Vol.2019-HPC-170 No.43
2019/7/26

分散共有メモリシステムを利用した Barnes-Hut アルゴリズムの初期並列実装と性能評価

緑川 博子[†1]，柴山 悠[†1]

概要： 筆者らは，マルチコアマルチノード並列システムにおいて，グローバルビューによるプログラミングモデルを用い，高性能計算のための処理を効率的に行うことを目指している．本報告では，ステンシル計算などに見られる配列などの規則的データ構造へのアクセスを主体する従来の典型的並列処理とは異なり，ツリーデータ構造を扱う Barnes-Hut アルゴリズムによる N 体問題処理について，ポインタによるグローバルデータアクセスとメモリアクセス局所性を考慮した初期並列実装を行い，性能評価を行ったので，報告する．

キーワード：PGAS，並列プログラミング，クラスタ，ソフトウエア分散共有メモリ，グローバルビュー，大域アドレス空間，並列言語，N 体問題，Barnes-Hut アルゴリズム

1. はじめに

HPC 分野において，大規模な問題を解くためには，マルチノード・マルチコアを用いた並列処理高速化が求められる．現在，MPI＋X（OpenMP[1]，OpenACC[2]，Cuda など）の手法が広く使われているが，データがローカルビューであること，MPI によるプログラミング開発は煩雑で，生産性が低いことから．これを改善するため，共有データのグローバルビューを実現する PGAS(Partitioned Global Address Space)モデルと総称される様々な言語や API が提案されてきた[3,4,5,6].

分散メモリ型の計算機クラスタで，実装が容易で高性能が見込める並列処理は，ステンシル計算などに代表される，規則的データ構造（配列）を各ノードで等分割しデータ並列で処理する応用で，多くの場合，処理の大部分はローカルデータアクセスで，他の計算ノードの遠隔データをアクセスの頻度は少なく，アクセスタイミング，アクセス範囲，サイズなどが事前に既知であることが多い．このような応用は，PGAS 専用コンパイラにより事前に遠隔データアクセスを，MPI 片側通信などに変更することも可能で，高性能化が可能である．

一方，グラフやツリーなどの不規則なデータ構造を扱う処理では，上述のように各ノードに対しあらかじめデータを均等に分割することが困難な場合も多く，データアクセスにポインタ変数（アドレス）を使用する．また，データのアクセス順序や，アクセス範囲，アクセスのタイミングが動的で，あらかじめ予想が難しく，コンパイラによる静的解析には限界がある．このような応用には，PGAS コンパイラだけでなく，実行時のランタイム処理が重要になる．

現在広く用いられている MPI プログラム開発の困難性の一つは，分散メモリ並列システムにおいて共有データがグローバルビュー（大域名前空間）で扱えないという問題だけではなく，不規則かつ動的な処理応用における実行時の問題．たとえば，実行要求に応じた動的な遠隔データの取得や提供，通信タイミング，マルチスレッド間の計算や通信のオーバーラップ，遠隔データのローカルノードへのキャッシュ操作とデータ一貫性などを，すべてユーザが応用プログラムの一部として，作りこまなければならないことである[18]. もちろん，応用に特化した作り込みは性能上，有利であることは確かであるが，必ずしもその開発コスト（金銭，時間，必要技術の習得）は誰にでも払えるわけではない．

本報告では，ツリー構造を扱う古典的問題の一つである N 体問題（Barnes-Hut アルゴリズム[7]）を例にとり，我々の開発したランタイムシステムである mSMS（マルチスレッドプログラム対応ソフトウエア分散共有メモリ）を用い，どの程度の性能と開発の容易さを実現できるか調査した．本報告の実装は，まだ，きわめて単純な初期実装であるが，クラスタ上に共有データとして定義されたツリー構造に対するアクセスオーバーヘッドや，MPI プログラムでよく用いられる LET 法[8,9]ではなく SFC 法[10]に準拠した質点のソートによるメモリアクセス局所性の向上についても示す．

2. マルチスレッド対応分散共有メモリ mSMS

mSMS [19]は，図 1(a)に示すように，クラスタの各計算ノード上に仮想的な共有メモリを実現する．mSMS では，クラスタの各計算ノードで，実際に図 1(b)に示す共通の大規模アドレス空間を持つプロセスが実行される．mSMS では，従来の C の記法で記述することができ，C のポインタ変数やアドレスによる大域データのアクセスが可能で，データアクセス領域に制限がない．また，OpenMP，OpenACC など同時に利用することで，柔軟なマルチノード・マルチコア並列処理が可能である．

mSMS は，古典的なページベースのソフトウエア分散共

†1 成蹊大学 Seikei University.

有メモリ（SDSM）であるが，図 1(c)のように，通信スレッド，受信スレッド，ページ送信スレッドの３つのシステムスレッドを自動生成しユーザプログラムのマルチスレッド計算と同時に高性能な並列ノード間通信機構を実現する．ノード間通信には，下層通信機構によらない MPI（古典的両側通信）を利用している．mSMS において提供される仮想大域メモリの規模は，1 ノードで利用できる物理メモリ容量とノード数の積である．大域アドレス空間において，実際にローカル物理メモリにマップされている領域(オーナページ領域)以外にアクセスすると，遠隔ノードから対応するページをローカルにキャッシュする．ページフェッチは指定された SMS ページサイズ単位で行う．同期（バリア）時に各ノードのデータの一貫性を取る緩和型メモリ一貫性モデルを採用している．また，遠隔メモリアクセスの検知は，OS メモリ保護機構を利用した Segv シグナルを用いる．mSMS において，大域データ領域を確保するには，malloc とほぼ同様な sms_alloc 関数などを使用する．

mSMS のプログラミング環境として，現在，３つの API が提供しているが[20]，いずれの API も最終的には SMS ライブラリ関数使用による C プログラムへ変換される．本報告では，sms_alloc 関数とポインタを用いた C 言語で処理を記述する．

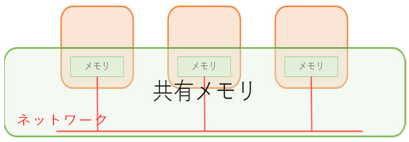


図 1 (a) ソフトウエア分散共有メモリシステム

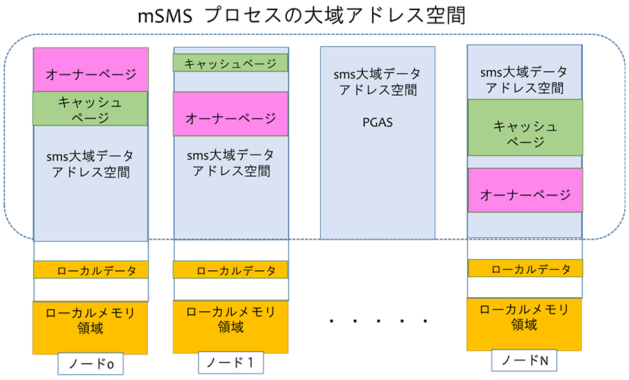


図 1 (b) 各計算ノードにおける mSMS プロセスの共有グローバルアドレス空間とローカルアドレス空間

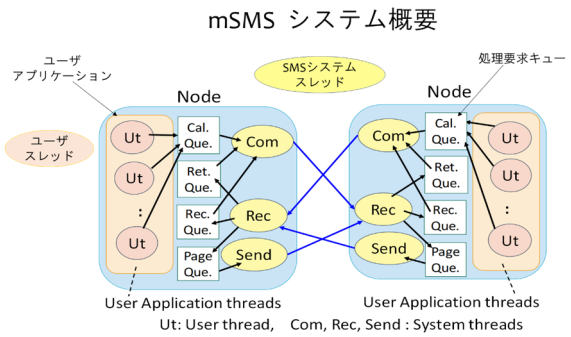


図 1 (c) mSMS システムのノード間通信機構

3. 関連研究：UPC による BH-Tree の実装

PGAS 言語の中には，ポインタ変数が使えない,あるいは遠隔データアクセス領域に制限がある言語[7]もあり，Barnes-Hut の実装が行われているものは，限られている．このなかで，UPC[4,5]は C 言語をベースとして，アクセス制限なくポインタを利用できる数少ない言語である．UPC では，mSMS と同様に，通常のデータ型の前に，拡張データ型 shared を付けることで，グローバル領域にデータ配列を宣言し，計算ノードに分散マップして，アクセスすることができる．また sms_alloc と同様に upc_alloc という動的なデータ確保のための関数も用意されている．UPC は，大域(shared)データを指すポインタと，ローカル計算ノードにあるデータをさすポインタとは型が違っており，さらに，ポインタ変数自体も，ローカル変数かグローバル（shared）変数かという型の違いがあるため，4 種のポインタ型が存在する．これらを明示したプログラムを，UPC コンパイラが静的に解析し，遠隔計算ノードへのデータ通信を持つ実行プログラムへ変換している．UPC では，グローバルポインタやグローバル配列を用いたグローバルビュープログラミングより，ローカルデータを指すポインタ型に変換してアクセスするほうが，高速になるため，性能評価ベンチマークプログラムのほとんどはローカルビューで記述される[22].

J.Zhang らは，UPC を用いて，単一計算ノード向け（共有メモリ型マルチプロセッサ向け）の Splash2 ベンチマーク[12,13]にある Barnes-Hut プログラムをなるべく変更せずにクラスタ向けに書き直す試みを行った[14,15]．しかし，ほとんど手を加えない単純な移植では，悲惨な性能になることを確認している．このため，アルゴリズムをどのように分散メモリシステム向け(すなわち PGAS モデルとして，データローカリティを意識したモデル）に，段階的に変更して，それぞれのコード変更ステップでどの程度，性能が改善されるのかについて報告している．彼らは，共有メモリプログラミングモデルの本質は，(1) shared データがどの計算スレッドからでもアクセスできること．(2) shared データがデータ名を変更せずにキャッシュできることであるとし，PGAS が適切に発展されればそれが可能なはずだと結論づけている[15].

その後，彼らは，独自に C++テンプレート関数ライブラリ PPL[17]を UPC 上に加えて，マルチコアクラスタ向けのマルチスレッド Barnes-Hut プログラムの実装を行っている[16]. 前実装[14]では，CPU コアに 1 つの UPC THREAD（実際にはプロセス）が実行され，プロセス内スレッド並列処理は行われていなかった．一方，この実装[16]では，マルチ

Vol.2019-HPC-170 No.43
2019/7/26

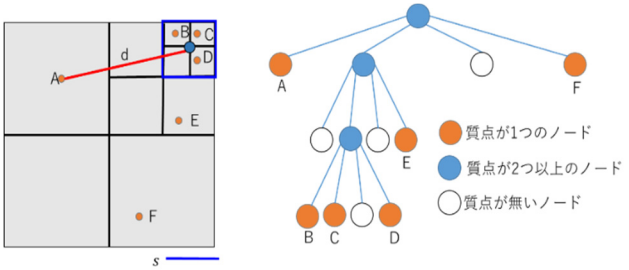
スレッドにより，通信遅延の隠蔽，計算負荷分散の改善，ノード間通信とメモリ使用量の削減を実現している．UPC の欠点（あるいは MPI 片側通信の問題？）ともいえる大域データと局所データの変換時（ローカライズ処理）の大きな同期・通信オーバーヘッドを計算で隠蔽するために，Barnes-Hut の応用プログラム内部に，様々な制御機構を組み込んでいる．ツリーを走査する複数スレッドのいずれかが遠隔データを必要とした場合には，そのデータのローカライズが終了するまで，ローカルにある他のツリー部分の走査を行う．遠隔データがそろった段階で，再度，中断したツリー処理を再開する．

このような処理のために，データローカライズ要求キューやローカライズ終了済みキューなどを用意し，これらを管理する専用スレッドを導入して，マルチスレッド制御のためのランタイム機構をプログラムに組み込んでいる．すなわち，単に UPC という C 言語に準拠した環境提供だけでは不十分で，他の応用にも役立つような，もっと普遍的なプログラミングサポート機能が必要という観点から，PPL[17]を開発している．しかし，筆者らも認めている通り，PPL ライブラリを利用したこの BH プログラム[16]を書くのは大変で，プログラム開発が容易とは言えない．

4. Barnes-Hut アルゴリズム

Barnes-Hut アルゴリズム[7]は，N 体問題などの各質点間の相互作用の計算（力計算）において，計算量 $O(N^2)$を $O(NlogN)$に減らすアルゴリズムである．図 2 は，Barnes-Hut アルゴリズムのイメージ図である．質点間に働く力の計算において，質点 A から大きく離れている質点が複数あるとき(質点 B, C, D)，離れている質点からの作用は基本的に小さいため，そうした複数の質点の集まりから 1 つの重心を求め，この重心を代表値として相互作用を計算することで，この重心を構成する個々の質点との計算を省く．どの程度距離が離れていれば重心点を代替え値として取るかは基準値 θ で制御できる．対象とする質点群の存在する領域の幅 s を，質点 A と質点群の重心との間の距離 d で除した値が，$\theta >= s/d$ の時に，重心による計算に置き換える．通常 θ は 0.4 - 1.2 程度[11]に設定される．この値が小さいほど，計算量が増加するが，シミュレーション精度は高くなる．$\theta = 0$ は，全ての質点間での計算 $O(N^2)$に該当する．

Barnes-Hut (BH)アルゴリズムの提案は古く，用いる計算機環境に応じてこれまで様々な実装や並列アルゴリズムが提案されている．(1)各プロセッサの力計算の負荷バランス，(2)各プロセッサ間で共有データ（ツリー）の効率的通信の 2 点を実現するために，ツリーをどのように複数プロセッサで分割するかがキーになる．Salmon らは，質点の存在する空間を再帰的に 2 分割していく Orthogonal Recursive Bisection (ORB)をベースとし，メッセージパッシング並列処理向け LET(Local Essential Tree)法を提案した[8,9]．各プロセッサが自分の担当する質点の力計算に必要な部分木（LET）を遠隔プロセッサから集めてから，ローカルのみで力計算を行う方法が，現在も，MPI で広く用いられている．LET は，部分木の所在がわかりやすく，明示的な通信記述を行う MPI ベースのプログラム実装に向いている．一方，SFC（Space Filling Curve）法は，空間充填曲線（Morton や Peano-Hilbert 曲線など）を利用して，ツリーにおける空間局所性を反映する部分木を各プロセッサに割り付ける手法である．しかし，データの空間においては，分割線が複雑になるため，明示的な通信記述が必要な MPI では利用しにくい面がある．SFC 法は，ツリーに対するメモリアクセス局所性を高める効果もある．



パラメタ θ　　$\theta \geqq s/d$　の時，重心で計算

図 2　Barnes-Hut アルゴリズム

本報告で用いた Barnes-Hut の初期実装では，問題データ独自の影響を避けるため，空間上の質点分布は均一として，各種パラメタにおける mSMS の遠隔メモリアクセス性能を調査した．したがって，各計算ノードへの計算負荷分散（質点の均一分散は）は，現在は領域の均一分割により行っている．アルゴリズムは，タイムステップごとに(1)木生成，(2)質点間の力計算，(3)座標更新，(4)木削除の 4 つの過程がある．(1)木生成は，空間を再帰的に 4 分割(2 次元空間の場合)して，最終的に 1 つの領域に 1 つの質点が収まるようにする．(多くの実装では，最終領域(木のリーフ)に複数の質点を割り当てていることが多い．)この分割を基に 4 分木を生成する．(2)力の計算は，基本的には，質点 i ごとに，木のリーフノードにあるすべての質点 j との力を求める．ただし，木をたどる過程で，中間ノードの重心が十分質点 i から離れている場合には，中間ノード下の部分木をたどらず，この中間ノードの重心と力の計算を行う．(3)座標更新は，(2)によって求まった力と質点の現在の座標から移動先の座標を計算し，Body 配列を更新する．(4)木削除は，新たな質点座標(3)に応じたツリーに再構築するため，(1)で作成したツリーを削除する．Body 配列は，Tree に比べるとメモリ消費は小さく，質点数が大きくなると，Tree サイズがほとんどを占める．

5. 大域アドレス空間を利用したマルチノード並列 Barnes-Hut アルゴリズム

ここでは，二次元空間に質点が均一に分布している単純な N 体問題を対象に，mSMS の大域アドレス空間を利用したマルチノード並列 Barnes-Hut アルゴリズムをスクラッチから作成した．本報告では，4 つの計算ノードを利用した初期実装における各種調査を示す．

5.1 木生成

マルチノード向け並列アルゴリズムの木生成の概念図を図 3 に示す．二次元空間を 4 分割した各領域を，各計算ノードに割り当てることで，ツリーの分散配置を行う．各計算ノードは，自身が担当する領域内にある質点について，ツリーを作成する．このため，木生成は，各計算ノードが自分の担当の部分木を並列に作成する．

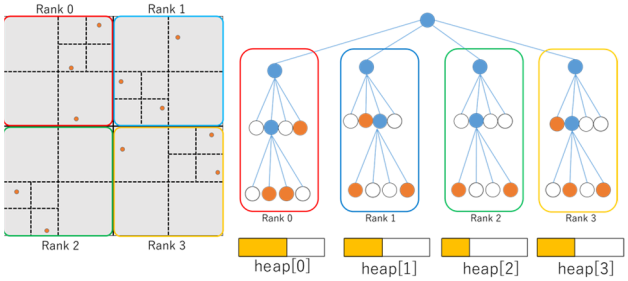


図 3　マルチノード向け木生成

ツリーは，力計算の際に，全ての計算ノードから参照できる必要があるため，各計算ノードでツリー作成に用いる各 heap 領域は sms_alloc を用いて大域データとしてそれぞれ確保し，どの計算ノードからでもアクセス可能とする．すなわち，各計算ノードが作成担当する近傍質点で構成された部分木は，ローカルノード heap に配置される．これにより，近傍質点の力計算におけるメモリアクセスも，多くがローカルアクセスになり，メモリ局所性を向上させることができる．

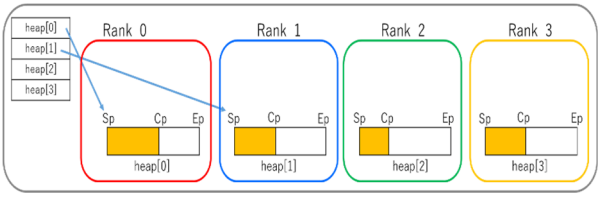


図 4　ツリーノード作成の heap 領域の分散マッピング

図 4 はツリー作成に用いる各計算ノードの heap 領域の内部ポインタを示す．左上の heap ポインタ配列 heap[]も大域データで（今回は計算ノード 0 に確保），heap[i]は，各計算ノード i にそれぞれ割り当てられた heap を指す．ツリー生成時には，各計算ノードで，ローカルの heap にツリーセルが生成され，ポインタ Cp（Current Pointer）がセル生成の度にツリーセル 1 つ分 heap の位置を移動する．

5.2 質点の受ける力の計算

質点の座標や重さなどの情報が格納されている Body 配列は，現在のところ，各計算ノードが malloc でローカルデータとして確保する．今回の実験では，簡単のため，質点の初期位置は各計算ノードが担当する範囲の質点を乱数で一様に生成している，質点の位置更新後に移動した質点も，移動後の領域を担当する計算ノードの Body 配列に挿入するため，Body 配列を他の計算ノードがアクセスする必要がないからである．

図 5 に示すように，各質点にかかる力の計算は，各計算ノードの Body 配列の質点 i が，グローバルな heap にあるツリーをたどり質点 j（あるいは質点 j を含む重心）と計算を行う．力の計算部は単純で，図 6 のように各計算ノードで，担当する body_num 個の body を，OpenMP の parallel for を用いて並列に計算する．システム全体の並列度は，指定したスレッド数と計算ノード数の積である．

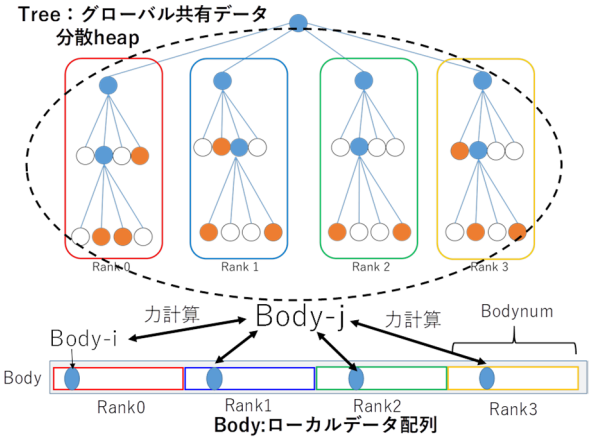


図 5　ツリーを用いた力の計算

```
#pragma omp parallel for num_threads(thread_num)
  for(i=0;i<Body_num;i++){ //質点にかかる力の計算
    struct Force force;
    force.Fx=0.0; force.Fy=0.0;
    force=cal_body(body,root,i,force,max,theta);
    body.vx[i] += dt*force.Fx; body.vy[i] += dt*force.Fy;
  }
```

図 6　OpenMP による各計算ノードの力計算

5.3 質点の座標の更新

質点 i の受ける力により座標を更新して，当該計算ノードの担当する二次元空間領域から飛び出した質点が現れた場合，図 7 に示すように，グローバルに(sms_alloc で)確保された OBody 配列に書き込む．全計算ノードがそれぞれの担当する質点の力計算と座標更新（Body 配列の更新）を終了すると，次に，各計算ノードが，他の計算ノードの OBody 配列を走査し，自分の領域内にある質点を自分のローカルの Body 配列に copy する．この更新された body 配列を，次の時間ステップの計算に用いる．

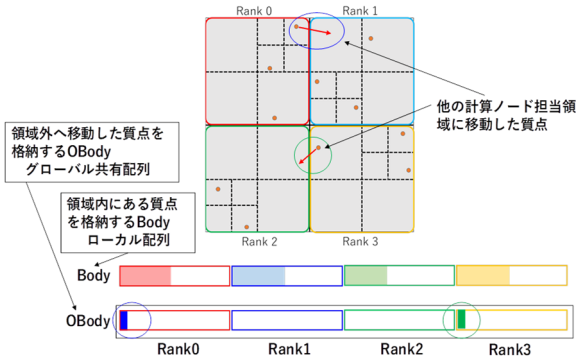


図 7　ツリーを用いた力の計算

5.4 木削除

5.1 で述べたように，ツリーは各計算ノードが担当する部分ツリーをローカル heap にツリーセルとして生成している．新たな body 配列によるツリー生成前に，古いツリーを削除するが，ここでは，単に．各 heap で移動したポインタ変数 Cp を heap の先頭 Sp に戻すのみである．sms_alloc による大域データ領域 heap の確保は，プログラム開始時の 1 回のみで，時間ステップ毎のツリーの削除は，全体処理時間において無視できるほど小さい．

6. 性能評価実験

6.1 実験環境

本報告の性能調査には東工大の Tsubame3.0 [21]を用いた．測定環境は表 1 に示す．計算ノード当たり，28 コア/56 スレッドが利用可能である．gcc 最適化は，-O3．本報告での実行時間とは，時間ステップあたりの時間を指す．

表 1　Tsubame 3.0　システム構成

CPU	Intel Xeon CPU E5-2680 v4 @ 2.40GHz * 2CPU
Num of Core / Threads	14 Core / 28 Threads
Memory	256GiB
Network	Intel Omni-Path HFI 100Gbps *4
OS	SUSE Linux Enterprise Server 12 SP2
Compiler	gcc 4.8.5
MPI	intel-mpi/18.1.163

6.2 予備実験

性能評価実験を行うにあたり，並列実行時に使用する 2 つのパラメタについて事前に調査した．

第 1 は，力計算時の並列処理スレッド数である．スレッド数が少ないと単純に処理に時間がかかってしまうが，スレッド数を増やし過ぎても，メモリアクセスと計算との比率により 1 ノード内のメモリバスに飽和が生じる，mSMS のように，遠隔メモリにアクセスする場合，ローカルメモリアクセスに比べ，さらにアクセス時間が大きいため，計算量に対しスレッド数が多すぎても，処理時間が速くならない．しかし，遠隔データのフェッチ待ちのユーザスレッドが止まっている間も，他のユーザスレッドが計算を進められるので，通信と計算を並列的に行える最適な数のスレッド数が望ましい．

第 2 のパラメタは，mSMS において他ノードへのデータ転送の単位である SMS ページサイズである．SMS ページサイズが大きければ，一度に得られるデータが大きいため，計算ノード間の通信回数は減る可能性があるが，1 回のデータフェッチに時間がかかり，無駄なデータまで持ってきてしまう可能性もある．逆に小さい場合，データを持ってくる時間は短くなるが，通信回数が多くなりオーバーヘッドが増加する．

ステンシル計算などのように，規則的なデータ構造を持つ配列とその遠隔データアクセス領域が明確である場合には，mSMS プログラム実行時に最適なページサイズを指定することも可能であるが，今回のような遠隔ノードの heap にあるツリーアクセスにおいて，どの程度の連続近傍データをフェッチすると，効果があるかは，質点の空間上の分布や 6.7 で述べるようなツリー生成時の質点の処理順序にも影響をうける．今回は，この観点から，質点の空間における分布は均一にし，問題を単純化している．

多くの MPI 実装[18]や，UPC 実装[16]では，計算に必要な部分木（LET）を明示的に遠隔ノードからフェッチする応用に特化した記述がプログラムに挿入されている．一方，今回の mSMS 実装では，sms_get のような明示的データフェッチ関数を用いず，汎用の遠隔メモリページングのような機構でユーザプログラム中のポインタアクセスに起因するデータフェッチが行われるので，通信タイミングや他スレッドとの並行性は，ユーザから隠蔽されている．しかしフェッチ粒度（SMS ページサイズ）は実行時に選ぶことができる．ローカル計算ノードに利用したいツリーの部分木がすでにフェッチされていれば，他のユーザスレッドはデータフェッチで待たされることがない．

6.3 予備実験 1 ：各計算ノードのスレッド数の選定

問題質点数 160 万，BH の計算量パラメタ θ =0.5，mSMS ページサイズ 1MB の場合において，様々なスレッド数における実行時間と，この時の遠隔ページ要求数（および Segv シグナル発生数）を，図 8 と図 9 それぞれ示す．

図 8 は，木の生成，力の計算，木の削除の 3 つの実行時間成分を示しているが，木の削除は非常に短い．木の生成は，各計算ノードでローカル Body 配列からローカルの heap にツリー生成するため，遠隔メモリアクセスが起こらない．木生成の時間はスレッド数増加により，多少短縮するものの，ほぼ一定である．力計算では，BH アルゴリズムの効果により，木生成時間と同等レベルまで短縮化されている．トータル実行時間では，スレッド数 12，16，32 が高速である．少なくとも，この質点数では，40 スレッド以上のスレッドを用いると，明らかに力計算の時間は低速化する．

図 9 は，全スレッドの Segv シグナルハンドラのコール回数と実際の遠隔ページ要求数を示す．遠隔ページ要求数は，処理が同じなので，各計算ノードのスレッド数によらず一定である．複数のスレッドが要求するページが同一の場合，実際の遠隔ページ要求は最初のスレッドのアクセス時にしか起こらないようにしているためである．ローカルにない SMS ページにスレッドがアクセスすると，そのスレッドは，Segv ハンドラに飛び，ページが来るまで中断する．遠隔ページを受け取った時点で，そのページを待つスレッドがすべて起こされる．図 9 からわかるように，スレッド数が増えると，同一ページを要求するスレッドが増え，Segv 発生回数が増える．各計算ノードの合計 Segv 数で見ると，16 スレッドでの処理が，32 スレッドでの処理に比べて，約 45%少なかった．しかし，実行時間は，32 スレッドのほうが 16 スレッドよりも 5.6%ほど高速である．

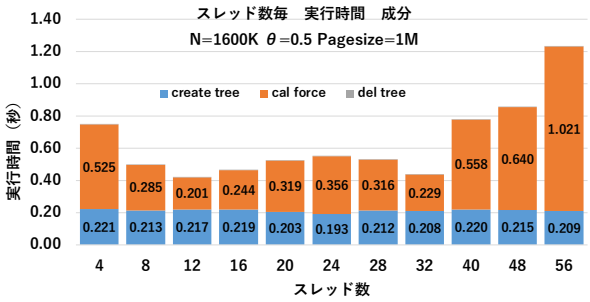


図 8 計算ノード当たりのスレッド数と実行時間

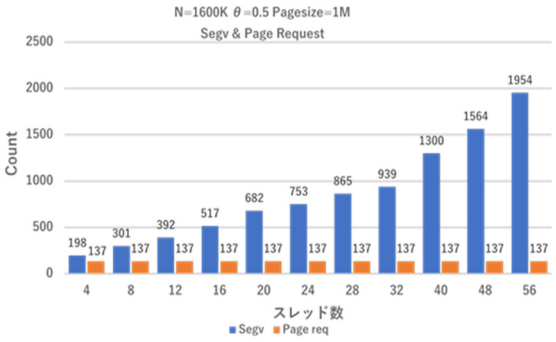


図 9 計算ノード当たりのスレッド数と Segv 数と遠隔ページ要求数

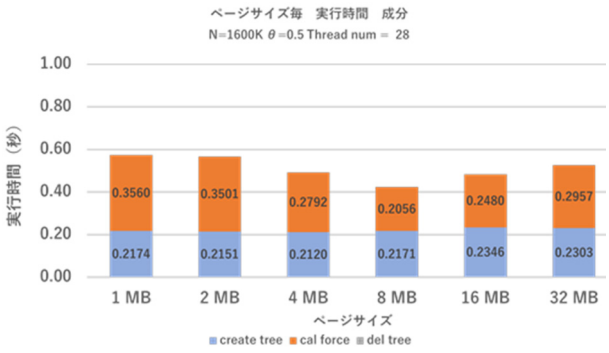


図 10　SMS ページサイズと実行時間

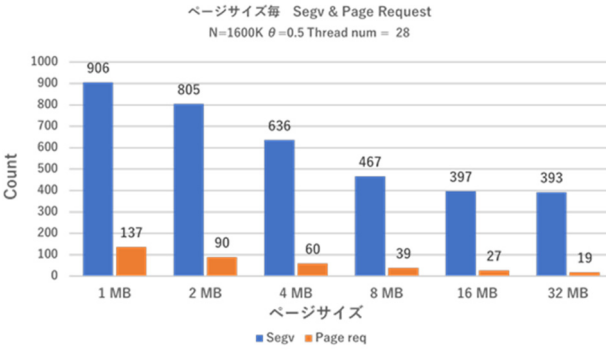


図 11 SMS ページサイズと Segv 数と遠隔ページ要求数

6.4 予備実験 2　SMS ページサイズの選定

図 10 と図 11 は，6.3 節と同様に，問題質点数 160 万，Barnes-Hut パラメタ θ=0.5 の場合における 1 ステップ当たりの実行時間と遠隔ページ要求数における SMS ページサイズの影響を示す．スレッド数は 1 つの計算ノードに搭載されている CPU コア・スレッド数(56)のちょうど半分，スレッド数 28 を用いている．

図 10 によれば，SMS ページサイズが 8MB の場合が最も高速であった．この時の Segv 数はページサイズ 1MB の時と比較し，約 50%に減少し，実際の遠隔ページ要求は 1MB のページサイズの時（137 回）の 28%（39 回）になっている．

本実験では，力計算では，16 スレッドを用い，mSMS ページサイズは 8MB で行うこととした．

6.5 Barnes-Hut パラメタ θ による影響

Barnes-Hut アルゴリズムにおいて計算量を減らすパラメタ θ により，実際に，SMS プログラムの実行時間，力計算における質点毎の平均計算相手質点数，遠隔計算ノードにあるツリーをアクセスする際の Segv 回数と実際のページリクエスト送信回数がどのように変化するのか調査した．図 12,13,14（縦軸は対数表示）は，θ が 0 から 1 までの値に対して，時間ステップ当たりの実行時間，1 質点当たりの計算相手数の平均，Segv 数と遠隔ページ要求数を示している．θ＝0 は，全質点間での力計算をすべて行い計算時間が膨大になるため，質点数は 80 万個としている．θ により，計算相手数（質点数）が急速に減少するため，力計算の実行時間も指数的に高速になっている．遠隔ページ要求数と Segv 数についても，30%ほど減少しているが，実行時間への効果は相対的に少ない．

6.6 問題規模による処理時間

問題規模による処理時間を調査した．質点数 160～2560 万個の問題における θ=0.5 の時の実行時間と Segv 数（ステップあたりの値を質点数で除した質点数毎の値）を図 15,16 に示す．質点数が多くなるほどツリー作成時間，計算時間が増えるのはもちろん，他ノードへのアクセスが増え

るため，Segv 数，ページリクエスト数共に増加する．また，質点数 2,560 万の時，heap のサイズは 1 ノード当たり 6.8GB であった．今回の質点が均一分布であるためツリー形状はバランスしていて小さいが，不均一分布になるほど，メモリ消費は高くなる．

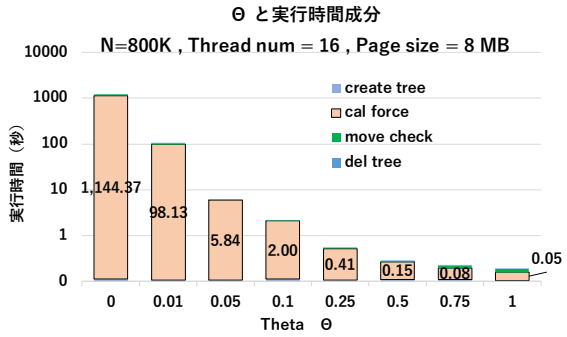


図 12 実行時間と θ

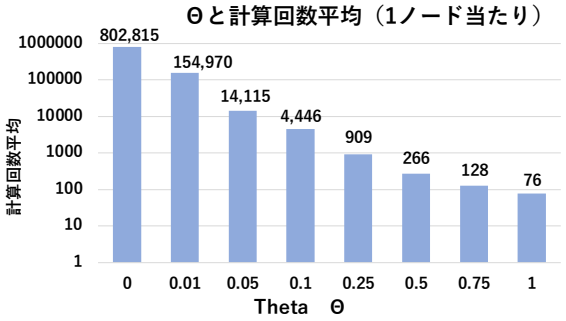


図 13 1 質点あたりの平均計算相手数と θ

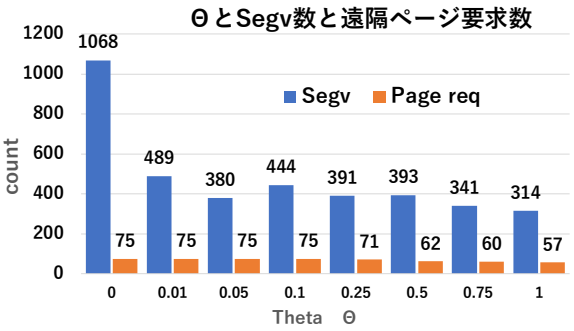


図 14 Segv 数と遠隔ページ要求数と θ

6.7 メモリアクセス局所性向上手法の導入

BH アルゴリズムでは，質点の座標(Body 配列)からツリーを生成する．そのツリーを左から順にアクセスすることは，空間充填曲線 SFC によるアクセスと同じ効果となる．このツリーを用いて，SFC 順に Body 配列の質点を並び替えることによって，ツリーアクセスにおけるメモリアクセスの局所性を向上することが可能になる．これにより少ない SMS ページ要求で必要なツリーがまとまって取得できることになる．ここでは，BodySort と呼び，一連の流れを図 17

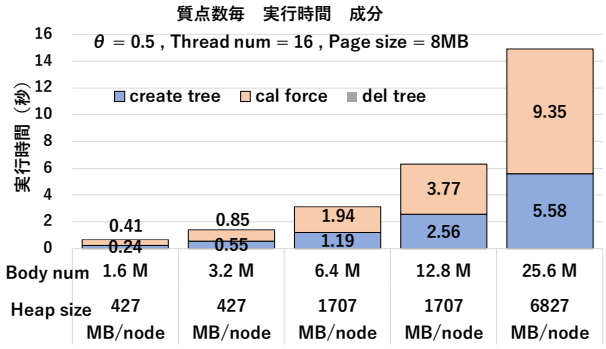


図 15 質点数毎の実行時間

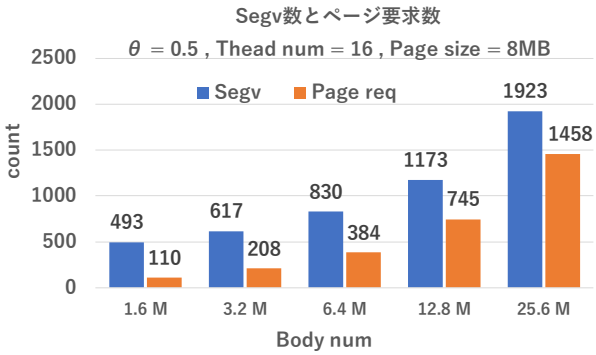


図 16 質点数毎の Segv 数と遠隔ページ要求数

に示す．図 17 の上部の body 配列が，作成したツリーを利用して，下部の body 配列に並び変えられる．元の Body 配列の質点を順番に力計算する場合，ツリーへのアクセス順序がバラバラで，アクセス局所性が低くなるが，この手法を用いることによって，近い質点同士が Body 配列のメモリアドレス空間上近くに来るように並び替えられるため，ツリーの階層構造（heap におけるアクセス）のメモリアクセス局所性が向上する．

BodySort の有無による実行時間と Segv 数・遠隔ページ要求数の違いを図 18,19 に示す．質点数 4 億個(400M)の場合，BodySort なしの場合と比べ，3.63 倍速度が向上し，ページ要求数は，1/16 と大幅に減少した．この時のメモリ使

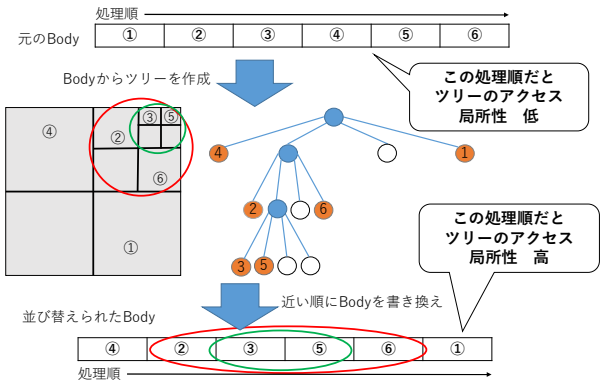


図 17 BodySort の効果

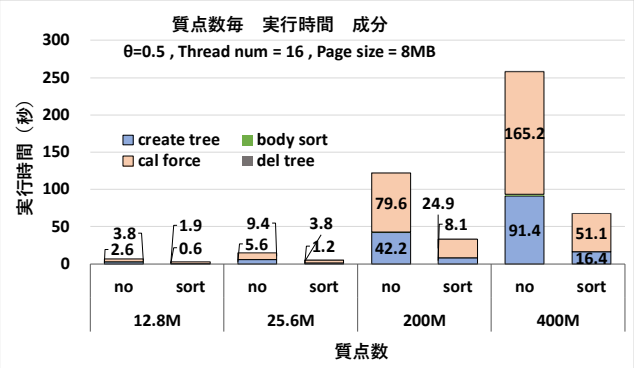


図 18 質点数毎の実行時間比較

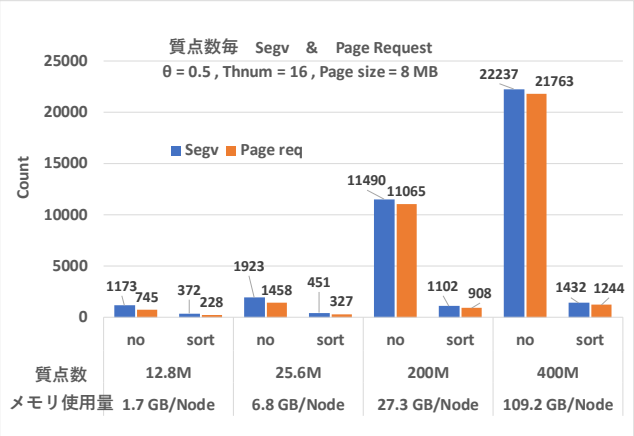


図 19 質点数毎の Segv 数，ページ要求数比較

用量は 1 計算ノード当たり 109GB となる．

7. 共有アドレス空間利用のプログラミング

7.1 単一ノードマルチスレッド版との実行時間比較

マルチノード並列アルゴリズムをそのまま用いて，単一ノードマルチスレッドで動作するプログラムを作成し，比較を行った．マルチノード版と単一ノード版の違いは，(1)heap の確保を sms_alloc から malloc に変更し，(2)マルチ計算ノードとスレッド並列を(4 ノード×16 スレッド)を，単一計算ノードにおける 2 段階のスレッド並列（1 ノード×4 スレッド×16 スレッド）へ変更したのみで，並列度は 64 で同一である．使用する関数（木作成や力計算），データ構造は，両社とも同一である．単一ノード版では，ローカルメモリのみのアクセスとなるため，ツリーのアクセスが高速になる一方で，コア数やメモリ容量に上限がある．また，スレッド数が多くなると，メモリバスの飽和が起きやすい．図 20 に単一ノード版(左)とマルチノード版(右)の実行時間を示す．並列度は 64 で同一であるが，どの質点数においても，mSMS によるマルチノード版の方が単一ノード版に比べ，1.5～1.7 倍の速度向上が見られた．

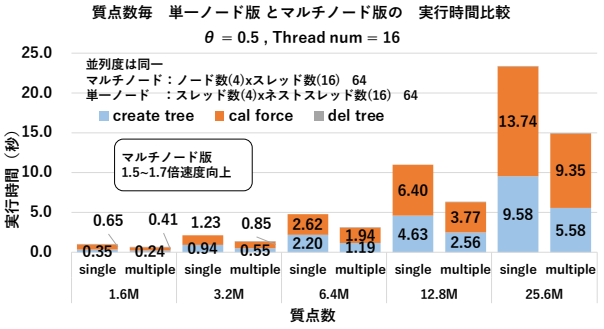


図 20　単一ノード版とマルチノード版の実行時間比較

7.2 mSMS マルチノード並列プログラムの記述性

マルチノードプログラム開発の生産性の観点から．単一ノード版スレッド並列プログラムとマルチノード版プログラムの記述を比較する．

図 21 はこの 2 つのプログラムによる Barnes-Hut アルゴリズムの記述のスケルトンである．この 2 つのプログラムの相違点は，①単一ノード版のプログラムにおいて OpenMP のスレッド並列セクションの開始と終了を，ノード並列開始関数 sms_startup，終了関数 sms_shutdown に変更し，②処理の分担をスレッド番号で指定している部分をノード番号 sms_rank に変更する，③heap 領域の確保を malloc から sms_alloc 関数に変更するだけである．力の並列計算部分など使用している関数やデータ構造は全く同じものを使用している．すなわち，マルチノード利用プログラムであっても，計算ノード間の通信記述や計算と通信のオーバーラップなど，制御のためのスレッド動作記述が，一切ない．反対に，単一ノードで動作するプログラムを mSMS を用いたマルチノード並列プログラムに変換することも非常に容易であることがわかる．

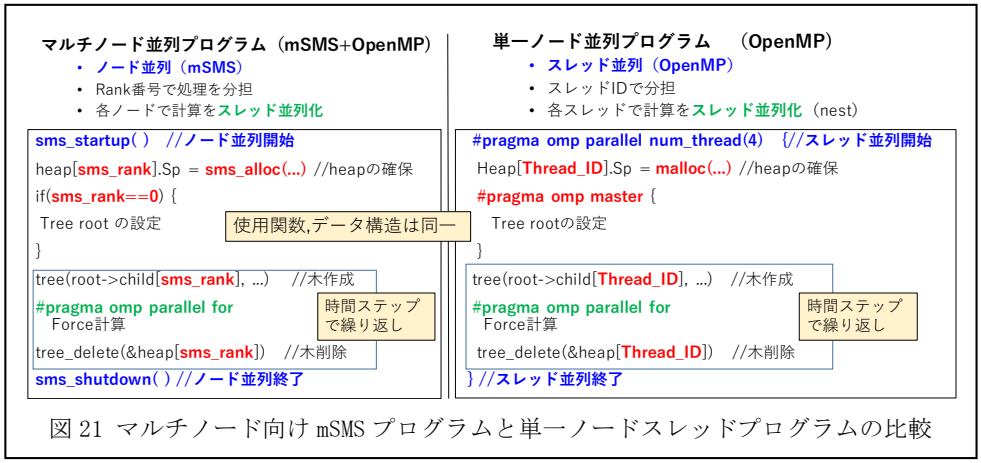


図 21 マルチノード向け mSMS プログラムと単一ノードスレッドプログラムの比較

8. おわりに

本報告では，マルチノードにおける Burns-Hut アルゴリズムの初期実装を行い，ツリー作成用に大域メモリ heap を用い，メモリアクセス局所性の向上を図った，ローカルメモリのみにアクセスする単一ノード上で動作する同一のプログラムと比較したところ，質点数 25.6M 個の時において，1.5～1.7 倍の速度が向上した．さらに，メモリアクセス局所性を向上させる BodySort 手法を用いることで，用いなかった場合に比べ，最大 3.63 倍速度が向上した．

また，mSMS による並列プログラムは，UPC をベースにした既存実装[16]に比べ，プログラム複雑性が著しく改善できることが確認できた．

PPL を用いた UPC 実装（PPL-UPC）[16]は，3 次元空間向けに 8 分木を使っており，初期データは星座分布の人工データとしてよく使われる Plummer モデル（中心部が密で周辺部が疎の分布）を使い，我々と同様に SFC 順のデータアクセスを導入している．ツリーリーフには最大 10 個の質点を含むことができる．PPL-UPC[16]によれば 4 計算ノード（12 スレッド/node），θ =0.5，質点数 4M 個（1M 個/node）において，1 ステップの力計算のみの時間が，約 5 秒弱である．

本研究では，2 次元データ向け 4 分木であり，質点の分布は，均一分布を利用していており，ツリーリーフに質点 1 個を割り当てているため，単純な比較はできないが，4 ノード（16 スレッド/node），質点数 6.4M 個，θ =0.5 で，1 ステップ全体（ツリー作成，力計算，ツリー削除を含む）で 1.3 秒 ，力計算部分のみでは 0.3 秒となっている．

筆者は，本報告の実装を 3 次元空間に拡張した 8 分木による実験も開始しており，この結果によれば，64 ノード（ノード当たり 32 スレッド）利用時，力計算時間は，質点数 819M 個で 45.52 秒，質点数 64MB 個で 4.77 秒となっている．PPL-UPC[16]の結果では，質点 64MB 個，64 ノードで約 5.8 秒である．

今後，不均一分布の質点に対して，各計算ノードの負荷を均一にする質点の配分を行う機構を導入し評価する必要がある．しかし，汎用なランタイムとして mSMS を利用することで，大域共有アドレス空間を利用し，ノード間通信やスレッド間の計算・通信の並行実行などを，ユーザが明示的に記述することなしに，動的かつ不規則構造データを扱う Barnes-Hut 処理で，一定レベルの性能が得られる可能性があることがわかった．

参考文献

[1] OpenMP https://www.openmp.org/
[2] OpenACC https://www.openacc.org/specification
[3] M.D. Wael, et al.: “Partitioned Global Address Space Languages”, Journal of ACM Computing Surveys (CSUR), Vol.47, No.62 (2015)
[4] Berkeley UPC http://upc.lbl.gov/ ver.2.28.9(2018.7.20)
[5] “Tarek el-ghazasi, et al. “UPC Distributed Shared Memory Programming”, WILEY, 2005, ISBN-10-471-22048-5
[6] Xcalable MP http://www.xcalablemp.org/ja/
[7] Barnes, P Hut：”A hierarchical O (N log N) force-calculation algorithm”, Nature, volume 324, pages 446–449, (1986)
[8] J.K.Salmon: “Parallel Implementation of the BH Algorithm”, ,PhD. dissertation, Phys., Math. Astron. Dept., California Inst. Technol., Pasadena, CA, USA, (1991)
[9] M.S.Warren, J.K.Salmon: “Astrophysical N-body Simulations Using Hierarchical Tree Data Structures”,SC12, IEEE Computer.Soc. (1992)
[10] MS Warren, JK Salmon, “A parallel hashed oct-tree n-body algorithm”, Supercomputing '93, proc. of the 1993 ACM/IEEE Conference on Supercomputing, pp.12-21, (1993)
[11] J. P. Singh, J. L .Hennessy, A. Gupta, “Implications of Hierarchical N-Body Methods for Multiprocessor Architectures”, ACM Trans. on Computer Systems, Vol.19, No.2, pp.141-202 (1995)
[12] S.C.Woo, M.Ohara, E. Torrie, J. P. Singh, A.Gupta: “The SPLASH-2 Programs: Characterization and Methodological Considerations”, ISCA '95 Proceedings of the 22nd annual international symposium on Computer architecture, pp. 24-36
[13] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta,“, Methodological considerations and characterization of the splash-2 parallel application suite,” in Proc. 22nd Annu. Int. Symp.Comput. Archit., 1995, pp. 24–36.
[14] Junchao Zhang, Babak Behzad, Marc Snir:: “Optimizing the Barnes-Hut algorithm in UPC”, SC '11: Proc. of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp.1- 11, (2011)
[15] [UPC-BH-SLIDE] Junchao Zhang,et al.:Optimizing the Barnes-Hut Algorithm in UPC ,SC11, November 17, 2011,UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN [2019.6 online] https://pdfs.semanticscholar.org/972d/713118c33b26ed718cd734458ce7d6225ea5.pdf
[16] Junchao Zhang, Babak Behzad, Marc Snir:: “Design of a Multithreaded Barnes-Hut Algorithm for Multicore Clusters”, IEEE Transactions on Parallel and Distributed Systems, Vol. 26 , Issue: 7 , July 1 2015, pp.1861 – 1873, (2015)
[17] “PPL:an abstract runtime system for hybrid parallel programming”, ESPN15, 1st Intl. Workshop on ExtreemScale Programming Models and Middleware, ACM, pp.2-9, (2015)
[18] J. Bedorf, E. Gaburov, M.S.Fujii, “24.77 Pflops on a Gravitational Tree-Code to Simulate the Milky Way Galaxy with 18600 GPUs”, Proc. of 2014 International Conference for High Performance Computing, Networking, Storage and Analysis, SC14, pp.54-65, (2014)
[19] 緑川博子：″ソフトウエア分散共有メモリシステムmSMSによる大規模マルチコアノードにおけるステンシル計算″, 情報処理学会，ハイパフォーマンスコンピューティング研究会報告（HPC）,Vol 2018-HPC-165, No22, pp.1-9 ， (2018)
[20] 阪口裕梧，西矢和生，緑川博子 ：″逐次プログラムからマルチコア・マルチノード並列処理への変換を容易にするディレクティブベース API SMint″，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 2018-HPC-167,No 5,pp.1-9,（2018）
[21] Tsubame3 http://www.gsic.titech.ac.jp/tsubame3
[22] Parallel Research Kernels GitHub, https://github.com/ParRes [2019. Online]