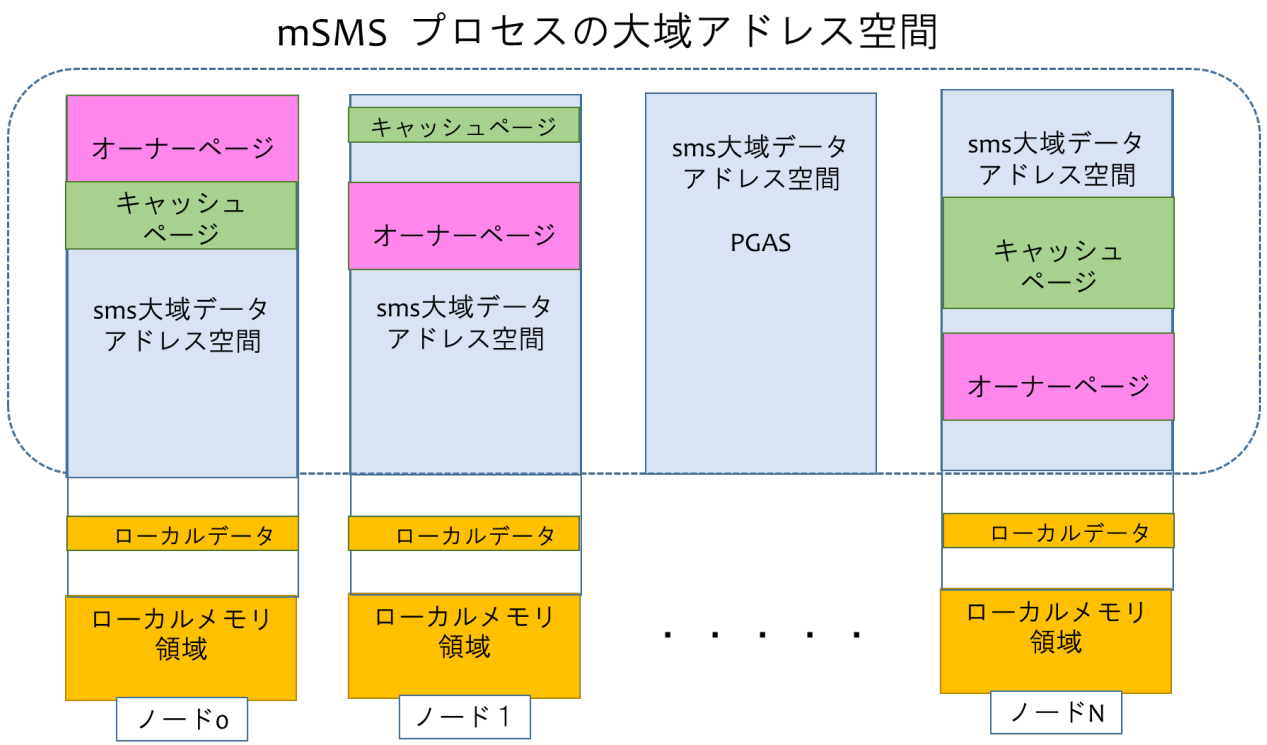
高性能，高生産性を実現する大規模メモリ・並列処理システムソフトウエアの研究

学際大規模情報基盤共同利用・共同研究拠点公募型共同研究
平成31年度採択課題　jh190039-ISH

緑川博子　（成蹊大学　理工学部）

研究目的

クラスタシステム上に大規模大域アドレス空間をに実現するSMSランタイムシステム



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
(multithreaded Shared Memory System)

研究目的

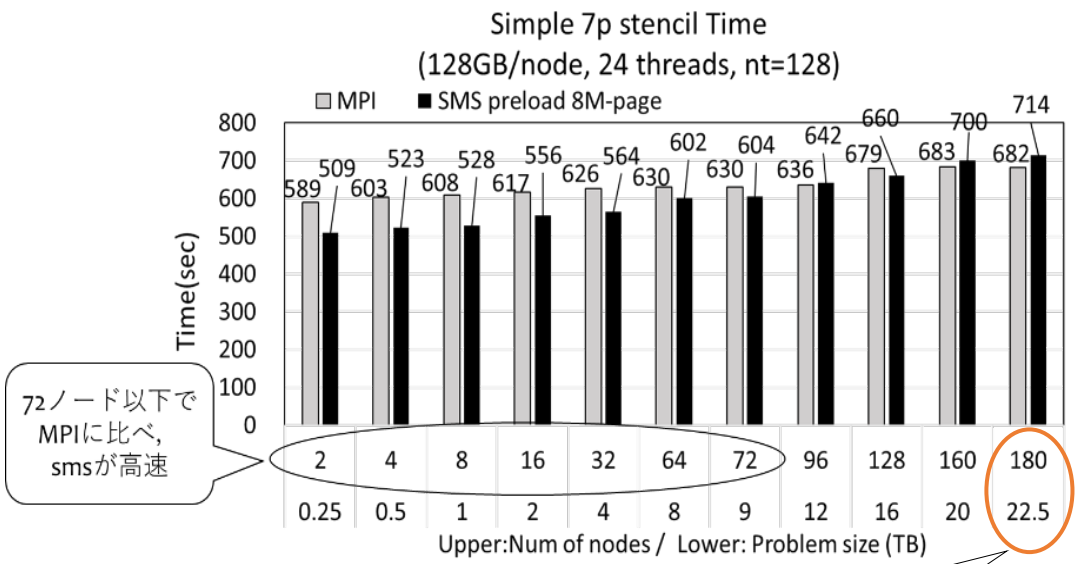
大規模クラスタシステムにおいて，以下の４つを実現するシステムソフトウエアの構築

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.

クラスタにおいて共有メモリモデルによる楽な記述と高性能を両立

3次元データ7点ステンシル計算 MPIプログラムと同等以上の性能を獲得



Tsubame3.0 （東工大）
128GBデータ/node，2-180ノード
最大データサイズ　22.5TB/180ノード
128の時間ステップ　実行時間

7点ステンシル計算におけるMPIとmSMSにおける実行時間の比較

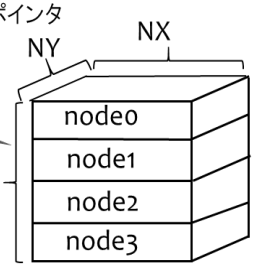
大域データshared宣言により，通常Cプログラムと同等な記述で．マルチコア並列（OpenMP）とマルチノード並列（mSMS）を実現

shared指定子により，クラスタシステム上に．大域データ配列を定義

```
shared double A[NZ][NY][NX]::[NPROCS][1][1](0, NPROCS); // 大域データ配列
shared double B[NZ][NY][NX]::[NPROCS][1][1](0, NPROCS); // Z分割分散マップ
main()
{   double (*src)[NY][NX]; double (*dst)[NY][NX]; double (*tmp)[NY][NX]; // 3次元arrayを指すポインタ
    mpc_init(&argc,&argv);   // sms_startup() に変換
    nx = NX, ny = NY, nz = NZ;  // 配列サイズ
    bx = nx, by = ny; bz = nz / NPROCS;  // ブロックサイズ，Z次元分割
    // 各ノードの計算領域を始点，終点設定
    sx = 0; ex = bx;   sy = 0; ey = by;
    sz = MYPID * bz;  ez = (MYPID + 1) * bz;
        :
    for (z = sz; z < ez; z++)  for (y = sy; y < ey; y++)  for (x = sx; x < ex; x++) {A[z][y][x] = ?; B[z][y][x] = ?; } // 配列初期化
    mpc_barrier();  // データー貫性(ここでは通信トラフィックなし）と実行同期，sms_barrier()に変換
    src = A;  dst = B;
    for (t = 0; t < nt; t++) {   // 時間ステップ
    #pragma omp parallel for
      for (z = sz; z < ez; z++)  for (y = sy; y < ey; y++)  for (x = sx; x < ex; x++) { // 7点ステンシル計算
        dst[z][y][x]=0.4*src[z][y][x]+
                0.1*(src[z-1][y][x]+src[z+1][y][x]+src[z][y-1][x]+src[z][y+1][x]+src[z][y][x-1]+src[z][y][x+1]);
      }
      sms_sync_drop();   //   実行同期，各ノードキャッシュページ廃棄
      tmp = dst;   dst = src;   src = tmp;      // srcとdstの交換
    }
    mpc_exit();   // sms_shutdown() に変換
}
```

7点単純ステンシル計算スケルトン
（MpCプログラム→ SMSライブラリ関数へ変換）

大域データ配列の分散マップ
Z分割4nodeへのマップ例



MYPID：MPI rankと同等
NPROCS：MPI 総プロセス数

実際は，omp parallel を利用して，各種変数を定義，6重ループによる空間ブロック化を用いている

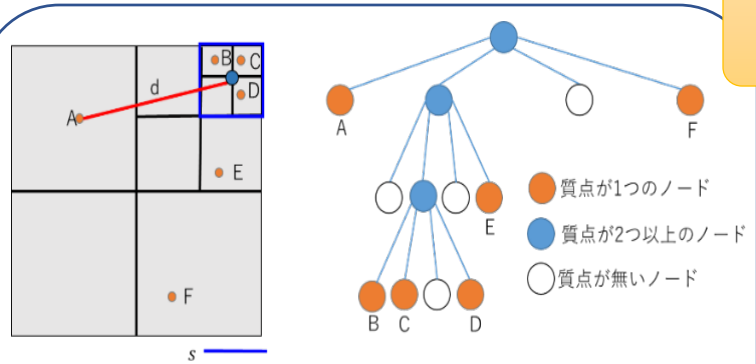
SMSを利用したクラスタ向け７点ステンシル計算プログラム
MpC：Cに最低限の拡張（shared 分散マップ配列）＋　OpenMP

クラスタシステム上のグローバルツリーによるBarnes-Hutアルゴリズム

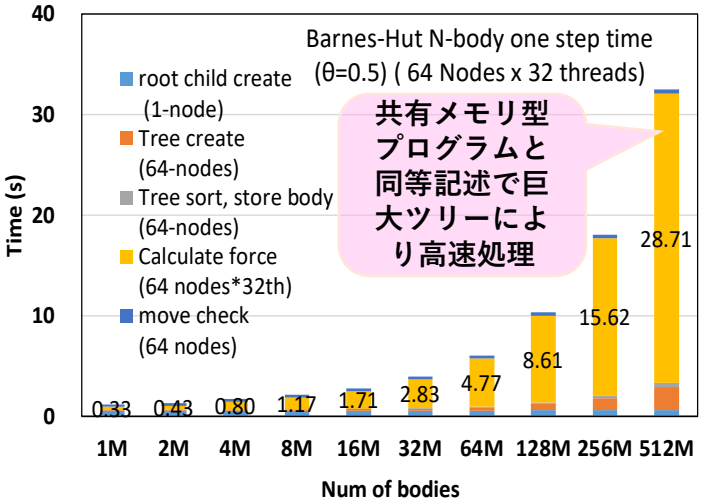
事前のデータ配置の難しい，不規則データ構造，非同期アクセス応用に対しても，mSMSの有効性を確認

グローバルツリーへのアクセス局所性を高めるBodySortにより大幅な性能向上

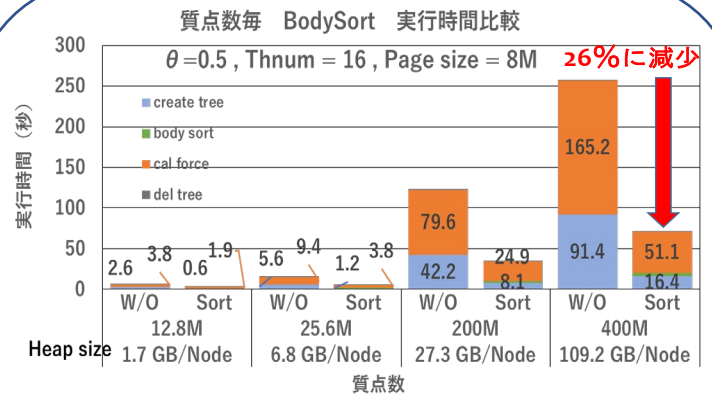
クラスタシステムにおけるmSMSを利用したグローバルツリーデータの効率的アクセス実装



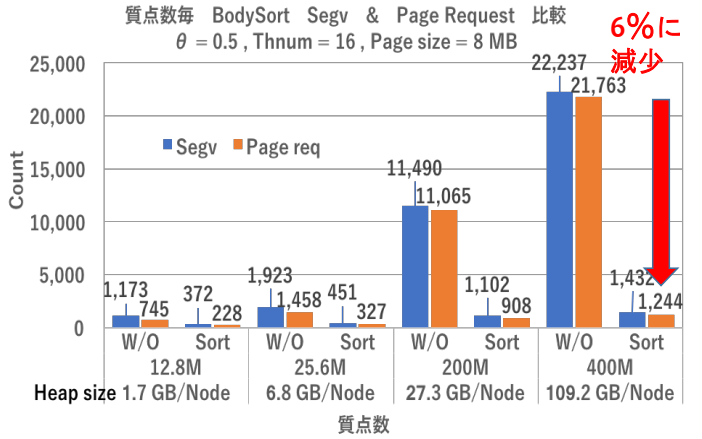
Barnes-Hutアルゴリズムとツリーデータ構造
パラメタ θ　θ ≧ s/d　の時，重心で計算



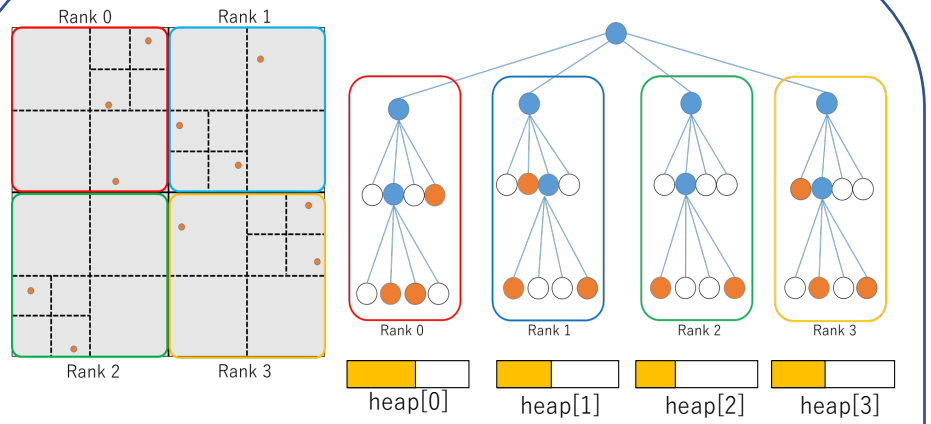
3次元空間N体問題（質点数:1M－512M）
(Barnes-Hut，Oct-Tree，θ＝0.5，64ｘ32スレッド)



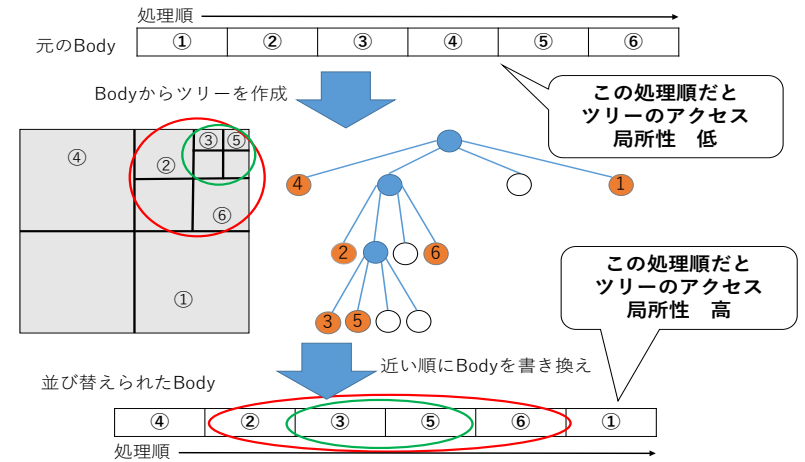
質点当たりの1ステップ実行時間とメモリ使用量



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）



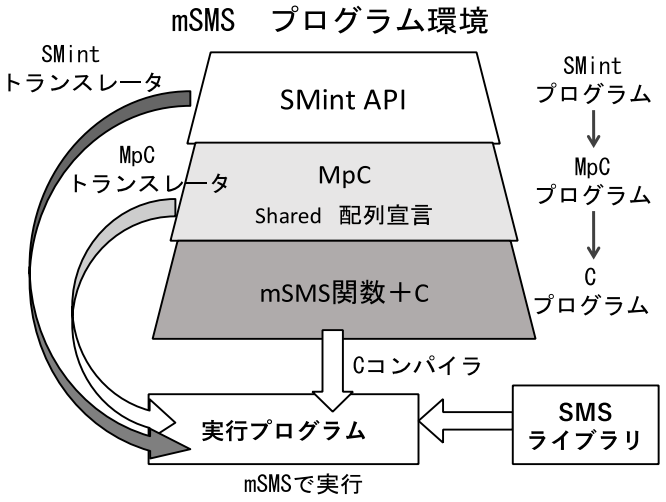
(a)　クラスタシステム向けグローバルツリーの実装



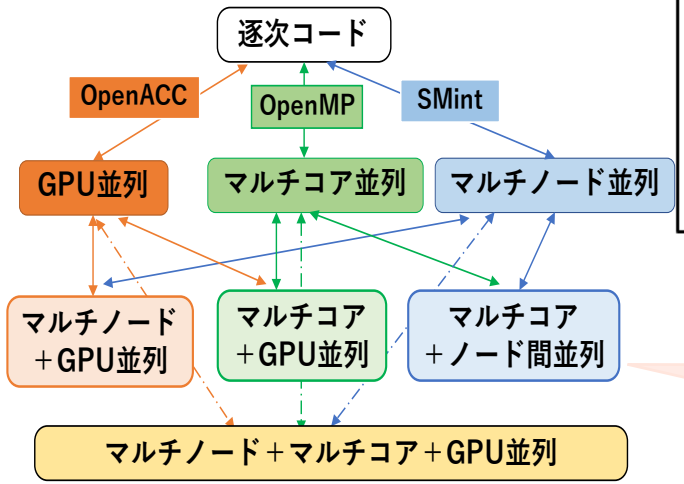
(b)　グローバルツリーのメモリアクセス局所性を高めるBodySort手法の導入（Tree Traverseによる）

グローバルビューモデルに基づくSMS並列プログラミング環境
３つのAPI（C＋SMSライブラリ関数，MpC，SMint ）

SMSにおける３つのAPI



インクリメンタルプログラミング



ディレクティブベースAPI　SMint
（OpenMP, OpenACC，pthread，CUDA併用可）

ループ並列処理などの典型的記述向け
インクリメンタルプログラミングにより，
逐次コードからの容易な拡張が可能

```
#include <smint.h>          // #pragma SMint 利用プログラム
#define N …
#pragma SMint shared ::[1](0,1) ;     // node0にマップ
double vec1[N] ;
#pragma SMint shared ::[](1,1) ;      // node1にマップ ,省略記述
double vec2[N] ;
#pragma SMint shared ::[NPROCS][](0,NPROCS) ; //全ノードに分散マップ
double array[N][N] ;

int main(int argc, char *argv[] )
{   int size, st, ed;              // 各ノードの担当領域

  #pragma SMint parallel for  // 全ノードで並列実行
  #pragma omp parallel for    //各ノードでマルチスレッド実行
   for( i=0; i<N; i++) {
     for(k=0; k<N; k++)
          vec2[i]= array[i][k] * vec1[k];     // 行列ベクトル積
   }
   // バリア同期は自動挿入
}
```

SMintによる行列ベクトル積プログラム

逐次コードにpragma文を加えるだけで
マルチノード・マルコア並列実行可能

C ＋ SMSライブラリ関数
（OpenMP, OpenACC，pthread，CUDA併用可）

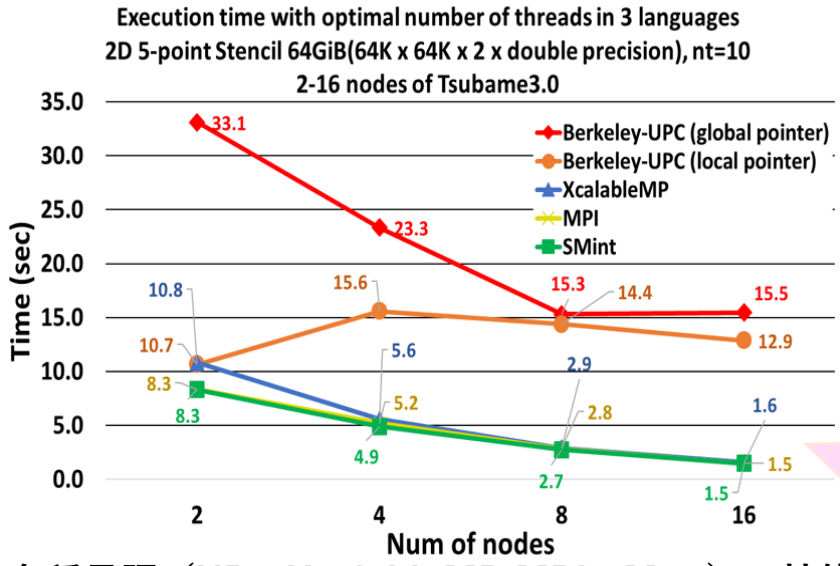
```
#include <sms.h>                  //  SMSライブラリ関数利用によるC プログラム
#define N …
int main(int argc, char *argv[] )
{   int size, st, ed;                  // 各ノードの担当領域
    double *vec1, *vec2;  // 1次元配列 vec1[N], vec2[N] のためのポインタ
    double (*array) [N];  // 2次元配列 array[N][N] のためのポインタ
    int dim[3]={N, N,-1}, div[3]={1, 1,-1}; // dim: 配列サイズ, div:分散マップ分割数

    sms_startup(&argc, &argv);

    vec1 = (double*)sms_alloc(sizeof(double), N, 0); // vec1[N] node0に割り付け
    vec2 = (double*)sms_alloc(sizeof(double), N, 1); // vec2[N] node1に割り付け
    div[0]=sms_nprocs;          //   arrayをバンド分割，全ノードに分散マッピング
    array= (double(*)[N]) sms_mapalloc(dim, div, sizeof(double), 0, sms_nprocs );

    size=N/sms_nprocs;
    st=size * sms_rank; ed=size * (sms_rank+1); //各ノード担当領域
  #pragma omp parallel for          //   各ノードでは，マルチスレッド実行
    for( i=st; i<ed; i++) {               // 全ノードで for(i=0; i<N; i++) を並列実行
       for(k=0; k<N; k++)    vec2[i]= array[i][k] * vec1[k];     // 行列ベクトル積
    }
    sms_barrier();
    sms_shutdown();
}
```

SMSライブラリ関数による行列ベクトル積

他のPGAS言語との記述性，性能の比較

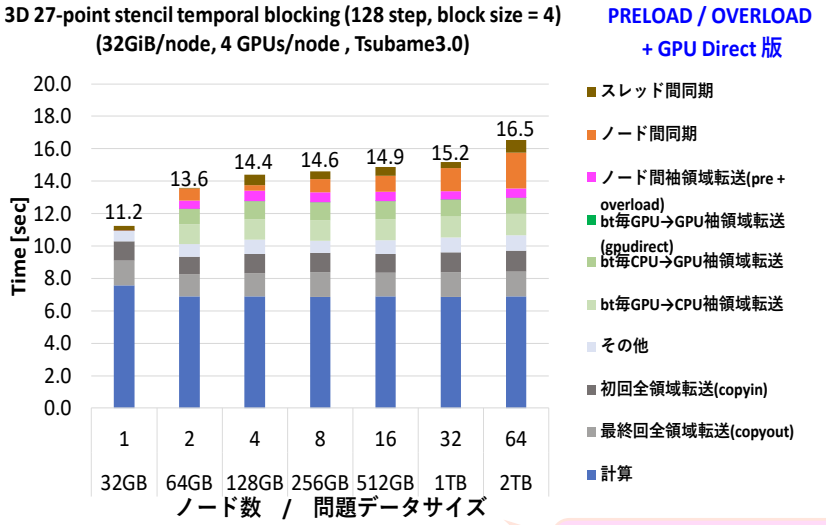
各種言語	プログラムの記述性		性能	特徴
	行数	逐次に対する行数増加率	MPIに対する相対実行時間＊	扱えるグローバルデータサイズ、アクセス可能領域、記述の制限など。
UPC (Global view model)	29	1.07	4.0 ～ 10.4	グローバルデータサイズに制限有 小さい
UPC (Local view model)	57	2.11	1.3 ～ 8.9	MPIと同様のノードを意識したローカル記述
UPC (動的確保 upc_alloc)	71	2.63	9.4 ～ 77.0	コードが複雑
XcalableMP	40	1.48	1.0 ～ 1.3	グローバルデータのアクセス可能領域に制限有
SMint	31	1.15	0.9 - 1.0	グローバルデータのサイズ、アクセス制限なし
MPI	73	2.70	1	
Sequential (OpenMP)	27	1.00		

＊ノード、スレッド数により異なる



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）

SMSによるマルチノードマルチGPU並列処理



Weak Scaling性能

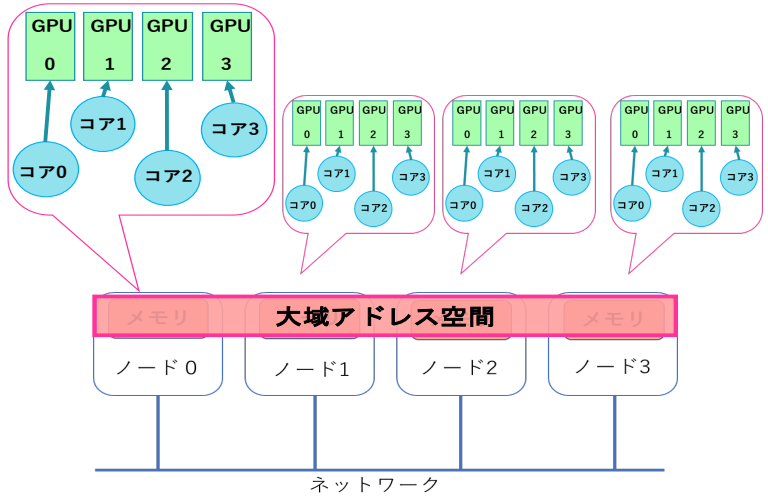
CPUコア利用時の約10倍高速

Tsubame3.0　27点ステンシル実行時間
ノード内 GPU-Direct（4GPU）利用による
データ交換

SMSは，UPC ,XMPに比べ，高速MPIと同等以上

共有メモリモデルでマルチGPU処理が容易に記述可能

各ノードプロセス内の各CPUコアがGPUを使う



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

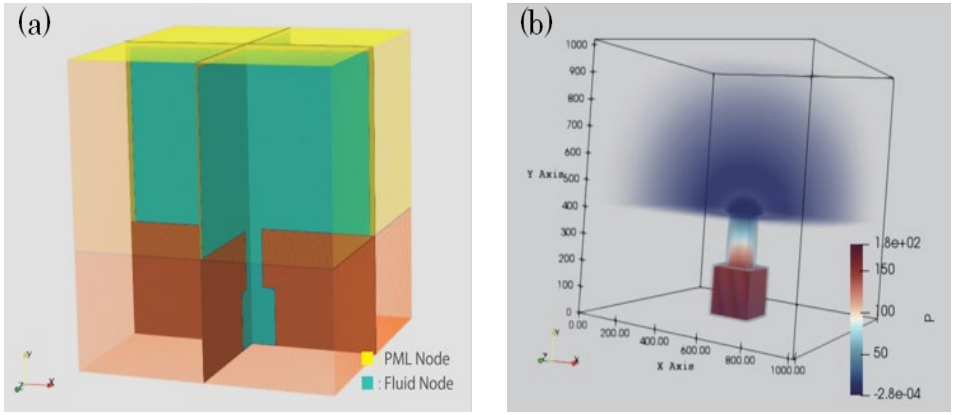
```
#include <stdio.h>
#include <omp.h>
#include <openacc.h>
#include <sms.h>
…                                   ヘッダインクルード

int main(int argc, char **argv)
{                                   マルチノード実行
  sms_startup(&argc, &argv);
  int numgpus = acc_get_num_devices( acc_device_nvidia );
  …                                 マルチコア実行
#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
    {                               GPU実行
      …
    }
    #pragma acc exit data copyout(…)
  }
  sms_shutdown();
}
```

（b） SMS関数 + OpenMP + OpenACC + (CUDA)による記述

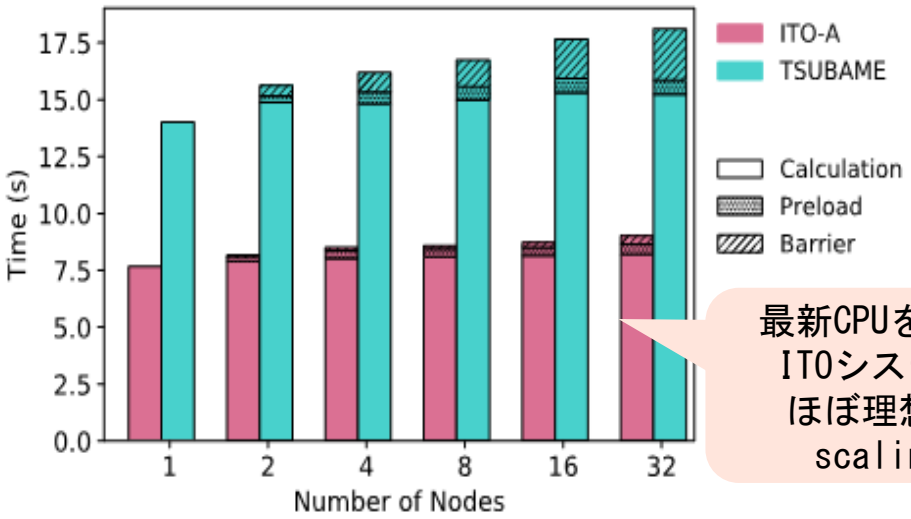
マルチノードマルチGPUシステムの利用例

SMSによる音響解析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る.
SMSを利用し，時空間ブロッキング等の導入により音響ソルバーに適した高効率実装手法を開発．また、楽器や音響機器等の実問題への応用向けの境界条件等も含めた処理が，容易に記述でき，ほぼ理想的な性能を得られた.



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
(PML層数：20，格子数：1024^3)



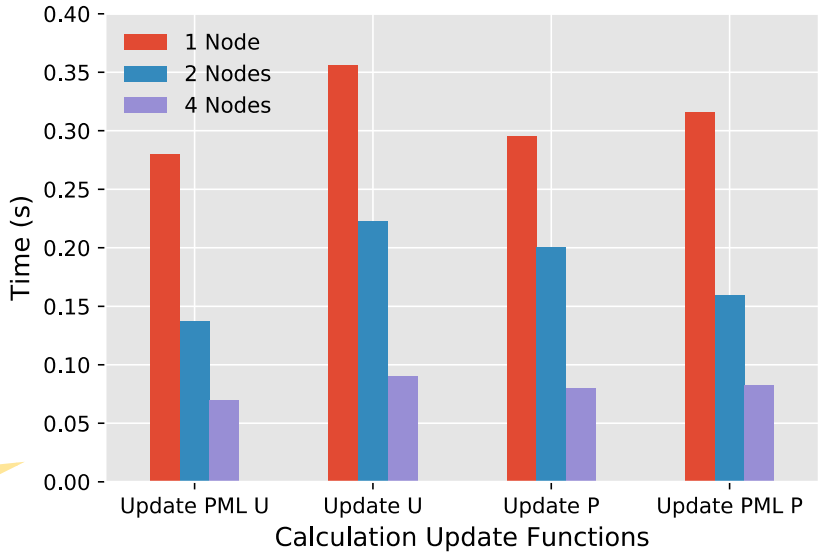
FDTD(2,4)法のSMSによるマルチノード並列処理，Weak scaling性能，SMS preload利用
1024x 1024x 1024格子/ノード，24スレッド/ノード

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SMSの有効性を確認

最新CPUを搭載するITOシステムではほぼ理想的weak scaling性能

SMSの提供する大域的配列に対し，データの所在（ノード内外の区別）を意識せずに複雑な記述が可能

PMLノードの更新時間(Update PML U，P)は、ほぼ理想的な高速化を達成

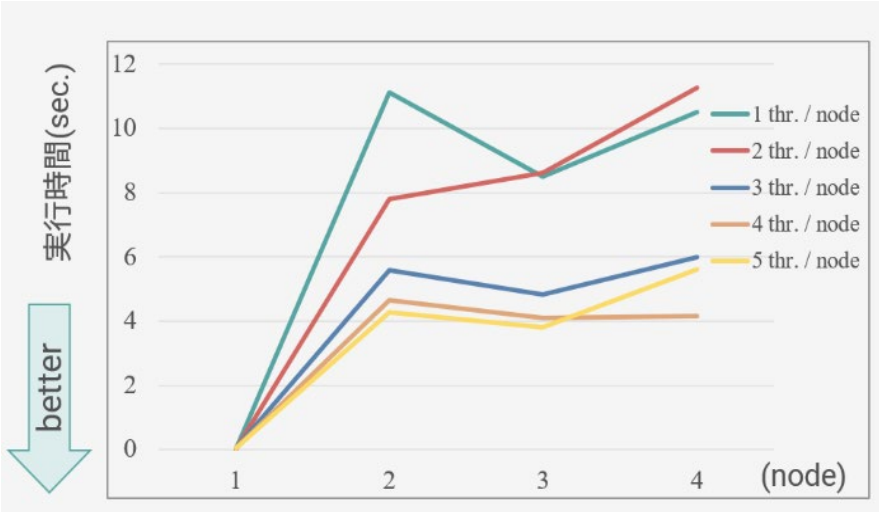


PMLを導入したFDTD(2,4)法の各計算関数の平均実行時間（ITO)，SMS segv利用

担当者：田畑諒也(九工大)，緑川博子（成蹊大)，高橋公也（九工大)

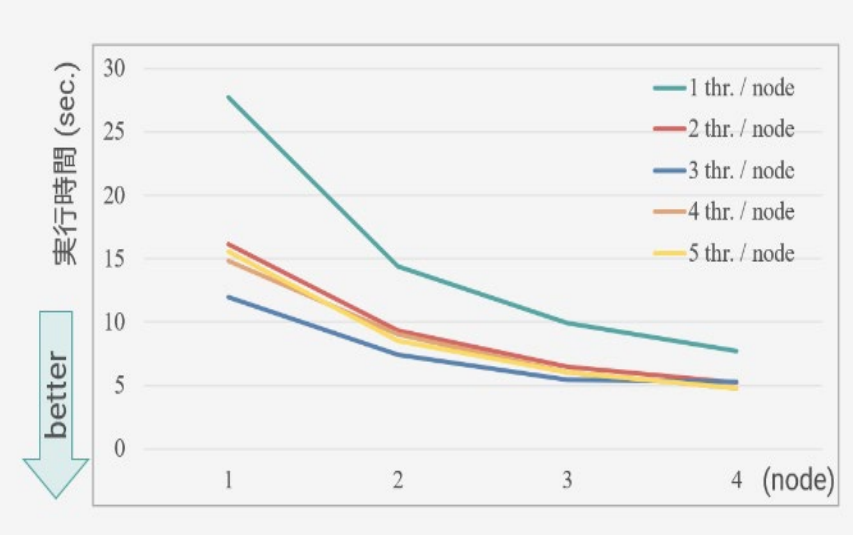
クラスタ向けTransactional Memory APIの検討

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．今年度は，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。



赤黒木による評価

逐次プログラムから の容易な変更で記述可能，
複数のメモリへの書き込み競合により性能が低下



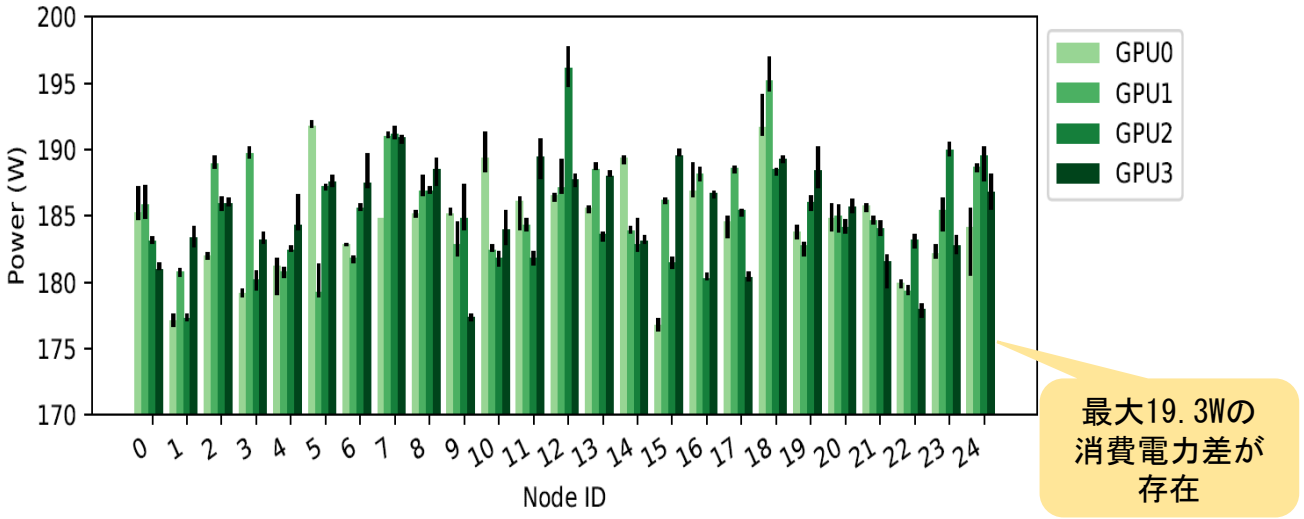
K-Meansによる評価

ノード数の増大に伴って速度が向上

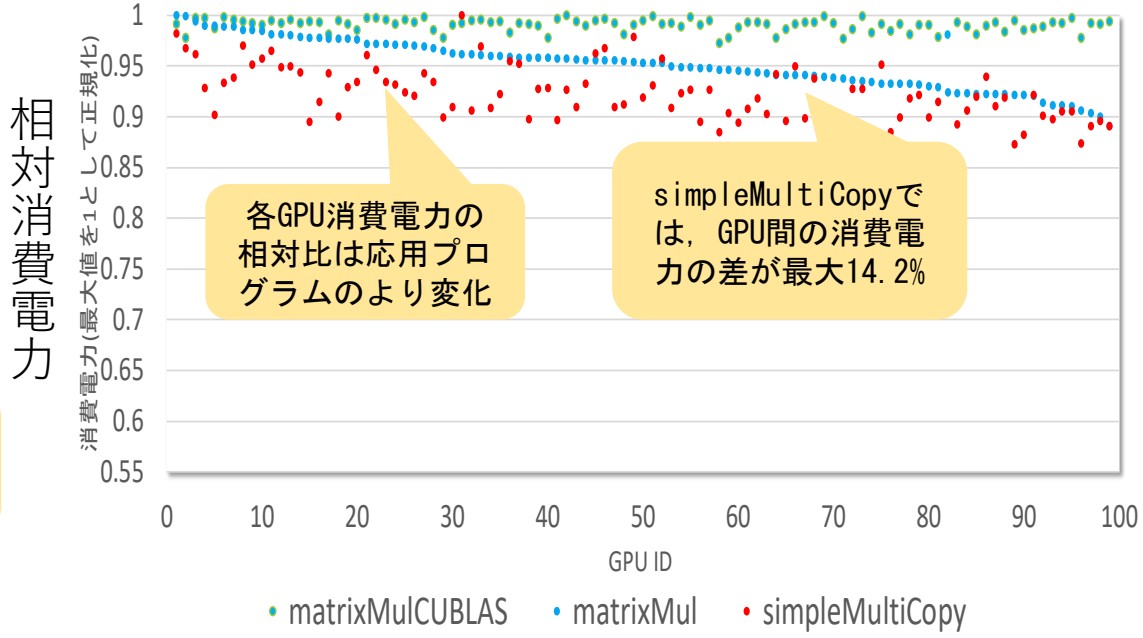
担当者： 飯田凌大，二間瀬悠希，小林龍之介，川口優樹，津邑公暁（名工大）

mSMSの電力評価と省電力方式の検討

TSUBAME3.0における大規模並列計算アプリケーション実行時の電力消費の評価・分析を行い，アプリ実行性能を保ちつつシステムの消費電力を削減するmSMSランタイムについて検討することが目的である．今年度は，高性能計算に欠かせないマルチノード・マルチGPU実行におけるGPU消費電力を計測し，そのばらつきを明らかにした．SMS実行時の処理プロセスの配置においてこの情報を活かすことも一つの手法と考えられる．ただし，今回，メモリ，CPUの消費電力測定を行うためのツール（RAPL）の利用が認められず，GPU以外の部分についての消費電力を明らかにすることができなかった．



TSUBAME3.0におけるGPUの消費電力ばらつき
ステンシル計算（シングルGPU実行）
25ノード，100GPU（Tesla P100）



TSUBAME3.0におけるGPUの消費電力ばらつき（50回の平均）
matrixMul，matrixMulCUBLAS，simpleMultiCopy（CUDA SDK）
nvidia-smi（ホストで5ms毎に実行）による測定

担当者：　大八木哲哉，三輪忍（電通大）

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 SMint開発と評価

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.