jh190039-ISH

# 高性能，高生産性を実現する大規模メモリ・並列処理システムソフトウエアの研究

緑川 博子（成蹊大学）

**概要**

大規模計算クラスタシステムにおいて，巨大な大域共有メモリを実現するランタイムシステム mSMS とそれを利用するための API を構築し，高性能な並列プログラムを容易に開発できる環境を実現した.C を基本とする他の PGAS 言語との比較では，プログラム記述性と実行性能ともに，最も高性能であるばかりか柔軟性の高いプログラム記述が容易にできることを示した．また従来，困難であったクラスタにおけるグローバルツリーを利用する Barnes-Hut を実現し，大規模 N 体問題を容易に記述し，高性能実行できることを示した．音響解析処理では，記述の容易さと性能を両立できる事が明らかにした．さらに，クラスタにおけるトランザクショナルメモリの実現にむけての第一次実装と評価を行った．最後に，消費電力低減に向けて，大規模システムにおける GPU 消費電力のばらつきを調査し，GPU 消費電力低減のための指針を示した.

## 1. 共同研究に関する情報

### (1) 共同研究を実施した拠点名

東京工業大学 (Tsubame3.0)

九州大学 (ITO-B)

### (2) 共同研究分野

**ロ 超大規模情報システム関連研究分野**

### (3) 参加研究者の役割分担

**成蹊大学グループ**：

緑川 博子：高性能計算・システムソフトウエア設計，構築，評価

阪口 裕梧：mSMS むけ並列処理 API 開発

**九州工業大学グループ**：

高橋 公也：音響解析

田畑 諒也：mSMS による音響 FDTD(2,4)法の高効率実装手法の開発

**電気通信大学グループ**：

三輪 忍：システム消費電力調査，削減

大八木 哲哉：mSMS＋GPU 消費電力調査

**名古屋工業大学グループ**：

津邑 公暁：トランザクショナルメモリ(TM)による実行方式の検討

飯田 凌大，川口 優樹：TM のハード・ソフト実装の協調方式の検討と実装

二間瀬 悠希，小林 龍之介：TM における一貫性制御緩和方式の検討と実装

## 2. 研究の目的と意義

大規模クラスタシステムにおいて，以下の４つを実現するシステムソフトウエアの構築を目的とする.(1)計算ノードメモリを超える大域共有アドレス空間の実現と大規模データの高速アクセス，(2)クラスタシステムにおける並列プログラム開発の生産性向上，(3)従来の配列の同期的並列処理にとどまらず，不規則構造データに対する非同期並列処理の効率実行，(4)消費電力低減.

これにより，幅広い応用分野における大規模データの高性能計算が可能となる.

## 3. 当拠点公募型研究として実施した意義

第一に，応用分野からシステムソフトウエア，省電力などの異なる分野の研究者によるコラボレーションが可能であったこと．第二に，各拠点の高性能計算システムにおいて，普段，予算的に難しい多数ノード，多数 CPU，GPU を利用して実装実験，性能評価ができたことである．具体的には，TSUBAME3.0 では,1 ノードに搭載された 4GPU を連携する NVLINK などを利用する実験，多数ノード利用による大規模メモリを扱う大規模シミュレーション，これまでにない多数 GPU を利用した大規模消費電力評価実験など

が可能であった．また，ITO-B では，準占有利用により ITO の利用規定範囲で、長時間の音響解析シミュレーションを行うことが可能であった．また，ITO-B は,CPU に Skylake-SP を搭載しており，トランザクショナルメモリ命令を利用することも可能であった．

## 4. 前年度までに得られた研究成果の概要

今年度採択のため,該当せず.

## 5. 今年度の研究成果の詳細

### (1)分散共有メモリ並列システム（mSMS）の通信高性能化および高性能 API の構築と他言語との比較

遠隔データの高速プリフェッチ機能を提供する新たな preload,overload 関数，グローバルデータへのファイル入出力関数などを開発した.

また,逐次プログラムにpragma文を加えるだけで,容易にマルチノード並列プログラムに変換できる API として SMint を提案し，OpenMP や OpenACC と自由に組み合わせて，利用環境や応用分野に応じてマルチノードマルチコアプログラミングが容易にできる環境を構築した．SMint(mSMS)では，定義できるグローバルデータサイズに制限がなく(C ポインタの扱える範囲までのデータ領域が定義可能で，実際には，利用ノード数と各ノードの物理メモリ容量の積まで.)，グローバルデータのどのような領域へも各ノードの各スレッドからアクセスできる.

C を基本とした代表的な PGAS 言語である UPC と XcalableMP を Tsubame3 に実際にインストールして詳細に比較調査を行った[1,6]．図 2,3 にその結果の一部を示す.比較には，比較的小規模な二次元データ５点ステンシル計算を採用した．この理由は，後述のように，１）UPC のグローバル記述で利用可能なグローバルデータサイズには制限があり大規模データ処理ができないこと，２）XcalableMP では，実際にはグローバルデータのどこにでもアクセスできるわけではなく，各ノードの隣接したグローバルデータにしかアクセスできず，記述できる応用が限られているためである．図２は，この条件下で，各種言語の比較を行った．用いるスレッド数とノード数によっても各言語の性能が異なるため，この図では，ノード数を横軸に，各言語でもっとも最適だったスレッド数を用いた最適値を示している．この結果，全ノードにおいて，SMint が高性能を獲得している．また，各言語の記述性の詳細は紙面の都合で割愛するが，全体のソースコードと性能の比較を表１に示す．(詳細は[1][6]を参照)

UPC グローバル型記述（表 1,1 行目）では，記述がシンプルになるものの，実行性能は劣っている．また，実際に利用できるグローバルサイズに実装上の制限があり，全体として数十 GB を以上のグローバルデータを扱うことが難しい．一方，XcalableMP は，記述も簡単で，SMint と同様に高性能で，大規模なグローバルデータ宣言も可能であるものの，実際にアクセルできる領域は，自ノードの隣接領域に限られており，ステンシル計算のような近傍処理応用以外は実装できず，適用可能な応用は非常に限られる.

UPC で大規模なグローバルデータを扱うには upc_alloc を利用したポインタによる記述(表 1,3 行目）が必要であるが，UPC では言語仕様上，shared data とノードローカルデータは型が異なり,合計４

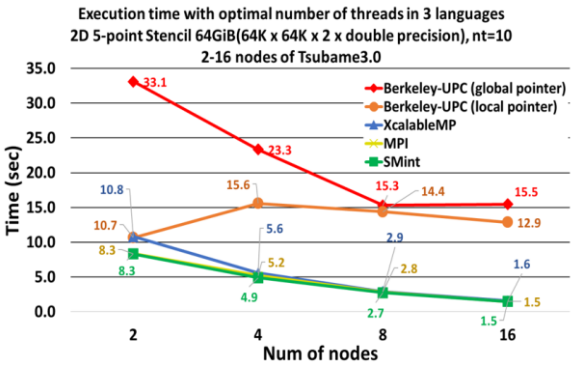



**図１ 各種言語（UPC,XcalableMP,MPI,SMint）の性能比較**

（各言語のそれぞれの最適スレッド数で実行したときの各ノード数での５点ステンシル計算の Tsubame3 の性能）

**表１ 各種言語（UPC, XcalableMP, SMint）の比較**

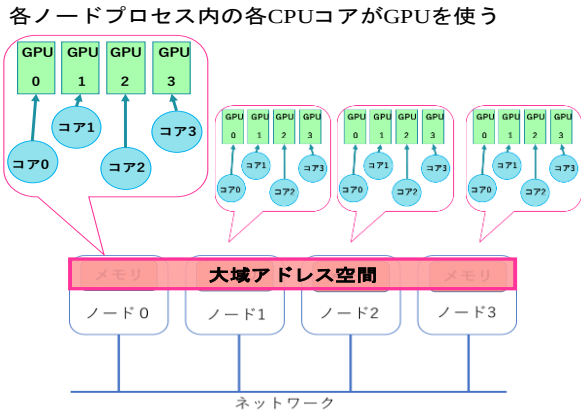
| 各種言語 | プログラムの記述性 | | 性能 | 特徴 |
|---|---|---|---|---|
| | 逐次コードに付加 | | SMintに対する実行時間の比 大きいほど悪い | 扱えるグローバルデータサイズ、アクセス可能領域、記述の制限など。 |
| | 行数 | 増加率 | | |
| UPC (Global view model) | 4 | 5% | 3.5 - 10.4 | グローバルデータサイズに制限有 小さい |
| UPC (Local view model) | 20 | 20% | 1.2 - 8.6 | MPIと同様のノードを意識したローカル記述 |
| UPC (動的確保 upc_alloc) | 28 | 28% | 9.4 - 77.0 | 低速，コードが複雑，グローバルデータのサイズ、アクセス制限なし |
| XcalableMP | 12 | 12% | 1 - 1.2 | 高速，グローバルデータのアクセス可能領域に制限あり |
| SMint | 4 | 5% | 1 | 高速，グローバルデータのサイズ、アクセス制限なし |

種類のポインタ型が存在する．このため，型変換も必要でプログラムの複雑になり，実行も低速になる．

これ以外に，UPC で大規模グローバルデータを扱うには，UPC ローカル記述(表 1, 2 行目）を利用するしかないが，これは，MPI と同様に，ノード内部はローカルデータとして記述，他ノードのデータはグローバルデータへアクセスというような別々の記述が必要となる．

### (2) mSMS とマルチ GPU 処理方式の実装と性能調査

1 ノードに複数 GPU を持つシステムにおいて，mSMS を利用することで，ノード全体にグローバルデータを配置して，容易にマルチノードマルチ GPU 処理が実現できることを示した[9]．




1 ノードに複数GPUが搭載されてる場合の利用イメージ

(a) マルチノードマルチ GPU システム

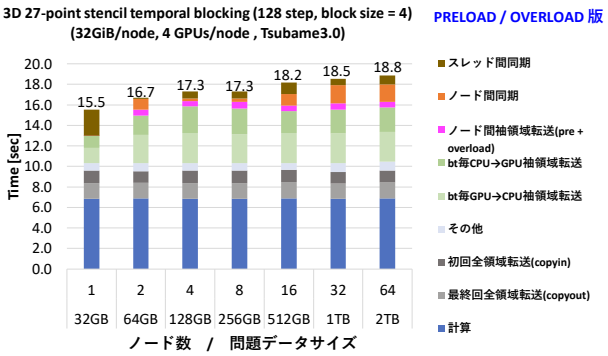
```
#include <stdio.h>
#include <omp.h>
#include <openacc.h>
#include <sms.h>
...
int main(int argc, char **argv)
{
  sms_startup(&argc, &argv);
  int numgpus = acc_get_num_devices( acc_device_nvidia );
  ...
#pragma omp parallel num_threads(numgpus)
  {
    int tid = omp_get_thread_num();
    acc_set_device_num(tid, acc_device_nvidia);
    #pragma acc enter data copyin(…) create(…)
    #pragma acc kernels
    {
      ...
    }
    #pragma acc exit data copyout(…)
  }
  sms_shutdown();
}
```

ヘッダインクルード
マルチノード実行
マルチコア実行
GPU実行

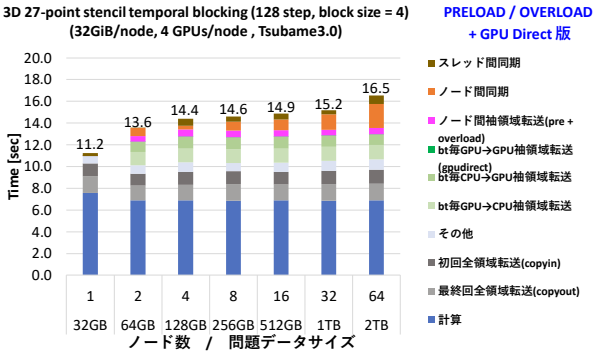
(b) SMS 関数 + OpenMP + OpenACC による記述

図 2 マルチノードマルチ GPU システムの利用例

図 2 に，想定するシステムと対応するプログラムの概要を示す．この例では，sms ライブラリ関数と OpenMP と OpenACC を利用している．文献[6]に記述はないが，1 ノード内のマルチ GPU 処理において，ホスト-GPU 間のデータ通信を高速化するため，CUDA コードを加えて，GPUDirect による GPU 間直接通信を用いたプログラムも作成し評価している．評価には，東工大 Tsubame3（InfiniBand）と東大 Reedbush-L（Omni-Path）を用い，異なる環境においても，SMS, OpenMP, OpenACC, CUDA を共存させて，ユーザが利用するシステムのハードウエア環境に応じた並列処理が可能であることを示した．




(a) ホストメモリ経由の GPU データ交換




(b) ノード内 GPU-Direct 利用時によるデータ交換

図 3 Tsubame3.0 27 点ステンシル実行時間

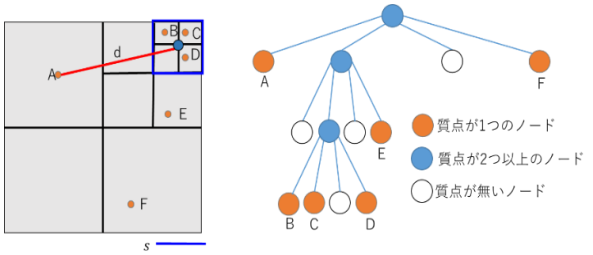
図 3 は，Tsubame3 における 27 点ステンシル計算（時間ブロッキング利用）の GPU 利用時の性能を示す．GPU ダイレクトを利用すると（図 3(b)），利用しない場合（図 3(a)）に比べ，ホストーGPU 間のデータ転送コストが低減できる．1 ノードに搭載する GPU のメモリ総量(Tsubame3: 4GPU x 16GB = 64GB)にあわせて，各ノードの処理データ量を定めている．1 ノードの主メモリ（Tsubame3：256GB）を利用できるマルチ CPU コア処理に比べ，1 ノードが担当する処理データ量は小規模になるため，CPU 利用時と同程度の問題サイズを GPU 処理するのであればより多くのノード台数が必要になる場

合がある.

### (3) 不規則構造データとポインタ利用する応用の実装を可能にする mSMS

クラスタによる並列処理のうち, 配列などの静的かつ規則的データへの同期的処理では, 各ノードが処理するデータ領域が既知で事前にノードにデータを割り当て効率よく処理することができる. 一方, ツリーの生成やアクセスのように, 不規則なデータ構造をもち, 実行時にアクセス先が決まる応用に対しては, 事前にデータを分割して配置することは難しく, 効率のよいアルゴリズムやプログラムの構築は容易ではなく, 大きなコストがかかる

このような応用の一例として, N 体問題に広く用いられている Barnes-Hut がある（図 4）. 質点の位置に対応したツリーを構築し, 質点同士の力を計算するが, 一定の条件のもと, 遠い距離にある質点群の計算はその重心で代替えして, 計算を効率化する.

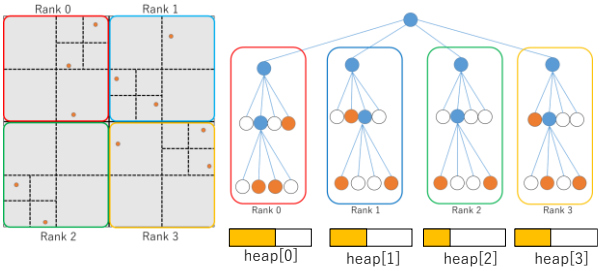



パラメタ θ　θ ≧ s/d　の時, 重心で計算

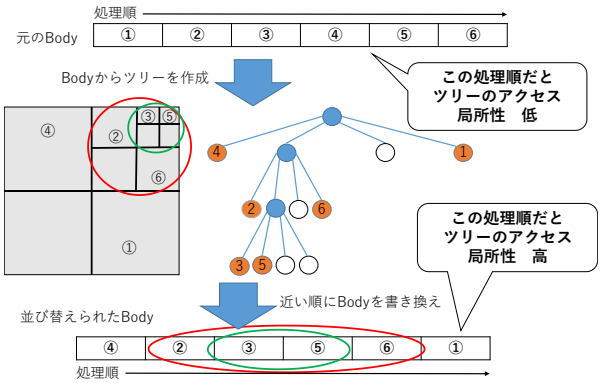
図 4　Barnes-Hut アルゴリズムとツリーデータ構造

共有メモリシステムではポインタを用いたツリーへの再帰的アクセスにより Barnes-Hut は実装できるが, クラスタシステムでは, 制限なしにポインタを使ってグローバルツリーにアクセスする仕組みを提供するものはほとんどない. MPI では, グローバルツリーは実装できないため, 必要なデータ（部分木）を各フェーズで計算してノードに転送して処理を進めるため, 非常に複雑なコードになっている.

そこで今回, mSMS を利用して, OpenMP で実装した Barnes-Hut プログラムをほぼそのままの形で, mSMS に変換して, 性能を評価した[7]. 同様の試みは UPC でも行われているが, 性能は非常に悪く, しかも SMS と異なり, UPC には実行時ランタイム

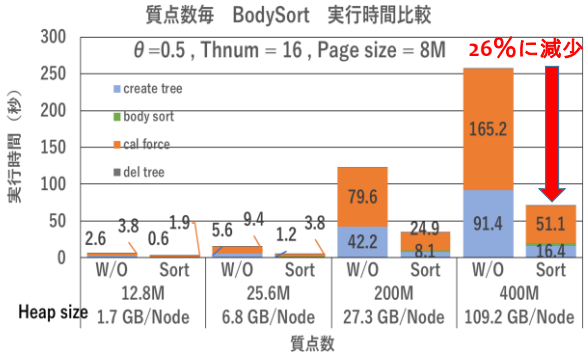



(a)　マルチノード向けグローバルツリーの実装

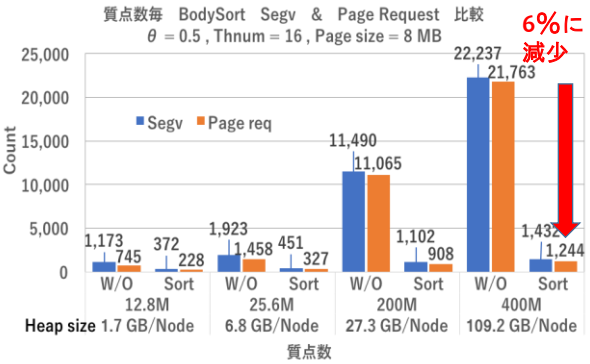



(b) グローバルツリーのメモリアクセス局所性を高める BodySort 手の導入（Tree Traverse による）

図 5 クラスタシステムにおける mSMS を利用したグローバルツリーデータの効率的アクセス実装




(a) 質点当たりの 1 ステップ実行時間とメモリ使用量




(b) 質点当たりの遠隔ノードデータのアクセス回数と実際のデータ転送回数（Pagereq）

図 6　グローバルツリーへのアクセス局所性を高める BodySort による性能向上
（質点数：1280 万-40 億,　4 ノード x 16 スレッド）

システムがないため，複数スレッドを制御する実行制御コードもユーザが応用プログラムに埋め込む必要もあり，コードが非常に複雑化している．

mSMS では，アクセス制限のない大規模なグローバルデータを，汎用の C ポインタで利用できるため，プログラムの改変はほとんどない．さらに，懸案であった遠隔データアクセスのためのオーバー

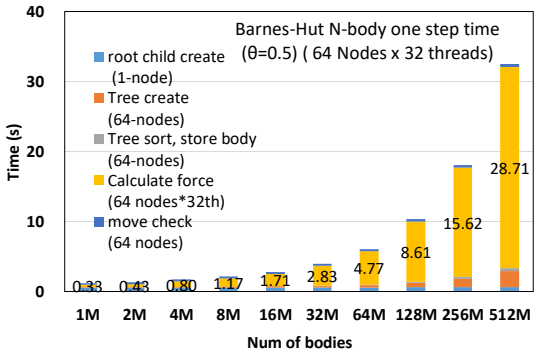



図 7　3 次元空間 N 体問題（質点数:1M－512M）
(Barnes-Hut，Oct-Tree，$\theta$ ＝0.5，64 x 32 スレッド)

ヘッドは，ツリーのアクセス局所性を高める BodySort（図 5）の導入により，非常に低減され，十分に実用に耐えることが明らかになった(図 6).

文献[7]では，４分木を利用し，最大 16 ノードを利用した２次元空間を対象とする Barnes-Hut の例を提示したが，３次元空間を対象とし 64 ノード x 32 スレッド（2048 スレッド）利用による実装と評価も行っている（図 7）．最大 819M 個の質点の 1 更新ステップあたり 45.5 秒，64M 個の質点では 5.8 秒で処理できる．

mSMS 利用により大規模なグローバルツリーをクラスタ上に構築し，汎用 C ポインタを利用し，共有メモリ向けのプログラムをほぼそのままで利用できることは，大きな利点である．しかも，利用できるグローバルデータは，ノード数×ノード当たりのメモリサイズと，大規模な質点の処理が可能となる．

### (4) 音響解析 FDTD(2,4) 計算の並列処理と性能評価

音響数値解析手法の一種である音響 FDTD(2,4) 法は、通常の FDTD 法と比較して袖領域のステンシル読み込み幅が増加するが、空間方向に高次精度の計算が可能となるため、効率的な大規模音響解析に向けたマルチノード並列化が求められている。

そこで、mSMS の実応用例として、FDTD(2,4)法のマルチノード実装を行い、TSUBAME 3.0 及び ITO において性能評価を行った(研究業績[5])。

まず、袖領域の通信において、各計算に必要なデータのプリフェッチ (SIGSEGV シグナルハンドラを介さず、指定範囲のデータを cache ページとして取得) を実現するプリロード API を利用した実装を行い、性能評価を行った。

図 8 は，プリロード API を利用した並列 FDTD(2,4) ソルバーのウィークスケーリング性能の計測結果を示している。１ノードあたり 1024^3 の格子数を持ち、ノード内並列化は OpenMP(各ノード 24 スレッド)を利用した。TSUBAME 3.0 及び ITO における１ノードの実行時間を基準とした実行時間比は 2-32 ノードにおいて、1.09-1.30(TSUBAME 3.0)、1.07-1.16(ITO)となった。TSUBAME 3.0 では主に同期時間の増加による性能低下が確認されるが、ITO においては、ほぼ理想的な結果が得られたといえる。

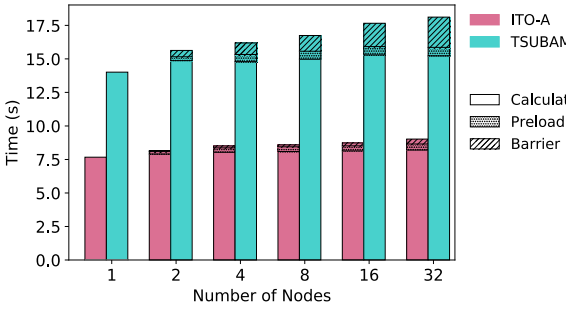



図 8　Preload 機能を用いた Weak scaling 性能 [5]

次に、実際的な計算で利用される複雑な境界条件を導入した際の、mSMS の有効性を調査した。実応用においては、Berenger の PML 吸収境界条件や境界埋め込み法の導入が一般的であるが、マルチノード並列化を実現するには、袖領域等の通信の実装コストが問題となる。上述の様に、mSMS ではプリロード API を用いることにより通信の最適化が可能であるが、ユーザの目的によっては、明示的な通信を意識せずに一定の性能が達成可能な、グローバルビューによるプログラミングが有効となる場合が予想される。そこで、FDTD(2,4)法へ PML 境界条件を導入し、図 9 の様な実応用に近い計算条件での性能評価を行った。この場合、袖領域の通信は全て SEGV ハンドラを介し

て行われ、プリロード API を用いた明示的な通信領域の設定は行っていない。

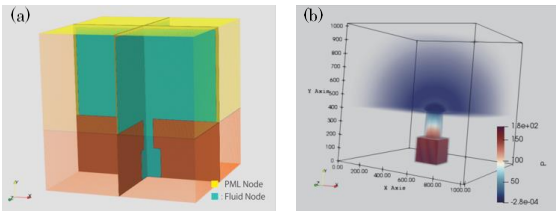



図 9 ヘルムホルツ共鳴器モデル（PML 層数: 20, 格子数: 1024^3). (b) 音圧分布（time step: 5000). [5]

図 10 に PML を導入した FDTD(2,4)法の各計算更新関数の ITO 上での平均実行時間を示す。通常の流体ノードの更新(Update U, P)は上述のプリロード API を利用した場合と比べ、幾分の性能の低下は見られるが、PML ノードの更新時間(Update PML U, P)は、ほぼ理想的な高速化が達成できている。

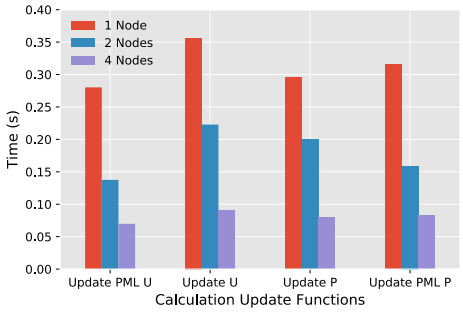



図 10　PML を導入した FDTD(2,4)法の各計算関数の平均実行時間 [5]

これらの結果より、FDTD(2,4)法の並列化において mSMS の利用により良好な性能が得られ、実応用的なステンシル計算においても mSMS の利用が有効である可能性が示された。ユーザの目的によって低い実装コストによる一定の性能の獲得から、明示的な通信の実装による最適化まで幅広く選択が可能であることは、数値解析を道具として用いる場合の研究サイクルを円滑に回す上で、非常に有用であると考えられる。

### (5) クラスタシステムにおけるトランザクショナルメモリ実行モデルの検討

現代の科学技術分野の進歩を支えている高性能な大規模並列計算基盤は，その計算速度の向上のために，ハードウェア・ソフトウェア両面において多大な労力が費やされてきた．ハードウェア面では，コア数やノード数を増大させることで計算基盤の性能を向上させてきた．そうしたハードウェアを活用するためのソフトウェアも改良が続けられている．例えば，コンパイラによる自動 SIMD 化や，OpenMP による自動並列化などが実用化されている．これら以外にも多くのソフトウェアが研究・開発され，マルチコア・プロセッサの性能を活用する基盤として利用されている．

一方で問題となっているのが，分散メモリ型と呼ばれるハードウェアの形態と，それを前提としたプログラミングモデルの生産性である．分散メモリ型の並列計算機は，それぞれ独立したメモリを備えた計算ノードが，多数並列に動作する．そしてこのようなアーキテクチャを活用するためのプログラミングモデルが，MPI などの通信インターフェースである．このインターフェースを使用したソフトウェア・システムでは，高性能なプログラムを実現するために，長い期間を費やして手動でチューニングする必要がある．本来アプリケーションプログラマは，アルゴリズム自体の開発に注力するべきであり，個々のアプリケーションに特化したチューニング作業からは解放されるべきであるが，このような開発形態は 20 数年間変わっていない．

そこで本研究では，PGAS をベースとした共有メモリ型並列計算基盤に対して TM（トランザクショナルメモリ）の機能を提供し，これをコヒーレンス制御に活用することにより，生産性と性能を両立する分散共有メモリ処理系の実現を目指す．マルチコアプロセッサ向けの TM をベースとして，分散用の機能を追加して拡張することでシステムを実現する．PGAS モデルのライブラリ実装である UPC++を使用して分散共有メモリインターフェースおよび，それに対する TM システムを設計・実装し，評価を行った．

赤黒木および K-Means の 2 つのマイクロベンチマークにより評価を行った。

赤黒木では，ノード数を増やすことによる速度向上は確認できなかった(図 11)．赤黒木の性質から，頻繁に要素を挿入すると，木のバランシングのために回転操作が発生する．すなわち，複数のメモリに

対して書き込みアクセスが発生するため，頻繁に競合が発生し，それに伴って性能が低下したと考えられる．しかし，軽微な変更だけで逐次プログラムから分散並列プログラム用のデータ構造を定義でき，それがノード数・スレッド数によらず正常に動作した．この「分散並列プログラムが容易に記述でき，それが正常に動作する」という性質は，様々なアプリケーションの特に開発初期段階において非常に重要である．

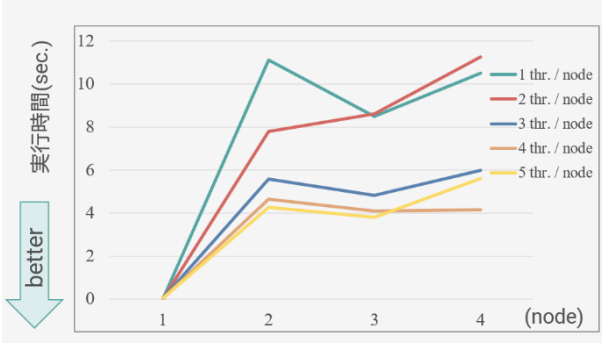



図 11　赤黒木による評価

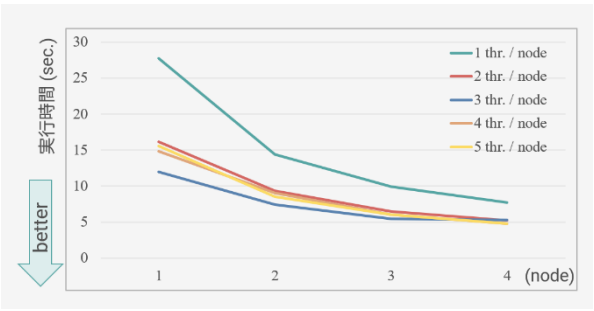



図 12 K-Means による評価

K-Means では，逐次実行した場合と比較していずれの場合も速度向上していることを確認した（図 12）．また，各ノードで実行するスレッド数によらず，ノード数の増大に伴って速度が向上しており，分散化することによる高速化を確認できた．

### (6) mSMS の省電力化方式の検討

mSMS の省電力化方式を検討するための基礎データとして，TSUBAME3.0 において CUDA アプリケーションを実行中の消費電力を測定した（研究業績[8]）．測定には，CUDA SDK にサンプルプログラムとして内包されているプログラムの中から 3 つ（matrixMul, matrixMulCUBLAS, simpleMultiCopy）を選んで使用した．matrixMul と matrixMulCUBLAS が計算インテンシブ，simpleMultiCopy がメモリインテンシブなプログラムである．これらのプログラムはシングル GPU 実行用に記述されたプログラムであることから，TSUBAME3.0 の全 2,160 台の GPU の中から無作為に 1 台の GPU を選択して上記プログラムを実行し，実行中の消費電力を計測した．

なお，TSUBAME3.0 の GPU はすべて同一ファミリーの GPU（Tesla P100）であるが，後述するように，各 GPU の消費電力にはばらつきが存在する．そこで計 100 台の GPU に対して上記の実験を行うことにより，TSUBAME3.0 における GPU の電力ばらつきの傾向を調査した．

GPU の消費電力は，ホスト側で nvidia-smi コマンドを実行することにより計測する．CUDA プログラムの実行中に上記コマンドを 5ms 毎に実行し，GPU 使用率が 100%の時の消費電力の平均値を当該プログラムの消費電力とする．

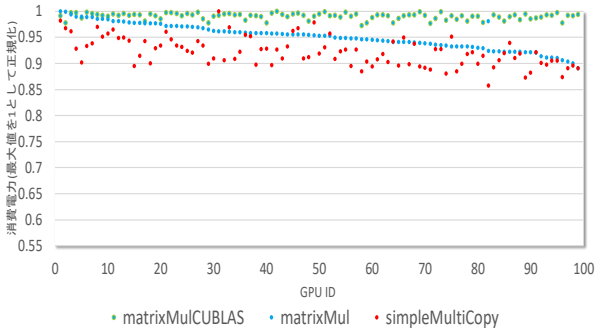



図 13　TSUBAME3.0 における GPU の電力ばらつき

評価結果を図 13 に示す．グラフの横軸は（実験に使用した）100 台の GPU の通し番号，縦軸は消費電力である．各 GPU は，matrixMul を実行した際の消費電力の降順に通し番号を付与した．また，消費電力の値は，各プログラムにおいて最大の消費電力を示した GPU の消費電力で正規化してある．なお，図の各点は，当該プログラムを当該 GPU で 50 回実行した時の消費電力の平均値である．

グラフより，GPU 電力のばらつきの大きさはプログラムによって異なることがわかる．特に，simpleMultiCopy では，GPU 間の消費電力の差が最大 14.2%に達することが確認できた．また，同グラフから，GPU 電力のばらつきの傾向はプログラムによって異なることがわかる．例えば，計算インテンシブな matrixMul では 0 番の GPU が，メモリインテン

シブな simpleMultiCopy では 31 番の GPU が最大電力を消費する．これは，GPU においてはコアとメモリが別チップであり，それぞれが独立に製造ばらつきの影響を受けたためと考えられる．

以上の結果より，mSMS の省電力化においては以下を考慮する必要があると考えられる．

- アプリケーションのメモリアクセス頻度に応じたプロセス配置：プロセスの実行に最適な GPU はアプリケーションのメモリアクセス頻度によって異なると考えられる．
- プロセス間のデータ転送量に応じたプロセス配置：データ転送量が多い程プロセスのメモリアクセス頻度は高くなることから，メモリアクセス頻度はアプリケーションだけでなくプロセスによっても異なると考えられる．

このため，GPU の消費電力を最小化の手法として，mSMS の機能を拡張し，アプリケーションやプロセスごとのメモリアクセス頻度に応じて，各プロセスの GPU を割り当てる仕組みが必要と考えている．

## 6. 今年度の進捗状況と今後の展望

### (1)-(3)の mSMS の API 実装と各種性能評価

当初目標としていた内容の 90%程度達成できたと考えている．特に，(4)の音響解析の応用分野への適用では，SMS 利用によりクラスタを利用して細かな境界条件の設定が非常にやりやすかったというフィードバックを受け，さらに多くの応用分野の方に利用して頂きたいと考えている．今年度，(5)(6)のテーマと十分な連携までには，いたらなかったが，(5)のクラスタにおける TM については，SMS を実装のサポートとして利用する方式が，今後考えられる．(6)の消費電力については，今年度，GPU 消費電力のばらつきを考慮したプロセス配置という一つの指針を得た．一方，一つの応用実行の観点からは，ユーザの求める速度性能・データの規模を考慮しつつ，利用する GPU 数，CPU コア数，ノード数をどう選択するかも消費電力低減になりうる．今後，消費電力設定に応じた柔軟なハードウエア選択を行うプログラム実行も一つの手段と考えられる．

### (4) 音響解析 FDTD(2,4)計算の並列処理と性能評価

申請時に設定していた目標である、mSMS を用いた音響 FDTD 法のマルチノード並列プログラムの開発及び、実応用計算で利用される境界条件を含めたソルバーの性能評価を予定通り実行し、FDTD(2,4)の並列化において mSMS が有効であることを示した。また、追加目標として、FDTD(2,4)への時空間ブロッキング手法(冗長計算有り)の導入を検討したが、袖領域の通信量の問題により、実装コストに見合う程の高速化は達成できていない。今後、冗長計算の無い時空間ブロッキング手法を導入することにより、さらなる効率化が図れると考えられる。その他、mSMS+ノード内複数 GPU 利用や非同期プリロード API を用いた計算と通信のオーバーラップなどが、今後の展望である。以上の理由から、目標の達成率を 8 割程度とする。

### (5) クラスタシステムにおけるトランザクショナルメモリ実行モデルの検討

PGAS をベースとした Intra-node 向け TM の設計および実装について，そのプロトタイプ実装まで完了し，複数のプログラムで正常に動作することを確認した。しかし，その性能がまだ十分でないことが評価により確認された。特に，他ノードのメモリ上にある変数にトランザクション内でアクセスする際に発生する通信回数がボトルネックとなっていることを確認した。以上のことから，今年度の進捗状況は 6 割程度と判断する。

上記の性能問題は，キャッシュ機構を導入することで大きく削減できる可能性が高い．本研究で設計した基盤にキャッシュ機構を検討中である．

マルチコアプロセッサを対象とした HTM では，キャッシュコヒーレンスプロトコルに対して，新たに Sticky 状態と呼ばれるステートを追加することで，TM の機能を実現している．しかし，ソフトウェアで実装する際に必ずしも踏襲すべき方法ではない．

キャッシュの実装方法の一つに，Write-Invalidate 方式がある．しかしこの方式はトランザクション中の書き込みが多い場合に invalidate が頻発する．STAMP ベンチマークスイートのプログラムに含まれるトランザクションの Read/Write 比を調べると，Write の比率がかなり低いことが確認できる．したが

って，トランザクションでの書き込みは高い頻度で発生しないと想定できる．これらの特徴をふまえたキャッシュ方式の検討・実装が今後の課題である．

### (6) mSMS の省電力化方式の検討

TSUBAME3.0 において GPU 電力の基礎データを収集し，mSMS の省電力化方式の検討を行ったという点においては，計画書の目標を達成できた．しかしながら，当初の計画とは異なり，実際に mSMS を使用するアプリケーションの消費電力データを元に mSMS の省電力化方式を検討するには至らなかった点はマイナスである．これは CUDA+mSMS，OpenACC+mSMS のアプリケーション開発が間に合わなかったためである．また，当初の計画では，CPU やメモリの消費電力も計測する予定であったが，計測に必要なツール（RAPL）の使用許可が下りなかったため，GPU 電力のみにフォーカスせざるを得なかった．以上から，今年度の進捗は 6 割程度である．

今後の課題としては，mSMS を使用するアプリケーションの消費電力評価，および，上記アプリケーションを実行中の CPU とメモリの消費電力の測定が挙げられる．

## 7. 研究業績一覧（発表予定も含む）

### (1) 学術論文 （査読あり）

なし

### (2) 会議プロシーディングス （査読あり）

[1] Y. Sakaguchi, H. Midorikawa: "The Programability and Performance of Global-View Programming API: SMint for Multi-node and Multi-core Processing", 2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, DOI: 10.1109/PACRIM47961.2019.8985126 pp.1-8, 2019.8

[2] H. Midorikawa, K. Kitagawa, Y. Sakaguchi: "mSMS : PGAS Runtime with Efficient Thread-based Communication for Global-view Programming", 2019 IEEE International Conference on Cluster Computing (Cluster2019), **DOI:** 10.1109/CLUSTER.2019.8891009, pp.1-2, 2019.9

[3] R. Tabata, T. Kobayashi, K. Takahashi, 'Numerical study of Synchronization Phenomena of an Air-Jet Instrument using Finite-Difference Lattice Boltzmann Method', International Symposium on Music Acoustics (ISMA), pp.296-303, Detmold Germany, Sep. 2019.9 http://pub.dega-akustik.de/ISMA2019/data/articles/000020.pdf

[4] Y.Inouchi, H.Yamaki, S.Miwa, T.Tsumura: "Functionally-Predefined Kernel: a Way to Reduce CNN computation", Proc. IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing (PacRim 2019), 6p, 2019.9, **received Best Paper Award for Computers Track (1/27 = 3.7%)**

### (3) 国際会議発表 （査読あり）

[5] Ryoya Tabata, Hiroko Midorikawa, Ki'nya Takahashi: "Performance Evaluation of Acoustic FDTD(2,4) Method Using Distributed Shared Memory System mSMS", 2020 International Conference on High Performance Computing in Asia-Pacific Region HPC Asia 2020 , pp.1-2 , Fukuoka, Japan, 2020.1
Abstract:http://sighpc.ipsj.or.jp/HPCAsia2020/hpcasia2020_poster_abstracts/poster_abstract_52.pdf
Poster:http://sighpc.ipsj.or.jp/HPCAsia2020/hpcasia2020_posters/poster_52.pdf

### (4) 国内会議発表 （査読なし、査読あり）

[6] 阪口裕悟，緑川博子:"グローバルビュープログラミングをサポートする PGAS 言語の記述性と性能の比較"，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.2019-HPC-170, No.41,pp.1-8, （July 2019）

[7] 緑川博子, 柴山悠:"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"， 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.2019-HPC-170, No.43,pp.1-8, (July 2019)

[8] 大八木哲哉，浅田 風太，三輪 忍，八巻 隼人，本多 弘樹："TSUBAME3.0 における製造ばらつきを考慮した GPU の電力モデリングの高速化"， 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.2019-HPC-172, No.24, pp.1-8, (Dec. 2019)

[9] 阪口裕悟，緑川博子:"マルチノード・マルチ GPU プログラミングを容易にする分散共有メモリシステム"， 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.2020-HPC-173,No.7,pp.1-8, (March 2020)

[10] 小林 龍之介, 二間瀬 悠希, 塩谷 亮太, 五島 正裕, 津邑 公暁: "メモリアクセス解析に基づくトランザクショナルメモリのポリシー動的切り替え手法", 情処研報 (HotSPA2019), Vol.2019-ARC-236, No.25, pp.1-11 (May. 2019)

[11] 川口 優樹, 津邑 公暁: "メモリアクセスパターンに基づく GPU スケジューリングポリシーの選択手法", 情処研報 (HotSPA2019), Vol.2019-ARC-236, No.2, pp.1-9 (May. 2019)

[12] （査読あり）小林 龍之介, 二間瀬 悠希, 多治見 知紀, 塩谷 亮太, 五島 正裕, 津邑 公暁, "トランザクショナルメモリにおけるメモリアクセスパターンを考慮したポリシー動的切り替え手法", The 3rd cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG 2019), Best Undergraduate Student Award 受賞

[13] 川口 優樹, 津邑 公暁: "静的解析に基づく GPU スケジューリングポリシーの選択手法", The 3rd cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG 2019), Outstanding Effort Award 受賞

[14] （査読あり）飯田 凌大, 塩谷 亮太, 五島 正裕, 津邑 公暁, "一貫性検証手法の動的切り替えによるソフトウェアトランザクショナルメモリの高速化", The 3rd cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG 2019), Outstanding M1 Student Award 受賞

[15] 浅井 優太, 山下 淳, 小林 龍之介, 二間瀬 悠希, 塩谷 亮太, 五島 正裕, 津邑 公暁, "ハードウェア機構の活用によるハイブリッドトランザクショナルメモリ高速化の検討", 情処研報 (ETNET2020), Vol.2020-ARC-240, No.34, pp.1-11 (Feb. 2020)

[16] 山下 淳, 浅井 優太, 小林 龍之介, 二間瀬 悠希, 塩谷 亮太, 五島 正裕, 津邑 公暁, "グラフ処理を題材とした最適なトランザクショナルメモリプログラミングの検討", 情処研報 (ETNET2020), Vol.2020-ARC-240, No.7, pp.1-10 (Feb. 2020), 電子情報通信学会 コンピュータシステム研究会 優秀若手発表賞 受賞

### (5) その他（特許，プレスリリース，著書等）

なし