Vol.2011-HPC-130 No.56
2011/7/29

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

松井 健[†1] 平石 拓[†2] 八杉昌宏[†1]
馬谷誠二[†1] 湯淺太一[†1]

分散メモリ環境上でワークスティーリングに基づいた並列計算を行う場合，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，我々の提案している並列言語 Tascell には，従来までユーザが自由に取り扱うことのできる，そのような通信機能が存在しなかった．そこで本研究では，Tascell フレームワークにブロードキャスト機能を新たに導入した．また，それを用いた行列積，及び重力多体問題の計算を行う Tascell プログラムの性能評価を行い，既存の Tascell プログラムとの比較を行った．評価の結果，ブロードキャスト機能を用いることで従来の手法よりも性能を改善することができたものの，Tascell の通信機能の実装の枠組みに，性能を低下させる要因があることも分かった．

Broadcast Functionality for a Work-Stealing Framework

Ken Matsui,[†1] Tasuku Hiraishi,[†2] Masahiro Yasugi,[†1]
Seiji Umatani[†1] and Taiichi Yuasa[†1]

When we perform parallel computing based on work-stealing techniques in distributed memory environments, communication functionality is important that enables each computational node to properly refer to data required for its computation. In the parallel language Tascell that we proposed, however, such flexible communication functionality was not available to users. In this research, we newly introduced broadcast functionality to the Tascell framework. We evaluated the performance of Tascell programs that perform matrix multiplication and n-body simulation that use the broadcast functionality, and compared their performance to that of the existing programs. By using the broadcast functionality, we achieved better performance results over previous methods, but we also found that the current implementation framework of Tascell's communication functionality has some causes of performance degradation.

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，並列計算向けの高生産性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言語 Tascell[2] を提案している．Tascell ワーカは本物のタスクを生成 (spawn) するが，それは他のアイドルなワーカから要求されたときであり，一時的なバックトラックによる最古のタスク生成可能状態の復元に基づく．この手法は，論理スレッド生成・管理コストの削減，作業空間の再利用促進，参照局所性改善などの利点を持つとともに，作業空間の遅延コピーによるすっきりとした効率良いバックトラック探索アルゴリズムを実現する．また，分散メモリ環境における並列計算にも対応しており，PC クラスタや広域分散環境における性能評価[3] 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．

分散メモリ環境上での並列計算は，単にアルゴリズムの並列化手法について検討するのみならず，計算に必要なデータの配布や計算結果の集約といった，各計算ノード間でのデータ管理の手法についても考えなければならない．定型的な並列計算では，あらかじめ各計算ノードに計算させるタスクを静的に決定し，それに基づいて必要なデータを各計算ノードに配布し，後にそれぞれの計算ノードから計算結果を集約するといった手法を採る．一方 Tascell では，ワークスティール発生時に計算に必要なデータを送信し，計算完了時にその結果を返信するという方法で分散メモリ環境での並列計算に対応しており，ユーザがワークスティールとは独立して扱うことのできる通信機能が存在しなかった．そのため，計算ノード間で共有可能なデータが存在するアプリケーションなど，特定のアプリケーションを Tascell で並列化した場合に，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，我々はユーザが取り扱うことのできるブロードキャスト機能を実装し，新たに Tascell フレームワークに取り入れた．本稿では，新

†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

たに導入した通信機能についての概説をし，具体的なアプリケーションの構築を通じて得られた性能評価の結果を示す．評価を行うアプリケーションとしては，行列積，及び重力多体問題における Barnes-Hut アルゴリズムを選択した．

本稿の構成は以下の通りである．第 2 章では，既存の並列言語 Tascell について概説する．第 3 章では，今回選択したアプリケーションを実装する際に，既存の Tascell の通信機能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，今回新たに導入したブロードキャスト機能の仕様について説明し，それを用いた行列積，及び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す．第 5 章で性能評価の結果を示し，考察する．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．

2. 並列言語 Tascell

2.1 概　　要

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持っている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．これを繰り返すことにより，全体として 1 つのタスクが計算される仕組みとなっている．

タスクの分割は第 1 章でも触れたように，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することによって行われる．すなわち，ワーカは他のワーカからのタスク要求を受けると，

(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

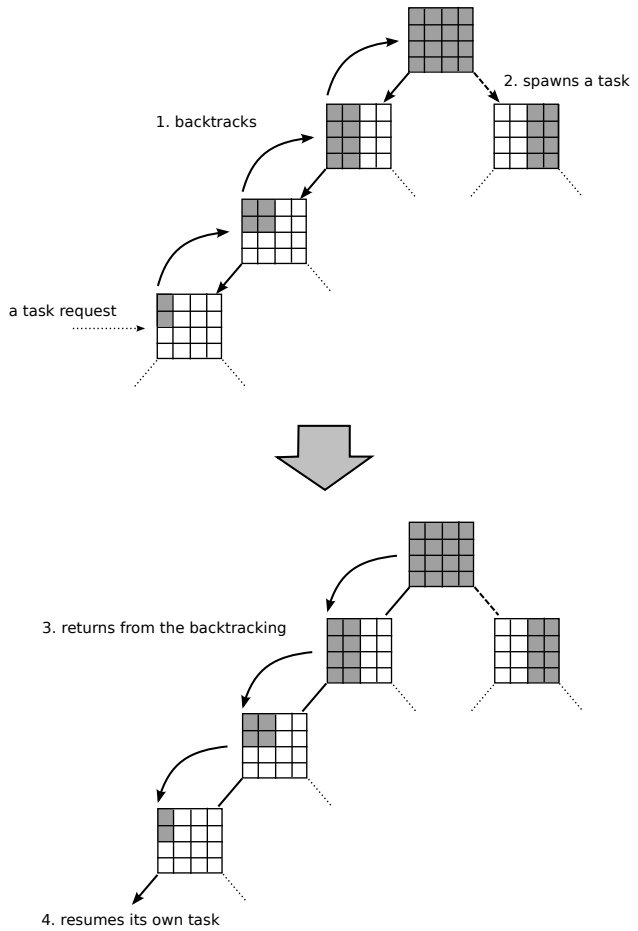


図 1　行列積 $C = AB$ の計算における，バックトラックに基づくタスクの遅延生成．ワーカが実行すべきタスクを「計算すべき行列 C の範囲」として定義し，計算範囲を再帰的に分割しながら行列計算を実行するものとした．ワーカはタスク要求を受けると過去の状態にバックトラックし，自身のタスクを分割して別のワーカに受け渡す．

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.

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）．バックトラックに基づいたタスクの生成は，生成されるタスクの粒度を極大化し，効率の良い動的な負荷分散をフレームワークレベルで実現するための礎となっている．また，タスクを遅延分割することによりワークスペースの不必要な生成・複製が抑えられると同時に，同一のワークスペース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．Cilk をはじめとしたマルチスレッド言語のように，多数の論理スレッドを維持・管理する必要がなくなるという利点も持つ．

2.2 分散メモリ環境

分散メモリ環境では，計算ノード間の通信を制御する仕組みと計算ノード間の通信プロトコルが必要となる．前者は Tascell サーバによって行われ，後者は Tascell 独自のメッセージプロトコルがその役割を担う．

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．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，計算ノード間のメッセージの中継や，ユーザインタフェースとの入出力処理，各計算ノードの負荷情報の管理などを行う．

計算ノード間の通信は，実際には各計算ノード内の特定のワーカを対象として行われる．ワーカ間で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．計算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送り返す．こうしたサーバ・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いる．

従来までの Tascell では，各タスクの実行に必要なデータは，タスクオブジェクトの中に格納して送信しなければ別の計算ノードに転送できないという制約があった．これは，ユーザが特定の計算ノード間でワークスティーリングによるタスクオブジェクトの受け渡しとは無関係に，任意の通信を行うための機能が存在しなかったからである．そのため，計算ノード間で共有可能なデータが存在するアプリケーションなど，特定のアプリケーションを Tascell で並列化した場合に，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．

3. 既存の Tascell における問題点

本章では，具体的な問題例として行列積と重力多体問題を考え，既存の Tascell を用いて分散メモリ環境上で並列計算をする際に発生する問題点について記述する．

3.1 行 列 積

行列積 $C = AB$ の計算を考える．行列積の並列計算を行う場合，行列 A, B を分割して部分行列にし，それぞれの部分行列同士の積を並列化するのが最も自然であると言える．一般に，分散メモリ環境上で行列積の並列計算を行う場合には，あらかじめ行列 A, B を静的に分割して各計算ノードに転送し[⋆1]，計算終了後にそれぞれの計算ノードから計算した部分行列を集約するといった手法が考えられる．このような計算手法は，例えば MPI 実装の `MPI_Scatter` 関数，`MPI_Gather` 関数などを用いることにより，比較的容易に実装可能である．

Tascell を用いて行列積の計算を行う場合，上記の計算手法をそのまま適用することは出来ない．Tascell における計算は，全計算ノード間でのワークスティーリングに基づいて動的に決定されるため，前述したように行列を静的に分割して各計算ノードに配布するという手法は適さないためである．Tascell では，計算すべき行列 C の範囲は動的に決定されるため，積の計算に必要な部分行列のデータをタスクオブジェクトの中に格納して転送し，その計算結果をやはりタスクオブジェクトの中に格納して集約するのが，既存の Tascell の枠組みの中では最も自然な計算手法であった．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た．

- タスクを受け取った計算ノードは，指示された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．本例のように，行列のデータをタスクオブジェクトに格納する場合，タスク受信後に発生するタスクスティールによって一部の行列データが再び別の計算ノードに転送されれば，そのデータは複数の計算ノードを中継して転送される結果を生じてしまう．行列のサイズが大きくなるほど，通信効率への影響もより顕著に表れる．
- タスクを要求する際に，計算に必要な行列のデータを既に保持している場合であっても，それを送信側の計算ノードに通知する手段が既存の Tascell には存在しなかった．

⋆1 事前に行列のデータを計算ノード上に静的に配置しておくことも考えられるが，ここでは 1 つの計算ノードだけが全ての行列データを持つという状況を考える．これは，後に示す多体問題の例のように，計算に必要なデータが動的に生成されるようなアルゴリズムを並列化する場合も考慮するためである．

Vol.2011-HPC-130 No.56
2011/7/29

これにより，タスクオブジェクトには毎回必ず何らかの行列データが格納されることとなり，不必要なデータ転送が発生する場合がある．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．

こうした通信効率の問題を解決するため，我々は Tascell の枠組みに新たな通信機能を導入することにより，計算手法を改善することを試みた．具体的には，次のような計算手法の検討を行った．

(1) あらかじめ行列 A, B の全体を，全ての計算ノードにブロードキャストしておくことにより，タスクオブジェクトを送信する際の行列データの格納を不要とする手法．

(2) 任意のワーカが，任意の計算ノードに対してタスクオブジェクトの授受とは無関係にデータを要求できるようにすることで，各ワーカが計算に必要な部分行列のデータを自身で判断し，必要に応じて他の計算ノードから取得する手法．

(3) 分散共有メモリシステム，あるいはそれに準ずる機能を導入することにより，どの計算ノードからも行列 A, B を論理的に参照可能とする手法．

本研究では，Tascell フレームワークにブロードキャスト機能を追加することにより，上記の手法 (1) の実装を行った．その他の手法については，Tascell の拡張に伴う実装上の課題がブロードキャスト機能と比較して多いため，本研究での実装は見送ることとした．

3.2 重力多体問題

重力多体問題とは，万有引力によって互いの運動に作用する多数の質点の系を取り扱う問題であり，宇宙空間における天体運動の進行を求める問題などを指す．本稿では，分散メモリ環境における重力多体問題の計算について取り扱う．特に多体問題を近似的に解くアルゴリズムの 1 つである Barnes-Hut アルゴリズム[4] の並列計算について考える．

Barnes-Hut アルゴリズムは，任意の 2 つの質点が互いに及ぼす力を全て計算するのではなく，可能な場合には複数の質点を 1 つにまとめて計算することにより，計算量を削減するアルゴリズムである．具体的には，はじめに空間を質点の位置に基づいてセルと呼ばれる部分空間に再帰的に分割し，それを階層化された木構造で表現する（図 2）．そして，ある質点に力を及ぼす質点が遠く離れている場合には，そのような質点をセルによってまとめ，複数の質点を 1 つの質点と見なして力の計算を行う．計算機上での Barnes-Hut アルゴリズムの典型的な実装は，大きく次のような 4 つの処理に分かれる．

⋆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．

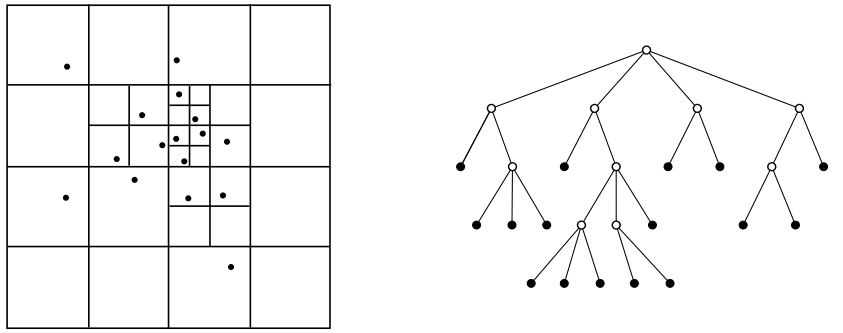


図 2 再帰的に分割された空間と，それを表す階層化された木構造．Barnes-Hut アルゴリズムは，空間をセルと呼ばれる部分空間に再帰的に分割する（簡単のためここでは平面で表している）．左図において，黒点は質点を表す．右図において，黒と白の節点はそれぞれ質点とセルを表す．

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.

(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ないし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．

分散メモリ環境上で Barnes-Hut アルゴリズムの並列計算を行う場合，計算範囲の分割対象としては空間を選択している研究例が多い[5)6)7)]．すなわち，あらかじめ問題空間を計算ノードの数だけ適当な手法で分割した後，各計算ノードにそれぞれの部分空間を割り当て，その部分空間内に存在する質点に対して及ぼされる力を計算する，といった手法が採られる．Barnes-Hut アルゴリズムでは，質点と木構造のデータが計算に必要なデータであり，これらのデータを計算ノード間でどのように管理するかについては，様々な手法が提案されている．

Tascell を用いて Barnes-Hut アルゴリズムによる重力多体問題の計算を並列化する場合も，行列積の場合と同様の，通信効率の問題が発生する．すなわち，割り当てられた部分空間の計算に必要なデータをタスクオブジェクトに格納する場合，

- データが複数の計算ノードを中継してしまう．
- 既に保有しているデータを重複して送信してしまう．

といった問題点が挙げられる．また，Barnes-Hut アルゴリズムは行列積と比較して以下のような相違点があり，既存の Tascell の通信機能を用いた実装では，通信効率による問題がより顕著に発生する原因となっていた．

- Barnes-Hut アルゴリズムでは，各計算ノードが木構造のどの部分を参照し，力の計算にどの質点の情報を必要とするかは，実際に計算を始めるまで完全には決定できないという性質がある．そのためタスクオブジェクトには，全ての質点と全ての木構造のデータを毎回格納するか，何らかの手法に基づいてタスクの計算に必要なデータを予測し，格納するデータを動的に選択するといった実装を行う必要があった．いずれの場合も，タスクとして送信したデータの全てが実際の計算に使用されるとは限らず，不必要なデータ転送を生じてしまう可能性がある．
- 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ため，既に保有しているデータをタスクの要求時に通知できない既存の Tascell では，前述した重複データの送信がより大きな問題となる．

こうした問題に対し，我々は 3.1 節と同様の，新たな通信機能の導入による計算手法の改善を検討した．本研究では，ブロードキャスト機能を用いることにより，質点と木構造のデータをブロードキャストし，タスクオブジェクトの中に格納するデータのサイズを減らす計算手法の実装を行った．

4. 提 案 手 法

4.1 ブロードキャスト機能

Tascell におけるブロードキャスト機能は，分散メモリ環境における任意の計算ノードが，Tascell サーバによって接続されているネットワーク上の全ての計算ノードに対して，同一のデータを送信する仕組みを指しており，ネットワークにおける一般的なブロードキャストの機能を Tascell の枠組みにおいて提供するものである．

2.2 節でも述べたとおり，Tascell における分散メモリ環境上での計算ノード間の通信は，全て Tascell サーバを中継している．本研究で実装したブロードキャスト機能も，送受信するデータは全て Tascell サーバを中継することとした．

ブロードキャスト機能は，次に示す Tascell サーバと計算ノード間の通信を経て実行される．

(1) ブロードキャストは，データを送信しようとする任意のワーカが，接続している Tascell サーバに対し，ブロードキャストのメッセージを送信することで開始される．ワーカは送信したいデータ本体だけではなく，ブロードキャストデータの種類を識別するための ID などといった内部データも含めて Tascell サーバに送信し，サーバからの応答を待つ．
(2) ブロードキャストメッセージを受信した Tascell サーバは，自分が接続している他の全ての計算機に対し，受信したメッセージを転送し，それぞれの計算機からの応答を待つ．
(3) ブロードキャストメッセージを受信した計算ノードは，データ本体を計算機内に格納し，サーバに対して受信が正常に完了したことを示す確認メッセージを送信する．応答を待っている計算ノード及び Tascell サーバはこのメッセージの受信をもって，ブロードキャストの完了を確認する．

Tascell サーバと計算ノードとの間で交わされるこれらの通信は，他のメッセージと同様に Tascell 実行フレームワークによって自動的に処理されることとした．

4.2 行列積への応用

Tascell のブロードキャスト機能を用いた行列積の並列計算の手法は，3.1 節で述べたとおりである．つまり，計算を実行する前に行列 A, B のデータを全ての計算ノードにブロードキャストしておき，タスクオブジェクトを送信する際に，計算範囲内の部分行列の送信を不要とする実装を行った．これにより，実際にタスクオブジェクトの中に格納するデータは，行列の計算範囲を示すいくつかの整数値だけで済むようになった．なお，計算した行列 C の部分行列の集約方法に関しては，従来の Tascell における実装方法と同様にタスクオブジェクトの中に格納して，タスクの分割元の計算ノードに対して送り返す実装とした．

4.3 重力多体問題への応用

本研究で実装した重力多体問題の Tascell プログラムは，Barnes-Hut アルゴリズムの逐次実装プログラムである treecode[10] を参考にして作成した．また，簡単のため木構造の探索及び力の計算を行う部分に絞って並列化を行った．具体的な計算手順としては，以下のような実装とした．

(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) 質点の情報と生成された木構造を，全ての計算ノードにブロードキャストする
(3) 全計算ノード間で力の計算を分担して行い，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
(4) (3) で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

ブロードキャスト機能を用いることで，(3) の並列計算実行時にタスクオブジェクトへの質

表 1 評価環境
Table 1 Evaluation environment

	Tascell サーバ	計算ノード
CPU	AMD Opteron Processor 244 1.8GHz 1 コア × 2	AMD Opteron Processor 265 1.8GHz 2 コア × 2
メモリ	1GB	2GB
OS	Rocks 4.0 (Linux kernel 2.6.9)	
コンパイラ	Allegro Common Lisp 8.1 最適化オプション (speed 3) (safety 3) (space 1)	Tascell: XC-cube (x86_64 GCC 3.4.6 ベース), L-Closure に基づく入れ子関数[8)9)] MPI: MPICH 1.2.7 + GCC 3.4.6 最適化オプション-O2, -O3（Barnes-Hut の力の計算部のみ）
ノード間接続	Gigabit Ethernet	

点と木構造のデータの格納を不要としている．計算した力のデータの集約方法に関しては，行列積の実装と同様にタスクオブジェクトの中に格納してタスクの分割元の計算ノードに送り返す実装とした．

なお，木構造の生成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は，今後の研究を通じて実装し，評価を行っていく予定である．併せて，木構造を含めた並列化の手法に関しては，関連研究の章も参照されたい．

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．なお，いずれの測定も表 1 に示す評価環境を用いて，性能評価を行っている．

5.1 行列積

はじめに，行列積の性能評価結果について述べる．評価方法は，各要素が倍精度浮動小数点数からなるサイズ 2,000 の正方行列 A, B を適当に生成し，分散メモリ環境上で行列積 $C = AB$ を計算するのに要した時間を測定することとした．評価には「既存の Tascell の通信機能のみを用いた Tascell プログラム」と「ブロードキャスト機能を用いた Tascell プログラム」を作成し，2 つのプログラムの計算時間の差異を比較した．また，参考としてブロードキャスト機能のみを MPI による実装に置換した Tascell プログラムも作成し，合計 3 つのプログラムの計算時間を測定した．

測定結果は，図 3 のとおりである．図 3(a) は既存の Tascell の通信機能のみを用いた計算時間，図 3(b) は Tascell のブロードキャスト機能を用いた計算時間，図 3(c) は MPI のブロードキャスト機能を用いた計算時間をそれぞれ表している．また，グラフ中の項目 “bcast” はブロードキャスト時間を，“multiply” は行列積の計算時間を表している．計算ノード数は 1, 2, 4, 8, 16 の場合を測定した．なお，各計算ノード内のワーカ数はいずれも 1 としている．

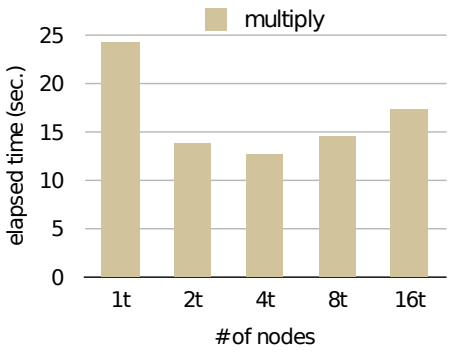
測定結果から，特に 4 ノード以上の場合において，ブロードキャスト機能を用いることによる計算時間の改善が認められる．しかし，図 3(b) では 16 ノードの場合における計算時間が 8 ノードのそれを上回っており，スケーラビリティの問題が指摘できる．この計算時間の増大は，ブロードキャストに要する時間が計算ノード数に比例して増加していることに起因すると考えられる．MPI のブロードキャスト機能を用いた図 3(c) では，ブロードキャストに要する時間が Tascell のブロードキャスト時間と比較して短く，計算ノード数の増加に伴って全体の計算時間が単調に減少していることが認められるためである．

MPI の実装と比較して Tascell のブロードキャスト時間が長い理由としては，Tascell サーバの存在が原因の 1 つとして挙げられる．これまで記述してきたとおり，現在の Tascell における計算ノード間の通信は全て Tascell サーバを中継するという仕様になっており，計算ノード間で直接データを送受信していない．そのため，サーバの中継に伴うオーバーヘッドの増加や，多数の計算ノードが Tascell サーバに接続している場合に帯域飽和による遅延が発生するといった問題点がある．

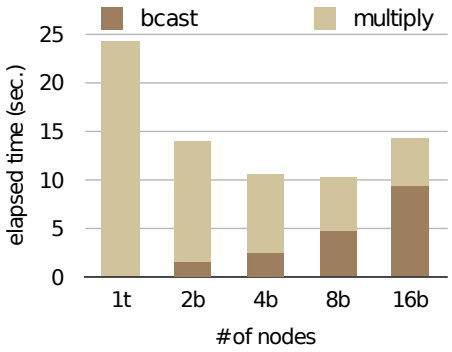
5.2 重力多体問題

続いて，重力多体問題の性能評価結果について述べる．評価方法は，テスト用の質点 20,000 個を生成し[⋆1]，分散メモリ環境上で 32 単位時間後の各質点の位置を求めた時の，力の計算及びブロードキャストに要した時間を測定することとした．評価には，行列積と同じく，3

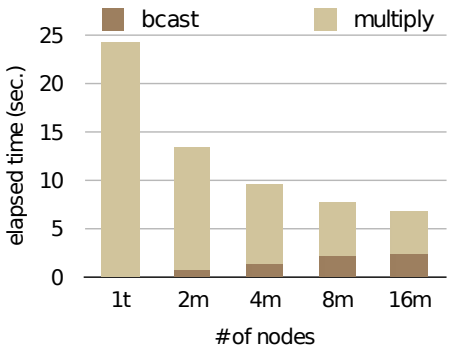
⋆1 元々の treecode[10)] に，プラマーモデルに基づいたテスト用の質点データを生成するための機能が備わっており，本研究でもそれを用いている．



(a) Tascell

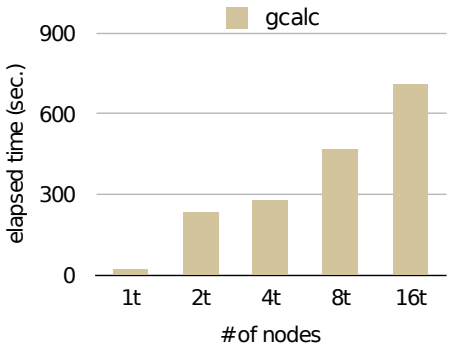


(b) Tascell+Broadcast

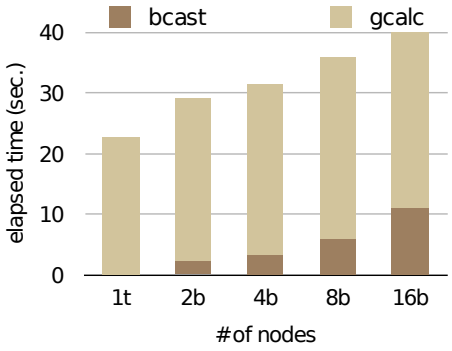


(c) Tascell+MPI_Bcast

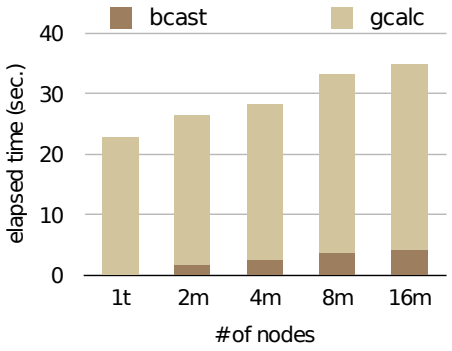
図 3 行列積の性能評価結果
Fig. 3 Evaluation result of the matrix multiplication



(a) Tascell



(b) Tascell+Broadcast



(c) Tascell+MPI_Bcast

図 4 Barnes-Hut アルゴリズムの性能評価結果
Fig. 4 Evaluation result of the Barnes-Hut algorithm

種類の異なる通信方式により並列計算を行う Tascell プログラムを用いた．

測定結果は，図 4 のとおりである．図 3 と同じく，3 つのグラフは，既存の Tascell の通信機能のみを用いた計算時間，Tascell のブロードキャスト機能を用いた計算時間，MPI のブロードキャスト機能を用いた計算時間をそれぞれ表している．また，グラフ中の項目 “bcast” はブロードキャスト時間を，“gcalc” は力の計算に要した時間を表している．計算ノード数についても，行列積と同じく 1, 2, 4, 8, 16 の場合を測定した．各計算ノード内のワーカ数はいずれも 1 としている．

評価結果から，ブロードキャスト機能を用いることによる計算時間の改善は認められるものの，いずれの Tascell プログラムについても計算ノード数の増加に伴って計算時間が単調に増加する結果となってしまっている．これは力の計算に要する時間について，計算ノード数の増加に伴う台数効果が得られていないことが最も大きな理由である．この原因として考えられるのは，力の計算結果の集約に伴う通信のオーバーヘッドが，台数効果による性能向上を相殺してしまっていることである．Tascell におけるタスクの計算結果は，必ずタスクの分割元の計算ノードに対して送信されることから，計算結果が複数の計算ノードを中継してしまうことや重複するデータを送信してしまうといった，タスク分割・送信時と同様の問題が発生しうるためである．この問題の解決策としては，計算結果をある特定の計算ノードに 1 度の通信で集約する機能を実装することが考えられる．例えば，MPI 実装における `MPI_Gather` 関数がこれに相当する．計算に必要なデータを最初にブロードキャストする手法と同様に，全ての計算結果を計算終了後にまとめて集約することで，計算結果のタスクオブジェクトへの格納を不要とし，より計算ノード間の通信効率を高めるというものである．

6. 関 連 研 究

分散メモリ環境における Barnes-Hut アルゴリズムの古典的な実装例としては，Warren らによる Hashed Oct-Tree[5] に基づく手法がある．Warren らはデータの共有方法に関して，

- 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，分割された空間内に含まれる質点を力の計算を行うべき質点として各計算ノードに配布する
- 各計算ノードは受け取った質点から部分的な木構造を生成し，力の計算中に自身の計算ノード内にな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

といった実装を行っている．この手法では，計算ノードがそれぞれ独立して部分的な木構造を生成し，後に必要に応じて不足している木構造のデータを他の計算ノードに問い合わせることで，木構造の生成を並列化している．このような手法を Tascell の枠組みの中で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機能が必要となる．Tascell に対し柔軟性の高い通信機能を導入することを，今後も継続して検討する予定である．

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[6][7] が増えてきている．また，GPGPU の提供する高性能な浮動小数点演算は，行列積に対しても有効であると考えられる．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．

高速なブロードキャストの転送方法に関する研究としては，Izmailov らによる FPFR[12] がある．これは，ネットワークのトポロジ情報を用いてブロードキャストに参加しているノードを深さ優先でたどり，ノード間に複数のパイプラインを構築することで転送性能の向上を図る手法である．これを変形した Takahashi らによる Stable Broadcast[13] といった手法も提唱されている．これらは，現在のブロードキャスト機能の問題点の 1 つである，転送が全て Tascell サーバを介することによって生じる帯域飽和を解消する方法として，効果が期待できる．

7. 終 わ り に

本研究では，並列言語 Tascell のフレームワークに分散メモリ環境のためのブロードキャスト機能を新たに導入し，行列積及び多体問題のアプリケーションプログラムを実装して，その性能を評価した．従来の Tascell 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，これら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．本稿で提案したブロードキャスト機能を使用することにより，以前の Tascell プログラムよりも性能が向上することが明らかになったが，ブロードキャスト機能の実装に関して更なる改善の必要があるということも分かった．また重力多体問題の性能評価では計算ノード数の増加による台数効果を得ることができず，課題を残す結果となった．

今後は，前述した通信機能の最適化や計算結果を集約するためのギャザー機能の実装などを行い，望ましい性能向上率が得られるかどうか再度評価を行う予定である．その他，3.1 節で言及した「ワークスティーリングとは独立して，他の計算ノードにデータの問い合わせ

を可能とする通信機能」の実装を行い，ブロードキャスト機能を用いたアプリケーションプログラムとの性能比較を行いたいと考えている．また，Barnes-Hut アルゴリズムについては，木構造の生成を含めたアルゴリズム全体の並列化を Tascell を用いて実現する手法の検討・実装を行い，再度評価を行う予定である．

謝辞 本研究の一部は，「並列分散計算環境を安定有効活用する要求駆動型負荷分散」(21013027)(科学研究費特定領域研究「情報爆発時代に向けた新しい IT 基盤技術の研究」)，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) Barnes, J. and Hut, P.: A hierarchical $O(N \log N)$ force-calculation algorithm, *Nature*, Vol.324, pp.446–449 (1986).
5) 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).
6) 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).
7) Jetley, P., Wesolowski, L., Gioachin, F., Kalé, L.V. and Quinn, T.R.: Scaling Hierarchical N-body Simulations on GPU Clusters, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp.1–11 (2010).
8) Yasugi, M., Hiraishi, T. and Yuasa, T.: Lightweight Lexical Closures for Legitimate Execution Stack Access, *Proceedings of 15th International Conference on Compiler Construction*, Lecture Notes in Computer Science, No.3923, pp.170–184 (2006).
9) 八杉昌宏，平石　拓，篠原丈成，湯淺太一：L-Closure:高性能・高信頼プログラミング言語の実装向け言語機構，情報処理学会論文誌：プログラミング， Vol.49, No.SIG 1 (PRO 35), pp.63–83 (2008).
10) Barnes, J.E.: Treecode Guide. http://www.ifa.hawaii.edu/~barnes/treecode/treeguide.html.
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).
12) Izmailov, R., Ganguly, S. and Tu, N.: Fast Parallel File Replication in Data Grid, *Future of Grid Data Environments workshop* (2004).
13) Takahashi, K., Saito, H., Shibata, T. and Taura, K.: A Stable Broadcast Algorithm, *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pp.392–400 (2008).