情報処理学会論文誌　コンピューティングシステム　Vol.13　No.2　13–27 (Sep. 2020)

大規模メタゲノムデータに対する分散並列相同性検索システムGHOSTZ PW/GFの提案

町田 健太[1,a)]　建部 修見[2,b)]

受付日 2020年1月14日, 採録日 2020年5月11日

概要：メタゲノム解析では，ある環境サンプルから直接得られた大量のゲノム DNA を一度に用いて環境中に含まれる微生物の群集構造や機能の解析を行う．メタゲノムデータは，代表的なビッグデータの 1 つであり，次世代シーケンサの性能向上や普及により得られるデータの量は年々増大している．メタゲノム解析の過程で行われる相同性検索とは，未知の DNA 配列をクエリとして，既知の DNA 配列データベースから相同なものを収集する手法である．現在までに様々なアルゴリズムを用いた相同性検索ツールが開発されているが，検索対象のクエリだけでなく非検索対象の DB までもが大規模なものになっている現在において，既存のツールでは実行時間の肥大化と実行時のメモリ不足が問題となってしまう．本研究では，クエリと DB の両方をいくつかのチャンクに分割してそれらを入力とする相同性検索を並列に実行することでこれらの問題を解決する，分散並列相同性検索システム GHOSTZ PW/GF を提案し実装した．結果として，提案システムは，Apache Hadoop や Spark を用いた関連研究との比較では同等以上の性能を示し，TSUBAME3.0 を 180 ノード用いた実験では，62 GB の DB と 71 GB のクエリに対して約 2 時間で相同性検索を実行した．また，ストロングスケーリングに関する実験では比較的小規模なデータセットに対して高いスケーラビリティを示し，システムの有用性を確認した．

キーワード：メタゲノム解析，分散並列相同性検索，ノードローカルストレージ，ワークフローエンジン，大規模実験

Proposal of Distributed Parallel Homology Search System GHOSTZ PW/GF

Kenta Machida[1,a)]　Osamu Tatebe[2,b)]

Received: January 14, 2020, Accepted: May 11, 2020

Abstract: In metagenomic analysis, a large amount of genomic DNA directly obtained from an environment is used at once to analyze the structure and function of the microbial community in the environment. Metagenomic data is one of the big data now, and the amount of the acquired data is increasing year by year owing to improved performance and extensive usage of next-generation sequencers. Homology search, which is used for genome analysis, is a technique in which homologous DNA from a known DNA sequence database is collected using unknown DNA sequences as a query. Although many homology search tools using various algorithms have been developed to accelerate processing, the conventional method suffers from performance issues and memory shortage. In this study, we proposed and implemented a distributed parallel homology search system GHOSTZ PW/GF using Gfarm, a distributed file system, and Pwrake, a dynamic workflow engine and evaluated them in TSUBAME3.0. The results indicate the high scalability of the proposed system; additionally, using a prebuilt non-redundant database comprising approximately 100 million records and sequence data comprising approximately 500 million records, the proposed system completed the execution of all processes on 180 nodes in approximately 2 hours.

Keywords: metagenomic analysis, distributed and parallel homology search, node-local storage, workflow engine, large-scale experiment

1. はじめに

細菌は地球上の生物を含むあらゆる環境中に存在しており，周囲の環境と共生・総合扶助の関係を持ち，独自のエコシステムを築き上げている．ヒトは自身の細胞数の 10 倍以上に相当する細菌を有しているとされており，それらの細菌叢はヒトが食べた物を栄養分に代謝したり，病原性細菌からの感染を防いだりと，ヒトの健康状態の維持や発病の抑制に関与している [1]．一方，これらの細菌叢の異常が人体に及ぼす影響も大きく，生活習慣病やがん，アレルギー，自閉症等，様々な疾患の原因となることが明らかになっている．そのため，細菌叢の群集構造や機能を明らかにすることがこれらの疾患の治療や予防につながると期待されているが，地球上のほとんどの細菌は培養が困難であるため，従来の手法では解析が容易ではなかった．

メタゲノム解析は，ある環境中から直接得られたゲノム DNA を用いて，その集合構造を明らかにすることにより，遺伝子プールの変動や環境との相互作用の解明を可能にしたゲノム解析手法である [2]．メタゲノム解析は細菌に対する培養過程を経ずに細菌叢から直接そのゲノム DNA を調製することが可能であるため，ヒトの体内の微生物群集構造を明らかにする手段として有用である．また，メタゲノム解析は数千種類もの生物を同時に解析することが可能な次世代シーケンサー（NGS）テクノロジーと親和性が高く，近年その関連論文数は増加し続けており [3]，メタゲノム関連市場のさらなる活性化も予測されている [4]．

メタゲノム解析は，一般に以下のようなプロセスを通して解析される．

(1) メタゲノムデータとして被験者や自然環境といった環境中からサンプルを抽出する．
(2) NGS を用いて抽出されたサンプルから細菌叢 DNA を直接シーケンシングし，その結果を品質等のメトリクスによってフィルタリングする．
(3) 先の過程で得られた DNA を用いて，KEGG [5], [6] や COG [7] 等が提供するリファレンスデータベース（DB）に対して類似度や相同性の検索を行い，各遺伝子の機能や代謝経路等の情報を収集し，含まれる種の構成と存在量等を知る．このステップは相同性検索と呼ばれる．
(4) 必要に応じて，さらなる解析や可視化を実施し，環境中の微生物や細菌の群集構造や機能を推測する．

[1] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate school of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[a)] machida@hpcs.cs.tsukuba.ac.jp
[b)] tatebe@cs.tsukuba.ac.jp

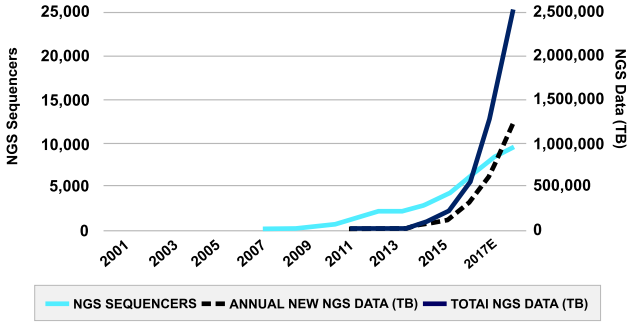

図 1　精密医療分野におけるゲノムデータサイズの推移
Fig. 1　The changes in size of genomic data in precision medicine research.

図 1 は精密医療分野における NGS によって解析されたゲノムデータ量を示すグラフ [8] であるが，このグラフから分かるように NGS の性能向上や普及により，近年サンプルから取得されるゲノムデータの量は爆発的に増大しており，DNA の相同性検索ステップにおいて，従来の解析手法では膨大な時間を要してしまう点が問題となっている．

2. 背景

相同性検索ツールとして，処理の高速化を目指して BLAST [9] や GHOSTX [10]，GHOSTZ [11]，DIAMOND [12] 等といった様々なアリゴリズムを用いたソフトウェアの開発が現在に至るまで行われている．しかし，これらのツールは単一ホスト上での実行を想定して開発されており，増大するデータ，特にメタゲノムデータのような大規模な入力に対してメモリの枯渇や実行時間の肥大化が問題となってしまう．NGS の誕生にともなってクエリとして用いられるデータ量が増大する一方，クエリをいくつかのチャンクに分割しこれをスーパーコンピュータのような分散メモリシステム上で並列処理することで高速化を目指す研究が行われている．たとえば，GHOST-MP は理化学研究所によって運用されていたスーパーコンピュータ「京」や東京工業大学により運用されていた TSUBAME2.5 上での動作を想定して作成されており，実際に京において 49,152 コアを用いて実験が行われている．これらの研究の詳細は 3 章に示すが，これらの研究に見られるように，ゲノム解析におけるスーパーコンピュータの利用はもはや不可欠なものであり，各遺伝子研究機関においては独自のスーパーコンピュータを保持している場合が多い．たとえば，日本国内の例をあげると，国立遺伝学研究所では 2019 年の 3 月から可動を開始した 204 ノード（内 64 ノードが Tesla V100 を搭載し InfiniBand により内部接続）から構成されるスーパーコンピュータシステムを保有しており，東京大学のヒトゲノム解析センターでは，2019 年の 4 月から運用を開始した 335 ノードからなる Shirokane5 と呼ばれるスーパーコンピュータを保有している．

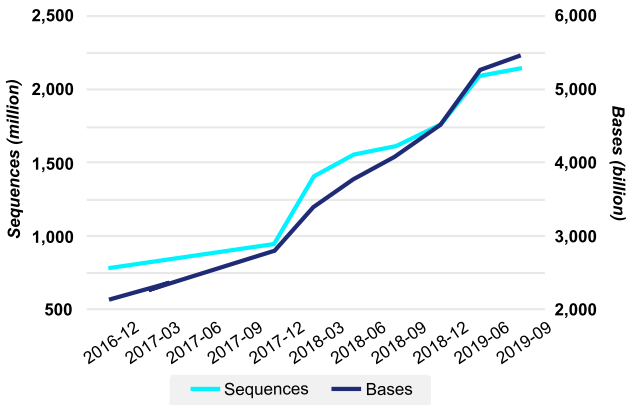

図 2　INSDC における DB サイズの推移
Fig. 2　The changes in size of genomic DB in INSDC.

関連研究に多く見られるクエリを分散する手法は，クエリに関して理論上使用ノード数を増加させれば相同性検索時のメモリ不足は発生しない．しかし，多くの未知のゲノム配列が解析され DB に反映されるサイクルが繰り返される中，**図 2** に示すように DB 自体の塩基対およびシーケンス数が 1 つのサーバのメモリに収まりきらない程に増大しており [13]，DB に対するメモリ不足の問題とさらなる実行時間の肥大化が懸念される．

そこで，本研究ではクエリと DB の両方をチャンクに分割しそれらの全組合せに対して相同性検索を並列に実行することでこれらの問題を解決する，分散並列相同性検索システム GHOSTZ PW/GF を提案し実装した．このシステムは，相同性検索の際に GPU を利用しかつファイル I/O に関してクラスタの各ノードに固有の NVMe SSD 等の高速なストレージデバイスを最大限活用する設計であるため，近年のスーパコンピュータのアーキテクチャのトレンドに追従したシステムであるといえる．本研究では，このシステムを用いて関連研究との比較や TSUBAME3.0 を用いた大規模データに対する相同性検索の実験を行った．以降の章では，著者らの過去の論文 [14] を基に関連研究や提案システムの概要，これらの実験結果についてより詳細に記述する．

3.　関連研究

相同性検索を分散並列実行環境に拡張することで，処理の高速化とメモリの枯渇問題を解決する研究がいくつか行われている．分散処理フレームワークとして有名なものに MPI や MapReduce，Spark があり，これらを用いた研究が多い．MPI を用いた実装では，その利点として細かな部分まで性能をチューニングすることができるため，比較的少量のデータセットに対する実行に関しては，一般に Hadoop や Spark における実行よりも高速に動作させることが可能である．しかし，MPI はメッセージパッシングによる並列計算のためのインタフェースであるため，所謂ビッグデータと呼ばれるような大規模なデータセットに関して，データアクセス性能に問題を生じる可能性がある．Hadoop や Spark はファイルシステムとして HDFS を採用し，ファイルのローカリティを活用する設計となっているため，高速なローカルストレージを効率的に利用することで，通信がボトルネックとなりにくいという利点がある．Hadoop はプログラミングモデルとして MapReduce を用いているのに対して Spark では RDD というプログラミングインタフェースを提供しデータ単位での並列化がより容易に実現可能である．ゲノム解析のパイプラインはいくつものツールをそのつどファイルを読み込み，中間結果を出力しながら実行していくため，近年バイオインフォマティクスにおいて，MPI や Hadoop よりも制限されずに並列化が可能である Spark を用いたツールが多く開発されている [15]．以下では，これらのフレームワークを用いた研究についてその概要を説明する．

3.1　mpiBLAST

mpiBLAST [16] は，MPI を用いて BLAST の処理を並列化することで処理の高速化を達成している．mpiBLAST は BLAST と比較して 2 つの利点がある．1 つ目は DB を分割してノード上に分散することによって，それぞれのチャンクサイズが小さくなりバッファキャッシュに存在しやすくなるため，ディスク I/O の削減によるかなりの高速化が期待できる点である．2 つ目は，DB のセグメンテーションによって大量のプロセス間通信の発生を防いでいるため，効果的にコモディティ Linux クラスタのアーキテクチャを活かすことができる点である．なお，DB はセグメンテーション化するがクエリは分割しない．

結果として，mpiBLAST は BLAST との比較においてノード数に対して線形に近い並列化効率を示し，全体の実行時間に占める通信時間や結果のマージおよび出力時間が BLAST の検索時間に隠蔽されることにより，少なくとも 100 ノードに対して性能がスケールアウトしている．

3.2　GHOST-MP

GHOST-MP [17] は GHOSTX のアルゴリズムを MPI ライブラリを用いて並列化することで，京や TSUBAME3.0 等の分散メモリシステム上での大規模並列検索に用いられるツールである．ハイブリッド並列型の相同性検索ツールであり，ノード間の並列化には MPI を，ノード内のタスク並列化には OpenMP を用いている．ノードレベルではマスターワーカモデルを採用し，入力クエリファイルがワーカープロセス数分のチャンクに分割され，マスタープロセスがそれぞれのワーカープロセスにクエリのチャンクを割り当てる．次に，ノード内ではクエリのチャンクがいくつかのシーケンスごとに分割され，それらがタスクの入力データとしてキューにエンキューされる．さらに，OpenMP のスレッドがロックを獲得してタスクをデキュー

することで相同性検索の並列実行を実現している．

実験では，京を用いて BLAST の MPI 実装である mpiBLAST と比較を行っている．データセットとして DB に 8,578,853 アミノ酸シーケンスから成る KEGG GENES データベースを，クエリには HMP DACC からダウンロード可能なヒト口腔メタゲノムデータを 107 サンプル用いており，京を 24,756 コア用いた実験では約 1.73 時間でこれらの全データに対して相同性検索を実行した．しかしこのコア数付近で並列化効率の理想値との差が大きくなり，これ以上のリソース増加による高速化の余地はないと結論付けられている．この理由として，マスタとワーカとの P2P 通信の衝突が原因であると考察されている．結果として，このツールは BLAST-MPI と比べて高速に動作し高いスケーラビリティを示したが，メモリ使用量の見積もりを課題としている．

3.3 SparkBLAST

SparkBLAST [18] は，BLAST を Spark を用いて分散並列実行するというものであり，クエリを複数に分割しクラスタ上に分散するが DB ファイルに関しては分割せずに各ノードで共有する．Spark の pipedRDD を用いることで BLAST を実行するジョブをクラスタ上の各ノードで実行する．Google Cloud と Microsoft Azure を用いて，Hadoop ベースの相同性検索システムである CloudBLAST と比較を行っており，64 ノードにおいてクエリに 805 MB のものと 11 GB のものを用いた実験では，Cloud BLAST に対し高速に動作し，使用メモリの効率も優れていることを示している．CloudBLAST よりも性能が優れている大きな要因として，Spark の RDD によるインメモリ処理とそれに起因する I/O 時間の削減をあげている．なお，クエリの分割にともなう断片的な相同性検索の結果ファイルを集約するタスクは実装されていない．

3.4 HAMOND

HAMOND [19] は，DIAMOND を Hadoop を用いて分散並列処理するというものである．Amazon Web Services（AWS）の Amazon EMR と互換性があり，また GUI での操作が可能であるように，専門知識のない生物学者向けの UI 設計がなされている．なお，DB は分割せずにクエリのみを分割および分散して DIAMOND を実行する．そのため DB に関しては全ノードで共有され，Mapper タスクは分割されたクエリの数と同数生成される．実験では，AWS を用いて 100 ノード上でシングルホストでの DIAMOND の実行（10 スレッド）と比較しており，HAMOND において 100 スレッドを用いた際に約 10 倍高速であることを示している．また，100 スレッドから 400 スレッド（100 ノードにおいて単一 DIAMONDO タスクあたり 4 スレッド使用）にかけて性能がスケールアウトしたことを報告している．

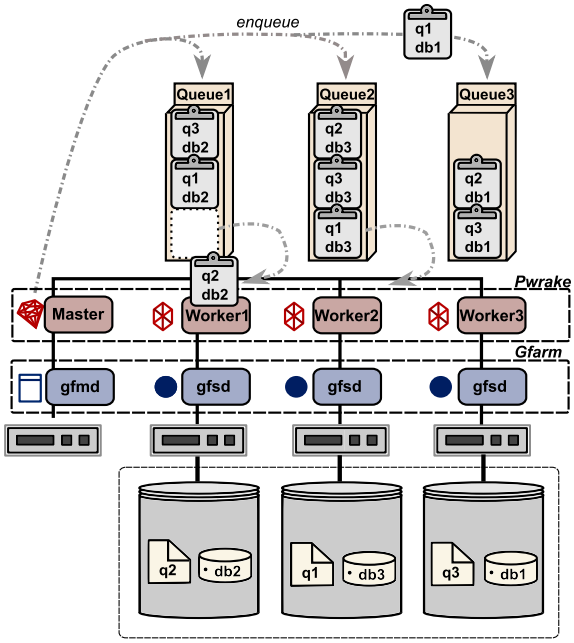

図 3　GHOSTZ PW/GF の構成
Fig. 3　System components of GHOSTZ PW/GF.

4. 提案システム

本研究における提案手法は，相同性検索に用いるクエリと DB の両方をチャンクに分割し，すべての組合せに対して相同性検索タスクを並列に実行するというものである．DB はそれぞれのチャンクごとに独立して構成し，相同性検索の完了後にそれぞれの DB を用いたタスクにおける結果ファイルを集約する．入力ファイルのチャンクへの分割により，実行時のメモリ不足によるエラーは回避され，タスクを並列に実行することが可能になり処理が高速化される．また，増大し続けるゲノムデータに対してシステムは高いスケーラビリティが要求される．相同性検索の分散並列実行において，スケーラビリティを低下させる大きな要因は大量の通信の発生による低速な I/O であるため，I/O 性能がスケールアウトするような実行基盤が必要である．また，ゲノム解析は複数のツールを連続的に用いて処理するような場合が多いため，将来的な拡張のために，MapReduce のようなプログラミングモデルに制限されずに，タスクレベルで並列処理が可能であることが望ましい．

そこで，本研究では広域分散ファイルシステムである Gfarm [20] と Gfarm と親和性の高い動的ワークフローエンジンである Pwrake [21] を用いた実行基盤を用いて，相同性検索を分散並列実行するシステムを提案する．提案システムのシステム構成を**図 3** に示す．

Gfarm ファイルシステムは，メタデータサーバ（MDS）とファイルサーバ（FSN）で構成される．実際の処理は各ノード上で動作するデーモンプロセスである gfmd と gfsd

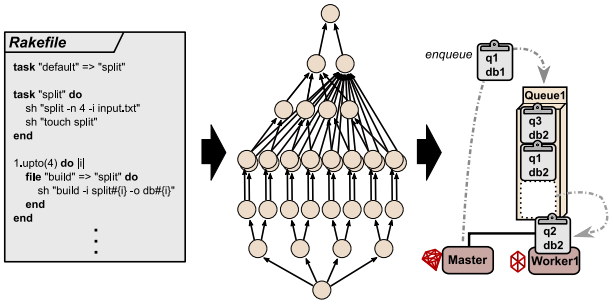

図 4　Pwrake の動作概要
Fig. 4　Overview of operations in Pwrake.

が担う．gfsd によって各サーバのストレージが束ねられ，gfmd がクライアントにファイルアクセスのための名前空間を提供する．実際にクライアントがファイルにアクセスする際は，gfmd からファイルの所在地を聞き，該当する gfsd と直接データのやり取りを行う．

Pwrake では，**図 4** に示すように，マスタープロセスが Rakefile と呼ばれるタスクのワークフローが記述されたファイルを基にタスクの DAG を生成し，ssh 接続によってワーカースレッドが動作する各ノードのタスクキューにそのノードの実行候補タスクをエンキューする．各ワーカプロセスはキューからタスクをデキューしタスクを実行する．

タスクの実行スケジューリングは動的に決定されるため，タスクをできるだけ細かい粒度に分割して Rakefile を記述することで，実行時のロードバランスを高めることができる．なお，Pwrake は ruby で実装されており Rakefile は rake と呼ばれる ruby の内部 DSL を用いて記述する．

また，本研究では相同性検索ツールとして GHOSTZ-GPU を用いる．GHOSTZ-GPU は C++で実装された相同性検索ツールでありクエリと DB を入力としてそれらのファイルに含まれる DNA の塩基配列ごとの比較結果を 1 つのファイルに出力する．またこのツールは，OpenMP を用いたマルチスレッドによる高速化と CUDA による GPU を利用した高速化が施されている．

このシステムの特徴として，Gfarm の機能によりタスクのファイルの出力先にはプロセスが実行されているノードが選択され，かつ Pwrake の I/O 効率の良いスケジューリングにより入力ファイルが存在するノードがタスク実行ノードとして選ばれるため，NVMe SSD 等の高速なローカルストレージを搭載するクラスタのアーキテクチャを最大限活用できる点があげられる．そのため，ファイルアクセスが各ノードに分散していれば，MDS に対するアクセスがボトルネックにならない限り，合計の転送速度はスケールアウトする．また，単一ホスト上で動作する相同性検索ツールをそのまま用いるため，先にあげた既存の様々な相同性検索ツールを容易に分散システム上で実行可能である．加えて，ruby の高い柔軟性によりシステムの入力として複数のクエリファイルと DB ファイルを用いることが

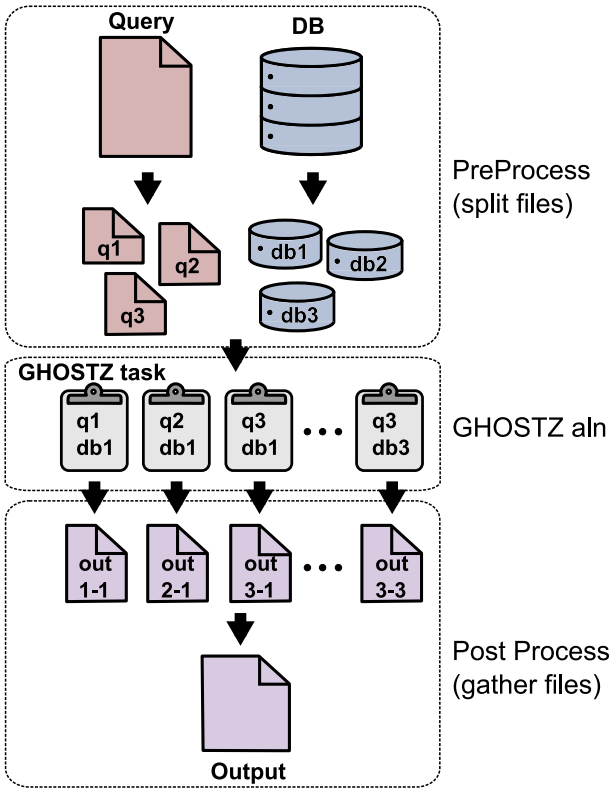

図 5　GHOSTZ PW/GF における相同性検索のワークフロー
Fig. 5　Workflow to do homology search in GHOSTZ PW/GF.

可能となっている．

提案システムでは，ノードレベルおよびノード内プロセスレベルの並列性は Pwrake によって管理される．ノード内の同時実行プロセス数は，プロセスの合計のメモリ使用量がノードの RAM のサイズを超えないように設定する必要がある．メモリ使用量の見積りは，GHOSTZ-GPU を用いて実験的に行った．また，スレッドレベルおよび GPU レベルの並列性は GHOSTZ-GPU によって管理される．なお，CPU や GPU のコア等に対するリソース割当てスケジューリングは OS に依存している．

4.1　ワークフロー

本システムにおける相同性検索のワークフローの概要を**図 5** に示す．このワークフローをタスクの依存関係として定義し，Rakefile に記述することで相同性検索を分散並列実行する．

4.1.1　イニシャライズ

自作のシェルスクリプトを用いて Gfarm の gfmd と gfsd を並列に立ち上げ，Rakefile や Pwrake の設定ファイル等を配置する．この際に Gfarm の設定としてオーバヘッド最小化のために MDS はバックエンド DB を用いず，config ファイルの schedule_idle_load_threash パラメータを 100 にすることでつねに出力ファイルがローカルノードに書き込まれるように設定している．なお，gfarm2fs コマンドによる Gfarm のマウントの際には direct_io パラメータを指定

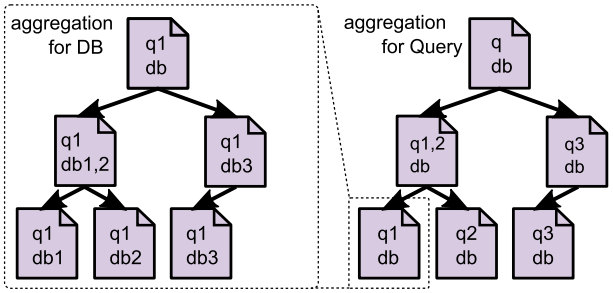

図 6　出力ファイルに対する木構造の依存関係

Fig. 6　The tree structure of tasks to gather multiple outputs into an output.

している．

4.1.2　プリプロセス

システムの入力は，ゲノムの塩基配列が羅列されたFASTA フォーマットのクエリファイルと DB の元となるFASTA ファイルである．相同性検索の前処理としてこれらの入力に関して，ファイルを分割し各ノードにバランス良く分散配置する．この際に用いるファイル分割プログラムは C++で実装されている．詳細は 5.1 節に示す．なお，分割されたファイルのうち DB の元となる FASTA ファイルは GHOSTZ-GPU によって 6 つのファイル群（GHOSTZ の場合 5 つのファイル群）からなる DB にビルドされた後に分散される．また，このステップにおいて，入力ファイルの複製を複数ノードに持たせることで，後の相同性検索タスクにおいて通信の発生を抑え，高速なローカル I/O による read の割合を高めることが可能である．

4.1.3　アライメントプロセス

前処理が完了すると，GHOSTZ-GPU のアライメントを実行するタスクが並列に実行される．合計のタスク数はプリプロセスにおいて分割されたクエリと DB のチャンク数の組合せになる．GHOSTZ-GPU の入力のうち DB ファイルに関しては事前にビルドされている必要があるが，ビルドは前処理タスクにおいて実行されている．このステップにおいて，ノードあたりの実行コア数やファイル複製数，タスクあたりの問題サイズ，1 タスクで用いるスレッド数等，様々なパラメータがシステムの性能に影響を及ぼす．

4.1.4　ポストプロセス

アライメントプロセスで出力された断片的な出力ファイルを，分割前のクエリ単位で集約するタスクが実行される．ファイルの集約タスクの依存関係は，**図 6** のような木構造になるため，各層における依存関係のないタスクを並列に処理することが可能である．また出力ファイルは，システムの入力ファイルサイズに依存して大きくなるため必要に応じて圧縮を行う．

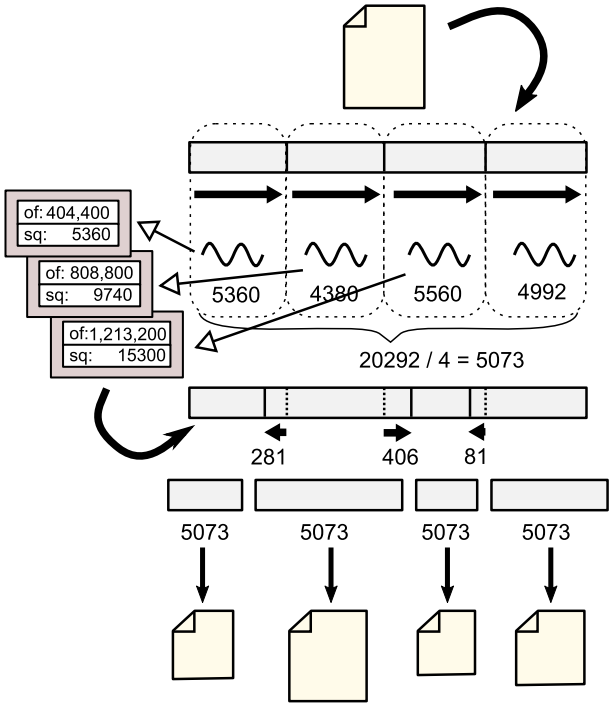

図 7　ファイル分割プログラムの概要

Fig. 7　The algorithm of the program to split input files.

5.　提案システムの最適化

5.1　ファイル分割プログラムの高速化

システムの入力であるクエリと DB は，相同性検索に先立って実行時にメモリ不足にならない大きさに分割する必要がある．ファイルの分割方法として，ファイルサイズによる分割が考えられるが，GHOSTZ の実行時間は入力に用いる FASTA ファイルのサイズよりもそのシーケンス数に強く依存する．ここでシーケンス数とは，FASTA ファイルを構成するひとまとまりの塩基配列であるシーケンスの本数のことである．既存のファイル分割では，分割後の各ファイルシーケンス数が等しくなるようになっているが，シーケンシャルな方法でファイル分割を行っているため処理が低速であった．そこで，本研究ではこのファイル分割プログラムに関して高速化を施した．高速化の概要を**図 7** に示す．

シーケンス数による分割を行うには，1 度全体のシーケンス数をカウントする必要があるが，この部分に関して mmap でファイルをメモリ空間にマップし OpenMP を用いてマルチスレッドでシーケンス数をカウントすることにより処理を高速化した．図 7 は，1 つのファイルをシーケンス数が均等になるように 4 つに分割する例を示しており，まず初めにメモリにマップされたファイル全体を 4 スレッドが分担して各部分のシーケンス数を並列にカウントし，最後に足し合わせて全体のシーケンス数を求める．ここでは，全体のシーケンス数は 20,292 であることが分かるため分割後の各ファイルのシーケンス数は 5,073 となる．ま

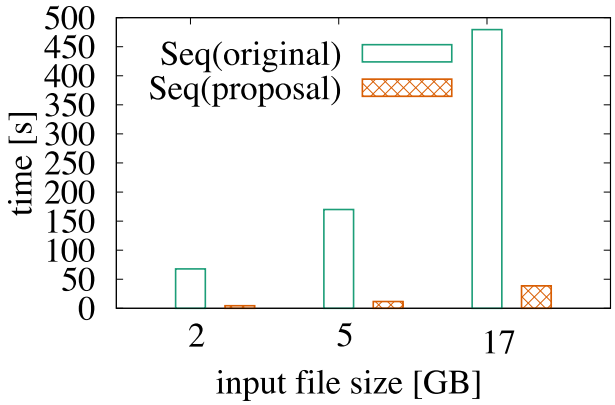

図 8　ファイル分割プログラムの高速化

Fig. 8　An experimental result conducted to evaluate speed up in file split program.

た，ファイル走査の際に，ファイルサイズとシーケンス数の強い相関関係を用いて，ファイルサイズを分割数で割った値に対応するバイトオフセットの位置においてオフセットとシーケンス数の対応関係を保存しておき，それを基に正しい分割点を決定することでファイル分割処理に要する時間を削減している．ファイルサイズはプログラム実行前に既知であり，図の例ではファイルサイズは 1,617,600 B であるため，これを 4 等分した場合に該当するバイトオフセットである 404,400，808,800，1,213,200 の各点を分割候補点とする．シーケンス数のカウントの際のこれらの位置におけるシーケンス数は，それぞれ 5,360，9,740，15,300 であるためこれらの情報を保存しておく．カウント後正しいシーケンス数の分割点は 5,073 であることが分かるため，あらかじめ記録していた情報に基づいて，1 つ目の分割候補点から前方に探索を続けることでシーケンス数が 5,073 となるオフセットの位置が求まる．同様に 2 つ目の候補点から後方，3 つ目の候補点から前方に探索をすることで正しい分割点が求まる．以上で最小限のファイル走査によってシーケンス数によるファイル分割が実現される．高速化を施したプログラムとオリジナルの実装との比較を**図 8** に示す．

結果より，オリジナルの実装よりも 10 倍以上高速であることが分かる．

5.2　GHOSTZ の高速化

提案システムは，ローカルストレージを活かす設計となっているため，ストレージデバイスに NVMe SSD 等の高速な SSD を用いることでファイル I/O のさらなる高速化が期待できる．NVMe SSD は近年注目されているフラッシュメモリベースのストレージであり，PCI Express 接続により高バンド幅・低レイテンシを実現する．本研究では，提案システムのローカルストレージとして NVMe SSD を使用するにあたって GHOSTZ の I/O を解析し，NVMe SSD の性能を発揮できるように GHOSTZ の I/O 処理を高速化した．

表 1　プライベートクラスタ（chris）のノード構成

Table 1　Components of a node on private cluster.

OS	CentOS 6.9
CPU	Intel Xeon E5-2665 × 2
memory	64 GB
Storage	NVMe SSD（Ultrastar SN260）
	Seq Read（Max 6,170 MB/s）
	Seq Write（Max 2,200 MB/s）
Storage	HDD（6 Gbps SAS, 15,000 rpm）RAID0
Network	InfiniBand FDR

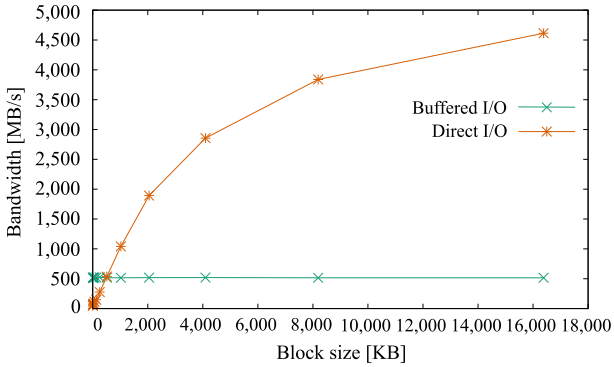

図 9　fio による NVMe SSD のベンチマーク結果

Fig. 9　A result of benchmark to NVMe SSD using fio.

初めに**表 1** に示す NVMe SSD を用いた fio によるベンチマーク結果を**図 9** に示す．なお，結果は 1 プロセスがシーケンシャル読み込みをする場合のブロックサイズによるスループットを表しており，読み込みファイルサイズは 10 GB に，iodepth は 1 に設定している．

この結果から分かるように，NVMe SSD の高いバンド幅を活かすには Direct I/O を用いて比較的大きなブロックサイズでファイル読み込みをする必要がある．

続いて GHOSTZ の I/O の解析では，ソースコードにおける I/O 部分を特定し，strace や HPC 向けの I/O モニタリングツールである DARSHAN [22] を用いた．GHOSTZ の入力として 1 GB のクエリと DB を用いた際の各入力ファイルに対する I/O の統計情報を**表 2** に示す．

表は，DB に関して 5 つの構成ファイルのうち実行時間に影響しうる 4 ファイルのみ記載している．解析の結果，クエリの読み込みに関してファイルサイズに対して 1 回のファイルオープンにかかわらず総バイト数が大きくかつバンド幅が低いことが分かった．この理由として，クエリの読み込みに関しては，C++の std::ifstream.getline を用いて 1 行ずつ読み込みを行っており，かつ getline が 1 回の read 命令に用いるブロックサイズが 8 KB と小さい点があげられる．これに加えて，1 行読み込んだ後にファイルのオフセットを巻き戻す処理により，余分な lseek システムコールが呼ばれていたことも判明した．

そこで，このファイルオフセットの巻き戻し処理をなく

表 2　GHOSTZ の各入力ファイルに対する I/O
Table 2　I/O statics for input files of GHOSTZ.

ファイル名	ファイルサイズ	open 数	総読み込みバイト	総時間	合計バンド幅
DB.seq	509 MB	24	12,197 MB	3.42 s	3,562 MB/s
DB.off	6.2 MB	11	66 MB	0.0096 s	6,880 MB/s
DB.src	2347 MB	23	53,957 MB	19.01 s	2,838 MB/s
DB.nam	485 MB	12	5,816 MB	1.92 s	889.6 MB/s
Query	1,000 MB	1	68,608 MB	78.16 s	877.8 MB/s

表 3　TSUBAME3.0 の構成
Table 3　Components of a node on TSUBAME3.0.

OS	SUSE Linux Enterprise Server 12 SP2
CPU	Intel Xeon E5-2680 V4（14 cores）× 2
GPU	NVIDIA Tesla P100 × 4
memory	256 GiB
Storage	NVMe SSD（Intel DC P3500 2 TB）
Network	Intel Omni-Path 100 Gb/s × 4

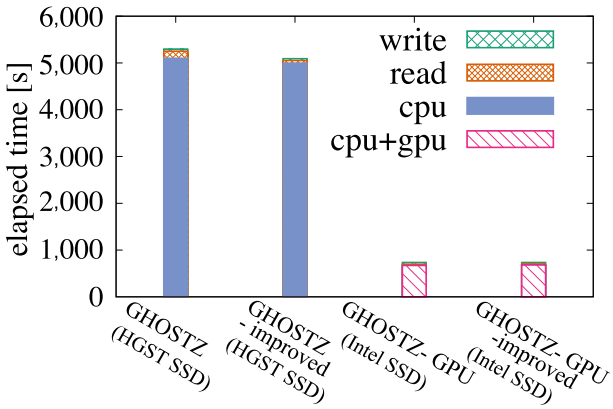

図 10　GHOSTZ の実行時間
Fig. 10　Total elapsed time to do homology search in GHOSTZ.

し，また，NVMe SSD の高バンド幅を活かすために read 時のブロックサイズを変更可能な 1 行読み込み関数を実装することにより，クエリの読み込みに関して高速化を施した．関数内でファイル走査の際に次のシーケンス部のヘッダをあらかじめ検出しておき，次の関数呼び出し時にその場所から処理が再開するようにすることで，ファイルオフセットの巻き戻しによるオーバヘッドをなくしている．なお，事前に行った NVMe SSD のベンチマーク結果を基に，ファイルの読み込みには Direct I/O を用いた．

オリジナルの GHOSTZ の実装と高速化を施したものとの比較を**図 10** に示す．また，図 10 における，I/O 時間のみを抜き出したものを**図 11** に示す．入力ファイルサイズはクエリと DB ともに 1 GB であり，I/O 時間の取得には，DARSHAN を利用した．GHOSTZ の比較は表 1 と**表 3** に示す環境で実施され，NVMe SSD 上のファイルに対して前者の環境では GHOSTZ が，後者の環境では GHOSTZ-GPU が実行される．

結果より，CPU 時間と I/O 時間を含めた全体の実行時

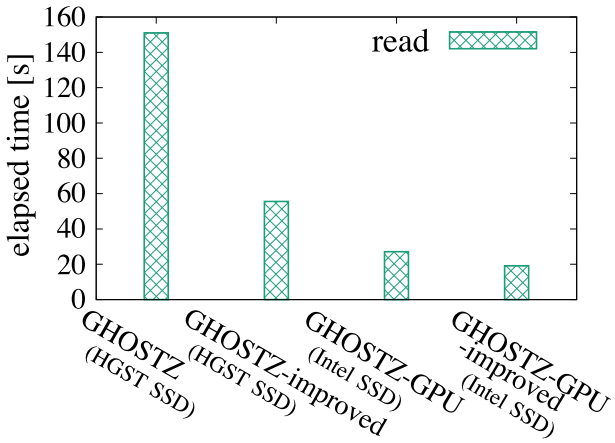

図 11　GHOSTZ の I/O 時間
Fig. 11　Total I/O time to do homology search in GHOSTZ.

間の減少割合は小さいが，クエリの読み込み時間の削減により合計のファイル読み込み時間が減少していることが見て取れる．

6. 評価

6.1 関連研究との比較

本節では，実装した提案システムである GHOSTZ PW/GF と関連システムである GHOST-MP，HAMOND，SparkBLAST の 3 つを比較した結果について述べる．

本実験では，表 1 に示す計算ノードで構成されるクラスタ上にて，全体の実行時間に関するスケーラビリティを比較する目的で実験を行った．

GHOST-MP との比較においては，7 ノードを用いてノードあたりプロセス数を 1 に設定し，2 GB のクエリと 5 GB の DB を用いて実験を行った．実験結果を**図 12** に示す．なお，1 ノードから 4 ノードを用いた実験では GHOST-MP に関してはメモリ不足によるエラーにより実行できなかった．

結果より，提案システムはノード数に対して性能がスケールアウトし，7 ノードを用いた場合に GHOST-MP の約 4 倍高速であることが分かる．この高速化は，GHOST-MP が相同性検索の際に用いている GHOSTX と提案システムにおいて使用している GHOSTZ のアルゴリズムの違いによるものである．なお，ノード数 5 からノード数 7 にかけては GHOST-MP がより高い並列化効率を示していること

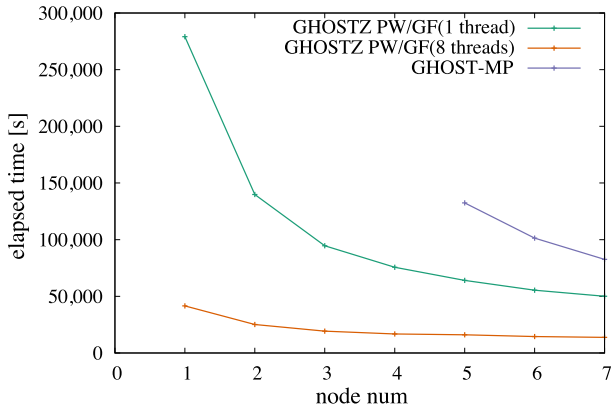

図 12　GHOST-MP との比較結果
Fig. 12　Comparison result with GHOST-MP.

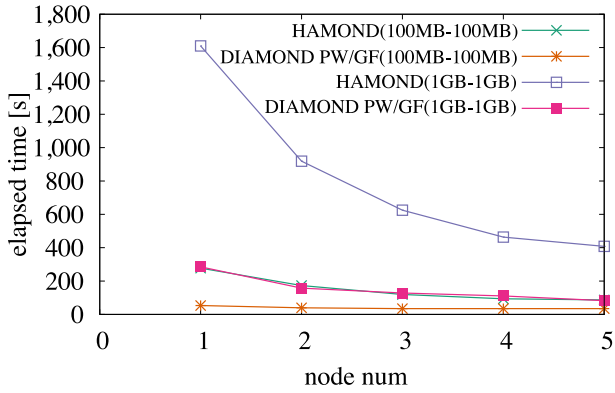

図 13　HAMOND との比較結果
Fig. 13　Comparison result with HAMOND.

が分かるが，MPI による実装により GHOSTX の実行に対して細かい性能チューニングが成されていることによる．

次に HAMOND との比較結果に関して述べる．この実験では，5 ノードを用いて Hadoop クラスタを構成し，Hadoop と Pwrake/Gfarm による実行基盤に関して可能な限り同条件下で比較を行うようパラメータを調整した．Hadoop の実行にあたってはファイルシステムに HDFS を，リソースマネージャには YARN を用いている．なお，Hadoop のバーションは 2.7.7，DIAMOND のバージョンは 0.9.14 であり，クラスタのチューニングは CloudEra の記事を参考に行った [23]．

DB とクエリのそれぞれに対して 100 MB のものと 1 GB のものを用いた実験結果を**図 13** に示す．なお，HAMOND との比較にあたっては，提案システムにおける相同性検索ツールに DIAMOND を用いることで相同性検索のアルゴリズムの違いによる実行時間への影響を取り除いている．以下では DIAMOND を用いた提案システムの実装を DIAMOND PW/GF と記す．

実験では，HAMOND と DIAMOND PW/GF 両者において 1 つの DIAMOND 実行タスクが処理の際に 3 スレッドを用いるようにしている．また，HAMOND の実装において入力ファイルの分割サイズは 2 MB となっているため，

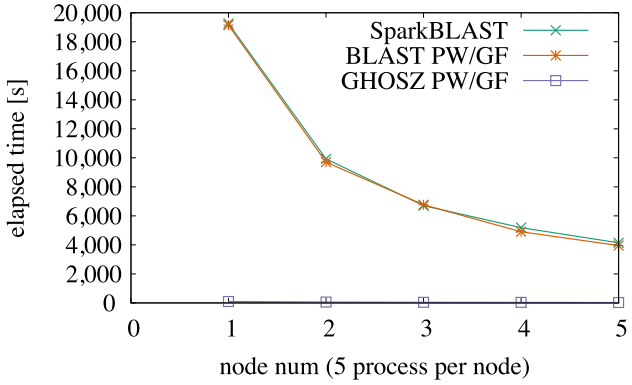

図 14　SparkBLAST との比較結果
Fig. 14　Comparison result with SparkBLAST.

今回の実験では Mapper の数が 50 になっている．提案手法においてはクエリの分割の際のパラメータをこれらの値に設定し，また，ノードあたりの実行コア数を HAMOND において 1 ノードで実行される Mapper 数である 1 に合わせた．結果より，100 MB のクエリと DB を用いた実験では，5 ノードにおいて約 2.47 倍の高速化，1 GB を用いた実験では約 4.96 倍の高速化を達成した．この高速化の理由として，Hadoop では暗黙的に行われる shuffle 処理やディスク書き込み等といったタスクの実行に対するオーバヘッドが存在するため，これが全体の性能に影響を及ぼしていると考えられる．結論として，HAMOND には Hadoop クラスタのさらなるチューニングや適切な Mapper 数の割当て等による性能向上の余地があるものの，結果は提案手法が HAMOND よりも高速に相同性検索を実行するということを示している．

最後に SparkBLAST との比較結果を**図 14** に示す．この実験では HAMOND との比較と同様に提案システムにおいて相同性検索ツールに BLAST を用いたもの（BLAST PW/GF）と GHOSTZ を用いたもの（GHOSTZ PW/GF）の両方を用いている．また，クエリの分割数を 50，DB の分割数を 1 に設定し，Spark の 1 ノードあたり Executor 数が 5 であるため，Pwrake におけるノードあたり実行コア数も 5 に設定している．

SparkBLAST と BLAST PW/GF の比較結果より，提案システムはすべてのノード数において SparkBLAST 以下の実行時間であるがほとんど差がないということが分かった．しかし，GHOSTZ PW/GF と SparkBLAST との比較に関しては，相同性検索ツールのアルゴリズムの違いが大きく影響し，提案システムは 5 ノードを用いた際に約 155 倍高速に動作した．なお，評価に際して，SparkBLAST では結果ファイルの集約処理は実装されていないため，提案手法におけるポストプロセスは除外している．BLAST PW/GF と SparkBLAST の性能が同等なものになった理由としては，Spark において MapReduce の Map にあたる処理のみが存在するためシャッフルが発生せず，また，イ

ンメモリで動作するため，余分なオーバヘッドが少ないからであると考えられる．また，SparkBLAST においてBLAST の処理はシェルコマンドとして各ノード上で実行するような実装であり，これは提案手法と同様の手法であるため，BLAST の実行においても両者における差異は少ない．今後の課題として，チャンクファイル数や用いるデータセットのサイズを大きくした場合に関して，結果がどのように変化するか確認したい．

6.2　大規模実行環境における評価

本節では，提案システムに関して東京工業大学のスーパーコンピュータである TSUBAME3.0 を用いて実施した性能評価の結果を述べる．

TSUBAME3.0 の計算ノードの概要は表 3 のとおりである．全 540 ノードで構成され各ノードは Intel Omni-Path によって内部接続されている．また，ジョブスケジューリングシステムには UNIVA Grid Engine（UGE）が採用されている．提案システムは，GPU を用いる相同性検索ツールを用いており，またクラスタの各ノードのローカルストレージを束ねて合計の I/O 性能をスケールアウトさせる狙いがあるため，各ノードに NVMe SSD と GPU を搭載している TSUBAME3.0 における性能評価は提案システムの性能を評価するのに適している環境であるといえる．

6.2.1　大規模データに対する評価

大規模データにおける性能評価では，システムの入力として DB の元ファイルに約 62 GB のものを，クエリに 70 GB と 250 GB のデータを用いて実験を行った．ここで，クエリに 250 GB のものを用いた実験では，Gfarm の使用するファイルディスクリプタ数が UGE で設定されている上限値に達してしまう問題により，ポストプロセスにおいてエラーが発生したため，アライメントプロセスにおける実行結果のみ示す．なお，クエリに用いた FASTA ファイルの合計のシーケンス数は 587,335,484，DB に用いた FASTA ファイルのシーケンス数は 106,867,961 である．DB ファイルとして用いているデータは，National Center for Biotechnology Information の提供する nr（non-redudant protein）であり，これは複数のデータベースに収録されているアミノ酸配列のコレクションである．また，クエリには Human Microbiome Project によって解析されたデータから得られたヒト口腔内メタゲノムデータ（Supragingival Plaque）を用いている．

本実験では，180 ノードのうち 1 ノードで Gfarm の MDS と Pwrake のマスターを動作させ，残りの 179 ノードを Gfarm の FSN と Pwrake のワーカとして動作させる．

実験の流れは以下のとおりである．

(1) bzip2 で圧縮されているクエリファイルを解凍し，解凍後の FASTQ ファイルを FASTA ファイルに変換する．続いて，ファイル分割プログラムを用いて各クエリファイルを 1 ファイルあたり 850,000 シーケンス（約 100 MB）になるように分割し，DB ファイルは 18 ファイル（1 ファイルあたり約 3 GB）に分割する．分割後の各 DB チャンクに関しては GHOSTZ-GPU よってそれぞれ 6 つのファイル群からなる DB にビルドされる．

(2) 179 ノードの FSN にクエリチャンクおよびビルド済みの DB チャンクをバランス良く配置（順番に割り当て）し，DB に関しては 18 ファイルそれぞれに対して複製を 10 ずつ作成することで全 FSN のローカルストレージに DB の実体が存在する状態にする．これにより，アライメントプロセスにおける各タスクの DB の読み込みはつねにローカル I/O になる．

(3) GHOSTZ-GPU によるアライメントタスクがクエリチャンクと DB チャンクの組合せの数だけ並列に実行される．

(4) 各 GHOSTZ-GPU の出力結果ファイルを，ファイル分割前のクエリ単位で集約する．

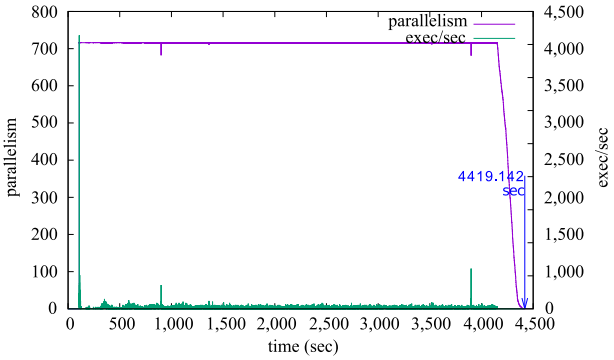

図 15　アライメントプロセスにおけるプロセスレベルの並列度（クエリ：70 GB

Fig. 15　Parallelism at process level in alignment process using 70 GB of queries.

6.2.1.1　アライメントプロセス

70 GB のクエリを用いた実験結果として，各ノードにおけるタスク（プロセス）レベルにおける並列度の推移を**図 15** に，スレッドレベルにおける並列度の推移を**図 16** に示す．また，250 GB のクエリを用いた実験結果として，タスクレベル並列度の推移を**図 17** に示す．

結果より，コアの利用率およびノード間の負荷分散が高いことが確認できる．なお，アライメントプロセスにおいては，1 ノードあたりのタスク実行プロセス数は 4 であり 1 プロセスあたり 14 スレッドを用いて相同性検索を実行している．このコア数は 180 ノードを用いたシステムの最適なパラメータを求める事前実験において各タスクのメモリ使用量を見積もったうえで実験的に導出した最適なコア数である．並列度はプロセスレベルで最大 716 であり，スレッドレベルでは最大 10,024（平均 9,380）並列で相同性検索が実行されていることが分かる．また，全体の実行時

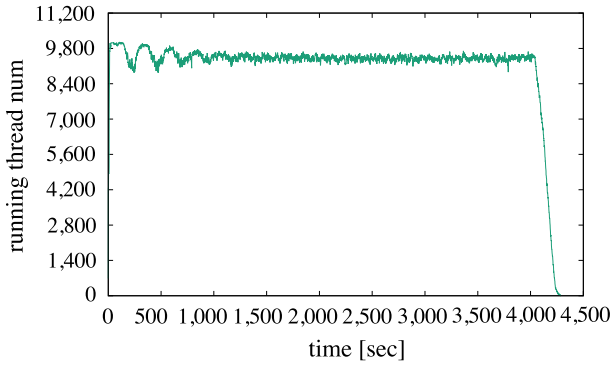

図 16　アライメントプロセスにおけるスレッドレベルの並列度
Fig. 16　Parallelism at thread level in alignment process using 70 GB of queries.

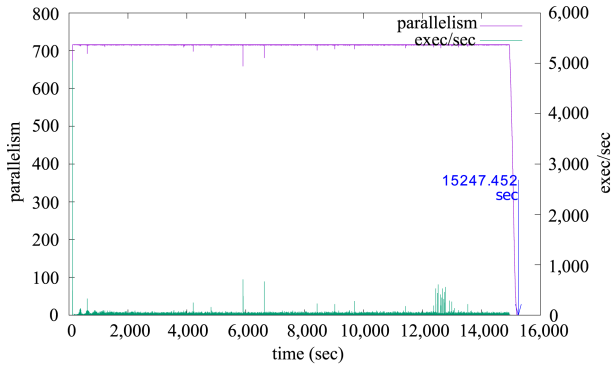

図 17　アライメントプロセスにおけるプロセスレベルの並列度（クエリ：250 GB
Fig. 17　Parallelism at process level in alignment process using 250 GB of queries.

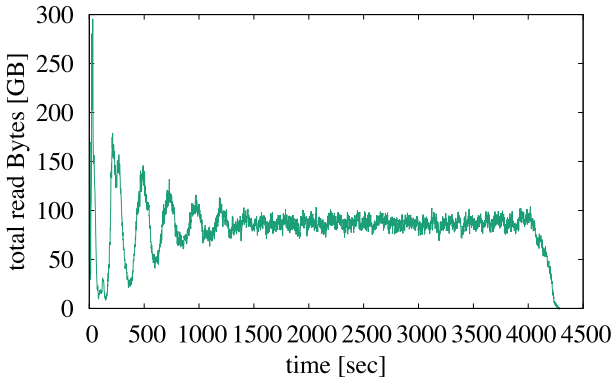

図 18　アライメントプロセスにおける合計読み込みバイト数
Fig. 18　Total reading bytes in alignment process.

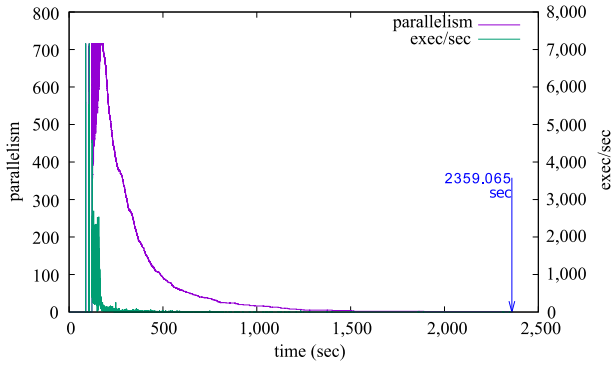

図 19　ポストプロセスにおけるプロセスレベルの並列度
Fig. 19　Parallelism at process level in post process using 70 GB queries.

間は 4,419.14 秒であり，合計タスク数は 12,690，1 プロセスあたりの平均実行時間は 233 秒であった．なお，1 プロセスあたりの最大実行時間は 417 秒，最小実行時間は 47 秒，標準偏差は 37 であった．1 プロセスあたりの平均実行時間は，18 ノードで行った事前実験における実行時間とほぼ同じであり，これは，リモートノードへの I/O によるネットワークの輻輳が発生せずにローカル I/O を活かすというシステムの特徴を発揮できていることを示している．実際に，各タスクにおいて DB に関してローカル読み込みが行われていることを実行ログより確認している．

最後に，アライメントプロセスにおけるシステム全体のファイル読み込みバンド幅の推移を**図 18** に示す．なお，バンド幅の導出は GHOSTZ-GPU のソースコード内のすべてのファイル読み込み関数の前後に gettimeofday() を利用したタイマーを挿入し，読み込みおよび書き込みバイト数を経過時間で割った値を，プロセスの経過時間を横軸とした積み上げグラフの形で全タスク分足し合わせることで求めている．

6.2.1.2　ポストプロセス

次に，アライメントプロセスにおいて出力された結果ファイルをクエリごとに集約するタスクの実験結果について述べる．クエリに 70 GB のものを用いた実験おけるプロセスレベルの並列度を**図 19** に示す．

ファイル集約タスクは図 6 に示したとおり，木構造の形のタスク依存関係になっているため並列度はタスク開始時点で最も高く時間が進むに連れて減少している．

6.2.2　相同性検索結果ファイルの比較

相同性検索における重要な性能指標として処理速度と出力の正確性の 2 つがあげられるが，前者に関しては前節において確認することができた．後者に関して，提案システムでは DB を分割して相同性検索を実行可能であるが，GHOSTZ は DB をクラスタリングする特徴があるため，DB ファイルを分割した場合としない場合でクラスタの代表点が異なり，出力結果も異なってしまう可能性がある．

本項では，この DB 分割による出力への影響について調査した結果を示す．提案システムは GHOSTZ を分散並列実行に拡張したものであるため，提案システムにおいて出力結果が完全に一致し実行時間が短縮されるのが理想的である．そこで，GHOSTZ 単体で実行した場合の結果が提案システムの出力結果にどれほど含まれるかを提案システムにおける GHOSTZ の結果に対する類似度として定義しそれを評価する実験を行った．なお，GHOSTZ の実行結果は**図 20** に示すような内容を含む．出力は，図の上部に

表 4　1 GB の DB を用いた実験の出力ファイルに関する比較結果
Table 4　The results of an experiment to evaluate accuracy of outputs using 1 GB of DB.

DB の分割数	出力サイズ	オリジナルの結果に対する類似度
1	680.4 MB	99.999%
2	1.1 GB	96.931%
3	1.4 GB	96.876%
4	1.8 GB	97.012%
5	2.4 GB	97.218%
6	2.5 GB	97.355%
7	3.1 GB	97.546%
8	3.4 GB	97.594%
9	3.7 GB	97.770%
10	4.2 GB	97.851%

示されるように結果のレコードが羅列された CSV 形式のファイルになっており，各カラムの説明は図中の下部に示したとおりである．特に重要なパラメータとして，1，2，3，11 番目のものがあげられ，それぞれ対象クエリ名，DB 中の対象プロテイン名，スコア（類似度），エラー値を表す．

初めに，テストデータとして 100 MB のクエリと DB ファイルに対して，提案システムにおける DB の分割数を 1 から 10 に変化させて実行した出力結果と，単一ホストで動作する GHOSTZ を用いて DB を分割せずに実行した結果との比較を行った．なお，比較にあたって，ある結果 A のレコードのなかでもう一方の結果 B にも現れているレコードの B の全体のレコードに対する割合を 2 つの結果の類似度として求めるプログラムを Spark で実装し，それを評価に用いた．今回の場合，このファイル A は DB 分割を用いる提案システムの結果を指し，ファイル B は非分割の GHOSTZ による結果を指す．比較結果を**表 4** に示す．表中のサイズはファイル A のサイズを表している．なお，非分割による結果であるファイル B のサイズは 680.4 MB であった．

結果より，分割数を 1 にした場合は単一の GHOSTZ によるものと同じ結果が現れているが，DB の分割数を増やすに連れて，出力ファイルサイズが大きくなっていることが分かる．出力ファイルサイズの肥大化の原因としては，今回の実験で GHOSTZ のパラメータの 1 つであるクエリごとの検索結果の最大数を，提案システムにおける GHOSTZ と単一の GHOSTZ で同じにしているため，提案システムが最大で DB の分割数倍の大きさのファイルを出力し得ることによる．類似度は DB の分割によって少し低下していることが分かるが，分割数にはあまり依存していない．類似度の低下に関しては，DB の分割によってクラスタが変化することで，それに起因して本来結果として現れて欲しいレコードが現れていないと考えられる．

また，6.2.1 項における実験の一部のクエリに対する結果に関して，同様の比較を行った結果を**表 5** に示す．

表 5　62 GB の DB を用いた実験の比較結果
Table 5　The results in an experiment to evaluate accuracy of outputs using 62 GB of DB.

DB の分割数	出力サイズ	オリジナルの結果に対する類似度
18	28.9 GB	66.71%

表 6　DB のサイズに対する DB 分割の影響に関する実験結果
Table 6　Experimental results about the impact for DB partitioning by DB size.

DB のサイズ	出力サイズ	オリジナルの結果に対する類似度
5 GB	56.8 GB	97.927%
10 GB	44.6 GB	98.365%
20 GB	35.2 GB	91.885%
30 GB	30.1 GB	79.63%
40 GB	26.4 GB	63.352%

結果より，類似度が比較的低いことが分かるが，これは**表 6** から分かるように DB のサイズが大きくなるほど，分割による影響が大きいためであると考えられる．なお，表 6 における実験では，100 MB のクエリを用いて，DB の分割数を 18 に固定して実験を行っている．また，表 6 より，DB のサイズが大きくなるほど出力ファイルサイズが小さくなっているが，これは DB のチャンクサイズが大きくなることでクラスタリングによる効率化の影響も大きくなり，より重要な結果のみが出力に現れているためである．

以上の結果より，GHOSTZ を用いて提案システムにおいて DB を分割して実行した結果は，オリジナルの GHOSTZ の結果と一致せず，その入力 DB のサイズが増大するほど分割時の結果への影響が増大することが分かった．しかし，相同性検索結果としての正確性という意味ではオリジナルの GHOSTZ の結果も提案システムの結果もどちらも生物学的に正しいといえる結果のサブセットであるといえるため，どちらが真に正しいデータが多く含まれているかは別の指標で評価する必要がある．GHOSTZ の正確性に関する性能は，ある程度小さいエラー値の設定の場合に BLAT や BLASTX，RAPSearch 等のツールと比較しても高い性能であると結論付けられている [11]．また，実際の相同性検索の際にはこのエラー値を 1.0E-8 や 1.0E-5 に指定するのが良いことが知られている [24], [25]．

そこで，図 20 に示すレコードのうち，実際の解析で重要となる E-value および Identity のパラメータに関してそれぞれ 1.0E-8 以下および 95 以上となるようなデータが，オリジナルの GHOSTZ および提案システムの結果にどれほど含まれるか調査した．結果を**表 7** に示す．なお，実験において用いたクエリと DB のサイズは 1 GB である．

表中の値は，両者で重複せずにどちらか一方の結果にのみ出現したレコードの，ぞれぞれの全レコード数に対する割合を示している．結果より，信頼度が高く重要と考えられるデータが，オリジナルの GHOSTZ の結果および提案

```
hsa:124045...  hsa:12405...  100  139  0  0  1  139  1  139  2.04391e-76  283.878
hsa:124045...  ptr:454320...  99.2126  127  1  0  13  139  14  140  5.96068e-68  255.758
hsa:124045...  mcc:71436...  88.9764  127  14  0  13  139  14  140  5.05773e-59  226.098
hsa:124045...  rno:29208...  58.6777  121  46  2  13  133  14  130  1.38697e-3a2  138.272
hsa:124045...  mmu:3269...  55.9055  127  50  3  13  139  12  132  1.17414e-31  135.191
```

1. Name of a query sequence
2. Name of a homologue sequence (subject)
3. Sequence Identity
4. Alignment lengtha
5. The number of mismatches in the alignment
6. The number of gap openingsin the alignemt
7. Start position of the query in the alignment
8. End position of the query in the alignemnt
9. Start position of the subject in the alignment
10. End position of the subject in the alignment
11. E-value
12. Normalized score

図 20　GHOSTZ の出力結果の例
Fig. 20　An example of outputs in GHOSTZ.

表 7　実際の解析を想定した場合に重要と見なされるデータの割合
Table 7　Percentage of important data assuming actual genomic analysis.

分割数	GHOSTZ に固有なもの	提案システムに固有なもの
2	13.9%	13.6%
3	19.2%	22.9%
4	34.9%	28.1%
5	18.2%	32.0%

表 8　1 GB の DB を用いた場合の GHOSTX との出力結果に関する比較結果
Table 8　The results of an experiment to evaluate accuracy of outputs using 1 GB of DB in the proposed system using GHOSTX.

DB の分割数	オリジナルの結果に対する類似度
2	99.40%
3	99.57%
4	99.64%
5	99.66%
6	99.68%
7	99.70%
8	99.72%

システムの結果どちらか一方にのみ独立して存在する割合は両者でほとんど変わらないことが分かる．これは，どちらの結果にも重要と考えられる情報が同程度含まれていることを示している．

結論として，提案システムにおいて相同性検索ツールとして GHOSTZ を用いて DB を分割した場合，その結果はオリジナルの GHOSTZ による結果と一致しない点を考慮する必要があるが，どちらがより正確な結果を含むかはエラー値やスコア等を基に生物学的に調査する必要がある．

最後に，DB のクラスタリングを行わない GHOSTZ に似たツールである GHOSTX を提案システムに適用させ，表 4 と同様の実験を行った結果を**表 8** に示す．

結果より，どの分割数に対しても類似度は高く保たれていることが分かる．表には記載されていないが，DB に 40 GB のものとクエリに 500 MB のものを用いて DB を 18 分割した場合にも類似度は 99.59%と高い数値であったことを確認している．

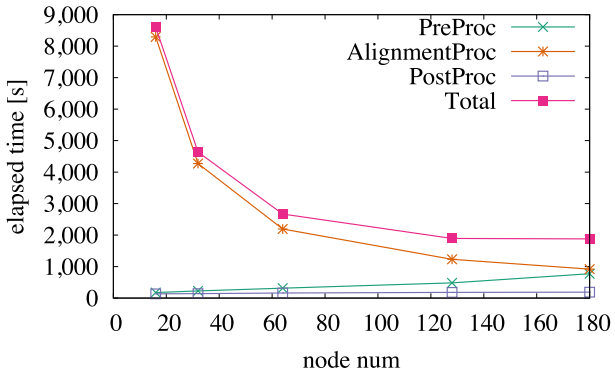

図 21　提案システムの実行時間に関するスケーラビリティ
Fig. 21　Scalability about execution time in GHOSTZ PW/GF.

結論として，相同性検索ツールとして GHOSTX を用いた場合には，提案システムの出力は DB の分割にかかわらずほぼすべてのオリジナルの結果を含み，この結果は DB をクラスタリングするようなアルゴリズムを採用しない相同性検索ツールを用いる場合に，オリジナルの実行結果に対する DB の分割によるデータの欠損がほとんどないことを示す．

6.2.3　スケーラビリティに関する実験

スケーラビリティに関する実験では，DB の元ファイルに 3 GB のものを 1 つ，クエリに 2,148 個のファイルから成る合計 21 GB のクエリ（179 ノード× 4 プロセスで 3 ステップ分）を用いて，180，128，64，32，16 ノードにおいてストロングスケーリングに関する評価を行った．なお，DB に関しては各使用ノードに対してできる限り条件を合わせるために，全ノードに複製を持たせている．実験結果を**図 21** に示す．

結果より，アライメントプロセスに関してノード数が増えるに従って性能がスケールアウトしていることが分かる．また，プリプロセスに関しては FSN どうしの通信が発生するためノード数が増えるに従い実行時間が増加しているが，ポストプロセスに関しては今回の実験では DB を分割しておらずアライメントプロセスにおけるタスク数が比

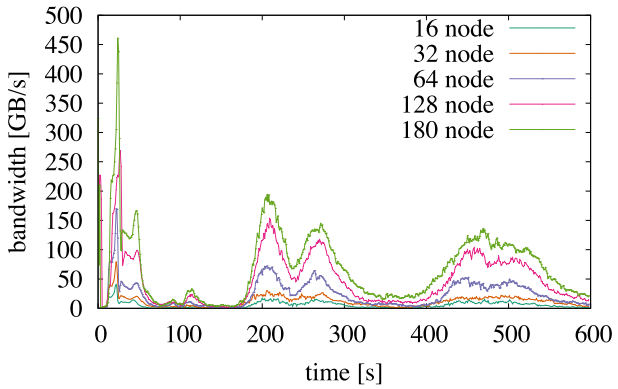

図 22　アライメントプロセスにおける読み込みバンド幅に関するスケーラビリティ

Fig. 22　Scalability about reading bandwidth in alignment process.

較的少ないためノード数の増加にかかわらずほぼ一定である．全体として高いスケーラビリティを示しているが，128 ノードから 180 ノードにかけては全体の性能がスケールアウトしていないように見える．これに対して，入力ファイルに対してクエリと DB のチャンクサイズをより増大させることで，アライメントプロセスにおける 1 タスクあたりの実行時間が増加して全体に占める比重が高まり，かつプリプロセスおよびポストプロセスにおけるタスク数が減少するため，システム全体のスケーラビリティをさらに高めることが可能であると考える．

最後に図 21 のアライメントプロセスにおけるファイル読み込みの合計のバンド幅の推移を**図 22** に示す．なお，図 22 は，全体で 9,000 秒あるアライメントプロセスのうち最初の 600 秒までを抜粋したものである．

グラフより，最大の合計バンド幅は約 450 GB/s に達しており，時間が経過するに連れて各ノードにおける I/O の実行タイミングの差が広がっていきグラフ上の山がなだらかなものになっていくのが見て取れる．また，ノード数の増加に対してバンド幅がスケールアウトしているがこれは図 21 におけるアライメントプロセスの高いスケーラビリティの要因となっている．

7.　まとめと今後の課題

本研究では，増大するメタゲノムデータに対する分散並列相同性検索システムを開発し，相同性検索の高速化とメモリ不足問題の解決を達成した．

相同性検索の 2 つの入力である DB とクエリの両方を実行時にメモリ不足にならない大きさのチャンクに分割およびそれらの全組合せの相同性検索を独立したタスクとして並列実行し，最後に各出力ファイルを集約するという提案手法を Gfarm，Pwrake，GHOSTZ-GPU を用いて実装した．

結果として，提案システムは MPI や Hadoop，Spark といった分散処理フレームを用いた関連システムに対して同等以上の性能を示し，NVMe SSD や GPU を搭載した大規模環境におけるストロングスケーリングに関する実験では高いスケーラビリティを示した．また，TSUBAME3.0 を 180 ノードを用いた実験では，既存の手法では問題となってしまう大規模なメタゲノムデータ（約 1 億シーケンスで構成される nr DB と約 5 億シーケンスから構成されるデンタルプラークゲノムデータ）に対しておよそ 2 時間で相同性検索を実行した．

今後の課題として，今回の実験ではノードあたりのプロセス数やプロセスあたりのスレッド数および GPU の数等のパラメータ最適化を実験的に行っているが，これらのパラメータを基に実行時間をモデル化し最適化を行うことでさらなるシステムの性能改善が期待できるため検討を行う．さらに，相同性検索の出力結果に関して，ゲノム解析で利用しやすいような形の検討やその結果を基にした可視化プロセスの追加等といった，相同性検索以外のゲノム解析タスクの組み込みも検討したい．

謝辞　本研究の一部は，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（Cygnus），国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）および富士通研究所との共同研究の助成を受けたものです．

なお，TSUBAME3.0 の利用に関しては，TSUBAME グランドチャレンジ大規模計算制度実行委員会の主催する TSUBAME グランドチャンレンジ制度を用いました．

参考文献

[1]　イルミナ社：ヒトの健康における細菌およびメタゲノム（オンライン），入手先 ⟨https://jp.illumina.com/content/dam/illumina-marketing/apac/japan/documents/pdf/publicaton_metagenome_for-human-health-j.pdf⟩（参照 2018）．

[2]　山田拓司：ヒト腸内メタゲノム解析が広げる医療展開（オンライン），入手先 ⟨https://www.jstage.jst.go.jp/article/kagakutoseibutsu51/12/51_802/_pdf/-char//ja⟩（参照 2018）．

[3]　Taroncher-Oldenburg, G. et al.: Translating microbiome futures, *Nature Biotechnology*, Vol.36, pp.1037–1042 (2018).

[4]　Anon: Metagenomics Market Size, Share, Industry Trends Report, 2018–2025 (online), available from ⟨https://www.grandviewresearch.com/industry-analysis/metagenomics-market⟩ (accessed 2018).

[5]　Kanehisa, M. and Goto, S.: KEGG: Kyoto Encyclopedia of Genes and Genomes, *Nucleic Acids Research*, Vol.28, No.1, pp.27–30 (2000).

[6]　Kanehisa, M. et al.: KEGG: Kyoto Encyclopedia of Genes and Genomes, *Nucleic Acids Research*, Vol.28, No.1, pp.27–30 (2000).

[7]　Tatusov, R.L. et al.: The COG database: An updated version includes eukaryotes, *BMC Bioinformatics*, Vol.4, No.1, p.41 (2003).

[8]　Davis-Dusenbery, B.: Precision Medicine Research in the Million-Genome Era, *Genetic DEngineering & Biotechnology News*, Vol.37, No.2, pp.26–27 (2017).

[9] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J.: Precision Medicine Research in the Million-Genome Era, *J. Mol. Biol.*, Vol.215, No.3, pp.403–410 (1990).
[10] Suzuki, S., Kakuta, M., Ishida, T. and Akiyama, Y.: GHOSTX: An Improved Sequence Homology Search Algorithm Using a Query Suffix Array and a Database Suffix Array, *PLoS One*, Vol.9, No.8, pp.1–8 (2014).
[11] Suzuki, S., Kakuta, M., Ishida, T. and Akiyama, Y.: Faster sequence homology searches by clustering subsequences, *Bioinformatics*, Vol.31, No.8, pp.1183–1190 (2014).
[12] Buchfink, B., Xie, C. and Huson, D.H.: Fast and Sensitive Protein Alignment using DIAMOND, *Nature Methods*, Vol.12, No.1, pp.59–60 (2015).
[13] 生命情報・DDBJ センター：DDBJ リリース統計（オンライン），入手先 ⟨https://www.ddbj.nig.ac.jp/stats/release.html#total_data⟩（参照 2019）.
[14] Machida, K. and Tatebe, O.: GHOSTZ PW/GF: Distributed Parallel Homology Search System for Large-scale Metagenomic Analysis, *Proc. 3rd IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications* (*BPOD 2019*), pp.3492–3700 (2019).
[15] Guo, R., Zhao, Y., Zou, Q., Fang, X. and Peng, S.: Bioinformatics applications on Apache Spark, *Gigascience*, Vol.7, No.8, giy098 (2018).
[16] Darling, A., Carey, L. and Feng, W.: The design, implementation, and evaluation of mpiBLAST, *ClusterWorld Conference and Expo*, LA-UR 03-2862 (2003).
[17] Kakuta, M., Suzuki, S., Izawa, K., ishida, T. and Akiyama, Y.: A Massively Parallel Sequence Similarity Search for Metagenomic Sequencing Data, *International Journal of Molecular Sciences*, Vol.18, No.10, pp.1–11 (2017).
[18] de Castro, M.R., dos Santos Tostes, C., et al.: SparkBLAST: scalable BLAST processing using in-memory operations, *BMC Bioinformatics*, Vol.18, No.318, pp.1–13 (2017).
[19] Yua, J., Blomb, J., Sczyrbac, A. and Goesmann, A.: Rapid protein alignment in the cloud: HAMOND combines fast DIAMOND alignments with Hadoop parallelism, *Journal of Biotechnology*, Vol.257, No.318, pp.1–13 (2017).
[20] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing, Ohmsha, Ltd. and Springer*, Vol.28, No.3, pp.257–275 (2010).
[21] Tanaka, M. and Tatebe, O.: Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing, *Proc. ACM International Symposium on High Performance Distributed Computing* (*HPDC*), pp.356–359 (2010).
[22] Carns, P., Harms, K., Allcock, W., Bacon, C., Lang, S., Latham, R. and Ross, R.: Understanding and improving computational science storage access through continuous characterization, *ACM Trans. Storage*, Vol.7, No.3, pp.8:1–8:26 (2011).
[23] Anon: Tuning YARN (online), available from ⟨https://www.cloudera.com/documentation/enterprise/latest/topics/cdh_ig_yarn_tuning.html⟩ (accessed 2018).
[24] Turnbaugh, P.J., Ley, R.E., Mahowald, M.A., Magrini, V., Mardis, E.R. and Gordon, J.I.: An obesity-associated gut microbiome with increased capacity for energy harvest, *Nature*, Vol.444, pp.1027–1031 (2006).
[25] Kurokawa, K., Itoh, T., Kuwahara, T., Oshima, K., Toh, H., Toyoda, A., Takami, H., et al.: An obesity-associated gut microbiome with increased capacity for energy harvest, *Nature*, Vol.444, pp.1027–1031 (2007).

町田 健太

平成 7 年生．平成 30 年筑波大学情報学群情報科学類卒業．令和 2 年同大学大学院システム情報工学研究科コンピュータサイエンス専攻修士課程修了．

建部 修見　（正会員）

昭和 44 年生．平成 4 年東京大学理学部情報科学科卒業．平成 9 年同大学大学院理学系研究科情報科学専攻博士課程修了．同年電子技術総合研究所入所．平成 17 年独立行政法人産業技術総合研究所主任研究員．平成 18 年筑波大学大学院システム情報工学研究科准教授．平成 27 年同教授．博士（理学）（東京大学）．超高速計算システム，グリッドコンピューティング，並列分散システムソフトウェアの研究に従事．日本応用数理学会，ACM 各会員．