jh190039-ISH

緑川博子 （成蹊大学　理工学部）

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

概要

本研究では，高性能計算において広く利用される大規模マルチコア計算クラスタにおいて，高性能かつ並列プログラム開発の生産性を高めるためのシステムソフトウエアの構築を目的とする．具体的には，以下の研究項目を含む．

1. 複数ノードに分散したメモリを大域メモリとして一元化し，リモート／ローカルメモリにおけるメモリアクセス局所性を高め，高性能かつ大規模メモリ並列処理システムをコンピュータクラスタ上に実現する．（mSMSの構築と性能評価）
2. 既存の逐次プログラムから容易にクラスタシステム向けの並列プログラムに変換できる生産性の高い並列プログラミング開発環境，APIを構築する．実応用として音響解析FDTD（Finite Difference Time Domain）計算を行い，処理性能とプログラム開発環境の評価を行う．
3. クラスタでの典型的処理（各ノードに処理データを静的に等分した同期型並列処理など）だけでなく，グラフ，ツリーなどの不規則構造データ処理，非同期実行，データフロー型実行，トランザクショナルメモリモデルにみられる投機的並列実行など，クラスタシステムにおける大域メモリモデルを基本とした新たな実行方式，実行モデルを，性能および生産性という2つの観点から検討し，実行方式とAPIの検討を行う．（クラスタ向けTransactional memory　API）
4. 現在の並列プログラムにおいて，性能と生産性がエンドユーザの最大の関心事であるが，今後，省電力性が重要になることから，大規模メモリ・並列処理システムの低消費電力化を行う．（mSMSの電力オーバヘッドを解析し，省電力化方式を検討）

1. 大域アドレス空間をクラスタ上に実現するmSMSシステム

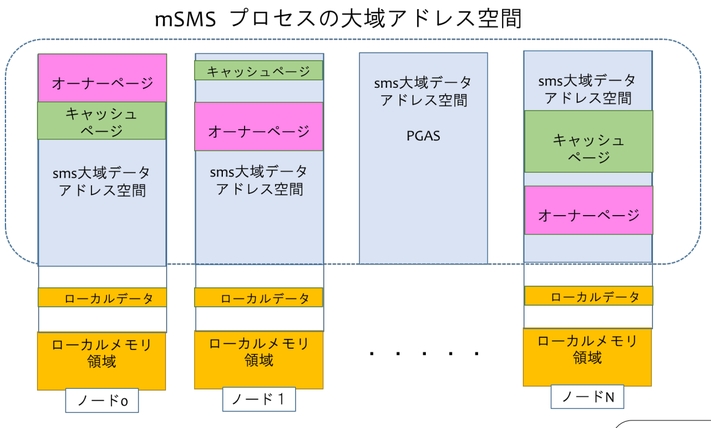


図1　クラスタ上で大域アドレス空間を実現するソフトウエア分散メモリmSMS

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

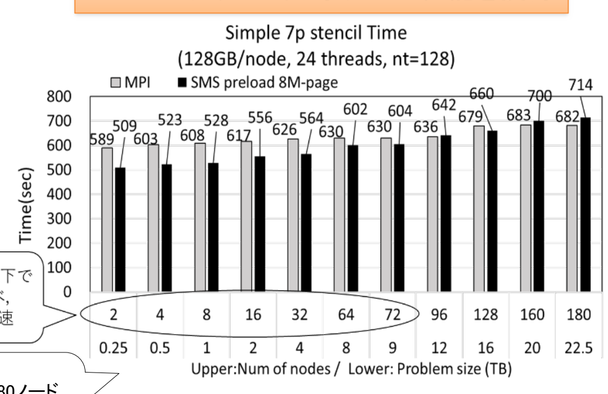
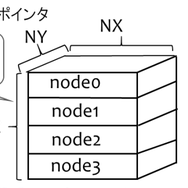


図2　7点ステンシル計算におけるMPIとmSMSにおける実行時間の比較

```
shared double A[NZ][NY][NX]::[NPROCS][1][1] (o, NPROCS); // 大域データ配列
shared double B[NZ][NY][NX]::[NPROCS][1][1] (o, 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



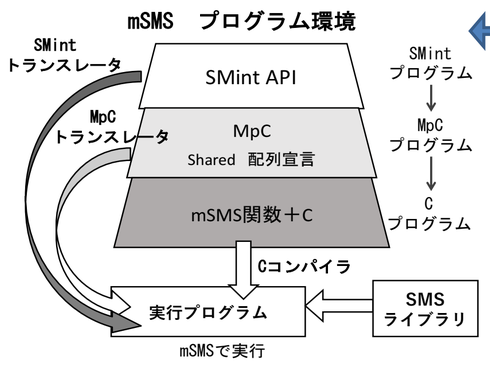
MYPID：MPI rankと同等
NPROCS：MPI 総プロセス数

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

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）とノード並列（mSMS）を実現

図3 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）

2a. グローバルビューモデルに基づくmSMS並列プログラミング環境と3つのAPI



mSMSにおいて3つのAPI　が利用可能　（SMSライブラリ関数利用，MpC，SMint）

ディレクティブベースAPI　SMint
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
＊GPU向けOpenACC
＊マルチコア向けOpenMP　との併用可能

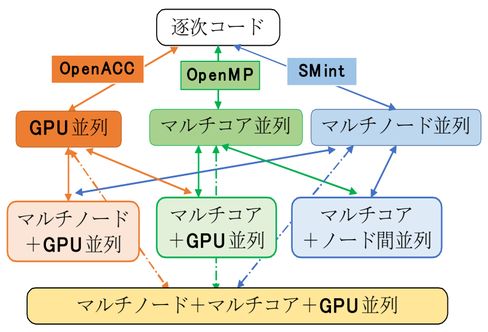


図4 mSMSにおける3つのAPI

```
#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();
}
```

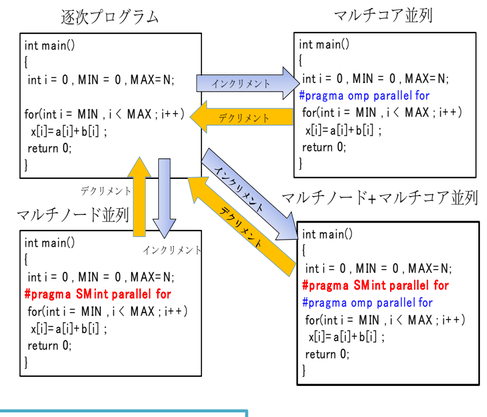
図5　SMSライブラリ関数によるプログラム

```
#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
}
```

図6 SMintによるプログラム



担当者：　阪口裕梧，緑川博子（成蹊大学理工学研究科）

2b. 実応用（音響FDTD法）における高効率実装と性能評価

mSMSを用いた3次元FDTD(2,4)計算のマルチノード並列化と性能評価実験を行う．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の開発を検討する．また、楽器や音響機器等の実問題への応用(図2)を検討し，境界条件等も含めた性能調査を行う．

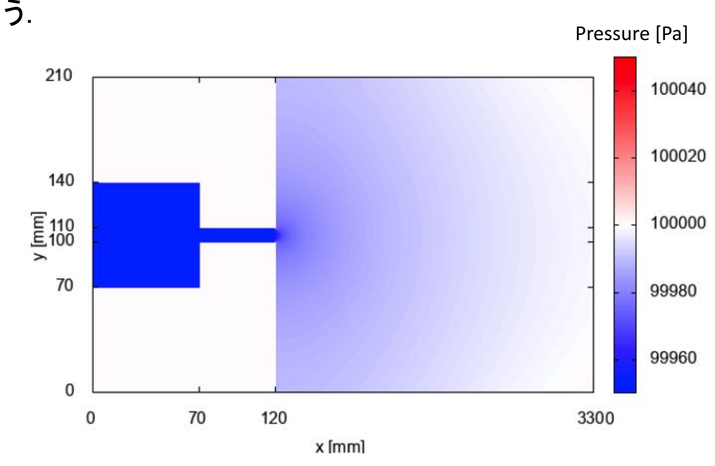


図 7　簡易バスレフポートスピーカーモデルの音響解析例

担当者：　田畑諒也（九州工業大学大学院情報工学府）
緑川博子（成蹊大学理工学部），
高橋公也（九州工業大学大学院情報工学研究院）

3. クラスタ向けTransactional Memory APIの検討

2a.で検討するAPIに加え，より直感的な記述が可能なトランザクショナルメモリ（TM）の記法をmSMS向けに適用することを検討する．既存のSoftware TM（STM）やDistributed TM（DTM）の実装を参考に，変数ベースやオブジェクトベースでのアクセス競合処理や，スレッドスケジューリングの，mSMS向け実装についても検討する．また，ハードウェアTM（HTM）であるIntel TSXをノード内並列処理に活用することで，マルチコア・マルチノードの並列処理をTMの枠組みで統一的・透過的にプログラミング可能とすることを目指す．

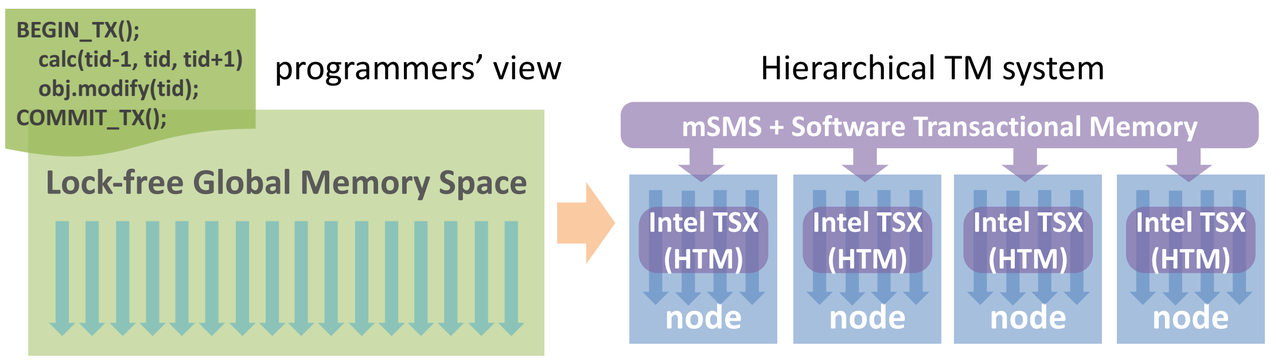


図 8　Transactional Memory API　の実装イメージ

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（名古屋工業大学大学院情報工学専攻）

4. mSMSの電力評価と省電力方式の検討

TSUBAME3.0上で大規模並列計算アプリケーションを実行した際のノード電力およびノード間通信電力の評価・分析を行い，アプリケーションの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．TSUBAME3.0の25ノード（計100台のGPU）上でステンシル計算プログラムを実行した際のGPU電力を計測し，GPU間に19.3Wの消費電力差が存在することをこれまでに確認した．

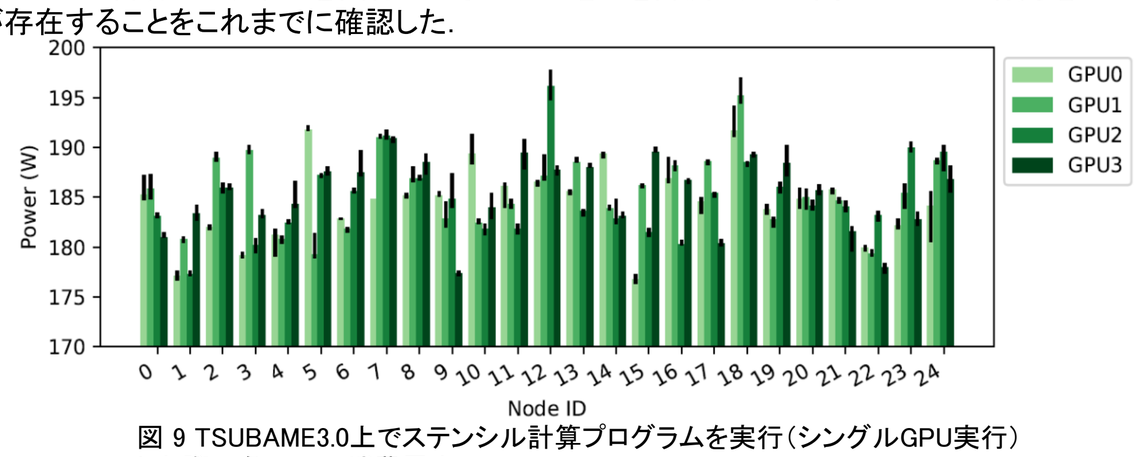


図 9 TSUBAME3.0上でステンシル計算プログラムを実行（シングルGPU実行）した際の各GPUの消費電力

担当者：　大八木哲哉，三輪忍（電気通信大学大学院情報理工学研究科）
緑川博子（成蹊大学理工学部）