情報処理学会論文誌　コンピューティングシステム　Vol.9　No.2　15–33 (July 2016)

スーパーコンピュータ「京」上での エクソーム解析パイプラインの開発

青山 健人[1,2,a)]　角田 将典[1,b)]　松崎 由理[2,c)]　石田 貴士[1,2,d)]　秋山 泰[1,2,e)]

受付日 2015年11月6日, 採録日 2016年2月20日

概要：近年，次世代シーケンサなどに代表される実験技術の向上による爆発的な生物学データの増加に対応するため，スーパーコンピュータを用いた効率的なデータ解析，処理技術の開発は喫緊の課題となっている．ゲノム情報のうちタンパク質に翻訳されるエクソン領域の配列のみを網羅的に解析するエクソーム解析は，ゲノム配列全体を対象とする場合と比べて処理量は大幅に削減されるため効率的な解析が可能となるが，一方で将来の個別化医療に向けた解析では，数百人から千人規模のデータを現実的な時間で処理する必要があり，小型の PC クラスタでは処理が追いつかない大規模な解析が必要である．本研究では，理化学研究所のスーパーコンピュータ「京」上にエクソーム解析パイプラインを開発し，大規模エクソーム解析を目的とした生命情報解析環境を構築した．「京」上で実際に動作するエクソーム解析パイプラインの構築に加え，パイプラインの各処理で MPI による Master-Worker モデルでタスク分散処理を行うことで投入ジョブ数を軽減し，さらにタスクの分割などを改良することで，並列性能を改善して処理の高速化を図った．

キーワード：スーパーコンピュータ「京」, エクソーム解析, パイプライン, Genomon-exome, MPI, MPIDP

Development of Exome Analysis Pipeline on the K Computer

Kento Aoyama[1,2,a)]　Masanori Kakuta[1,b)]　Yuri Matsuzaki[2,c)]　Takashi Ishida[1,2,d)]
Yutaka Akiyama[1,2,e)]

Received: November 6, 2015, Accepted: February 20, 2016

Abstract: Recently, development of efficient biological data analysis systems on a supercomputer has been highly required in order to tackle the vast amount of biological data generated by the latest experimental techniques such as a next-generation DNA sequencer. Exome analysis, which analyzes the regions in a genome that will remain in a matured RNA, is useful because it targets only exonic sequences in a genome and enables effective search for important mutations throughout the genome. On the other hand, to meet the demands of current medical researches such as application to personalized genome analysis, we need to deal with the situation in which hundreds to thousand exome sequences are needed to be analyzed in realistic time. It is of significant importance to develop a high-performance large-scale sequence analysis environment. In this study, we developed an exome analysis pipeline on the K computer. We not only developed a pipeline useful for biologists on a supercomputer but also improved the parallel performance of the pipeline using a master-worker model task distribution framework implemented by MPI and efficient task partitioning strategy.

Keywords: the K computer, exome analysis, pipeline, Genomon-exome, MPI, MPIDP

[1] 東京工業大学大学院情報理工学研究科計算工学専攻
Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[2] 東京工業大学情報生命博士教育院
Education Academy of Computational Life Sciences, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan

[a)] aoyama@bi.cs.titech.ac.jp
[b)] kakuta@bi.cs.titech.ac.jp
[c)] matsuzaki@acls.titech.ac.jp
[d)] ishida@cs.titech.ac.jp
[e)] akiyama@cs.titech.ac.jp

1. 背景

近年の遺伝子情報解析分野では，次世代シーケンサの登場にともなう塩基配列読み取りスループットの飛躍的な向上によって日々蓄積される膨大な生命情報の計算機上の解析がボトルネックとなっており，効率的な解析手法やデータ解析のための大規模な生命情報解析環境の需要が高まっている．そこで計算機上の一連の解析作業を自動化し，計算資源を有効活用した解析を提供するパイプラインソフトウェアが多く開発されており，実際の研究現場でも並列計算機上に実装したパイプラインを利用した解析が行われている [1], [2], [3].

日本における大規模な並列計算機の例として代表的なものが，理化学研究所に設置されたスーパーコンピュータ「京」である [4].「京」は 2011 年に計算性能の指標である LINPACK ベンチマークで 10.51 PFLOPS を記録した実績を持つ，日本で最大規模の並列計算機である．生命情報の解析においても「京」を利用することで大きな成果が得られることが期待されているが，現時点ではその活用成果は限定的である．生命情報解析分野の研究では，これまでに多くの開発主体によるソフトウェアが研究コミュニティで共有され，解析現場ではそれらから数種類のソフトウェアを組み合わせて研究が実施されてきた．大規模な生命情報解析を「京」で実施する際には，開発言語，利用ライブラリ，ライセンスなどがそれぞれ異なる多種多様なソフトウェア群を，一般的なアーキテクチャとは異なる「京」に移植する必要がある．この制約のため，「京」上で生命情報解析を行う環境の整備が大きな課題となっている．

具体的には，現在使用されている解析パイプラインが起動する様々なソフトウェアを実行するために必要な開発言語環境が「京」で提供されていないこと，生命情報解析アプリケーションでは消費メモリが増大しがちであるのに対し，「京」の仕様では 1 ノードあたりのメモリ容量が少ないといった問題点があげられる．また，大規模並列計算機システム上での生命情報解析ではタスク数の増大や入出力のデータ容量の大きさが実行時に並列性能を低下させる要因となることが多く，「京」のように多数のノードを使用する計算機システム上で効率的に計算を実行するためにはこれらの問題の解決も必要となっている．

大規模な計算機環境を必要とする生命情報解析課題の 1 つとして，本研究ではエクソーム解析を扱う．エクソーム解析は遺伝子配列解析の手法の 1 つで，ヒトの全塩基配列の中のエクソン（exon）と呼ばれる領域のみを解析することで，機能的に重要な変異を効率的に検出する [5]．がんゲノムや遺伝性疾患などの研究を行う際には，統計的に有意となる差を検出するために多くの患者のサンプルに対する解析が必要となり，近年では数百人規模のデータに対する解析も計画，実行されている [6]．そのため，通常の全ゲノム解析に比べると 1 サンプルあたりのシーケンシング，および計算機による解析コストは低くなるものの，全体としては多数のサンプルを扱うため，より多くのコスト，資源が必要となる．数百人規模のサンプルに対するエクソーム解析を実現するためには，「京」のような大規模計算機システム上での効率的なエクソーム解析パイプラインの構築が急務である．

本研究では，エクソーム解析ワークフローを有するパイプラインソフトウェアを，東京大学医科学研究所で開発された Genomon-exome [1] をベースとしてスーパーコンピュータ「京」上で開発する．「京」上での効率的な実行のため，パイプラインの各処理を Master-Worker モデルでタスク分散を行って投入ジョブ数を軽減する．さらに，タスク分割を工夫することで処理の並列性能を改善して高速化を図ることで，より効率的な解析パイプラインを提案する．

2. エクソーム解析パイプライン Genomon-exome

2.1 エクソーム解析の流れ

一般的な遺伝子解析はサンプルの塩基配列を取得する生物学的な実験（wet experiments）と，計算機上で解析する計算科学的な実験（dry experiments）に分けられる（図 1）．

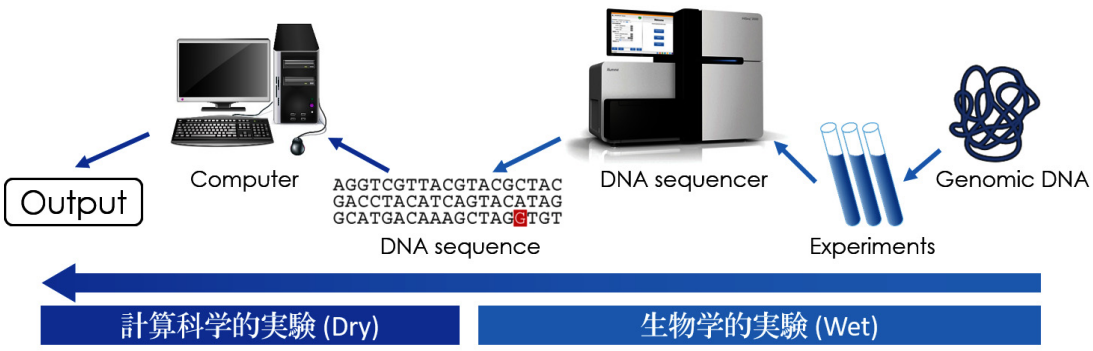


図 1　遺伝子解析の流れ（文献 [5] を改変）
Fig. 1　Gene analysis procedure (Ref. [5] modified).

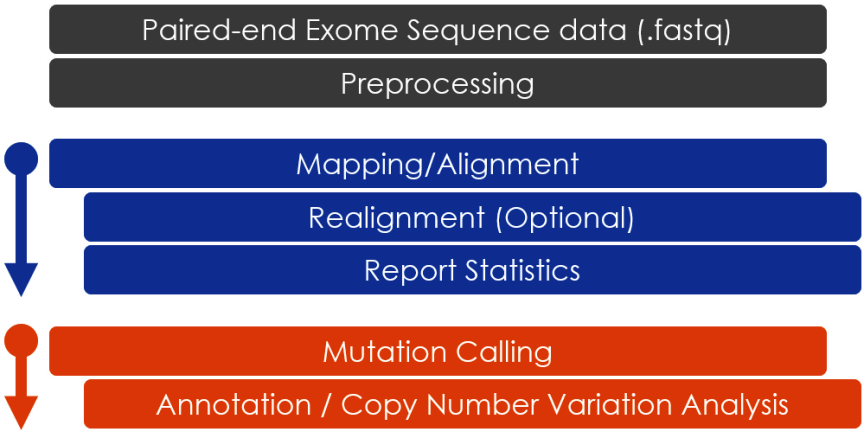


図 2　Genomon-exome によるエクソーム解析ワークフロー
Fig. 2　Exome analysis workflow on Genomon-exome.

表 1　Genomon-exome で利用されるソフトウェア群
Table 1　Software required in Genomon-exome.

ソフトウェア名	利用機能の内容	使用言語
BWA [7]	参照配列へのマッピング，アライメント	C
GATK [8]	マッピング結果のリアライメント	Java
SAMtools [9]	マッピング結果ファイル（SAM/BAM）の操作	C
Picard [10]	マッピング処理の統計情報の出力，SAM/BAM の操作	Java
ANNOVAR [11]	変異の候補の一覧にアノテーションを付与	Perl
cutadapt [12]	PCR アーティファクト（アダプタ配列）の除去	Python
maq [13]	Solexa 形式の入力配列の Sanger 形式への変換	C++
bedtools [14]	エクソン領域情報を示す BED 形式ファイルの操作	C++
Bioconductor [15]	Copy Number Variation 解析	R

計算科学的な実験では生物学的な実験の結果としてシーケンサから得られる大量の短い DNA 断片の塩基配列（リード）を参照となる配列の類似した位置に配置（アライメント）して元の配列を復元するマッピング処理，マップされた配列の比較からサンプルが含む 1 塩基の変異を検出する処理などが行われ，さらに出力結果をデータベースと関連付けることでサンプルに含まれる変異と遺伝子の関連を調べることがよく行われる．

2.2　Genomon-exome

Genomon-exome [1] は文部科学省新学術領域研究「システムがん」のもとで，東京大学医科学研究所ヒトゲノム解析センター宮野研究室および京都大学大学院医学研究科腫瘍生物学講座小川研究室の共同研究で 2012 年に開発された，ジョブ管理システムを備えた汎用 PC クラスタ向けのエクソーム解析パイプラインソフトウェアである．様々な OSS（Open Source Software）を組み合わせて実現され，シーケンサが出力した FASTQ 形式のファイルをヒトゲノム参照配列（hg19）に対してマッピングし，変異の候補一覧やアノテーション結果を出力するなど，エクソーム解析の一連のワークフローが利用できる（**図 2**）．また，パイプラインの各処理は入力データを分割して複数の実行単位（ジョブ）としてシステムに投入されるため，PC クラスタの計算資源を効率的に利用した高速な処理の実行が可能である．以上の特徴に加え，研究現場での利用実績が豊富なことから，本研究では Genomon-exome を「京」上で開発するパイプラインのモデルとして選んだ．Genomon-exome の各機能では**表 1** に示すソフトウェア群が使用される．

2.3　Genomon-exome の入出力

2.3.1　入力

エクソーム解析の生物学的実験においてシーケンサが出力する，サンプルの塩基配列と各塩基の信頼性を示すクオリティスコアがテキスト形式で記述された FASTQ 形式 [16] のファイルを入力とする．同じ DNA 断片の両側から読み取りを行うことで解析精度を向上させるペアエンド法にも対応している．

2.3.2　出力

マッピング結果が記述される SAM（Sequence Alignment/Map）形式 [17] のファイル，マッピングや前処理などのパイプラインの処理に関する統計情報，マッピング結果をもとにサンプルに含まれる変異（SNPs: Single Nucleotide Polymorphism）を検出して遺伝子と関連付けた結果，遺伝子の複製数の個人差である CNV（Copy Number

Variation）の解析結果などを出力する．

2.4　代表的な機能と構成

Genomon-exome は前処理，マッピング，解析のプロセスによって構成されている．解析プロセスで搭載される機能はソフトウェアの目的によって異なるが，この構成は一般的な遺伝子解析パイプラインソフトウェアと同様である．

2.4.1　前処理（Preprocessing/Read Quality Control）

シーケンサによる入力形式の違いを吸収し，マッピング処理に入力可能な形式に変換する処理である．クオリティスコアの異なる Solexa 形式の Sanger 形式 [16] への変換，生物学的な実験で付与されるが解析に不要なアダプタ配列の除去などの前処理を行う．基本的には単純な文字列処理であり，必要となる計算時間はわずかである．

2.4.2　マッピング，アライメント（Mapping/Alignment）

前処理で得られた FASTQ 形式のファイルに含まれる短い DNA 断片の塩基配列を参照配列（reference genome）の対応する箇所に配置し，サンプルのもとの塩基配列を再構成する処理である．大域的なアライメントの際にサンプルの挿入や欠損となる塩基の存在によって生じるアライメントのミスマッチを修正するため，再度局所的なアライメント（リアライメント）を行うこともある．アライメントには Burrows-Wheeler 変換を用いる BWA [7]，リアライメントには GATK [8] を用いる．この処理は巨大なヒトゲノム配列に対して，すべての DNA 断片について挿入や欠損を考慮して類似する文字列を検索する必要があるため，その計算時間は DNA 断片の数に比例し，特に計算時間を必要とする処理となっている．

2.4.3　変異検出とアノテーション（Mutation Calling and Annotation）

マッピング結果の SAM 形式（BAM 形式）を入力として，含まれる変異のカウントや検定が行われる．Genomon-exome では SAMtools [9]，Picard [10]，オリジナルのスクリプトなどによってリードのフィルタリングを行いながら変異や挿入/欠損を探し，経験ベイズもしくはフィッシャー検定を用いて 1 塩基多型（SNPs）の検出を行う．検出された 1 塩基多型（SNPs）はどのような遺伝子と関連を持つのかを，ANNOVAR [11] によって公開されている遺伝子データベース上の情報と関連付けられる．この処理は DNA 断片の数が少ない場合にはマッピングと同程度の計算時間を必要とするが，計算量は最終的な変異の数などに依存するため，解析する DNA 断片の数が増えるに従って相対的にその計算時間は短いものとなる．

2.5　BWA によるマッピング処理の流れ

Genomon-exome ではジョブ管理システムを利用してパイプラインの各処理を並列に実行しており，特にアライメント処理など計算時間がかかる部分を多数のジョブで並列実行することで実行効率を向上させている．参照配列へのマッピング処理を例として，処理の流れを示す（図 3）．

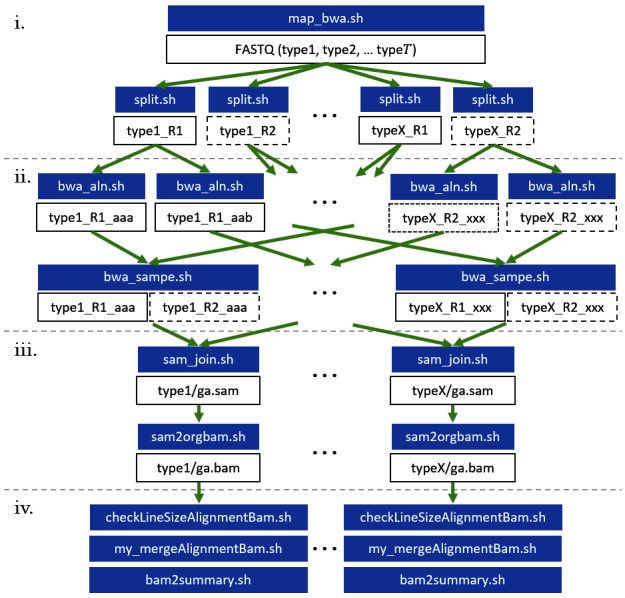


図 3　マッピング処理におけるジョブの流れ
Fig. 3　Job flow of the mapping process.

i. 入力ファイルの分割
入力となる FASTQ ファイル，パラメータを指定してジョブを投入する（`map_bwa.sh`）．並列実行のために入力された FASTQ 形式ファイルを一定クエリごとに分割する（`split.sh`）．

ii. アライメントの実行
分割した各ファイルに対して並列に BWA によるアライメントを実行し（`bwa_aln.sh`），ペアエンド情報を用いてマッピング結果の SAM 形式ファイルを出力する（`bwa_sampe.sh`）．

iii. ファイルの統合と形式の変換
分割された各ファイルを結合して（`sam_join.sh`）同一のサンプル（Type）ごとにバイナリ形式の BAM 形式に変換する（`sam2orgbam.sh`）．また，リード位置による整列，重複リードの除去，インデックスの作成を行う．

iv. 出力の確認と統計出力
入力ファイルと出力ファイルの行数で整合性を確認する（`checkLineSizeAlignmentBam.sh`）．同一サンプル（Type）で別に出力された BAM ファイルがある場合は，マージ後に再度インデックスの作成を行い（`my_mergeAlignmentBam.sh`），最後にマッピング処理の統計情報を出力する（`bam2summary.sh`）．

3.　スーパーコンピュータ「京」

スーパーコンピュータ「京」（略称「京」）[4] は日本の科

学技術分野の発展を牽引するスーパーコンピュータ開発の国家プロジェクトとして取り組まれて 2012 年 7 月に完成した，理化学研究所に設置されている大規模並列計算機である．「京」は科学技術分野の発展を牽引するという理念から幅広い分野の応用に適した汎用計算機として開発されており，CPU には扱いやすい汎用スカラ型 CPU を採用しているほか，特徴的な 6 次元メッシュ/トーラス形状を構成する Tofu インターコネクト [18] でノード間の結合を行うことで良好なスケーラビリティと高性能，高信頼性，高可用性を実現している．

3.1 「京」システム環境

「京」は合計ノード数 88,128 からなる大規模並列計算機である．1 つのノードは 8 プロセッサコア，キャッシュメモリおよびメモリコントローラを 1 チップに集積した富士通株式会社製の汎用スカラ型プロセッサ SPARC64 VIIIfx を 1 つ備えている．102 個のノードが 1 ラックに収容されており，計算システムは合計 864 のラックにより構成される．各ノードは Tofu インターコネクト [18] によって接続された直接網 6 次元メッシュトーラス（Tofu）を構成しており，片方向 5 GB/sec の高性能な通信や故障ノードを迂回した通信による高可用性が実現されている．また，各ノードはローカルディスクを持たず，Tofu インターコネクトを介した通信によりファイルシステムとの入出力を行う．1 ラックに収容される 102 ノードのうち利用者が計算に使用できるノード（計算ノード）はラック合計 96 個，システム合計で 82,944 個である．

また，ソフトウェアとしては「京」と PC クラスタ向けに統一した実行環境の提供を目標として OSS（Open Source Software）を最大限に活用しており，Linux ベースの OS を採用し，ファイルシステムには分散型クラスタファイルシステムである Lustre [19] をもととした Fujitsu Exabyte File System [20] を採用するなど，様々な OSS をもとにしたソフトウェアを利用している．「京」を構成する計算ノードのシステム環境を**表 2** に示す．

3.2 ファイルシステム

「京」のファイルシステムは数万ノードからの大規模なファイルアクセスに対応するため，計算ノードで実行中のジョブ専用の高速一時保存領域として利用するローカルファイルシステム（Local-FS）と，ユーザのファイルを保存する大容量の共用保存領域として使用するグローバルファイルシステム（Global-FS）の 2 つが存在する．両方とも，ファイルを分割しメタ情報データと実体データで分散管理することで負荷の分散や冗長性を確保した高信頼なクラスタ型ファイルシステム Lustre [19] をベースとした Fujitsu Exabyte File System [20] を採用している．

ユーザは計算ノードとは異なるハードウェア構成であるログインノードを利用して大容量の Global-FS に計算に必要なデータやプログラムを保存する．計算ノードを利用する際にはジョブ投入時に Global-FS から Local-FS に必要なファイル群を転送（ステージイン）し，ジョブ実行中には高速な Local-FS を利用してファイル I/O を行い，ジョブ終了後には Local-FS から計算結果のファイル群を Global-FS に転送（ステージアウト）することによって，計算ノードを利用した計算が可能である．この Global-FS と Local-FS 間のファイル転送をステージングと呼ぶ．

4. エクソーム解析パイプラインの「京」上での開発

本研究では汎用 PC クラスタ用エクソーム解析パイプライン Genomon-exome をモデルとして，「京」上で同等の処理を可能とする新たなエクソーム解析パイプラインの開発を行う．本章では「京」上のパイプライン開発にあたり障害となった 2 つの問題と本研究での対処方法を述べる．

4.1 問題 1：「京」のソフトウェア環境

モデルとした Genomon-exome は多数の OSS を利用するパイプラインであり，計算環境は個々のソフトウェアが要求する言語環境をすべて備える必要がある．しかし，「京」の計算ノードはすべてに対応する言語環境を備えていない．本研究で「京」上に開発するパイプラインに必要な言語環境と，対応する 2015 年 1 月時点の「京」計算ノードの言語環境を**表 3** に示す．

最も大きな問題は「京」の計算ノードに Java 言語の実行環境が存在しないことである．「京」は富士通株式会社の開発した SPARC アーキテクチャによるプロセッサを搭

表 2 「京」計算ノードのシステム環境
Table 2 Specification of a computing node of the K computer.

# nodes	82,944
CPU	SPARC64 VIIIfx [2.0 GHz] (8 cores)
Memory	DDR3 SDRAM 16 [GB]
Interconnect	Tofu (6D mesh/torus interconnect)
OS	Linux version 2.6.25.8
File system	Fujitsu Exabyte File System

表 3 パイプラインの必要言語環境と「京」計算ノードの言語環境（2015 年 1 月時点）
Table 3 Required programming language specifications and availability on the K computer (as of Jan. 2015).

必要な言語環境	「京」計算ノードの言語環境
C/C++	fcc ver. 1.2.0
Java	N/A
Python ver.2.6 以上	python ver. 2.6.2
Perl	perl ver. 5.10.0
R	R ver. 3.0.1[*1]

載しており，その Java 実行環境は現在公式に提供されていない．Genomon-exome は Java 言語を GATK によるリアライメント，Picard によるマッピング統計情報の出力および SAM（BAM）ファイルに対する各種操作で使用している．「京」ではログインノードとネットワークでつながれた Pre/Post 処理ノードと呼ばれる環境を利用して Java 言語の実行環境を得ることが可能であるが，計算ノードと比較して小規模な環境である Pre/Post 処理ノードを利用してすべての計算を行うことは現実的ではない．

本研究では，GATK によるリアライメント処理は計算時間に対して解析精度向上への寄与が小さく，Genomon-exome でも工程を省略する利用者が多いなどの理由により，「京」上に開発するパイプラインでの実装を見送った．一方，Picard によるマッピング結果の処理で利用される SAM（BAM）ファイルに対する機能は，C 言語で動作する SAMtools の提供する同等機能で処理を代替することで解決を図った．同じく Picard によるマッピング統計情報出力は，以降の解析工程に対する依存関係がなく独立しているため，計算ノードによる処理の終了後に Pre/Post 処理ノードによる実行を検討しており，今回は実装を見送った．

4.2 問題 2：ワークフロー管理

Genomon-exome では，ワークフロー全体を管理する 1 つのジョブがパイプラインの各処理をさらにジョブとしてシステムに投入して処理を進行する．ユーザが進行管理のジョブをシステムに投入すると，管理用のジョブがパイプラインの各ステップを分割してジョブとしてシステムに投入し，同時に投入されたすべてのジョブの終了を待って，次のジョブを投入する．この処理を繰り返して，ワークフローを進行する．しかし，このワークフロー管理の方法を「京」システム上で開発する場合，以下の問題が発生する．

4.2.1 ジョブ投入数の増大による転送時間の増加

Genomon-exome のジョブ投入数の具体例として，マッピング処理を 1 サンプルに対して行った際の並列度（ジョブ投入数）を**表 4** に示す．合計 T 個のサンプル（Type）の処理における，ペアエンド配列を入力としたマッピング処理を仮定し，i 番目のサンプルを t_i，i 番目のサンプルの FASTQ ファイルに対する分割数を S_{t_i} とする．

多くの場合に解析対象のサンプル数 T は小さく，正常細胞と腫瘍細胞のサンプル 1 組を比較する場合などは $T = 2$ となる．S_{t_i} は入力ファイルが分割される数を示し，並列度に直接的に影響を及ぼす要素である．分割数 S_{t_i} が増加すると入力ファイルがより細かく分割されて並列度が向上するが，従来方式では並列度の増加とともにジョブ投入数が増大するという問題がある．

*1 本研究の開始当初は R 環境が存在しなかったが，2013 年末に HPCI 運用事務局ヘルプデスクにより実行環境が提供された．R の実行ファイルをステージインすることで利用が可能である．

表 4　マッピング処理の各ジョブの並列度
Table 4　Degree of parallelism of each job in the mapping process.

ジョブ名	並列度
map_bwa.sh	1
split.sh	$2T$
bwa_aln.sh	$2\sum_T S_{t_i}$
bwa_sampe.sh	$\sum_T S_{t_i}$
sam_join.sh	T
sam2orgbam.sh	T
checkLineSizeAlignmentBam.sh	T
my_mergeAlignmentBam.sh	T
bam2summary.sh	T

「京」は異なる 2 つのファイルシステムを備えており，ジョブ投入のたびにステージング処理によるファイル転送コストがかかる．また，ジョブ投入ごとに計算資源の割当てを待つ時間も存在するため，ジョブ投入数の増大は「京」では大きな問題となる．本研究では多数のジョブを投入するかわりに MPI を利用した Master-Worker モデルでタスク分散を行うことで，問題の解決を試みる．

4.2.2 同時実行ジョブの同期処理の非効率性

同時投入された全ジョブの終了を待つ同期待ちも問題である．「京」上に実装するパイプラインの機能のうち，前述した BWA のマッピング処理の入力は大量の断片配列の情報が記述された FASTQ 形式のファイルであり，一定の配列ごとに分割してノードに配分することで処理を分散可能である．しかしながら，モデルとなる Genomon-exome では配分されたノードごとに実行時間の差が生じるにもかかわらず，最も実行時間の長いジョブの終了を他の全ジョブが待機して次の処理に移行する待ち時間が発生している．

この問題を解決するために，全ジョブの実行時間を可能な限り均等に配分して同期待ち時間を減らす，または依存する実行単位ごとに処理を分けて同期回数自体を削減するなどのアプローチが有効である．本研究では後者のアプローチを採用してタスク分散フレームワーク MPIDP [21] に依存関係解決処理を導入し，同期処理の非効率性の問題の解決を試みる．

5. MPIDP を用いた Master-Worker モデルによるタスク分散

前節で述べた問題 2（ワークフロー管理の問題）を解決するため，本研究ではパイプラインに MPIDP を拡張して導入した．MPIDP は 2012 年に我々のグループで開発された MPI ライブラリを用いた Master-Worker モデルによるタスク並列分散処理フレームワークである [21]．MPIDP は，MPI におけるランク 0 のプロセスをタスク管理を担う Master に選び，残りのプロセスをタスク実行を担う Worker とする．Master はタスクが記述されたリストを読み込み，

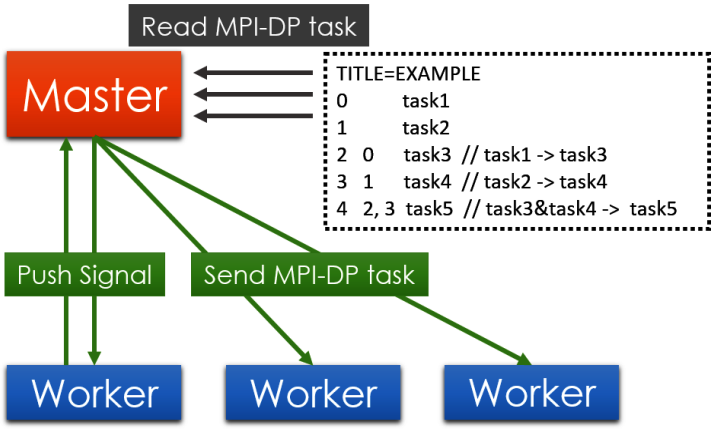


図 4　MPIDP によるタスク分散の仕組み
Fig. 4　Task distribution by MPIDP.

Point-to-Point 通信により Worker にタスクを振り分ける．Worker はタスクの実行終了後に Master へ通知を行い，通知を受け取った Master は次のタスクを Worker に与える．以上を繰り返し実行し，タスクを動的に割り当てた並列分散処理を可能とする（図 4）．このとき，プロセス間のデータの受け渡しはファイルを経由して行われる．

MPIDP は並列計算機環境で多数のクエリファイルに対する高速な相同性検索を行う GHOST-MP [22], [23] にも使用されており，「京」における動作実績がある．本研究ではパイプライン処理の並列分散に MPIDP を利用し，ジョブシステムを介さずに処理を分散して転送時間増大の原因であるジョブ投入数を抑え，さらに MPIDP を拡張して実行単位ごとに依存関係を解決する機能を実装することで，同期コストを軽減した並列実行が可能なパイプラインを開発した．

5.1　MPIDP への依存関係解決処理の導入

MPIDP は MPI によるシンプルな Master-Worker モデルのタスク分散の枠組みであり，Master がリストの各行に記述されたタスクを上から順に待機中の Worker に振り分ける．しかし，遺伝子解析パイプラインでは，ある処理の出力を別の処理が入力として利用する依存関係が存在する．本研究では新たに，Master がタスクの振り分けを行う際，依存関係のあるタスクの終了を確認してから Worker に振り分けを行う，タスク間の依存関係を解決する処理を導入した．

5.2　MPIDP 導入後のタスクフロー

拡張された MPIDP を用いて「京」上で開発したエクソーム解析パイプラインのタスクフローを，マッピング処理を例として図 5 に示す．解析対象の入力データ（FASTQ ファイル），必要なプログラム群（BWA など）はログインノードのデータ領域（Global-FS）に保存されていると

する．

i. ユーザによるジョブの投入
ユーザがログインノード上でパラメータを指定してスクリプトを実行すると，マッピング処理に必要なタスクリスト，ステージング処理情報，占有する計算資源の記述を含むジョブ投入用スクリプトが自動生成される．ユーザは出力されたスクリプトをそのままジョブ管理システムに投入する．

ii. 計算ノードの確保とステージイン
ジョブ管理システムはスクリプトの記述をもとに計算資源を確保し，ステージング対象のファイルを Global-FS から Local-FS へ転送する（ステージイン）．

iii. ジョブの実行と MPIDP による並列処理
ステージインの終了後，確保された計算ノード上では MPIDP が Master を 1 つの計算ノードに割り当て，残りの計算ノードには 1 ノードあたり 1 プロセスずつ Worker を割り当てる．Master はタスクリストをもとに Worker にタスクを分配して，パイプラインの各処理が並列に処理される．

iv. ステージアウトによる出力
リストに記述されたすべてのタスクの実行が終了すると，最終的なマッピング結果を含むファイル群を Local-FS から Global-FS へ転送する（ステージアウト）．

拡張された MPIDP の導入により，異なるパラメータのタスクの動的な分散処理を実現し，タスク間の依存関係を解決することでタスクの処理順序を考慮した実行を実現した．また，単一のジョブ内で複数の MPI プロセスとして一連の処理を実行し，合計のジョブ投入回数を初めの 1 回のみとしたことで，ジョブ投入数の増大による実行待ち時間や転送時間の増加を抑えている．

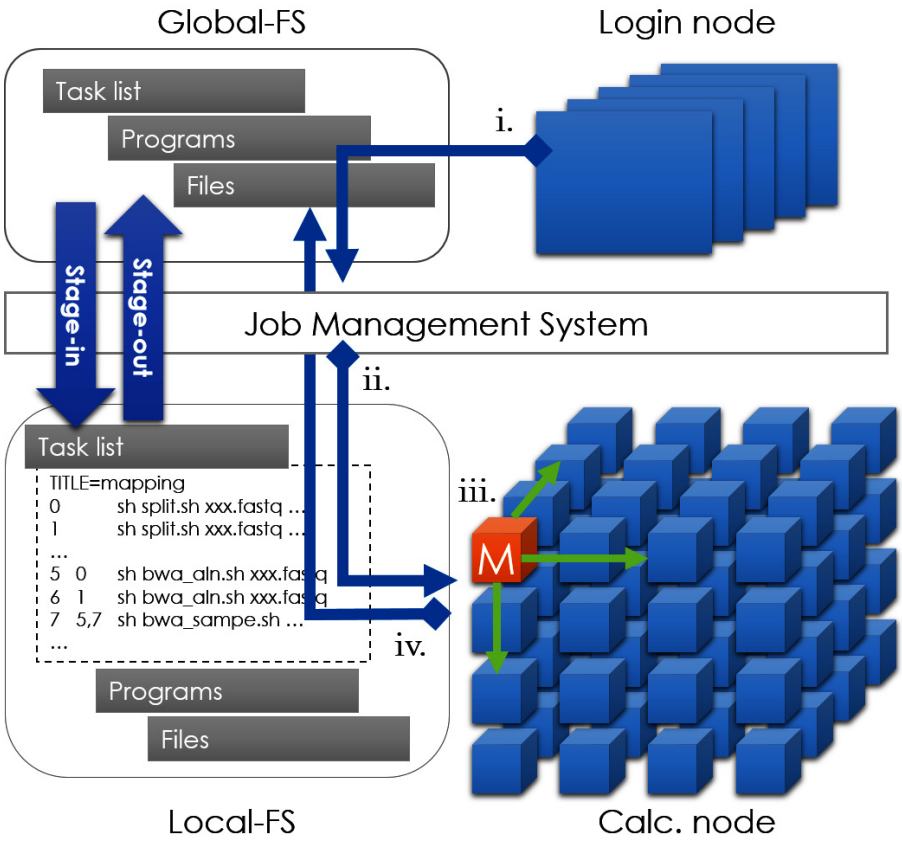


図 5　スーパーコンピュータ「京」における MPIDP を利用したエクソーム解析パイプラインのタスク分散処理の流れ

Fig. 5　Task distribution flow of the exome analysis pipeline on the K computer using MPIDP.

6. マッピング結果処理の並列化による高速化の提案

以上で「京」への実装に特有の問題とその対策について述べた．本章では，前章で実装したパイプラインの実行結果を検討し，パイプラインの各処理の見直しによって，さらに並列実行性能を改善するための提案を行う．

6.1 マッピング結果処理の並列化における問題

本研究で「京」上で開発したパイプラインを用いて肺腺がんのエクソームデータ（**表 5**）の参照配列へのマッピング処理を行う際，実行されるタスク（スクリプト）ごとに，各ノードにおける実行時間を合算した数値を**表 6** に示す．今回は実行時間の関係から計測に「京」計算ノード 16 台を用いたため，表 6 の TOTAL は 16 ノードすべての実行時間を合算した値となる．そのため，特に BWA によるアラインメント処理ステップにおいては参照配列の読み取りなどのファイルの読み書きにともなう I/O のオーバヘッドが重複して含まれてしまっている．そこで，同エクソームデータを 1/100 サイズにランダムサンプリングしたものについて，1 ノードで処理した際の実行時間と 16 ノー

表 5　肺腺がんのエクソームデータ詳細

Table 5　Exome data of lung adenocarcinoma.

Run	ERR160121	ERR166339
Type	normal	tumor
Length	100 [bp]	100 [bp]
Paired	yes	yes
Platform	Illumina HiSeq 2000	Illumina HiSeq 2000
Total Size	13,324 [MB]×2	23,726 [MB]×2

表 6　各処理で実行されるタスクごとの合計実行時間

Table 6　Total execution time of each script.

Task Name	Elapsed Time [sec]	ratio
split.sh	683	0.005
bwa_aln.sh	59,876	0.448
bwa_sampe.sh	29,555	0.221
sam_join.sh	1,920	0.014
sam2orgbam.sh	39,419	0.295
checkLineSizeAlignmentBam.sh	3,081	0.023
my_mergeAlignmentBam.sh	125	0.001
bam2summary.sh	-	-
TOTAL	127,094	1.00

ドで処理した際の実行時間を計測し，I/O のオーバヘッドおよび並列処理による実行効率の低下を推定した．その結果，分割数 $S = 128$ のとき，1 ノードでの実行時間は 6,509 秒，16 ノードでの実行時間は 474 秒であったため，$(474 \times 16) - 6509 = 1075$ 秒が I/O のオーバヘッドとして過剰に推定されていると考えられる．そのため，BWA によるアラインメント処理の合計実行時間は 60,951 秒であったが，表 6 ではその分の補正を行っている．これにより，対象のエクソームデータのマッピングを 1 ノードで実行した場合の合計時間を見積もることができる．

タスクごとに全ノードの実行時間の合算を見ると，マッピング処理において最もコストが高い処理は `bwa_aln.sh`（44.8%），次いで `sam2orgbam.sh`（29.5%），`bwa_sampe.sh`（22.1%）となっている．各処理に対する従来の手法の並列度（表 4）を確認すると，全体実行時間の約 3 割を占めるマッピング結果の SAM 形式ファイルの処理は，入力されたサンプル数（今回は $T = 2$）と等しくなっている．このため，他の部分の並列数を増加させた場合，マッピング結果ファイルに対する処理の並列性の低さが全体のボトルネックとなり，全体の実行時間の短縮が困難となるという問題がある．そこで本研究ではマッピング結果の SAM 形式ファイルの処理を変更し，並列実行性能の改善を試みる．

また，このマッピング処理後に行う変異検出や遺伝子への関連付けの工程については今回の実験で使用したデータの場合，Pre/Post 処理ノードの 1 ノードを用いて 147,629 秒の計算時間を必要とした．計算環境が異なるため直接比較できるものではなく参考程度ではあるが，「京」の計算ノードにおけるマッピング処理とほぼ同じ計算時間となっており，本来はこのマッピング後処理の工程についてもマッピング処理と同様に計算ノードを利用した並列処理の実装を検討する必要がある．しかし，この後処理部分は応用によって変更される可能性もあり，また前述のとおり処理する DNA 断片の数の増加にともない相対的に計算量が少ないものとなるため，今回は変異検出などの後処理部分の並列化は見送り，マッピング処理部分の高速化のみを行った．

6.1.1　SAM/BAM ファイル形式

パイプラインのボトルネックであるマッピング結果処理は，SAM（Sequence Alignment/Map format）形式ファイルの処理を中心としている．SAM 形式はマッピング結果の格納に用いる形式であり，リードの参照配列へのアライメントの状態などの情報がテキスト形式で記述される．生命情報解析で広く普及している BWA や Bowtie などのソフトウェアにより出力され，これをもとにサンプルの変異の解析などが行われる．一方，BAM（Binary-sequence Alignment/Map）形式は，SAM 形式ファイルを容量を削減する目的でバイナリ圧縮した形式である．圧縮には BGZF（Blocked GNU Zip Format）が用いられており，SAM 形式と BAM 形式で有するマッピング情報は等価であることから各種ツールにより情報を損失せず相互変換が可能である．BAM 形式はファイル内のリードが開始位置で整列済みであることを前提として索引を作成し，これを用いた特定領域への高速なアクセスが可能である．

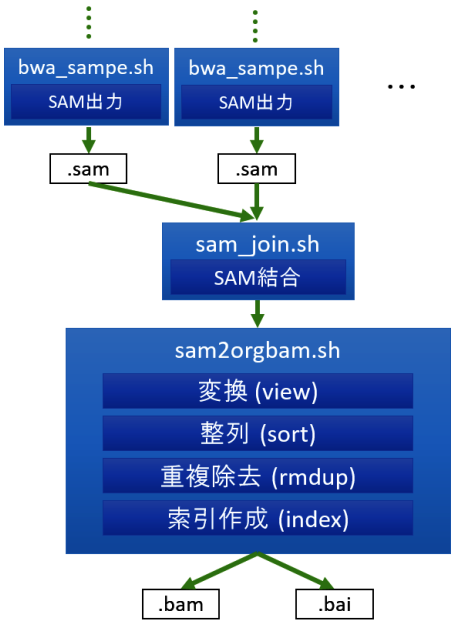


図 6　従来の Genomon-exome のマッピング結果処理の流れ
Fig. 6　Flow of the post-mapping process in original Genomon-exome.

6.1.2　マッピング結果処理の概要

モデルとした Genomon-exome におけるマッピング結果に対する一連の処理は次のとおりである（**図 6**）．入力としては `bwa_sampe.sh` によりマッピング結果の SAM 形式ファイルが複数与えられる．はじめに，`sam_join.sh` によって分割された SAM ファイルがサンプルごとに 1 つに統合される．次に，`sam2orgbam.sh` によって SAM 形式ファイルをバイナリに圧縮した BAM 形式へ変換してデータサイズを削減（変換）し，記述されたリードを参照配列の開始位置でソート（整列）する．そして，同じ位置で重複する冗長なリードを除去（重複除去）して，BAM 形式ファイルの索引を作成（索引作成）する．

マッピング結果処理は分割された SAM 形式ファイルを入力として，整列済みかつ重複除去済みの 1 つの BAM 形式ファイル，その索引を出力する処理である．本研究ではマッピング結果処理の並列度向上を含む，複数のアプローチを試みることでマッピング処理全体の並列実行の効率化を図り，処理の高速化を目指す．

6.2　高速化のためのアプローチ

マッピング結果処理の高速化のために本研究で行った以下の 4 つのアプローチに関して述べる．

A）BAM 形式ファイルの圧縮レベルの変更

図 7　高速化のための各アプローチの内部処理，並列実行の概略

Fig. 7　Schematic view of internal steps and their parallelization in each proposed approach.

表 7　SAMtools による SAM-BAM 変換の実行時間と出力ファイルサイズ

Table 7　Execution time and output file size in SAM-BAM conversion by SAMtools.

	time [sec]	deflate [sec]	size [MB]	ratio
SAM	-	-	2,573	1.00
BAM (Lv.6)	248.75	131.86	264	0.10
BAM (Lv.0)	126	36.60	2,035	0.79

B)　最大並列数を考慮した統合タイミングの変更

C)　マージソートの導入

D)　染色体ごとの領域分割による並列化

A は BAM 形式ファイルに操作を行うすべての処理に影響を及ぼす変更である．一方，B，C，D は並列処理部分の改善の提案である（図 7）．

6.2.1　A：BAM 形式ファイルの圧縮レベルの変更

BAM 形式は BGZF（Blocked GNU Zip Format）と呼ばれる GZIP 形式 [24] の標準のブロック圧縮方式で SAM 形式を圧縮したものである．BAM 形式ファイルの容量は元の SAM 形式ファイルの容量と圧縮時に設定した圧縮レベル（COMPESSION_LEVEL）に依存し，圧縮レベルが高いほど圧縮や解凍の時間は増大する．BAM 形式は SAMtools や Picard などのソフトウェアを用いて変換や整列などの操作が可能であるが，内部では操作のたびに BGZF 形式の解凍と圧縮処理が行われており，この処理が律速となる場合も多い．このため，圧縮レベルを調整して実行時間を削減することが有効である．

BAM 形式で用いられている BGZF の圧縮レベルは 0 から 9 までの段階が有効であり，0 は無圧縮，9 は最も高い圧縮率となる．多くはデフォルト値として 6 が設定されている．例として SAMtools を用いた SAM 形式から BAM 形式への変換時間，圧縮の処理時間，出力ファイルサイズ，圧縮率を表 7 に示す．本研究では処理の高速化の観点から，連続する BAM 形式ファイル処理では圧縮レベルを低く設定して圧縮コストを削減し，最終段階で出力する処理のみ圧縮レベルを高めて設定することで，出力ファイルサイズを抑えつつ処理全体の高速化を図る．

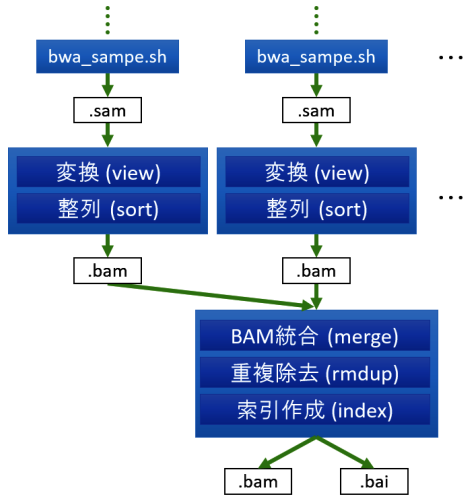


図 8　統合タイミングを変更したマッピング結果処理の内部処理の流れ

Fig. 8　Post-mapping process flow with delayed merging scheme.

6.2.2　B：最大並列数を考慮した統合タイミングの変更

次に，マッピング結果の処理方式改善の提案を行う．マッピング結果処理の大きな問題は初めに行われる分割された SAM 形式ファイルの統合であり，これにより以降の処理の並列数が大幅に減少している．よって，1 つ目のシンプルな改善方法として統合タイミングの変更が考えられる．幸い，変換は各 SAM 形式ファイルごとに独立した実行が可能であり，BAM 形式ファイルの整列は SAMtools を用いて整列済みの BAM 形式ファイルを 1 つに統合可能である．

一方，重複除去は重複リードが別のファイルに存在する可能性がある場合の個別処理は意味をなさないため統合後の処理が必要になり，変換や整列と同じ時点で処理はできない．索引作成も同様に統合後に処理する必要がある．これらをもとに，最大並列数を維持するために統合タイミングを変更した方式の処理の流れを図 8 に示す．

6.2.3　C：統合処理のマージソートの導入

上述の BAM 形式ファイルの統合処理には並列マージ

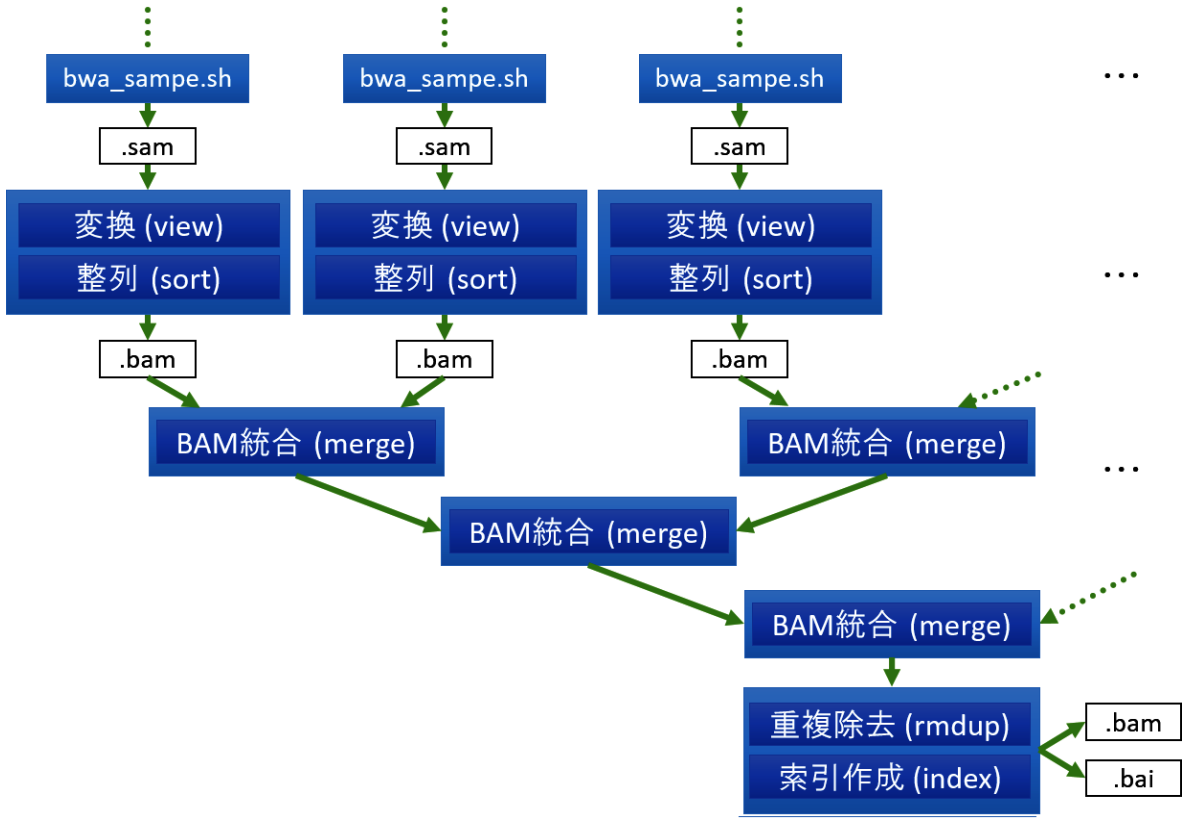


図 9　マージソートによるマッピング結果処理の内部処理の流れ
Fig. 9　Post-mapping process flow with merge sort.

ソートを導入することが可能である．統合処理ですべての BAM 形式ファイルを一度に統合する方式を採用していたが，たとえば 2 つずつファイルを統合する 2-way merge sort による並列ソートが可能である．統合処理は分割された SAM 形式のファイルの総数を N としたとき，$O(\log_2 N)$ 段のマージで完了する．統合における整列処理が処理の律速であるならば，統合処理をマージソートで並列化することで処理の高速化が期待できる．処理の流れを図 **9** に示す．

6.2.4　D：染色体ごとの領域分割による並列化

これまでのアプローチでは変換と整列の処理が並列化可能であるが，BAM 形式ファイルに含まれる重複リードの除去を並列化できない．これは分割された SAM（BAM）形式ファイル間では互いに重複するリードの存在を知ることができず，複数ファイルにわたった重複リードを除去できないためである．そこで，アライメント先の染色体情報を用いて事前にファイルを分割することで重複除去の処理の並列化を試みる．別の染色体にアライメントされたリードは互いに重複することはないため，入力の SAM 形式ファイル内に記述される染色体の情報を用いてファイルを事前に分割し，処理の流れを染色体ごとに分けることで重複除去の処理を含めた並列処理が可能となる．この処理の流れを図 **10** に示す．

まず，入力である分割された SAM 形式ファイルに記述された染色体情報をもとに，染色体ごとに別の SAM 形式ファイルに分割する．次に各染色体ごとに SAM 形式ファイルをまとめて（結合），BAM 形式への変換，リード開始位置で整列，重複除去を行う．最後に染色体ごとのすべての SAM 形式ファイルを 1 つに統合する．

また，染色体ごとに分割する手法には重複除去の並列化に加えて，最終的な統合時に単純な BAM 形式ファイルの結合が利用可能であるという利点がある．これは染色体ごとにすでに整列済みである場合には全体での整列処理が不要となるためである．

7.　評価実験結果と考察

「京」上に開発したエクソーム解析パイプラインの性能評価を行う．実験 1 では参照配列へのマッピング処理を対象に MPIDP を用いて開発した「京」上のパイプラインによる実行時間と，開発のモデルとした Genomon-exome の東京大学医科学研究所ヒトゲノム解析センター PC クラスタ Shirokane1 上での実行時間の比較を行う．実験 2 では同じくマッピング処理を対象にマッピング結果の SAM 形式ファイルの処理に並列実行性能の改善を適用し，並列実行時間の改善の評価を行う．

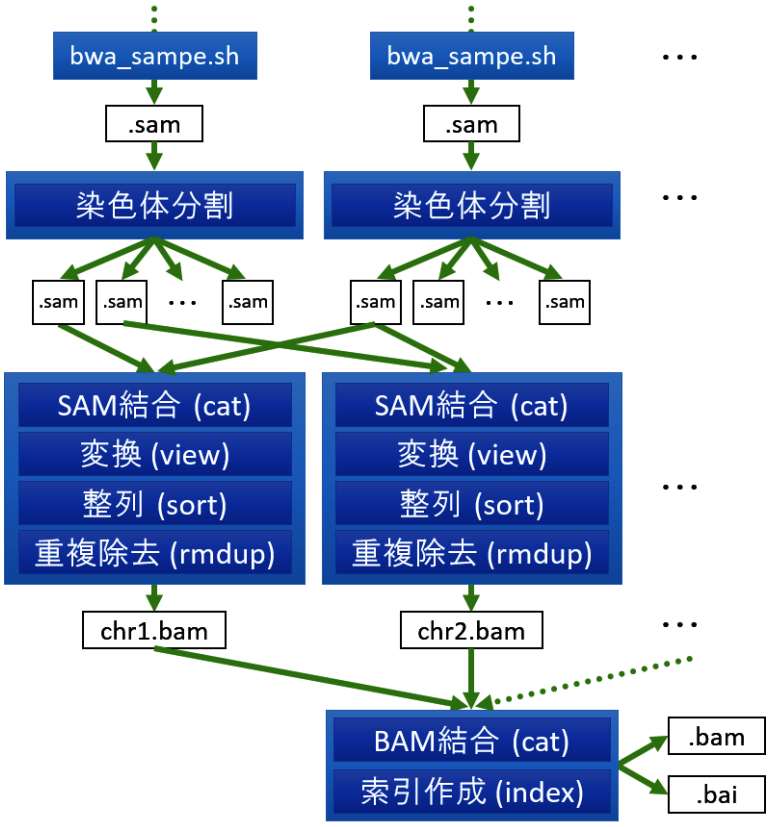


図 10　染色体ごとの分割によるマッピング結果処理の内部処理の流れ
Fig. 10　Post-mapping process flow with chromosome-based decomposition.

7.1　実験条件

7.1.1　対象データ

実験で使用したエクソームデータを表 5 に示す．これは肺腺がんの研究プロジェクトのエクソームデータセット [25] から任意に正常細胞（normal）と腫瘍細胞（tumor）を選んだものである．今回の解析対象のデータのサンプル数 T は，データが normal と tumor の 2 つのサンプルからなるため $T = 2$ となる．

7.1.2　計算機環境，ソフトウェア環境

実験に使用した計算機環境を**表 8** に示す．実行時間の計測では，全体実行時間などジョブ単位で計測が行えるものはシステムが示すジョブの実行時間を用い，MPIDP によるタスクごとの時間計測には gettimeofday 関数を用いた．

Shirokane1 はジョブ管理システムとして Univa Grid Engine を採用しており，オリジナルの Genomon-exome のパイプラインではタスクの割当てはバッチキューシステムによって行われるため，厳密な処理時間の計測は困難である．よって実験時には他のジョブが実行されていないことを確認後，使用ノード数を指定してジョブ投入を行い，実行中専有状態であることを qstat コマンドによって確認しながら計測を行った．

対象データは各サンプルごとの FASTQ ファイルのサイズが異なるため，FASTQ ファイルの分割数 S_{t_i} も異なる．今回はペアエンド配列であるため全サンプル分割数の総和を 2 倍した値 $2\sum_T S_{t_i}$ が利用するノード数と一致する値を設定した．たとえば，ノード数 $N = 128$ の場合は $\{S_{normal}, S_{tumor}\} = \{23, 41\}$ として，$N = 256$ の場合は $\{S_{normal}, S_{tumor}\} = \{46, 82\}$ のように設定される．これは Shirokane1 環境のバッチキューシステムを利用するオリジナルの Genomon-exome において，ノード数を指定して実行するために FASTQ ファイルの分割数を限定する必

表 8　実験環境：スーパーコンピュータ「京」，ヒトゲノム研究センター Shirokane1（2014 年 12 月時点）
Table 8　Specification: the K computer, and Shirokane1 (as of Dec. 2014).

	「京」コンピュータ	Shirokane1
# nodes	82944	760
CPU	SPARC64 VIIIfx [2.0 GHz] (8 cores)	Intel Quad Core Xeon [3.0 GHz] (8 cores)×2
Memory	16 [GB]	32 [GB]
OS	Linux version 2.6.32	CentOS (Linux version 2.6.32)
C/C++	fcc ver. 1.2.0	gcc4.6.2
Python	python ver.2.6.2	python ver.2.6.5
Java	N/A	java version.1.6.0
Perl	perl ver. 5.10	perl ver.5.12.1
MPI	FUJITSU MPI Library (OpenMPI ver. 1.4.3)	OpenMPI ver.1.5.4
JMS	Parallel Job Manager	Univa Grid Engine

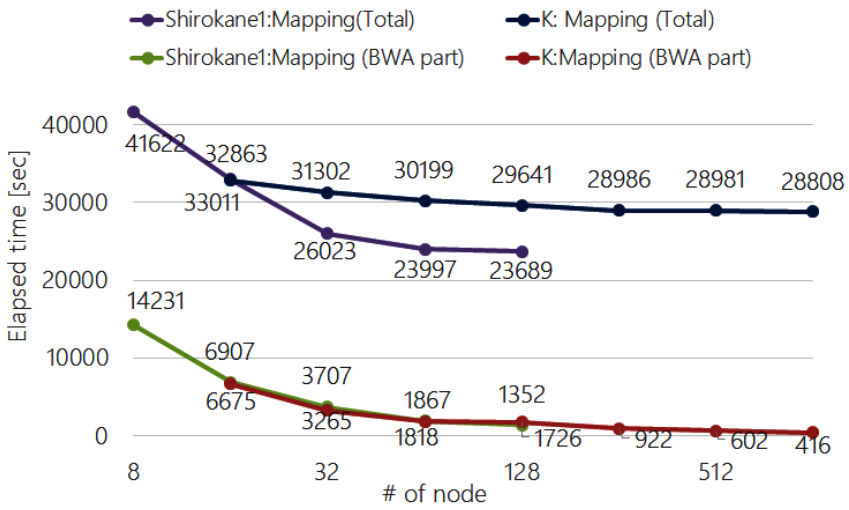


図 11　マッピング処理の実行時間の比較
Fig. 11　Comparison of the execution time of the mapping process.

要があるためである．ただし，「京」では前述の制限はないため，ノード数 $N = 128$ 以下の実行時は $2\sum_T S_{t_i} = 128$ となるように分割数 S_{t_i} の設定を行った．

パイプラインで実行されるプロセス内のスレッド実行は，BWA によるアライメントの処理（`bwa_aln.sh`）でのみスレッド数を 8 と指定して実行した．その他の処理はシングルスレッドで実行されている．

また，実験で使用した肺腺がんのエクソームデータに対して，本研究で開発したエクソーム解析パイプラインによる計算結果を，Shirokane1 環境の Genomon-exome による計算結果と比較し，それらが一致することを確認した．

7.2　実験結果

7.2.1　実験 1：Genomon-exome と並列実行性能比較

ヒトゲノム解析センターの Shirokane1 における Genomon-exome と，本研究で MPIDP を用いてスーパーコンピュータ「京」上で開発したエクソーム解析パイプラインのマッピング処理に関する並列実行性能の比較を行う．Shirokane1 と「京」における実行時間を，マッピングの一連の処理の全体実行時間を `Mapping (Total)`，BWA によるアライメント部分のみの実行時間を `Mapping (BWA part)` として**図 11** に示す．表 6 の結果とはノードごとの実行時間を合算していない点で算出方法が異なる．また，Shirokane1 におけるノード数 $N = 8$ を基準とした高速化率を**図 12** に，「京」におけるノード数 $N = 16$ を基準とした高速化率を**図 13** に示す．

なお，「京」側の環境では Java 言語環境の問題からマッピング統計情報を出力する `bam2summary.sh` の実行時間は含まれていない．また，BWA によるアライメント部分のみの実行時間は初めの `bwa_aln.sh` の実行からすべての `bwa_aln.sh` の実行が終了するまでとした．

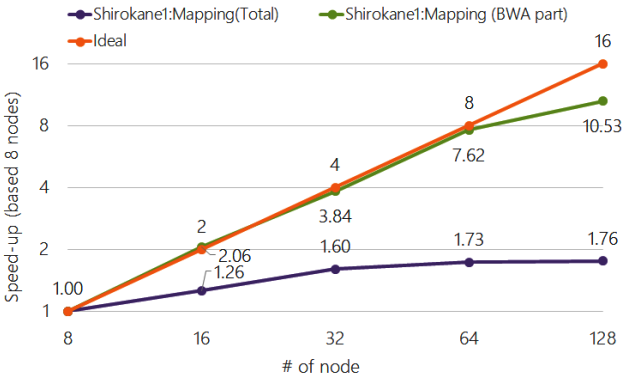


図 12　ノード数変化による高速化率（Shirokane1）
Fig. 12　Scalability of the mapping process on Shirokane1.

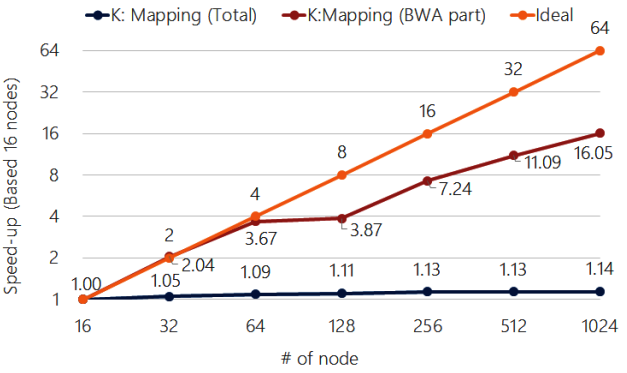


図 13　ノード数変化による高速化率（「京」コンピュータ）
Fig. 13　Scalability of the mapping process on the K computer.

マッピング処理全体では，Shirokane1 の Genomon-exome はノード数 $N = 8$ に対して $N = 128$ のとき最大で 1.76 倍の高速化，「京」上で開発したパイプラインは $N = 16$ に対して $N = 256$ で最大 1.13 倍の高速化と，両方の環境ともノード数を増加させても大きな全体実行時間の短縮には

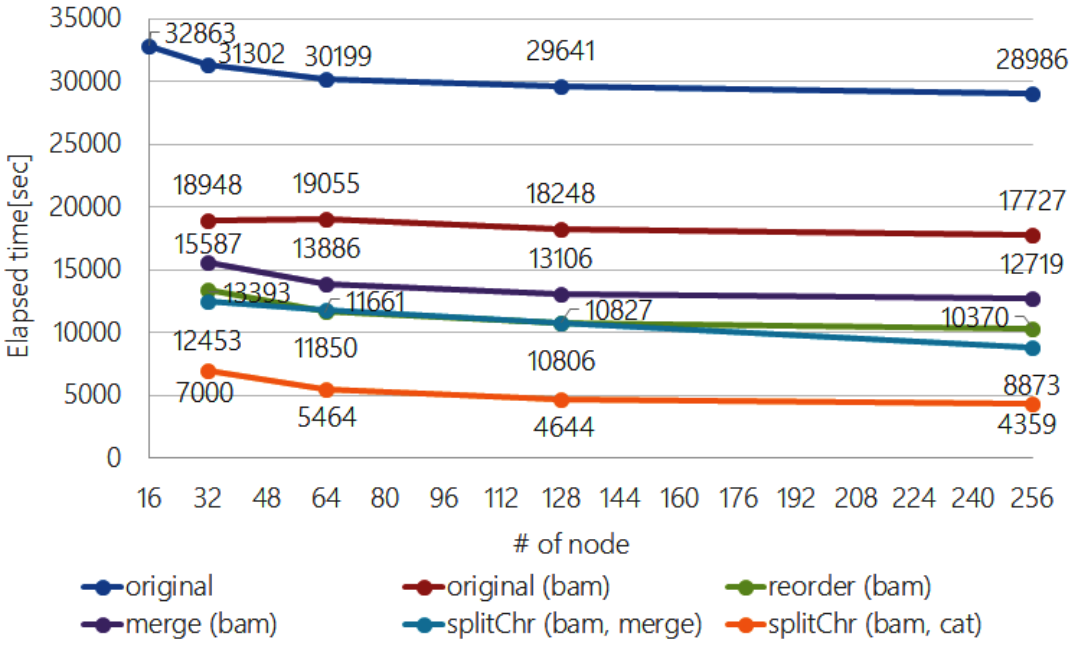


図 14 「京」上のパイプラインに対する高速化アプローチの適用

Fig. 14 Comparison of the execution time of each speed-up approach for the exome analysis pipeline on the K computer.

つながっていない．一方で，BWA によるアライメント部分のみに着目した場合には Shirokane1 の Genomon-exome では $N = 8$ に対して $N = 128$ で最大 10.5 倍の高速化，「京」上のパイプラインでは $N = 16$ に対して $N = 1024$ で最大 16.1 倍の高速化の効果が得られている．

本研究では BWA によるアライメント以外の処理の並列数が低いことがマッピング処理の全体実行時間がノード数増加に応じて短縮されないという問題の原因であると考えた．よって，実験 2 では本研究で提案した処理方式の変更を適用後のマッピング処理の実行時間を示す．

7.2.2 実験 2：マッピング結果処理の改善による並列実行性能の変化

「京」上に開発したエクソーム解析パイプラインのマッピング処理に対して提案したアプローチを適用し，変更後のマッピング処理の全体実行時間を計測した（図 14）．

`original` は実験 1 で示した「京」上のパイプラインの実行時間を示し，`reorder` は最大並列数を考慮した統合タイミングの変更を適用，`merge` はマージソートを導入，`splitChr` は染色体ごとの領域分割を適用したものであり，`cat` と `merge` は染色体ごとの BAM 形式ファイルの統合方法の違いを示す．また，`(bam)` は中間処理では BAM 形式を無圧縮（Lv.0）で出力し，最終出力時のみ圧縮（Lv.6）して出力した場合を示す．

まず，BAM 形式の圧縮レベルの調整によって `original` から `original (bam)` で $N = 32$ のとき最大 1.7 倍の高速化が得られており，中間出力における圧縮レベルの調整が有効であると示された．`original` 以外の手法にはすべて圧縮レベルの調整を導入している．

最も実行時間が短いのは染色体ごとに処理を分割し，染色体ごとの処理の出力である BAM 形式ファイルの統合処理で単純な結合が可能である SAMtools cat を使用した `splitChr (bam, cat)` の手法であり，$N = 256$ のとき最大で 6.7 倍の高速化が得られた．特にサンプル数×染色体数（$T \times 24$）まではその効果が大きい．次いで，同様の処理の統合部分で SAMtools merge を利用した `splitChr (bam, merge)` と，最大並列数を維持して変換と整列を行う `reorder (bam)` が続き，さらにマージソートを導入した `merge (bam)` である．いずれも `original (bam)` より実行時間が短く，提案した手法による高速化の効果が見られた．

しかし，一方でノード数変化による高速化の効果は最大でも `splitChr (bam, cat)` の $N = 32$ に対する $N = 256$ の 1.6 倍であり，ノード数増加による実行時間の大幅な短縮は確認できなかった．

7.3 考察

7.3.1 マッピング処理の並列実行性能

実験により提案手法によるマッピング処理の高速化を確認できた．しかし，ノード数変化による高速化の効果は提案手法の適用後も低く，良好な並列実行性能は得られていない．この原因を考えるため，本研究で提案したマッピング後の SAM 形式ファイルの処理方式に関する高速化手法について，`bwa_aln.sh` の並列数を N とした場合に，1 サンプルあたりの同時実行可能な最大並列数を表 9 に示

表 9　SAM 形式ファイルの内部処理における同時実行可能な最大並列数（1 サンプル）
Table 9　Maximum degree of parallelism of each process in post-mapping process (for 1 sample).

処理内容	original	reorder	merge (2-way)	splitChr
SAM 結合 (cat)	1			
変換 (view)	1	N	N	24 (23)
整列 (sort)	1	N	N	24 (23)
重複除去 (rmdup)	1	1	1	24 (23)
BAM 統合 (merge)		1	$\frac{N}{2^i}$ ($\log_2 N$ stages)	0 (1)
インデックス作成 (index)		1		1
BAM 結合 (cat)				1 (0)

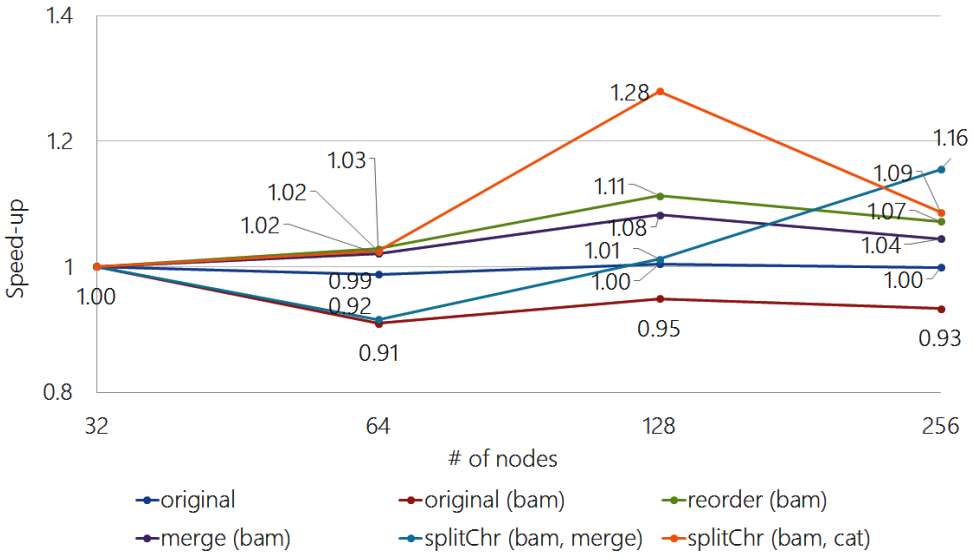


図 15　各手法における BWA によるアライメント（`bwa_aln.sh`, `bwa_sampe.sh`）以外の処理のスケーラビリティ
Fig. 15　Comparison of the scalability of each speed-up approach without BWA alignment part (`bwa_aln.sh` and `bwa_sampe.sh`).

す．処理内容列の括弧内は SAMtools のコマンド名を示している．

`reorder` と `merge (2-way)` は変換，整列を並列数 N で実行可能だが重複除去を並列実行できない．一方で，`splitChr` は染色体ごとに分割して処理を行うため並列数は染色体の本数に依存するが，重複除去も同様の並列数で実行可能である．

実験では `splitChr (bam, cat)` が最も良い性能を発揮していることから，同時実行可能な並列数が染色体数で上限を与えられた場合においても重複除去の並列化の効果が大きく現れたと推察できる．また，結合対象のファイルに対してマージソートによる統合を行わず，染色体ごとの BAM 形式ファイルの単純結合によって逐次的な実行時間を削減したことも要因であると考えられる．

しかし，最も良い高速化が見られた `splitChr (bam, cat)` でも実行ノード数増加による効果は残念ながら小さなものであった．図 13 で示すように，アライメント処理 `bwa_aln.sh` の実行ノード数の増加による実行時間の短縮は $N = 64$ から鈍化しており，それが大きな原因の 1 つであるが，同時にアライメント処理以外についても実行ノード数増加が高速化に寄与していないことが原因となっている．**図 15** は BWA によるアライメント以外の処理のスケーラビリティを示しているが，32 ノード以上については実行ノード数が増加しても，実行速度の変化は誤差の範囲内となっている．これは表 9 に示すように，アライメント処理以外の SAM 形式ファイル処理の最大並列処理数が `splitChr` でも 1 サンプルあたり 24 となっており，それ以上の実行ノードの増加が無意味なためである．$N = 24$ より少ないノード数の結果と比較することで並列処理による高速化の効果を確認できると予想されるが，現在 $N = 16$ 以下のノード数で実験データのマッピング処理を行った場合に `bwa_sampe.sh` の実行時に segmentation fault が発生してしまうという問題があり，計測ができなかった．エラーの原因究明と修正は今後の課題である．

7.3.2　ノード間の負荷不均衡

並列実行性能の低下を招く原因としてタスク間の実行

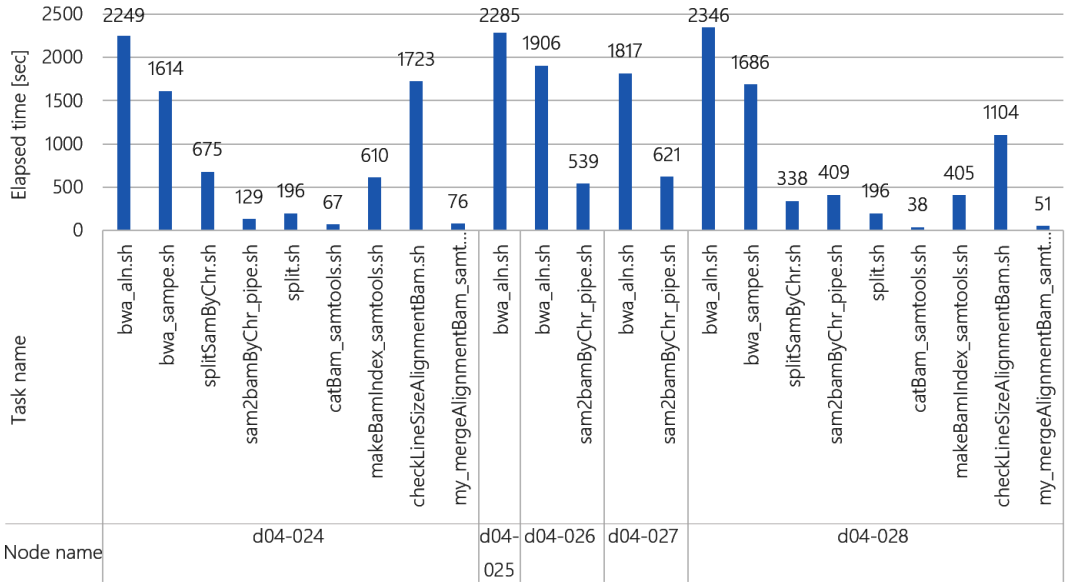


図 16　各ノードのタスク実行時間（実験 2 `splitChr(bam, cat)`（$N = 32$）の一部）

Fig. 16　Execution time of tasks performed at each node (five nodes shown from `splitChr(bam, cat)`, $N = 32$).

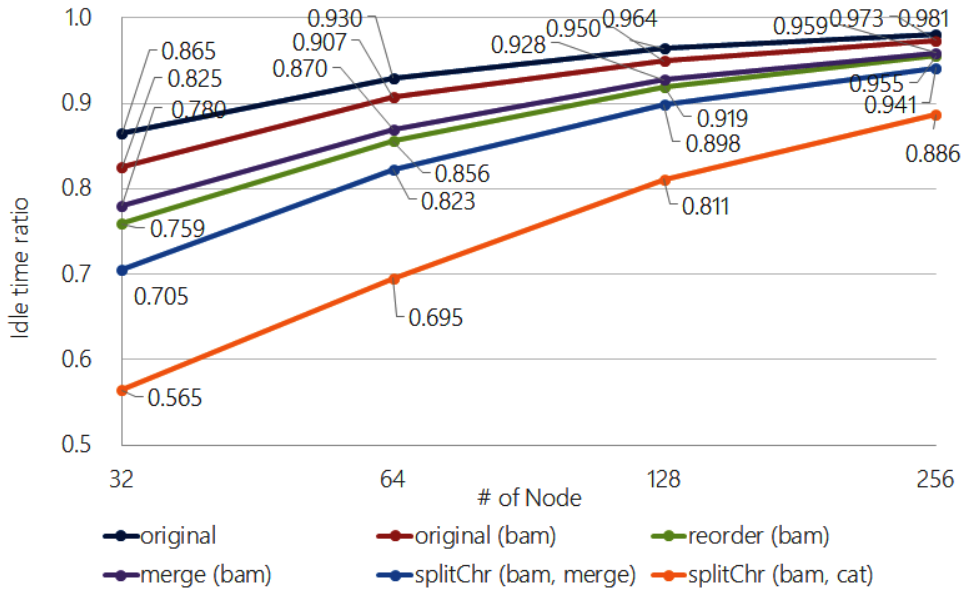


図 17　各ノードの平均待機時間

Fig. 17　Average waiting time at each node.

時間が均等でないことが考えられる．実験 2 の `splitChr (bam, cat)` におけるノード数 $N = 32$ の際に特定の 5 ノードが実行したタスクの実行時間を図 **16** に示す．ノード名，実行したタスク名，各タスクの実行時間が記されている．

d04-024 は処理したタスク数が 11，合計タスク実行時間は 7,338 秒と両方とも最も大きな値である．一方，d04-025 はタスク数が 1，合計タスク実行時間は 2,285 秒と例示した中では最も小さな値となっている．これは d04-025 が `bwa_aln.sh` の処理後にタスクを割り当てられず待ち状態であることが原因であり，実験 2 で最も実行時間の短縮が確認できた手法であっても，同時実行可能な並列数がノード数に対して不十分であることが示唆される．

この点を確認するため，各手法でのマッピング処理におけるノードの待機時間の比率を図 **17** に示す．これは各実行ノードでタスクが割り当てられていない時間をジョブ管理システムのログから計算し，全体の実行時間で割ったものである．待機時間はノード数の増加に従って増加しており，$N = 256$ のとき `splitChr (bam, cat)` では約 89%が待機時間となってしまっている．これはノード数が増えると律速となる BAM 形式ファイルのマージ処理部で実行可能な最大並列数がノード数に比べ相対的に少なくなってしまうためである．これらの結果から，我々が今回提案した

改善手法では不要な計算処理を除くことができた一方で，数百ノードを使用する場合の並列化効率の改善のためには，特にマッピング結果の処理についてさらなる改良が必要であることが分かる．たとえば，本研究で提案した染色体ごとに処理を分割して並列に実行する方式（splitChr）も同時実行可能な最大数は染色体数に依存しており，256 ノード実行時では約 200 ノードが待機状態となってしまう．そのため，今後は各染色体を細かく分割して複数ノードに処理を割り当てるような処理方法を考案し，同時実行並列数の改善とタスク間の負荷の偏りをさらに軽減することが今後の課題である．

8. 関連研究との比較

本研究では Genomon-exome をベースとして解析パイプラインの開発を行ったが，Genomon-exome のほかにもエクソーム解析のためのパイプラインは複数存在している．SIMPLEX [2] ではパイプラインをエクソーム解析に必要となる関連ソフトウェアを含めた仮想イメージとして提供しており，この形式であれば煩雑なインストール作業を避けることができ，商用クラウドなど様々な環境でも同一のパイプラインの実行が可能となる．しかし，「京」のようなスーパーコンピュータ上では仮想化環境は提供されておらず，またタスク分散や実行をシステムに最適化することができないためパフォーマンスの面からも問題があると考えられる．

また，本研究で開発されたエクソーム解析パイプライン以外にも「京」上で動作する生命情報解析パイプラインは構築されつつある．NGS analyzer [3] は理化学研究所で開発された Whole-genome の解析パイプラインであり，MPI を利用して並列実行され，スーパーコンピュータ「京」などの大型並列計算機で解析が実行可能である．本研究は対象であるエクソーム解析が目的ではないが，同じ「京」上の解析パイプラインとして実装の際の参考とした．さらに MapReduce [27] を利用した「京」上のフレームワークである KMR（K Map Reducer）[28] で NGS analyzer のパイプラインを実装してゲノム解析の効率化を図った報告 [29] もあり，これらの知見を取り込むことは本研究の今後の課題である．

9. 結論

9.1 本研究の結論

本研究では世界有数の大規模計算機であるスーパーコンピュータ「京」上で，エクソームシーケンスデータの解析を行うパイプラインソフトウェアの開発を行い，「京」のような特殊なファイルシステムやノード構成を持つ大規模並列環境においても効率的なエクソーム解析が可能であることを示した．我々の開発したパイプラインは「京」の言語環境に対応するだけでなく，MPI を用いた Master-Worker モデルによるタスク分散フレームワークである MPIDP を導入することで各処理を分散して実行することを可能とし，生命情報解析一般で問題となる膨大なジョブ投入数という問題を低減させることに成功した．さらに，参照配列へのマッピング処理において並列実行性能のボトルネックであったマッピング結果の SAM（BAM）形式ファイルの処理の同時並列実行数を改善する提案を行い，染色体ごとに処理を分割することで非分割時に比べ最大で 6.7 倍の実行速度の向上に成功した．

また，開発したパイプラインは，一般的な MPI によって並列化が実現されており，その使用はスーパーコンピュータ「京」上に限定されず，一般のクラスタ型計算機においても利用が可能である．

本研究によって，従来のエクソームシーケンスデータの解析のボトルネックの 1 つであったアラインメント処理の高速化に日本最大規模の並列計算機であるスーパーコンピュータ「京」の利用が可能となった．この成果は将来的にゲノム情報の計算機上の解析コストがボトルネックとなっている現状の改善だけでなく，ゲノム情報に基づいた個別化医療などの実現へとつながるものであると期待される．

9.2 今後の課題

本研究では単一のサンプルを対象としてマッピング処理の改善を図り，評価実験を通して高速化の効果を確認したが，マッピング処理そのものは使用するノード数に応じた高速化が達成できた一方で，マッピング結果ファイル処理の最大同時並列数の問題から全体としてのノード数増加に対する並列実行性能の向上は不十分なものであった．このため，同時実行可能な並列数を増加させる試みとして，各染色体についてもさらに細かな領域分割を行うことなどによってノード間の負荷をより分散させることが必要であると考えられる．

また，本研究ではマッピング処理の改善に注目したが，マッピング処理後の変異検出や遺伝子への関連付けの工程においても現状ではマッピング処理とほぼ同程度の計算時間を必要としている．そのため，今後はこの後処理の工程に関してもマッピング処理と同様に並列処理の実装を検討する必要がある．主要な計算となる変異検出処理部分については染色体ごとの並列化が可能であるが，それ以外の処理については並列化が困難であり，今後の課題の 1 つである．

また，実際の解析では，多数のサンプルを取得して同時に解析を行うケースが一般的である．本研究では単一のサンプルを対象とした解析の実験を行ったが，大規模な解析では大量のサンプルセットを同時並列的に実行することが必要であり，そのような場合にはノードあたりのタスク量が増大するため，実験で良好な結果が得られなかった並列実行性能の向上が期待できる．現時点でも手動で大量のサ

ンプルの解析を行うことは可能であるが，実際の解析の際により有用となるよう大量のサンプル解析を補助する機能の実装やその実証実験を行うことは今後の課題の 1 つである．

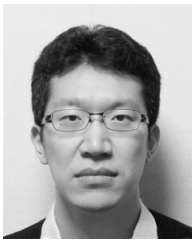
謝辞　本研究で使用した Genomon-exome をご提供いただき，また研究に関する様々な助言を賜った，東京大学医科学研究所宮野悟教授，井元清哉教授，白石友一助教，玉田嘉紀特任講師，伊東聰博士，千葉健一博士に厚く御礼申し上げる．本研究の結果は，理化学研究所のスーパーコンピュータ「京」を利用して得られたものである（課題番号：hp130017，hp140230）．本研究で，東京大学ヒトゲノム解析センター・スーパーコンピュータを利用した（http://sc.hgc.jp/shirokane.html）．本研究は，文部科学省博士課程教育リーディングプログラム東京工業大学「情報生命博士教育院」の支援を受けて行われた．

参考文献

[1] Genomon-exome—東京大学医科学研究所ヒトゲノム解析センター，入手先 ⟨http://genomon.hgc.jp/exome⟩.
[2] Fischer, M., Snajder, R. et al.: SIMPLEX: Cloud-enabled pipeline for the comprehensive analysis of exome sequencing data, *PLOS ONE*, Vol.7, No.8, e41948 (2012).
[3] NGS analyzer—理化学研究所統合生命医科学研究センター，入手先 ⟨http://www.csrp.riken.jp/application_d_j.html⟩.
[4] 宮崎博行，草野義博，新庄直樹，庄司文由，横川三津夫，渡邊　貞：スーパーコンピュータ「京」の概要，*FUJITSU*, Vol.63, No.3, pp.237–246 (2012).
[5] Michael, J.B., Sarah, B.N. et al.: Exome sequencing as a tool for Mendelian disease gene discovery, *Nature Reviews*, Vol.12, No.11, pp.745–755 (2011).
[6] Yoshida, K., Sanada, M, et al.: Frequent pathway mutations of splicing machinery in myelodysplasia, *Nature*, Vol.478, pp.64–69 (2011).
[7] Li, H. and Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform, *Bioinformatics*, Vol.25, No.14, pp.1754–1760 (2009).
[8] McKenna, A., Hanna, M. et al.: The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data, *Genome Research*, Vol.20, No.9, pp.1297–1303 (2010).
[9] Li, H., Handsaker, B. et al.: The Sequence Alignment/Map format and SAMtools, *Bioinformatics*, Vol.25, No.16, pp.2078–2079 (2009).
[10] Picard, available from ⟨http://broadinstitute.github.io/picard/⟩.
[11] Wang, K., Li, M. and Hakonarson, H.: ANNOVAR: Functional annotation of genetic variants from high-throughput sequencing data, *Nucleic Acids Research*, Vol.38, No.16, e164 (2010).
[12] Marcel, M.: Cutadapt removes adapter sequences from high-throughput sequencing reads, *EMBnet. journal*, Vol.17, No.1, pp.10–12 (2011).
[13] Heng, L., Jue, R. and Richard, D.: Mapping short DNA sequencing reads and calling variants using mapping quality scores, *Genome Research*, Vol.18, No.11, pp.1851–1858 (2008).
[14] Aaron, R.Q. and Ira, M.H.: BEDTools: A flexible suite of utilities for comparing genomic features, *Bioinformatics*, Vol.26, No.6, pp.841–842 (2010).
[15] Gentleman, R.C., Carey, V.J. et al.: Bioconductor: Open software development for computational biology and bioinformatics, *Genome Biology*, Vol.5, No.10, R80 (2004).
[16] Peter, J.A.C., Christopher, J.F., et al.: The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variant, *Nuclelic Acids Research*, Vol.38, No.6, pp.1767–1771 (2010).
[17] Sequence Alignment/Map Format Specification–The SAM/BAM Format Specification Working Group, available from ⟨http://samtools.github.io/hts-specs/SAMv1.pdf⟩.
[18] Ajima, Y., Inoue, T. et al.: The Tofu Interconnect, *IEEE Micro*, Vol.32, No.1 (2012).
[19] Peter, J.B.: *The Lustre Storage Architecture*, Cluster File Systems, Inc. (2004).
[20] Sakai, K., Sumimoto, S. and Kurokawa, M.: High-performance and highly reliable file system for the K computer, *Fujitsu Scientific and Technical Journal*, Vol.48, No.3, pp.302–309 (2012).
[21] Ohue, M., Shimoda, T., Suzuki, S., Matsuzaki, Y., Ishida, T. and Akiyama, Y.: MEGADOCK 4.0: An ultra-high-performance protein-protein docking software for heterogeneous supercomputers, *Bioinformatics*, Vol.30, No.22, pp.3281–3283 (2014).
[22] Kakuta, M., Suzuki, S., Ishida, T. and Akiyama, Y.: A massively parallel sequence similarity search for metagenomics sequencing data (submitted).
[23] GHOST-MP, available from ⟨http://www.bi.cs.titech.ac.jp/ghostmp/⟩.
[24] Deutsch, P.: GZIP file format specification version 4.3, *RFC Editor* (1996).
[25] Seo, J.S., Ju, Y.S. et al.: The transcriptional landscape and mutational profile of lung adenocarcinoma, *Genome Research*, Vol.22, No.11, pp.2109–2119 (2010).
[26] 佐藤芳樹：HPC 向け高水準プログラミング言語 X10 の評価，スーパーコンピューティングニュース，Vol.15, No.6 (2013).
[27] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *6th Symposium on Operating Systems Design and Implementation* (2004).
[28] Matsuda, M., Maruyama, N. and Takizawa, S.: K MapReduce: A scalable tool for data-processing and search/ensemble applications on large-scale supercomputers, *IEEE International Conference on Cluster Computing* (*CLUSTER*), pp.1–8 (2013).
[29] 滝沢真一郎，松田元彦，松山直也：MapReduce による計算科学アプリケーションのワークフロー実行支援，ハイパフォーマンスコンピューティングと計算科学シンポジウム（HPCS2014）(2014).

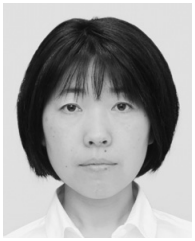
青山 健人

富士通株式会社ミドルウェア事業本部．2013 年電気通信大学電気通信学部情報工学科卒業．2015 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了，修士（工学）．2015 年富士通株式会社．



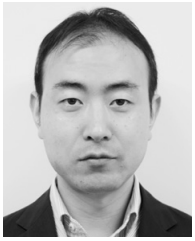
角田 将典

東京工業大学情報理工学研究科産学官連携研究員．2010 年東京大学農学生命科学研究科博士課程修了，博士（農学）．東京大学農学生命科学研究科特任研究員を経て，2011 年より現職．専門はバイオインフォマティクス．



松崎 由理　（正会員）

東京工業大学情報生命博士教育院特任助教．2006 年慶應義塾大学大学院政策・メディア研究科単位取得退学，博士（学術）．東京工業大学大学院情報理工学研究科産学官連携研究員を経て，2014 年より現職．システム生物学，タンパク質間相互作用の研究に従事．



石田 貴士　（正会員）

東京工業大学大学院情報理工学研究科准教授．2006 年東京大学大学院農学生命科学研究科博士課程修了，博士（農学）．東京大学医科学研究所研究員，東京工業大学大学院情報理工学研究科助教等を経て，2014 年より現職．専門はデータマイニング，バイオインフォマティクス．

秋山 泰　（正会員）

東京工業大学大学院情報理工学研究科教授．1990 年慶應義塾大学大学院理工学研究科電気工学専攻博士課程修了，工学博士．京都大学化学研究所助教授，産業技術総合研究所研究センター長等を経て，2007 年より現職．専門はバイオインフォマティクス，大規模並列計算応用．