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

滝澤 真一朗[1,a)]　松田 元彦[1,b)]　丸山 直也[1,c)]

概要：計算科学アプリケーションにはアンサンブル計算や多数のデータ処理等，多数のタスクをワークフロー実行できるものが多くある．我々は大規模並列システムで実行されている計算科学アプリケーションのワークフロー実行パターンを抽出し，MapReduce プログラミングモデルにてワークフローを構築するための機能要件を精査した．その機能要件を満たすべく，MapReduce 処理系 K MapReduce に追加機能として実装した．計算科学アプリケーションワークフローの MapReduce 実装事例として，レプリカ交換分子動力学法シミュレーション，ゲノム変異解析アプリケーションを実装した．MapReduce を用いない実装との比較評価を行った結果，両者にて性能面での優位性を，後者では記述面での優位性も確認した．

Supporting Workflow Management of Scientific Applications by MapReduce Programming Model

Shinichiro Takizawa[1,a)]　Motohiko Matsuda[1,b)]　Naoya Maruyama[1,c)]

Abstract: There are many applications that run many tasks as a workflow, such as ensemble simulation and data processing, in scientific applications. We surveyed various applications that run on large scale parallel systems, extracted their workflow patterns and then investigated requirements for building the workflow on the MapReduce programming model. To fulfill the requirements, we implemented them to 'K MapReduce' MapReduce implementation. To illustrate the scientific applications' workflows on the MapReduce model, we implemented two application workflows on the K MapReduce; Replica exchange molecular dynamics and Genome analysis. As a result of comparison with implementations without MapReduce, we confirmed performance benefits in the both cases and a code size benefit in the latter case.

1. はじめに

京コンピュータの様な大規模並列システムでは，宇宙科学や生命科学，気象科学等の自然科学分野から，ものづくりや社会経済予測等の多種多様な計算科学アプリケーションが実行されている．それらの計算は個々の計算が多数の CPU を必要とする大規模並列なものも存在するが，一方でデータ処理やアンサンブル計算など，多数の計算を Embarrassingly Parallel(EP) に実行し，その結果を集約するワークフロー実行形態を取るものも多い．例えば次世代シークエンサーの出力解析プログラムでは，個々の塩基配列は独立しているため，数億の塩基配列を分割して個別に平行してマッピング処理を行った後に，変異同定等の解析を行う．また，ゲノムデータであれば SAM フォーマット，気象データであれば NetCDF 等，大量の実験・観測データを計算機で処理するための標準的なフォーマットに変換する作業を並列に実行する計算機利用シナリオもある．

一方で，テキストマイニングやログ解析等の大規模データを並列に処理するプログラミングモデルとして MapRe-

[1] 理化学研究所 計算科学研究機構, RIKEN AICS
[a)] shinichiro.takizawa@riken.jp
[b)] m-matsuda@riken.jp
[c)] nmaruyama@riken.jp

duce [1] が広く用いられている．MapReduce を用いた並列プログラムの実装は，計算対象を Map タスクと Reduce タスクの 2 つの逐次処理として分割実装するだけで簡易に実装できるためである．MapReduce モデルにおいては一般的に，Map タスクは Key-Value (KV) を生成し，Reduce タスクは KV から値を抽出する処理として実装され，個々のタスクは互いに独立して並列に実行される．近年ではゲノム解析等の計算科学アプリケーションにおいても，MapReduce の EP なタスク実行形態がワークフロー実行要求に適しているということで，使われ始めている． [2], [3]

我々は MapReduce プログラミングモデルを用いて計算科学アプリケーションの汎用的なワークフロー実行システムの構築を目指す．大規模並列システムで実行される，様々な分野の計算科学アプリケーションの実行パターンを調査し，粗粒度なタスク実行を要求するアプリケーションを抽出し，MapReduce モデルでワークフローを実現するための機能要件を精査した．その結果，一般的な MapReduce 処理系が有する機能に加え，数千〜数万ノードでの高速な Shuffling と負荷分散，効率の良い MapReduce 繰り返し実行，IO を用いないオンメモリでの高速な実行と耐故障性の両立が必要であると判明した．

MapReduce モデルによる計算科学アプリケーションのワークフロー実装を検証するために，分子動力学シミュレーションとゲノム解析アプリケーションのワークフローを構築した．MapReduce 処理系としては，我々が開発している大規模並列システムでのスケーラブルな処理系である，K MapReduce [4] を用いた．結果，それぞれのワークフローが MapReduce モデルで構築できることを確認した．MapReduce を用いない既存実装と比較した結果，分子動力学シミュレーションではコード量が増加したものの，タスク管理の効率化のため実行時間の改善を確認した．ゲノム解析アプリケーションではコード量は同程度であり，IO 量削減のため実行時間の改善を確認した．これら知見を元に，MapReduce 処理系の改良を進めるとともに，他分野のアプリケーションへの適用を進める．

2. 計算科学アプリケーションのワークフロー実行要件

我々は「将来の HPCI システムのあり方の調査研究 - アプリケーション分野」にてとりまとめを行っている「計算科学ロードマップ」白書 [5] を参考に，将来のエクサスケールシステムに向けて重要となる，様々な計算科学分野のアプリケーションの実行パターンを調査した．

要求性能・実行要件が精査されている 106 のアプリケーションのうち，76%にあたる 81 アプリケーションにおいて，学術的知見を得るために複数ケースの実行を要することがわかった．複数ケース実行の規模はアプリケーションによるが，例えば防災シミュレーションでは 5,000 ケース，素粒子物理学では 100,000 ケースを要す．実際には 1 ケースの結果を得るにおいても，シミュレーション，データ処理，集計とステップに分けて計算を実行することがあり，システムで実行されるジョブ数はケース数の倍以上と予想できる．このため，スケジューラへの負荷増加を防ぐには，関連する計算はワークフローを構成してまとめて 1 つのジョブとして実行する対応が必要である．これにより，ジョブ実行にステージングを要するシステムでは，ステージイン・アウトを減らし IO 帯域の浪費削減にもつながる．一方で，ワークフロー実行する際には，並列実行されるタスク間での負荷バランスを考慮し，遊休計算資源の発生を最小限に押さえることも重要である．また，各アプリケーションは C/C++/Fortran だけではなくスクリプト言語等，多種多様な言語で記述されているため，ワークフロー管理システムでは言語に制約があってはならない．

複数ケースを実行を要する 81 アプリケーションの内，少なくとも 20 アプリケーションに関しては各ケース内において，EP なタスク実行，並列シミュレーション実行しアンサンブル計算を行う，と確認できた．例えば，ゲノム解析では 1 ケースあたり 1,000 人のゲノム解析を並列して行うが，1 人あたり 1TB ある塩基配列解析も配列毎に 100〜10,000 規模で EP に処理でき，また，分子構造解析においては 100〜10,000 メンバーのアンサンブル計算を繰り返し行う．ワークフローとして実行する場合，残りの 61 アプリケーションについても同等な実行形態が要求されると考えられ，数万規模でのタスク並列実行やその結果の効率的な集約技術が重要となる．

ケースあたりの実行時間について調査したところ，平均 180 時間要することを確認した．ワークフローとして実行する際には複数ケースを直列に実行する場合もあり，さらなる長時間にわたる実行が求められるため，ワークフロー管理システムには耐故障性の実現が求められる．

以上より，計算科学アプリケーションのワークフロー要件は以下にまとめられる．

- 数万規模のタスクの並列実行
- タスク実行結果の集約
- 並列タスク間の負荷分散
- 耐故障性の実現
- 任意のプログラムの実行

これらは一般的に謳われていることではあるが，将来重要となるアプリケーションについて網羅的・定量的に調査した点は我々の貢献である．

3. MapReduce による計算科学ワークフロー管理の提案

2 章で挙げた要件を満たす計算科学アプリケーション用ワークフロー管理システムを，我々は MapReduce モデル上に構築することを提案する．MapReduce はデータ解析

分野で広く用いられており，計算対象処理を Map タスクと Reduce タスクの 2 つの逐次処理として分割して実装するだけで，処理の並列化や耐故障性は MapReduce 処理系により透過的に実現される．MapReduce モデルにおいては一般的に，Map タスクは KV を生成し，Reduce タスクは KV から値を抽出する処理として実装されるが，個々のタスクは互いに独立して並列に実行されるため，データ処理だけではなく，アンサンブル計算のような EP なタスクの実行基盤としても適している．

MapReduce の処理系としては Hadoop [6] や，MPI で実装された MR-MPI [7] 等多数ある．しかしながら，2 章の要件を満たすには処理系は次の機能を有していなければならないが，これらを総合的に満たす既存の処理系はない．

高速な Shuffling Map タスクで処理するデータや計算が増えるにつれ，Map タスク実行ノードと Shuffle 対象となるデータ量が増える．数千〜数万並列システムでの高速な Shuffle が必要となるが，Hadoop などファイル IO を前提とした Shuffle では IO 性能がボトルネックとなるため，オンメモリでの Shuffle が求められる．

繰り返し処理のサポート アンサンブル計算を MapReduce モデルで実行する場合，個々のシミュレーションは Map タスクとして実行し，結果の集約を Shuffle，集計処理を Reduce タスクとして実行し，次ステップの実行条件を計算できる．この処理を繰り返し実行するため，Map/Shuffle/Reduce の繰り返し実行の機能が必要である．しかしながら，Hadoop や MR-MPI では繰り返し実行をサポートしていないため，同等の処理を行うには，クラスファイルの再ロードやオブジェクトの再生成，データの再読み込み等のオーバーヘッドが発生する．

オンメモリの高速な実行と耐故障性の両立 大規模並列システムで計算科学アプリケーションを MapReduce モデルで高性能に実行するには，不必要な IO は避け，処理をオンメモリに実行する必要がある．一方で，ノード障害やジョブ実行時間超過等で中断された処理を再開するためには，実行状態のディスク保存が必要となる．オンメモリ処理と IO のバランスが不可欠である．

負荷分散 アンサンブル計算，特に階層的マルチスケールシミュレーションにおいては，Map タスクとして実行する個々の計算の計算量が大きく異なるため，Map タスクを実行するノード間で負荷が均衡になるようタスクを割り当てる必要がある．また，Reduce タスクにおいても，MapReduce 一般の問題として，Key 毎の Value 量にばらつきが生じるため，処理データ量の均一化が必要となる．

4. 実装

3 章の機能を実現する MapReduce による計算科学アプリケーションワークフロー管理システムを，我々が開発している，大規模並列システム上でスケーラブルに動作する MapReduce 処理系である K MapReduce (KMR) [4][*1] をベースに構築する．KMR は京コンピュータの様な数万ノード規模の大規模並列システムでの高いスケーラビリティを実現するために，(1)IO を伴わないオンメモリ処理，(2)MPI/OpenMP によるハイブリッドタスク実行，(3) ネットワーク・ストレージアーキテクチャを考慮した通信・IO を行う．以降に提案の実装について述べるが，「高速な Shuffling」については文献 [4] にて実現されているため，それ以外の項目について述べる．また，要件にあげた，任意のプログラムを MapReduce タスク実行するための実装手法についても述べる．

4.1 任意のプログラムのタスク実行

利用者は KMR を MPI ライブラリの 1 つとして利用でき，Map/Reduce タスクを C/C++/Fortran にて実装できる．図 1 に KMR による MapReduce プログラムの擬似コードを示す．利用者は Map/Reduce の各タスクに加え，MapReduce 処理を実行するメインプログラムを実装する．KMR の MapReduce 処理は複数の KV を格納する，Key-Value Store (KVS) データ型を介して実行される．Map/Reduce タスクは入力として KVS を受け取り，そこに含まれる KV を処理して，出力 KVS に計算結果を保存する関数として実装する．また Hadoop 等では Shuffle は自動的に実行されるが，KMR では明示的に Shuffle 関数 `kmr_shuffle` を実行しなければならない．

既存のプログラムを活かすために，MPI ライブラリとしての利用以外に以下の利用方法を提供する．

スクリプト言語による Map/Reduce タスク実装 多量のデータのフォーマット変換等，既存のツール群を Map/Reduce タスクで利用するために，スクリプト言語で各種コマンドをラップすることでタスク実装できる機能 *KMR Shell* を実装した．KMR Shell は Hadoop Streaming に相当する機能であり，図 2 に示すとおり，Map タスクは空白文字区切りで KV を標準出力に書き出し，Reduce タスクは標準入力からそれを読み込むよう実装する．スクリプトは計算ノードがサポートする任意の言語で実装でき，各ノードで MPI プロセスとして実行される．

ただし，京コンピュータ等一部のシステムでは計算ノードで MPI 並列プログラムの起動を許可していないため，この方法で MPI 並列プログラムを実行することはできず，次に示す方法を用いなければならない．

MPI プログラムの Map タスク実行 アンサンブル計算では，個々のシミュレーションに既存の MPI 並列プ

[*1] `http://mt.aics.riken.jp/kmr/`

```
// Initialize environment
MPI_Init(...); KMR_Init(...);

// Input Key-Value Store (KVS)
ikvs = kmr_create_kvs(key_type, value_type);
// Output KVS of Map tasks
mkvs = kmr_create_kvs(key_type, value_type);
// Output KVS of Reduce tasks
okvs = kmr_create_kvs(key_type, value_type);

// Add Key-Value (KV)s to the input KVS
while (...) {
  kmr_add_kv(ikvs, kv);
}

// Run Map tasks to process each KV
kmr_map(ikvs, mkvs, map_function);
// Shuffle the resultant KVS of Map tasks
kmr_shuffle(mkvs);
// Run Reduce tasks
kmr_reduce(mkvs, okvs, red_function);

KMR_Finalize(); MPI_Finalize();
```

図 1 KMR の擬似コード

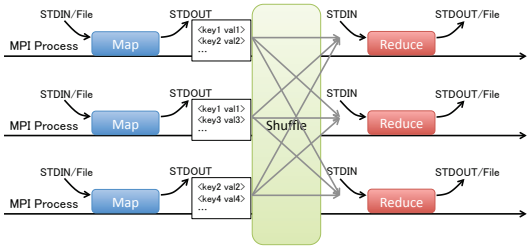

図 2 KMR Shell の動作概要

ログラムを利用するものがある．KMR にてこの実行パターンをサポートするため，`MPI_Comm_spawn` にて MPI プログラムを Map タスクとして実行する機能を提供する．この機能を使用するには，専用の関数 `kmr_map_via_spawn` に対して，(1)KV の Value として MPI プログラムとその引数がセットされた KVS，(2)MPI プログラム実行後に結果を取得するためのコールバック関数，を引数に与えて実行する．

4.2 繰り返し処理のサポートとオンメモリ実行

図 1 のとおり，KMR では KVS に格納した KV に対して Map/Reduce 処理を行う．タスクの出力 KVS にて明示的に要素を空にしない限り，任意の Map/Reduce タスクを繰り返し実行できる．このとき，KVS はメモリ上に確保されているので，データ入出力 IO は発生しない．

また，MapReduce モデルでは，各タスクはプロセス毎にローカルな処理であり，結果は Map/Reduce タスクを実行するプロセス間では共有されない．Hadoop で MapReduce

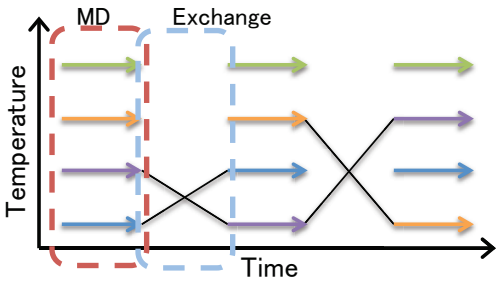

図 3 REMD のワークフロー

を繰り返し実行する場合には，前ステップの Reduce 結果をディスクから読み込むことで Map タスクを実行するプロセス間で結果を共有できるが，IO を必要とする．KMR では，Reduce 結果の共有を IO なしに実現するために，KV 共有の関数 `kmr_replicate` を提供する．これは `MPI_Allgather` に相当する機能であり，各 MPI プロセスが持つ KV を互いに送信し共有する関数である．

以上により MapReduce を繰り返し実行する場合にも，全てオンメモリにて処理が行える．

4.3 耐故障性と負荷分散

耐故障性と負荷分散機能は現状有しておらず，今後の課題である．

5. アプリケーション事例

計算科学アプリケーションのワークフローを，我々が提案する MapReduce モデルを用いて実装できることを示すために，2 つの事例を示す．1 つはレプリカ交換分子動力学法を用いたシミュレーション (REMD: Replica Exchange Molecular Dynamics)，もう 1 つは次世代 DNA シークエンサーにより読み取られた DNA の塩基配列から変異を同定するアプリケーションである．どちらも既存の実装があり，それらを KMR を用いた MapReduce モデルに当てはめ実装した．評価では既存実装とのコード量，性能の比較を行った．性能評価には京コンピュータ [8] を用いた．

5.1 REMD

5.1.1 問題説明

REMD では，分子構造の複数のレプリカを用意し，それぞれに異なる温度を割り振り，構造サンプリングを行う．サンプリングのためのシミュレーションには MD 法が用いられるが，レプリカ間には相互作用が無いため，サンプリングシミュレーションは独立して実行できる．レプリカの MD 実行完了後，メトロポリス法の基準でレプリカの温度パラメータを交換し，繰り返し MD を実行するアンサンブル計算を行う（図 3）．

我々は REMD を MapReduce で実装するにあたり，宮下，杉田によって開発された REIN (Replica Exchange Inter-

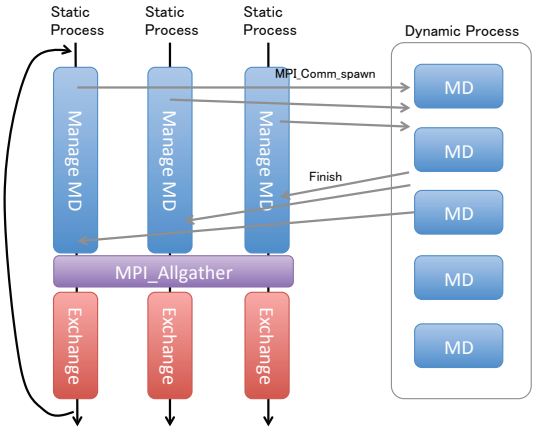

図 4 REIN の REMD 実装

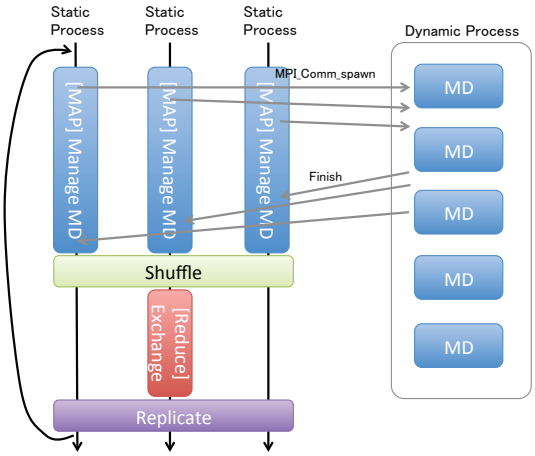

図 5 REMD の MapReduce ワークフロー

face Program) [9], [10] を用いた．REIN は既存の MD プログラムを用いて多次元 REMD を行うプログラムであり，Fortran と MPI にて実装されている．図 4 に示すとおり，REIN 実行には少数の静的プロセスと，`MPI_Comm_spawn` により生成された多数の動的プロセスが使われる．動的プロセスは MD プログラムの実行に使われる．静的プロセスは，各々が担当する MD プログラムの制御と，温度交換条件の計算に使われる．温度交換条件を計算するに際し，全ての MD の結果が必要となるため，`MPI_Allgather` を用いて MD 実行結果を全静的プロセス間で共有した後に，各々で計算を行っている．

5.1.2 MapReduce 実装

図 4 からわかるとおり，各静的プロセスは `MPI_Allgather` を除いて互いに独立していることがわかる．このワークフローを参考に構築した MapReduce 対応ワークフローを図 5 に示す．

Map タスクとして個々の MD を各静的プロセスが実行する点は等しいが，温度交換条件計算を単一プロセスで行っている点が異なる．これは，温度交換条件の計算には全 MD の実行結果が必要となるものの，MD 毎に生成する KV の Key を変えると Shuffle 時に複数の MPI プロセスにデータが分散され，計算できなくなるためである．今回は各 Map タスクの生成 KV の Key を統一することで，Shuffle 後に単一プロセスに全ての KV が集約されるようにした．これにより，Reduce タスクを実行しない遊休プロセスができるが，プロファイルの結果，温度交換条件計算の実行時間は総実行時間に比べて小さいことがわかっており（65 ノード 520 並列時で 0.001%），問題ではないと考えている．また，単一 MPI プロセスのみ温度交換条件を有するので，次のステップに移る前に `kmr_replicate` にて共有している．

この MapReduce ワークフローは Fortran で記述された REIN のコードをベースに，(1) 全 MPI 関数を削除し KMR 関数で置き換える，(2)MPI ランクに応じた明示的なタスク割り当てコードの削除，を行い実装した．(2) に関してはプログラムの冒頭にて，Key としてレプリカの ID，Value としてレプリカ情報からなる KV をレプリカ数分生成し，それを格納した KVS を Shuffle することで実現した．Shuffle では Key に応じて KV を MPI プロセスに分配するので，これにより暗黙的なタスク割り当てが実現される．

5.1.3 評価

REIN と KMR による MapReduce 実装のコード量について比較を行った．プリプロセッサ処理後のコードから空行やコメントを除いたコード量を比較したところ，MapReduce 実装の方が 528 行の増加となった．これは REIN のコードに対して 15%の増加に相当する．この理由は主に 2 つある．1 つはパラメータ変数を KMR 関数を用いてプロセス間で共有している点である．REIN では `MPI_Bcast` にて 40 個のパラメータを 1 つあたり高々 2 行程度のコード量で共有していたが，KMR では KVS の作成，KV の登録，`kmr_replicate` での共有，KV の取り出しを行い，1 パラメータあたり平均 8 行のコード量となった．また，KV に変数を登録する時の型変換オーバーヘッドもある．もう 1 つは Map/Reduce タスクを定義する関数のオーバーヘッド行である．今回 Map/Reduce タスク関数を計 11 個定義したが，タスク関数は特定の引数・返り値を持つ関数として定義しなければならず，引数の型宣言，引数の型変換とそのエラー処理コードの実装のため，関数あたり平均 10 行の増加となった．後者については削減の余地は無いが，前者については MPI 関数とハイブリッドなプログラムを書くことにより削減可能である．実際，KMR 実装にて `MPI_Bcast` でパラメータを共有する場合では，コード増加量は 137 行であり，4%の増加であった．

次に実行時間について比較を行った．入力データには REIN 付属のサンプルデータを用い，1 次元 REMD，イテレーション回数は 10 回とした．レプリカ数を 8, 16, 32, 64, 128, 256 と変化させて実行した．1 つの MD を実行する MPI プロセス数を 8 とし，京の 1 ノードで実行できる

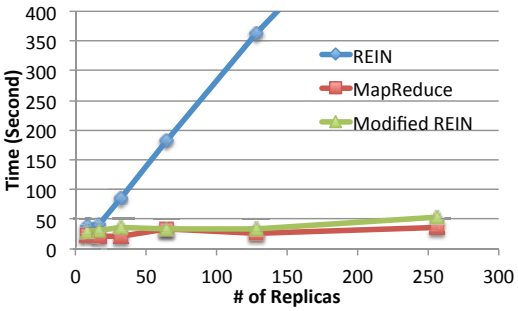

図 6 REMD ワークフローの REIN と MapReduce 実装の実行時間の比較

規模とした．また，1 つの静的プロセスが 1 つの MD 実行を管理するように設定した．京の 1 ノードの CPU コア数は 8 であるため，1 コア 1 プロセス実行とたところ，レプリカ数に応じて使用ノード数は 9, 18, 36, 72, 144, 288 となった．なお，MD プログラムには NAMD2 を用いた．

結果を図 6 に示す．横軸はレプリカ数であり，縦軸はイテレーションの平均実行時間を表す．凡例「REIN」は REIN の実行時間，「MapReduce」は KMR による MapReduce 実装の時間，「Modified REIN」は性能ボトルネック箇所を修正した REIN の実行時間を表す．行っている計算はワークフロー実装によらず等しいため，本来ならば REIN と MapReduce は同程度の性能となるが，後者のほうが優れた性能となった．表 1 に示す，レプリカ数 64 の時のワークフローの各ステップの実行時間内訳によると，MD 実行が実行時間の大部分を占めることと，REIN と MapReduce 実装に大きな差があることがわかる．調査したところ，REIN では MD を起動する時の `MPI_Comm_spawn` のコミュニケータ引数に `MPI_COMM_WORLD` を指定しているのに対して，KMR では `MPI_COMM_SELF` を指定している．`MPI_COMM_WORLD` の場合，`MPI_Comm_spawn` はコレクティブオペレーションとなり，特定 1 プロセスからのみ MPI プログラムが起動される．一方で `MPI_COMM_SELF` の場合にはプロセスローカルな処理となり各静的プロセスから MPI プログラムが起動される．実際，REIN のコミュニケータを `MPI_COMM_SELF` に変更し評価を行った結果が表の凡例「Modified REIN」であり，MapReduce と同程度の性能となっていることが確認できる．本件は MapReduce モデルを採用したために解決した問題では無いが，KMR を用いて実装すれば `MPI_Comm_spawn` の詳細は隠蔽されるため，利用者は実装上の差異を意識せず高性能を達成できる．なお，コミュニケータの件は REIN 開発者にフィードバック済みで，本体に修正が反映されている．

5.2 ゲノム変異解析

5.2.1 問題説明

次世代シーケンサー (Next Generation Sequencer) は DNA 塩基配列を高速に大量に読み込むことができる一方で，その膨大なデータの解析には多くの計算量を要求する．具体的には，ヒトゲノムの場合，総容量数百 GB のショートリード (配列長数十〜数百程度の解析対象塩基配列群) が既知の DNA 参照配列のどの位置に相当するか判定するマッピング処理を行った後に，参照配列との差異を解析する．

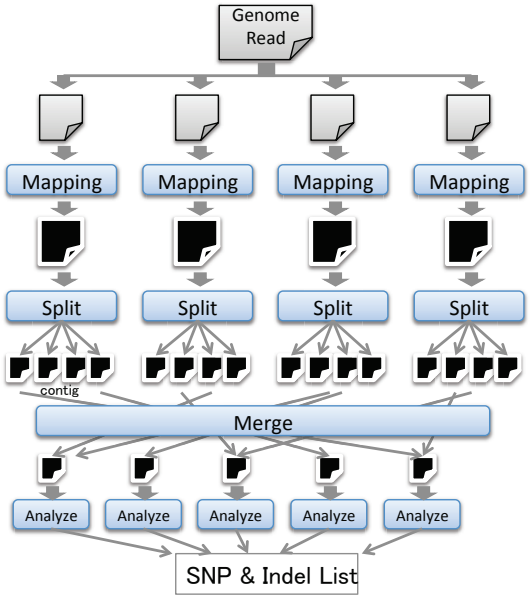

図 7 NGS Analyzer のワークフロー

我々はゲノム変異解析を行うアプリケーションとして，NGS Analyzer [11] を参考にした．NGS Analyzer は次世代シークエンサーの出力データを高速に解析し，ヒト個人間の遺伝的差異やがんゲノムの突然変異を高い精度で同定するプログラムであり，その解析手法の詳細は日本人男性の全ゲノム解析を行った文献 [12] に記されている．

NGS Analyzer のワークフローを図 7 に示す．処理対象となる塩基配列 (Genome Read) を一定量の塩基配列で分割し，それぞれをマッピング処理し (Mapping)，参照配列中の位置を判定する．その後，コンティグ (連続する塩基配列パターン) 単位でマッピングデータを分割する (Split)．マッピングとその結果の分割は入力毎に独立した処理であり並列実行できるが，結果解析の前に，コンティグ単位で各マッピング結果をマージする必要がある (Merge)．並列処理を行う場合，マージにて各プロセスは全プロセスの出力を参照する必要があるため，NGS Analyzer ではプロセスローカルストレージから共有ストレージに出力を移動し，マージ後の結果をローカルストレージに保存する実装となっている．マージ後，コンティグ単位で SNP，INDEL 等の変異同定を統計的に行う (Analyze)．この処理も並列実行可能である．NGS Analyzer はこのワークフローを既存ツールや独自に実装した解析用プログラムをシェルスクリプトで連結させたパイプラインとして実装している．

5.2.2 MapReduce 実装

図 8 に MapReduce 対応ワークフローを示す．塩基配列

表 1　レプリカ数 64 の場合の REMD の各ステップの実行時間 (秒)

	Setup	Rescaling	MD	Energy Distribution	Exchange	Output
REIN	0.652	0.800	180.560	0.074	0.001	0.707
MapReduce	3.727	1.156	28.057	0.055	0.001	0.056

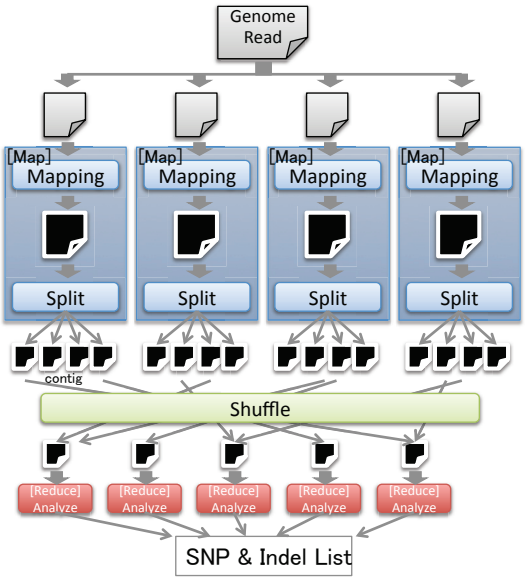

図 8　NGS Analyzer の MapReduce ワークフロー

のマッピングと分割を Map タスク，コンティグ毎の解析を Reduce タスクとして実装した．図 7 の Merge に相当する処理は Shuffle 処理に置換される．Map タスクにて分割結果を<コンティグ名, コンティグにマップされた塩基配列情報>形式の KV として生成すれば，Shuffle にて自動的にコンティグ毎に分割されて Reduce タスクに渡る．NGS Analyer では Merge にて共有ストレージへの IO が発生するが，Shuffle はオンメモリ処理であるため，IO 量が削減される．

このワークフローを KMR Shell の機能を用いて，Python にて Map/Reduce タスクを実装した．ただし，多数のファイルの入出力を伴うため，Map タスクへの入力，Reduce タスクからの出力は標準入出力ではなく，ファイル IO とした．また，各プログラムは入力と出力にファイルを用いているため，Map タスクでは分割プログラムが生成したファイルを読み込み，作成した KV を標準出力に書き出し，Reduce タスクでは標準入力から受け取った KV をファイルに保存してから解析プログラムを実行した．なお，シェルスクリプトによる実装も可能であったが，Reduce タスクにて標準入力から取得したデータを行分割する処理にて極端なオーバーヘッドを検知したため，より高速なテキスト処理が可能な Python を用いた．

5.2.3　評価

オリジナルの NGS Analyzer と MapReduce ワークフロー実装に要するコード量の比較を行った．オリジナルではマッピングや解析等の個々の処理毎にスクリプトが分割されており，さらに京コンピュータ対応のジョブ管理機構も備えているため 748 行であった．このコードからワークフローを実現する最低限のコードを抜き出し，単一シェルスクリプトとして実装したところ 124 行となった．一方，KMR Shell による Python での MapReduce 実装では 111 行と同程度の規模となった．

次に，上記単一シェルスクリプトと MapReduce 実装の性能比較を行った．DNA 参照配列としては NCBI にて公開されているヒトゲノムデータ[*2]を用いた．解析対象塩基配列としては，日本人初の全ゲノム解読にて用いられたデータの一部[*3]を用いた．NGS Analyzer で処理可能な形式に変換すると参照配列は 6.3GB となった．

結果を表 2 に示す．2 通りのシナリオを実行した．1 つは小規模実行として，上記日本人ゲノムデータファイルの 25 万配列 (118MB) を 12 ノード上で解析，もう 1 つは大規模実行として，ゲノムデータファイル全体の 5.4 億配列 (87.9GB) を 512 ノードで解析した．小規模実行では何方も同程度の実行時間であったが，大規模実行では MapReduce 実装は単一シェルスクリプト実装の 74%の実行時間となった．この理由は IO 量にある．単一シェルスクリプトではマージを行うためにローカルストレージと共有ストレージ間で 97GB のデータの移動を行っているが，MapReduce 実装では IO を伴わないオンメモリでのデータ転送となっている．実際，単一シェルスクリプト実装の Merge と MapReduce 実装の Shuffle に要した時間を計測したところ，前者では 1,466 秒，後者では 24 秒と大きな差があった．

6.　関連研究

計算科学アプリケーションを MapReduce モデルで実現するための既存研究について述べる．MR-MPI [7] では，KMR 同様 MPI 上に実装された MapReduce ライブラリである．ページ単位で KV を管理し IO を行うため，スケーラビリティは KMR に対して乏しく，また，任意のプログラムを Map/Reduce タスクとして実行する機能は有していない．Twister [13] では繰り返し処理を行う MapReduce プログラムに対して，処理ステップ毎の IO 量を減らすデータアクセス手法を提案している．ただ，タスク毎の負荷・データ量のばらつきを考慮しておらず，負荷分散機能を有していない．我々の過去の研究 [4] は KMR の内部実装が主題であり，本研究は計算科学アプリケーションへの

*2　ftp://ftp.ncbi.nlm.nih.gov/genomes/Homo_sapiens/
*3　http://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=viewer&m=data&s=viewer&run=DRR000617

表 2 ゲノム変異推定の実行時間 (秒)

	250,000 Reads/12 Node (118MB)	545,411,544 Reads/512 Node (87.9GB)
NGS Analyzer Single Shell	357	4,985
MapReduce	353	3,691

MapReduce モデル適用の定量的な評価が主題である．

MapReduce フレームワーク上で計算科学アプリケーションで標準的に使われるデータフォーマットに対しての透過的な並列アクセスを支援する既存研究もある．SciHadoop [14] では NetCDF データに対してのクエリ問い合わせを実現し，SciMATE [15] では MapReduce 風の API で NetCDF，HDF5 ファイルへの透過的なアクセスを提供し，Hadoop-BAM [16] ではゲノムデータフォーマットである BAM ファイルを透過的に MapReduce で扱えるように，BAM ファイルと KV の変換を暗黙的に行う．このような標準データへの透過的アクセス機能をワークフロー管理システムで提供することで，より容易にデータ処理ワークフローが記述できると考えている．

MapReduce による計算科学アプリケーションワークフローの構築は，データ解析 [17]，分子動力学 [18]，生物情報学 [2], [3] などの各分野で行われている．本研究では特定分野ではなく，計算科学アプリケーション全般についてのワークフローの調査を行い，全般的に要求される機能の MapReduce 処理系への実現を目指している．

7. まとめ

我々は大規模並列システムで実行されている計算科学アプリケーションのワークフロー実行パターンを抽出し，MapReduce モデルにてワークフローを構築するための機能要件を精査した．その機能要件を満たすべく，我々が開発しているスケーラブルな MapReduce 処理系 K MapReduce に追加機能として実装した．レプリカ交換分子動力学法シミュレーション，ゲノム変異解析アプリケーションのワークフローを MapReduce として実装し，評価を行った．MapReduce を用いない既存実装と比較した結果，分子動力学シミュレーションではコード量が増加したものの，タスク管理の効率化のため実行時間の改善を確認した．ゲノム解析アプリケーションではコード量は同程度であり，IO 量削減のため実行時間の改善を確認した．

今後の課題としては，機能要件に挙げたオンメモリ処理を行いつつ耐故障性を向上する機能，及び，数千〜数万ノード規模での Map/Reduce タスク負荷分散機能の実現がある．また，他分野のアプリケーションワークフローへの適用を考えている．現在想定しているアプリケーションは，データ同化手法を用いた局地的豪雨予測等の気象シミュレーションと，X 線自由電子レーザーの回折像解析の 2 つである．前者では予測精度の向上のために，500〜1000 の気象シミュレーションを計算し，その結果を元に局所アンサンブル変換カルマンフィルタを用いたデータ同化を行う，一連の処理を繰り返し実行する．REMD と同様なワークフローとなることが予想されるが，1 ステップあたり 100GB 強のシミュレーション結果のフィルタリングを行う点や，シミュレーション・フィルタリング・データ同化の各処理で要求される並列度が異なる点で解くべき課題がある．後者では 1 日最大 500 万枚生成される回折像からゴミデータを除いた有効回折像 100 万枚程度を抽出し，さらに 1 万枚程度の代表回折像を抽出した後に有効回折像と全対全で相関計算を行う．データ抽出も相関計算も各々は独立性が高く並列して実行できるため，Map タスクとして容易に実装できと考えられるが，大規模並列でのスケーラビリティや負荷分散，IO 競合回避等の課題がある．

謝辞 本論文の結果（の一部）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す．REIN の実装・利用方法についてご教授いただいた，開発者の理化学研究所生命システム研究センターの宮下尚之様，杉田有治様に感謝いたします．NGS Analyzer の実装・利用方法についてご教授いただいた，東京大学情報理工学系研究科コンピュータ科学専攻の玉田嘉紀様，および開発者の理化学研究所統合生命医科学研究センターの藤本明洋様に感謝いたします．

参考文献

[1] Dean, J. and Ghemawat, S.: MapReduce : Simplified Data Processing on Large Clusters, *Communications of the ACM*, Vol. 51, No. 1 (2008).
[2] Langmead, B., Schatz, M. C., Lin, J., Pop, M. and Salzberg, S. L.: Searching for SNPs with cloud computing, *Genome Biology*, Vol. 10, No. 11 (2009).
[3] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M. and DePristo, M. A.: The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data, *Genome Research*, Vol. 20, No. 9, pp. 1297–1303 (2010).
[4] Matsuda, M., Maruyama, N. and Takizawa, S.: K MapReduce: A Scalable Tool for Data-Processing and Search/Ensemble Applications on Large-Scale Supercomputers, *IEEE Cluster 2013 Conference* (2013).
[5] 計算科学ロードマップ中間報告書 (2013 年 9 月)：`http://hpci-aplfs.aics.riken.jp/`.
[6] Apache Hadoop: `http://hadoop.apache.org/`.
[7] Plimpton, S. J. and Devine, K. D.: MapReduce in MPI for Large-scale graph algorithms, *Parallel Computing*, Vol. 37, No. 9, pp. 610–632 (2011).
[8] Miyazaki, H., Kusano, Y., Shinjou, N., Shoji, F., Yukokawa, M. and Watanabe, T.: Overview of the K computer System, *FUJITSU Scientific & Technical Journal*, Vol. 48, No. 3, pp. 255–265 (2012).

[9] REIN: http://www.islim.org/application_m_e.html#M2.
[10] Sugita, Y., Miyashita, N., Li, P.-C., Yoda, T. and Okamoto, Y.: Recent Applications of Replica-Exchange Molecular Dynamics Simulations of Biomolecules, *Current Physical Chemistry*, Vol. 2, No. 4 (2012).
[11] NGS Analyzer: http://www.csrp.riken.jp/application_d_e.html#D2.
[12] Fujimoto, A., Nakagawa, H., Hosono, N., Nakano, K., Abe, T., Boroevich, K. A., Nagasaki, M., Yamaguchi, R., Shibuya, T., Kubo, M., Miyano, S., Nakamura, Y. and Tsunoda, T.: Whole-genome sequencing and comprehensive variant analysis of a Japanese individual using massively parallel sequencing., *Nature Genetics*, Vol. 42, No. 11, pp. 931–936 (2010).
[13] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J. and Fox, G.: Twister: a runtime for iterative MapReduce, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pp. 810–818 (2010).
[14] Buck, J. B., Watkins, N., LeFevre, J., Ioannidou, K., Maltzahn, C., Polyzotis, N. and Brandt, S.: SciHadoop: Array-based query processing in Hadoop, *International Conference for High Performance Computing Networking Storage and Analysis (SC '11)* (2011).
[15] Wang, Y., Jiang, W. and Agrawal, G.: SciMATE: A Novel MapReduce-Like Framework for Multiple Scientific Data Formats, *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pp. 443–450 (2012).
[16] Niemenmaa, M., Kallio, A., Schumacher, A., Klemelä, P., Korpelainen, E. and Heljanko, K.: Hadoop-BAM: Directly manipulating next generation sequencing data in the cloud., *Bioinformatics*, Vol. 28, No. 6 (2012).
[17] Ekanayake, J., Pallickara, S. and Fox, G.: MapReduce for Data Intensive Scientific Analyses, *2008 IEEE Fourth International Conference on eScience* (2008).
[18] Tu, T., Rendleman, C. A., Borhani, D. W., Dror, R. O., Gullingsrud, J., Jensen, M. O., Klepeis, J. L., Maragakis, P., Miller, P., Stafford, K. A. and Shaw, D. E.: A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories, *International Conference for High Performance Computing Networking Storage and Analysis (SC08)* (2008).