Vol.2022-HPC-183 No.3
2022/3/17

タンパク質立体構造予測システム AlphaFold の TSUBAME3.0 上での高速化

藤田 隼斗[1,a)]　野村 哲弘[2]　遠藤 敏夫[2,1]　関嶋 政和[1]

概要：タンパク質の立体構造の理解はタンパク質の機能の理解に役立ち，人体の働きを理解する基礎となる．ただし既知のタンパク質のアミノ酸配列に対して，実験的に立体構造が決定されているタンパク質は多いとは言えない．これらのギャップを埋めるために計算機によるアプローチが必要であり，そのアプローチの 1 つとして開発されたのが AlphaFold である．
AlphaFold は hhblits 等の既存のツールを使い，BFD などの巨大な遺伝子データベースから MSA を取得する．しかし，そのうちの 1 つの hhblits の実行に多大な時間がかかるという問題がある．この実行時間の一番の要因は I/O にあり，遺伝子データベースをどういったストレージに保管するかによって実行時間が大きく異なってくる．実際に TSUBAME3.0 の高速ストレージ領域にデフォルトの設定で保管した場合と Stripe 設定をして保管した場合とで実行時間が大きく異なっている．
そこで，本研究では TSUBAME3.0 上で選択できる遺伝子データベースの保管方法を変更し，実行時間を計測した．さらに，ボトルネックとなっている hhblits に対してプロファイリングを行い，その結果から並列化数のチューニングやデータベースの保存方法の変更，ソートの最適化を行った．また，AlphaFold の MSA 取得ツール間のデータ依存関係からツール実行を非同期に行うようにした．その結果，TSUBAME3.0 上で単一のアミノ酸配列から立体構造を予測する際の実行時間を平均して 2 分の 1 に短縮することに成功した．

1. はじめに

タンパク質のアミノ酸配列からそのタンパク質の立体構造を予測することは，「タンパク質のフォールディング (折り畳み) 問題」として知られている．

ただし既知のタンパク質のアミノ酸配列に対して，実験的に立体構造が決定されているタンパク質は約 1 万分の 1 である [1][2][3]．実験的にタンパク質の立体構造を推定する方法はいくつか存在するが，どの方法も多大なコストや時間がかかる．このような状況と近年の情報技術の発達を受けて，計算機によるアプローチ台頭してきている．そのような中で第 14 回タンパク質構造予測精密評価 (CASP14)[4] にて AlphaFold2[5] が登場し，他の手法を大きく上回る結果を残した．その精度は実験によるタンパク質の立体構造推定とほぼ同等である．

AlphaFold のソースコードは Apache License 2.0 でオープンソース化されており，学習済みパラメータはモデルパラメータは 2022 年 1 月 19 日より CC BY 4.0 のライセンスで公開されている [6]．遺伝子データベースは 2022 年 1 月 20 日より PDB70 のライセンスが CC BY 4.0 に変更された [7] ことによって，AlphaFold は商用利用を含めて誰でも利用できる状態である．

ただし，AlphaFold の実行には合計で 2TB 以上にも及ぶ巨大な遺伝子データベースが必要であり，長いアミノ酸配列の予測には大きな VRAM を持った強力な GPU が必要である．さらに高速に実行するためには巨大な遺伝子データベースを HDD よりも高速な SSD やシステムメモリ上に置く必要があり，記憶域の面でも高性能な計算機が求められる．AlphaFold でも，ある程度の品質をトレードオフにしてより小さい遺伝子データベースで実行するオプションがリリースされている [8]．

AlphaFold は TSUBAME3.0 でも 2021 年 8 月より利用できるようになっている [9]．ただし，TSUBAME3.0 では hhblits の実行時間が遅く，また実行時間にむらがあった．特に I/O に難があるとされており，遺伝子データベースの保管設定が変更された [10]．これによって hhblits の実行時間は大幅に改善されるが，依然としてそのツールが占め

[1] 東京工業大学情報理工学院
School of Computing, Tokyo Institute of Technology
[2] 東京工業大学学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology
[a)] fujita.h.ai@m.titech.ac.jp

ⓒ 2022 Information Processing Society of Japan

る実行時間は大きい．本研究では，hhblits の高速化を主として行う．その他のツールについては非同期実行することによって AlphaFold 全体の実行時間に関係することが少なくなるようにし，AlphaFold 全体の実行時間が小さくなるようにする．

2. AlphaFold

AlphaFold[5] は DeepMind によって開発されたタンパク質立体構造予測システムである．

AlphaFold の予測は大きく分けて 2 つステージで構成される．1 つは MSA(Multiple sequence alignment) を構築するステージ，もう 1 つは実際に立体構造を推論するステージである．1 つ目の MSA 構築ステージでは，既存のツールである jackhmmer, hhsearch, hmmsearch, hhblits を用いて MSA とテンプレートを構築する．これらには遺伝子データベースが必要で，jackhmmer[11] は Uniref90[12] と MGnify[13] を別々に実行し，hhsearch, hmmsearch は PDB70[14]，hhblits は BFD と Uniclust30[15] を同時に使用して実行する．2 つ目の推論ステージでは，Embedding モジュール，Evoformer モジュール，Structure モジュールで構成され，立体構造の推論を行っている．

2 つのステージで必要とするリソースは異なり，MSA 構築ステージでは CPU のみを用いて，推論ステージでは主に GPU を用いる．MSA 構築ステージでは巨大なデータベースに対してアクセスするため，I/O 性能も重要である．特に hhblits が BFD(1.7TB) を使用しており，I/O 性能によっては実行時間の大きな割合を占める．

2.1 TSUBAME3.0 上の AlphaFold

TSUBAME3.0 は東京工業大学のスーパーコンピュータである．TSUBAME3.0 は 540 台の計算ノードを備え，総計 12.15PFLOPS の演算能力を供給する．各計算ノードのスペックは，**表 1** の通りである．

ストレージ環境として高速ストレージ領域があり，これは Lustre ファイルシステムで構成されている [17]．Lustre とは並列分散ファイルシステムであり，多くの HPC で利用されている．Lustre に適するのは，大きなファイル，シーケンシャルアクセス，プロセス毎で異なるファイルに書き込む並列アプリケーションである．逆に苦手とするのは，大量の小さなファイル，莫大なメタデータ操作である．Lustre での実データは Object Storage Target(OST) と呼ばれるものに保管される．TSUBAME3.0 では 10 本の HDD(10TB 7200rpm) を用いた RAID6 で構成され，これが 68 ある．通常は 1 つのファイルは 1 つの OST に保存されるが，File Striping を使うことで複数の OST に分散してファイルが保存される．

AlphaFold の遺伝子データベースは高速ストレージ領域に Stripe されて保存されている．Stripe の設定は stripe_count が 68，stripe_size が 1MiB(1048576B) となっている．

以降では特に断りがない場合は計算ノードを 1 台使用するものとする．

```
>dummy_sequence
GWSTELEKHREELKEFLKKEGITNVEIRIDNGRLEVRVEGGTERLK
RFLEELRQKLEKKGYTVDIKIE
```

図 1 68 残基のダミーのアミノ酸配列

2.2 AlphaFold のボトルネック

Stripe されてないデータベースに対してはダミー配列を Stripe されているデータベースに対してはダミー配列と CASP14 の T1050 を用いてそれぞれ AlphaFold を実行する．

AlphaFold の標準エラー出力から各 MSA 取得ツールの実行時間とそれを除いた AlphaFold の推論の実行時間をまとめたものが**表 2** である．実行は AlphaFoldv2.0.1 を用いて，オプションに関しては TSUBAME3.0 の例 [18] にあるように実行した．(予測はモデル 1 つ) MSA 取得ツールについては標準エラー出力にある実行時間をそのまま使う．AlphaFold の推論部の実行時間については最初のログ出力と最後のログ出力を全体の実行時間とし，そこから各 MSA 取得ツールの実行時間を引いたものとした．

表 2 から実行時間の大部分を hhblits が占めていることや Stripe の有無というデータベースの保管方法の違いによって hhblits の実行時間が大きく改善されていることが分かる．ただし依然として hhblits の実行時間の割合は大きい．

T1050 は 779 残基とダミー配列に比べ配列が長く，実際のアミノ酸配列である．jackhmmer, hhsearch の実行時間は大きく変化せず，hhblits や AlphaFold の推論部の実行時間が増大している．これらの結果から TSUBAME3.0 上における AlphaFold のボトルネックは hhblits であると考えられ，AlphaFold を高速化するためには，hhblits を高速化する必要がある．

2.3 hhblits のボトルネック

hhblits[19] は HH-suite に含まれるツールの 1 つであり，hhsearch もその 1 つである．

HHblits(HMM-HMM–based lightningfast iterative sequence search) は非常に大きな遺伝子データベースをに対して，高速に MSA 検索を行うことができる．hhblits は名前にある通り，隠れマルコフモデル（HMM）を用いる．

図 2 は簡略な hhblits のワークフローである．hhblits が非常に大きなデータベースに対して，高速に MSA 検索を行うことが出来る訳は Prefilter にある．高速な Prefilter を用いることで，データベースを 1〜5%にフィルタリングをする．これによって Prefilter 以降の重い計算をデータベー

表 1 計算ノードのスペック [16]

CPU	Intel Xeon E5-2680 V4 (Broadwell-EP, 2.4GHz) × 2 ソケット ソケットあたり 14 コア, ノードあたり合計 28 コア
GPU	NVIDIA TESLA P100 for NVlink-Optimized servers × 4 基
メモリ	256GB (DDR4-2400 32GB モジュール × 8 本)
SSD	Intel DC P3500 2TB (NVMe, PCI-E 3.0 x4, R2700/W1800)
ネットワーク	Intel Omni-Path Architecture HFI (100Gbps) × 4 本

表 2 各ブロックの実行時間 (秒)

	no Stripe	Stripe	
	dummy	dummy	T1050
jackhmmer (uniref90)	345	339	432
jackhmmer (mgy_clusters)	406	408	434
hhsearch	230	608	464
hhblits	16264	1589	8456
alphafold(inference)	167	172	1478

表 3 主な関数の実行時間 (秒)

	dummy	T1050
prefilter_db	1029	1882
alignment	6	3233
perform_realign	0	1088
mergeHitsToQuery	—	1906
main(全体)	1153	8337

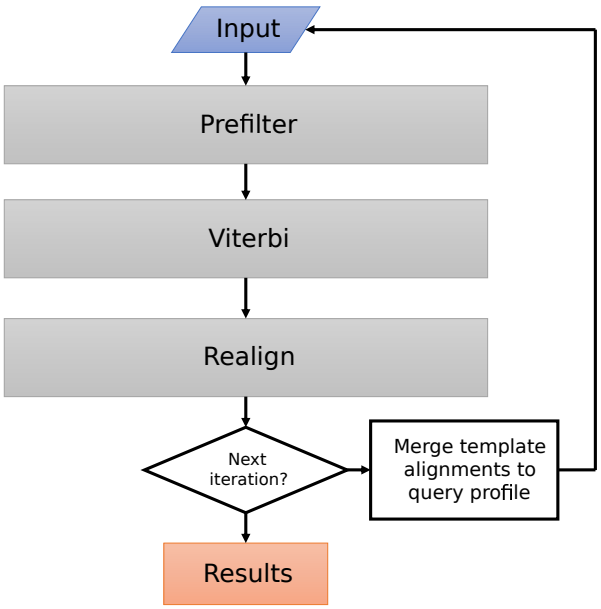

図 2 hhblits の簡略なワークフロー

```
$ module li
Currently Loaded Modulefiles:
  1) cuda/11.5.0                4) experimental
  2) openmpi/3.1.4-opa10.10     5) scorep/7.1
  3) gcc/10.2.0
```

図 3 Score-P を用いてビルドする際の環境

ス全体に対して行う必要がなくなる．

2 章より，hhblits の高速化をする必要がある．そのため，hhblits のボトルネックを解析する．Score-P[20] を用いてボトルネックとなる関数を特定する．

Score-P でプロファイリングする際は Score-P を含めてビルドする必要があり，ビルド時の環境は**図 3** を用いる．

Stripe されているデータベースに対して，ダミー配列と CASP14 の T1050 を入力配列として Score-P を含めてビルドした hhblits を実行する．

hhblits の本質的な処理が行われる関数は HHblits::run である．この関数で直接呼ばれる関数の中で，Master Thread での実行時間が大きい関数をまとめたものが**表 3** である．これらの 4 つ関数でどちらも全体の実行時間の 9 割ほどを占めている．実行されなかった関数は—で示した．

この 2 つの入力配列では関数の実行時間の傾向が大きく異なる．これは Prefilter を通過するアミノ酸配列の数が異なることに由来する．ダミー配列は配列に意味がなく，データベース内に似た配列は少なく Prefilter を通過する配列はごく少数である．そのため関数 prefilter_db 以外の実行時間が小さい．一方，T1050 はデータベース上に似た配列が数多く存在するために，関数 prefilter_db 以外の実行時間も相対的に大きい．ダミー配列と T1050 の関数 prefilter_db の実行時間は表 3 では差が大きいように見える．しかしこれは，AlphaFold での hhblits は 3 回のイテレーションを行う設定であるが，ダミー配列が 3 回のイテレーションをしていないためである．hhblits のログを解析すると，ダミー配列と T1050 の 1 回目の実行時間を見ると大きな差はない．また，関数 prefilter_db は 2, 3 回目に呼ばれる際の実行時間は 1 回目に呼ばれる際の実行時間よりも小さい．

ボトルネックの 4 つの関数の中で関数 prefilter_db, alignment は I/O 性能に強く影響を受ける．両者の関数の I/O アクセスは mmap を介して行われている．また，これらの関数がアクセスするファイルは異なる．ソースコード解析とデバッグの結果，データベースが Uniclust30(86GB) の場合，関数 prefilter_db では，合計で 4GB ほどのファイルにアクセスし，関数 alignment では少なくとも合計で 65GB ほどのファイルにアクセスする．

3. 手法

hhblits の高速化のために 3 つの手法を提案する．うち 2 つは TSUBAME3.0 上での高速化であるが，1 つは TSUBAME3.0 に依らない hhblits の高速化である．

ⓒ 2022 Information Processing Society of Japan

また，AlphaFold の MSA 取得の高速化のために各 MSA 取得を非同期に実行する手法を提案する．

3.1 hhblits の並列化数の適正化

hhblits は OpenMP でいくつかの関数が並列化されている．2.3 章で言及した 4 つの関数のうち関数 prefilter_db, alignment, perform_realign の 3 つの関数は実行時間的に大きな関数が並列化されている．

AlphaFold では hhblits の並列化数は 4 に設定されている．これを計算ノードの物理コア数の 28 に変更する．

hhblits は I/O に実行時間が大きく左右されるプログラムである．一般には I/O ボトルネックの場合に並列化数を上げることは，意味がない場合が多い．しかし TSUBAME3.0 上ではデータベースが Lustre ファイルシステム上で Stripe されて保管されているため，並列化数を上げることで I/O 性能が改善される場合がある．

3.2 hhblits のアクセスするデータベースの配置の変更

AlphaFold 中の hhblits は BFD(1.7TB) と Uniclust30(86GB) の 2 つのデータベースを使用している．関数 prefilter_db では BFD の 1.7TB のうち 18GB ほどのファイルのみにしかアクセスしていない．そのため，このファイルを高速ストレージから計算ノードの DRAM 上にコピーする．tmpfs ファイルシステムである/dev/shm にディレクトリを作成し，関数 prefilter_db がアクセスするファイルをコピーする．残りのファイルはシンボリックリンクを用いて高速ストレージにリンクする．

tmpfs は使用した分だけ DRAM 容量を消費するため，使用できる DRAM 容量が $235 - 18(\text{BFD}) - 4(\text{Uniclust30}) = 213\text{GB}$ に減少する．

また，DRAM 上に配置するためのコピー時間は 15 秒ほどである．

3.3 hhblits のソートの高速化

並列化が行われていない関数 mergeHitsToQuery を高速化する．この関数のコールツリーの子孫に関数 Filter2 があり，これが関数 QSortInt を呼び出している．

関数 QSortInt はクイックソートで実装されたユーザ定義のソート関数である．入力配列が T1050 の場合は，この関数だけで 900 秒ほどの実行時間を占めている．この関数呼び出しを c++標準のソート関数 std::stable_sort に置き換える．

この手法は TSUBAME3.0 に依存しない変更である．hhblits は Github 上で hh-suite というリポジトリで公開されており，この変更の Pull Request を提出済みである [21].

3.4 AlphaFold の MSA 取得の非同期実行

AlphaFold の MSA 取得は**図 4** のように同期的に実行している．各 MSA 取得ツール間の依存関係は hhsearch(hmmsearch) の実行が jackhmmer(UniRef90) の結果に依存しているのみであり，その他には依存関係は存在しない．これに着目して，**図 5** のように MSA 取得を非同期に実行する．実装は concurrent.futures.ThreadPoolExecutor を用いて，実装した．

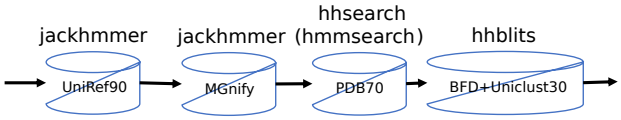

図 4 MSA 取得のパイプライン

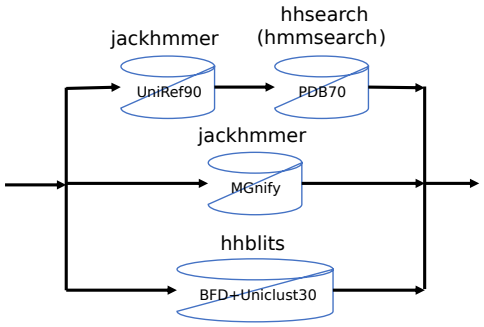

図 5 非同期化した MSA 取得のパイプライン

この手法によって，AlphaFold の MSA 取得全体の実行時間が 1 番大きいパスのみに依存するようになる．ただし，CPU, メモリ，I/O の必要量が増加する．また 3.1 章の手法を用いる場合，必要とされるコア数が最大 20 から 44 になる．

そのため，非同期化を行う前に比べて各 MSA 取得の実行時間が大きくなる可能性が考えられる．

4. 実験

4.1 実験環境

本実験では TSUBAME3.0 の計算ノードを 1 台用いる．AlphaFold は v2.1.1 を使用．実行の際は TSUBAME3.0 で提供されている run_alphafold.sh を用いて実行する．実行オプションは-a 0,1,2,3 –max_template_date=2020-05-14 とする．

ただし，run_alphafold.sh は v2.1.0 での preset, model_names の廃止と db_preset, model_preset の追加という変更に対応していないため，可能な限り似た動作するように変更する．run_alphafold.sh では preset のデフォルトの値が full_dbs になっている．よって，db_preset は full_dbs を指定し，model_preset は指定せずデフォルトを使用する．(予測はモデル 5 つ)

また，AlphaFold では hhblits のログが捨てられているが，ファイルに出力するようにソースコードを変更する．

ベースラインは，以上の変更を加えた run_alphafold.sh を実行する．

提案手法の hhblits は 3.3 章で述べた通りに実装し，**図 6**

```
$ module li
Currently Loaded Modulefiles:
  1) cuda/11.5.0                3) gcc/10.2.0
  2) openmpi/3.1.4-opa10.10
```

図 6 hhblits をビルドする際の環境

表 4 実験に用いたデータセット

Target ID	残基数
T1024	408
T1029	125
T1032	284
T1041	242
T1047s1	232
T1047s2	365
T1053	580
T1061	949
T1087	93
T1095	665

の環境でビルドして使用する．

また，データベースには高速ストレージを使用しているが，その影響を同じにするために，同時に高速ストレージにアクセスすることがないようにした．さらに実験は 3 回ずつ行い，中央値を用いる．

4.2 実験データセット

実験に用いるデータセットとして，CASP14 で用いられたものから長さに偏りがないように適当なものを 10 個用いる (**表 4**)．

5. 実験結果

ベースラインと提案手法の結果が**図 7** である．提案手法では，MSA 取得は jackhmmer(mgy_cluster), jackhmmer(uniref90) + hhsearch, hhblits のうち一番時間がかかるツールのみに実行時間が依存するため，その他のツールの実行時間は表示していない．

AlphaFold の実行時間が 10 配列のそれぞれの中央値で平均 2 分の 1 に短縮された．ベースラインの場合は，hhblits が実行時間で 1 位か 2 位になっており，少なくとも 4 割程度を占めている．提案手法の場合は T1024, T1061 を除いて，hhblits が合計の実行時間に寄与していない．T1061 でも hhblits の実行時間は alphafold 推論部の 2 分の 1 ほどとなっており，hhblits がボトルネックという状態にはなっていない．T1024 については提案手法の段階でも hhblits の実行時間が 6000 秒ほどある．

hhblits のログから各配列の大まかなボトルネックを特定する．関数 prefilter_db の一部を含む，“Prefiltering database” から “HMMs passed 1st prefilter (gapless profile-profile alignment)” を含む行のログ間の実行時間に着目する．Score-P によるプロファイリングの結果から，これは関数 prefilter_db の実行時間に近い値となる．T1024, T1061 を除く配列は，この箇所が 9 割ほどの実行時間を占めている．そのため，これらの配列は関数 prefilter_db がボトルネックと言える．提案手法では，これらの配列の hhblits 実行時間は隠蔽されるほど高速化されいるが，これは手法 1, 2 の高速化の影響が大きい．

表 5 手法 1, 2 の関数 prefilter_db の実行時間 (秒)

	なし	手法 1	手法 2	手法 1 + 2
T1024	1589	311	358	97

また，手法 1, 2, 4 によって hhblits 以外の MSA 取得ツールが扱える計算リソースが減少している．しかし，jackhmmer(mgy_cluster), jackhmmer(uniref90), hhsearch の両者の実行時間を見る限り，大きな差はなかった．

6. 各手法の寄与

提案手法の適用後でも hhblits の割合が大きい T1024 を用いて，各手法の寄与度を調査する．

まず手法 4 は，実験結果から hhblits 以外のツールに影響がなかったため，単純に 1 番実行時間の小さいパスと 2 番目に小さいパスの実行時間が削減されると考えられる．hhblits 以外のツールに絶対的に大きな実行時間の差は見られないため，hhblits の実行時間が中程度の配列で効果的な手法となる．

手法 3 は hhblits の関数 mergeHitsToQuery に対する高速化である．関数 mergeHitsToQuery に対してはこの手法以外で変更を行っていないため，関数 mergeHitsToQuery のみに着目すれば良く，元の関数 QSortInt の実行時間とおよそ等しい時間が短縮されている．この関数はデータベースにヒットする配列が多い入力配列で大きな実行時間を占める．そのような入力配列は全体の実行時間が比較して大きいため，実行時間が大きい配列で効果的な手法となる．

関数 prefilter_db は手法 1, 2 で高速化される．**表 5** に手法を適用した際の実行時間を示す．片方の手法を適用した時点で 300 秒台まで高速化されている．両方組み合わせることでさらに高速化されるが，T1024 の hhblits の実行時間から見ると 5%以下の改善である．関数 prefilter_db が大きな割合を占める入力配列では，両方適用することで hhblits が 2 倍以上高速化される可能性はあるが，手法 4 が存在するため，両方の適用は意味がない．ただし，高速ストレージの I/O 状況によっては手法 2 のようにコピーすることは役立つ可能性がある．

手法 1 の関数 prefilter_db を除く関数 aligmnet, perform_realign について，関数 alignment は I/O の影響を強く受けるが 3000 秒ほどかかっていた実行時間が 1.3〜2 倍ほど高速化し，関数 perform_realign は 660 秒から 110 秒に高速化した．またこれは手法 3 と同様に実行時間が大きい配列で効果的な手法となる．

ⓒ 2022 Information Processing Society of Japan

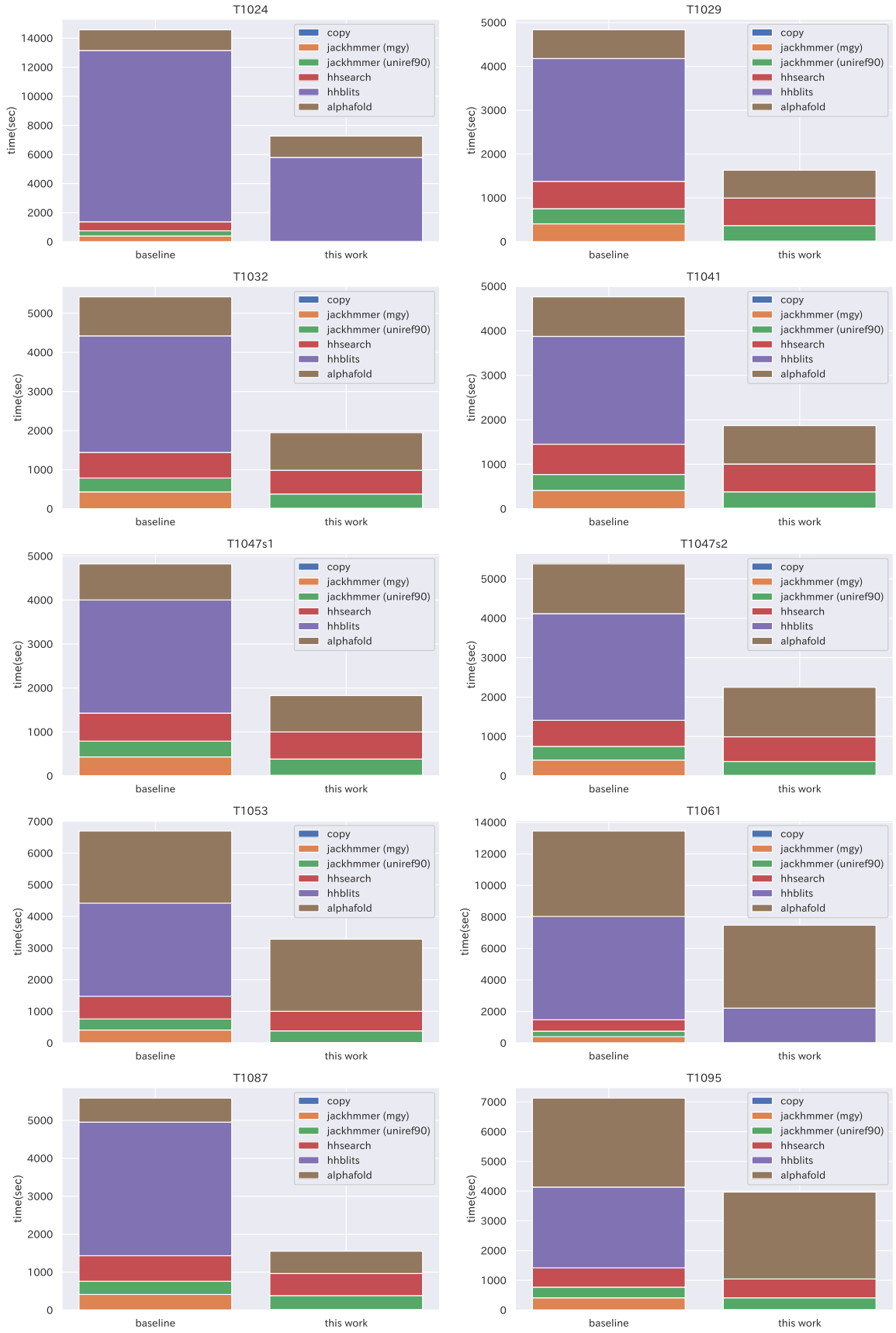

図 7 各配列の実行時間とその分布

I/O に強く影響を受ける 2 つの関数 (prefilter_db, alignment) では，手法 1 による高速化の寄与度が異なっている．関数 prefilter_db では 5 倍ほど高速化されているが，関数 alignment では最大で 2 倍となっている．関数 alignment

については高速ストレージを使わずに計算ノードの SSD を用いることで 100 秒ほどまで高速化されるため，手法 1 適用後でも I/O ボトルネックである．手法 1 は共に I/O ボトルネックとみられる関数であっても，高速化の影響は関数によって異なる．

7. おわりに

本研究では，AlphaFold を高速に実行するために hhblits の高速化を行った．Score-P を用いたプロファイリングやソースコード解析を行い，ボトルネックとなった箇所を修正していった．修正箇所は並列化数の上昇，データベースの配置の変更，ソートの標準ライブラリ化である．「並列化数の上昇」はコア数が 4 以上かつ I/O 性能に余裕がある計算機に有用であり，「データベースの配置の変更」はリモートストレージがある計算機でローカルストレージとの I/O 性能に開きがある場合に有用である．最後の「ソートの標準ライブラリ化」は計算機による制限のない変更と考えられる．また，MSA 取得の非同期化を行い，AlphaFold の MSA 取得の高速化を図った．これにより，単一の入力配列に対する AlphaFold の実行時間が 2 分の 1 に短縮された．また多くの入力配列で hhblits の実行時間はボトルネックとは言えない段階まで高速化された．

謝辞 本研究は，東京工業大学のスーパーコンピュータ TSUBAME3.0 を利用して実施しました．また，本研究の一部は，国立研究開発法人日本医療研究開発機構（AMED）創薬等ライフサイエンス研究支援基盤事業 創薬等先端技術支援基盤プラットフォーム（BINDS）の課題番号 JP21am0101112 の支援を受けました．

参考文献

[1] wwPDB Consortium. Protein Data Bank: the single global archive for 3D macromolecular structure data. Nucleic Acids Res. 47, D520–D528 (2018).
[2] Mitchell, A. L. et al. MGnify: the microbiome analysis resource in 2020. Nucleic Acids Res. 48, D570–D578 (2020).
[3] Steinegger, M., Mirdita, M. & Söding, J. Protein-level assembly increases protein sequence recovery from metagenomic samples manyfold. Nat. Methods 16, 603–606 (2019).
[4] Moult, J., Fidelis, K., Kryshtafovych, A., Schwede, T. & Topf, M. Critical assessment of techniques for protein structure prediction, fourteenth round. CASP 14 Abstract Book https://www.predictioncenter.org/casp14/doc/CASP14_Abstracts.pdf (2020).
[5] Jumper, J., Evans, R., Pritzel, A. et al. Highly accurate protein structure prediction with AlphaFold. Nature 596, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2
[6] Update license of AlphaFold parameters. · deepmind/alphafold@8173117 https://github.com/deepmind/alphafold/commit/8173117130e6df8749ab7349722ec465666df548 (Accessed on 01/27/2022)
[7] Martin Steinegger on Twitter https://twitter.com/thesteinegger/status/1484165053402464259 (Accessed on 01/27/2022)
[8] Release AlphaFold v2.0.1 · deepmind/alphafold https://github.com/deepmind/alphafold/releases/tag/v2.0.1 (Accessed on 01/23/2022)
[9] Alphafold2 公開のお知らせ — TSUBAME 計算サービス https://www.t3.gsic.titech.ac.jp/node/454 (Accessed on 01/15/2022)
[10] TSUBAME Computing Services on Twitter https://twitter.com/Titech_TSUBAME/status/1453594879007727617 (Accessed on 01/15/2022)
[11] Johnson, L. S., Eddy, S. R. & Portugaly, E. Hidden Markov model speed heuristic and iterative HMM search procedure. BMC Bioinformatics 11, 431 (2010).
[12] Suzek, B. E., Wang, Y., Huang, H., McGarvey, P. B. & Wu, C. H. UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. Bioinformatics 31, 926–932 (2015).
[13] Mitchell, A. L. et al. MGnify: the microbiome analysis resource in 2020. Nucleic Acids Res. 48, D570–D578 (2020).
[14] Steinegger, M. et al. HH-suite3 for fast remote homology detection and deep protein annotation. BMC Bioinformatics 20, 473 (2019).
[15] Mirdita, M. et al. Uniclust databases of clustered and deeply annotated protein sequences and alignments. Nucleic Acids Res. 45, D170–D176 (2017).
[16] TSUBAME3.0 ハードウェア・ソフトウェア仕様 https://www.gsic.titech.ac.jp/sites/default/files/spec30j.pdf (Accessed on 01/15/2022)
[17] TSUBAME3 Lustre Seminar 公開用_20191028 https://www.t3.gsic.titech.ac.jp/sites/upload/TSUBAME3_Lustre_Seminar.pdf (Accessed on 01/15/2022)
[18] 7. フリーウェア - TSUBAME3.0 利用の手引き https://helpdesk.t3.gsic.titech.ac.jp/manuals/handbook.ja/freesoft/#alphafold (Accessed on 01/15/2022)
[19] Remmert, M., Biegert, A., Hauser, A. et al. hhblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. Nat Methods 9, 173–175 (2012). https://doi.org/10.1038/nmeth.1818
[20] Knüpfer, Andreas, et al. “Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir.” Tools for High Performance Computing 2011. Springer, Berlin, Heidelberg, 2012. 79-91.
[21] Use std::sort instead of QSortInt by fuji8 · Pull Request #307 · soedinglab/hh-suite https://github.com/soedinglab/hh-suite/pull/307 (Accessed on 01/27/2022)

ⓒ 2022 Information Processing Society of Japan