2008－AL－120 (5)
2008／11／7

GPGPUによるGroverのアルゴリズムのシミュレーション

芝田 浩[†1 †4] 西野 哲朗[†2] 大久保 誠也[†3] 鈴木 智也[†1]
[†1] 電気通信大学大学院 電気通信学研究科 情報通信工学専攻
[†2] 電気通信大学 電気通信学部 情報通信工学科
[†3] 静岡県立大学 経営情報学部 経営情報学科
[†4] 広島商船高等専門学校 電子制御工学科

あらまし: 本論では，Groverの量子探索アルゴリズムの並列シミュレーション技法について取り扱う．Groverのアルゴリズムは代表的な量子探索アルゴリズムである．一方，量子アルゴリズムの効率的シミュレーションに関する研究は，量子計算の原理や振る舞い，その応用分野を理解する上で大変重要である．そこで，GPGPUを用いて，Groverのアルゴリズムの並列シミュレータを実現した．さらに，同一のアルゴリズムに対して，OpenMPを使用したシミュレーションを行い，実行時間の比較を行った．計算機実験の結果，OpenMPを使用した場合に比べ，GPGPUを使用したシミュレータは，最大で3.5倍の速度向上を達成できた．

Parallel Simulation of Grover's Algorithm Using GPGPU

Hiroshi Shibata[†1 †4] Tetsuro Nishino[†2] Seiya Okubo[†3] Tomoya Suzuki[†1]
[†1]The Graduate School of Electro-Communications,The University of Electro-Communications
[†2]Department of Information and Communication Engineering, The University of Electro-Communications
[†3]School of Administration and Informatics,The University of Shizuoka
[†4]Department of Electronic Control Engineering,Hiroshima National Colleg of Maritime Technology

Abstract: In this paper, we deal with parallel simulation methods of Grover's quantum search algorithm. Grover's algorithm is one of a well-known quantum search algorithms. On the other hand, the research on the efficient simulation of quantum algorithms is very important in order to understand the principle, behavior and application fields of quantum computing. We implemented a parallel simulator of Grover's search algorithm using GPGPU. Furthermore, we executed another parallel simulation of the algorithm using OpenMP and compared the execution time with that of GPGPU. As a result of computational experiments, by using GPGPU, the execution time of Grover's algorithm can be improved to 3.5 times as fast as that of OpenMP experiments.

1 はじめに

計算という概念を形式的に定義するために，1936年にA.Turingは**Turing機械** (Turing Machine) というモデルを提案した．Turing機械は計算の本質を抽象化しており，現在の計算機の標準的なモデルとなっている．

1985年に，D.Deutschは，量子力学に基づいた計算機である量子計算機のモデル化を行った[3] [4]．このとき提案された量子Turing機械は，Turing機械に量子並列計算機能を取り入れたものである．1994年にP.W.Shorは，整数の因数分解を多項式時間内に高い成功確率で行う量子アルゴリズムを示した[7]．また，1996年にはL.K.Groverが，データベース検索に関する効率的量子アルゴリズムを提案した[5]．このように，量子Turing機械は通常のTuring機械と比べて高速に計算を行うことができる可能性がある．

Groverのアルゴリズムについては様々な研究が行われており，アルゴリズム実行中にノイズが混入した場合の理論的研究[2] [8]や，そのシミュレーションに関する研究[1] [6]もなされている．シミュレーションに関する研究においては，任意の初期状態作成時に混入するノイズや，ユニタリ変換適用時にデコヒーレンスによる量子状態の崩壊のノイズが取り扱われている．また，研究[1]では，OpenMPを用いて効率的な並列処理シミュレーションを行っている．

量子アルゴリズムにおけるシミュレーションに関す

る研究は，量子計算の原理や振る舞いの理解，またその応用研究の道具として重要である．その際，大規模な計算機を使用すること無しにシミュレーションすることができる環境があれば実現の研究に寄与できる．また，Grover のアルゴリズムは，代表的な量子アルゴリズムであり，直感的に理解しやすく，実現の研究にも使用される場合がある．

本研究では，Grover のアルゴリズムのより高速なシミュレーションを行うため，近年注目されている GPU(Graphics Processing Unit) を使用した計算手法である GPGPU(General Purpose Computation on Graphics Processing Unit) を用いた並列処理シミュレータを構築した．そして，大規模計算機における OpenMP API (OpenMP Application Program Interface 以下 OpenMP と略す) を使用した並列シミュレーションと比較を行い，有効性の検証を行った．

2 諸定義

2.1 量子計算

量子 Turing 機械 (Quantum Turing Machine) は，通常の Turing 機械に量子並列計算機能を付加したものである．量子 Turing 機械では，テープ上の 1 つの区画に 0 と 1 の任意の重ね合わせを保持することが可能である．

定義 1 [10] 量子 Turing 機械 M は，以下の条件を満たす 7 項組 $\langle Q, \Sigma, \Gamma, \delta, q_0, B, F\rangle$ である．ただし，

1. Q は**状態**の有限集合，
2. Γ は**テープ記号**の有限集合，
3. $B \in \Gamma$ は**空白記号**，
4. $\Sigma \subseteq \Gamma - \{B\}$ は**入力アルファベット**，
5. $q_0 \in Q$ は**初期状態**，
6. $F \subseteq Q$ は**受理状態**の集合，
7. $\delta : Q \times \Gamma \times \Gamma \times Q \times \{L, R\} \to C$ (C は複素数全体の集合) は，M が次に行うべき 1 ステップの動作を指定する**状態遷移関数**とする．

$\delta(p, a, b, q, d) = c$ は M が状態 p で記号 a を読んでいるとき，状態 q に移り，記号 b を書き込み，ヘッドが方向 d に 1 区画移動するという事象の確率振幅を表している．ここで，このように遷移する確率は確率振幅の絶対値の 2 乗となる．

また，量子 Turing 機械においては，この状態遷移関数から誘導される状態遷移行列 M_δ が，ユニタリ行列でなければならない．ここで，ユニタリ行列とは，以下の式を満たす行列として定義される．

$$MM^\dagger = M^\dagger M = I$$

ただし，$M^\dagger$ は行列 M の転置共役行列であり，I は単位行列である． □

ここで，1 つの区画が保持できる情報量の単位を qubit (quantum bit の略）と言う．

2.2 Grover のアルゴリズム

Grover のアルゴリズムは著名な量子探索アルゴリズムである．ここで，Grover のアルゴリズムが対象とする探索問題とは，以下のような問題である．

入力 : N （$N = 2^n$）

問題 : N 個の状態が x_1 から x_N でラベル付けされている．各ラベル x_i は 2 進数で表現されており，整数としても取り扱うこととする．量子オラクル $C : \{x_1, ..., x_N\} \to \{0, 1\}$ が与えられたときに，$C(x) = 1$ を満たす状態 x を一つ発見しなさい．

ここで，量子オラクルは，$|x_1, 0\rangle + |x_2, 0\rangle + \cdots + |x_N, 0\rangle$ を入力すると，単位ステップで $|x_1, f(x_1)\rangle + |x_2, f(x_2)\rangle + \cdots + |x_N, f(N)\rangle$ を返す．

論文 [5] では，以下のアルゴリズムが示されている．

1. 量子メモリ・レジスタを，各状態がすべて同じ振幅を持つ重ね合わせ
   $\left(\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}, \cdots, \frac{1}{\sqrt{N}}\right)$
   になるように初期化する（ここで，各状態に対する振幅を並べてベクトルとして表記している．このような表記を**状態ベクトル**と呼ぶ）．
2. 以下のユニタリ変換を $O(\sqrt{N})$ 回繰り返す．
   (a) 量子メモリ・レジスタが状態 x にあるとする．
   $f(x) = 1$ の場合は**位相反転**を適用する．
   $f(x) = 0$ の場合は何も行わない．
   (b) 以下のような行列 D により定義される**拡散変換** D を適用する．
   $$D_{ij} = \frac{2}{N} \text{ if } i \neq j \text{ and } D_{ii} = -1 + \frac{2}{N}.$$
3. 最終的に得られた状態を観測する．すると，少なくとも 0.5 以上の確率で状態 x が観測される．

上のステップ 2 の部分が Grover のアルゴリズムの核心であり，これによって所望の状態の振幅を $O\left(\frac{1}{\sqrt{N}}\right)$ ずつ増やすことができる．したがって，ステップ 2 を $O(\sqrt{N})$ 回繰り返すことにより，所望の状態を得る確率を 1 に近づけることができる．

3 並列処理プログラミング

3.1 OpenMP を用いた並列処理

OpenMP は，共有メモリ型マルチプロセッサアーキテクチャに対する移植性のある並列プログラミングモデルである．OpenMP の仕様は，多数のコンピュータベンダが共同で開発し，OpenMP Architecture Review Board で作成，発行されている．この仕様には，コンパイラに対する指示文，関数やサブルーチン，また実行時の環境変数が含まれている．ユーザは，逐次処理を行うプログラムを記述し，並列処理させたい部分にだけ指示文（ディレクティブ）を記述すればよい．図 1 に，for 文を用いて 10000 回の繰り返しを行うプログラムの例を示す．図 1 中の，`#pragma omp parallel for` がディレクティブである．これにより，for 文を並列処理できる．このように，OpenMP を用いると，並列プログラムを容易に記述することができる．しかしながら，並列化による効率はコンパイラに依存し，また具体的な並列処理方法を詳細に指定することもできない．

```
#pragma omp parallel for
  for(i=0;i<=10000;i++){
    ...
  }
```

図 1: OpenMP による並列プログラム例

3.2 GPGPU を用いた並列処理

GPU は，グラフィックス処理を支援する半導体チップであり個人用コンピュータや家庭用ゲーム機に搭載されている．近年の GPU は CPU に比べて高い浮動小数点数演算能力を有している．両者の浮動小数点数演算性能の年代推移を図 2 に示す．近年，GPU の性能向上に伴い GPU に汎用的な計算を行わせる GPGPU という手法が注目されている．

しかし，従来の GPGPU においては，GPU のハードウェア・アーキテクチャや，グラフィックス処理に用いられ

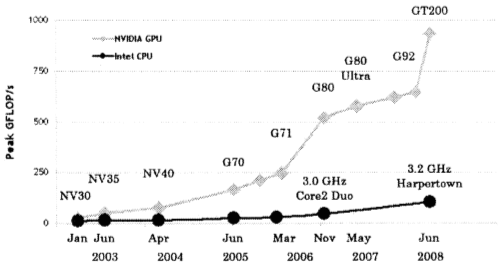

図 2: GPU と CPU の浮動小数点数演算性能

るシェーダ言語に関する知識が要求されることが障害となっていた．その障害を解消するため，NVIDIA 社により，GPU 上での汎用計算を行わせるための新しいハードウェア・アーキテクチャ及び，CUDA(Compute Unified Device Architecture) と呼ばれるソフトウェア・プラットフォームが開発された [9]．この CUDA の登場によって，シェーダ言語で記述する必要があった GPGPU プログラムを，C 言語に似た言語仕様で記述することが可能となった．なお，本研究ではこの CUDA を用いてシミュレータの作成を行った．

本研究で使用した NVIDIA 社製 GPU は，ストリーミング・マルチプロセッサ (Streaming Multi Processer , 以下 SM と略す) と呼ばれる回路ブロックが複数並んだ構成になっている．各 SM は，8 個のスカラ・プロセッサ (Scalar Processer , 以下 SP) を搭載している．複数の SM 及び SP を並列動作させることで，高い演算性能を得ることができる．CUDA を用いて GPU を制御する場合，GPU を搭載したコンピュータを「ホスト (Host)」と呼び，GPU を「デバイス (Device)」と呼ぶ．本研究で使用した GPU のハードウェア概念図を図 3 に示す．

CUDA の実行モデルでは，以下の処理が繰り返される．

1. ホストがメインメモリ上のデータを GPU 上のメモリである VRAM に転送．
2. ホストが GPU に対して関数の実行を指示し，GPU が処理を実行．
3. GPU の演算が終了した後，ホストが VRAM からメインメモリへ演算結果などのデータを転送．

CUDA では，並列実行の単位として，スレッド (thread) とブロック (block) を使用する．スレッドは SP で実行する処理単位であり，各ブロックはスレッド配

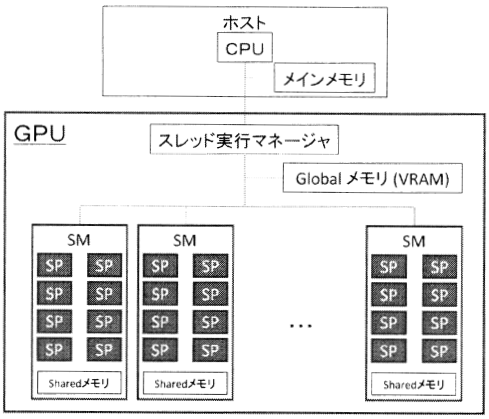

図 3: GPU のハードウェア概念図

表 1: CUDA で扱うメモリの一例

分類	説明
Global メモリ	全ブロック及び全スレッドから読み書き可能（低速, 大容量）
Shared メモリ	各 SM が独立に持ち，同一のブロックで管理されたスレッドからのみ共有メモリとして読み書き可能（高速, 低容量）

列の一部を管理し，これが SM の処理単位となる．プログラム上では，スレッド及びブロックに付けられたインデックスを用いることにより実行制御を行う．また，スレッド実行マネージャによって，スレッドの発行，実行順序などの管理を行っている．各 SM が各ブロックの実行を担当し，SM に搭載された各 SP がブロックで管理する各スレッドを実行することで並列処理を実現している．

また，CUDA を用いる際には特徴の異なるいくつかのメモリを使用することができる．その一例を表 1 に示す．

4 構築したシミュレータ

4.1 シミュレータ概要

量子状態は非常にノイズに弱いため，量子アルゴリズムを実行する際には，この影響を考慮する必要がある．本研究では，初期状態作成時に混入するノイズと，ユニタリ変換適用時にデコヒーレンスによる量子状態の崩壊のノイズが生じた場合のシミュレーションを行った．

本研究で作成したシミュレータの処理の流れは，以下の通りである．

1. 2^n 個の状態を作成する．
2. すべての状態の振幅を $\frac{1}{\sqrt{2^n}}$ にする．
3. ノイズとして，乱数値をすべての振幅に加算する．（初期状態作成時のノイズ）
4. 正規化を行い，状態ベクトルの長さを 1 にする．
5. 以下を繰り返す．
   (a) $f(x) = 1$ となる x の振幅の符号を反転する．
   (b) 拡散変換 D の対角成分を状態ベクトルに適用する．
   (c) 拡散変換 D の非対角成分を状態ベクトルに適用する．
   (d) ノイズとして，乱数値をすべての振幅に加算する．（デコヒーレンスによるノイズ）
   (e) 正規化を行い，状態ベクトルの長さを 1 にする．

本シミュレーションにおいては，各量子状態の振幅の実数成分と虚数成分に対して，一様にランダムな –1 以上 1 以下の実数値を割り当てたものをノイズとして使用した．ランダムな実数値の生成には，周期が長い擬似乱数であるメルセンヌ・ツイスター乱数 (Mersenne Twister) を使用した．

作成したシミュレータが実行する処理のフローチャートを図 4 に示す．OpenMP 及び GPGPU を用いた各シミュレータでは，図 4 の OpenMP/GPGPU で示された部分について並列処理を行っている．具体的には下記の個所である．

1. 乱数値をすべての振幅に加算．（初期状態生成及び，ループ中）
2. 拡散変換 D の適用．
3. 確率振幅の正規化．

これらはすべて配列の各要素に対して同様の計算を施す処理であり，各要素に対する処理を各スレッドで実行する．OpenMP では複数の CPU が，GPGPU では複

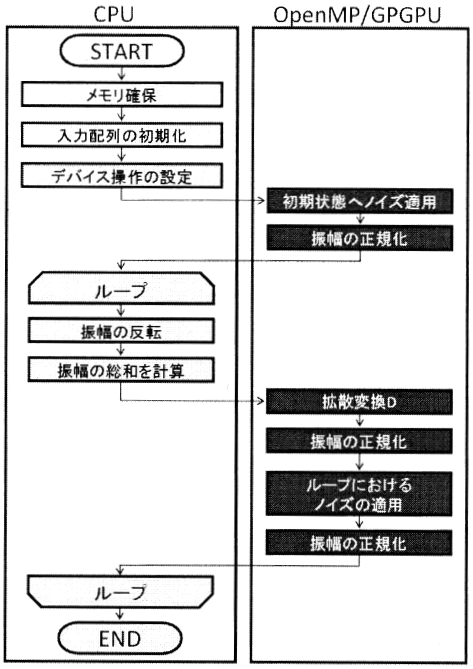

図 4: シミュレータのフローチャート

数の SM 及び SP が複数のスレッドを並列に処理している．

GPGPU を使用したシミュレータの出力画面を図 5 に示す．縦軸に所望の状態の得られる確率，横軸に拡散変換 D の適用回数を示したグラフが出力される．

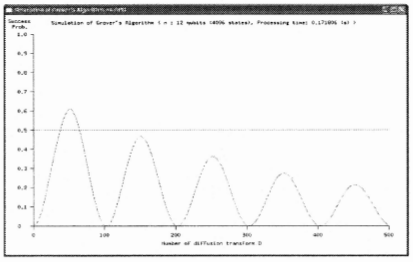

図 5: シミュレータの出力画面

4.2 GPGPU によるシミュレーションの高速化

CUDA の出現により，GPU のハードウェア・アーキテクチャを意識することなく，GPU 上で汎用計算を実現することができるようになった．それに伴い，GPU が本来不得意とする処理についても，難なく実行させることができるようになった．そのため，GPU を生かした効率の良い GPGPU プログラムを作成するには，未だに GPU のハードウェア・アーキテクチャに関する知識や，それに基づくプログラミングテクニックが必要となっている．GPGPU プログラムにおいては，メインメモリと VRAM 間のデータ転送処理が大きなオーバーヘッドとなる事，VRAM の容量が比較的小さい事に留意する必要がある．

上記を踏まえつつ，本研究でプログラムに施した工夫を以下に述べる．

- 拡散変換 D を適用する際に確率振幅の総和を求めることと，振幅を正規化する際に各確率振幅の絶対値の 2 乗の総和を求めることが必要となる．総和計算は，逐次的に各要素を足し合わせる処理であり，並列処理には向いていない．一方，一般的に GPU の SP はホストの CPU と比較して処理能力が低い．そのため，総和計算は GPU で実行するのではなく，CPU で実行することとした．

- メインメモリと VRAM 間のデータ転送処理は大きなオーバーヘッドとなるため，何度も行うと全体として処理時間が長くなってしまう．そのため，メインメモリと VRAM 間のデータ転送を必要最小限のデータのみに抑えた．

- メインメモリから VRAM に転送されたデータは，多くが Global メモリに格納される．Global メモリは大容量であるがデータアクセスに時間を要する．一方，Shared メモリは低容量であるが高速なデータアクセスが可能である．そのため，処理を行う前に，必要な Global メモリのデータを Shared メモリにコピーしておくことで，データアクセス時間を短縮した．

GPU にはメモリの排他処理機能がないため，並列処理を行う際に複数の SP による同一メモリ領域への不正なアクセスなどにより，データの不整合が起こりやすい．そのため，メモリの管理，スレッド及びブロックの指定，スレッドの同期等に留意する必要がある．

5 実験結果

OpenMP の実行環境を表 2 に，GPU の実行環境を表 3 に示す．なお，Sun Studio コンパイラは OpenMP をサポートしている．

表 2: OpenMP の実行環境

項目	内容
CPU	UltraSparc IV+ 1.5GHz
ノード数 (コア数)	44 ノード (88 コア)
メモリ	352GB
OS	Solaris 9
コンパイラ	Sun Studio 12

※国立大学法人電気通信大学情報基盤センターの計算機システムを利用．

表 3: GPGPU の実行環境

項目	内容
CPU	Intel Core2Quad Q9550 2.83GHz
メモリ	4GB
GPU	GeForce GTX 280 ・SM 数: 30 ・SP 数 (全体): 240 ・VRAM: 1GB
OS	Windows XP
開発環境	CUDA 2.0 SDK

OpenMP を用いたシミュレータの実行時間と，GPGPU を用いたシミュレータの実行時間の比較を行った．

各方法の実行時間を，ノイズ無しの結果を図 6（ノイズ無し）と表 4 に，ノイズ有りの結果を図 7（ノイズ有り）と表 5 に示す．ノイズ無し，ノイズ有りの両者の場合において，問題サイズは 12qubit から 25qubit，実行内容は拡散変換 D を 500 回適用するシミューレーションとした．なお，OpenMP の結果については，使用する CPU コア数，つまり使用するスレッド数を 1 スレッドから 32 スレッドまで変化させて測定し，最も実行時間が短かったデータを用いた．

これらの結果より，ノイズ無し，ノイズ有りともに 25qubit までのサイズのシミュレーションにおいて，GPGPU を用いたシミュレータの方が OpenMP を用いたシミュレータよりも処理に要する時間が短いことが分かる．また，GPGPU を用いたシミュレータについては，本研究で用いた GPU を使用することで，ノイズ無しで 26qubit，ノイズ有りで 25qubit の問題サイズまで実行できることを確認した．

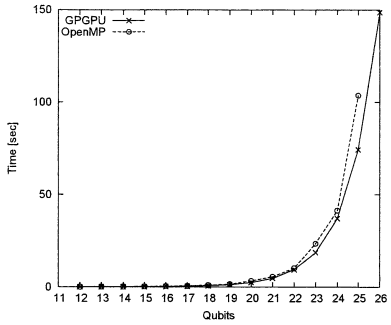

図 6: OpenMP と GPGPU の実行時間（ノイズ無し）

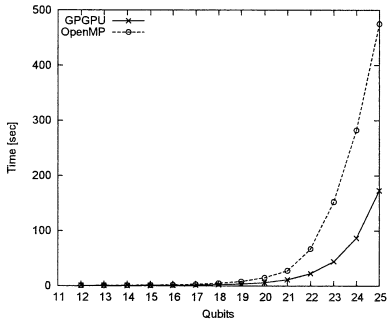

図 7: OpenMP と GPGPU の実行時間（ノイズ有り）

6 考察

6.1 実行時間に関する考察

並列処理において，その性能を引き出すためには，用いる複数のプロセッサすべてが常に稼動し，アイドルタイムがないことが重要である．しかし，実行するアルゴリズムの多くは処理の順番に依存関係があり，すべてのプロセッサを常に稼動させることができない．そのため，並列処理を適用させるアルゴリズムは，各プロセッサを独立に動作させることができることが望ましい．

量子計算機は，qubit に状態の重ね合わせを保持し，その重ね合わされた各状態を並列に処理することが可能である．ある状態の結果が，他の状態の結果に影響

表 4: OpenMP と GPGPU の実行時間（ノイズ無し）

Qubits	OpenMP[sec]	GPGPU[sec]	比率
12	0.09	0.06	1.5
13	0.15	0.07	2.1
14	0.22	0.10	2.2
15	0.3	0.14	2.1
16	0.43	0.23	1.9
17	0.65	0.39	1.7
18	1.01	0.72	1.4
19	1.56	1.29	1.2
20	3.28	2.45	1.3
21	5.57	4.76	1.2
22	10.21	9.42	1.1
23	23.49	18.72	1.3
24	41.45	37.21	1.1
25	103.65	74.32	1.4
26	-	148.65	-

※比率は,「OpenMP の実行時間 / GPGPU の実行時間」で表わされる.

表 5: OpenMP と GPGPU の実行時間（ノイズ有り）

Qubits	OpenMP[sec]	GPGPU[sec]	比率
12	0.39	0.42	0.9
13	0.53	0.46	1.2
14	0.77	0.50	1.5
15	1.21	0.60	2.0
16	1.78	0.80	2.2
17	2.81	1.18	2.4
18	4.79	1.98	2.4
19	7.63	3.33	2.3
20	14.82	6.04	2.5
21	27.69	11.43	2.4
22	67.00	22.18	3.0
23	152.32	43.90	3.5
24	282.21	86.87	3.2
25	475.17	173.00	2.7

※比率は,「OpenMP の実行時間 / GPGPU の実行時間」で表わされる.

することはなく，各状態の計算を独立に処理することができる．そのため，量子計算のアルゴリズムは並列処理に向いていると考えられる．

ノイズ無しの場合，GPGPU を用いたほうが OpenMP を用いるより若干高速になる程度であった．しかし，ノイズ有りの場合は，ノイズ無しの場合と比較して両者の差は広がっている．これは，GPGPU を用いたシミュレータではノイズに用いる乱数の生成を GPU 上で並列に行っていることが要因ではないかと考えられる．乱数が GPU 上で生成されるため，メインメモリと VRAM 間のデータ転送は発生せず，オーバーヘッドを抑えることもできている．OpenMP を用いたシミュレータでも乱数の並列生成を行っているが，並列化の効率はコンパイラに依存してしまうために，乱数の生成を効率的に並列化できなかったものと考えられる．

GPGPU を用いたシミュレータには，4.2 節に示したような工夫を施した．しかし，OpenMP を用いたシミュレータについては，逐次処理プログラムを比較的容易に並列処理プログラムにすることができるパラレル構文を使用して作成した．そのため，プログラムにスレッド単位の詳細な処理内容を記述していない．OpenMP には，各スレッドの処理を比較的詳細に定めるワークシェアリング構文が存在する．これを用いることにより，スレッド単位の処理を詳細に制御することで，実行時間を短縮できる可能性がある．ただし，そうした場合，プログラムが複雑になることが予想される．

6.2 実行環境に関する考察

本研究において，GPGPU を用いたシミュレータでは浮動小数点数データを単精度で計算しているのに対して，OpenMP を用いたシミュレータでは倍精度で計算している．これは，GPU がグラフィックス処理においては，単精度の浮動小数点数演算を使用するため，GPU のプロセッサも単精度の演算を専門としていることによる．そのため，GPGPU を用いたシミュレータの出力には問題のない範囲ではあるが若干の誤差が生じてしまう．また，最新の GPU では倍精度の演算にも対応している．しかし，倍精度の演算は単精度の演算に比べて多くの時間を要してしまう．

GPGPU を用いたシミュレータでは，各状態の確率振幅の値，各ループで加えるノイズに用いる乱数値を VRAM 上に保持する必要がある．そのため，VRAM 容量の制限により，計算可能な問題サイズ数が決定されてしまう．現状では，26qubit のノイズ有りの場合のシミュレーションを実施する場合，約 1GB の VRAM を使用する．GPU に付加されている VRAM の容量は

様々であり，使用デバイスに依存しないシミュレータにするためには，さらにVRAMへのデータ格納方法に改善を加える必要がある．

本研究の目的の1つとして，大規模計算機を使用すること無しに，Groverのアルゴリズムをシミュレーションする環境を実現することがあった．実験結果を踏まえつつ，本研究における2つの手法であるOpenMPとGPGPUを様々な観点から比較する．今回使用したOpenMPの実行環境には大規模計算機を使用したが，これを構築するためには数千万円の費用を要し，また利用する際にも一般的には高額な費用を必要とする．それに対しGPGPUでは，GPUが数万円から購入可能であるため，ホストコンピュータを含めても30万円程度で環境を構築することが可能であり，比較的容易に個人で所有することができる，また，設置面積については，GPUを搭載した個人用計算機に比べて大規模計算機は広大な設置スペースを必要とする．さらに，今回のOpenMPを用いたシミュレータの実行結果は，使用した大規模計算機をほぼ占有することで得られた結果である．一般的な大規模計算機は，占有して利用することは難しい場合が多い．このようにGPGPUを用いるメリットは大きい．加えて，本研究において，シミュレーションの実行時間についてもGPGPUを用いたシミュレータの方がより短時間で実行可能という結果を得ることができた．以上により，GPGPUという手法は量子計算の実現分野の研究に寄与できるものと考える．

7 まとめ

本論では，GPGPUの手法を用いて，Groverのアルゴリズムの並列シミュレータを実現した．計算機実験の結果，OpenMPを用いたシミュレータよりも，GPGPUを用いたシミュレータの方が，25qubitまでの問題サイズにおいて，より高速に並列シミュレーションを実行可能であることが確認できた．これにより，量子アルゴリズムの並列シミュレーションは，GPGPUを用いることで，より高速に実行可能であることを示すことができた．

今回は，OpenMP，GPGPUともに，現時点での最速の実行結果のみについて示した．今後は，両者の手法におけるスレッド数に関する処理効率について，さらに考察を加える必要がある．また，シミュレーションをさらに高速に実行するために，複数のGPUを使用した並列処理についても検討を行う必要がある．

謝辞 本研究のために，国立大学法人電気通信大学情報基盤センターの計算機システムを利用致しました．深く感謝致します．

参考文献

[1] 大久保誠也，西野哲朗: ノイズ環境化におけるGroverのアルゴリズムのシミュレーション，情報処理学会 研究報告, 2008-AL-116 (8), pp. 55-62 (2008)

[2] Biham, E., Biham, O., Biron, D., Grassl, M. and Lidar, D.: Grover's Quantum Search Algorithm for an Arbitrary Initial Amplitude Distribution, quant-ph/*9807027* (1998)

[3] Deutsch, D.: Quantum Computational Netwarks, *Proc. R. Soc. Lond.*, Vol. A400, pp. 97-117 (1985)

[4] Deutsch, D.: Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer, *Proc. R. Soc. Lond.*, Vol. A400, pp. 97-117 (1985)

[5] Grover, R.: Quantum Mechanics Helps in Serching for a Needle in a Haystack, *Physical Review Letters*, Vol. 79, No. 2, pp. 325-328 (1997)

[6] Niwa, J., Matsumoto, K. and Imai, H.: General purpose parallel simulater for quantum computing, *Physical Review A 66*, 062317 (2002)

[7] Shor, P.: Algorithms for Quantum Computation : Discrete Log and Factoring, *in Proceedings of the 35th Anniual IEEE Symposium on Foundations of Computer Science*, pp.124-134 (1994)

[8] Pedro J. Salas.: Noise effect on Grover algorithm, *arXiv:0801.1261v1 [quant-ph]*

[9] NVIDIA Corporation: *NVIDIA CUDA Compute Unified Device Architexture Programming Guide*, Version 2.0 (2008)

[10] 西野 哲朗: 量子コンピュータの理論, 培風館 (2002)