Vol.2016-OS-138 No.10
2016/8/8

ヘテロジニアスマルチコアにおける 木構造処理MapReduceアプリケーションの高速化

本田 舜[†1]　佐藤 未来子[†1]　並木 美太郎[†1]

概要：MapReduce フレームワークを用いた木構造処理は再帰処理の難しさなどから研究例が少なく，特にヘテロジニアスマルチコアでの MapReduce による実装例は確認できていない．そこで，本研究ではMapReduce による分枝限定法アルゴリズムを用いて，ヘテロジニアスマルチコア向けのフレームワーク上へ実装を行う．様々な高速化手法を試み，ヘテロジニアスマルチコアで木構造処理を行う際の高速化の指針を得る．シーケンシャルな処理をホストマシンで処理し，並列化が可能な処理はアクセラレータで行うことで実行特性に沿った処理が可能となるため，ホストマシン上で木の生成を行い，いくつかのブロックに分割してアクセラレータへ転送し探索を行う．さらに，ホストマシン上での木の生成・転送とアクセラレータ上での探索をオーバラップさせることで，転送待ち時間の削減を目指す．提案手法を Intel Xeon Phi 向けの MapReduce フレームワークへ実装を行い評価を行ったところ，木の生成処理の分担を行った場合に，アクセラレータ上と比較して生成処理で約５.4%，転送も含めた場合で約 32.6%の処理時間となった．また，木の生成・転送および探索のオーバラップを行った部分では約 86.0%の処理時間での探索を実現した．今後の課題として，二分木以外にも対応した完全な分枝限定法アルゴリズムの実装などが挙げられる．

Speeding Up of the MapReduce Applications of Tree Structure Processing in Heterogeneous Multi-core

Shun Honda[†1]　Mikiko Sato[†1]　Mitaro Namiki[†1]

1. はじめに

並列処理は，科学計算やビッグデータ解析をはじめ様々な場所で利用されており，システムの高速化における重要な要素となっている．特に，データ処理という分野においては Google が 2004 年に発表した MapReduce[1] という分散処理フレームワークが広く用いられている．MapReduce は通常では複雑になりやすい並列処理の設計が容易になるという特徴があり，大きく Map 処理と Reduce 処理という 2 つの段階から構成され，それぞれがワーカーに分散されて並列に処理されることにより並列化を可能としている．

MapReduce 実装は複数のマシンを連結したクラスタマシン上で処理を行う Apache Hadoop[2] が有名であるが，マルチコア CPU とメニーコアアクセラレータが混在するヘテロジニアスマルチコア上で実行される実装も複数存在する．アクセラレータ上には，CPU と比較して個々の処理性能が低いコアが複数存在しているため，それを利用してクラスタマシンを利用する場合よりも高い並列度で処理を行うことが可能となる．ヘテロジニアスマルチコア上での実装としては，NVIDIA が提供している GPU 向けの統合開発環境である CUDA 上で実装された Mars[3] や，Reduction-based MapReduce という手法を用いて実装された Jupitor[4] など，GPGPU 技術を利用した GPU 上での研究が多い．GPU 以外のアクセラレータでは，Intel Xeon Phi を用いた MapReduce の研究としては MrPhi[5] などが存在する．

一般的に利用されている MapReduce アプリケーションは数値演算を行うものとデータ処理を行うものに大別される．数値演算を行うアプリケーションには，モンテカル

†1 現在，東京農工大学
Presently with Tokyo University of Agriculture and Technology

© 2016 Information Processing Society of Japan

ロ法や K-means 法など，複数の数値に対して同じ演算を行い，それを集約して解を得るというものが多い．また，データ処理についても，テキストファイルを分割してそれぞれに同じ処理を行う，指定した Web ページからリンクを張られている各々の集客データを集約するなど，それぞれが互いに独立しているデータの処理に利用されることが多い．それに対して，漸化式や木構造の処理など，再帰処理を行う必要のあるデータに適用することは難しく，アプリケーションの実装例も少ない．

2. 課題

MapReduce では様々なアルゴリズムが研究されているが，木構造処理を行うアプリケーションの実装は少数である．特に，ヘテロジニアスマルチコア上での木構造処理 MapReduce アプリケーションとしては現時点で確認されていない．その理由として，木構造処理の MapReduce アプリケーションとしての課題，およびヘテロジニアスマルチコア上での木構造処理の課題という 2 点が挙げられる．

それぞれ Map 処理や Reduce 処理では，すべてのワーカーが独立した処理を行うということが条件となるため，基本的には互いに依存関係のないデータに対する処理を得意とし，他のワーカーの結果を利用して次の結果に反映するような処理には不向きである．また，MapReduce 処理を階層ごとに複数回行う必要がある場合では，MapReduce 処理とは別にデータ転送などの冗長時間が生じてしまうという問題点が存在している．そのような理由から，現状では木構造処理に焦点を当てた研究は少ない．

さらに，既存のアルゴリズムはクラスタマシン上へ実装されることが多く，特に木構造処理の MapReduce アルゴリズムはヘテロジニアスマルチコアでの実装例が存在していなかったため，テロジニアスマルチコアの実行特性に適しているかという指針についても判断ができていない状態となっている．例えば木構造処理のアルゴリズムのうち，最適化問題などの数値演算に近い性質の処理を行う場合，アクセラレータ上での実行特性に適している可能性がある．そのため，木構造処理の MapReduce アプリケーションをヘテロジニアスマルチコア上へ実現し，その実行適性を精査することで，高速化の方針決定のための一端とする必要がある．

3. 目標

本研究では，2 章で提示した課題を解決するべく，木構造処理の MapReduce アプリケーションをヘテロジニアスマルチコア上で実装し，その高速化を行うことで高速化の指針を得ることを目標とする．木構造処理には先行研究の分枝限定法 MapReduce アルゴリズム [6] を使用し，いくつかの処理をヘテロジニアスマルチコアへ最適化することで高速化を目指す．また，フレームワークについても既存の Intel Xeon Phi 向けの MapReduce 実装である MrPhi を使用し，フレームワークの改変は最小限に留め，アプリケーションの構成による高速化を目指す．

この目標を達成するために，ヘテロジニアスマルチコアへの木構造処理 MapReduce アプリケーションの実装を目指す．具体的に，ホストマシンとアクセラレータ間で転送可能な木構造の実現，MapReduce 処理における入出力に用いられる $< Key, Value >$ 形式のデータの検討といったことを行い，メニーコアマシン上で最適に動作する設計方針の確立を目指す．そのために，フレームワークに適用するようにアルゴリズムの修正も行う．また，実装したアプリケーションの性能評価を行い，メニーコアマシン上での MapReduce による木構造処理の適性を考察する．

また，実装した分枝限定法アプリケーションに対して，様々な方法により高速化を行う．前述のように，木構造処理は連続した階層のあるデータを用いるため，独立した処理を並列に行う MapReduce フレームワーク上で高速に実行することは難しい．そのため，ノードのメモリ配置や入出力データの管理方法というように様々に検討することで高速化を図る．また，マルチコアマシンであるホストマシンとアクセラレータである Xeon Phi において，探索など高並列に処理することが可能な箇所はメニーコアマシンで実行し，木の生成など並列化の行えないシーケンシャルな処理をホストマシンで行うといったように，各マシンの特性に合致した処理を割り当てることで高速な処理を実現する．さらに，木の生成と探索をそれぞれのマシンでオーバラップして実行し，アクセラレータ上では対象のノードを受け取り次第探索するようにすることにより，ヘテロジニアスマルチコアを構築する二種類のマシンを用いることを活かした実装も目指す．それぞれの修正箇所ごとに動作性能を検証し，提案する手法のうちから高速化の指針を得る．

4. システムモデル

4.1 MapReduce モデル

まず，一般的な MapReduce プログラミングモデルについて述べる．

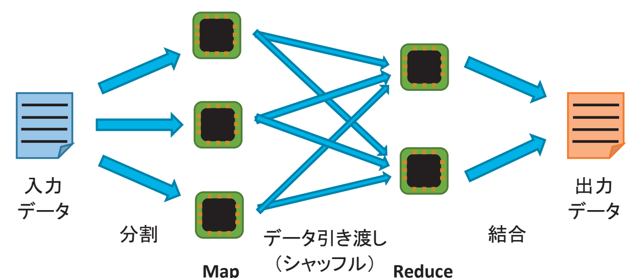

図 1 一般的な MapReduce プログラミングモデル

処理の流れとしては，まず入力するデータを分散させる

数もしくはそれ以上に分割し Map 処理を行うワーカーへと配布する．Map ワーカーではそれぞれのデータに対して同じ処理を行い，$< Key, Value >$ の形で出力する．その後，同じ Key 値の結果が同じ Reduce ワーカーに配置されるようにデータのシャッフルを行い，受け取ったワーカー上で Reduce 処理を実行する．最終的に，Reduce 処理の結果をすべて 1 つに集約し，MapReduce 処理結果として出力する．そのため，最終的な出力結果もすべて $< Key, Value >$ という形となる．これらの MapReduce ワーカーだけでなく，入力データの作成や各ワーカーへの分配，ワーカーの処理の監視，最終的な出力データの集約などを行うマスターワーカーを定める場合もある．

続いて，ヘテロジニアスマルチコアでの MapReduce 処理モデルを図 2 に示す．マスターワーカーはアクセラレータ上で動作し，ホストマシンとの入出力データのやり取りを行う．ホストマシン上に置かれているデータを生成するスレッド数に分割してアクセラレータへ転送し，受け取ったマスターが各ワーカーに配分する．各ワーカーは入力データを受け取り次第 Map 処理を開始し，マスターにより実行中かどうかを監視する．そして，それぞれのワーカーが MapReduce 処理を終えたとき，マスターがすべての結果をホストマシンへ転送し，集約された結果を出力する．

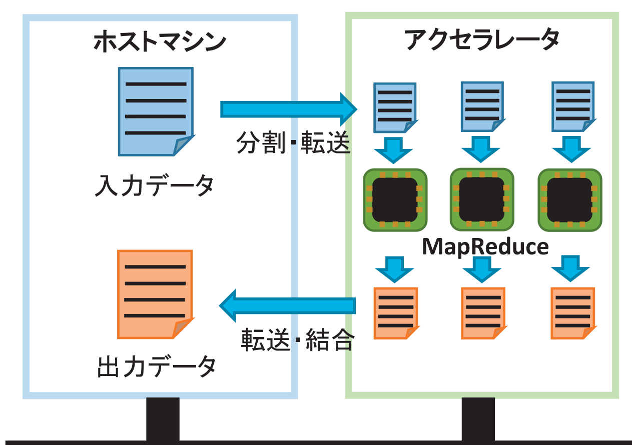

図 2　ヘテロジニアスマルチコアにおける MapReduce プログラミングモデル

4.2　MrPhi

本研究ではヘテロジニアスマルチコアとして，ホストマシンとなるマルチコア CPU マシンに Intel Xeon Phi が接続されている環境を使用する．そのため，使用する MapReduce 実装として Xeon Phi を対象に実装された MrPhi を使用している．MrPhi のタスクモデルを図 3 に示す．

MrPhi では，ホストマシンで 2 つ，Xeon Phi 上に 1 つのプロセスを実行する．それぞれの役割は以下のとおりとなる．

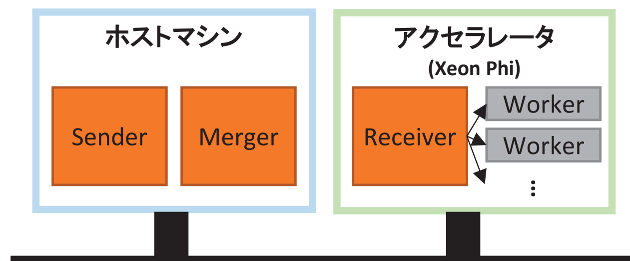

図 3　MrPhi のタスクモデル

ホストマシン

- Sender
  入力データの送信や結果の取りまとめを行う
- Merger
  MapReduce 処理結果を Receiver から受け取り集約やソートを行う

Xeon Phi

- Receiver
  実際に MapReduce 処理を行うワーカーの生成および実行管理
  具体的に，Sender から受け取ったデータを基にワーカーに MapReduce 処理を実行させ，すべてのワーカーの終了を待ってから完了したデータを Merger へ送信する．

5.　木構造処理 MapReduce アルゴリズムの設計

5.1　分枝限定法アルゴリズム

本研究では，木構造処理として先行研究 [6] の分枝限定法アルゴリズムをヘテロジニアスマルチコア上に実装している．分枝限定法は，探索により最適化問題を解決するための汎用アルゴリズムである．このアルゴリズムでは MapReduce 処理を繰り返すことにより最適解を求める．本アルゴリズムは Hadoop 上への実装へテロジニアスマルチコア上でより高速に処理できるように修正を行ったものを使用する．

修正を加えた分枝限定法アルゴリズムの処理の流れを図 4 に示す．初めに，ルートノードを入力データとして用意し，MapReduce 処理を実行する．Map 処理では与えられたノードから予め設定しておいたステップ数だけ木を探索し，探索でたどり着いたすべてのノードとそのノードの時点での探索中の解 (途中解) のペアを出力する．Map 処理が全て完了したところで，その出力結果を受け取り Reduce 処理を行う．ステップ数のイメージを図 5 に示す．

Reduce 処理は，受け取った途中解と現在探索している深さまでの最適解 (暫定解) を比較する探索対象のノード

© 2016 Information Processing Society of Japan

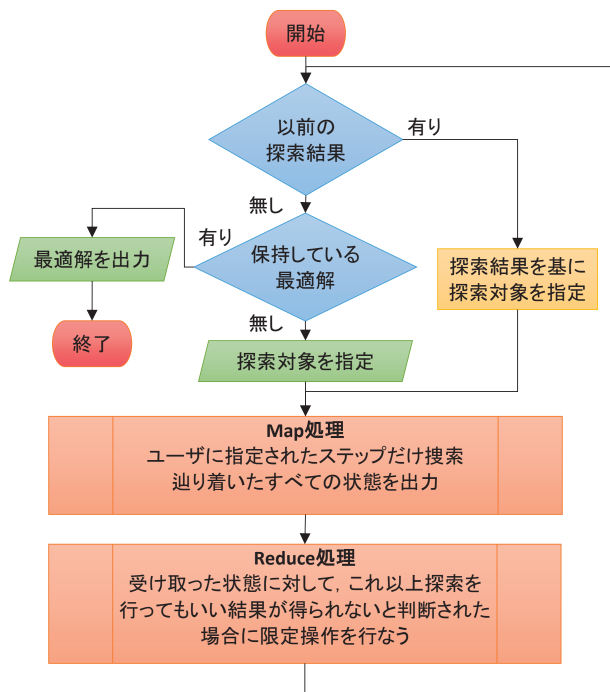

図 4　分枝限定法 MapReduce アルゴリズム

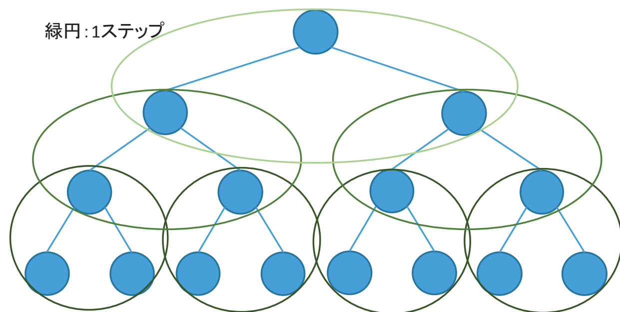

図 5　木構造に対するステップ数のイメージ

それぞれに最適解となり得る期待値を設定したポテンシャルという要素を算出し，途中解のポテンシャルが暫定解のポテンシャル以下と判断された場合にそのノードを枝刈りし，最適解となる可能性のある枝を保持しているノードのみを出力する．そして，出力結果から暫定解を更新し，次の入力データを生成する．新たな入力データを用いて一連の操作を繰り返し，探索可能なノードが無くなった時点で保持している暫定解が最適解となる．

本研究では，ポテンシャルについては「途中解に以降のノードすべての値を足した結果が暫定解以下となる場合」もしくは「解とは別に設定している制約条件の値が許容範囲を超えると判断された場合」に枝刈りを行うとする．また，元となったアルゴリズムのクラスタマシンによる並列化と異なり，メニーコア上で高並列な処理が可能となるため，元のアルゴリズムの「探索対象ノード＞並列数 の場合，探索対象ノードから並列数だけ，ポテンシャルの高い順に探索を開始する」という項目については考慮しない方針とした．

5.2　ヘテロジニアス空間上の木構造処理 MapReduce アプリケーションの高速化

5.1 節で示した分枝限定法アルゴリズムをヘテロジニアスマルチコア上で実現するにあたり，設計したシステムモデルを図 6 に示す．

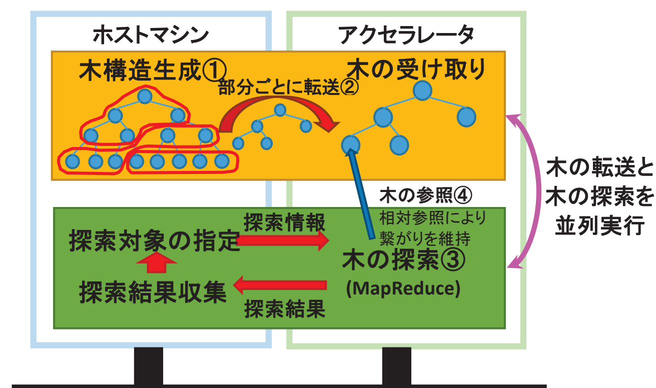

図 6　木構造処理 MapRecude アプリケーションのシステム

探索の流れとしては，まず探索を行うための木を生成し (図 6 内 ①)，それを生成された部分から逐一転送する (②)．木の生成と平行してアクセラレータ上で MapReduce 処理を実行し，受け取った木の部分から探索を開始する (③)．その際，木は相対参照により子ノードを設定しているため，異なるマシン上でも探索が可能となる (④)．このような構造にすることにより，木の生成・転送による待機時間を削減し，全体の処理時間の削減を可能としている．

本システムにおいて，高速化を達成するために実装した機能は以下のとおりである．

(1) ホストマシンとアクセラレータ間で転送可能な木構造クラス
(2) 木の生成・転送と木の探索のオーバラップ

(1) について，木の生成はシーケンシャルな処理となることにより，非並列な処理はアクセラレータよりもマルチコア CPU のほうが実行特性が適しているため，ホストマシン上で木を生成し，アクセラレータ上のメモリへ転送してから高並列で探索することで実行特性を最適化した処理が可能となる．その際，探索はルートノードから Breads-first で行われるため，図 7 のように木のメモリ配置を同じ深さのノードが並ぶように配置することで探索時のメモリアクセスについても効率化している．

さらに，(2) のオーバラップを実現するにあたり，木の

© 2016 Information Processing Society of Japan

メモリをブロックに分割して転送することで転送しながら探索することを可能としている．その際，分割方法は図 7 のように上から指定したメモリブロックのサイズごとに区切り，すべてのブロックが同じサイズになるようにしている．このブロックサイズは変更可能として，転送する木のサイズにより調整が行えるようにする．

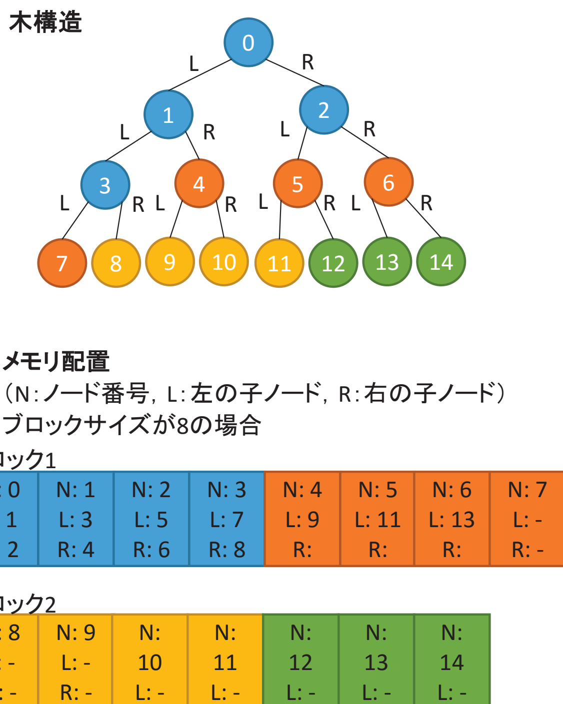

図 7　メモリブロックの配置方法

また，(2) については，木の生成が完了してから転送し，転送が終わってから探索するのではなく，ホストマシンは木を生成しつつメモリブロックが埋まったところから転送し，アクセラレータではメモリブロックを受け取り次第探索を行うことで，各マシンの待機時間の削減を行った．使用する MapReduce 実装である MrPhi が提供するタスクに加えて，木の転送用に新たに 2 つのタスクを生成している．MrPhi に新たに追加されたタスクを含むタスクモデルを図 8 に示す．

具体的に，Sender および Receiver 上にスレッドを追加した構成としている．Sender 上のスレッドとして生成される TreeSender では，Sender の実行とは独立して木の生成を行い，メモリブロックが埋まった部分から逐一転送を開始する．転送先は Receiver 上で生成された TreeReceiver では，こちらも Receiver 本体の実行とは独立して木の受信を行い，プロセス中に木の展開を行う．そして，Receiver 本体における探索開始前に転送が完了しているかを監視し，完了したノードから探索を行う．

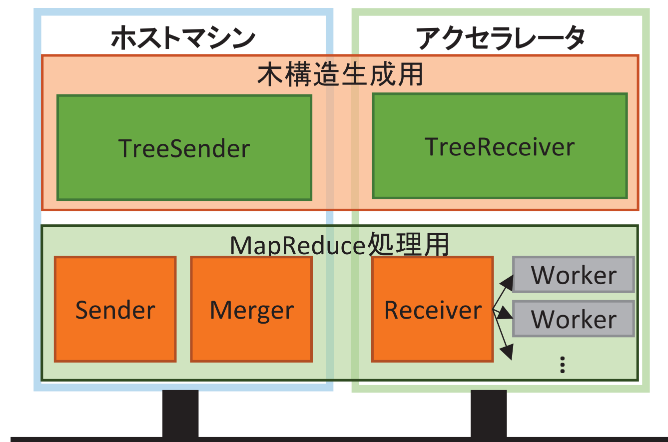

図 8　スレッドを追加したタスクモデル

6. 実装

Intel Xeon E5-1620 3.70GHz の CPU および 16GB のメモリを搭載したホストマシンに対して，アクセラレータである Intel Xeon Phi 5110P を PCI Expless で接続したヘテロジニアスマルチコアにおいて実装を行う．

6.1 木構造クラス

本実装が対象とする最適化問題の種類として，ナップサック問題などが該当するバイナリ変数を用いる問題とする．バイナリ変数のみを用いる場合，すべての部分問題は 0 か 1 により解答可能となる．なので，部分問題を基に生成する木は二分木のみとする．

木構造の各ノードは単方向リストのような構造とし，それぞれのノードはベルとなる数値，子となる二つのノードの情報，および解や制約条件に利用される数値を保有する．また，木のノード間を相対参照するために，全ノードに対して生成順に番号を付与し，子ノードの番号を所持することにより参照を行う．前述のブロックのメモリ領域について，転送の際に先頭アドレスを配列に保持しておき，以下のコードにより参照先ノードのポインタを取得する．

N: 参照先のノード番号
B: ブロック内のノード数
arr: 全ブロックの先頭アドレスを格納した配列
%: 除算の余りを求める演算子

arr[N / B][N % B]

6.2 MapReduce 処理の実装

MrPhi 上へ木構造処理 MapReduce アプリケーションを実装するにあたり，いくつかの処理を実装する際にフレームワークについても修正を行う必要が生じたため，Reduce

© 2016 Information Processing Society of Japan

処理の追加，入力データが少ない場合のワーカー起動の最適化，および MPI 通信のスレッド対応化といったことを行った．

Map 処理において探索対象のノードがすでに転送されているかを調べる際に，転送されているノードの総数を用いて判断を行う．具体的に，木構造は図 7 のように Breads-first で配置および転送されているため，転送されているノード数を N とすると，番号 $N-1$ までのノードの転送が完了していると判断できる．そのため，TreeReceiver はメモリブロックを受け取る際にブロックのサイズから転送されたノード数を導出して変数に保管し，Map ワーカーはこの変数を監視する．

7. 評価実験

本章では，6 章で実装したアプリケーションに対して様々な性能評価を行い，その結果についてそれぞれ考察する．

7.1 評価内容

本研究の評価として，以下の項目の性能を計測し，結果より高速化という部分での指針について考察する．ここで，各評価項目において特に記載しない場合は表 1 のパラメータを使用する．

(1) 分散数に対する性能評価
(2) 実行特性を配慮した処理性能評価
(3) 木の生成，転送と探索のオーバラップに対する性能評価

(1) について，Map 処理による木の探索部分について，並列性を高めて実行することによる有効性を示すために，分散させる数を変えることによる探索速度を調べる．木の生成，転送と探索のオーバラップをせず，Reduce 処理による限定操作を行わずに Map 処理のみを実行する場合の探索時間のみを，分散数を変えて計測する．

(2) について，シーケンシャルな処理と並列性の高い処理を各マシンの実行特性に合わせて分担することによる性能向上を確認するために，本手法を適用した処理を両マシン上で実行し，転送時間も考慮して処理速度を計測する．その際，比較対象とするのは以下の二箇所とする．

a) 木の生成場所
b) 入出力データの管理場所

(3) について，オーバラップによる高速化への有効性を確かめることを目的とし，両マシンで連携を行った場合および各マシン単体で動作させた場合でそれぞれオーバラップの有無による探索速度を計測し，処理速度に対する効果について考察を行う．両マシンの連携時，ホストマシン単体，および Xeon Phi 単体において，それぞれオーバラップを行う場合と行わない場合の二通りの実行時間を計測する．また，実行時間は以下のように処理内容ごとに区切って計測を行う．

- 木構造の生成および転送時間
- MapReduce 処理のためのクラス生成などの準備時間，
- MapReduce 処理本体
- MapReduce 処理結果をホストマシンへ転送する時間
- 結果を基に次の入力データを生成する時間
- 最終的な最適解の決定を行う時間

表 1 標準の評価環境

木のサイズ	深さ 25 (約 1.25GB)
MapReduce 処理の並列数	ホストマシン: 240 Xeon Phi: 8
メモリブロックのサイズ	16MB
探索ステップ数	2
実行時間の計測回数	5

7.2 評価結果

7.1 節で示した内容に基づき，評価を行った結果を示す．ここで，7.1 節における項目のうち (1) は図 9，(2) は図 10 および図 11，(3) は図 12 となる．

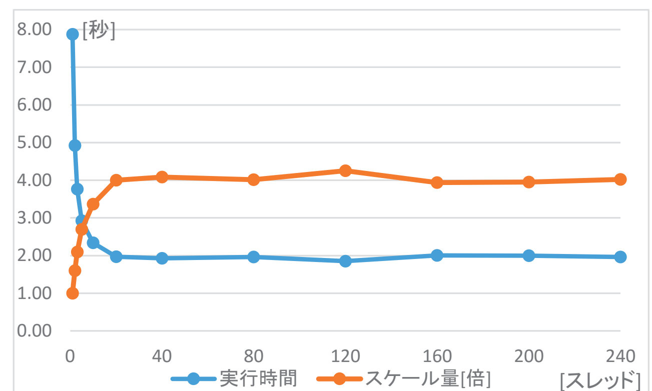

図 9 (1)Map 処理のみを実行した場合の並列数に伴う実行速度

7.3 考察

7.3.1 (1)Map 処理のみを実行した場合の並列数に伴う実行速度

結果より，並列数が 20 程度まではワーカー数に伴って探索速度が向上しており，並列数を高めることによる有効性が認められる．具体的に，並列処理を行わない場合と比較して，2 ワーカーによる並列の場合に約 1.60 倍，3 ワーカーによる並列の場合に約 2.09 倍の探索速度となっており，ワーカー数が 20 の時点で約 4.00 倍の探索速度を実現している．分散数が 20 を超えてからは実行時間がほぼ横ばいとなっているが，これは並列化による速度向上とワー

© 2016 Information Processing Society of Japan

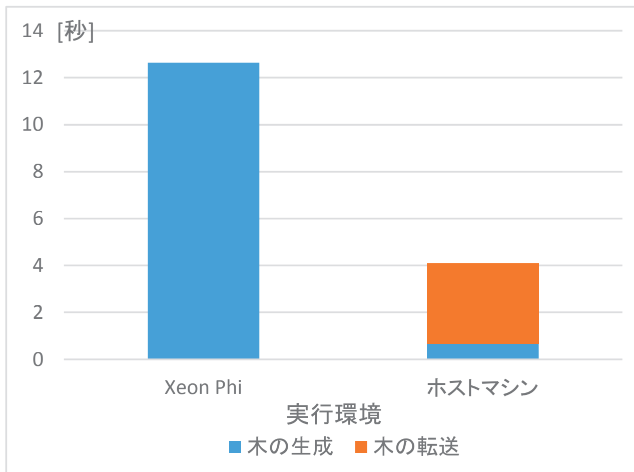

図 10 (2) a) 場所に伴う木の生成時間

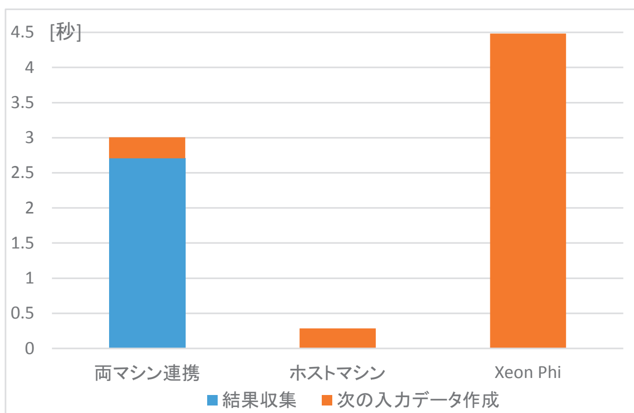

図 11 (2) b) 入出力データの管理場所に伴う処理時間

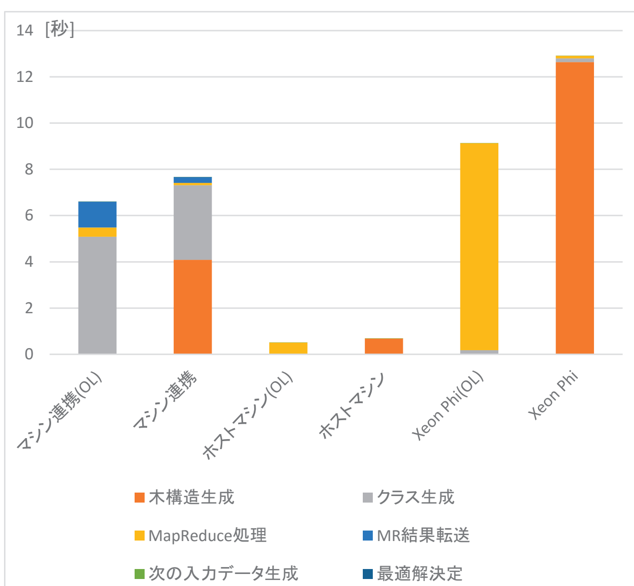

図 12 (3) 木の生成，転送と探索のオーバラップに対する性能評価結果

カーの生成やデータの分配，回収にかかる探索外の時間が均衡するためであると考えられる．

7.3.2 (2) 実行特性を配慮した処理性能評価

(2) のうち，a) 場所に伴う木の生成時間について，木の生成のみで比較した場合，ホストマシンで生成することにより Xeon Phi 上と比較して約 5.4%の生成時間となり，転送時間を含めた場合でも約 32.6%の時間で処理が可能であることが確認できる．

また，(2) のうち，b) 入出力データの管理場所に伴う処理時間について，オーバラップを行わなかった場合と比較して約 86.0%の処理時間による探索が行われていることが確認できる．両マシンの連携時とホストマシン単体時の入力データ作成時間が近いことから，両マシン連携時の増加分はほとんど転送時間であると推測できる．そのことにより，入力データ作成の 9.2 倍の時間が転送に費やされていることがわかる．

それぞれのいずれの場合でも，ホストマシンと Xeon Phi で連携を取るようにアプリケーションを設計することにより，Xeon Phi 単体で実行した場合と比較して高速に処理可能であることが確認された．このことにより，本設計方針はメニーコアマシン上で複雑な処理を行う際の高速化の指針となり得たといえる．しかし，どちらの評価においても共通して，転送時間が長いことによりマシンの連携による効果が表れ辛くなっている．通信方法により転送時間が変わることが考えられるため，今後はこの部分についても精査する必要がある．

7.3.3 (3) 木の生成，転送と探索のオーバラップに対する速度性能

結果より，両マシンの連携時において，オーバラップ (OL) を行わなかった場合と比較して，行った場合は約 86.0%の処理時間による探索が行われていることが確認された．各マシン単体で行った場合も，ホストマシンにおいて約 75.0%，Xeon Phi 上で約 71.0%の処理時間による探索を実現した．

本評価では，Xeon Phi による実行時に対しては優位性が見られたものの，ホストマシンでの結果が両マシンで連携を行う場合よりも高速に探索される結果となった．考えられる理由として，木構造処理がデータ処理系の実行特性であるため，数値演算に対して高速に処理できるように設計されている Xeon Phi 上での実行時に，高並列性により期待される実行速度が出なかったことによるということが挙げられる．

7.3.4 考察

結果の全体を通して，本研究における提案手法を用いることにより実行時間の削減を行うことができたため，いずれも高速化の指針となり得たといえる．しかし，(2) および (3) において転送時間が大きなオーバヘッドとなっており，転送時間の削減が今後の課題となる．解決するための案として，転送する木構造クラスや入出力データのサイズを減らすことによる転送するメモリサイズの削減といった

ことが挙げられる．特に木構造クラスは，ノード番号用の unsigned int 型の変数，および子ノードを指定するための unsigned int 型の変数二つの他に，最適化問題用の条件値となる値が int 型で二つ定義されている．対象とする最適化問題によっては 4 バイトを使わずに表現可能であるため，問題により short int 型や char 型を使用することによりメモリサイズを削減することが可能となる．転送するメモリサイズが 1/2 となった場合に，転送の準備などによるオーバヘッドを考慮しなければ転送時間も 1/2 となることが期待できる．

8. まとめと今後の課題

本研究では，Intel Xeon Phi 上で動作する MapReduce フレームワークを用いた木構造処理アプリケーションの設計および高速化手法を提案し，木の生成処理の分担を行った場合にアクセラレータ上で生成する場合と比較して，生成処理で約 5 .4%，転送も含めた場合で約 32.6%の時間による処理が可能となった．また，木の生成，転送および探索のオーバラップを行った部分では，オーバラップを行わなかった場合と比較して約 86.0% の処理時間での探索を実現した．

今後の課題として，2 分木以外の複雑な木構造の探索についても対応することによる完全な分枝限定法アルゴリズムの実装や，$\alpha\beta$ 法や遺伝的プログラミングなど，他の木構造処理 MapReduce アルゴリズムへの対応といったことが挙げられる．

参考文献

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, Vol. 51, No. 1, pp. 107–113, January 2008.
[2] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.
[3] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: A mapreduce framework on graphics processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pp. 260–269, New York, NY, USA, 2008. ACM.
[4] Ran Zheng, Kai Liu, Hai Jin, Qin Zhang, and Xiaowen Feng. Accelerate mapreduce on gpus with multi-level reduction. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, Internetware '13, pp. 10:1–10:8, New York, NY, USA, 2013. ACM.
[5] Mian Lu, Yun Liang, Huynh Phung Huynh, Zhongliang Ong, Bingsheng He, and R.S.M. Goh. Mrphi: An optimized mapreduce framework on intel xeon phi coprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 26, No. 11, pp. 3066–3078, Nov 2015.
[6] 中川遼. Mapreduce による列挙木探索手法の提案と評価. 第 10 回情報科学ワークショップ, pp. 1–9, sep 2014.

© 2016 Information Processing Society of Japan