分散 GPGPU フレームワークである．GPU を搭載したサーバで予めプログラムを動作させ処理を待ち受けておき，クライアントからサーバに対して処理を委譲し，処理結果をクライアントで利用する仕組みを支援する．これにより，GPU を持たないマシンからの GPU 処理や，複数サーバを利用した分散処理が実行可能になる．

フレームワークの構造を Fig.1 に示す．ParaRuby では，クライアントで実行するプログラムは Ruby で記述し，サーバで実行するプログラムは OpenCL C 言語または CUDA C 言語で記述する．サーバで実行するプログラムを Ruby プログラムの中に文字列として記述し，クライアントからサーバに送信することでサーバ上の GPU で処理を行い，結果をクライアントで利用する形となる．

4　評価

ParaRuby を用いた GPU 処理性能を評価するため，モンテカルロ法による多次元球の求積，およびマンデルブロ集合の計算の 2 つのアプリケーションで実行時間を評価した．評価に用いたマシンの環境を Table1 に示す．

4.1　モンテカルロ法による多次元球の求積

モンテカルロ法による多次元球の求積では，10^5 要素の座標点に対してスレッドを割り当て，4 つの実装方法による実行時間の違いを比較した．モンテカルロ法による多次元球求積の実行結果を Fig.2 に示す．

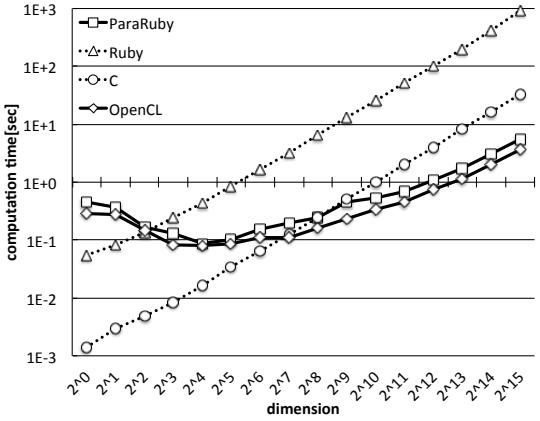

Fig. 2　多次元球求積の実行結果

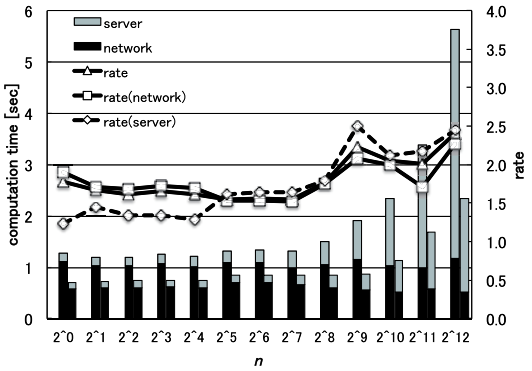

Fig. 3　マンデルブロ集合計算の実行結果

Fig.2 の縦軸は実行時間，横軸は球の次元数を表しており，ともに対数軸で表している．Fig.2 を見ると，C 言語は Ruby よりも高速なものの，グラフの傾きはどちらも同程度である．一方，ParaRuby や OpenCL により GPU を利用した実装では傾きが小さく，一定のオーバーヘッドは存在するものの負荷が大きくなるほど GPU の利用が有効であることが分かる．

4.2　マンデルブロ集合の計算

マンデルブロ集合は，複素平面上の集合が作り出すフラクタルであり，複素数列 $\{Z_n | z_{n+1} = z_n^2 + c, z_0 = 0\}_{n \in N}$ という条件を満たす複素数 c が作る集合である．Z の各要素はそれぞれ独立して求められることから，Z_n の値を GPU のスレッドを用いて並列に計算を行うことができる．

マンデルブロ集合の計算では，10^5 画素に対し，異なる n の値で 2 つのサーバで ParaRuby を用いた場合の計算時間を測定した．

マンデルブロ集合計算に要した時間を Fig.3 に示す．図中の縦軸は実行時間，横軸は n に与える値を表し，各 n についてサーバ 1 台の場合を左側の棒，2 台の場合を右側の棒に表している．network はサーバとクライアント間の通信時間，server はサーバ上での実行時間，rate はサーバ 1 台の場合に比べてサーバ 2 台の場合に何倍の速度向上が見られたかを表している．

通信時間と実行時間の内訳から，通信時間は処理の負荷に関係なく一定であり，実行時間は処理の負荷に応じて変化していることが分かる．

また 1.5 倍〜2.5 倍程度の台数効果が見られ，負荷が大きいほど台数効果が大きくなることがことが分かる．

Table 1　実行環境

	Client	Server
CPU	Core i7 1.8GHz	Core 2 Duo 2.26GHz
GPU	-	GeForce 9400
Memory	4GB	4GB
OS	Mac OS X 10.7	Mac OS X 10.7

5　まとめ

サーバ上の GPU を利用した GPGPU プログラミングを容易に実現するために，Ruby からサーバ上の GPU で処理を行う仕組みとして，分散 GPGPU フレームワーク ParaRuby を開発した．このフレームワークにより，複数のリモートノードの GPU を利用した並列処理を実現できる．

このフレームワークを用いて 2 つのアプリケーションで評価を行った結果，クライアントの CPU 上のみで処理した場合と比べ，フレームワークからネットワーク上のサーバの GPU を利用した場合に高い実行速度が得られることを確かめた．また，複数のノードに対して処理を分けて実行することでより高い実行速度が得られることを確かめた．

参考文献

1) 湯川英宜, 平野敏行, 西村康幸, 佐藤文俊. Gpu によるタンパク質高精度静電ポテンシャル計算の高速化. 生産研究, Vol. 61, No. 2, pp. 103–110, 2009.

2) 成瀬彰, 住元真司, 久門耕一. Gpgpu 上での流体アプリケーションの高速化手法：1gpu で姫野ベンチマーク 60gflops 超 (高性能計算とアクセラレータ). 情報処理学会研究報告, 2008-HPC, Vol. 2008, No. 99, pp. 49–54, 2008-10-08.

3) 東竜一, 藤本典幸, 萩原兼一. Gpu の汎用計算環境 cuda による主記憶上の大規模なテキストに対する高速な全文検索の検討, 2008-hpc. 情報処理学会研究報告, 2008-HPC, No. 19, pp. 139–144, 2008-03-05.

4) NVIDIA. Compute Unified Device Architecture Programming Guide, 2007.

5) John E Stone, David Gohara, and Guochun Shi. OpenCL: A parallel Programming Standard for Heterogenous Computing Systems. *Computing in Science Engineering, 12 Issue:3*, pp. 66–73, 2010.