Vol.2012-HPC-135 No.35
2012/8/3

タスク並列モデルを用いた Tree-based AMR の評価

河野 瑛[1,a)]　田浦 健次朗[1,b)]

概要：近年，Tree-based AMR に代表する不均衡性や動的な性質を持った複雑なアプリケーションが注目されており，これらのアプリケーションを効率的に並列することは難しい問題として認識されている．現在，分散環境における並列化のデファクトスタンダードは MPI であるが，MPI はプリミティブなモデルでありこのような背景から，不均衡・動的なアプリケーションを効率的に並列化する方法として，タスク並列プログラミングモデルを用いる手法が近年注目されている．
本研究は，Tree-based AMR のタスク並列モデルを用いた並列化を目的とし，そのために Tree-based AMR の重要なパートであるバランス条件をみたすためのアルゴリズムを新しく提案した．提案アルゴリズは分割統治に基づいており，タスク並列モデルを用いて容易に並列化可能である．本稿では，提案アルゴリズムをタスク並列処理系 Cilk をもちいて実装を行い，予備的な評価を行った．

キーワード：タスク並列，適合細分化格子法，AMR

An Evaluation of Parallel Tree-based AMR with Task Parallelism

Akira Kono[1,a)]　Kenjiro Taura[1,b)]

Abstract: Irregular and dynamic applications like tree-based AMR are difficult to parallelize efficiently. Current de facto standart technique of parallilizing programs in distributed environment is MPI, but MPI is a primitive programming model and is hard to parallelize efficiently such complex applications.
Task parallelism is one of the promising parallel programming models for such irregular and dynamic applications, because it can handles flexible parallelisms.
In this paper, we're trying to parallelize Tree-based AMR with task parallelism. We developed a novel 2:1 balancing algorithm which is based on divide-and-conquer method. We implemented it with Cilk and made some preliminary evaluations.

Keywords: Task parallelism, Tree-based AMR

1. はじめに

近年並列計算環境の普及に伴い，大規模な数値計算によるシミュレーションが可能になってきた．このような数値計算で近年注目されている計算の中には，FMM（Fast Multipole Method）のような粒子を用いた計算や AMR (Adaptive Mesh Refinement) のように，不均衡で動的な性質を持つものがある．これらの計算では均等に負荷分散を行うことが難しく，効率的に並列化することは難しい問題である．

この種の物理計算では現在， MPI（Message Passing Interface）[8] を用いて並列化がなされることが多いが，MPI はハードウェアの物理的な構成に素直にマッピングするように設計されており，極めてプリミティブなプログラミングモデルである．従って，実際に行われる計算や通信を把握しやすく性能を引き出しやすいという長所を持つ一方，データの分散配置や Send/Receive モデルによる通信を明示的に記述する必要があるために逐次プログラムからの飛躍が大きくプログラミングが難しい，という短所を

[1] 東京大学大学院情報理工学系研究科
The University of Tokyo
[a)] adilla@eidos.ic.i.u-tokyo.ac.jp
[b)] tau@eidos.ic.i.u-tokyo.ac.jp

© 2012 Information Processing Society of Japan

持つ．また，MPI を用いることによる制約から，現在用いられている並列アルゴリズムや並列プログラムが過度に複雑になっているのではないかという懸念も存在する．

このような背景から，より高水準・高生産に並列プログラムを記述することのできるプログラミングモデルやプログラミング言語が近年盛んに研究されており，それらによってより多くのプログラムが高生産に並列化されていくことが期待されている．

1.1 タスク並列モデル

タスク並列型のプログラミングモデルは広く知られており，現在研究開発が進められている高生産並列言語の多くでサポートされている．タスク並列モデルでは，プログラム中の任意の場所で「タスク」と呼ばれる並列処理可能な計算単位を生成することによって並列性を記述する．従って柔軟に並列性を記述することができ，MPI では難しい再帰や多重ループ等も容易に並列化することが可能である．

また効率的な並列計算のためには各プロセッサの負荷を均等にすることが肝要だが，多くのタスク並列処理系ではワークスティーリングと呼ばれる機構によって処理系がタスクの割り当てを行うため，細粒度にタスクを生成することによって自動的に均一な負荷分散を達成することが可能である．この性質は，ユーザが明示的にデータを分割しなければならない MPI に比べて大きな長所となっており，上で述べた不均衡・動的なアプリケーションに対する効率的な並列化に期待が高まっている．

しかしながら，タスク並列モデルによる実際的なアプリケーションの並列化は現在のところ共有メモリ上でのものが殆どである．これは分散メモリ上におけるタスク並列言語処理系の実装が研究段階であり，技術として成熟していないことが大きな原因だと考えられる．我々の研究グループでも MassiveThreads[14] と呼ぶ軽量マルチスレッド処理系を開発しており，こういった研究開発が進むにつれて現在の状況は変わっていくと考えられる．

1.2 Tree-based AMR

物理現象は時間的，空間的にマルチスケールな構造をとることがよくある．AMR（Adaptive Mesh Refinement）は，このようなマルチスケール問題において，差分法や有限要素法のような，格子を用いる数値理計算で必要な部分に局所的に細い格子を用い，全体としては粗い格子を用いて離散化を行うことで計算量やメモリ量を節約しつつ高解像度の計算を行う手法である．

Tree-based AMR は AMR の一種であり，柔軟かつ機械的に格子を生成することができるために近年注目されている手法である．AMR では領域中のノードの存在が不均衡であり，また時間依存問題の場合には時間が進むに連れて格子が変化する動的な性質を持っているため，従来の MPI を用いた領域分割型の並列化手法で効率的に負荷分散を行うことは単純な問題では無い．

加えて，解の精度を保証するために 2:1 バランス条件と呼ばれるセル間の細分化レベルに関する制約が設けられる事が多く，この制約を満たすプロセスをスケーラブルに処理することもチャレンジングな問題となっている．

1.3 本研究の目的

本研究では，タスク並列モデルを用いて Tree-based AMR の並列化を行う．特に，分割統治法を用いたバランス条件を満たすための効率的なアルゴリズムを新しく提案する．

またそのプロセスにおいて，タスク並列モデルを用いることで高生産に効率的な並列プログラムを記述可能であることを示し，その結果からタスク並列プログラミング処理系に求められる要件についても検討を行う．

1.4 本稿の構成

2 章では，本研究の対象アプリケーションである Tree-based AMR について述べる．3 章では，近年注目されている先行研究を紹介する．4 章では提案手法について述べ，分割統治法を用いたバランス条件を満たすためのアルゴリズムを解説する．5 章では提案アルゴリズムについての評価と結果の考察をを行う．最後に 6 章で本稿のまとめと今後の展望について述べる．

2. Tree-based AMR

Adaptive Mesh Refinement（AMR）は 1984 年に Berger らによって初めに考案された方法であり [1], [2]，有限要素法などの格子法と組み合わせて用いる，偏微分方程式の一般解法である．基本的な考え方は 1.2 節で述べたように，必要な部分に局所的に細かい格子で，全体は粗い格子で離散化を行うことにより，高解像度の計算を少ない計算量・データサイズで行うことである．

AMR は，使用する格子の違いによって大きく 2 種類に分類することができる．即ち，格子が規則的に並んだ構造格子を用いる Structured AMR と，格子の並びに規則性をもたない非構造格子を用いる Unstructured AMR である．Unstrctured AMR の方が複雑な形状に対応したメッシュを生成することが出来る一方でメッシュ生成が難しく，Structured AMR はメッシュの自由度で劣るもののメッシュが機械的に生成可能である．

Tree-based AMR は Structured AMR の一種であり，格子中のセルを再帰的に分割することによって木構造で表される階層的な細分化を行う．図 1 は，Tree-based AMR を用いた場合のメッシュと木構造による階層性の対応を表した図である．なお本稿では，図を用いて説明する際には簡単のために 2 次元の 4 分木を用いるが，3 次元の場合の 8 分木でも基本的に同様の考え方を適用可能である．

© 2012 Information Processing Society of Japan

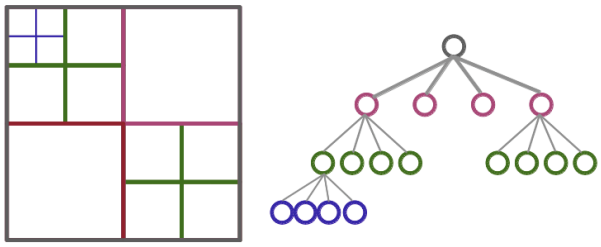

図 1 Tree-based AMR における 4 分木構造と細分化された格子中の各セルの対応

Fig. 1 An image of refined mesh and hierarchical structre of quad-tree in Tree-based AMR

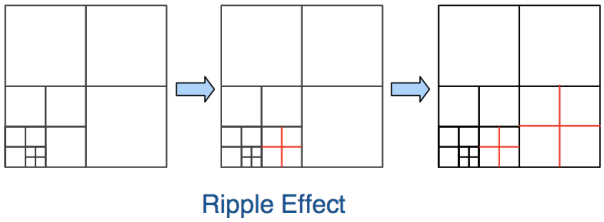

図 2 *Ripple Effect*：バランス条件を満たすための細分化によって，直接隣接していないセルの細分化が連鎖的に引き起こされる．

Fig. 2 *Ripple Effect*: An octant can trigger a sequence of splits to satisfy 2:1 balance constraint.

2.1 2:1 バランス条件

Tree-based AMR の多くでは，解の精度を保証するため 2:1 バランス条件とよばれる制約が存在する．これは隣り合うセルの細分化レベルが異なる場合，細かいセルのノードの持つ変数が補間によって決定されるためである．即ち隣接するセルのレベルの差が大きいと補間の精度が低くなってしまうので，これを避けるために隣接セル間の細分化レベル差に制限を設ける．一般に 1 より大きなレベル差は許容しないという制約を設けることが多く，この条件を 2:1 バランス条件と呼ぶ

これは即ち，以下の通りにに言い表すことができる．

- 細分化レベル l のセルと細分化レベル $l-2$ の境界間には、少なくとも 1 つの細分化レベル $l-1$ のセルが存在する．

ということである．但し，そのセルが計算領域の物理的な境界である場合はこの限りでは無い．

従って Tree-based AMR では，セルの細分化が行われるとバランス条件が満たされているかをチェック擦る必要があり，満たされていない場合には，条件を満たすように周囲のセルを細分化する．この処理を，バランスする，と今後表現する．

またこの制約によって，あるセルの細分化が引き金となって連鎖的に細分化が引き起こされる現象が存在し，これを *ripple effect* と呼ぶ．**図 2** はその典型的な状況である．

図からわかるように，あるセルの細分化が，直接隣接していないセルの細分化を引き起こす．MPI などを用いて並列化を行う場合には，格子を分割して各プロセッサに静的に割り当てることが多い．これは即ち ripple effect によって，あるプロセッサ内のセルの細分化が別のプロセッサの保有するセルの細分化を引き起こすことを意味しており，MPI による並列 Tree-based AMR の難しさの 1 つとなっている．

2.2 計算の流れ

Tree-based AMR の計算の流れは**図 3** のようになる．

以下で，各計算フェーズについて述べる．

NewTree まず NewTree の部分では基礎となる均一なメッシュを生成する．即ち木構造を根からあるレベルまで展開することに相当する．

Refine/Coarsen 次に Refine/Coarsen のフェーズでは，問題固有の条件に従ってセルを細分化していく．あるいは時間が経って高解像度の必要がなくなったセルでは集約化を行う．このとき，あるセルを細分化（集約化）するか否かを判定する条件は問題によって異なる．大きく分けると *a priori* な方法と *a posterioi* な方法の 2 種類に分かれる．前者ではそのときの格子に配置された変数やそれらの変数からわかる，例えば勾配のような，情報から判定できるものである．後者では，そのときの格子を用いて一度 Solver にかけて計算を行い，その結果の残差等を用いて各セルが 1 レベル細分化されるべきか否かを判定する．前者では複数細分化レベルを一度に細分化出来るする一方で，後者では各細分化レベルで逐次的に全体計算を行う必要がある．本研究における提案手法では，両者の違いによって状況が変わることは無いと考えられるが，基本的に前者を対象とすることを付記しておく．

Balance 次に Balance では，2.1 節で述べたバランス条件を満たすための処理を行う．

ExtractMesh ExtractMesh ではこれまでに作成したデータ構造からあとの Solver で必要となるデータを抽出し，変換を行う．

Solver 最後に，Solver において実際の計算を行う．この計算法には様々な物が考えられるが，有限要素法の場合には共役勾配法等を用いることが多い．また，細分化・集約化の機構を利用してマルチグリッド法を用いる手法も近年多く研究されている．

効率的な並列化を行う際に一筋縄ではいかない箇所は Balance フェーズである．これは Balance フェーズでは各セルについて隣接するセルとの比較が必須であるため，他のフェーズと異なり並列化を行う際に他プロセスとの協調が必要であるからである．また 2.1 節で述べた ripple effect を如何に処理するかという点も難しい点である．このような背景から，本研究では Balance フェーズにおける効率的な新しいアルゴリズムを提案する．

© 2012 Information Processing Society of Japan

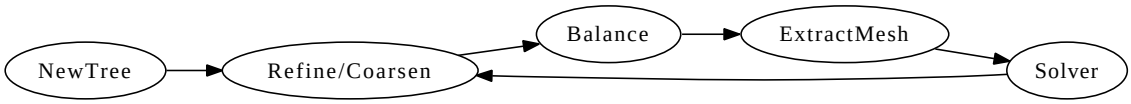

図 3 Tree-based AMR の計算の流れ
Fig. 3 Calculation flow of Tree-based AMR

3. 関連研究

Tree-based AMR の並列化に関する研究はこれまで多数行われてきた．今回は要素技術としてデータ構造とバランスアルゴリズムに着目して近年の技術動向を述べる．

3.1 Tree-based AMR におけるデータ構造

データ構造によって大きく二つに分けることができる．1 つ目はポインタベースの木構造を用いるものと，空間中の位置をエンコーディングすることで 1 次元のデータ構造を実現する Linear Octree である．ポインタベースツリーにおいては，Tree-based AMR では FTT(Fully Threaded Tree)[9] を用いることが多く，Linear Octree のエンコーディング方法には Morton Order が用いられることが多い．以下で，それぞれについて述べる．

3.1.1 FTT

FTT は初め 1998 年に Khokhlov らによって提案された [9]．FTT では各格子，すなわち木構造の各ノードに対して，子ノード，親ノード，隣接セル等のポインタを保持させる．これによって隣接セルへのアクセスのオーバヘッドを回避することができる．また，各ノードの操作は並列に行うことが可能である．Popinet らによる Gerris[10] は FTT を採用しており，近年多くの研究で用いられている．シンプルであり共有メモリ上でのプログラムではよく用いられるものの，木構造の効率的な分散配置や共有が難しいため大規模分散環境への適用は限定的である．

3.1.2 Linear Octree

Linear octree では，空間中の特定の位置位置を何らかの方法によってインデックス化し，木構造を一次元配列にマッピングする．即ち木構造の深さの最大を定義し，存在しうるノード全てを一意に順序付ける機構を用意する．その順序付けに従って一意な ID を割り振り，実際に存在するノードの情報及びその ID を一次元配列に格納することによって実現される．

順序付けの機構としては一般に SFC（Space Filling Curve）が用いられ，その中でも Morton Ordering がよく用いられる [7]．

図 4 は Morton ordering を用いた Linear Octree のイメージである．図の左側から分かるとおり，Linear Octree を用いると各セルを一次元で表現することができるため，単純にプロセス数で割った数を分配することによって均等な負荷分散が可能になることが何よりも大きいメリットである．また，SFC を用いることによってデータの局所性が高まりキャッシュを有効に使うことができるようになる．Linear octree を用いて並列化する際に重要な点はソート・検索である．これは，隣接セルにアクセスするなどの際に ID を計算し，ノードのリストから検索する必要があるからである．Sampath らは Sample sort と Bitonic sort を用い，高速な探索を実現している [11]．

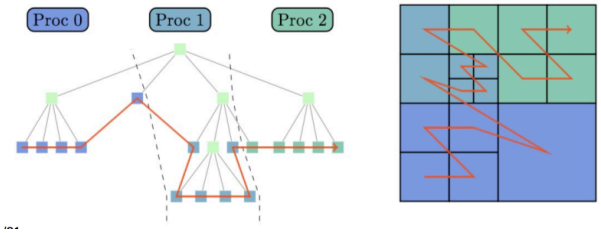

図 4 線形八分木の概念図．図中の色の違いは並列計算の際の所有プロセスを表している．[6] より引用
Fig. 4 Linear octree by morton ordering. Colors exhibits the owner process of each node/domain.[6].

3.2 バランスアルゴリズム

バランスアルゴリズムとは，2.1 節で述べたバランス条件を満たすためのアルゴリズムを指している．

3.2.1 ripple propagation algorithm

バランス条件を満たすためには，各セルにおいて隣接セルとの細分化レベルを比較することになる．このとき問題となるのは 2.1 節で述べた *ripple effect* である．即ち，あるセルの細分化によって直接隣接しないセルの細分化が引き起こされる可能性があるために，ただ単純に全セルを一度走査するだけでは不十分ということである．この問題を単純に解決するためには，バランスされるセルが無くなるまで何度も走査すれば良いがもちろんこの方法は非効率である．

そこで現在殆どの研究で用いられているのが *ripple propagation algorithm* である．このアルゴリズムでは小さなセルから大きなセルの細分化が引き起こされるという事実に着目し，小さなセル即ち細分化レベルの大きいセルから順に走査していくことによって一度きりの走査で条件を満たすことを可能にしている．このアルゴリズムは当初 Tu らによって提案され [13] たもので，近年では殆どの研究で *ripple propagation algorithm* を採用しているように見受けられる [4], [5], [6], [11], [12], [13].

しかしながらこの方法の欠点としては，並列計算を行う

© 2012 Information Processing Society of Japan

際に細分化レベルごとに同期する必要があることである．Linear Octree などを用いて各プロセスに同数のセルを配置した場合でも各レベル毎のセル数は異なるためにロードバランスが異なってしまう点や，全体通信を必要とする点がスケーラブルな並列化を難しくしている．

4. 提案アルゴリズム

本研究では，2.2 節でみた Tree-based AMR において各セルを 2:1 にバランスさせるアルゴリズムを新しく提案する．即ち，細分化・集約化が行われて 2:1 バランス条件が満たされていない八分木構造を入力として，正しくバランスされた八分木構造を出力とするアルゴリズムである．

本章では，まず 4.1 節でアルゴリズム全体を概観し，4.2 節と 4.3 節で要素アルゴリズムを説明する．その後，4.4 節で提案手法について考察を行う．

4.1 アルゴリズムの概観

提案アルゴリズムは分割統治法に基づいている．即ち，木構造の根から子に再帰関数を呼ぶことによって実現される．この再帰関数は木構造中の各ノードに対して呼ばれるものであり，呼ばれたノード以下をバランスすることになる．具体的には，以下の手続きを行う．

(0) もしリーフノードなら終了する
(1) 子ノードに対して再帰呼び出しを行う（この処理によって各子ノードの内部がバランスされていることが保証される）
(2) 隣接している子ノード間をバランスをさせる（各子ノードの面，辺，頂点と接する端のセルについてバランス処理が行われる．この処理によって親ノードがバランスされていることが保証される）

即ち，木構造に沿って再帰処理が行われ，八分木中の各ノードに対して呼び出される関数は，各ノードの内部がバランスされていることを保証する．つまり，(1) によって 8 つの子ノードの内部がバランス条件を満たしていることが保証され，(2) で各子ノード間でもバランス条件を満たすことが保証される．この関数を根ノードに対して適用することによって，全体がバランスされていることが保証できる．

次節で，子ノード間をバランスするアルゴリズムを述べる．

4.2 子ノード間をバランスするアルゴリズム

前節で述べた，内部でバランス条件が満たされている各子ノード間をバランスさせるアルゴリズムについて説明する．8 つの子ノードは互いに面，辺あるいは頂点によって隣接しているので，それぞれについてバランスされる必要がある．3 次元の場合には 1 つの子ノードは面で 3 つ，辺で 3 つ，頂点で 1 つの他の子ノードと接し，2 次元の場合には辺で 2 つ，頂点で 1 つ接することになる．

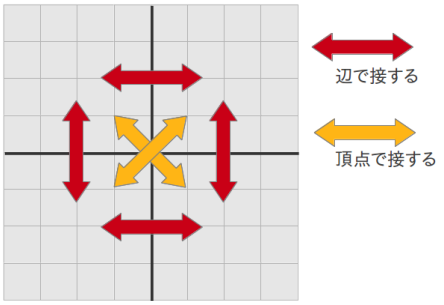

図 5 2 次元四分木において子ノードをバランスする際の概念図
Fig. 5 An image of balancing child nodes in quadtree 2D

具体的には以下のような手続きを行う．ここでは 2 次元の場合を想定する．

(1) x 軸と平行なで辺で接する 2 ノード，2 組をバランスさせる
(2) y 軸と平行なで辺で接する 2 ノード，2 組をバランスさせる
(3) 中心点で接する対角線上のの 2 ノード，2 組をバランスさせる

この手続きを 2 次元の際に模式的に表したのが**図 5** である．2 次元であるため，子ノードは辺ないし頂点で接しているが，3 次元でも同様である．

次章で面，辺あるいは頂点で隣接する 2 つのノード間をバランスするアルゴリズムについて述べる．

4.3 隣接する２つのノードをバランスするアルゴリズム

このアルゴリズムでは，隣接する２つのノードをバランスすることを考える．

ここで注意すべきなのは，あるノードの内部がバランス条件を満たしている場合，隣接ノードとのレベル差によって細分化される可能性があるのは境界（面，辺，頂点）上に存在するセルのみである，ということである．従って，例えば面によって隣接する 2 つのノードをバランスさせる場合，両ノードの該当面上に存在するセルについてのみ考えれば良い．隣接ノードによって細分化される可能性のある領域は Insulation Layer と呼ばれ，詳しい証明等は Sundar らによって行われている [12]．

本アルゴリズムでも，再帰呼び出しによる分割統治法を用いる．即ちある２ノードが面で接しているとき，リーフノードに達するまでその面に接している子ノードを再帰で呼び出し続ける，という処理を行う．その過程で，リーフノードの２レベル以上の差がある場合には，細分化を行う．

注意すべき点として，ノード中である平面に接している子ノードは４つ存在する．そのため，バランスする相手側のノードにも４つの子ノードが存在し，そのうち面で接するものが１つ，辺で接するものが２つ，点で接するものが１つ存在するということである．このうち面で接するものを再帰呼び出しすることになる．なお，説明では面で接する場合を述べたが，辺，頂点で接する場合も同様に処理可

© 2012 Information Processing Society of Japan

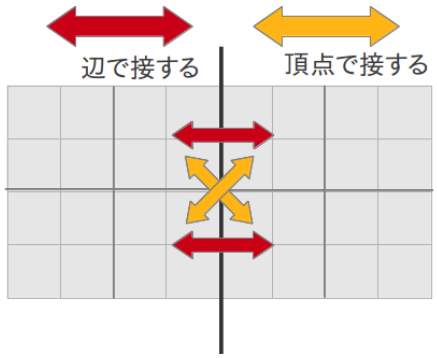

図 6 2次元四分木において 2 ノードをバランスする際の概念図
Fig. 6 An image of balancing 2 nodes in quadtree 2D

能である．これを 2 次元 4 分木の図で表したものが，**図 6** である．

従って，具体的な手続きは以下のようになる．ここでは，図と同様に 2 次元 4 分木中の辺で隣接する 2 ノードをバランスすることを考える．

- （両ノードがリーフ）もしくは（片方がリーフでも一方が孫ノードをもたないなら）なら終了
- 細分化が必要な場合は細分化を行う
- 辺で接する子ノード同士，2 組に対して再帰呼び出し（図中の赤矢印）
- 頂点で接する子ノード同士，2 組に対して再帰呼び出し（図中の黄色矢印）

簡単のために 2 次元で説明を行ったが，3 次元の場合にも同様の考え方が可能である．また，面で接する場合，頂点で接する場合も，上で説明した辺で接する場合と同様の考え方で可能である．

4.4 提案アルゴリズムの考察

従来手法では 3 章でみたように各セルに対して隣接セル探索を行い比較することによってバランスを行っていた．しかし，提案アルゴリズムでは分割統治法によるアプローチを用いるために，セル探索のためのソート・検索処理が不要となる他，木構造中の各ノード固有の値以外に 8 つの子ノードとの接続情報のみを有する簡便なデータ構造で実現可能である．

また，ripple propagation algorithm では細分化レベルごとの同期が必要だったが，提案アルゴリズムでは分割統治により各レベルでの細分化を独立して行うことができ，同期のオーバヘッドがかからないこともメリットとなる．

また再帰呼び出し部分をタスクとすることによって，極めて容易に並列化可能であり，プログラムの並列化及び最適化をインクリメンタルに行えるため，高生産に開発を行うことが出来る．

理論性能としては，全ノードを n としたとき $O(n)$ であると考えられる．

5. 評価

提案アルゴリズムを実装し，予備的な評価を行った．実験環境としては AMD の 24 コア cpu，256GB のメモリを搭載した linux サーバを用いた．

5.1 実装

逐次プログラムの実装には C 言語を，タスク並列モデルによる並列化には C 言語の拡張である Cilk[3] を用いた．並列化に際しては，4.1 章で述べたノードをバランスさせる再帰関数をタスクとして並列化を行った．タスクの呼び出し（Cilk の spawn）と待ち合わせ（Cilk の sync）のみを用いて並列化が可能であり，本質的な逐次プログラムとの差異は 10 ワード程度であった．

5.2 問題設定

評価のための問題設定は Sundar ら [12] にならって，Barnes-Hut 法における 8 分木生成に似た問題をもちいた．即ち空間中に粒子が存在しており，全てのセルが高々 1 つの粒子しか含まなく成るまで，空間を再帰的に 8 分割する問題である．

この問題設定によって作られた 8 分木をバランスする時間が今回の評価対象となる．本来は Tree-based AMR のコンテキストにそった問題を解くのが望ましいが，今回の問題設定を用いることで提案アルゴリズの理論性能を解析しやすくなると再現されており，提案アルゴリズムの性能を示すことは可能だと考えられる．

5.3 一様分布でのストロングスケーラビリティ

粒子の分布に一様分布を用いた場合のストロングスケーラビリティを示したグラフが**図 7** である．この実験で用いたのは，深さ 9 の完全な 8 分木であり，リーフノードの数は $8^9 = 128M$ 個である．また，タスク生成のオーバヘッドを回避するためタスクは深さ 5 までのものを用いた．

16 並列まではほぼ理論通りの性能を示しているが，以降は性能が頭打ちとなってしまっている．これに関しては更なる評価が必要だが，メモリバス幅がボトルネックになっていると考えられる．

6. おわりに

6.1 まとめ

本稿では，Tree-based AMR をタスク並列モデルによって並列化することを提案し，その有効性を議論した．また Tree-based AMR の計算過程で効率的な並列化が難しい，バランス条件を満たすための新しいアルゴリズムを提案した．提案アルゴリズムは分割統治法に基づいており，タスク並列を用いることによって極めて容易に並列化が可能で

© 2012 Information Processing Society of Japan

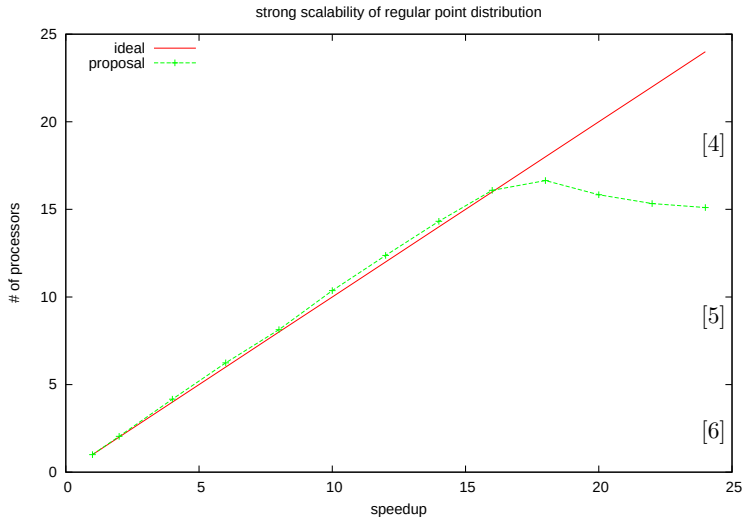

図 7 一様分布でのストロングスケーラビリティ
Fig. 7 Strong scalability of regular distribution

ある．実際にタスク並列処理系 Cilk[3] を用いて，C 言語による逐次プログラムを実装・評価を行った．その結果，予備的な評価ではあるが理論的な性能を支持する結果が得られた．

6.2 今後の課題

今後に向けて，以下のような課題が存在すると考えている．

- 分散メモリ環境での評価
- 今回は提案アルゴリズムの予備的な評価に留まっており，実際の Tree-based AMR による評価は行えていない．今後は，他の部分についても実装もしくは既存ツールと連携して，統合的な評価を行う必要がある．
- 既存手法との定量的な比較を行う必要がある
- 今回の評価では，一度バランスすることのみを想定しているが，時間依存問題や残差による誤差推定を行う場合など，変化しながら何度も計算を行うことが多い．このような場合での振る舞いや，時間方向のデータの局所性を考慮する必要がある．
- Cilk 以外のタスク並列処理系を用い，処理系の差異や Tree-based AMR で必要とされるスケージューリングポリシーやワークスティーリング機構の検討

参考文献

[1] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:64–84, May 1989.
[2] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484 – 512, 1984.
[3] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 37(1):55–69, August 25 1996. (An early version appeared in the *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '95)*, pages 207–216, Santa Barbara, California, July 1995.).
[4] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadlerg, T. Warburton, and L. Wilcox. Extreme-scale amr. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.
[5] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.
[6] Carsten Burstedde, Omar Ghattas, Michael Gurnis, Georg Stadler, Eh Tan, Tiankai Tu, Lucas C. Wilcox, and Shijie Zhong. Scalable adaptive mantle convection simulation on petascale supercomputers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 62:1–62:15, Piscataway, NJ, USA, 2008. IEEE Press.
[7] Paul M. Campbell, Karen D. Devine, Joseph E. Flaherty, Luis G. Gervasio, and James D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.
[8] Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.
[9] A. M. Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *J. Comput. Phys.*, 143(2):519–543, 1998.
[10] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *J. Comput. Phys.*, 190(2):572–600, 2003.
[11] R. S. Sampath, S. S. Adavani, H. Sundari, I. Lashukf, and G. Biros. Dendro: parallel algorithms for multigrid and amr methods on 2:1 balanced octrees. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 18:1–18:12, Piscataway, NJ, USA, 2008. IEEE Press.
[12] Hari Sundar, Rahul S. Sampath, and George Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM J. Sci. Comput.*, 30(5):2675–2708, August 2008.
[13] TIANKAI TU, DAVID R. O'HALLARON, and OMAR GHATTAS. Scalable parallel octree meshing for terascale applications. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 4–, Washington, DC, USA, 2005. IEEE Computer Society.
[14] 中島 潤 and 田浦 健次朗. 高効率な i/o と軽量性を両立させるマルチスレッド処理系. **情報処理学会論文誌プログラミング（*PRO*）**, 4(1):13–26, mar 2011.

© 2012 Information Processing Society of Japan