TSUBAME 共同利用　令和3年度　学術利用　成果報告書

# 利用課題名　高性能計算向け分散メモリ・ストレージ統合システムの研究

英文：The Unified System of Distributed Memory and Storages for High Performance Computing

**利用課題責任者**
緑川 博子
**所属**
成蹊大学 理工学部
http://www.ci.seikei.ac.jp/midori/

**邦文抄録（300 字程度）**

本研究では，スパコンをはじめとするクラスタシステムにおいて，複数の計算ノードメモリを一つの共有アドレス空間にマップし大規模共有メモリとして提供し，各ノードコアから制限のない高速アクセスを実現するソフトウエア分散共有メモリ mSMS を構築している．これにより，単一計算ノードで広く用いられる OpenMP, OpenACC や C 言語による大規模配列への記述を，クラスタ上の大規模共有データに対して利用可能とした．本年度は，並列処理未経験である研究者に対し mSMS を提供し，音響流体解析応用 FDTD を実装して高速処理を実現し，複数ノードにまたがる大規模データに対する複雑な境界条件の設定などが非常に容易であるなどの成果を得た．

**英文抄録（100 words 程度）**

The mSMS, software-based distributed shared memory system, provides a very large shared memory, which can be accessed from any CPU core on any computing-node in a cluster system without accessible-area limitations. It realizes a parallel programming environment for shared large data on a cluster system which is easy to use and similar to the single-node parallel programming environment, such as C programming with OpenMP and/or OpenACC. The mSMS is used for large-scale acoustic energy simulations in this paper and shows its effectiveness not only in the processing performance but also in the program development.

*Keywords: software distributed shared memory; partitioned global address space; cluster computing;*

## 背景と目的

クラスタ利用による高性能計算では，現在も MPI が広く用いられ，分散メモリモデルによるプログラム開発の低生産性が未だ解決されたとは言えない．これを軽減するため，PGAS（Partitioned Global Address Space）モデルと総称される様々な言語・API が提案されてきたが[1]，大域データ配列や大域インデックスを利用可能とするものはあるものの，可能なアクセス範囲がノード隣接データ領域に限らていたり，範囲を超えた大域データアクセスには明示的な MPI のような局所的記述が必須であったり，定義できる大域データサイズに制限があり大規模計算には利用できないなど，多くの不自由さが存在する[2][3].

本研究は，スパコンをはじめとする大規模クラスタシステムにおいて，以下を実現する汎用かつ大規模なソフトウエア分散共有メモリシステムを構築し，従来の並列プログラム開発を容易にすることをめざしている[4].

- 計算ノードメモリの範囲を超える大域共有アドレス空間の実現と計算ノード全体に配置された大規模データへの制限のない高速アクセス
- 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のためのプログラミン環境，API の提供

本年度は，並列処理未経験である研究者に対し mSMS を提供し，音響流体解析応用を実装して高速処理を実現し，複数ノードにまたがる大規模データに対する複雑な境界条件の設定などが非常に容易であるなどの成果を得た.

## 概要

### ソフトウエア分散共有メモリシステム mSMS

mSMS (multithreaded Shared Memory System) は，クラスタ上の分散メモリを大規模共有メモリとして利用可能にするため，各ノードのローカルメモリアドレス空

間の他に，どのクラスタノードからもアクセスできる大規模な共有メモリアドレス空間を構築する．図1に示すように，各ノードが共有メモリとして提供するメモリは，「オーナーページエリア」として共有メモリアドレス空間上のそれぞれ違うアドレス領域として割り当てられる．プログラムが遠隔ノードの共有データアドレス領域にアクセスした場合には，ユーザ（プログラム）には見えない形で，SEGV シグナルで検知し，ページ単位でローカルノードに遠隔ページをフェッチして該当アドレス領域にキャッシュする． 並列プログラムにおける実行バリア同期などの際に，共有データの一貫性同期が行われ，各ノードにキャッシュされたページの更新情報は，そのページのホームノードに送られてページに反映される．

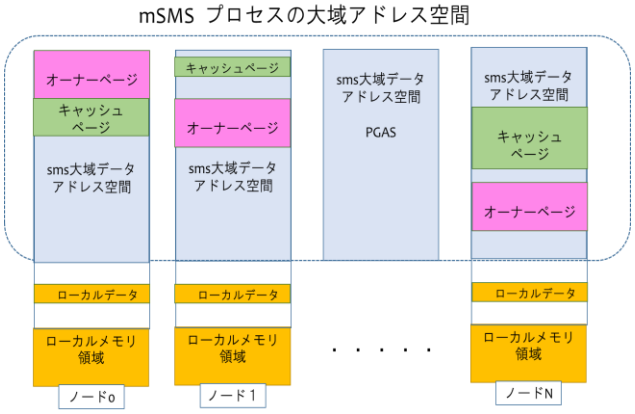


図 1　クラスタ上で大域アドレス空間を実現する
ソフトウエア分散メモリ mSMS

mSMS では，従来の単一ノードにおける C 言語による並列プログラム記述を，ほぼそのままの形でクラスタ上の大規模共有データに対して利用できるように設計されている．クラスタシステムでは，マルチノード，マルチ CPU コア，マルチ GPU コアなど，様々な並列方式が利用可能であるが，複数の API を自由に組み合わせて，mSMS では高性能計算に利用できる[4]．図2に示すように，単一計算ノードで広く用いられるマルチコア向け OpenMP や GPU 向け OpenACC などと同等のマルチノード向けの API である SMint も提供しており，逐次コードから容易に好みの API を組み合わせてインクリメンタルにプログラムを作成することもできる．

応用分野の計算特性や利用可能なシステム環境，ユーザの好みに応じ，(1)標準の C 言語と sms ライブラリ関数による記述，(2)大域共有配列宣言が可能な拡張 C 言語 MpC，(3)pragma 文 SMint によるプログラミングと，図3に示すように，複数のプログラミングインターフェースを提供している．

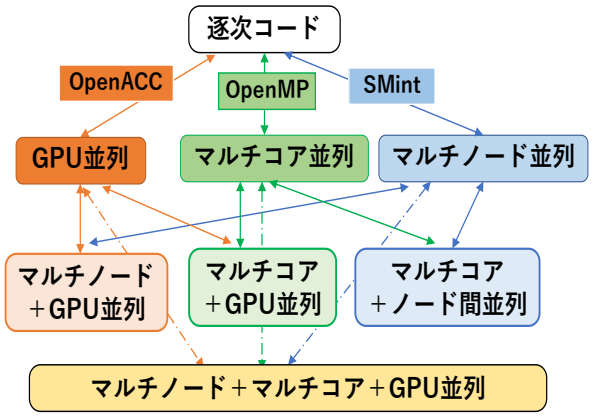


図 2　インクリメンタルプログラミング
逐次コードに，OpenACC，OpenMP，SMint の並列コメントを加えるだけで自由な組み合わせでプログラムが書ける

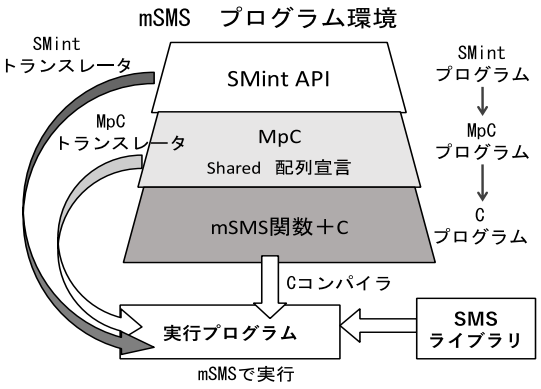


図 3　SMS における 3 つの API
C と SMS ライブラリ関数利用から，SMint API まで，好みのレベルの API を利用可能

### FDTD（2,4）法のマルチノード並列処理

並列処理の経験がない応用分野（音響解析）の研究者と共同で TSUBAME3.0 において mSMS を用いて，音響エネルギーシミュレーションの並列処理を行った．

大規模な音響解析を効率的に行うために，FDTD(2,4) 法等の高次 FDTD 法の有効性が検討されている[7]．マルチノード環境でのプログラミングの生産性を向上させるため，ページベース分散共有メモリシステム mSMS を利用し，図 4 のスケルトンコードに示す様に，複数の計算ノード全体にある大域データを対象としてグローバルビューでプログラミングを行った．mSMS を用いたマルチノード実装では，SIGSEGV シグナルハンドラによるリモートページフェッチ機能による手法の他に，図 5 のスケルト

ンコードに示す様なリモートデータプリロード API（sms_preload_array 関数）を用いて，計算利用前に事前に遠隔ノードからデータをプリフェッチする機能もある．これを隣接ノードとの袖領域データ交換に使用し，ノード内の並列化には OpenMP を利用することもできる．

**FDTD 並列処理の流れ**

図４では，sms_startup 関数で SMS が起動され各ノードに同一プログラムが並列に実行される．最後の sms_shutdown でマルチノード実行が終了となる．各ノードでは,大域データを sms_mapalloc で確保した後，全ノードで実行同期 sms_sync を行った後，NT 時間ステップの計算ループを実行する．計算ループ内では明示的な遠隔データのフェッチの記述はないが，SMS システムにより遠隔ノードより袖データが自動的にフェッチされて計算に用いられる．FDTD であるため，時間ループ内には２つの処理ループが含まれ,いずれも OpenMP を用いてマルチコアによる並列処理が行われる．各処理ループの終わりに sms_sync_drop 関数により，実行同期と各ノードにおけるキャッシュページ廃棄が行われる．

**遠隔データ一括プリフェッチ関数 sms_preload**

今回の FDTD や典型的なステンシル計算のように,予め次にアクセスする遠隔データの範囲がわかっている応用処理では，プログラムが遠隔データにアクセスしてから SEGV シグナルで検知し,ページ単位にデータをフェッチするのではなく，予め計算開始前（データアクセス前に）に次に使う連続データ領域をアドレスで指定して，指定領域を含む複数ページを一括でプリフェッチできる sms_preload 関数を利用し，さらに高速化できる．事前に遠隔データアクセス範囲が既知の場合,ページ毎の SEGV ハンドラが省き，高速化が図れる．

sms_preload_array 関数は，多次元大域配列の中の矩形の部分配列を指定して preload する関数で，関数内部で指定部分配列に含まれるページを自動的に計算し一括で preload する関数である．図４の２つの配列アップデートの for 文の前に，図５に示すようなデータのプリフェッチを加えると，高速化が図れる．

**遠隔データ再プリフェッチ関数 sms_overload**

ローカルノードに実データのないアドレス範囲のページ（図１のオーナーページ以外の領域）は，当初，アクセス不可(NO)に設定されているので，preload 関数で，遠隔データをアドレス空間に配置する際には，そのアドレス範囲のページをリードライト可能（RW）に設定してからデータを受け取る．その後，受け取ったノードで当該ページへの書き込みがあったかどうかの有無を記録するためキャッシュページは読み出し専用(RO)に設定し直す．本応用のように遠隔データの袖領域への書き込みがない場合には，バリアデータ同期（sms_barrier）の際に，キャッシュページは廃棄（すなわちアクセス不可 NO へ設定）される．しかし，ステンシル計算のように，時間ステップ毎に，繰り返し同じデータ領域を遠隔ノードから繰り返し読み込む場合には，キャッシュページは NO->RW->RO->NO のようにアクセス設定（mprotect システムコール）を繰り返すことになる．同じ領域に遠隔データを繰り返し読み込む場合（ステンシル計算のような処理の場合）には，最初の遠隔データプリフェッチで sms_preload を用いた後，実行同期(sms_sync)のみを行いキャッシュページをそのまま（RW）で保持し，2 回目以降の時間ステップでは，すでに存在するキャッシュページにそのまま上書きする sms_overload を用いることで高速化が図れる．実際のコードでは，時間ブロッキングアルゴリズムにより，複数時間ステップ（BT）の隣接境界データを一度に preload，overload することで，遠隔データのフェッチをさらに効率化している．

**結果および考察**

**mSMS 利用による FDTD (2,4) 処理の性能評価**

ここでは，前述のような高速化を測った場合（Preload API 利用，ステンシル計算部分のみ）の，マルチノード環境における並列性能評価[6]を示す．性能評価は，TSUBAME3.0（Intel Xeon E5-2680 v4, 14 core, 2.4GHz × 2 / node, Intel Omni-Path 100Gb/s × 4, Intel MPI 2018.1.163）と 九州大学の ITO Subsystem A（Intel Xeon Gold 6154, 18 core, 3.0GHz × 2 / node, InfiniBand EDR 4 ×

100Gb/s, MVAPICH2-X 2.2)上で実行した. OpenMP スレッド数は両アーキテクチャともに 24 とする. データメッシュサイズは倍精度で, z 方向に分割して並列処理する. 各ノードの z サイズは 1024 である.

図 6 は, 1 ノードから 32 ノードまでの並列化された FDTD(2,4) ソルバーの Weak スケーリングの実行時間を示している. 凡例の'Calculation','Preload' 及び'Barrier' はそれぞれ, 計算時間, Preload API によるデータ交換時間, バリア同期時間を示す. 図 6 のノード 1 の値はベースライン, すなわち, 各アーキテクチャで mSMS を使用しなかった場合のシングルノードの実行時間を示している. ITO-A および TSUBAME3.0 を用いて求めたベースライン実行時間に対する 2-32 ノードの実行時間の比率は, それぞれ 1.07～1.16 および 1.09～1.30 である. 図に示す様に, mSMS と OpenMP で並列化した FDTD(2,4) ソルバーはほぼ理想的なスケーリング結果を示している.

```
#include <sms.h>
int main(int argc, char const *argv[]) {
  double (*matrix)[y_size][x_size];
  matrix=sms_mapalloc(dim,div,sizeof(double),0,sms_nprocs);
  sms_startup(&argc, &argv); // start mSMS system
  bz = NZ/sms_nprocs; // block size for one node
  sz = sms_rank * bz; ez = (sms_rank + 1) * bz;// z-division
  sms_sync(); // execution-synch.
  for ( int time_step = 0; time_step < NT; i++) { // main loop of fdtd(2,4) method
  #pragma omp parallel for
  for (k = sz; k < ez; k++) for (j = sy; j < ey; j++) for (i = sx; i < ex; i++) {
    qx[k][j][i]-=CQ*(K1*(p[k][j][i+1]-p[k][j][i])-K2*(p[k][j][i+2]-p[k][j][i-1]));
    qy[k][j][i]-=CQ*(K1*(p[k][j+1][i]-p[k][j][i])-K2*(p[k][j+2][i]-p[k][j-1][i]));
    qz[k][j][i]-=CQ*(K1*(p[k+1][j][i]-p[k][j][i])-K2*(p[k+2][j][i]-p[k-1][j][i]));
  }
  sms_sync_drop(); // execution-synch. & discard cache pages
  #pragma omp parallel for
  for (k = sz; k < ez; k++) for (j = sy; j < ey; j++) for (i = sx; i < ex; i++) {
  p[k][j][i]-=CP*(K1*(qx[k][j][i]-qx[k][j][i-1])-K2*(qx[k][j][i+1]-qx[k][j][i-2]))
             +CP*(K1*(qy[k][j][i]-qy[k][j-1][i])-K2*(qy[k][j+1][i]-qy[k][j-2][i]))
             +CP*(K1*(qz[k][j][i]-qz[k-1][j][i])-K2*(qz[k+1][j][i]-qz[k-2][j][i]));
  }
  sms_sync_drop();
  }
  sms_shutdown();// finalize mSMS system
}
```

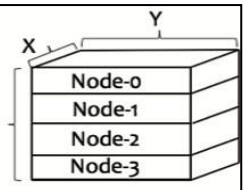


図 4　mSMS における FDTD (2, 4) 並列スケルトンコード

```
size_t g_stld[3]= { NZ,NY,NX}; // global data size
size_t e_size[3]= { load_ez - ez,NY,NX}; // prefetch data size
if( time_step == 0)
  // prefetch e_size data from &matrix[ez][0][0] pointer to cache page
  sms_preload_array(&matrix[ez][0][0], g_stld, e_size, 3, sizeof(double), 1);
else{
  // overwrite cache page
  sms_overload_array(&matrix[ez][0][0], g_stld, e_size, 3, sizeof(double),1);
}
```

図 5　データプリフェッチ利用時のスケルトンコード

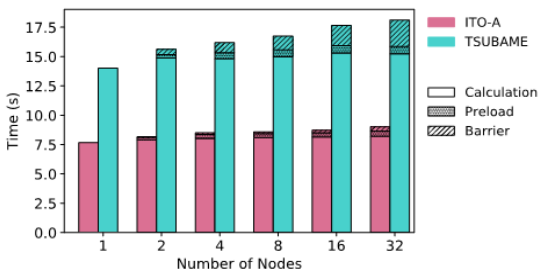


図 6 FDTD(2,4)法の SMS によるマルチノード並列処理
weak scaling 性能　sms_preload 利用
1024x1024x1024 格子／ノード, 24 スレッド／ノード

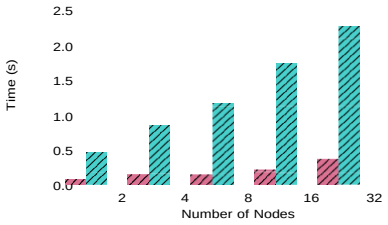


図 7 バリア時間　エラーバーは標準偏差

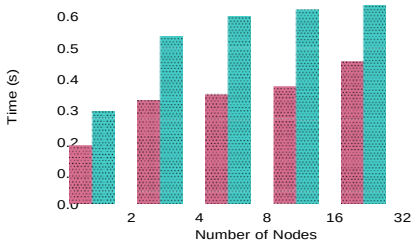


図 8 preload 時間　エラーバーは標準偏差

また，図 7，図 8 はそれぞれ，図 6 の’Barrier’ 及び’Preload’ の平均実行時間を示す（エラーバーは標準偏差を示す）．図の様に，TSUBAME3.0 における実験では より新しいシステムである ITO-A と比較して，比較的長い傾向があり，それぞれノード数の増加に伴い向上する．また，Preload API によるデータ交換時間はノード数の増加に伴い飽和する傾向にあり，ほぼ理想的な結果を示している．

さらに実応用として PML 吸収境界条件を導入し，ヘルムホルツ共鳴器モデルの解析を行った結果を図 9 に示す[6]．このようなシミュレーションでは通常，複雑な境界条件を記述する．これまでの分散データ型の MPI などでは，記述が複雑でミスを生みやすくコストが大きかった．今回，一つの共有大域データに対し境界条件などが容易に記述できる点は，mSMS の大きな利点として指摘された．

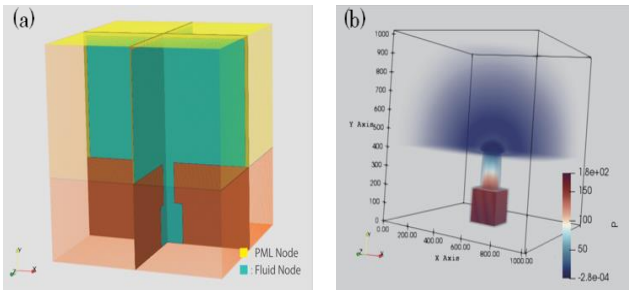


図 9　FDTD(2,4)法へ Berenger PML 吸収境界条件(実応用に近い条件)を導入した，ヘルムホルツ共鳴器モデルの解析結果

## まとめ，今後の課題

2020 年度から，並列処理プログラム開発経験のない音響解析研究者と共同で TSUBAME3.0 において mSMS を利用しソフトウエアを開発し，2021 年度，その成果が米国音響学会の論文誌に採択された[5]．

mSMS では，ユーザの経験やシステムの環境に応じた複数のプログラミング API を提供しており，既存の CPU コア，GPU コア並列 API（OpenMP, OpenACC)とも組み合わせることが可能である．これらを組み合わせ，スパコンクラスタにおいて高性能処理が比較的容易に可能であることを示した．また，mSMS の提供する大域共有配列に対し，データの所在(計算ノードの内外)を意識せずに，複雑な記述が可能である点は，並列処理を専門としない応用分野のユーザにとって，大きな利点であることが明らかになった．

今後，応用処理プログラムを作成するユーザが容易に mSMS を利用したプログラム開発ができるように，マニュアルやサンプルプログラムなどを整備，提供することが重要と考える．

参考文献

[1] M.D. Wael, et al.: “Partitioned Global Address Space Languages”, Journal of ACM Computing Surveys (CSUR), Vol.47, No.62, 2015

[2] 阪口裕梧，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”，情処学会, HPC 研究会報告，Vol 2019-HPC-170, No 41，pp. 1-10，2019.7

[3] Y.Sakaguchi, H.Midorikawa : “Programmability and Performance of New Global-View Programming API for Multi-Node and Multi-Core Processing”, IEEE Pacific Rim Conference on Communications Computers and Signal Processing, 2019.8

[4] 緑川博子，阪口裕悟:"分散共有メモリシステム mSMS におけるマルチノード・マルチ CPU・マルチ GPU プログラミング"，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.2021-HPC-178,No.24,pp.1-10, 2021.3

[5] Ryoya Tabata, Rei Mastuda, Toshiaki Koiwaya, Sho iwagami, Hiroko Midorikawa, Taizo Kobayashi, Kinya Takahashi: "Three-dimensional numerical analysis of acoustic energy absorption and generation in an air-jet instrument based on Howe's energy corollary", The Journal of the Acoustical Society of America, Vol.149, No.6, pp.4000-4012, 2021.6

[6] 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, Jan. 15-17, 2020. 1

[7] Y. Sendo, H.Kudo, T. Kashiwa, and T. Ohtani: The fdtd(2,4) method for highly accu-rate acoustic analysis in three-dimensional space. Electronics and Communications in Japan (Part III: Fundamental Electronic Science), 86:30 – 37, 11 2003.