Vol.2021-HPC-178 No.18
2021/3/16

ヒト全ゲノム解析プログラム Genomon の スーパーコンピュータ「富岳」向け最適化と性能評価

鈴木 惣一朗[1,a)]　伊東 聰[2,b)]　酒井 憲一郎[3,c)]　稲田 由江[3]　三吉 郁夫[3]　石川 裕[1]　宮野 悟[2]

概要：我々はスーパーコンピュータ「富岳」開発プロジェクトのターゲットアプリケーションのひとつとして，ヒト全ゲノム解析プログラム Genomon の「富岳」向け移植と最適化を行ってきた．本稿では，その最適化内容と「富岳」での性能評価について報告する．Genomon の実行性能はディスク I/O 性能に依存するため，特に「富岳」に実装された LLIO（Lightweight Layered IO-Accelerator）ファイルシステムをどのように活用したかを中心に報告する．また，Genomon のスループット性能向上のために試みた複数パイプライン同時実行についても報告する．

1. はじめに

2021 年 3 月より共用が開始されるスーパーコンピュータ「富岳」[1] では，その開発において，ハードウェアとソフトウェアで協調して設計を行うコデザインの手法が取られた [2].「富岳」開発プロジェクトでは，国家的に取り組むべき社会的・科学的課題として 9 つの重点課題が挙げられ，その内の「重点課題 2: 個別化・予防医療を支援する統合計算生命科学」[3] におけるコデザイン対象アプリケーションとしてヒト全ゲノム解析プログラム Genomon[4] が選定された．

我々は，ハードウェア開発チームおよびシステムソフトウェア開発チームとの協調のもと，「富岳」への Genomon の移植および最適化作業を続けてきた．既に，Genomon のホットスポットの 1 つであるアライメントプログラム BWA については，その成果を報告している [5]．Genomon は入出力ファイルの合計が数 TB となるため，その実行性能はディスク I/O 性能に大きく依存する．本稿では，「富岳」に実装された LLIO（Lightweight Layered IO-Accelerator）ファイルシステムをどのように活用し最適化を実施したかについて報告する．また，Genomon では実行中に並列度が大きく変化するため遊休ノードが発生し，それが実行性能向上の妨げになっている．我々は，複数パイプライン同時実行により遊休ノード発生を低減することにより，単位時間あたりに解析可能なサンプル数としてのスループット性能を向上させることができた．それについても報告する．

本稿の構成は以下のとおりである．まず，続く 2 章でヒト全ゲノム解析プログラム Genomon について紹介する．3 章では LLIO を含む「富岳」のファイルシステムについて説明する．4 章では LLIO を用いた最適化について報告する．5 章では複数パイプライン同時実行によるスループット性能向上について報告する．最後の 6 章をまとめに当てる．

本稿で報告する「富岳」上での実行性能計測では，都合により，4 章の LLIO 利用の最適化では「ノーマルモード」を 5 章の複数パイプライン同時実行では「ブーストモード」を使用している．ノーマルモードでは，CPU クロック周波数が I/O ノードを兼ねる計算ノードで 2.2GHz，その他の計算ノードで 2.0GHz となる．ブーストモードでは，全計算ノードとも CPU クロック周波数は 2.2GHz である．

2. ヒト全ゲノム解析プログラム Genomon

Genomon は，東京大学医科学研究所ヒトゲノム解析センター（東大医科研 HGC）で開発中の，がんのヒト全ゲノム関連データ解析プログラムである [4]．入力データを加工して出力する処理をパイプライン的につなげることによりゲノム解析を行う，ワークフロー型のアプリケーションである．

1 理化学研究所計算科学研究センター
RIKEN R-CCS
2 東京医科歯科大学 M&D データ科学センター
M&D Data Science Center, Tokyo Medical and Dental University
3 富士通株式会社
Fujitsu Limited
a) soichiro.suzuki@riken.jp
b) sito.dsc@tmd.ac.jp
c) ksakai@fujitsu.com

ⓒ 2021 Information Processing Society of Japan

2.1 Genomon パイプライン

図 1 に「富岳」移植版 Genomon パイプラインのワークフローを示す．解析対象となるゲノムデータは，DNA 断片（リード）の塩基配列情報の集合からなるデータで，1 サンプルあたり tumor と normal と呼ばれる 2 種類のデータが用いられる．

本稿では，ワークフローを構成する処理単位を「タスク」，各タスクの実行実体を「ジョブ」と呼ぶことにする．データ分割により並列実行可能なタスクは，複数のジョブ（バルクジョブ）として実行される．タスクレベルでのワークフロー制御には，Ruffus フレームワーク [6] を用いている．また，ジョブの並列実行制御には，オリジナル Genomon では Sun/Univa Grid Engine ジョブスケジューラを用いていたが，「京」および「富岳」移植版 Genomon では Grid Engine 相当の機能を MPI 上に実装した Virtual Grid Engine (VGE)[7] を使用している．Genomon 本体，Ruffus フレームワーク，VGE は Python で記述されている．

各タスクとそれを構成するコマンドは以下のとおりである（カッコ内はプログラム言語）．

split_files　マッピング計算の前処理
- count_lines リードデータファイルの行数をカウント (C)
- round_robin_split ラウンドロビンでファイル分割 (C)

map_dna_sequence　マッピング計算
- bwa リードの参照配列へのマッピング
- scatter_sam マッピング結果ファイルをマッピング先配列ごとに分割

markdup　マッピング計算の後処理 (1)
- cat マッピング結果ファイルの結合 (Unix コマンド)
- bamsort マッピング結果のソート，フォーマット変換 (C++)
- bammarkduplicates 重複削除，インデックスファイル作成 (C++)
- remove_chr_dir 一時ファイルの消去 (Unix コマンド)

identify_mutations　遺伝子変異解析
- fisher 変異候補抽出 (Python)
- realignment 変異候補検証 (Python)
- indel 変異候補検証 (Python)
- breakpoint 変異候補検証 (Python)
- simplerepeat 変異候補検証 (Python)
- ebfilter 変異候補検証 (Python)

merge_mutation　変異解析結果のマージ
- cat 結果ファイルの結合 (Unix コマンド)

merge_bam　マッピング計算の後処理 (2)
- samtools cat マッピング結果ファイルの結合 (C)
- bai_merge インデックスファイルの結合 (C)
- md5sum 結合マッピング結果ファイルの MD5 計算 (Unix コマンド)

2.2 ファイルサイズ

タスク間およびジョブ内のコマンド間では，全てファイルを通してデータの受け渡しを行っている．今回実行性能計測に用いたリード長が 100 塩基のサンプルデータでは，ファイルサイズは以下のとおりである．

入力リードデータファイル

FASTQ 形式の 2 ファイルを一組として，normal で 136GiB×2，tumor で 180GiB×2．

マッピング結果ファイル (後処理前)

SAM ファイル形式で，合計すると normal で 363GiB, tumor で 473GiB．タスク map_dna_sequence から markdup の間で，リードデータ単位の分割からマッピング先配列単位の分割に並列化の軸が変わるため，scatter–gather 型のファイル操作が行われる．

マッピング結果ファイル (後処理後)

BAM ファイル形式で，合計すると normal で 95GiB，tumor で 124GiB．また，タスク markdup 内の bamsort および bammarkduplicates コマンドでは，最大で 40GiB 程度の入出力ファイルと 10GiB 程度のコマンド内テンポラリファイルを扱う．

変異検証結果ファイル

タスク identify_mutations の各コマンドが出力する変異検証結果ファイルは，合計しても数百 KB 程度の大きさである．

コントロールパネルファイル

タスク identify_mutations の ebfilter コマンドでは，統計検証用に，コントロールパネルと呼ばれる解析対象とは別の 11 サンプルの BAM 形式マッピング結果ファイル（76〜124GiB）にランダムアクセスする．コントロールパネルファイルの総量は 1.15TiB である．

参照配列ファイル，DB ファイル

タスク map_dna_sequence の bwa コマンドおよびタスク identify_mutation 内のいくつかのコマンドでは，参照配列データと変異データベースを読み込む．これらファイル群の合計サイズは 8.7GiB である．

2.3 使用ノード数の決定

我々は，与えられた計算機資源（例えば「富岳」全ノードなど）を使用すると，1 日に何サンプルの遺伝子解析計算が可能か，といった意味での計算性能に興味がある．そのためには，Genomon の計算性能を，使用したノード数で正規化したスループット性能で評価しておくと都合がよい．

「富岳」は 1 ノードに 1CPU を搭載し，その CPU A64FX[8] は 4 つの Core Memory Group (CMG) からなる NUMA 構成となっている．そのため，CMG を単位として 1 ノードに 4 ジョブを割り振ることとする．Genomon では，計算中に同時実行可能なジョブ数が，4 (split_files)，5424 (map_dna_sequence)，174 (markdup)，

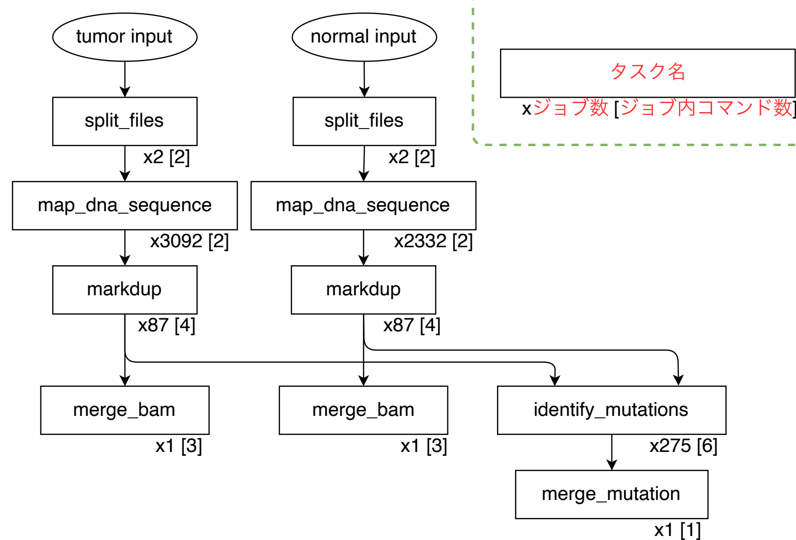

図 1 「富岳」移植版 Genomon パイプラインのワークフロー

277 (identify_mutations, merg_bam) と大きく変化する．そのため使用ノード数を増やしても，計算途中での遊休ノードの発生により，正規化したスループット性能は低下する．逆に，1 サンプルの解析に必要な時間にこだわらなければ，使用ノード数を減らし遊休ノードの発生を抑えることにより，正規化したスループット性能をいくらでも上げることができる．したがって，スループット性能を議論するためには，使用ノード数になんらかの制限を設ける必要がある．我々は，Genomon の実際の医療面への応用を考慮して，「富岳」運用開始時に入手可能な 150 塩基長のサンプルデータの解析にかかる時間の現実的に許容可能な値を 1 日と想定し，計算に必要なノード数の下限を 96 と見積もった．本稿での計算性能評価は，全て 96 ノード使用時のものである．ただし，今現在 150 塩基長の完全なサンプルデータが入手できていないため，本稿での計算は全て 100 塩基長サンプルデータに対して行っている．

96 ノード使用時の MPI プロセス数は 384 となり，この内 1 つを VGE のマスタプロセスに割り振るため，同時実行可能なジョブ数は 383 となる．

3. 「富岳」のファイルシステム

ここでは，LLIO を含む「富岳」のファイルシステムについて紹介する [9]．なお，この章では「ジョブ」を，Genomon パイプラインを構成するタスクの実行実体としてではなく，通常の意味（スーパーコンピュータの「ジョブ」管理システム，等）で用いている．

3.1 階層ストレージ

図 2 に「富岳」のシステム構成を示す．「富岳」は大容量かつ高速なストレージシステムを実現するため，階層化ストレージを採用している．第 1 階層は半導体ディスク（以下 SSD）から構成されるストレージ，第 2 階層はハードディスクから構成されるストレージである．第 1 階層用の SSD は，計算ノード 16 ノードあたり 1 ノードに搭載される．第 2 階層ストレージは 6 ボリュームで構成される．

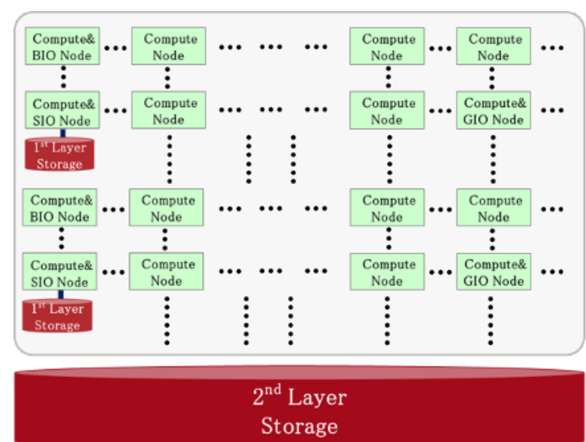

図 2 「富岳」階層ストレージ

第 1 階層ストレージ用にはファイル I/O ミドルウェアとして LLIO (Lightweight Layered IO-Accelerator) が提供される．LLIO は SSD を第 2 階層ストレージのキャッシュあるいは一時ファイル格納領域として使い，TofuD インターコネクトの RDMA 機能を非同期 I/O 処理に用いることにより高速化している．

第 2 階層ストレージ用にはファイルシステムとして Lustre ベースの FEFS (Fujitsu Exabyte Filesystem) が提供される．FEFS はユーザのホーム領域やデータ保存領域として用いる大容量のグローバルファイルシステムである．

第 1 階層ストレージおよび第 2 階層ストレージの I/O スループット性能を表 1 に示す．ここで，第 1 階層は計算ノードあたり，第 2 階層ストレージは 1 つのボリュームあたりの性能である．

ⓒ 2021 Information Processing Society of Japan

表 1　階層ストレージのスループット性能: IOR ベンチマーク実測値

第 1 階層	Write	126 MiB/s/node
	Read	310 MiB/s/node
第 2 階層	Write	197 GiB/s/vol.
	Read	205 GiB/s/vol.

3.2　LLIO が提供する機能

LLIO は，表 2 に示す，用途に応じた 3 種類の機能を計算ノード上のアプリケーションに提供する．

ノード内テンポラリ

ノード内テンポラリは，計算ノード内のみで使用される一時ファイルを格納するのに適したファイルシステムである．ノード内テンポラリの名前空間は，各計算ノードで独立している．「富岳」の計算ノードはローカルディスクを持たないため，ローカルディスクの代用としても利用できる．

共有テンポラリ

共有テンポラリは，計算ノード間で共有される一時ファイルを保存するのに適したファイルシステムである．共有テンポラリ領域の名前空間は，ジョブに割り当てられた計算ノード間で共有される．並列プロセスから 1 つのファイルにアクセスしたり，ジョブ内のプロセス間でファイルを受け渡したりする用途に利用する．

第 2 階層ストレージのキャッシュ

第 2 階層ストレージのキャッシュは，グローバルファイルシステムである FEFS のキャッシュ領域である．第 2 階層ストレージのキャッシュに書き込まれたデータは，アプリケーションの実行とは非同期に SSD から FEFS 上のファイルに書き込まれる．ジョブ終了時には，SSD 上の未書き出しデータのグローバルファイルシステムへの書き出し完了を待ち合わせる．第 2 階層ストレージのキャッシュに読み込まれたデータは，SSD 上に残すことで再読み込みが高速化される．また，実行ファイルや入力ファイルなどジョブ内の全計算ノードから読み込まれるファイルは，専用の llio_transfer コマンドを使用して，ジョブ開始時にグローバルファイルシステムからジョブに割り当てられた全 SSD にコピーできる．これによりグローバルファイルシステム上の 1 ファイルへのアクセス集中を防ぎ，実行ファイルや入力ファイルの読み込み時間を短縮できる．

3.3　LLIO のファイル容量

第 1 階層用に，16 計算ノードに 1 つ，容量 160GB の SSD が搭載されている．1 ノードあたりの容量は 100GB であるが，実際にユーザが LLIO 領域として使用できる容量はノードあたり約 87GiB となっている．ユーザはジョブ実行開始時に，ジョブ内で使用するノード内テンポラリ領域と共有テンポラリ領域の容量を，それぞれノードあたりの量で指定し，87GiB からこれらを引いた残りが第 2 階層キャッシュ領域に割り当てられる．

ノード内テンポラリ領域に関しては，可能な最大ファイルサイズは，このジョブ開始時に指定した値に等しい．共有テンポラリ領域に関しては，可能な最大ファイルサイズはストライプ数に依存する．96 ノード使用時にストライプ数 6 を指定するように，使用可能な SSD 数に等しいストライプ数を指定した場合には，最大可能ファイルサイズはノードあたりで指定した量に使用ノード数を乗じた値になる．

4.　LLIO を用いた最適化

まず，以下の方針で LLIO を使用することを検討した．

ノード内テンポラリ　markdup 以外のタスクにおいて，タスク内プログラム間でのデータ受け渡しに使用．

共有テンポラリ　タスク間でのデータ受け渡しに使用．また，ファイルサイズが大きいタスク markdup において，タスク内プログラム間でのデータ受け渡し，およびプログラム内テンポラリファイルに使用．

第 2 階層キャッシュ　入力 FASTQ ファイルと計算結果ファイル，および，Genomon で使用するシステム提供以外のプログラム，共有ライブラリファイル，Python モジュール類もキャッシングの対象とする．

ストライピングの設定は，第 2 階層上の入出データファイルはストライプサイズ 2MiB，ストライプ数 32，LLIO の共有テンポラリおよび第 2 階層キャッシュ領域はストライプサイズ 2MiB，ストライプ数は 96 ノード使用時に可能な最大数である 6 とした．

以下では，各タスクごとに，LLIO の利用方法をどのように調整したかについて報告する．map_dna_sequenece 以外のタスクでは，各 CMG 上では最大でも 1 つのジョブしか実行されないため，パイプラインの進行で律速となる最も実行時間のかかったジョブについてのみ議論している．タスク map_dna_sequence では，CMG あたり 14 程度のジョブが実行されるため，タスク内コマンドの実行時間としては，ジョブ間平均値を考察の対象としている．

なお，Genomon では入力ファイルを全て実行開始前に LLIO 領域に移動させることができないため，実行中に他ユーザと共用している第 2 階層ファイルシステムへのアクセスが必須となる．その他にも，VGE によるジョブ割り振り先 CMG の選択，ストライピング時のファイルの SSD への分散順などにもランダム性があり，実行性能計測結果の再現性に影響を与えてる．特に実行時の第 2 階層ファイルシステムの負荷状況によっては，実行時間に大きな遅延が発生する．本稿の報告では，実行時間に明らかな異常が見られれば時間をおいて再計測する，また，可能なら同条件で複数回計測して最速実行時の値を採用することにしている．しかし，それでも上記の問題が原因で，それなりの大きさの実行時間のぶれが含まれていることを留意いただきたい．

表 2　LLIO の提供機能

機能	名前空間	用途
ノード内テンポラリ	計算ノード内	一時ファイル
共有テンポラリ	ジョブ内共有	一時ファイル
第 2 階層ストレージのキャッシュ	グローバルファイルシステム透過	保存ファイル

表 3　split_files: 最も時間のかかったジョブの実行時間内訳（秒）

ケース	A	B	C	D
共有テンポラリ		✓	✓	✓
第 2 階層キャッシュ			✓	✓
一部キャッシュ停止				✓
split_file 改良				✓
count_lines	352	368	690	347
split_file	2173	1905	1511	1146

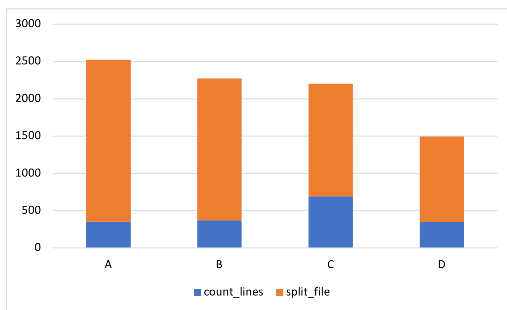

図 3　split_files: 最も時間のかかったジョブの実行時間内訳（秒）

4.1　タスク split_files

タスク split_files では，count_lines コマンドにより第 2 階層ファイルシステム上の入力 FASTQ ファイルの行数をカウントし，split_file コマンドにより同ファイルを 4 行単位でラウンドロビン分割する処理が，normal と tumor で 2 つずつ計 4 つのジョブが同時実行される．表 3，図 3 に検討した各ケースについて，最も時間のかかったジョブでの実行時間内訳を示す．ケース (A) が LLIO 未使用時の実行時間，ケース (B) が分割後ファイルの出力先を共有テンポラリ領域にしたもの，ケース (C) は count_lines コマンド実行時に FASTQ ファイルをキャッシュに載せ，split_file コマンドはキャッシュ上のデータに対して実行するようにしたものである．第 2 階層キャッシュの使用により，split_file は 400 秒ほど高速化されたが，逆に count_lines は 320 秒ほど遅くなっている．これは，キャッシュに載せるためのオーバヘッドが原因と考えられる．そこで，ケース (D) では FASTQ ファイルのみ第 2 階層キャッシュの使用は止め，同時に split_file コマンドのファイル入力部分を，マルチスレッドによる並列読み出し化により高速化を図った．改良後の split_file コマンドは，改良前のキャッシュ上データ読み込み時より高速であったため，ケース (D) を採用することとした．なお，split_file コマンドに施したファイル入力部分の高速化は，既に count_lines に実施済みである．

表 4　map_dna_sequence: 各コマンド実行時間（秒）のジョブ間平均

ケース	A	B	C	D
ノード内テンポラリ		✓	✓	✓
共有テンポラリ		✓	✓	✓
第 2 階層キャッシュ			✓	✓
llio_transfer				✓
normal				
bwa	95	140	100	88
scatter_sam	33	8	11	9
tumor				
bwa	108	131	109	91
scatter_sam	24	4	9	8

4.2　タスク map_dna_sequence

タスク map_dna_sequence は，分割 FASTQ ファイルに対して bwa コマンドにより参照配列へのマッピングを行い，続いて scatter_sam コマンドによりマッピング結果ファイルをマッピング先配列ごとに分割する．タスク map_dna_sequence は，normal と tumor で合計 5424 ジョブあり，CMG あたり 14 前後のジョブが実行される．表 4，図 4 に検討した各ケースについて，実行時間内訳を，normal と tumor それぞれについてのジョブ間平均値として示す．ケース (A) が LLIO 未使用時の実行時間，ケース (B) は，タスク map_dna_sequence の入力となる分割 FASTQ ファイルと出力となるマッピング先配列ごとに分割れたマッピング結果ファイルを共有テンポラリに置き，タスク内での一時ファイルである bwa の出力をノード内テンポラリに置いたものである．scatter_sam には LLIO の効果が素直に現れているが，bwa に関しては逆に実行時間が増えている．ケース (C) は第 2 階層キャッシュを有効にしたもので，bwa コマンドの入力となる参照配列データをキャシングすることの効果を期待したが，ケース (B) よりは高速化されたが，LLIO 未使用のケース (A) とは同程度の実行速度となった．ケース (D) は，llio_transfer コマンドにより，Genomon 実行開始前に予め参照配列ファイルを全 SSD に転送しておいたケースである．このケースでは，bwa コマンド実行時の第 2 階層ストレージへのアクセスが完全に無くなり，また SSD へのアクセス競合も減っているため，最速となった．これにより，ケース (D) を採用した．

4.3　タスク markdup

タスク markdup は，マッピング先配列単位で並列化され，174 ジョブが同時実行される．bamsort コマンドと

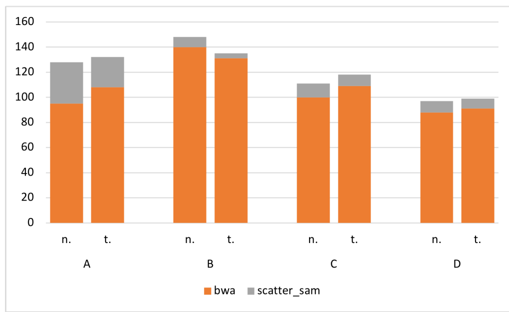

図 4 map_dna_sequence: 各コマンド実行時間（秒）のジョブ間平均（n.=normal, t.=tumor）

表 5 markdup: 最も時間のかかったジョブの実行時間内訳（秒）

ケース	A	B	C
共有テンポラリ		✓	✓
第 2 階層キャッシュ			✓
cat	433	329	326
bamsort	4221	1276	1513
bammarkduplicates	3582	3177	3370
remove_chr_dir	1	16	16

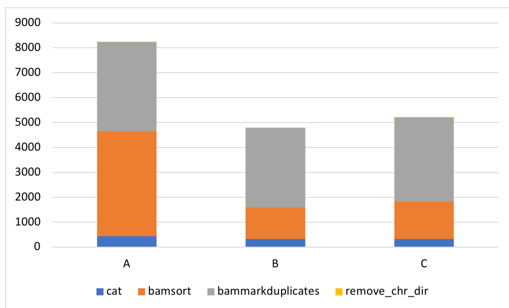

図 5 markdup: 最も時間のかかったジョブの実行時間内訳（秒）

bammarkduplicates コマンドとも，入出力ファイルおよびコマンド内テンポラリファイルが数十 GB になるジョブがあるため，これらはノード内テンポラリ領域に置くことはできない．表 5，図 5 に検討した各ケースについて，最も時間のかかったジョブでの実行時間内訳を示す．ケース (A) が LLIO 未使用時の実行時間内訳である．ケース (B) では，全ての入出力ファイルを共有テンポラリ領域に置いたもので，素直に LLIO 使用の効果が出ている．ケース (C) は第 2 階層キャッシュを使用した場合だが，タスク markdup ではキャッシングの対象になるのは，bamsort および bammarkduplicates コマンド自身とその共有ライブラリファイルのみである．ケース (B) と比較して，若干速度低下の傾向が見られるが，ケース (C) を採用した．なお，remove_chr_dir では約 3 千個のファイルが存在するディレクトリを消去する処理を行うが，対象ディレクトリが共有テンポラリ領域にある時に，明らかな実行時間増加が見られる．この原因については，今後検討する予定である．

表 6 identify_mutations, merge_bam: 最も時間のかかったジョブの実行時間内訳（秒）

ケース	A	B	C	D
ノード内テンポラリ		✓	✓	✓
共有テンポラリ		✓	✓	✓
第 2 階層キャッシュ			✓	✓
一部キャッシュ停止				✓
identify_mutations				
fisher	911	727	767	621
realignment	182	104	154	97
indel	208	144	274	134
breakpoint	134	120	128	110
simplerepeat	7	9	9	6
ebfilter	372	292	297	279
merge_bam				
samtools cat	1002	164	1089	191
bai_merge	7	5	7	5
md5sum	959	634	591	635

4.4 タスク identify_mutations, タスク merge_bam

タスク identify_mutations の 275 ジョブとタスク merge_bam の 2 ジョブが同時実行される．表 6，図 6 に検討した各ケースについて，最も時間のかかったジョブでの実行時間内訳を示す．ケース (A) は LLIO 未使用時の実行時間内訳である．ケース (B) では，共有テンポラリ領域上のマッピング結果ファイルを参照するように，また，タスク identify_mutations 内の一時ファイルをノード内テンポラリ領域に置くようにしたもので，素直に LLIO の効果が出ている．ケース (C) は，第 2 階層キャッシュを有効にしたものだが，タスク identify_mutations では若干の，タスク merge_bam では samtools cat コマンドで顕著な速度低下が見られる．samtools cat は，マッピング結果ファイルを結合する，ほぼ I/O 処理だけのコマンドである．第 2 階層キャッシュの有効化により，SSD のアクセス競合が増加し，I/O 性能が低下しているものと考えられる．ケース (D) は，タスク identify_mutations においてコントロールパネルファイルをキャッシングの対象から外したものである．コントロールパネルファイルは，ランダムアクセスのためキャッシングの効果は低く，キャシング無効化の影響はほとんどないものと思わる．逆に，SSD のアクセス競合が減少したことにより，タスク identify_mutations では若干の，タスク merge_bam では samtools cat コマンドで顕著な計算速度の回復が見られた．このことから，ケース (D) を採用した．

4.5 LLIO 最適化まとめ

96 ノードを使用した 100 塩基長サンプルに対する Genomon の全実行時間は，LLIO 未使用時は 4.05 時間であったが，LLIO 最適化後は 2.72 時間まで短縮できた．図 7 に両実行時の，各 CMG 上でのジョブ実行状況を示す．

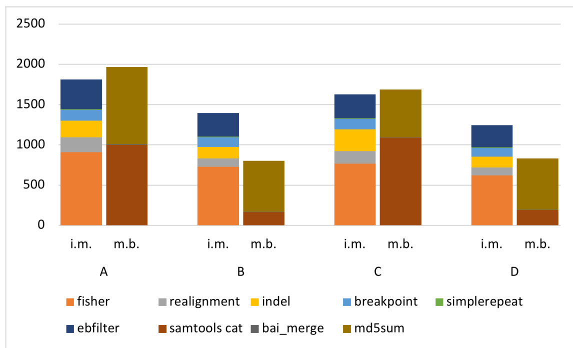

図 6 identify_mutations(i.b.), merge_bam(m.b.): 最も時間のかかったジョブの実行時間内訳（秒）

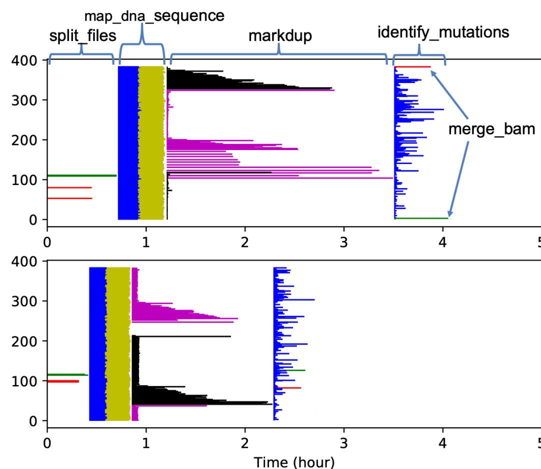

図 7 ジョブ実行状況（96 ノード 384CMG 使用）．上が LLIO 未使用時，下が LLIO 最適化後．縦軸方向に CMG が配置され，各 CMG 上でのジョブ実行を横線で表している．

LLIO 最適化後の実行における LLIO 使用量は，ノードあたりで，ノード内テンポラリが 40MiB, 共有テンポラリが 21.4GiB，第 2 階層キャッシュが 2.9GiB であった．

5. 複数パイプライン同時実行によるスループット性能向上

図 8 に，前章までと同様に，1 サンプルを対象に Genomon を単独実行した時の各 CMG の稼働状況を示す（本章の性能計測では「ブーストモード」を用いているため，「ノーマルモード」を使用した前章の結果より約 14 分実行時間が短縮している）．この図では，縦軸方向に CMG が配置され，横線はその CMG 上でジョブが実行されていることを表している．粒度の小さい 5424 個のジョブからなるタスク map_dna_sequence を除くと，計算に寄与していない CMG がかなり存在する．全 CMG における計算に寄与している時間の割合により CMG 稼働率を定義すると，その値は 22.4%でしかない．

Genomon のスループット性能を向上させるには，個々のジョブの計算速度向上だけでなく，CMG 稼働率も増加させる必要がある．そこで，我々は同一なノード配置上で

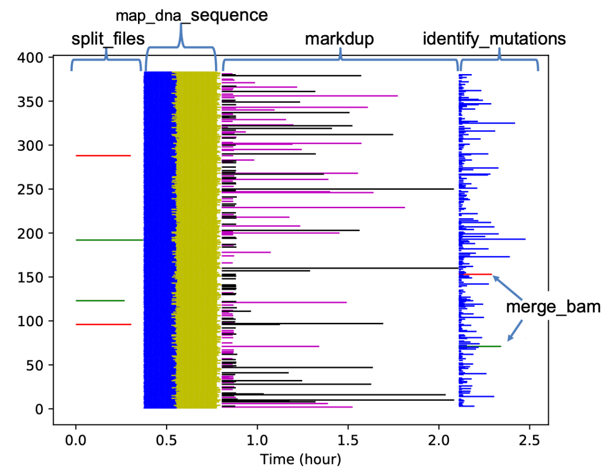

図 8 96 ノード 384CMG 上での 1 パイプライン実行時の各 CMG でのジョブ実行状況．CMG 稼働率 22.4%．スループット 0.40/h.

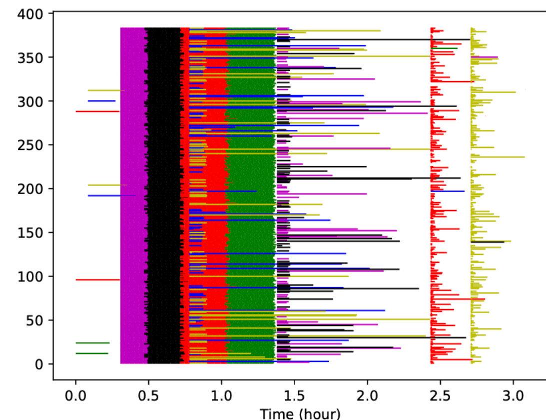

図 9 2 パイプライン同時実行時の各 CMG 上でのジョブ実行状況．CMG 稼働率 42.1%．スループット 0.65/h.

複数のパイプラインを同時実行させ，遊休 CMG による隙間を他パイプラインのタスク map_dna_sequence のジョブで埋めることで CMG 稼働率を増加させることを試みた．その実現のために，以下の 2 つのアイデアを用いた．

- 複数のパイプラインは，完全に同時ではなく，ある間隔をおいて実行開始する．今回は，5 分間隔とした．
- ジョブの並列実行を管理する VGE は，キューに投入された順にジョブを空き CMG に割り振るが，新たに低優先度キューを追加して，低優先度キューのジョブは通常キュー内の待ちジョブ数が 0 になってから空き CMG に割り振るようにした．そして，タスク map_dna_sequence のジョブを全て低優先度キューに入れることにした．

図 9 に 2 パイプライン同時実行の様子を示す．期待したように，第 1 パイプラインのタスク markdup のジョブが，既に低優先度キューに入っている第 2 パイプラインのタスク map_dna_sequence のジョブを追い越して実行され，そ

ⓒ 2021 Information Processing Society of Japan

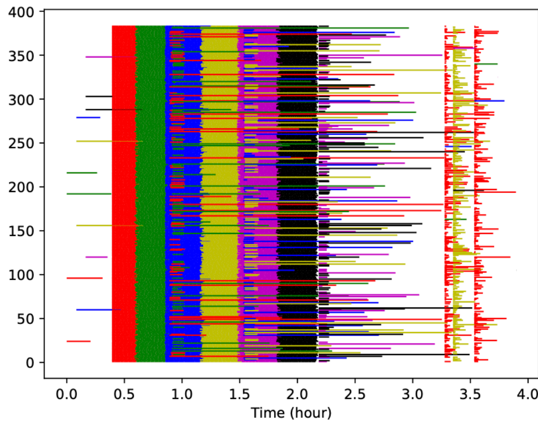

図 10 3 パイプライン同時実行時の各 CMG 上でのジョブ実行状況．CMG 稼働率 54.4%．スループット 0.77/h．

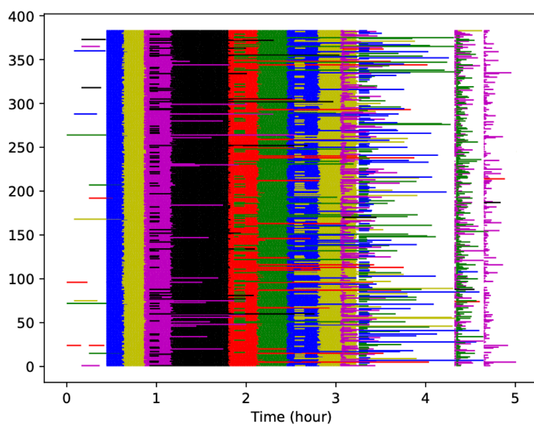

図 11 4 パイプライン同時実行時の各 CMG 上でのジョブ実行状況．CMG 稼働率 63.4%．スループット 0.80/h．

の後，その隙間を埋める形で第 2 パイプラインのタスク map_dna_sequence のジョブが実行されている．これにより，CMG 稼働率は 22.4%から 42.1%に増加し，1 時間あたりの計算可能サンプル数としてのスループットも 0.40 から 0.65 に向上した．

図 10 に同様な手法で行った 3 パイプライン同時実行の様子を，図 11 に 4 パイプライン同時実行の様子を示す．5 パイプライン以上の同時実行は LLIO の容量不足で不可能であった．

図 12 に複数パイプライン同時実行時の CMG 稼働率とスループット性能の関係を示す．4 パイプライン同時実行まで，パイプライン数の増加とともに CMG 稼働率が増加し，スループット性能も向上していく傾向が得られた．1 パイプライン実行と 4 パイプライン実行を比較すると，CMG 稼働率は 2.8 倍になっているがスループットは 2 倍にしか増加していない．これは，CMG 稼働率の増加にともない SSD へのアクセス競合も増加してしまい，I/O 性能

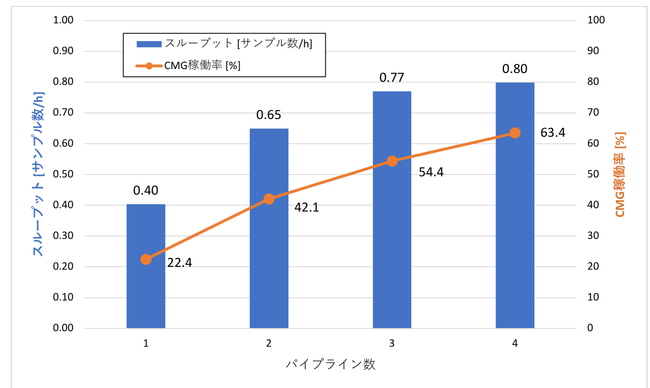

図 12 複数パイプライン同時実行時の CMG 稼働率とスループット性能の関係

が低下しているためだと考えられる．

表 7 に複数パイプライン同時実行時の，CMG 稼働率，スループット性能，LLIO 使用量についてまとめた．

6. まとめ

本稿では，ヒト全ゲノム解析プログラム Genomon のスーパーコンピュータ「富岳」上での最適化，特に「富岳」に実装された LLIO（Lightweight Layered IO-Accelerator）ファイルシステムの有効活用による I/O 性能の向上，および，遊休ノード削減のための複数パイプライン同時実行によるスループット性能向上について報告した．

LLIO を有効活用することにより，100 塩基長のサンプルデータに対する 96 ノード使用での計算時間を，約 4 時間から 2 時間 45 分程度まで短縮できた．我々は，ノード内テンポラリ，共有テンポラリ，第 2 階層キャッシュからなる LLIO の 3 機能を単純に使える箇所全てで利用すると，SSD のアクセス競合が増加して，逆に実行性能が低下する可能性があることを経験した．特に第 2 階層キャッシュについては，アクセス競合の増加だけでなく，キャッシュに載せるためのオーバヘッドの発生なども見られた．ランダムアクセスのためキャッシュに載せてもデータ再利用性が低いファイルについては，その利用の是非を慎重に検討する必要がある．

複数パイプラインの同時実行では，Virtual Grid Engine (VGE) への低優先度キューの導入という単純なアイデアで，1 パイプライン実行時に 24.4%であった CMG 稼働率を，4 パイプライン同時実行時に 63.4%まで増加させることができた．これにより，1 時間あたりの解析可能サンプル数としてのスループット性能を 0.4 から 0.8 へと 2 倍向上させることができた．

「富岳」を「京」と比較すると，遺伝子解析計算の実行環境としては，本稿で報告した LLIO の活用以外にも，次のような利点があげられる．CPU アーキテクチャが SPARC64 から Armv8-A となり，遺伝子解析分野で広く使われているオープンソースソフトウェアの利用がより容易になった．

ⓒ 2021 Information Processing Society of Japan

表 7 複数パイプライン同時実行

パイプライン数	1	2	3	4
全体実行時間 (h)	2.48	3.08	3.90	5.01
スループット (サンプル数/h)	0.40	0.65	0.77	0.80
CMG 稼働率 (%)	22.4	42.1	54.4	63.4
ノード内テンポラリ使用量 (GiB/node)	0.04	0.04	0.04	0.04
共有テンポラリ使用量 (GiB/node)	21.6	35.3	51.6	52.8
第 2 階層キャッシュ使用量 (GiB/node)	2.9	5.2	7.5	9.9

MPI プロセスあたりのコア数が 8 から 12 へ増え，SIMD 幅も 128bit から 512bit に増えたことにより，解析パイプラインを構成する個々のプログラムをより高速に実行できる可能性がある．また，アウトオブオーダ資源の増加などにより，Python インタプリタ自身の実行性能も向上している．我々は今後，Genomon の「富岳」上での計算性能と，他アーキテクチャ実行環境上での計算性能とを比較することにより,「富岳」の遺伝子解析計算の実行環境としての可能性を更に追求していく予定である.

なお，本稿で報告した計算性能は,「富岳」共用前評価環境において評価したもので,「富岳」共用開始時の性能を保証するものではありません.

謝辞 池田奈生氏には，Genomon ワーキンググループの初期の活動においてサポートしていただきましたことを感謝いたします．本研究の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等)」および文部科学省ポスト「京」重点課題 2「個別化・予防医療を支援する統合計算生命科学」のもとで実施されたものです.

参考文献

[1] RIKEN R-CCS: スーパーコンピュータ「富岳」について，https://www.r-ccs.riken.jp/jp/fugaku.
[2] Sato, M., Ishikawa, Y., Tomita, H. et al.: Co-Design for A64FX Manycore Processor and "Fugaku", *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20, IEEE Press (2020).
[3] ポスト「京」重点課題 2： 個別化・予防医療を支援する統合計算生命科学，http://postk.hgc.jp.
[4] Chiba, K., Okada, A. and Shiraishi, Y.: Genomon, https://genomon.readthedocs.io/.
[5] 鈴木惣一朗，伊東　聰，稲田由江，池田奈生，三吉郁夫，松葉浩也，石川　裕，宮野　悟: DNA アライメントプログラム BWA-MEM の Arm SVE 移植と「富岳」試作機での性能評価，情報処理学会研究報告，Vol. 2020-HPC-173, No. 16, pp. 1–8 (2020).
[6] Goodstadt, L.: Ruffus: a lightweight Python library for computational pipelines, *Bioinformatics*, Vol. 25 21, p. 2778–2779 (2010).
[7] Ito, S., et al.: Virtual Grid Engine: Accelerating thousands of omics sample analyses using large-scale supercomputers, *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 387–392 (2018).
[8] Fujitsu: A64FX, https://github.com/fujitsu/A64FX.
[9] Fujitsu: 特集：スーパーコンピュータ「富岳」，富士通テクニカルレビュー， No. 3 (2020).