量子シミュレータのホットスポット分析と考察

青木　望美†　山崎　雅文††　平井　聡††　山岡　茉莉††　福本　尚人††
小口　正人†

† お茶の水女子大学　〒 112–8610 東京都文京区大塚２丁目１－１
†† 富士通株式会社　〒 211–8588 神奈川県川崎市中原区上小田中４丁目１－１
E-mail: †nozomi-a@ogl.is.ocha.ac.jp, ††{m.yamazaki,ahirai,yamaoka.mari,fukumoto.naoto}@fujitsu.com,
†††oguchi@is.ocha.ac.jp

あらまし　近年注目されている量子コンピュータは，従来のコンピュータと比較して非常に高速な並列計算を行うことができる可能性のあるコンピュータである．現在も様々な研究が行われているが，完全な形での実現は数年から数十年かかるとされている．そこで，従来型のコンピュータ上で量子コンピュータの挙動を表現・シミュレーションする量子シミュレータは量子アルゴリズムやアプリケーションの開発に有用であり，量子コンピュータの研究と並行してこれらの研究をすることは重要である．今回は 2 種類の State Vector 方式シミュレータそれぞれについてホットスポット分析を行い，分析結果の解析と考察を行う．

キーワード　性能分析，アプリ分析，プロファイラ，量子回路シミュレータ

1　はじめに

近年注目が集まっている量子コンピュータは，従来型コンピュータと比べて非常に高速な計算を行うことができる可能性があるとして盛んに研究，開発がなされている．従来型コンピュータでは最小単位であるビットを 0 または 1 で表すが，量子コンピュータでは量子の持つ性質を利用して 0 と 1 の重ね合わせ状態も保持可能な量子ビットを扱うことが大きな特徴であり，この量子ビットを用い，量子ゲートなどを適切に組み合わせることで計算の解が出力される確率を高めて答えを導き出す．また，このとき複数通りの計算を同時に行うため高速な計算を実現することができる（図 1 参照）．量子コンピュータにより高速化が期待されている計算は複数存在し，その一例として例えば量子化学計算などが挙げられる．しかし，現時点でもすでに複数の量子コンピュータが存在するものの，いずれもノイズの影響で計算結果にエラーが含まれるなど完全な挙動をするものとは言い難い．実際に理想的な挙動をする量子コンピュータが実現するまで，数年から数十年ほどが必要とされている．

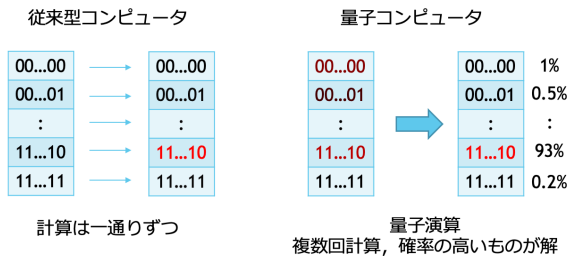

図 1　量子コンピュータの挙動

ここで，従来型コンピュータ上で量子コンピュータの挙動を表現してシミュレートする量子シミュレータが存在する．量子シミュレータを用いると計算結果は常に正確なものを得ることが可能である．量子コンピュータの実現まで数十年かかる可能性がある中で，量子コンピュータの開発と並行して量子シミュレータを使用して量子アプリ開発などを行うことは有用である．現在も IBM が開発した Qiskit Aer [1] や国立大学法人大阪大学と株式会社 Qunasys が共同開発した Qulacs [2]，Google が開発した Cirq [3] などの量子シミュレータが存在し，各シミュレータのユーザも多く有用であるが，それぞれの性能特性の違いは明らかになっていない．

そこで本研究では，複数の量子シミュレータの性能分析を行い，性能特性の違いを明らかにする．今回は実験用サーバ上で量子シミュレータを動作させると同時にシステム稼働状況を収集し，収集したデータを用いてホットスポット分析を行い，ボトルネックの要因を特定する．また，これにより分析対象の量子シミュレータの動作改善方法について検討する．

2　関連研究

量子シミュレータには State Vector 方式や Tensor Network 方式など複数の方式があり，各方式の様々な量子シミュレータが存在する．例えば State Vector 方式の量子シミュレータに，Intel-QS がある [4]．Intel-QS は CPU ベースの分散型 State Vector 方式シミュレータであり，マルチコア CPU での使用のために OpenMP でマルチスレッド化されている．また，ひとつの大規模な回路だけでなく，複数の小/中規模回路のシミュレートも可能である．Tensor Network 方式の量子シミュレータには例えば TeNeS [5] が挙げられる．TeNeS は多体量子状態計算を行うためのオープンソースプログラムパッケージであり，OpenMP と MPI ハイブリッド並列に対応している．またこの

他，Density Matrix 方式などのシミュレーション方式が存在する．State Vector 方式は，すべての量子状態をメモリに保持するため状態ベクトルをまとめて取得可能であり，計算量が量子回路の形状の影響を受けにくい．また，このような構造から動作途中の状態ベクトルを得ることも可能であるため，例えば量子アプリのデバッグなどに有用である．Tensor Network 方式では量子回路をテンソルネットワークとして表す．また，シミュレートにあたって必要な量子状態のみを計算する．Density Matrix 方式では，量子状態を行列として表現しつつシミュレートを行う．このとき，必要なメモリ量は非常に多くなる．

続いて量子シミュレータの性能向上に関する関連研究を示す．mpiQulacs [6] は，State Vector 方式の量子シミュレータのひとつである Qulacs [2] を，A64FX CPU に対して最適化した量子シミュレータであり，MPI により並列分散処理に対応している．演算を行う際は，CPU に搭載されている SVE 命令と呼ばれる，複数の演算を一度にまとめて実行することができる命令を使用することでメモリバンド幅を引き上げている．また，状態ベクトルを複数に分割し，各プロセスで均等に保持して各々計算を行い，他プロセスと状態ベクトルを共有する時のみ MPI 通信を使用することで並列処理を実現している．この他にもダブルバッファリングを実装するなど高速化のための工夫をすることで，数十ノード単位の大規模シミュレーションに関して，他の量子シミュレータと比較して性能が高くなったことを確認している．

3 量子シミュレータ

前章でも述べたように，量子シミュレータには State Vector 方式と Tensor Network 方式など複数の方式が存在する．量子アプリ等の開発における有用性の観点から，今回の実験では State Vector 方式のシミュレータについて分析を行う．以下で分析に用いる量子シミュレータについて紹介する．

3.1 Qiskit Aer

Qiskit Aer [1] は量子コンピュータ用のフレームワークである Qiskit [7] に含まれる量子シミュレータである．Qiskit では量子回路を構築して実際の量子デバイス上で実行することが可能であり，モジュール構造のため最適化などの処理も行いやすく設計されている．Python3.7 以降をサポートしており，従来型コンピュータ上でも量子回路を表現してシミュレートすることが可能となっており，量子シミュレータの方式は State Vector 方式と Tensor Network 方式の両方に対応している．

3.2 Qulacs

Qulacs [2] は国立大学法人大阪大学と株式会社 Qunasys によって開発された量子シミュレータである．Qulacs は C/C++ で実装されており，使用時は Python インタフェースを介して使用することができるため高速なシミュレーションと高い利便性を実現している．

使用には C++コンパイラ，Python2.7 または 3 以降などが必要である．

4 性能分析ツール

本稿で利用した性能分析ツールについて説明する．

4.1 mpstat

mpstat は，CPU 使用率やハードウェア割り込みなどの情報を取得することが可能なコマンドである．通常各カテゴリについて，すべての CPU の平均値と各 CPU ごとの統計情報を得ることができる．

mpstat を用いることで各コアの挙動を詳細に確認することができるため，I/O 競合発生の可能性を発見することができるなどボトルネックの要因特定に有用である．

4.2 vmstat

vmstat は，プロセスやメモリ，CPU などに関する情報をリアルタイムで収集することができるコマンドである．取得することができるデータの例として，例えばプロセス数や空きメモリの大きさ，CPU のユーザ時間などがある．

vmstat で収集することができるデータの範囲は広く，システム稼働状況を大まかに把握する際に有用である．

4.3 Linux Perf

Linux Perf [8] は，主に Linux Kernel のイベントデータ収集，トレース機能を提供しているツールであり，Linux Kernel 2.6.31 以降で利用可能である．Linux Perf にはいくつかのサブコマンドが存在し，Linux Perf を使用する際は通常いずれかのサブコマンドと併用する．

今回の実験では stat と record，report というサブコマンドを使用する．perf stat は，あるプログラムの実行中に指定したイベント値を出力することなどが可能である．取得することができるデータに，ページフォールトの回数やキャッシュヒット/ミス数などがある．perf record と perf report は，プログラムの実行中に実行時間が長い関数を，ランキング形式で出力することが可能なサブコマンドである．

4.4 PCM Tools

PCM Tools [9] は CPU のリソース利用に関する情報をリアルタイムで取得することができるツールである．PCM にはいくつかのコマンドが含まれており，例えば pcm-memory を用いるとメモリバンド幅のモニタリングが可能である．実際に計測する際は必要に応じてこのようなコマンドを併用する．

5 実　　験

5.1 実験概要

本稿では，Qiskit Aer と Qulacs それぞれのライブラリを使用した量子シミュレータの Python プログラムを実行し，実行中のシステム稼働状況を複数の性能分析ツールを用いて収集する．

使用する量子シミュレータのバージョンを表 1 に，各性能分析ツールのバージョンを表 2 に示す．なお，mpstat は sysstat

に含まれるツールであるため，sysstat のバージョンを表記している．

表 1　量子シミュレータのバージョン

qiskit	0.39.3
qiskit aer	0.11.1
qulacs	0.5.3

表 2　性能分析ツールのバージョン

sysstat	11.7.3
vmstat	procps-ng 3.3.15
perf	4.18.0-425.3.1.el8.x86_64
pcm	202107-5.el8

量子体積 (QV: Quantum Volume) は，量子コンピュータのシステムを評価する目的で，どの程度複雑な処理が可能かを示す指標として提案 [10] された．量子体積モデル回路はこの量子体積を評価するためのものであり，ランダムな組み合わせの量子ビットのペアに 2 量子ゲートを作用させる．今回の実験には，Qiskit Aer，Qulacs を使用した各 Python プログラムに，この量子体積モデル回路を用いた．測定時の量子ビット数は 31，回路の深さは 10 に設定した．なお，perf record での計測時は生成されるファイルサイズが大きくなりすぎることを防ぐために，量子ビット数を 27 個に変更する．

なお Python プログラムの実行開始は，各性能分析ツールの実行開始から約 10 秒後とする．ただし，perf record については分析対象のプログラムと関連がない関数が上位に入ることを防ぐために，分析対象のコマンドを指定してデータの収集を行う．性能分析ツールの設定について，mpstat と vmstat は fflush() を用いることでデータ収集の間もバッファのフラッシュを行う．また，perf stat は収集するイベントデータを以下に示す複数のグループに分けてデータ収集を行う．

- コアサイクル数，実行命令数，L1 キャッシュミス数，L1 キャッシュ参照数，
- コアサイクル数，実行命令数，L2 キャッシュミス数，L2 キャッシュ参照数
- コアサイクル数，実行命令数，L3 キャッシュミス数，L3 キャッシュ参照数
- コアサイクル数，実行命令数，マイクロオペレーション実行数，ロード命令数，ストア命令数

pcm は，pcm-memory コマンドを用いてメモリバンド幅のモニタリングを行う．また，これら性能分析ツールは，すべて 1 秒間隔でデータの収集を行う．ただし perf record のみ 1 ミリ秒間隔でデータの収集を行う．

5.2　実験環境

実験環境を表 3 に示す．

表 3　実験環境

実験用サーバ	Primergy RX2540 M1
OS	Rocky Linux 8.6
CPU	Intel Xeon プロセッサー E5-2697v3(2.60GHz)
メモリ	128GB

5.3　実験結果

5.3.1　Qiskit

はじめに Qiskit Aer を使用したシミュレータについての結果を記述する．

性能分析ツールを用いた分析は複数回に分けて行う．Python プログラムの実行時間は約 261 秒であった．

mpstat について，実験サーバに搭載されている計 14 個の CPU すべてにおいて各 CPU の%usr（ユーザ時間の割合）の値が 100 %近くなった．よって Qiskit ライブラリを用いたシミュレータについては，CPU 単位での並列処理が行われていることがわかる．図 2 にすべての CPU のユーザ時間割合の変化の平均に関するグラフを示す．

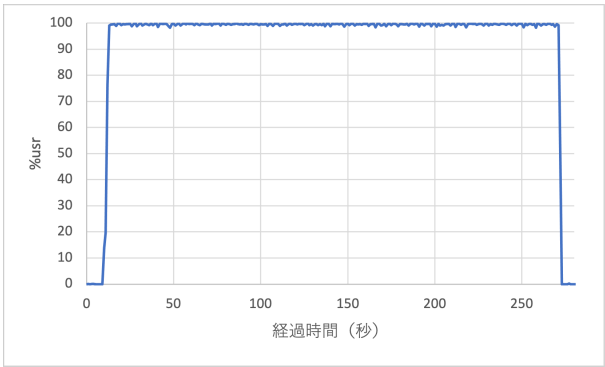

図 2　全 CPU の%usr 平均値の変化（Qiskit）

また，vmstat の計測結果では空きメモリ量を表す free 項目から，プログラム実行時にメモリが約 33.7 GB 確保され続けていることがわかった．図 3 に空きメモリ量に関するグラフを示す．なお，他の性能分析ツールに合わせて計測開始 0 秒時点のデータは無視している．

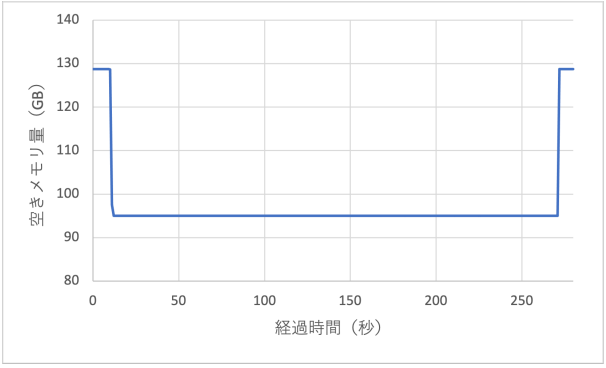

図 3　空きメモリ量の変化（Qiskit）

続いて perf stat での計測結果について記す．

各 CPU コアの IPC の値は 3 前後と比較的高い値を保っていた．実験に使用したサーバにおける IPC の理論値は 4 程度である．図 4 に，全コアの IPC の平均値のグラフを示す．

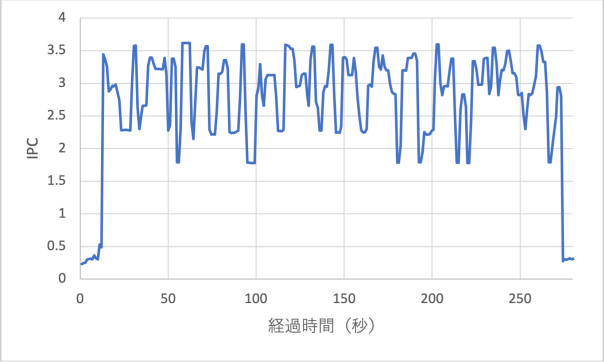

図 4　IPC 値の変化（Qiskit）

またプログラム実行中，すべての命令の中でロード命令が占める割合は約 16 %であった．

各 CPU コアにおける L1，L2，L3 キャッシュそれぞれのキャッシュミス率の全コア平均値はそれぞれ約 1.7 %，約 45.2 %，約 65.4 %であった．図 5 に L1 キャッシュミス率の変化，図 6 に L2 キャッシュミス率の変化，図 7 に L3 キャッシュミス率の変化を示す．

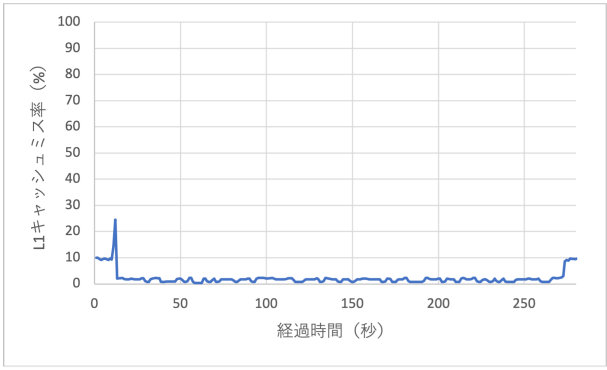

図 5　L1 キャッシュミス率の変化（Qiskit）

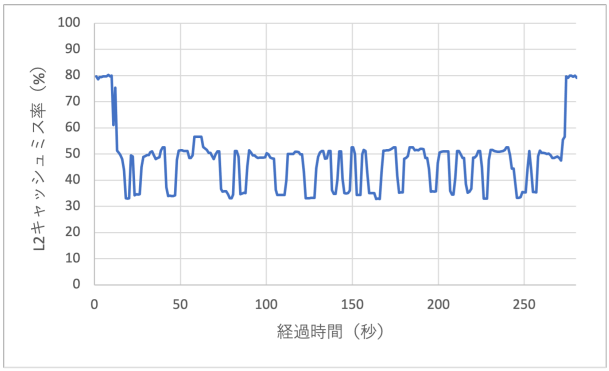

図 6　L2 キャッシュミス率の変化（Qiskit）

次に perf record / perf report での計測結果について記す．データの採取は 1 秒あたり 1000 回に設定して実行したところ，Object 名と Symbol 名が以下の表 4 に示すものについて，実行時間全体の約 97 %もの時間を消費していることがわかった．apply_matrix_n 内には二重ループを含むため，実行時間の消費が激しい理由はこの部分にあると考えられる．また，ホットスポットはこの部分であると考えられる．

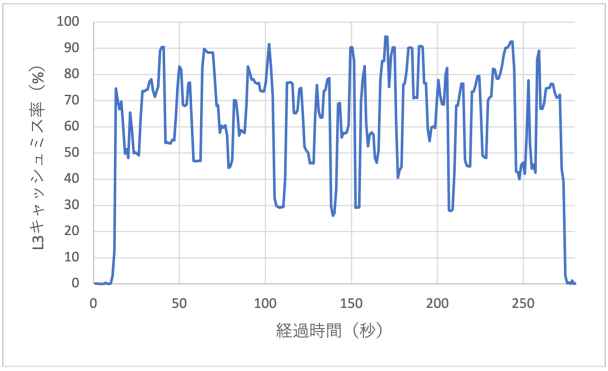

図 7　L3 キャッシュミス率の変化（Qiskit）

表 4　perf record / perf report の出力結果（Qiskit）

Shared Object	controller_wrappers.cpython-39-x86_64-linux-gnu.so
Symbol	apply_matrix_n

続いて pcm-memory での測定結果について記す．プログラム実行中のメモリバンド幅は約 20 GB/s から 40 GB/s の間で変化した．図 8 にグラフを示す．

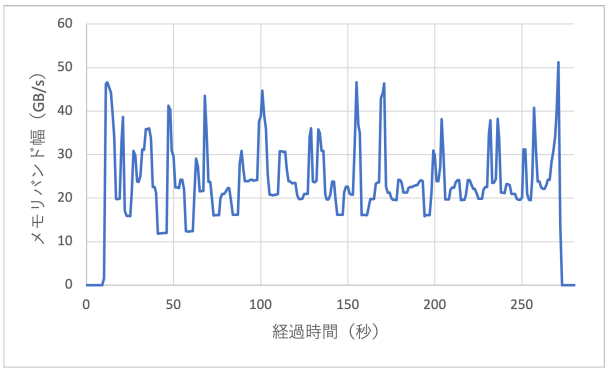

図 8　メモリバンド幅の変化（Qiskit）

実験に使用したサーバのメモリバンド幅の理論値は 68 GB/s である．また，実効的なメモリバンド幅を引き出すことが可能として知られる STREAM ベンチマーク [11] を使用して同様に pcm-memory でメモリバンド幅の計測を行ったところ，ベンチマーク実行中の最大メモリバンド幅は約 57 GB/s となった．これが実効メモリバンド幅と考えると，Qiskit Aer シミュレータのメモリバンド幅はさらに引き出すことができる可能性があり，またそれにより性能向上を期待できると考えられる．

以上の結果から，Qiskit Aer のライブラリを用いたプログラムにおいて計算実行は効率よく行われていることがわかった．また，状態ベクトルの更新に最も時間を費やしていることがわ

かった．さらに，計測したメモリバンド幅は実効メモリバンド幅よりも値が低く，この点を改善することでシミュレータ全体の性能向上が期待できることがわかった．

5.3.2 Qulacs

続いて Qulacs を使用したシミュレータについての実験結果を記す．

こちらも性能分析ツールを用いた分析は複数回に分けて行う．Qulacs を使用したプログラムの実行時間は約 241 秒であった．

mpstat での分析結果については Qiskit Aer を使用したシミュレータと同様に各 CPU の%usr の値はいずれも 100%近い値を保っていた．よって Qulacs ライブラリを使用したシミュレータに関しても，CPU 単位での並列処理はされていると考えられる．図 9 にすべての CPU のユーザ時間割合の変化の平均に関するグラフを示す．

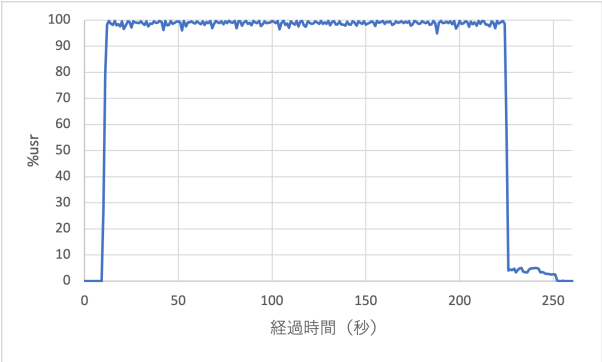

図 9　全 CPU の%usr 平均値の変化（Qulacs）

また vmstat の出力結果から，プログラム実行途中までは約 33.7 GB 程度のメモリが確保されており，最後にさらに約 32.4 GB のメモリが確保されたことがわかった．図 10 に空きメモリ量の変化に関するグラフを示す．

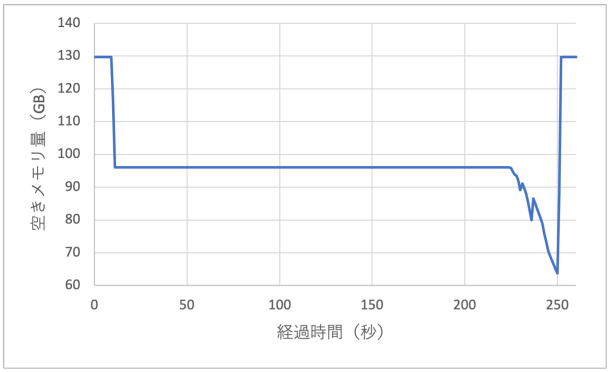

図 10　空きメモリ量の変化（Qulacs）

続いて perf stat での計測結果について記す．

各 CPU コアの IPC の値は，Qiskit Aer シミュレータと同様に 3 前後と比較的高い値となった．ただし，Qiskit Aer シミュレータよりも値のばらつきは少ないことがわかる．図 11 に，全コアの IPC 値の平均のグラフを示す．

また，全命令数の中でロード命令が占める割合の全コア平均値は約 17.7 %程度と，Qiskit Aer シミュレータと比較すると

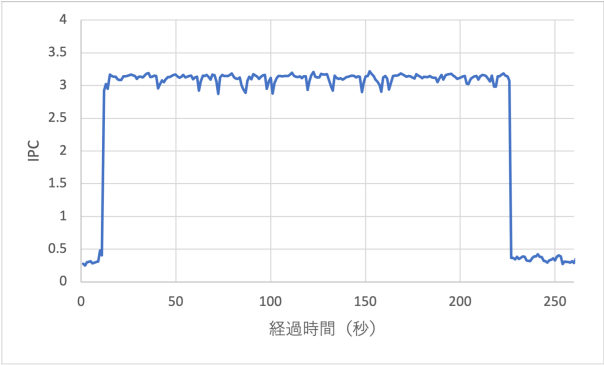

図 11　IPC 値の変化（Qulacs）

少し高い結果となった．

次に L1 キャッシュ，L2 キャッシュ，L3 キャッシュにおけるキャッシュミス率の全コア平均値についてはそれぞれ約 3.2 %，約 55.1 %，約 36.2 %となった．Qiskit Aer シミュレータと比較して L1 キャッシュミス率と L2 キャッシュミス率はやや高く，L3 キャッシュミス率は低いが振れ幅が大きい結果となった．図 12 に L1 キャッシュミス率，図 13 に L2 キャッシュミス率，図 14 に L3 キャッシュミス率の変化をそれぞれ示す．

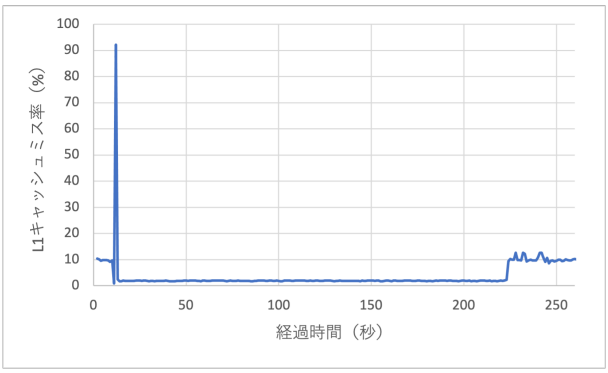

図 12　L1 キャッシュミス率の変化（Qulacs）

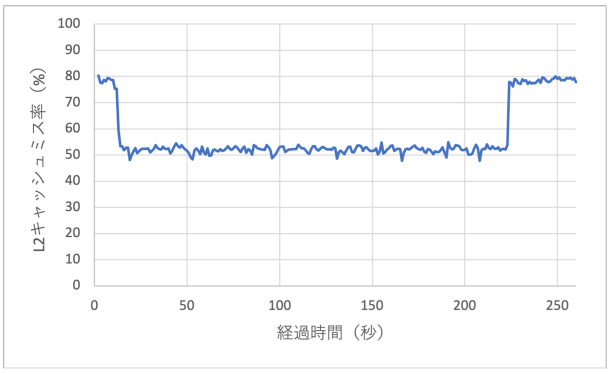

図 13　L2 キャッシュミス率の変化（Qulacs）

次に perf reecord / perf report の出力結果を記す．

こちらもデータの採取を 1 秒あたり 1000 回に設定して実行したところ，Object 名と Symbol 名が以下の表 5 に示すものについて，実行時間全体の約 96 %もの時間を消費していること

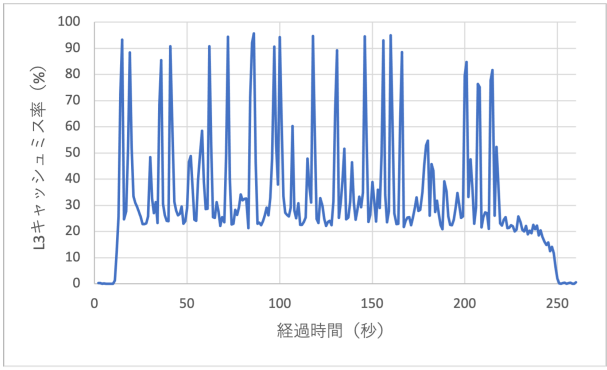

図 14　L3 キャッシュミス率の変化（Qulacs）

がわかった．ただし，double_qubit_dense_matrix_gate_nosimd 自体の実行時間割合は約 12 %であり，それ以外は呼び出し先関数が占めている．呼び出し先関数は C++の complex クラスに含まれる複素数の演算を行う関数が大半であった．この呼び出しは状態ベクトルの更新のために行われていると考えられ，また Qulacs シミュレータにおけるホットスポットはこの部分と考えられる．

表 5　perf record / perf report の出力結果（Qulacs）

Shared Object	qulacs_core.cpython-39-x86_64-linux-gnu.so
Symbol	double_qubit_dense_matrix_gate_nosimd

また PCM Tools での計測結果より，プログラム実行中のメモリバンド幅は約 45 GB/s から 50 GB/s 程度で変化しており，Qiskit Aer シミュレータと比較すると大きい数値となった．しかし，先ほど記述した実効メモリバンド幅（約 57GB/s）と比較すると，比較的低いと考えられる．図 15 にメモリバンド幅の変化に関するグラフを示す．

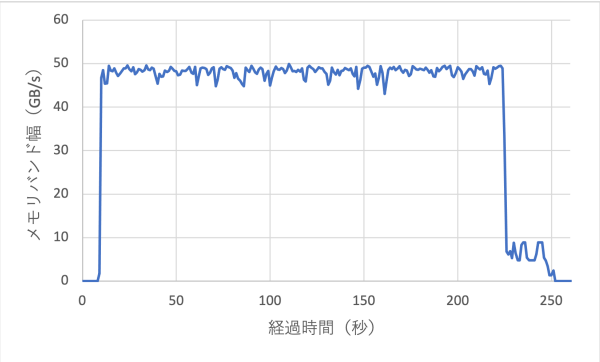

図 15　メモリバンド幅の変化（Qulacs）

よって Qiskit Aer シミュレータと比較するとメモリバンド幅を引き出す工夫がされていると考えられるが，実効メモリバンド幅と比べると値が小さいため，工夫次第でメモリバンド幅を引き出すことができる可能性があると考えられる．

以上の結果から，Qulacs を使用したプログラムは Qiskit Aer シミュレータと同様に IPC の値が高く，計算処理は効率的に行われていると考えられる．またホットスポットとなっている箇所は複素数演算を行う関数部分であることがわかった．さらに，メモリバンド幅の値は比較的高いことがわかった．

6　まとめと今後の課題

今回は Qiskit Aer と Qulacs それぞれのライブラリを使用した Python プログラムについて，プログラム実行中のシステム稼働状況のデータを収集することで性能分析を行った．その結果，Qiskit Aer ライブラリを使用したプログラムは計算の実行効率が良いことがわかった．また，状態ベクトルの更新を行う関数が最も時間消費が激しいことがわかった．さらに，メモリバンド幅は比較的低い値となっているため，この部分を改善することが性能向上に繋がると考えられる．この点については，Qiskit Aer を使用した State Vector 方式のシミュレータは連続メモリにアクセスすることを繰り返す挙動をすることから，実際にメモリバンド幅を改善することは可能と考えられる．また Qulacs ライブラリを使用したプログラムは Qiskit Aer シミュレータと同様に計算の処理効率が良いことがわかった．また実行時間を多く消費しているのは複素数演算を行う関数であり，これは状態ベクトル更新のために実行されていると考えられる．さらにメモリバンド幅は Qiskit Aer ライブラリを使用したプログラムと比べて大きいため効率的にデータ転送を行なっていると考えられる．しかし Qulacs シミュレータも Qiskit Aer シミュレータと同様に，工夫次第でメモリバンド幅をさらに引き上げることは可能ではないかと考えられる．

今後は今回分析を行った量子シミュレータのさらなる分析に加えて，他の量子シミュレータについても同様の分析を行っていく．また分析した量子シミュレータについて，その性能特性から性能向上のための具体的な手法を検討し，実装して検証を行う．特にメモリバンド幅の改善策については，メモリアクセス順序等の分析やプログラム中のホットスポット箇所の命令実行の見直しを行い，改善可能な箇所の特定を行う．

謝　　辞

本研究の一部はお茶の水女子大学と富士通株式会社との共同研究契約に基づくものである．

文　　献

[1] Qiskit aer simulator. https://qiskit.org/documentation/tutorials/simulators/1_aer_provider.html.
[2] Qulacs ドキュメンテーション — qulacs ドキュメント. http://docs.qulacs.org/ja/latest/index.html.
[3] Cirq. https://quantumai.google/cirq.
[4] Intel quantum simulator — intel qs documentation. https://intel-qs.readthedocs.io/en/docs/getting-started.html#.
[5] Tenes - pasums - 東京大学. https://www.pasums.issp.u-tokyo.ac.jp/tenes.
[6] Imamura Satoshi, Yamazaki Masafumi, Honda Takumi, Kasagi Akihiko, Tabuchi Akihiro, Nakao Hiroshi, Fukumoto Naoto, and Nakashima Kohta. mpiqulacs: A distributed quantum computer simulator for a64fx-based cluster sys-

tems. *Distributed, Parallel, and Cluster Computing*, 2022.

[7] Qiskit. https://qiskit.org.

[8] Perf wiki. https://perf.wiki.kernel.org/index.php/Main_Page.

[9] Intel® performance counter monitor - a better way to measure cpu utilization. https://www.intel.com/content/www/us/en/developer/articles/technical/performance-counter-monitor.html.

[10] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta. Validating quantum computers using randomized model circuits. *Phys. Rev. A*, Vol. 100:p.32328(online), 2019. DOI: 10.1103/PhysRevA.100.032328.

[11] Stream: Sustainable memory bandwidth in high performance computers. https://www.cs.virginia.edu/stream/.