Ruby を用いた分散 GPGPU フレームワークの開発と評価

中村 涼

1 はじめに

GPU(Graphics Processing Units) 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．GPU は CPU と比べ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良い傾向にある．

これらの環境を利用して開発を行うには，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．また，GPU の計算資源を持ったサーバに対してジョブオフロードを行う場合，クライアントとサーバ間のプログラムやデータの通信処理など，多くの処理の実装を行う必要がある．

本稿では，GPGPU プログラミングを容易に行える環境を提供することを目標として，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，サーバの GPU 上での処理を 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．また，ネットワークを介した複数ノードの GPU による分散処理なども容易に実装可能になる．

Ruby を採用した理由としては，手軽なオブジェクト指向プログラミングを実現できる言語であり，シンプルな文法でかつ柔軟性や拡張性が高いこと，移植性が高いことなどが挙げられる．また，Ruby には分散オブジェクトシステムを実現するためのライブラリ dRuby が標準添付されており，ノード間通信を実現するための機能が充実していたことも理由として挙げられる．

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，ParaRuby の利用の有無で実行時間を比較した．また，サーバを 2 台用いて処理を分割した場合の実行時間を比較し，複数サーバを利用時の処理性能を評価した．

2 GPGPU プログラミング

2.1 GPGPU の適用例

GPGPU に関する研究では，GPU の並列処理能力を活かした処理の高速化の事例が多く報告されている．例として，高速フーリエ変換 [3] や，ソーティング [4]，k 近傍探索 [5] などの処理の高速化が研究されている．また，GPU を利用した PC クラスタに関する研究は国内でも積極的に行われている [6]．

2.2 GPGPU プログラミングの特徴

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 や，Apple によって提唱された OpenCL 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(以下デバイス) 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(以下ホスト) による逐次処理の中でデバイスで実行する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する．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．

ホストとデバイスでは基本的にメモリを共有していないため，ホストで用意したデータをプログラムの中でデバイスのメモリに転送し，デバイスでの処理結果を再びホストのメモリに転送する必要がある．デバイスで処理を行う際の大まかな処理の流れは以下の通りである．

1. ホストでデバイスのメモリを確保する
2. ホストのメモリからデバイスのメモリへ，処理に必要なデータを転送する
3. 転送されたデータをデバイスで処理する
4. デバイスのメモリからホストのメモリへ，処理結果のデータを転送する
5. ホストでデバイスのメモリを解放する

このように，デバイスで処理を行うためにはホストとメモリ間のデータ転送のために多くの処理を行う必要がある．

2.3 GPGPU 言語処理系

GPGPU プログラミングの負担を軽減するため，様々な言語処理系が開発されている．

PGI Accelerator[7] では，既存の C 言語または Fortran 言語のプログラムに特殊なコメントであるディレクティブを挿入することで，並列性を持つループ処理が CUDA を用いた GPU 上での並列処理に変換される．Accelerate[8] では，CPU と GPU のプログラムを同じ言語で記述でき，デバイスで並列に計算を行うための高階関数が用意されている．PyCUDA[9]，SGC Ruby CUDA[10] では，それぞれ Python，Ruby のプログラムから CUDA C 言語で記述された GPU 用の関数を呼び出すことができる．Accelerate，PyCUDA，SGC Ruby CUDA の言語処理系では，CPU/GPU 間のデータ転送はプログラム中で明示的に行う必要がある．

3 分散 GPGPU フレームワークの提案

3.1 概要

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くことで，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でも GPU での処理が容易に実行可能になり，また複数サーバを利用した分散処理を容易する．クライアントとサーバ間の通信は TCP/IP ソケット通信により行われ，デバイスで実行するプログラムとその入出力が転送される．

ParaRuby では，ホストで実行するプログラムは Ruby で記述し，デバイスで実行するプログラム (以下カーネル関数) は OpenCL C 言語または CUDA C 言語で記述する．カーネル関数は，Ruby プログラムの中に文字列として挿入して記述する．

ParaRuby ではデバイスとのプロキシとなるクラスを提供しており，このクラスのインスタンスをサーバ/クライアント間で共有する．カーネル関数を文字列として共有インスタンスに与えることで同名のインスタンスメソッドが定義され，通常の Ruby のメソッドと同様に扱えるようになる．このインスタンスメソッドを呼び出すと，引数の値が内部的にサーバに転送され，カーネル関数として実行された結果がメソッドの結果として返される．これにより，サーバやデバイスとの通信はメソッド呼び出しという形で隠蔽され，カーネル関数の定義部分以外は通常の Ruby プログラムと同様に記述することができる．

3.2 利用例

ParaRuby を利用して GPGPU プログラミングを行う例として，単純な積和演算を行うプログラムを List1 に示す．CUDA および OpenCL のどちらでも記述が可能だが，List1 では OpenCL を用いてカーネル関数を定義する場合の例を示す．Ruby に標準で付属している dRuby の機能を利用しており，クライアント側でライブラリ等のインストールを行う必要はない．calculate というカーネル関数を定義した文字列を渡すことで，calculate という Ruby のインスタンスメソッドが利用できるようになっている．要素数 n の配列を引数として渡すことで，GPU 上で配列の要素数分のスレッドが動作し，calculate で定義した計算が行われる．

List 1 ParaRuby を利用したコード例

```
1  require 'drb/drb'
2
3  # サーバアドレスを指定してインスタンスを共有
4  agent = DRbObject.new_with_uri('druby
       ://1.2.3.4:5678')
5
6  # カーネル関数を定義
7  agent.program <<EOS
8  __kernel void calculate(
9    __global int *a,
10   __global int *b,
11   __global int *c,
12   __global int *d)
13 {
14   // スレッドID を取得
15   int id = get_global_id(0);
16
17   // a = b * c + d を実行
18   a[id] = b[id] * c[id] + d[id];
19 }
20 EOS
21
22 # 要素数n の配列を用意
23 n = 1_000_000
24 a = Array.new(n)
25 b = (1..n).to_a
26 c = (1..n).to_a
27 d = (1..n).to_a
28
29 # カーネル関数で定義した処理を実行
30 a = agent.calculate(a, b, c, d)
```

4 評価

ParaRuby を用いて GPU で処理を実行した場合の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．評価に用いた環境を Fig.1 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 万要素の座標点を利用し，異なる次元の球で求積を行った場合の実行時間を測定した．GPU では，各座標点ごとにスレッドが割り当てられて処理される．実装言語による比較を行うため，CPU で実行するプログラムの実装には Ruby と C 言語を用いた．モンテカルロ法による多次元球求積の実行結果を Fig.1 に示す．

Fig.1 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．図中の CPU は Ruby 実装によるプログラム，CPU(C) は C 言語実装によるプログラム，ParaRuby は ParaRuby 実装によるプログラムの実行時間を示している．次元数が低い場合の結果を見ると，GPU を利用する場合に比べ利用しない場合の方が実行時間が短くなっている．一方，次元数が大きい場合の結果を見ると，GPU を利用する場合の方が実行時間が短くなっている．これは，次元数が低い場合は GPU で処理を実行するためのオーバーヘッドが支配的な大き

Table1 実行環境

	クライアント	サーバ
CPU	Core i7 1.8GHz	Core 2 Duo 2.26GHz
GPU	-	GeForce 9400

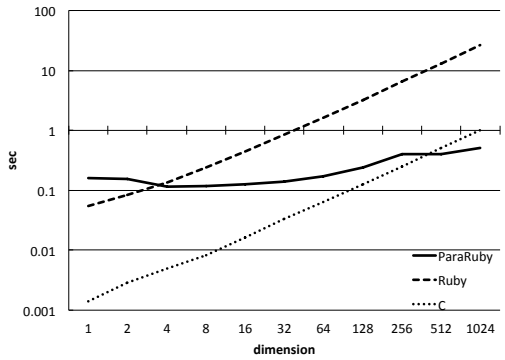

Fig.1　多次元球求積の実行結果

さとなることが原因だと考えられる．モンテカルロ法による多次元球の求積では，クライアントとサーバ間で値の転送がほとんど行われず，オーバーヘッドの大きさは問題の複雑さとは関係しないため，次元数が大きくなるほど計算部分の実行時間が増大していく．次元数が大きい場合では，計算部分の実行時間が支配的な大きさとなり，GPU を利用する場合の方が実行時間が短くなったものと考えられる．

CPU の実装に C 言語を用いた場合でも，Ruby を用いた場合に比べて幾らかの速度向上が見られるが，次元数が大きくなるほど GPU の方が実行時間が短くなり，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，特定の漸化式を満たす複素平面上の点の集合である．プログラムの実行により，Fig.2 のようなマンデルブロ集合の画像が生成される．

マンデルブロ集合の計算では，100 万画素に対し，異なる試行回数 (発散までの試行回数) でマンデルブロ集合を計算した場合の実行時間を測定した．また，サーバを複数利用した場合の効果を調べるため，2 台のサーバを用いて実行した場合の実行時間も測定した．

マンデルブロ集合の計算の実行結果を Fig.3 に示す．Fig.3 の縦軸は実行時間，横軸は各画素での発散するまでの試行回数を示しており，ともに対数軸で表している．図中の CPU は Ruby 実装によるプログラム，1GPU はサーバ 1 台を用いた ParaRuby 実装によるプログラム，2GPUs はサーバ 2 台を用いた ParaRuby 実装によるプログラムの実行時間を示している．2GPUs ではサーバを 2 台用いるため，計算する画素を 2 つに分割し，それぞれのサーバで分散して画素を処理している．

前述した多次元球の求積に比べ，マンデルブロ集合の計算では CPU に比べ GPU の方が常に実行時間が短い結果となっている．これは，発散するまでの試行回数が低い場合でも計算部分の実行時間が大きく支配的であるためだと考えられる．GPU を利用すると，試行回数が低い場合では数倍，試行回数が大きい場合では百倍程度実行時間が短くなっており，処理の負荷が増すほど GPU の利用が有効であることが分かる．

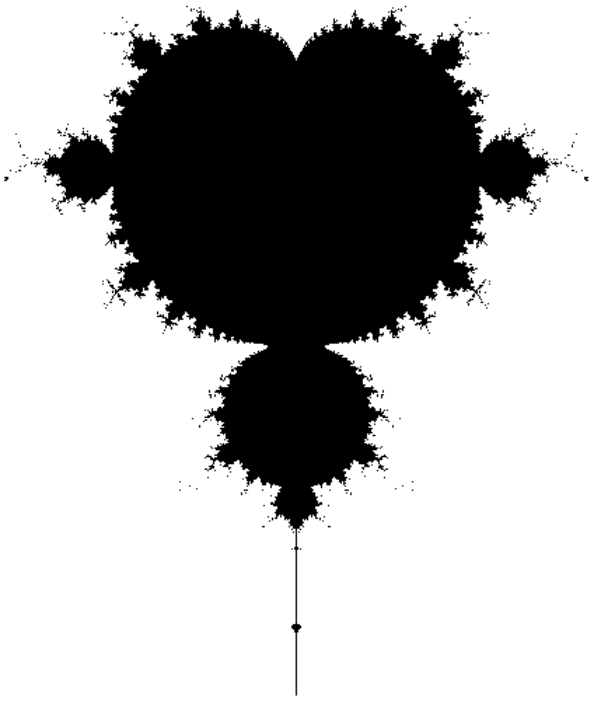

Fig.2　マンデルブロ集合

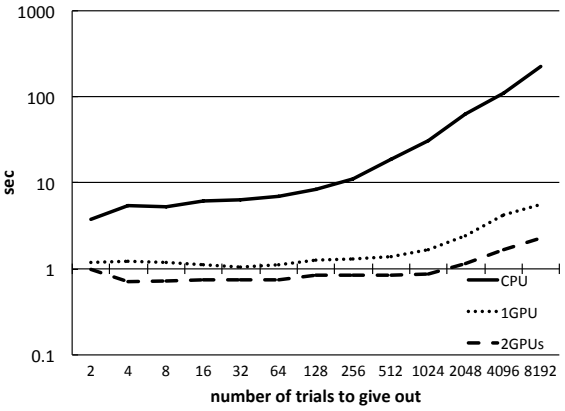

Fig.3　マンデルブロ集合計算の実行結果

また 1GPU と比べ 2GPUs は実行時間が 0.5〜0.7 倍という結果になっており，複数のサーバを利用した場合，単一のサーバを利用する場合に比べて実行速度が向上することが確認できた．複数のサーバを用いた場合の効果は，特に試行回数の増加により負荷が増加した場合に得られた．1 台のみサーバを利用する場合と比べて単純に実行速度の向上が見られ，複数サーバの利用による性能向上が確認できた．

5　関連研究

5.1　GPGPU 処理系

GPGPU プログラミングを容易に行うための処理系は多数存在しており，これらは 2 章で述べた通りである．

5.2 Ruby の GPGPU フレームワーク

Ruby を用いて GPGPU プログラミングを行うことを扱った研究として，Ikra[11] がある．Ikra は，GPU のメモリに実体がある配列を Ruby のクラスとして提供し，配列に対する繰り返し処理などのイテレータを GPU 上で並列実行可能にするものである．Ikra は Ruby を利用した処理系として共通点があるが，Ikra では特定の処理のみが GPU の処理に変換される一方，ParaRuby では記述した任意の処理を実行するという点で異なっている．

5.3 GPU での分散処理

複数の GPU ノードを利用した並列実行環境としては，OpenCL を利用したミドルウェアの実装が提案されている [12]．ネットワーク上の複数のノードに搭載された OpenCL アクセラレータを仮想的に 1 つのホストに搭載されているように見せることでノード間通信を隠蔽し，OpenCL のみで複数のノードを利用した分散 GPGPU プログラミングを実現するものである．GPU を搭載した複数ノードを利用できる点で共通点があるが，複数ノードを仮想的に扱える一方，提供されている関数しか扱えない点や，ホスト側も OpenCL に合わせた言語で記述するという点で異なっている．

6 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 に処理を委譲する仕組みとして，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を用いることで，リモートノードの GPU を利用した並列処理を容易に実現することができる．またクライアント自身をサーバとして利用することや，複数のサーバに分散して処理を委譲することも可能である．

このフレームワークを用いて幾つか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て，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．

今後の課題として，複数サーバに処理を委譲する場合に自動でタスクスケジューリングを行うことや，GPU での計算粒度を変更可能にすること，定型的な処理については Ruby のみで処理を記述できるようにすることなどを検討している．

参考文献

1) CUDA: http://www.nvidia.com/object/cuda_home_new.html
2) OpenCL: http://www.khronos.org/opencl/
3) Kenneth Moreland, and Edward Angel, The FFT on a GPU, In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, Eurographics Association, pp.112-119, (2003),
4) Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey, FAST Sort on CPUs and GPUs: A Case for Bandwidth Oblivious SIMD Sort, ACM, SIGMOD, pp.351-362 (2010)
5) Vincent Garcia, and Frank Nielsen, Searching HighDimensional Neighbours:CPU-Based Tailored Data-Structures Versus GPU-Based Brute-Force Method, MIRAGE 2009, LNCS 5496, pp.425-436 (2009)
6) T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori and M. Taiji, 42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence, Conference on High Performance Networking and Computing, Gordon Bell nalists (2009)
7) PGI — Resources — Accelerator: http://www.pgroup.com/resources/accel.htm
8) Chakravarty M.M.T, Keller G, Lee S, McDonell T.L, and Grover V., Accelerating Haskell array codes with multicore GPUs, In Proceedings of the sixth workshop on Declarative Aspects of Multicore Programming, DAMP ’11, pp. 3-14 (2011)
9) PyCUDA — Andreas Klckner’s web page: http://mathema.tician.de/software/pycuda
10) RubyForge: SGC-Ruby-CUDA: Project Info: http://rubyforge.org/projects/rubycuda
11) 西口裕介, 増原英彦, GPU 汎用計算を配列イテレータとして記述する Ruby 言語処理系の提案, 日本ソフトウェア科学会全国大会第 28 回大会論文集, 1D-2 (2011)
12) 設樂明宏, 鎌田俊昭, 山田昌弘, 西川由理, 吉見真聡, 天野英晴, OpenCL 互換アクセラレータのマルチノード環境における開発負担軽減のためのミドルウェアの実装, 情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol.2010-HPC-128, No.22, pp.1-8 (オンライン), 入手先 http://ci.nii.ac.jp/naid/110007995560 (2010-12-09)