Vol.2018-HPC-165 No.22
2018/7/31

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

緑川 博子[†1]

● **概要**：筆者らは，マルチノードに分散マップされた大規模ブローバルデータを提供し，マルチノードマルチスレッド並列による効率的な処理が可能で，生産性の高いプログラミング環境を提供するソフトウエア分散共有メモリ mSMS を構築している．これまで 72 ノードまでのクラスタシステムにおいて，ノード当たり 24-52 スレッドを用いて，3 種のステンシル計算アルゴリズムを実装し，MPI プログラムとの性能比較を行った．この結果，各ノードに同一の大規模共有アドレス空間を提供しつつ，ステンシル計算では MPI と比較しても十分高速な性能で処理できることを示した．本報告では，さらに大規模なシステム（180 ノード）における，性能調査を行った．この結果，180 ノード利用で，約 23TB の大規模データを各ノードプロセスへ提供し，ポインタを含む共有メモリモデルによるプログラミング環境を実現した．また，大規模ノード特有の問題点や，MPI 実装の影響などを明らかにした．

キーワード：ソフトウエア分散共有メモリ，PGAS，マルチノードマルチスレッドプログラム，MPI，大規模メモリ，共有メモリプログラミングモデル，マルチスレッド

1. はじめに

高性能計算応用においては，高速ネットワークで結ばれた複数計算ノードとノード内マルチコア(あるいは GPU などのアクセラレータ）を有効利用するため，MPI＋X（OpenMP，OpenACC など）によるプログラミング手法が広く用いられている．また，分散メモリモデルである MPI によるプログラムの書きにくさ，プログラム開発の生産性の低さを改善するために，PGAS（Partitioned Global Address Space）モデルで総称される様々な言語や API などが提案されている[9]． 現在，PGAS として，大域的データのグローバルビューを提供しながらデータの所在に考慮した様々なシステムが登場し，一定レベル以上の性能と MPI よりも高いプログラム生産性，あるいは，新しい並列実行モデルの提供，もしくは特定応用向けに高性能化など，様々な方向への研究が行われている．このため，PGAS とは，事実上，多種多様な実行モデルとシステム，言語を包含する名称になっている．多くの PGAS では，MPI と異なり，大域データ配列宣言や大域インデックスを利用できることでグローバルビューを謳っているが，各ノードプロセスがアクセスできる遠隔データの範囲に制限があったり，遠隔データ利用前に事前にアクセス領域の宣言が必要である場合もある．遠隔データアクセス API においても，データの所在（ノード番号など）を付してデータにアクセスする，あるいは MPI 片側通信のような API などを必要とする場合もある．多くの場合，真の同一共有メモリアドレス空間を提供しておらず，特定のコンパイラを介して，上記の API を下層通信実装の関数（MPI など）に変換している．すなわち，全ノードプロセスが，同一のアドレス空間を共有して，C 言語のポインタ変数やアドレスを使ったアクセスを行うことも可能とするシステムはほとんどない．

このような視点から，筆者らは，古典的ページベースのソフトウエア分散共有メモリ（SDSM）に，以下のような機能を新たに導入した mSMS を構築し，クラスタの各ノードプロセスに同一共有アドレス空間を提供し，ノード内マルチコアプログラミングと，ノード間マルチノードプログラミングをシームレスに行うプログラミング環境を実現した．

- 動的に生成・消滅する複数のユーザスレッドからの非同期・範囲無制限の遠隔データアクセスに対応．（マルチノード＋pthread，OpenMP，OpenACC，Cuda）
- 計算ノード間のページ送受信を高速化するマルチスレッドによる送受信通信機構
- 予めデータアクセス範囲が予測可能な時，計算（アクセス）前に大域データをまとめてローカルにフェッチする preload 機構
- マルチノード間の実行同期，通信低減型データ一貫性同期の提供

mSMS では，図 1 に示すように，マルチノードマルチスレッドシステムにおいて，各ノードの物理メモリサイズとノード数に応じた大規模大域データを定義でき，各ノードにおいて，各データ部分をスレッド並列で処理することができる．大域データは，任意のノードへの非対称な割り付けも可能である．大域データのアクセスに制限はないが，ローカルメモリアクセスを高めるように，ユーザがデータ割り付けを自由に決めることができる．mSMS は，現在，MPI を下層通信に用い，ユーザレベルソフトウエアで実装されているため，管理者権限は不要で誰でも容易に利用可能である．

†1 成蹊大学　Seikei University.

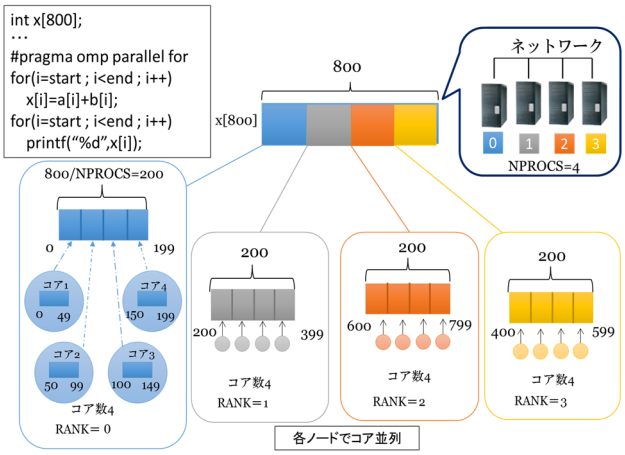

図 1　マルチノードマルチスレッド共有メモリプログラミングのイメージ

2. mSMS の概要

2.1 mSMS におけるプログラム

mSMS を利用するプログラムは，MPI と同様に SMS ライブラリ関数のみを用いて図 2(a)のように C で作成する．あるいは，MpC トランスレータを用い，図 2(b)のように大域データをデータ分散マッピング指定付き多次元配列宣言で利用することもできる[2]．いずれのプログラムでも，スレッド実装された既存の汎用数学ライブラリ関数や，OpenMP, OpenACC，pthread 関数などを，各ノード処理部分にそのまま使用できる．

ユーザプログラムは MPI と同様に各ノードでプロセスとして実行され，sms_alloc，sms_mapalloc などで確保された大域データは各ノードから見て同一アドレス空間上に確保され，グローバルビューを提供するだけでなく，どのノードからもアクセス可能である．このため，アドレスポインタを用いる C プログラムにもシームレスに対応でき，既存のプログラムを容易に移植できる．

2.2 mSMS における大域データと SMS ページ

ローカルノードにないデータにユーザプログラムスレッドがアクセスすると，SEGV ハンドラが起動し，該当ページを持つ遠隔ノードからページを取得し，図 3 のように，キャッシュページ領域として確保されたローカル物理メモリ上に取得し，ローカルノードの大域アドレス空間上にアクセス可能領域としてマップされる．

大域データ送受信の単位（ページサイズ）は SMS 独自のページサイズ(OS のページサイズの倍数)で行い，SMS ページ表により，どのページをどのノードが保持しているかを管理している．SMS ページサイズは，応用のデータアクセス特性に応じて，プログラム実行時にユーザが指定することも可能である．

MPI と同様に，用いる計算ノード名を列挙したマシンファイルと，利用可能な物理メモリ総容量，ローカルページとキャッシュページ，作業領域の各サイズ割合などを指定した mSMS メモリ構成ファイルを実行時に指定する．

```
int main( )
{
   sms_startup(&argc, &argv);

   array = (int*) sms_alloc(sizeof(int) * N, node);
      または 以下の分散マップ
   array = (int*) sms_mapalloc( dim, div, .....);

if (sms_pid ==0 ){       //   node別記述も可能
 #pragma omp parallel for
   for ( i = 0; i < N; i++ ) {
        array[i] = i;     .... マルチスレッド処理
   }
}
   :
 sms_shutdown();
}
```

malloc を変更するだけ 遠隔メモリを利用可能

OpenMP, pthread 利用可能

図 2 (a)　動的データ確保する mSMS プログラム

```
shared int a[NZ][NY][NX]::[NPROCS][1][1](0,NPROCS);
int main (int argc, char *argv[])
{
  int i, j, k;
  int size = NZ / NPROCS;
  //各ノードのアクセス範囲を計算　st – en
  int st = MYPID * size, en = (MYPID+1)*size;

  mpc_init ();
  #pragma omp parallel for
   for ( i = st; i< en; i++) { // st – en　範囲
    for ( j = 0; j < NY; j++)
     for ( k = 0; k < NX; k++)
       a[i][j][k] = .... マルチスレッド処理
   }
   mpc_barrier();
   mpc_exit();
}
```

配列宣言による 遠隔メモリ利用も可能

図 2 (b)　MpC トランスレータ利用 mSMS プログラム

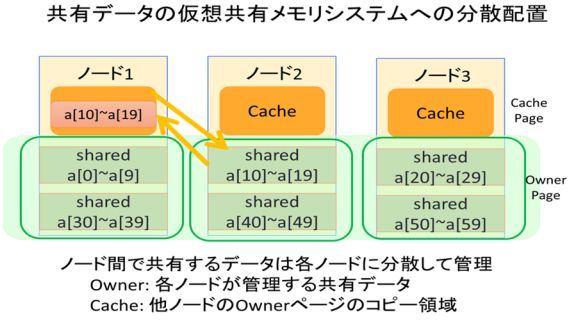

図 3　mSMS における大域データ分散例

2.3 マルチスレッドによる大域データへの非同期アクセスの実現

mSMS では，多くの PGAS 基盤システムのように，GET や PUT といったユーザが明示的に指定した時のみにデータを取得できる，あるいは，大域データアクセス範囲に制限を設ける，などを行っていない．このため，ユーザプログラムを構成する複数スレッドから非同期にページ要求が生成される．これに対応し，ユーザに一貫性のあるデータを提供するため，遠隔ノードから受け取ったページをユーザプログラムのアドレス空間に張り付ける瞬間は，ページ要求スレッド以外の実行中の全ユーザスレッドを一時的にサスペンドする機構を用いている．この手法は，out-of-core 処理のため，複数の遠隔ノードメモリを利用する分散大容量メモリシステム m-DLM [4-6]において開発した機構をベースにし，改良を加えている．ユーザスレッドの一時的なサスペンドは，オーバーヘッドが高いのではと当初危惧したが，実際に調べてみると，遠隔ページへのアクセスが非常に高い状況では，多くのスレッドが自分の要求したページのフェッチ待ちになっていること，遠隔ページへのアクセスが低い場合には，ページフェッチの機会が減り，サスペンドの機会が限られることなどから，実際には，サスペンドの影響は，実用に耐えうるレベルであることがわかっている[6].

この機構を実現するには，pthread や OpenMP プログラムにおいて動的に生成・消滅するスレッドに対し，現在実行中のユーザスレッドを正確に捕捉し，サスペンド・解除シグナルを送る必要がある．スレッド生成については，pthread_create を hook することで正確に捕捉できるが，スレッド終了については，pthread_exit を呼ぶとは限らない上，存在しないスレッドへのシグナル送信時に pthread_kill 関数が返すはずの失敗の返値が実現されていない Linux 実装に対処するため，現在は/proc 下の情報を用いて pthread のスレッド ID と Linux のプロセス ID を関連づけて，動的に生成，消滅するユーザスレッドの変動に対処している．

2.4 複数通信スレッドによるノード間通信の実現

mSMS では，図 4 に示すように，3 つの SMS システムスレッドを内部で用いている．ユーザプログラムが，初期化関数 sms_startup を呼ぶと，自動的に SMS システムスレッドが生成され，各種システムデータの初期化が各ノードで行われる．

mSMS では，ユーザスレッドからの様々な処理要求（メモリ割りつけ，ページ要求，終了処理など）は，図 4 の計算キュー（Cal. Que.）に登録され，起動時に自動生成された通信スレッド（Com）が計算キューから各ユーザスレッドの要求を取り出し，順次，処理する．通信スレッドは，ユーザプログラムからの様々な要求に応じ，該当する遠隔ノードに要求メッセージを送信し，担当ノード内で中心的な管理制御を行う．一方，他ノードへ要求したページの受信や，他ノードからのページ要求など，外からのメッセージ受信は，すべて受信スレッド（Rec）が行う．受け取ったメッセージの内，通信スレッドによる処理が必要な場合には，受信キュー（Rec. Que.）に要求を入れて通信スレッドに処理に任せる．他ノードからの返値などがある時は，返値キュー（Ret. Que.）に格納する．一方，非同期に送られてくる他ノードからのページ要求は，ページキュー（Page Que.）に入れて，ページ送信専用スレッド（Send）に処理を任せる．

通信には，古典的で単純な MPI 両側通信のみを用いている．理由は，MPI の内部実装レベルの差や制限などの影響を受けにくく安定している，また片側通信と異なり，アクセス可能データ範囲制約がないからである．MPI スレッドサポートレベルは，最高位の Multiple を利用している．一般に，Multiple 設定での複数スレッド通信は，単一スレッド（プロセス）通信よりも低性能と言われているが，機能別に設計された限られた数の通信スレッドが同時に処理を行うことにより，単一スレッドによる通信に比べ，効率的な通信が行われている．いずれの通信スレッドも，通信効率の良いコアにそれぞれバインディングしている．コアバインディングは，通信性能に大きな向上をもたらすことが，わかっている．

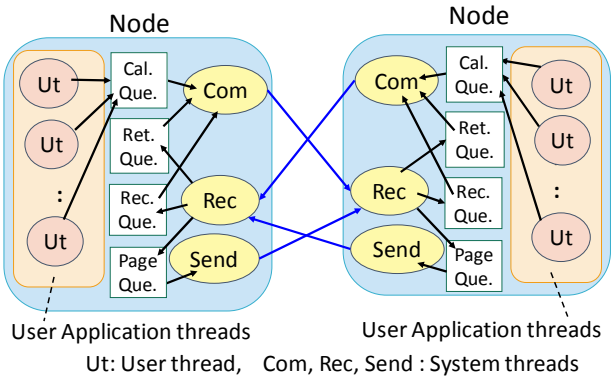

図 4　MSMS の内部実装

2.5 遠隔ノードからのページ取得プロトコル

ここでは，遠隔ノードからページ取得機構について述べる．遠隔ノードとのページ「交換」プロトコル・通信手法に関しては，Multi-SMS[3]や，m-DLM [5,7,8]において数十のマルチスレッド通信実装方式の性能調査を行ってきた．mSMS の現実装では，m-DLM と異なり，遠隔ノードとのページの交換（swap）は行わず，cache 領域への遠隔ページ取得のみを行う．m-DLM では，最も効率が良いと思われるプロトコルが用いた MPI 内部実装の制限（バグ？）により，実現できない場合があったが，swap-out を行わない現

mSMS では，安定かつ高効率のプロトコルの実装が可能であった．

ユーザプログラム中の１スレッドが，ローカルノードにないデータへアクセスしてから，SEGV シグナルハンドラ内で，他ノードから該当ページ取得，貼り付けを完了し，該当ユーザスレッドの実行再開までの，手順を以下に示す．図 5 はページ要求送信ノードでの処理，図 6 はページ要求受信ノードでの処理を示す．

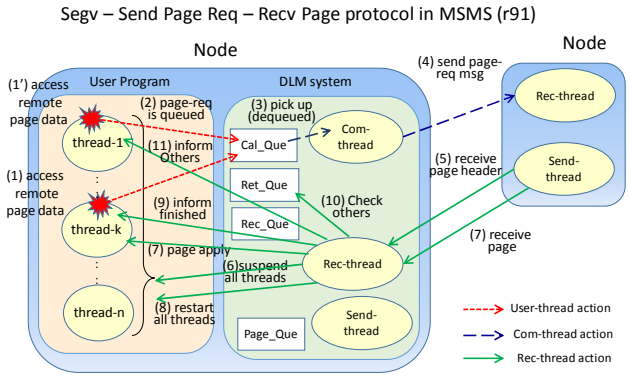

図 5　送信ノード：ユーザスレッドのページ要求処理プロトコル

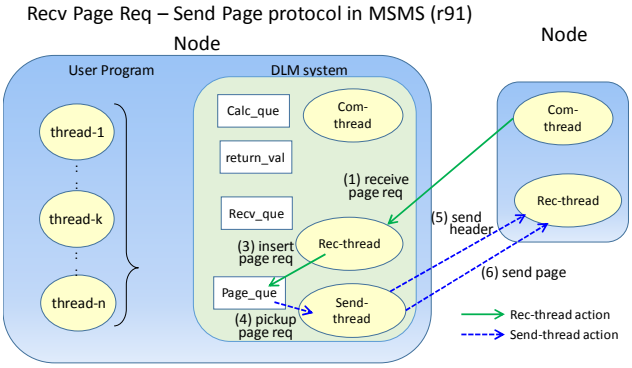

図 6　受信ノード：他ノードからのページ要求処理プロトコル

遠隔ページ取得手順

(1) ユーザスレッドがローカルメモリにないデータをアクセスすると Segv シグナルハンドラが起動される．

(2) ユーザスレッドはハンドラ内で Cal キューにページ要求を登録して，ページを待つ．

(3) 通信スレッドが，Cal 要求キューからページ要求を取り出す．

(4) 通信スレッドが SMS ページ表を見て，該当ページを持つ遠隔ノードにページ要求を送信．

(5) ページ要求を受け取った遠隔ノードは該当するページを取り出し，メッセージヘッダーとページ本体を要求元ノードへ送る．受信ノードの受信スレッドは，メッセージヘッダーのみを受け取る．

(6) 受信スレッドが，ユーザプログラムから起動された実行中の全てのユーザスレッドを一時停止させる．

(7) 受信スレッドは，メモリサーバからの送られたページを，ユーザデータ領域に直接，受け取る．

(8) 受信スレッドが，(6)で一時停止させたユーザスレッドを再開させる．

(9) 受信スレッドが，SEGV ハンドラ内で要求ページを待っているユーザスレッドを起こす

(10) 受信スレッドが，受け取ったページと同じページを要求していて SEGV ハンドラ内でページを待つユーザスレッドがあるか調べる．

(11)もし，待っているスレッドがある場合には，このスレッドを起こす．

2.6 実行同期とデーター貫性同期

mSMS では，データー貫性管理を単純化するため，同期型データー貫性保持（weak consistency model）のみを現在は実装している．図 1，図 3 に示すように，ノード数を増やすほど利用できる大域データのサイズを大きくできるように，各ノードが大域データを分担して owner ページとして保持し，他ノードからの cache ページと区別する．データー貫性同期時に，（１）各ノードの cache ページ変更部分をそのページの owner ノードに伝え，大域データに反映される（sms_barrier），あるいは，（２）cache ページに変更があってもそのまま cache ページを破棄し，大域データに反映しない(sms_sync_drop)，の 2 種を更新方式がある．いずれも，cache ページは捨てられる（該当アドレスページ領域はアクセス不可に設定され，次回以降に segv が起こるようにする）．一方，データー貫性制御はおこなわず，cache ページを保持したまま実行同期のみ行うこともできる(sms_sync)．

2.7 大域データの事前フェッチ：preload

mSMS では，通常，ユーザスレッドが実行中にローカルにない大域データにアクセスしてから，遠隔ノードからのページ取得が行われるため，当該スレッドが計算を一時中断してデータの到着を待つ遅延時間が生じる，また，当該ページをアドレス空間に張り付ける瞬間にも，データー貫性保持のため，その他のユーザスレッドの実行も一時的にサスペンドされる．しかし，多くの応用で，配列データを小規模のブロックに分割して処理を進める場合など，あらかじめ，アクセスするデータ範囲が計算前にわかっている場合も多い．このような応用処理パターンに対し，計算開始前にあらかじめアクセスする領域を，SIGSEGV シグナルハンドラを介さずに，事前フェッチを行う関数 sms_preload_array，sms_preload を用意している．いずれの関数もアクセス前に指定した範囲のページをまとめて cache ページとして取得する．通常のページフェッチでは，SEGV を起こしたアドレスを含む当該ページ 1 枚をフェッチするだけであるが， sms_preload は，大域データの開始アドレスから指定サイズの連続ページを一度で転送する．

sms_preload_array では，大域配列データの中から，任意の次元サイズの部分配列データの取得が可能で，関数内部で，指定データ範囲のアドレス連続性を調べ，連続ページはまとめて転送する．指定範囲にあるデータが owner ページにある，あるいはすでにローカルにフェッチされている場合には実際の転送を行わない．さらに関数引数で read か write かをあらかじめ指定できるため，同じデータをリードしてからライトするなど，SEGV ハンドラ経由では，2 回のメモリアクセス属性変更が必要な場合でも，一度でアクセス属性設定ができる．

また，read はせずに write のみ行う領域については，遠隔ノードから実際のデータ領域転送はせずに，指定範囲のページをリードライト可能と設定するのみ．あるいは，すでにキャッシュされているページに対し，上書き書き込みをする overload 関数など，幾つかのユーティリティを設計，実装中である．

3. 大規模クラスタおける mSMS によるステンシル計算

すでに，mSMS の初期性能評価実験として，72 ノードまでの TSUBAME3 [10,11] を用い，典型的な計算カーネルの一つとして，3 次元配列のステンシル計算のマルチノードマルチスレッド並列処理を行った[12]．今回は，Taubame3.0 の全体ノード数（540）の 1/3 にあたる 180 ノードを利用して，さらに大規模なクラスタにおける mSMS の稼働・性能の調査を行った．Tsubame3.0 のノード間接続は，Intel Omni-path（HFI100Gbps x 4）で，1 ノードに 4 GPU と 2 CPU (E5-2680 v4, 2.4 GHz, 14 cores / 28 threads)を有する．今回の実験では，ノード内の 2CPU を OpenMP で利用する．TSUBAME3 は，1 ノードあたり，256GiB の主メモリを持つが，そのうちの半分（128GB）を，問題全体の 3 次元配列の部分ブロックとして割り当てることとし，ノード当たり（bx, by, bz）= (4096, 2048, 1024) のブロックを割り当て，全体として，大域データ配列を z 方向に分割する単純な分割方式とした．各ノードでは，担当するブロックデータ領域を，プロセッサの L2, L3 キャッシュサイズを意識した小ブロックにさらに分割し，OpenMP を用いマルチスレッド処理している．実行には，1 ノードをすべて占有にして実行しており，CPU 処理に他の job の影響を受けないようにしているが，72 ノード利用時より，同じプログラムでも，TSUBAME3 では，job 実行毎に性能がばらつくことがある．

さらに，今回の実験では，128 ノード以上の多数ノードにおける intel-MPI の動作が安定せず（128 ノード未満と実装形態が異なるため），多数ノードになると，MPI 起動時のノード間接続にリトライが何度も繰り返される，規定時間内に通信が確立せずに，エラー終了する，MPI 起動時にサスペンドして，アプリプログラムが実行できないまま，job が終了するなどの様々な状況が発生した．

3.1 ステンシル計算アルゴリズム

用いたステンシルアルゴリズムは，（1）毎時間ステップ毎，隣接ノードとデータ交換を行うアルゴリズム A1（simple-stencil）と，（2）ノード間の通信回数を減らすための時間ブロッキング（全体時間ステップ 128，時間ブロックサイズ 16）を用いて，時間ブロックステップ毎のデータ交換にしたもので行った．時間ブロッキングでは，冗長計算が生じるアルゴリズム A2 (redundant-stencil) と，冗長計算を省くために内部を 2 フェーズの処理の分けた冗長計算のないアルゴリズム A3 (no-redundant-stencil) を実装し，合計 3 種類のアルゴリズムを用いた．

また，各アルゴリズムに対し，前述の preload を利用して，計算前に事前に隣接ノードからの一括データフェッチを行うものと，計算時のアクセス時の segv ハンドラ経由での遠隔ページフェッチを行うものの 2 種（例 A1 と A1p）を調査した．

ステンシル計算としては，同じデータアクセスに対し，計算量の多い近傍 27 点ステンシルと計算量の少ない近傍 7 点ステンシル計算の 2 種を行った． それぞれの計算で用いる 1 ノード内で用いる適切なスレッド数（OpenMP 利用時）を 4 ノードによる並列処理で事前調査した．この結果，図 7 に示すように，それぞれ性能が飽和し始めるスレッド数として，7 点ステンシルでは 24 スレッド，27 点ステンシル 52 スレッドを用いた．ここでは紙面の制約から，7 点ステンシルの結果のみを示す．

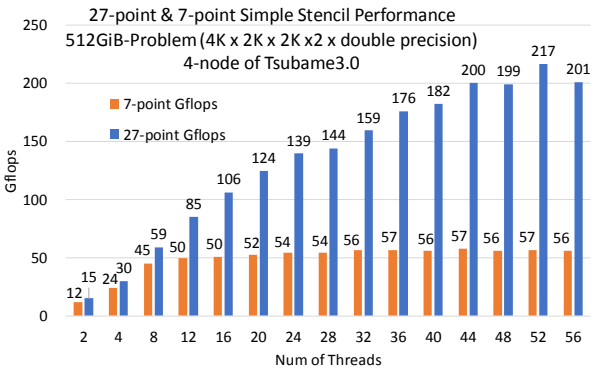

図 7　4 ノード mSMS 利用時単純ステンシル性能

3.2 単純ステンシル計算（空間ブロッキング）A1 の性能

図 8(a)(b)に 2 ノード（256GiB 問題）から 180 ノード（23TiB 問題）までの単純ステンシル A1 の実行時間と性能を示す．各ノードの担当データ領域の両側の袖領域のデータを通常の sigsegv によりフェッチした場合（A1）と sms_preload_array 関数を用いて事前フェッチした場合(A1p)を示す．性能，実行時間ともに，多数ノードになるに従い，preload 利用の効果が大きくなる．図 8(c)(d)は，segv と preload の各方式で rank0 における総実行時間の処理成分を示す．Segv 方式の計算時間（Cal）には，segv による

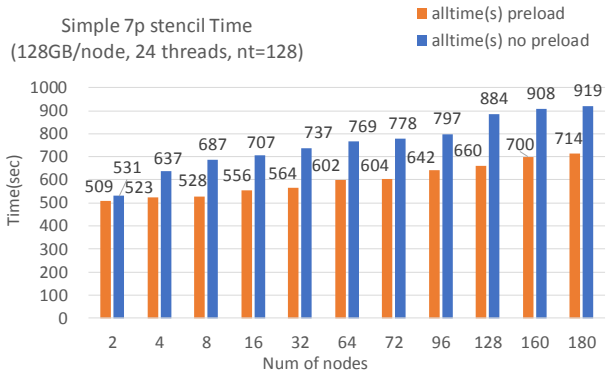

図 8 (a)　mSMS 単純ステンシル計算 (segv)　実行時間

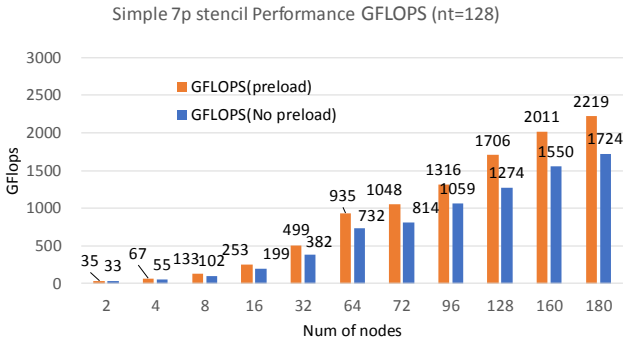

図 8 (b)　mSMS 単純ステンシル計算 (preload)　性能

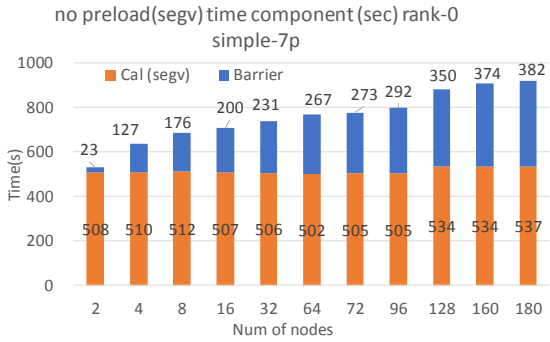

図 8 (c)　単純ステンシル preload なし(segv)　時間成分

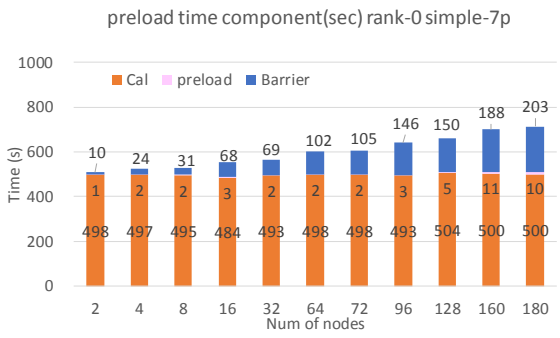

図 8(d)　単純ステンシル preload あり　時間成分

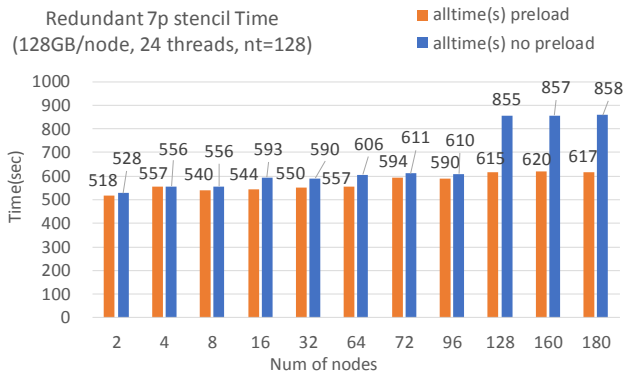

図 9 (a)　mSMS 時間ブロッキングステンシル計算　実行時間

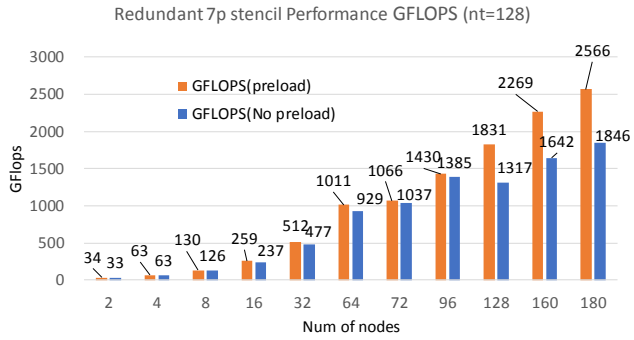

図 9 (b)　mSMS 時間ブロッキングステンシル計算　性能

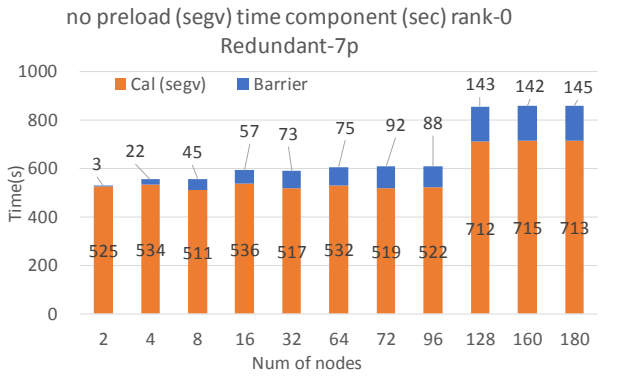

図 9 (c)　時間ブロッキングステンシル preload なし　時間成分

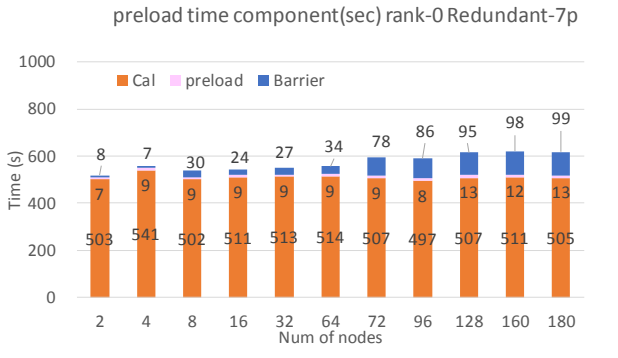

図 9 (d)　時間ブロッキングステンシル preload あり　時間成分

オーバーヘッドが含まれる．preload 方式では，Cal 時間の他に preload 時間が示されている．

単純ステンシル計算では，全体で 128 回の時間ステップのデータ更新，データのフェッチがある．Preload 方式では，計算時間が segv 方式に対し 2-7%程度高速化されているが，全体実行時間における大きな差は主にバリア時間から生じ

ている．この原因は，preload を用いない場合，毎回の各ノードの計算時間がノードによって非常にばらつき（segv によるページフェッチの待ち時間が一定でないことも一因と思われる），毎回のバリア同期の度に最も遅いノードを待つことにより，バリア時間が膨らんでしまう．しかし，今回の実験では， preload を用いて計算時に一切の segv や通信が発生しない状況にしても，各ノードでの計算部分のみの時間にばらつきが発生しており，その原因は調査中である．この計算時間の各ノードのばらつきにより，多数ノードになるほどバリア時間が非常に大きくなって，全体の性能を下げている．単純ステンシルでは，更新回数，バリア回数が多いために，preload 利用の有無による各ステップの実行時間のばらつきが顕著に表れる．

3.3 時間ブロッキングステンシル計算 A2 の性能

空間，時間ブロッキングによりデータアクセス局所性を高めた時間ブロッキング処理の性能を図 9 に示す．時間ブロックサイズ(bt)が 16 の場合，近傍データの交換サイズは bt 倍に増えるものの，ノード間のデータ交換回数は 128 回から 8 回に減少する．単純ステンシル計算に比べ，全体の実行時間は短縮化され，preload の有無による性能差も小さくなっているが，128 ノード以上を用いた場合とそれ以下のノード数では明らかに違う状況が生まれている．ここでは紙面スペースの制限で示していないが，27 点ステンシルの場合も同様の状況を示す．すなわち，segv 方式の計算時間とバリア時間が 128 ノードを境に急激に増大している．この原因は，調査中であるが，128 ノード以上とそれ未満の MPI の実装方式が非常に異なることに起因するのは明らかで，SMS ページサイズを大きくして，segv 発生回数とページ要求回数を減らすと，顕著な差は小さくなる．図 9 (c)(d)のバリア待ち時間の成分は，単純ステンシル計算 A1 に比べるといずれも小さく，preload の有無による短縮化率も 128 ノード未満においては小さい．

単純ステンシル計算と時間ブロッキングステンシルの両方で，時間ステップ毎の同期時には，データ一貫性同期のうち，キャッシュページを破棄する sms_sync_drop を用いている．このため，同期時にデータ更新データを owner ノードに通知したり，cache ページの更新部分を抽出する作業は省略されている．

3.4 MPI プログラムとの性能比較

MPI プログラムと mSMS との性能を比較するため，単純ステンシルにおける性能を比較した．図 10 に実行時間と性能を示す．この結果，128 ノード未満の場合，preload を用いると mSMS プログラムのほうが MPI プログラムよりも高速であることがわかる．

MPI も mSMS プログラムも，実際の隣接ノードとの大域データの転送の回数は 1 時間ステップあたり 2 回で等しく通信データサイズも同じになる．ただし，mSMS では，図 5，6 で示したようにページや preload データの転送前に，固定サイズの短いメッセージヘッダーを送受信するため，実際の通信回数は倍になる．また，preload ではデータ領域がすでにローカルノードにキャッシュされているか，連続データとして 1 回の MPI 通信でできるか，MPI 実装において 1 回の最大通信サイズを超えないかなどをチェックしており、preload 指定範囲が大きい場合には、この計算オーバーヘッドが大きくなる．

MPI プログラムは，非同期送受信，同期送受信の 2 種，マルチスレッドサポートレベルの変更などを行ってみたが，いずれも 128 ノード未満では実行時間に影響はなかった．しかし，128 ノード以上では，非同期通信よりも同期通信のほうが高速である．図 10 は，同期通信による MPI プログラムとの比較を示している．

128 ノード未満における各プログラムの実行時間の処理成分の詳細を分析したところ，1 回当たりの隣接データの送受信時間（2 回の MPI 通信）が，mSMS での preload 時間（ユーザスレッドによるデータ要求からデータの受信まで）の時間よりも長くかかっていることがわかった．詳細に調査しても，この現象は安定しており，これにより，マルチスレッドによる mSMS の同時通信機構が，シングルスレッドによる MPI プログラムの通信よりも効率的である可能性がある．

一方，128 ノード以上の実験では，Tsubame3.0 の利用可能な時間的制約から，MPI 通信の詳細状況は調査できてい

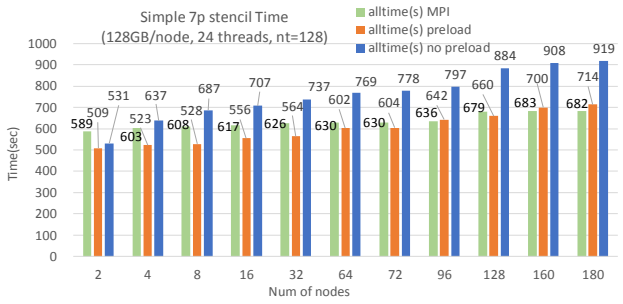

図 10 (a)　mSMS と MPI の比較 7 点単純ステンシル計算実行時間

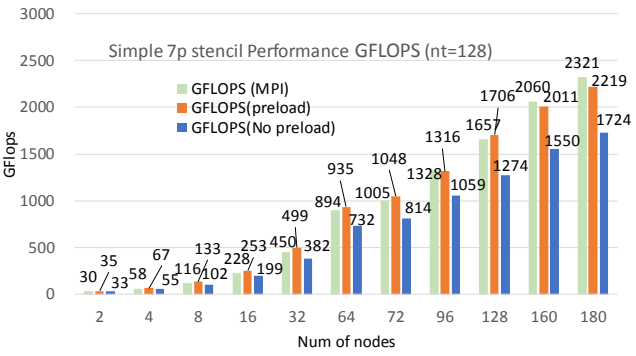

図 10 (b)　mSMS と MPI の比較 7 点単純ステンシル計算 性能

ない．しかし，128 ノード以上のクラスタでは，MPI 実装と MPI 通信状況が大幅に変化しており，多数ノード利用時の MPI の利用メモリの増大や，多数ノード時に生成される MPI システムスレッドなどにより，mSMS の通信スレッドとの利用コアの競合がおきている可能性もあり，今後，調査していく予定である．

4. 終わりに

本報告では，マルチノードマルチスレッド向けの高性能 SDSM システムとして新たに設計した mSMS を用い，大規模クラスタシステムにおいて共有メモリプログラミングモデルで記述したステンシル計算の稼働実験を行った．この結果，180 ノードを利用し，各ノードに約 130GB のデータを分散配置して，全体として 23TiB の大域共有データを定義し，約 30TiB の仮想アドレス空間を持つプロセスを各ノードで稼働させて，大規模な共有メモリデータプログラミング環境を実現した．

また，通信が頻繁な単純ステンシルアルゴリズムにおいても，preload 機能などにより，MPI に匹敵する（場合によっては MPI 以上の）性能が mSMS により得られることを示した．通信頻度を低減した時間ブロッキングアルゴリズムにおいては，128 ノード未満のシステムでは，preload 利用の有無での性能差は小さく，segv 方式であっても，MPI プログラムの性能に近い性能が得られる．

今回の実験で，これまでの 128 ノード未満の状況と，128 ノード以上の MPI 通信環境が大幅に違うことが明らかになり，大規模ノードにおいて考慮すべき点，これまでとは違う様々な現象については，今後，調査していく予定である．これまで mSMS は，InfiniBand と MVAPICH を用いて設計，実装してきた経緯もあり，Tsubame3.0 では，通信媒体と MPI 実装が変更されたことにより，性能チューニングの点で未知の部分がある．preload に関しても，効率を重視する点から連続アドレス領域データは，MPI が許容する最大サイズまで，一度に通信するようにしていたが，大規模ノードでは，一斉に preload が始まると通信バッファサイズなどの点で弊害が生じる可能性もある．また，今回，OS のメモリ管理における overcommit 機能を生かし，物理メモリサイズを超える大規模なアドレス空間を持つプロセスを各ノードで実行させたが，これにより既存の MPI 実装などにどのような影響がでるのかも調査が必要と考えている．

また，今後は，ステンシル計算のように MPI で記述しても，それほど，大変ではない応用ではなく，共有メモリプログラミングモデルを生かした応用処理についても性能を調査していく．ただし，いずれの応用においても，メモリアクセス局所性を高めたアルゴリズムを用いること第一とし，プログラミング生産性と性能のバランスにおいて，実用に十分利用可能な分野への適用を考えている．

謝辞

今回の mSMS を使った大規模クラスタ利用実験では，東京工業大学学術国際情報センターの TSUBAME グランドチャレンジ大規模計算制度（平成 30 年度春期 6 月カテゴリ B）を利用させて頂きました．実験にあたっては，センターの関係各位に多くのご支援に頂きましたことをここに深謝いたします．

参考文献

[1] 緑川博子，飯塚肇 : "ユーザレベル・ソフトウエア分散共有メモリ SMS の設計と実装", 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム,Vol.42, No.SIG9(HPS 3), pp.170-190, (2001,8)
[2] 緑川博子, 飯塚肇:"メタプロセスモデルに基づくポータブルな並列プログラミングインターフェース MpC",情報処理学会論文誌：コンピューテイングシステム,Vol.46 No.SIG4(ACS9), pp.69-85,（2005,3)
[3] 緑川博子,岩井田匡俊："マルチスレッド対応型分散共有メモリシステムの設計と実装", ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS2015, HPCS2015 論文集,（2015,5-19）
[4] 緑川博子、齋藤和広、佐藤三久、朴 泰祐："クラスタをメモリ資源として利用するための MPI による高速大容量メモリ"、情報処理学会論文誌，コンピューティングシステム，Vol.2, No.4, pp.15-36, (2009.12)
[5] H. Midorikawa, K.Saito, M.Sato, T.Boku: "Using a Cluster as a Memory Resource: A Fast and Large Virtual Memory on MPI", Proc. of IEEE Cluster2009, 2009-09, Page(s): 1-10 (DOI: 10.1109/CLUSTR.2009.5289180)
[6] 鈴木悠一郎, 鷹見友博, 緑川博子："マルチスレッドプログラムのための遠隔メモリ利用による仮想大容量メモリシステムの設計と初期評価", 情報処理学会、Hokke2011,ハイパフォーマンス研究会 Vol.2011-HPC-132, No.13, pp.1-6, (2011.11)
[7] 大浦陽，緑川博子，甲斐宗徳："遠隔メモリ利用による Out-Of-Core OpenMP プログラムの性能評価実験", 第 15 回情報科学技術フォーラム FIT2016， FIT2016 論文集 第一分冊 B-004， p.177-178， 富山大（富山）(2016,9.9)
[8] 緑川博子，北川健司， 大浦 陽: "マルチスレッドプログラム向け遠隔メモリサーバにおけるページ交換プロトコルの評価実験",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）,2017-HPC-160(36),pp.1-9 ，（秋田県秋田市）(2017-07-26)
[9] M.D. Wael, et al.: “Partitioned Global Address Space Languages”, Journal of ACM Computing Surveys (CSUR), Vol.47, No.62 (2015)
[10] Tsubame3 http://www.gsic.titech.ac.jp/tsubame3
[11] S. Matsuoka, T. Endo, et.al “Overview of TSUBAME3.0, Green Cloud Supercomputer for Convergence of HPC”, AI and Big-Data, GSIC, Tokyo Institute of Technology, e-Science Journal, Vol. 16, pp. 2–9, 2017
[12] 緑川 博子 ，北川 健司:"高柔軟性と高性能を提供するマルチノードマルチスレッドプログラム向け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.2018-HPC-163, No.10,pp.1-8 (2018-02-21)