疑似量子アニーリングにおける自動チューニングの適用

片桐 孝洋[1]，森下 誠[2]，河合 直聡[1]，星野 哲也[1]，永井 亨[1]
[1]名古屋大学情報基盤センター，[2]名古屋大学大学院情報学研究科
e-mail: katagiri@cc.nagoya-u.ac.jp

1　概要

近年 Google や中国科学技術大学による量子超越性の主張[1]があり，量子コンピュータの実用性に注目が集まっている．しかしこの一方で，量子超越性が主張された問題を解くためのアルゴリズムを古典コンピュータ向きに改良し，かつ並列化と高性能実装を行ったうえで最新の GPU スパコンで実行すると，量子コンピュータで解くより高速かつ低電力で求解可能である主張もされている[2]．このことから現在，本当に量子超越性が達成されているかは明らかではないと著者は考えている．しかし現在，量子コンピュータに注目が集まっていることに疑いはない．

一方で，量子アルゴリズムから着想を受けたアルゴリズム，もしくはハードウェアの開発が盛んに行われてきた．これらは量子特性活用していないため，量子コンピュータではない．そのことから量子「関連」技術と呼ばれており，注目されている．特に我が国においては，「疑似」量子アニーラと呼ばれる組合せ最適化問題の求解に特化したハードウェア開発が盛んであり，商用化もされている．例えば，日立の CMOS アニーリングマシン[3]や富士通のデジタルアニーラなどがある．これらは現在，クラウドサービスとして利用可能である．

本研究は，CMOS アニーリングマシンの性能評価を目的とする．特に，解の性能向上のための性能パラメタチューニングの問題を取り扱う．

2　疑似量子アニーラへの AT の適用

疑似量子アニーラである CMOS アニーリングマシン[3]は，イジングモデルの基底状態探索を行うものであり，専用ハードウェア(ASIC)もしくは GPU に実装されている．クラウドサービスは，「Annealing Cloud Web」[4]（GPU 版，32bit，Float）で提供されている．これらの量子アニーラでは，QUBO（Quadratic Unconstrained Binary Optimization，二次制約なし二値最適化)形式の二値変数の最適化問題に帰着させ，コスト関数を定義することで，組合せ最適化問題の求解が可能となる．

疑似量子アニーラでは,QUBO 上とアニーリングアルゴリズム上の性能パラメタが存在する．例えば CMOS アニーリングマシンでは，*Wa*：QUBO 制約項の重み，*Wb*：QUBO コスト項の重み，*chain_strength*：エッジの重み，*temperature_num_steps*：アニーリングのステップ数，などの性能パラメタがある．これらのパラメタは解の品質に影響するため，実行ごとにチューニングする必要がある．そのため，チューニングの手間がかかるだけではなく，高品質な解を得るための必須処理であり，性能チューニングの自動化の要請がある．

本研究では，典型的な組合せ最適化問題である「最小頂点被覆問題」を扱う．最小頂点被覆問題の QUBO は，式(1)のようになる．

$$H = W_a \Sigma_{(u,v)\in E}(1-x_u)(1-x_v) + W_b \Sigma_{v\in V'} x_v \qquad (1)$$

ここで，$x_u, x_v \in \{0,1\}$である．式(1)は，エネルギーを定義しているとみなすことができる．エネルギーは小さいほど最適である．ここで，式(1)の前半の項は「制約項」と呼ばれる．式

(1)の後半の項は「コスト項」と呼ばれる．式(1)には，制約項の重みWaと，コスト項の重みWbがある．これらは重要な性能パラメタとなる．この性能パラメタのチューニングに，数値計算ライブラリ分野で長く研究をされてきた「AT技術」[5]の適用を試みるのが本研究である．

3　予備評価手法

Annealing Cloud Web API V2 (GPU version 32bit(float))[6]を利用した．クライアントPCは，MacBookAir(macOS Big Sur)を利用した．CPUは1.6GHz Dual Core Intel Core i5(8GBメモリ)である．Python(Version 3.8.2)を利用した．Cloud WEB 上の実行には，FIXSTARS Amplify [7]を利用した．これらは，CMOSアニーリングマシンを使うためのWeb APIであり，バージョンは0.5.13である．

本予備評価においては，最小頂点被覆問題の解の品質を評価できる人工問題を作成して解の品質評価を行った．具体的には，2次元格子状のグラフ問題をベンチマークとして与えることで最適解を事前に知ることができる．そこで，ATを行った場合の最適解求解率[%]を評価した．利用したATフレームワークはPreferred Networks社のOptuna[7]を利用した．

OptunaによるATでは，Wa，Wb，$chain_strength$の3パラメタのチューニングを対象にした．最適解回答率（100 回施行した場合の最適解を回答する割合）を目的関数に設定した．Optuna はベイズ推定によるブラックボックス最適化を行うが，探索アルゴリズムとしては，CMA-ES（共分散行列適応進化戦略），およびTPE (Tree-Structured Parzen Estimator)を中心にATの経緯を評価した．詳しい性能評価結果については，当日発表する．

謝辞　本研究は，学際大規模情報基盤共同利用・共同研究拠点、および、革新的ハイパフォーマンス・コンピューティング・インフラ（課題番号: jh240002）の支援による．CMOSアニーリングマシンの利用に関して助言を頂いた，日立製作所の小埜和夫氏と山岡雅直氏に感謝します．また，Amplify の利用に関して助言を頂いた，株式会社フィックスターズの松田佳希氏に感謝します．

参考文献

[1] E. Gibney, Hello Quantum World! Google Publishes Landmark Quantum Supremacy Claim, Nature, Vol.574, (2019), 461-462.

[2] R. Fu, et al., Achieving Energetic Superiority Through System-Level Quantum Circuit Simulation, arXiv:2407.00769, (2024).

[3] M. Yamaoka, et al., 20k-spin Ising chip for combinational optimization problem with CMOS annealing, Proceedings of 2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers, (2015).

[4] Annealing Cloud Web, https://annealing-cloud.com/ja/index.html.

[5] T. Katagiri, D. Takahashi, Japanese Autotuning Research: Autotuning Languages and FFT, Proceedings of the IEEE, Vol.106, Issue 11, (2018), 2056-2067.

[6] FIXSTARS Amplify, 量子コンピューティングプラットフォーム, FIXSTARS, https://amplify.fixstars.com/ja/.

[7] Preferred Networks, Inc.: An open source hyperparameter optimization framework to automate hyperparameter search. https://optuna.org/.

スーパーコンピュータの運用状況に応じた並列自動チューニング機構の提案

矢島 雄河, 藤井 昭宏, 田中 輝雄
工学院大学
e-mail： em23045@ns.kogakuin.ac.jp

1　はじめに

ソフトウェア自動チューニング (AT : software auto-tuning) とは，ソフトウェアの性能に影響を与えるパラメタ (性能パラメタ) を操作し，性能を自動的にチューニングする手法である [1]．AT では，プログラムを繰り返し実行しながらより良い性能パラメタを探索する．そのため，機械学習などの実行時間が長いプログラムを対象とした場合，AT には非常に長い時間を要する．これに対し，我々はプログラムの実行を並列に行うことで時間短縮を図ってきた．しかし後述するように，単純な並列化では必ずしもシステムの計算能力を十分に引き出すことができるとは限らない．本研究ではスーパーコンピュータシステムの運用状況を考慮し，より並列性を活用した AT を提案する．

2　並列なソフトウェア自動チューニング

我々はスーパーコンピュータでの AT として，ジョブを並列単位とする方式を提案してきた [2]．ここでジョブとは，スーパーコンピュータにおけるプログラム実行単位である．ジョブを並列単位とすると，対象のユーザプログラムに大きな変更を加えずに AT を行うことができる．

図 1 に流れを示す．探索は実行対象列挙，実行，集計の繰り返しである．実行対象列挙の際，性能パラメタの全組合せを考えると，対象の数はパラメタ数に応じて指数的に増加する．そのため，図 1 の右側に示すように対象を絞り込む．図中の直線はある基準点からどのパラメタを変化させるかを表しており，1 パラメタのみ変化させる場合は軸に沿った直線，複数パラメタを同じ値ずつ変化させる場合は斜めの直線が対象となる．同時に変化するパラメタ数を段階的に増やすことにより，最初は各パラメタによる影響を考え，その後パラメタ同士の相関を考慮した調整を行う仕組みである．

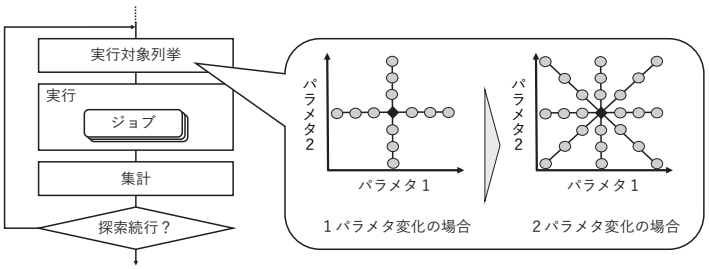


図 1. 並列化した自動チューニング機構

3　スーパーコンピュータの運用状況の考慮

前述の方式では，実行対象列挙時にシステムのリソースを考慮しない．そのためシステムの最大並列数に対して対象が過剰，もしくは不足となり得る．またスーパーコンピュータのように複数ユーザが同時に利用する環境では，その時々で利用可能なリソースが変動する．したがって事前にシステムの最大並列数を取得しても，その瞬間実際に投入可能なジョブ数との間に差が生じる可能性がある．

そこで，並列 AT の性質に着目する．まずプログラムそのものを並列単位とするため，各実行対象

の独立性は自動的に保証されている．また集計はその時点までに得た任意の対象の結果のみで行う．結果が実行対象の一部の場合，推定には影響を与えても，探索を中断する要因にはならない．

以上のことを踏まえ，我々は集計のタイミングをアルゴリズムではなくシステムベースとする方式を提案する．この方式では実行対象を毎回多めに列挙し，その後システムの最大並列数での実行を試みる．具体的にはまず，基準点から伸びる直線すべてを実行対象とする．このとき，同時に変化するパラメタ数が少ない対象ほど優先度を高く設定する．続いて，システムで利用可能な最大並列数を認識し，実行対象一覧のうち一部のみを実行する．

4　実験

表 1. AT の結果

	AT 前	アルゴリズムベース	システムベース
性能	1.85	1.10	1.10
総ジョブ数	-	230.8	243.2
総実行時間	-	3h29m	2h16m
1 集計あたりの平均並列数	-	27.5	45.0

以上の内容を，実際の機械学習プログラムに対する AT に適用した．対象としたプログラムは，加藤らによる歩行者経路予測アプリケーション [3] である．性能パラメタは 5 つ，それぞれ取りうる値が 5 つずつであり，実行対象は合計 3125 存在する．また，実験は名古屋大学情報基盤センターのスーパーコンピュータ「不老」Type II サブシステムで実施した．それぞれの方式で AT を 5 回ずつ行った結果の平均を表 1 に示す．なお実験時，システムの最大並列数は常時 50 であった．

アルゴリズムベースの場合とシステムベースの場合で，AT 後の性能差は誤差の範囲だった．実行されたジョブの総数はシステムベースの方がやや多くなったが，1 集計あたりの平均並列数が 1.64 倍となり，総実行時間は 35% 短縮された．

5　おわりに

本研究では並列に実行する AT を対象に，よりシステムの並列性を活かす方式を提案した．実行対象列挙をシステムベースにすることで，AT 性能を維持したまま並列性の向上させることができた．

謝辞

本研究の一部は JSPS 科研費 JP23K11126 の助成により実施した．

参考文献

[1] 今村俊幸, 荻田武史, 尾崎克久, 片桐孝洋, 須田礼仁, 高橋大介, 滝沢寛之, 中島研吾, ソフトウェア自動チューニング―科学技術計算のためのコード最適化技術, 森北出版, 2021.

[2] 多部田敏樹, 藤家空太郎, 藤井昭宏, 田中輝雄, 加藤由花, 大島聡史, 片桐孝洋, マルチ GPU 環境における機械学習ハイパーパラメータの自動チューニング (1), 情処 83 回全国大会, 2021.

[3] R. Akabane, Y. Kato, Pedestrian Trajectory Prediction Based on Transfer Learning for Human-Following Mobile Robots, IEEE ACCESS, Vol.9, 2021.

縦長行列の QR 分解に対する様々なアルゴリズムの性能評価

深谷 猛 [1]
[1] 北海道大学
e-mail : fukaya@iic.hokudai.ac.jp

1　はじめに

本発表では，縦長行列 $A \in \mathbb{R}^{m\times n}$ ($m \gg n$) の Thin QR 分解（$A = QR$）を計算するアルゴリズムを扱う．ここで，$Q \in \mathbb{R}^{m\times n}$ は列直交行列，$R \in \mathbb{R}^{n\times n}$ は上三角行列である．なお，今回は A が（数値的に）列フルランクであることを仮定する．縦長行列の QR 分解は基本的な行列計算の一種であり，ベクトルの直交化等の応用を持つ．そのため，現在までに，高速化や高精度化を目的として，様々な特徴を持った異なるアルゴリズムが提案されている．本発表では，異なる特徴を持つ複数のアルゴリズムに関して，最近の計算機環境で実施した性能評価の結果を報告する．なお，本発表の内容は国際会議で発表した内容 [1] の主要部と，それ以降に得られた結果の一部を予定している．

2　縦長行列の QR 分解に対する代表的なアルゴリズム

行列の QR 分解を計算するアルゴリズムは，Orthogonal Triangularization 型と Triangular Orthogonalization 型の 2 種類に大別できる．前者は，直交変換を A の左側から作用させ，A を上三角行列に変形する．代表的なアルゴリズムとして，Householder QR アルゴリズムが知られている．一方，後者は，上三角行列を A の右側から作用させ，A を列直交行列に変換する．代表的なアルゴリズムとして，Gram-Schmidt の直交化に基づくアルゴリズムが知られている．

また，大規模分散並列環境では集団通信のコストが強スケーリングにおけるボトルネックとなることが多く，その観点で，通信回避（CA: Communication Avoiding）の重要性が指摘されている．上で挙げたアルゴリズムは，いずれも集団通信の回数が $O(n)$ であり，非通信回避型アルゴリズムに分類される．これに対して，Orthogonal Triangularization 型かつ通信回避型のアルゴリズムに該当する TSQR アルゴリズム [2] が提案されている．また，Triangular Orthogonalization 型で通信回避の特徴を持つアルゴリズムとしては，CholeskyQR 型のアルゴリズム [3] が該当する．これらのアルゴリズムは，集団通信の回数が $O(1)$ である．

計算精度の観点では，Orthogonal Triangularization 型のアルゴリズム（Householder QR や TSQR）は，A の条件数に関わらず，結果の精度が十分良いことが知られている．一方，Triangular Orthogonalization 型のアルゴリズム（Gram-Schmidt や Cholesky QR）は，A の条件数に応じて，結果の精度（Q の直交性）が悪化することが知られており，精度を改善するための手法（再直交化や前処理）が提案されている．

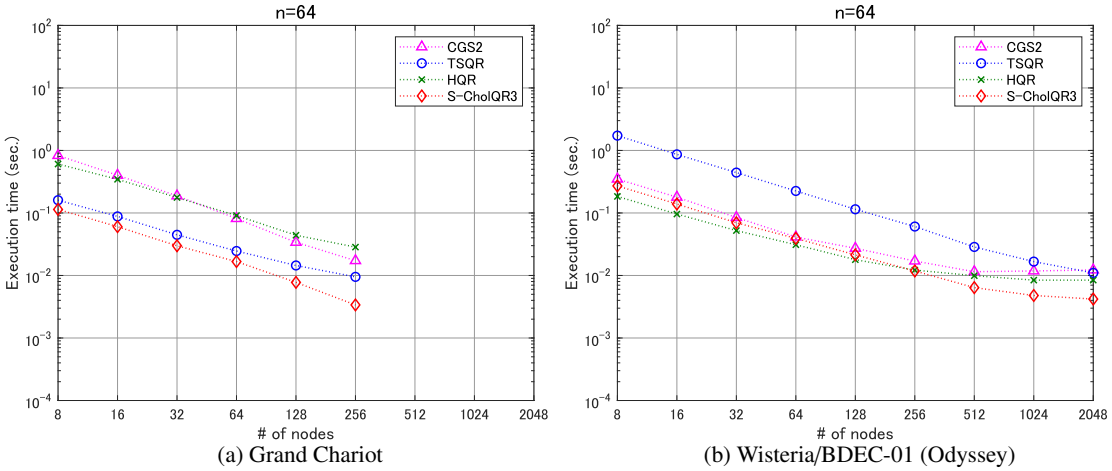
なお，近年，Randomized アルゴリズムが活発に研究されており，縦長行列の QR 分解に対しても有力なアルゴリズムが報告されている．詳細については，文献 [4] などを参照されたい．

3　性能評価の概要

今回の性能評価では，表 1 に挙げた 4 種類のアルゴリズムを対象とする．これらのアルゴリズムは，A の条件数が十分大きい場合でも，十分な精度で QR 分解を計算することができ，一方で，互いに異なる特徴を有している．図 1 に，2 種類のスーパーコンピュータシステムにおける 4 種類のアル

表 1: 評価対象のアルゴリズムと位置づけ.

	Orthogonal Triangularization	Triangular Orthogonalization
非 CA	Householder QR（HQR）	再直交化付き古典 Gram-Schmidt（CGS2）
CA	TSQR	Shifted CholeskyQR3（S-CholQR3）



(a) Grand Chariot (b) Wisteria/BDEC-01 (Odyssey)

図 1: 大規模分散並列計算機における性能評価結果の例（強スケーリング, $m = 16777216$）.

ゴリズムの実行時間の評価結果（強スケーリング）の一例を示す．ここで，Grand Chariot は北海道大学情報基盤センター，Wisteria/BDEC-01 (Odyssey) は東京大学情報基盤センターで運用されているスーパーコンピュータである．図より，各アルゴリズムの実行時間やノード数（並列数）に対する挙動が異なることが確認できる．その他の評価結果および結果の分析については，当日の発表で報告する予定である．

謝辞　本研究は JSPS 科研費（21K11909）および JHPCN・HPCI（jh240053）の支援を受けています。

参考文献

[1] T. Fukaya, Distributed Parallel Tall-Skinny QR Factorization: Performance Evaluation of Various Algorithms on Various Systems, in: Proc. of PDCAT 2022, pp. 275–287, 2023.

[2] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communicationoptimal Parallel and Sequential QR and LU Factorizations, SIAM SISC, Vol. 32 (2012), pp. A206–A239.

[3] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa, Shifted Cholesky QR for Computing the QR Factorization of Ill-Conditioned Matrices, SIAM SISC, Vol. 42 (2020), pp. A477–A503.

[4] M. Melnichenko, O. Balabanov, R. Murray, J. Demmel, M. Mahoney, P. Luszczek, CholeskyQR with Randomization and Pivoting for Tall Matrices (CQRRPT), arXiv: 2311.08316, 2024.

GPU クラスタにおける並列数論変換の実現と評価

高橋 大介 [1]
[1] 筑波大学計算科学研究センター
e-mail：daisuke@cs.tsukuba.ac.jp

1　はじめに

数論変換（number-theoretic transform，以下 NTT）は，離散 Fourier 変換を有限体に一般化したものであり、準同型暗号，多項式乗算，多倍長精度乗算などに広く用いられている．NTT の実装がいくつか提案されている [1, 2]．本論文では，GPU クラスタにおいて並列 NTT を実現し性能評価を行った結果について述べる．

2　数論変換

n 点 NTT は $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$（$p$ は素数）において以下のように表すことができる．

$$y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \bmod p, \quad 0 \le k \le n-1 \tag{1}$$

ここで，ω_n は 1 の原始 n 乗根である．式 (1) は高速 Fourier 変換（fast Fourier transform，以下 FFT）と同様のアルゴリズムを適用することで，演算回数を $O(n \log n)$ に削減することができる．Out-of-place FFT アルゴリズムとして知られる Stockham FFT アルゴリズムを基数 2 の NTT に適用すると，Algorithm 1 に示す基数 2 の Stockham NTT アルゴリズム [2] が得られる．

Algorithm 1 の 8 行目では剰余加算が行われ，9 行目では剰余減算と剰余乗算が行われる．Montgomery 乗算や Shoup 乗算を用いることで時間の掛かる除算を実質的に行うことなく，加算，減算，乗算，ビットマスク，およびシフト演算のみで剰余乗算を行えることが知られている．また，four-step FFT アルゴリズムとして知られる手法は NTT に適用可能であるので，MPI と OpenACC を用いて four-step NTT を並列化した．

3　性能評価

性能評価にあたっては, four-step NTT の GPU 実装, CPU と GPU 間の転送時間を含む four-step NTT の GPU 実装，および six-step NTT の CPU 実装の性能を比較した．測定に際しては，weak scaling における順方向 NTT を連続 10 回実行し，その平均の経過時間を測定した．GPU クラスタとして，筑波大学計算科学研究センターに設置されている Pegasus（120 ノード）の 1~32 ノードを用いた．コンパイラは NVIDIA HPC Compilers 23.9 を用いた．コンパイルオプションは GPU 実装では`-fast -acc=gpu -gpu=cc90` を，CPU 実装では`-fast -mp -tp=sapphirerapids` を指定した．MPI ライブラリは OpenMPI 4.1.5 を用いた．各ノードあたりの MPI プロセス数は 1，各 MPI プロセスあたりのスレッド数は 48 に設定した．N 点 NTT の Giga-operations per second（Gops）値は $(3/2)N \log_2 N$ より算出している．

並列 NTT の weak scaling 性能（$N = 2^{30}\times$ ノード数）を図 1 に示す．図 1 から分かるように，GPU 実装が CPU 実装よりも高速であるが，4 ノード以上において CPU と GPU 間の転送時間を含む GPU 実装は CPU 実装とほぼ同じ性能になっている．ノード数が増加するに従って全対全通信のメッセージサイズが小さくなり，通信バンド幅が小さくなる．したがって，GPU 実装と CPU 実装の性能差は，ノード数が増加するに従って小さくなる．

Algorithm 1 基数 2 の Stockham NTT アルゴリズム [2]

Input: $n = 2^q$, $X_0(j) = x(j)$, $0 \le j \le n-1$, and ω_n is the primitive n-th root of unity
Output: $y(k) = X_q(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \bmod p, 0 \le k \le n-1$

$l \leftarrow n/2$
$m \leftarrow 1$
for t **from** 1 **to** q **do**
  for j **from** 0 **to** $l-1$ **do**
    for k **from** 0 **to** $m-1$ **do**
      $c_0 \leftarrow X_{t-1}(k+jm)$
      $c_1 \leftarrow X_{t-1}(k+jm+lm)$
      $X_t(k+2jm) \leftarrow (c_0+c_1) \bmod p$
      $X_t(k+2jm+m) \leftarrow \omega_n^{jm}(c_0-c_1) \bmod p$
    end for
  end for
  $l \leftarrow l/2$
  $m \leftarrow 2m$
end for

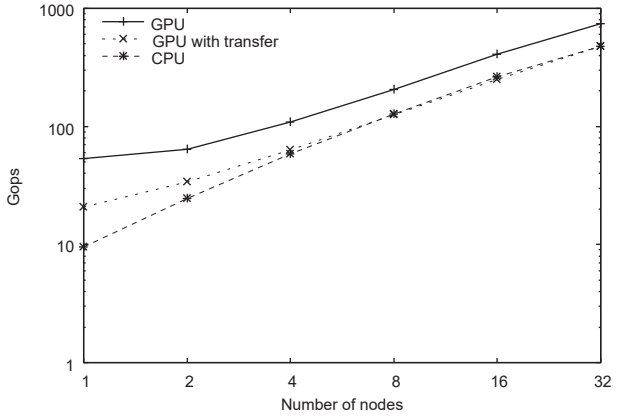


図 1. 並列 NTT の weak scaling 性能（$N = 2^{30}\times$ ノード数）

4 まとめ

本論文では，GPU クラスタにおいて並列 NTT を実現し性能評価を行った結果について述べた．NTT のバタフライ演算は，剰余加算，剰余減算，および剰余乗算を用いて行うことができる．MPI と OpenACC を用いて four-step NTT を並列化した．NVIDIA H100 PCIe GPU を搭載したノードからなる GPU クラスタの 32 ノードを用いて，2^{35} 点 NTT において 745 Gops 以上の性能を得ることができた．

謝辞 本研究成果は筑波大学計算科学研究センターの学際共同利用プログラム（Pegasus）を利用して得られたものである．本研究は，JSPS 科研費 22K12045 の支援によって行われた．

参考文献

[1] F. Boemer *et al.*, “Intel HEXL,” https://github.com/intel/hexl.

[2] D. Takahashi, “An implementation of parallel number-theoretic transform using Intel AVX-512 instructions,” in *CASC 2022*, ser. LNCS, vol. 13366. Springer, 2022, pp. 318–332.

