Vol.2012-HPC-135 No.10
2012/8/1

ワークスティーリングフレームワークにおける集団通信機能

松井 健[1,a)]　平石 拓[2,b)]　八杉 昌宏[3,c)]　馬谷 誠二[1,d)]

概要：分散メモリ環境上で高速な並列計算を実現するためには，各計算ノードが計算に必要なデータを適切な形で参照できるようにするための通信機能が重要である．しかし，我々が提案しているワークスティーリングに基づく負荷分散フレームワーク Tascell では，分散メモリ環境上でユーザが取り扱うことのできる通信機能が限られているため，一部のアプリケーションの並列化性能が伸び悩む原因となっていた．本研究では，Tascell フレームワークと MPI ライブラリの集団通信機能を使用し，動的負荷分散を行いながら多体シミュレーションを行う Tascell プログラムを実装した．性能評価を行った結果，従来手法を用いた Tascell プログラムよりも大幅に性能が改善されることが確認できた．

Collective Communication Functionality for a Work-Stealing Framework

Ken Matsui[1,a)]　Tasuku Hiraishi[2,b)]　Masahiro Yasugi[3,c)]　Seiji Umatani[1,d)]

Abstract: In order to realize fast parallel computing in distributed memory environments, communication functionality is important that enables each computational node to properly refer to data required for its computation. However, in the work-stealing-based load-balancing framework Tascell that we have proposed, limited communication functionality can be used by users in distributed memory environments, which hinders parallel performance improvements of some applications. In this research, we implemented Tascell programs that perform n-body simulation with dynamic load balancing, employing the Tascell framework and the collective communication functionality of an MPI library. By conducting performance evaluations, we confirmed that these Tascell programs achieved much better performances over previously implemented ones.

1. はじめに

マルチコアプロセッサなどを含む並列計算環境が一般的になるに伴い，並列言語の重要性が高まっている．Cilk[1] はそのような言語の 1 つであり，多数の論理スレッドを生成して最古優先のワークスティーリングを行うことにより，不規則なアプリケーションを含む多くのアプリケーションにおいて良好な負荷分散を実現する．それに対し，我々は論理スレッドフリーな並列プログラミング/実行フレームワーク Tascell [2] を提案している．Tascell ワーカは他のワーカからタスクを要求されない限り逐次計算を続けるが，タスクを要求されると一時的なバックトラックによって最古のタスク生成可能状態を復元し，本物のタスクを生成して要求元のワーカに受け渡す．この手法は，論理スレッドの生成や管理に伴うコストを不要とする，同一の作業空間の長期にわたる再利用を可能とし参照局所性を改善する，作業空間の複製を必要があるまで遅延させる，といった利点を持つ．また，Tascell フレームワークは単一のプログラムを共有メモリ環境と分散メモリ環境の双方にお

[1] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
[2] 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University
[3] 九州工業大学大学院情報工学研究院
Department of Artificial Intelligence, Kyushu Institute of Technology
[a)] kmatsui@kuis.kyoto-u.ac.jp
[b)] tasuku@media.kuis.kyoto-u.ac.jp
[c)] yasugi@ai.kyutech.ac.jp
[d)] umatani@kuis.kyoto-u.ac.jp

いて実行させることができ，PC クラスタや広域分散環境における性能評価においても，期待される性能が得られることを確認している．[3]

分散メモリ環境上で並列計算を行う場合，計算ノード上の各ワーカが計算中に必要とするデータを，ノード間で効率良く共有する手法について考えなければならない．一般には，計算の開始前に各ノードが必要とするデータをあらかじめ配置しておく，あるいは計算中にデータを持っているノードに対してオンデマンドにデータを要求する，などといった実装を行う．ワークスティーリングに基づいてノード間の動的な負荷分散を行う Tascell フレームワークでは，あるタスクの計算に必要なデータはワークスティールの発生時に送信することができるようになっている．また，タスクの計算結果はその実行の完了時に，スティール元の計算ノードに返信できるようにすることで，計算に必要なデータを各ノードが参照できるようになっている．しかし，こうしたデータ転送はワークスティールの発生とは独立したタイミングで，ユーザが自由に行うことができないため，多数の計算ノード間で共有可能なデータが存在するような特定のアプリケーションを並列化した際に，通信効率が悪く望ましい性能向上を達成できないという問題が存在した．

このような問題に対する解決策の 1 つとして，我々はブロードキャスト通信を用いて共有可能なデータをあらかじめすべての計算ノードに配置しておく実装を過去に試みた．[4] しかし，計算結果を集約する際の通信効率については考慮しなかったため，一部のアプリケーションでは期待される並列化性能を得ることができなかった．そこで本研究では，計算結果を集約する際にも集団通信を用いることによって，Tascell フレームワークを用いた分散メモリ環境におけるアプリケーション性能のさらなる向上を試みた．本稿では，Tascell フレームワークと MPI ライブラリの集団通信を用いた Tascell プログラムの実装の概要，及びその性能評価結果を示す．評価を行うアプリケーションとしては，過去の試みにおいて期待される並列化性能を得ることができなかった，多体問題における Barnes-Hut アルゴリズム [5] を選択した．

本稿の構成は以下の通りである．第 2 章では，並列言語 Tascell，及び Barnes-Hut アルゴリズムについて概説する．第 3 章では，Barnes-Hut アルゴリズムを Tascell を用いて実装する際に，既存の Tascell の枠組みではどのような問題が生じるのかを具体的に説明する．第 4 章では，本研究で実際に作成した重力多体問題のシミュレーションプログラムの概要を示し，第 5 章で性能評価の結果を示す．第 6 章で関連研究について述べ，最後に第 7 章でまとめる．

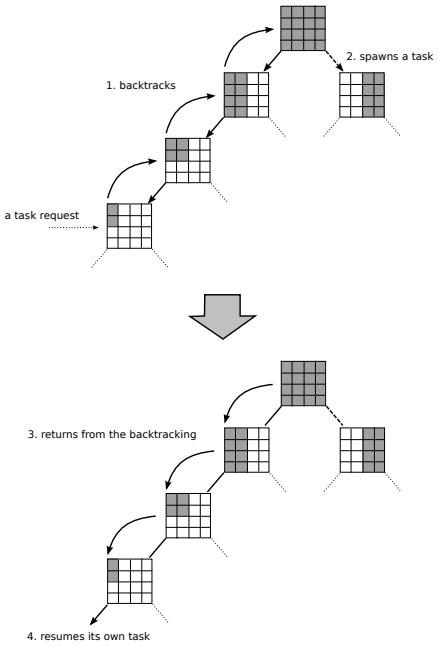

図 1 行列積 $C = AB$ の計算における，バックトラックに基づくタスクの遅延生成．ワーカが実行すべきタスクを「計算すべき行列 C の範囲」として定義し，計算範囲を再帰的に分割しながら行列計算を実行するものとした．ワーカはタスク要求を受けると過去の状態にバックトラックし，自身のタスクを分割して別のワーカに受け渡す．

Fig. 1 Lazy task spawning based on backtracking in an example of a matrix multiplication $C = AB$. We defined a task as the range of the matrix C to be calculated. The matrix calculation is performed by recursively dividing the range of the matrix. When a task is requested, the Tascell worker backtracks to the past, divides its own task and returns it to the other worker.

2. 背景

2.1 並列言語 Tascell

Tascell は，C 言語を拡張した独自の Tascell 言語と，我々の提案する並列計算手法を実現するための Tascell 実行フレームワークから成る．Tascell 実行フレームワークは，Tascell コンパイラと，分散メモリ環境において並列計算を実行する際に必要な Tascell サーバにより構成されている．

Tascell では，ある計算を行うために必要な情報を転送可能な形式にまとめたデータのことをタスクと呼び，タスクを用いて並列計算を実行する計算の主体をワーカと呼んでいる．Tascell における計算は，アイドルなワーカがタスクを実行中のワーカにタスク要求を行い，タスクを分割して負荷の分散を行うワークスティーリングに基づいて処理される．分割され，複数のワーカにより計算されたタスクの結果は，タスクを分割した元のワーカに送り返された後，計算結果が統合される．タスク及びその結果はタスクオブジェクトとしてやりとりされ，その構造はユーザが自由に定義できるようになっている．

Tascell ワーカはタスク要求を受信すると，一時的なバックトラックに基づいて計算状態を遡り，最古のタスク生成可能状態を復元して，新しいタスクを生成する．すなわち，ワーカは他のワーカからタスクを要求されると，

(1) まず過去の時点にバックトラックして計算状態を復元し，
(2) その時点でタスク要求があったかのようにタスクを分割し，
(3) バックトラックを行う前の元の計算状態に戻り，
(4) 自分の行っていた計算を再開する．

つまりワーカは，最初は常に「タスクを分割しない」ことを選択するが，他のワーカからタスク要求が生じると，過去の選択を変更したかのようにタスクを生成する (図 1)．タスクを遅延分割することにより，ワーカの作業空間の不必要な生成・複製が抑えられると同時に，同一の作業空間を再利用することによる参照局所性の改善が期待できる．また，タスクはアイドルなワーカから要求されるまで生成されないため，Cilk をはじめとしたマルチスレッド言語と異なり，多数の論理スレッドを維持・管理する必要がないといった特徴も持つ．

2.1.1 分散メモリ環境

分散メモリ環境では，Tascell サーバと呼ばれるプロセスが，各計算ノードの負荷情報の管理やノード間のタスク転送の制御を行う．Tascell における計算ノード間の通信には TCP/IP 通信が用いられており，全ての通信は Tascell サーバを介して行われる (本論文の実装で用いた MPI による通信を除く)．計算ノード上の Tascell プログラムは，起動時に接続する Tascell サーバを指定することにより，その Tascell サーバに接続された全ての計算ノード間で並列計算を行うことが可能になる．Tascell サーバは上記の役割を担うほか，ユーザインタフェースとの入出力処理や，複数の Tascell サーバを相互に接続することにより NAT 越えが必要なクラスタ間で計算を行わせることも可能となっている．

計算ノード間の通信は，実際にはノード内の特定のワーカを対象とし，ワーカ間で通信が行われる．この時に送受信されるデータはメッセージと呼ばれ，あらかじめその仕様が決められている．ノード内のワーカ及び Tascell サーバは，ネットワークから (受信専用に割り当てられたスレッドを介して) メッセージを受信すると，対応する規定の処理を実行する．例えば，タスク要求を表すメッセージを受け取ったワーカは，自身が実行中のタスクを分割し，要求元のワーカに対して分割したタスクを表現するタスクオブジェクトを自動的に送り返す．サーバ・ノード間のメッセージ通信は，全て Tascell 実行フレームワークによって自動的に処理されるようになっており，ユーザはタスクあるいは計算結果として送信したいデータをタスクオブジェクトに格納するだけでよい．

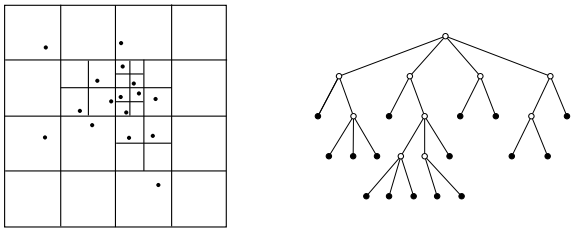

図 2 再帰的に分割された空間と，それを表す階層化された木構造．Barnes-Hut アルゴリズムは，空間をセルと呼ばれる部分空間に再帰的に分割する (簡単のためここでは平面で表している)．左図において，黒点は質点を表す．右図において，黒と白の節点はそれぞれ質点とセルを表す．

Fig. 2 An illustration of hierarchical boxing and a hierarchical tree representing the spatial structure. The Barnes-Hut algorithm recursively decomposes space into subspaces called cells (presenting in two dimensions for simplicity). In the left figure, each black dot represents a body. In the right figure, black and white nodes represent bodies and cells, respectively.

2.2 Barnes-Hut アルゴリズム

Barnes-Hut アルゴリズムは近似的な計算によって多体問題の解を高速に求めるアルゴリズムである．Barnes-Hut アルゴリズムに基づく多体シミュレーションの高速な実装に関する研究は今日でも広く行われている．

本稿では，特に重力多体問題を考える．質点の数を N とすると，ニュートン力学に基づいてそれぞれの質点に作用する万有引力を求めるための計算時間は通常 $O(N^2)$ となる．Barnes-Hut アルゴリズムでは，力の計算を始める前に，空間をセルと呼ばれる部分空間に再帰的に分割し，その空間構造を表す木構造を生成する (図 2)．そして，生成された木を探索しながら各々の質点に及ぼされる力を計算する．その際に，質点間の距離が十分に離れている場合は力を及ぼす複数の質点をセル単位でまとめ，これを 1 つの質点と見なして力の計算を行うことで，計算量を $O(N \log N)$ に削減する．

シミュレーションにおける単位時間が経過した後の全質点の物理量を求めるための実装は，一般には次のようになる．

(1) 質点の位置に基づいて木構造を生成する．
(2) 木構造の探索により，各質点に力を及ぼす質点及びセルのリストを求める．
(3) それぞれの質点に及ぼされる力を実際に計算する．
(4) (3) の計算結果に基づいて，質点の速度と位置を更新する．

これらの手順を繰り返し行うことで，質点の運動をシミュレートする．

3. 既存の Tascell における問題点

Barnes-Hut アルゴリズムでは，シミュレーションの進行に伴って，質点の物理量や木の構造などが動的に更新さ

れる．シミュレーションの実行時には，各ワーカは通常広範囲の質点や木の節点にアクセスするため，分散メモリ環境においては，こうしたデータを計算ノード間で効率良く管理・共有する必要がある．

動的負荷分散を行わない一般的な Barnes-Hut アルゴリズムの実装手法としては，空間を適当な手法で分割して各計算ノードに割り当てた後，それぞれの計算ノードに対して部分空間内の各質点に及ぼされる力を計算させる手法がある．空間の分割は力の計算を開始する前に行い，計算中は空間の再割り当ては通常行わない．この手法では，力の計算時に各ノードが質点や木のどの部分を参照するかを，割り当てられた空間情報に基づいて，計算開始前にある程度まで予測することが可能である．そのため，計算に必要と予測されるデータを事前に計算ノードに配置しておくことによって，計算中のノード間の通信を極力減らすことができる．また，データの参照局所性を向上させるために，質点の局所性を考慮した空間分割の手法も数多く提案されている．詳細については，第 6 章の関連研究も参照されたい．

一方，Tascell のような動的負荷分散フレームワークを用いて Barnes-Hut アルゴリズムを実装する場合は，前述のような一般的な手法をそのまま適用することは難しい．動的負荷分散を用いる場合，計算ノードに割り当てられるタスクは計算負荷に応じて実行時に動的に決定されるため，各計算ノードがどのようなタスクを実行するかを事前に知ることはできない．計算ノードが参照するデータはタスクに依存していることから，力の計算を開始する前に各ノードが将来参照するデータを予測し，必要なデータを事前に配置しておくことは，動的負荷分散フレームワークの下では有効ではないと考えられる．そのため，計算に必要なデータはタスクの割り当てが決定した時点で転送するか，あるいはすべてのデータを事前に計算ノードに配置しておく，などといった実装を行う必要がある．

ところが，現在の Tascell には，転送するデータをユーザが自由に決定できる通信機能が，タスクオブジェクトを介した計算ノード間の通信しか存在しない．そのため，Tascell を用いて Barnes-Hut アルゴリズムを実装する場合，力の計算に必要な質点や木のデータはタスク送信時にタスクオブジェクトに格納して送信し，力の計算結果はタスクの実行完了後にタスクオブジェクトに格納して返送するのが，唯一の計算手法となっている．しかし，この手法には次に挙げるような通信効率の問題が存在し，分散メモリ環境において理想的な性能を得られない大きな原因となっていた．

- Barnes-Hut アルゴリズムでは，木構造の探索や力の計算を行う際に，同じ質点やセルのデータを何度も参照する場合がある．Tascell ではある計算ノードが既に保有しているデータを他の計算ノードに通知できないため，同じノードに対してデータを重複して送信する可能性がある．このような状況は，例えば分割したタスクの取り返し[*1]の際に生ずる．
- タスクを受け取った計算ノードは，そのタスクを全て自分で実行するとは限らない．他の計算ノードからのタスク要求に応じて，自身のタスクを分割する可能性があるからである．もし，受信した質点や木のデータがこのようにして他の計算ノードに送信された場合，そのデータは複数の計算ノードを中継して送信されることを意味する．中継されたデータが片方の計算ノードによってしか参照されなければ，不必要なデータ転送が発生したことになる．

そこで本研究では，Tascell フレームワークと MPI ライブラリを併用し，より直接的な通信手段を柔軟に用いることで性能の改善を試みた．

4. 提案手法

本章では，本研究で作成した多体シミュレーションを行う Tascell プログラムの実装について説明する．本研究では，C 言語で実装された Barnes-Hut アルゴリズムの逐次プログラムである treecode[6] を基にして，分散メモリ環境用の Tascell プログラムを作成した．treecode の逐次実装は，2.2 節に記述した手法に基づいている．

MPI を用いた通信の実装については既存の手法も含めて様々な手法が考えられるが，選択する通信の実装によっては逐次版の Barnes-Hut アルゴリズムの実装を大幅に変更する必要がある．本研究では，簡単のため，並列化を行う部分は Barnes-Hut アルゴリズム全体の計算時間の大部分を占めている力の計算のみに限定することとした．また，逐次実装である treecode の実装を大幅に変更することなく並列化するために，MPI を用いた計算ノード間の通信には集団通信のみを用いることとした．具体的な実装は以下の通りである．

(1) 任意の 1 つの計算ノード上で，木構造の生成を逐次的に行う
(2) すべての質点と生成された木構造のデータをブロードキャストする
(3) 全計算ノードで力の計算を並列的に行う
(4) 全計算ノードの計算結果を木構造の生成を行った計算ノード上に集約する
(5) 集約された力の計算結果を基に，各質点の速度と位置の情報を逐次的に更新する

これらの手順を繰り返し行うことで，シミュレーションを進行する．MPI による集団通信は，力の計算の前後で使

[*1] あるワーカ A が自身のタスクを分割して他のワーカ B に受け渡した後，ワーカ A がワーカ B よりも先にアイドル状態になった時，ワーカ A がワーカ B に対してタスクを要求することをこのように呼んでいる．

表 1　評価環境
Table 1　Evaluation environment

	Appro GreenBlade 8000 (Tascell サーバ，計算ノード共通)
CPU	Intel Xeon E5-2670 2.6GHz 8 コア × 2
メモリ	DDR3-1600 64GB
OS	Red Hat Enterprise Linux Server release 6.2
コンパイラ (Tascell サーバ)	Allegro Common Lisp 8.1 最適化オプション `(speed 3) (safety 3) (space 1)`
コンパイラ (計算ノード)	Tascell コンパイラ (ICC 12.1.3 ベース) + Intel MPI Library 4.0 最適化オプション`-O2`, `-O3`(力の計算部のみ)
ノード間接続	InfiniBand FDR × 2

用することとした．上記のように，力の計算を行う前にすべてのデータをブロードキャストし，全体の計算が終了した後にその結果を一斉に集約することにより，ワークスティールが発生した時にサイズの大きなデータを送信しなくても済むようにしている．これにより，タスク要求に対して送信するデータは計算パラメータを示すいくつかの変数だけとなり，タスクの実行完了を通知する際に何らかの計算結果を送信する必要はなくなっている．

なお，力の計算についてはノード間で並列化を行っているだけでなく，ノード内においても並列化を行っている．その実装は文献 [7] で示されている手法に基づく．

ブロードキャストの実装には，`MPI_Bcast` を使用した．計算結果の集約の実装には，次に示すように `MPI_Gatherv` を用いた実装と `MPI_Reduce` を用いた 2 つの実装を用意し，性能を比較することとした．

`MPI_Gatherv`　MPI プロセスごとに可変個のデータを送信できるギャザー通信関数である `MPI_Gatherv` を用いて，計算結果を集約する方式である[*2]．実際には `MPI_Gatherv` を実行する前に，最初に `MPI_Gather` を実行してノードごとに送信するデータの個数を集約先の計算ノードに通知し，その結果に基づいて `MPI_Gatherv` を実行する実装を行った．

`MPI_Reduce`　計算結果の縮約演算を定義し，全質点の計算結果に対して `MPI_Reduce` による縮約を行って集約する方式である．ある 2 つの計算結果の縮約演算は，それぞれの結果が持つ対応する質点の物理量同士の加算演算として定義した．力の計算を開始する前に，計算によって更新される物理量のデータをゼロで初期化しておくことにより，縮約によって実際に更新された物理量の値のみが集約される実装を行った．

2 つの実装を用意した理由は，集団通信の種類によって通信パターンが異なることを想定し，性能に違いが見られる可能性を考慮したためである．`MPI_Gatherv` を用いた集約では，すべての計算ノードが 1 つの計算ノードに対してデータを少しずつ転送すると考えられるが，`MPI_Reduce` を用いた集約では，計算ノード間で木を構成し，全質点の計算結果を樹状に縮約していくと考えられる．また，`MPI_Gatherv` を用いた集約では，送信される総通信量は `MPI_Reduce` を用いた集約よりも少なく済むものの，メモリ上の不連続なデータを正しく送受信するためのパック／アンパック処理をユーザが実装する必要がある．`MPI_Reduce` を用いた集約では送信される総通信量は多いが，パック／アンパック処理を実装する必要はなく，MPI ライブラリの実装によって縮約処理の最適化が可能であるという違いがある．

通信の実装手法に関しては，上記のように集団通信を用いてあらかじめすべてのデータを各計算ノードに配置する他に，1 対 1 通信や分散共有メモリを用いて計算に必要なデータをオンデマンドに取得する実装も考えられる．しかし，この実装では集団通信の実装と比較して全体の通信量を少なく抑えられる可能性が高いものの，各ノードが無秩序に通信を行うことにより，性能の予測や最適化が難しい問題があると我々は考えている．こうした他の通信の実装手法との比較考察については，今後の課題である．

5. 性能評価

本章では，実装した Tascell プログラムの性能評価の結果を示す．評価を行ったプログラムは，次の 3 つである．

Bcast　文献 [4] において我々が以前に提案した Tascell プログラムである．力の計算を始める前に `MPI_Bcast` を用いて質点と木構造のデータをブロードキャストする点は前章で説明した実装と同じであるが，力の計算結果は Tascell の枠組みのみを用いて集約する実装を行った．すなわち，計算結果はタスクオブジェクトの中に格納され，タスクの実行完了とともにスティール元の計算ノードに送信されて，最終的に 1 台の計算ノード上に集約される方式である．

Bcast+Gatherv　`MPI_Bcast` に加え，計算結果の集約に `MPI_Gatherv` を用いる方式である．

Bcast+Reduce　`MPI_Bcast` に加え，計算結果の集約に `MPI_Reduce` を用いる方式である．

[*2] 動的負荷分散を行っていることから，計算ノードが生成する計算結果のデータサイズは普通ノードごとに異なっている．

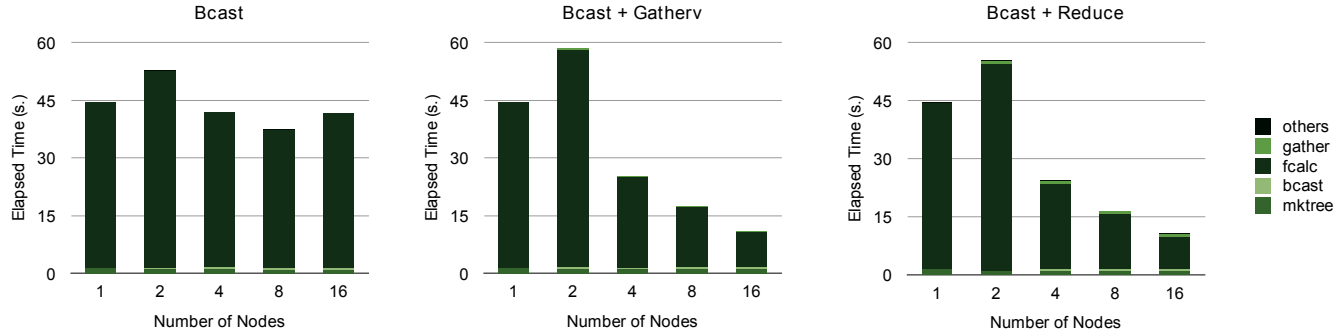

図 3 実行時間の内訳．(粒子数 300,000，ノード内ワーカ数が 1 つの場合)
Fig. 3 Computation time breakdowns. (300,000 bodies, 1 worker run in each node)

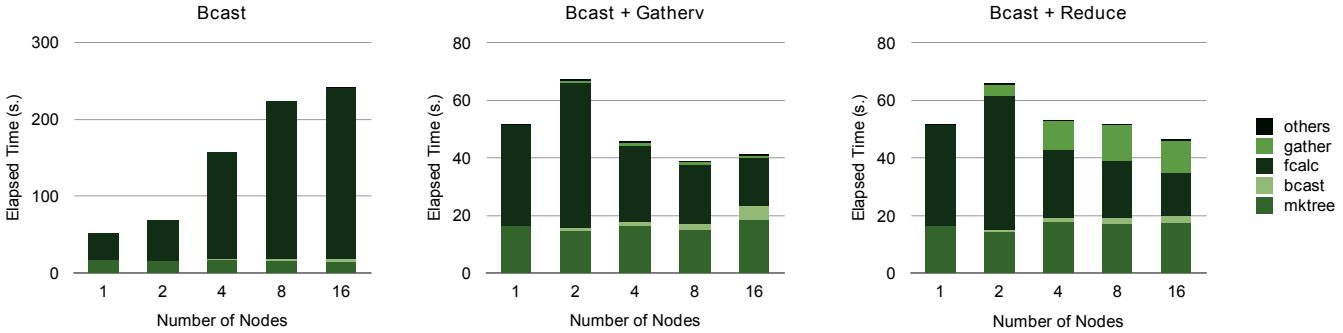

図 4 実行時間の内訳．(粒子数 3,000,000，ノード内ワーカ数が 16 の場合)
Fig. 4 Computation time breakdowns. (3,000,000 bodies, 16 workers run in each node)

なお，質点と木構造のデータのブロードキャストを `MPI_Bcast` ではなく Tascell サーバを介して行う実装の性能は，上記 3 つの実装より明らかに悪く，今回とは別の環境ではあるが具体的な評価結果も文献 [4] で示しているため，本稿では省略することとした．

評価環境は表 1 に示すとおりである．評価は，ノード内のワーカ数が 1 つの条件下 (ノード内並列実行なし) と 16 の条件下 (ノード内並列実行あり) で測定した．なお，Taseclll サーバはワーカプロセスを起動した計算ノードのうちの 1 つと同一のノードで起動している．いずれの場合も，作成した Tascell プログラムにテスト用の質点を生成させ[*3]，8 単位時間のシミュレーションを行わせて，実行時間を測定した．生成した質点の個数は，前者の条件下では 300,000 個，後者の条件下では 3,000,000 個である．

ノード内ワーカ数が 1 つの場合の評価結果を図 3 に，16 の場合の評価結果を図 4 に示した．評価結果では，実行時間を，木の構築 (mktree)，ブロードキャスト (bcast)，力の計算 (fcalc)，計算結果の集約 (gather)，その他の計算時間 (others) の 5 つに分けて示している．

Bcast では，ノード数の増加による性能向上がほとんど得られていない (図 3) か，性能の低下が顕著に見られる (図 4)．計算結果の集約に集団通信を使用している 2 つのプログラムでは速度向上が得られていることから，Bcast では計算結果の集約が性能上の大きなボトルネックとなっていることが分かる．ノード内ワーカ数の増加によって力の計算時間 (fcalc) が著しく増加しているのは，各ワーカがそれぞれ独立して他の計算ノードに対してタスクを要求する結果，通信量が増加し，却って計算時間が増加してしまったためと考えられる．

本研究で実装した，計算結果の集約方法が異なる 2 つのプログラムを比較すると，計算結果の集約時間 (gather) については `MPI_Gatherv` を用いた Tascell プログラムの方が高速な結果が得られた．特に，図 3 においては両者の集約時間には僅かな差しか見られないものの，図 4 においてはその差が顕著に見られる．このことから，計算結果のデータサイズが増加した時により良いスケーラビリティを示せるのは，`MPI_Gatherv` を用いた実装であると判断できる．この理由としては，`MPI_Gatherv` 及び `MPI_Reduce` に第 4 章で想定したような内部実装がなされていると仮定すれば，転送される総通信量の違いがそのまま性能として現れているためと考えられる．

ノード数の差異による力の計算時間の変化については，本研究で実装した両方のプログラムにおいて一定の台数効果が見られるものの，理想的な並列性能が得られているとは言えない．この原因として考えられるのは，タスクオブジェクトの送受信に伴う通信コストの存在である．集団通

[*3] 元々の treecode[6] に，プラマーモデルに基づいたテスト用の質点データを生成するための機能が備わっており，本研究でもそれを用いている．

信を用いることによって，力の計算中に転送されるタスクオブジェクトのサイズは減少したものの，通信そのものが不要となった訳ではない．いずれの結果においても 2 ノードを用いた際の力の計算時間が 1 ノードのそれよりも増加しているのは，ノード間の通信コストが大きく，台数効果による性能向上を上回ってしまっているためと考えられる．また，2.1.1 項で述べたとおり，Tascell フレームワークにおける計算ノード間のタスクの送受信は，すべて Tascell サーバを経由するようになっている．そのため，1 台の Tascell サーバに接続する計算ノード数が大きい場合には，Tascell サーバの負荷が大きくなりすぎる可能性も考えられる．本研究で実装した 2 つのプログラムについて，ノード数の増加に伴って力の計算時間が理想的に減少しないのは，そうした理由も考えられる．

全体の計算時間については，Bcast+Gatherv，Bcast+Reduce 共に台数効果を得ることができ，Bcast と比較して大幅な性能の向上を確認できる．一方で，ノード内ワーカ数が 16 の場合では，力の計算時間 (fcalc) が短くなった結果，並列化されていない木の構築 (mktree) にかかる時間の実行時間全体に占める割合が無視できなくなってきている．このことから，これ以上の並列環境において全体の性能向上を得るためには木の構築の並列化も不可欠であることがわかる．

6. 関連研究

分散メモリ環境における負荷分散のための空間分割手法の例としては，Orthogonal Recursive Bisection (ORB) [8] が挙げられる．ORB は，空間の各次元軸に対して垂直な平面で空間を再帰的に 2 つに分割していく手法である．分割する平面の位置は粒子数によって決定する手法の他，シミュレーションの前単位時間における各計算ノードの負荷情報を用いるなど，様々な手法が提案されている．ORB は，近年の大規模な多体シミュレーション [9] においても用いられている．

その他の古典的な実装例としては，Warren らによる Hashed Oct-Tree[10] に基づく手法がある．Warren らは空間を Morton 順序により順序付けを行った上で適当に分割し，分割された空間内の質点を各計算ノードに配布している．Morton 順序に基づいて空間を分割することで，配布される質点の空間局所性が高まることが期待される．また，計算ノード間のデータの共有方法に関しては，力の計算中に計算ノード上に無い質点や木構造の情報を逐一他の計算ノードに問い合わせる実装を行っている．このような手法を実現しようとする場合，任意の計算ノードにワークスティーリングとは無関係にデータの問い合わせを可能とする通信手段が必要となるため，現在の Tascell の枠組みの中だけでは実現することはできない．

重力多体問題は，かつてはベクトル型プロセッサや GRAPE[11] などの専用ハードウェアを用いて，計算時間を要する浮動小数点演算を高速に計算する手法が主流であったが，近年では GPGPU を用いた計算手法 [9] が増えてきている．一方，我々が実装した Tascell プログラムは，計算範囲を分割して並列化を施すのみに留まっており，浮動小数点演算そのものの高速化は行っていない．従って，浮動小数点演算の高速化技法により，更なる性能向上が見込まれる．

7. おわりに

本研究では，負荷分散フレームワーク Tascell と MPI ライブラリの集団通信機能を用いて，動的負荷分散を行いながら多体シミュレーションを行う Tascell プログラムを実装し，評価を行った．現在の Tascell のみの枠組みでは，分散メモリ環境においてユーザが利用できる通信機能に制限があり，一部のアプリケーションを並列化しても望ましい性能向上が得られないという問題点が存在した．本稿では，Tascell フレームワークと MPI ライブラリを併用することによりこうした制限を緩和できることを示すとともに，分散メモリ環境における Barnes-Hut アルゴリズムの並列化の性能評価において，従来手法を用いた Tascell プログラムよりも大幅に性能が改善されることを示した．また，計算結果の集約方法として 2 つの異なる実装を比較した結果，縮約通信よりもギャザー通信を用いて計算結果を集約した方が，データサイズに対して良いスケーラビリティを示せるということも分かった．

動的負荷分散フレームワークを用いて分散メモリ環境のための Barnes-Hut アルゴリズムを実装するためには，木の構築の並列化やノード間通信の実装改善など多くの課題が残されていることが，本研究を通じて示された．分散メモリ環境において動的負荷分散を行うことには，一部の計算ノードに対する外部要因による予期しない高負荷の発生や，計算ノードの途中参加といった場合に対しても，全体の計算を中断することなく負荷を適切に分散できるといった利点がある．しかし，本研究のように MPI ライブラリと Tascell フレームワークを組み合わせることによって，後者のような利点は実現が難しくなるといった問題も残されている．なお，Barnes-Hut アルゴリズムについては，既存の分散メモリ環境のためのノード間並列化手法と，Tascell を用いたノード内並列化手法を同時に使用することは可能である．

本研究で評価を行った Tascell プログラムの実装は単純であり，性能評価において表面化したスケーラビリティの問題を解決するためには，計算に必要なデータをオンデマンドに送受信するなどの，集団通信以外の通信手法を用いた実装が有効である可能性がある．今後そのような実装およびその評価も行い，動的負荷分散フレームワークに求められる通信の在り方について検討を行う予定である．ま

た，GPGPU を利用したノード内並列処理の高速化や，木の構築と質点の位置情報の更新処理を含めた Barnes-Hut アルゴリズム全体の並列化についても検討し，再度評価を行っていきたいと考えている．

謝辞 本研究の一部は，科学研究費基盤研究 (B)「安全な計算状態操作機構の実用化」(21300008) ならびに，科学研究費若手研究 (B)「後戻りに基づく動的負荷分散による並列化技法の実用化」(22700030) の助成を得て行った．

参考文献

[1] Frigo, M., Leiserson, C. E. and Randall, K. H.: The Implementation of the Cilk-5 Multithreaded Language, *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, pp. 212–223 (1998).

[2] Hiraishi, T., Yasugi, M., Umatani, S. and Yuasa, T.: Backtracking-based Load Balancing, *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 55–64 (2009).

[3] 平石 拓, 八杉昌宏, 馬谷誠二：動的負荷分散フレームワーク Tascell の広域分散およびメニーコア環境における評価, 先進的計算基盤システムシンポジウム（SACSIS2011）, pp. 55–63 (2011).

[4] 松井 健, 平石 拓, 八杉昌宏, 馬谷誠二, 湯淺太一：ワークスティーリングフレームワークにおけるブロードキャスト機能，情報処理学会研究報告-ハイパフォーマンスコンピューティング（HPC）， Vol. 2011-HPC-130, No. 56, pp. 1–9 (2011).

[5] Barnes, J. and Hut, P.: A hierarchical $O(N \log N)$ force-calculation algorithm, *Nature*, Vol. 324, pp. 446–449 (1986).

[6] Barnes, J. E.: Treecode Guide. `http://www.ifa.hawaii.edu/~barnes/treecode/treeguide.html`.

[7] 松井 健, 平石 拓, 八杉昌宏, 馬谷誠二：高速版 Barnes-Hut 多体シミュレーションの並列実装，先進的計算基盤システムシンポジウム（SACSIS2012） (2012).

[8] Warren, M. S. and Salmon, J. K.: Astrophysical N-body simulations using hierarchical tree data structures, *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pp. 570–576 (1992).

[9] Hamada, T. and Nitadori, K.: 190 TFlops Astrophysical *N*-body Simulation on a Cluster of GPUs, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–9 (2010).

[10] Warren, M. S. and Salmon, J. K.: A Parallel Hashed Oct-Tree N-Body Algorithm, *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pp. 12–21 (1993).

[11] Sugimoto, D., Chikada, Y., Makino, J., Ito, T., Ebisuzaki, T. and Uemura, M.: A special-purpose computer for gravitational many-body problems, *Nature*, Vol. 345, pp. 33–35 (1990).