Vol.2009-MPS-76 No.2
Vol.2009-BIO-19 No.2
2009/12/17

GPGPUによるGroverのアルゴリズムの大規模シミュレーションについて

芝田　浩[†1,†2]　鈴木智也[†1]
大久保誠也[†3]　西野哲朗[†4]

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，OpenMP と 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．その際，各方法におけるプログラムの改善により，高速化及びより大きな qubit への対応を実現した．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 1.47 倍の速度向上すると共に，より多くの qubit のシミュレーションを実行することを可能にした．

On Large Scale Simulation of Grover's Algorithm by Using GPGPU

Hiroshi Shibata,[†1,†2] Tomoya Suzuki,[†1]
Seiya Okubo[†3] and Tetsuro Nishino [†4]

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 OpenMP and GPGPU. We improved the execution time and the executable qubit size of simulation by revised of each method. As a result of computational experiments, by using GPGPU, the execution time of Grover's algorithm can be improved to 1.4 times as fast as that of OpenMP experiments. Also the executable qubit size of it can be inproved.

1. はじめに

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．これは，通常の 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．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 のアルゴリズムはその応用性の高さから様々な研究が行われている．その中に，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 4) 7) がある．これらの研究では，任意の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の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．

一方，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非常に重要である．量子計算機のシミュレーションには，多くの計算時間と記憶容量を必要とする．そのため，従来では大型計算機や多くの計算機による並列計算によって行われてきた．しかしながら，近年の情報処理技術の発展により，並列処理の実行環境を個人でも容易かつ身近に利用できる状況になってきている．

本論では，Grover のアルゴリズムの高速なシミュレーションを，二つの手法を用いて実現した．一つは，OpenMP API(OpenMP Application Program Interface，以下 OpenMP と略す) を用いた並列シミュレータであり，もう一つは，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) を用いた並列シミュレータである．その際，より現実に近いシミュレーションにするため，Grover のアルゴリズム実行中のノイズの影響を考慮した．本論では，はじめにそれぞれの実行環境と並列計算の特徴について述べた後，各手法においてシミュレーションを高速化する方法を示す．次に，GPGPU においてより大規模な問題サイズを計算可能にする機能とその計算結果を示す．最後に，OpenMP における実行結果と

†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
The Graduate School of Electro-Communications, The University of Electro-Communications
†2 広島商船高等専門学校 電子制御工学科
Department of Electronic Control Engineering, Hiroshima National College of Maritime Technology
†3 静岡県立大学 経営情報学部 経営情報学科
School of Administration and Informatics, The University of Shizuoka
†4 電気通信大学 電気通信学部 情報通信工学科
Department of Information and Communication Engineering, The University of Electro-Communications

Vol.2009-MPS-76 No.2
Vol.2009-BIO-19 No.2
2009/12/17

GPGPU における実行結果について処理速度を比較することにより，GPGPU を利用した量子計算シミュレータの有効性を示す．

2. 諸 定 義

2.1 量 子 計 算

量子 Turing 機械 (Quantum Turing Machine，以下 QTM と略す) 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．QTM 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 [8] 量子 Turing 機械 M は，7 項組 $\langle Q, \Sigma, \Gamma, \delta, q_0, B, F\rangle$ である．ただし，Q は状態の有限集合，Γ はテープ記号の有限集合，$B \in \Gamma$ は空白記号，$\Sigma \subseteq \Gamma - \{B\}$ は入力アルファベット，$q_0 \in Q$ は初期状態，$F \subseteq Q$ は受理状態の集合，$\delta : Q \times \Gamma \times \Gamma \times Q \times \{L, R\} \to C$ (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，QTM 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．

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 $x_1, x_2 \cdots x_N$ 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．各状態 $x_1, x_2 \cdots x_N$ に対して，$f(x_i) : \{0,1\}^n \to \{0,1\}$ を計算する量子オラクルが与えられたときに，$f(x_i) = 1$ を満たす唯一の状態 x_i を一つ発見しなさい．

ここで，量子オラクルは，$\alpha_1 |x_1, 0\rangle + \alpha_2 |x_2, 0\rangle + \cdots + \alpha_N |x_N, 0\rangle$ を入力すると，単位ステップで $\alpha_1 |x_1, f(x_1)\rangle + \alpha_2 |x_2, f(x_2)\rangle + \cdots + \alpha_N |x_N, f(x_N)\rangle$ 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 $\mathcal{O}(\sqrt{N})$ 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．

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 $\mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$ ずつ増やすことができる．したがって，ステップ (2) を $\mathcal{O}(\sqrt{N})$ 回繰り返すことにより，ステップ (3) で所望の状態を得る確率を 1 に近づけることができる．

3. 並列処理とシミュレーション環境

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り，複数の CPU を搭載した計算機等で利用される．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はできない．

3.2 GPGPU を用いた並列処理

GPGPU とは，GPU に汎用計算を行わせる手法である．ここで GPU とは，グラフィックス処理に特化した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のグラフィックス処理，汎用大型計算機のコプロセッサとして利用されている．GPU は単純な構造のプロセッサを多数搭載したマルチコアプロセッサであり，これらを並列に動作させることで高い浮動小数点数演算能力を達成している．両者の浮動小数点数演算性能の推移を図 2 に示す．このように，GPU は理論的には高い浮動小数点数演算性能を持つため，シミュレーション等のグラフィックス以外の処理へ応用することが盛んに行われている．しかしながら，実際にその性能を引き出すには，並列処理を効率的に行わせるための様々な工夫が必要となることが知られている．また，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るシェーダ言語に関する知識が要求されるこ

Vol.2009-MPS-76 No.2
Vol.2009-BIO-19 No.2
2009/12/17

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

図 1 OpenMP による並列プログラム例
Fig. 1 An example of a parallel program using OpenMP.

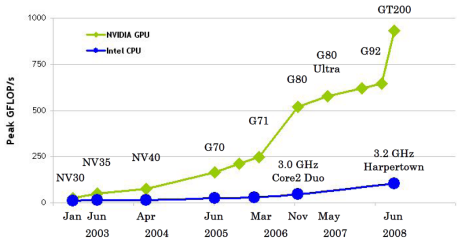

図 2 GPU と CPU の浮動小数点数演算性能
Fig. 2 Floating-Point Operations per Second for the CPU and GPU.

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を作成した．

本研究で使用した NVIDIA 社製の 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) を複数並べた構成になっている．そして，各 SM には 8 個のスカラプロセッサ (Scalar Processer , 以下 SP) が搭載されている．複数の SP 及び SM を使用することで，並列処理を実現している．

CUDA では，並列処理の単位として，スレッド (thread) とブロック (block) を使用する．スレッドは SP で実行する処理の最小単位であり，ブロックは一定量のスレッドをまとめたものである．1 ブロックの処理は，同一 SM で処理され，1 スレッドの処理は 1 つの SP によって処理される．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．実行制御は，スレッド実行マネージャによって管理される．SM 内に搭載された複数の SP が，対応する各スレッドを実行することで，並列処理を実現する．

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．その一例として，GPU に搭載されているビデオメモリ (VRAM) で構成される Global メモリと，各 SM にて保持する Shared メモリがある．

4. 構築したシミュレータ

4.1 シミュレータ概要

量子計算機は量子状態を利用して計算を行うが，この量子状態は非常にノイズの影響を受けやすいことが知られている．したがって，量子アルゴリズムの実行においても，ノイズの影響を考慮することが非常に重要である．そこで本研究では Grover のアルゴリズムについて，初期状態作成時に混入するノイズと，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) すべての状態の振幅を $\frac{1}{\sqrt{2^n}}|x\rangle$ にする．
(2) 初期化時のノイズとして，乱数値を，すべての振幅に加算する．
(3) 以下のように正規化を行い，状態ベクトルの長さを 1 にする．
  (a) すべての振幅の 2 乗和を計算する．
  (b) すべての振幅を 2 乗和で割る．
(4) 以下の変換を繰り返す．
  (a) $f(x) = 1$ となる x の振幅の符合を反転する．
  (b) すべての振幅の平均値を計算する．
  (c) すべての振幅を平均値で折り返す．
  (d) ユニタリ変換適用時とデコヒーレンスによるノイズとして，乱数値を，すべての振幅に加算する．
  (e) 以下のように正規化を行い，状態ベクトルの長さを 1 にする．
    (i) すべての振幅の 2 乗和を計算する．
    (ii) すべての振幅を 2 乗和で割る．

プログラムのステップ (1)〜(3) までがアルゴリズムのステップ (1) に，プログラムのステップ (4) がアルゴリズムのステップ (2) に相当する．ステップ (2) とステップ (4d) ではノイズとして，各量子状態の振幅の実数成分と虚数成分に対して，一様ランダムに $-\alpha$ 以上 α 以下の値を加算している．ここで，α はノイズの影響の強さを決めるパラメータである．また，乱数生成アルゴリズムとしてはメルセンヌツイスタ (Mersenne Twister) を利用している．

4.2 OpenMP: シミュレーションの高速化

OpenMP を用いたシミュレータでは，プログラムのステップ (4b)〜(4e) で使用されている for 文に対して並列処理を適用した．OpenMP を使用することにより，指定した部分を容易に並列処理することが出来る．しかし，その効率はコンパイラに依存するため，素朴な実装をしただけでは，十分なパフォーマンスを得ることは難しい場合がある．そこで，素朴な実装と比較して，以下のようなプログラムの改善を行った．

Vol.2009-MPS-76 No.2
Vol.2009-BIO-19 No.2
2009/12/17

- プログラム中に出現する for 文を減らすために，計算順序の組み替えを行った．具体的には，ステップ (4(e)i) の処理を (4d) の処理と同時に，ステップ (4(e)ii) の処理を (4b) の処理と同時に，それぞれ行った．
- OpenMP の場合は，GPU 等と異なり，コア専用のメモリは準備されておらず，各コアのキャッシュがその役割を果たす．そこで，プログラム全体を見直し，使用するメモリ容量を減少させることで，キャッシュにデータが入りやすいようにした．また，多くの量子ビットのシミュレーションが可能となった．

4.3 GPGPU: シミュレーションの高速化

GPGPU を用いたシミュレータでは，プログラムのステップ (2)，(3) 及び (4b)〜(4e) の部分で並列処理を行った．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のに適している．シミュレーションを高速化するために工夫した点を以下に示す 1) 2)．

- ステップ (4b) ならびに (4(e)i) において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る．GPU に搭載されている個々の SP は，CPU と比べ処理能力が低く，逐次処理が必要となる総和計算との相性がよくない．そのため，本シミュレータでは，総和計算は CPU で実行することとした．
- メインメモリと GPU 間のデータ転送は各プロセッサの処理時間の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 GPU に転送されたデータは，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．
- CUDA を使用したプログラムを作成する際に注意する必要がある項目に，1 ブロック当たりのスレッド数がある．本シミュレータでは，研究 1) における実験結果より，1 ブロック当たりのスレッド数を 256 に設定した．

4.4 GPGPU: GPU に対する処理の分割

GPGPU を用いた本シミュレータは，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 GPU の VRAM 上に保持している．これらの値の保存には非常に多くの記憶容量を使用する．また，増設が比較的容易なメインメモリとは異なり，GPU 上の VRAM を増設することは一般的に難しい．これらの理由により，研究 1) 及び 2) における実装では，シミュレーション可能な量子ビット数が限定されてしまっていた．そこで，処理を VRAM に収まるサイズに分割することにより，VRAM の容量に制限されないシミュレーションを可能にした．

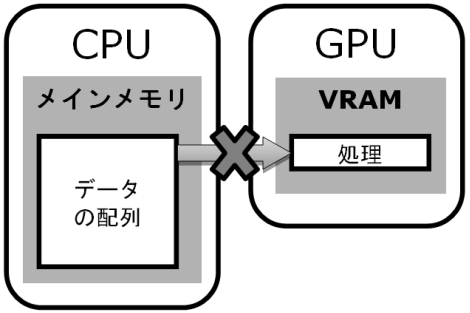

図 3 GPU への送信データの限界
Fig. 3 Limitation of send data to GPU.

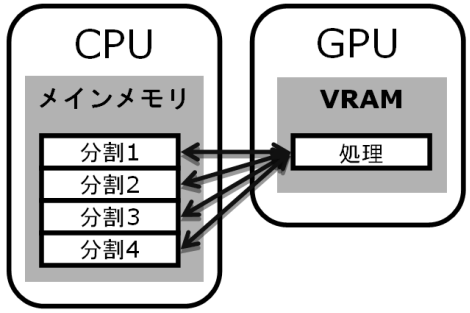

図 4 GPGPU における分割処理
Fig. 4 Divided execution of GPGPU.

GPU に搭載されている VRAM よりも大きい状態数のデータを処理する場合，これまではシミュレーションすることが出来なかった（図 3）．このとき，GPU に搭載されている VRAM の容量に合わせ，CPU で管理されているデータを分割して GPU へ送信する．分割されたデータ単位に，GPU にて処理し計算結果を CPU に出力する．CPU は GPU からの計算結果を受信後直ちに，次のデータを GPU に送信するというように，順次データを処理する（図 4）．このように，GPU にて一度に処理する量を制限し，シミュレーション全体として VRAM の容量に依存しない qubit 数の処理を実現した．本機能により，メインメモリの容量に保持できる要素数を使用して表現できる qubit 数の範囲内であれば，シミュレーションを実行することができる．

しかし，処理を分割せずに全ての要素の情報を VRAM 内に保持する場合と比較し，処理を分割した場合には CPU と GPU 間のデータ転送の回数が増大する．それにより，各プロセッサにおけるデータ転送の待ち時間のオーバーヘッドが増大し，シミュレーションの実行には多くの処理時間を必要とすることに注意する必要がある．

5. 実験環境及び結果

5.1 OpenMP における実験結果

作成したシミュレータを，2006 年に導入された大型計算機と 2009 年に導入された個人用パソコンで，それぞれ実行した．なお，実用上耐えうる実行時間を 3000sec とし，それを越えない範囲での実験を行った．実行環境の詳細を表 1 に示す．

500 ステップ実行するのに要した実行時間を，図 5 及び，表 3 の “OpenMP-US IV+” 及び “OpenMP-Core i7” に示す．大規模計算機を使用した場合よりも Core i7 を使用したとき

Vol.2009-MPS-76 No.2
Vol.2009-BIO-19 No.2
2009/12/17

表 1 OpenMP の実行環境：大規模計算機
Table 1 Execution environment of OpenMP.

項目	大規模計算機 (OpenMP-US IV+)	Core i7 使用 (OpenMP-Core i7)
CPU	UltraSparc IV+ 1.5GHz	Core i7-920 2.66GHz
ノード数 (最大並列度)	44 ノード × 2 コア (最大 88 並列)	4 コア × 2 スレッド (最大 8 並列)
メモリ	352GB	9GB
OS	Solaris 9	Windows Vista x64 sp2
開発環境	Sun Studio 12	Visual Studio 2005

の方がより処理時間を短縮できる結果となった．大規模計算機を使用した場合は 27qubit まで Core i7 を使用した場合は 26qubit まで実行可能であることを確認した．それ以降の qubit に対して，“OpenMP-US IV+” については実用上耐えうる時間を越えること，“OpenMP-Core i7” については開発環境における使用するデータ構造の限界を超えることにより，現在未計測となっている．

5.2 GPU に対する処理の分割による実行結果

GPGPU においても，OpenMP と同様の実験を行った．実行環境を表 2 に示す．このとき，1 ブロック当たりのスレッド数を 256 とし，27qubit までの実行については全ての要素の処理を GPU で 1 度に計算し，28qubit 以上につては分割し処理した．

qubit 数と実行時間の関係を図 5 と表 3 の “GPGPU-Div” に, 分割処理を行わなかった場合を “GPGPU-Single” に示す．分割処理を行わなかった場合は 27qubit までしか実行できなかったが，分割処理を行うことにより 29qubit までの実行に成功した．また，分割処理を行っていない 27qubit 以前と分割処理を行っている 28qubit 以降では，グラフの傾きが大きく異なっていることがわかる．

各シミュレーション環境の実行時間の測定結果を，図 5 と表 3 に示す．GPGPU を用いたシミュレータは OpenMP を用いたシミュレータと比べ，最大 1.47 倍の処理速度が改善した．また，OpenMP と GPGPU-Div において，シミュレーション可能な qubit 数は，メインメモリの量に依存する．そのため，使用できるメインメモリを増加することで，シミュレーション可能な qubit 数も共に大きくなると予想される．

6. 考　　察

6.1 OpenMP の実装方法について

5.1 節の結果は，OpenMP における処理の効率は基本的にコンパイラに依存するものの，それでも処理内容やデータの格納方法を見直すことで，より高いパフォーマンスを得ることが出来ることを示している．

また，比較的新しい CPU である Core i7 を用いた結果が最も処理時間が短い結果となった．これより，ある種の計算においては、多少古い大規模計算機よりも最新の CPU の方が高い処理能力を持つ可能性があると予想できる．

表 2 GPGPU の実行環境
Table 2 Execution environment of GPGPU.

項目	内容
CPU	Intel Core i7-920 2.66GHz
メモリ	9GB
GPU	Tesla C1060 ・SM 数: 30 ・SP 数 (全体): 240 ・VRAM: 4GB
OS	Windows Vista x64 sp2
開発環境	CUDA 2.3 SDK

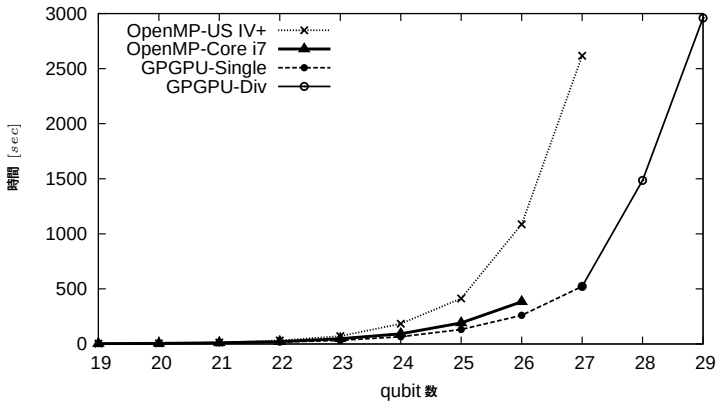

図 5 OpenMP と GPGPU の実行時間
Fig. 5 Execution time of OpenMP and GPGPU.

6.2 GPU に対する処理の分割による実行について

作成したシミュレータにより，28qubit 以上のシミュレーションが可能となった．さらに，今回作成したシミュレータは，問題を分割するサイズを設定する機能を備えている．この機能を使用すると，GPU に搭載された VRAM の容量に制限されずに，シミュレーションを行うことができる．たとえば，一般的に VRAM の容量が小さいノートパソコン等でも，シミュレーションを実行可能である．

Vol.2009-MPS-76 No.2
Vol.2009-BIO-19 No.2
2009/12/17

表 3 OpenMP と GPGPU の実行時間
Table 3 Execution time of OpenMP and GPGPU.

Qubits	OpenMP -US IV+ [sec]	OpenMP -Core i7 [sec]	GPGPU -Div [sec]	GPGPU -Single [sec]	比率
19	2.76	2.72	-	2.88	0.94
20	5.27	5.34	-	5.00	1.07
21	13.12	10.90	-	9.04	1.21
22	31.42	21.11	-	17.24	1.22
23	71.28	47.78	-	33.19	1.44
24	183.31	92.33	-	65.34	1.41
25	412.90	191.02	-	131.48	1.45
26	1086.78	383.44	-	260.30	1.47
27	2616.07	未計測	-	522.17	-
28	未計測	未計測	1485.17	実行不可	-
29	未計測	未計測	2960.01	実行不可	-

※比率は,「OpenMP-Core i7 の実行時間 / GPGPU-Single の実行時間」で表わされる.

その一方で，図 5 における “GPGPU-Single” と “GPGPU-Div” の傾きの差から分かるように，処理効率が悪くなっている．これは，分割したデータをメインメモリと VRAM との間での通信に時間がとられたためであると考えられる．

6.3 OpenMP と GPGPU の実行時間の比較

図 2 に示したとおり，GPGPU は理論的には高い浮動小数点数演算性能を持っている．しかしながら，処理能力が小さい演算機を多数並べる機構であるため，処理によっては必ずしも CPU より高速になるとは限らない．さらに，実際にパフォーマンスを発揮するためには，様々な工夫が必要となる．

本研究で作成した GPGPU によるシミュレータは，19〜26qubit の Grover のアルゴリズムのシミュレーションを OpenMP よりも約 1.5 倍高速に実行する．このことは，GPGPU が量子アルゴリズムの並列シミュレーションに適していることを表している．さらに，基本的に並列計算の方法はコンパイラ任せである OpenMP と異なり，GPGPU はハードウェアを意識したプログラミングが可能である．今後，各 SM や SP をより効率的に並列動作させるようにチューニングすることにより，より一層の高速化が期待できる．

その一方で，GPGPU で 27qubit 以上のシミュレーションを行う場合，分割処理をするための時間が余計に必要となり，全体の実行時間に少なからず影響を与えている．“OpenMP-Core i7” における 27qubit 以上の場合は未計測であるが，26qubit 以下のときと同様に 1qubit 増加する毎に実行時間が倍になると仮定すると，27qubit で 766sec，28qubit で 1532sec，29qubit で 3064sec となることが予想される．これは，“GPGPU-Div” とほぼ変わらない値であり，GPGPU が OpenMP に対して大きい優位性を持つためには，より一層のチューニングによる最適化・高速化が必要となる．

7. おわりに

本論では，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，OpenMP と GPGPU の二つの手法を用いて実現した．その際，より現実に近いシミュレーションにするため，ノイズの影響を考慮した．さらに，プログラムの改善により，OpenMP における高速化，GPGPU における高速化及びより大きな qubit 数へ対応を実現した．計算機実験の結果から，Grover のアルゴリズムのシミュレーションに対して，GPGPU を用いることで高速かつ大きな qubit 数におけるシミュレーションの実行が可能であることを示した．以上から，Grover のアルゴリズムのシミュレーションに対する GPGPU の有効性を示した．

今後の課題として，更なる処理の高速化が挙げられる．その為には，アルゴリズムの実行手順，複数の GPU の利用，各要素のデータの保存方法等をさらに検討する必要がある．また，他の量子アルゴリズムや，任意の量子アルゴリズムを実行するシミュレータの作成を試みることも重要である．以上に対応することで，量子コンピュータの実現分野の研究に対する寄与することが期待される．

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しました．謹んで感謝の意を表します．

参 考 文 献

1) 芝田浩，鈴木智也，大久保誠也，西野哲朗: Grover のアルゴリズムのシミュレーションにおける GPGPU の利用について，第 21 回量子情報技術研究会資料 QIT2009-58, pp.72-77(2008)
2) 芝田浩，西野哲朗，大久保誠也，鈴木智也: GPGPU による 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-120 (5), pp.33-40(2008)
3) 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)
4) 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)
5) Grover, L.: 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) Pedro J. Salas.: Noise effect on Grover algorithm, *arXiv:0801.1261v1 [quant-ph]*
8) 西野 哲朗: 量子コンピュータの理論, 培風館 (2002)
9) NVIDIA Corporation: *NVIDIA CUDA Compute Unified Device Architexture Programming Guide*, Version 2.3 (2009)