Vol.2018-HPC-163 No.10 の改訂版
2018/3/22

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

緑川 博子[†1]　北川健司[†2]

(1) **概要**：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 M-SMS を新たに構築した．M-SMS は，(1)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・範囲無制限のデータアクセスに対応，(2) 計算ノード間のページ送受信を高速化する専用マルチスレッドによる送受信通信機構，(3) 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，(4)マルチノード間の通信低減型データ一貫性同期などを実装している．2 種のステンシル計算アルゴリズムに対し，Tsubame3 を用いて 72 ノードまでのマルチノードマルチスレッド処理を行ったところ，preload 機能の効果は高く，単純ステンシル計算において，MPI プログラムによる実行性能を上回る高性能を獲得した．

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，最近では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].

筆者らは，PGAS モデルが広く認知される以前に，クラスタシステム上に大域アドレス空間を提供するページベースのソフトウエア分散共有メモリシステム（SDSM）SMS[1]と，大域データを各ノードに分散マッピングをするための API を持つ C の拡張言語 MpC を開発した[2]. 当時の SDSM は，対象とするユーザプログラムはシングルスレッドプログラムで，マルチスレッドユーザプログラムに対応するには OS，ノード間通信その他に，スレッドに関わる様々な未成熟部分があり，実装上の問題があった．さらに，ページベース SDSM では，分散ノード上に仮想的な大域データが提供され，どのノードからも制限なく，ローカルメモリにあるデータと同じようにアクセスできたが，多くのシステムでは分散データマッピング機能が未熟であったため，多くのユーザがデータの所在やアクセス局所性を無視したプログラムを作成し，その遅さに失望した．この結果，SDSM のブームは去り，MPI プログラムこそ正しい道という風潮が当時広まった．現在，CPU 性能に対するメモリアクセスの相対性能は，当時に比べさらに劣化し，たとえローカルメモリデータであっても，メモリアクセス局所性を考慮しないプログラムがいかに低性能であるかは常識になっている．現在，PGAS という新しい名称の下，大域アドレス空間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，そこそこの性能と 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．

本報告では，古典的ページベース SDSM に，以下のような機能を新たに導入した SDSM システム M-SMS を構築したので，報告する．

(2) 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・範囲無制限のデータアクセスに対応.
（マルチノード＋pthread，OpenMP，OpenACC，Cuda）
(3) 計算ノード間のページ送受信を高速化する専用マルチスレッドによる送受信通信機構
(4) 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
(5) 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

M-SMS 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，

†1 成蹊大学　Seikei University.　JST CREST
†2 (株)アルファシステムズ Alpha Systems, Inc.

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．

また，M-SMS は，ユーザレベルソフトウエアで実装されているため，管理者権限が不要で誰でも容易に利用可能である．

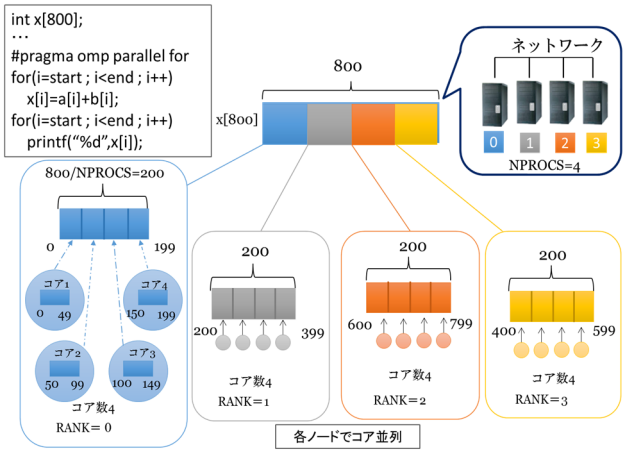

図 1　マルチノードマルチスレッド共有メモリプログラミングのイメージ

2. M-SMS の概要

2.1 M-SMS におけるプログラム

M-SMS 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 M-SMS 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，ユーザ指定可能)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 と同様な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

```
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 M-SMS 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 M-SMS プログラム

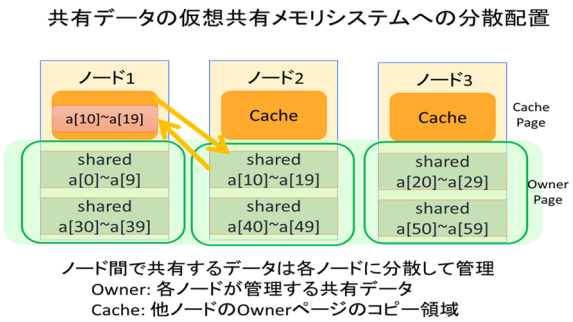

図 3　M-SMS における大域データ分散例

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メモリ構成ファイルを実行時に指定する．

2.3 大域データへのマルチスレッド非同期アクセスの実現

M-SMS 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，今回，改良を加えた．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，限られた状況でしか影響しないことがわかり，実用に耐えうるレベルの実装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装されていない 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，動的な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

今回，設計・実装した M-SMS 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．

M-SMS 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．

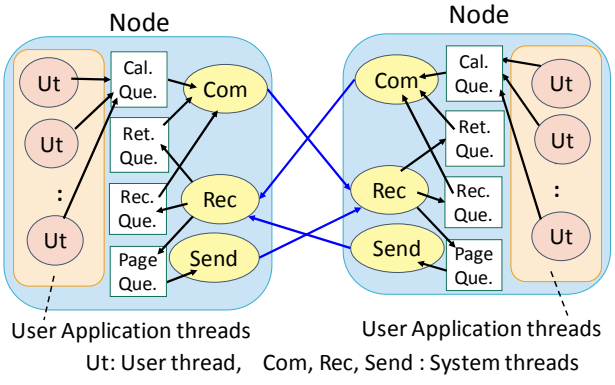

図４　M-SMS の内部実装

2.5 遠隔ノードからのページ取得プロトコル

ここでは，今回新たに設計・実装した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．M-SMS 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 MPI の内部実装の制限（バグ？）により，実現できない場合もあったが，swap-out を行わない現 M-SMS 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，図６は，それぞれ，ページ要求送信ノードでの処理，ページ要求受信ノードでの処理を示す．

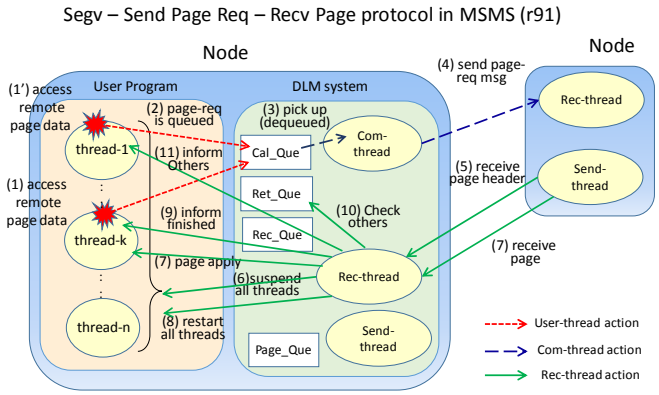

図 5　送信ノード：ユーザスレッドのページ要求処理プロトコル

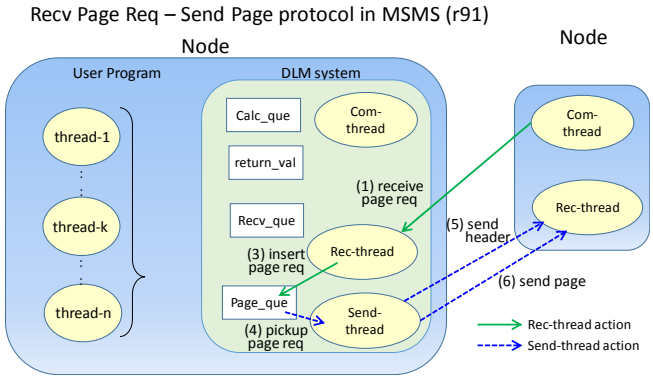

図 6　受信ノード：他ノードからのページ要求処理プロトコル

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 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，該当ページを持つメモリサーバにページ要求を送信．

(5) ページ要求を受け取った送信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計算ノードへ送る．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　実行同期とデータ一貫性同期

M-SMS では，データ一貫性管理を単純化するため，同期型データ一貫性保持（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．データ一貫性同期時に，（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，あるいは，（２）cache ページに変更があってもそのまま cache ページを破棄する，の 2 種をサポートする．この他に 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．

2.7　大域データの事前フェッチ：preload

M-SMS 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，データ一貫性保持のため，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，規則的な配列データなどに対し，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．このような応用の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．

3. M-SMS の初期性能評価実験

M-SMS の性能を評価するため，通常アカウントで，最大 72 ノードまでノード数が利用できる TSUBAME3 [10] を用いて，典型的な応用，３次元配列のステンシル計算を並列処理した．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．

昨年夏に新しく利用可能になったばかりの Tsubame3 における実行は，この M-SMS 利用の有無とは無関係に，現在，ネットワークや CPU，バッチシステムなどの不具合により，実行エラーになることがあり，job 実行までの待ち時間も非常に長いため，投稿時点で図 8 の一部のデータは取得できていないものがある．計測には，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，同じプログラムでも，job 実行毎に性能がばらつくことがあり，原因は定かではないが，多数ノード利用時に，明らかに遅いノードが 1,2 ノード出現することがあり，全体の実行時間を引っ張り，特異点とわかるほど，実行時間が長くなることがある．このため，何回かの実行を余儀なくされることがある．

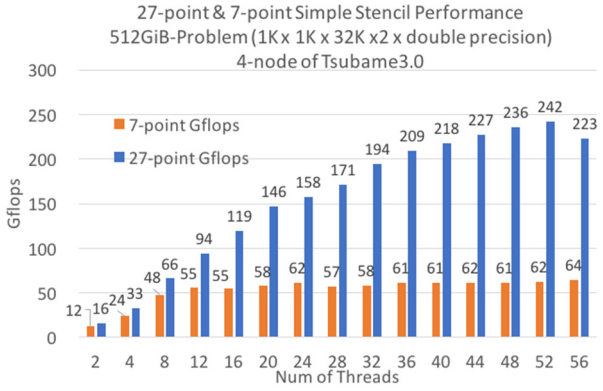

図 7　4 ノード MSMS 利用時単純ステンシル性能

用いたステンシルアルゴリズムは，（１）マルチノードへの単純な空間ブロッキングによる毎時間ステップ毎の隣接データ交換・計算処理（simple-stencil）と，（２）ノード間の通信回数を減らすための時間グロッキング（全体時間ステップ 128，時間ブロックサイズ 16，もしくは 32）の 2 種で行った．利用スレッド数は，図 7 に示す事前調査により，それぞれ性能が飽和し始めるスレッド数，24 スレッド（7 点ステンシル），52 スレッド（27 点ステンシル）を用いた．ここでは，計算負荷に対しメモリアクセス負荷の高い近傍 7 点ステンシル計算の結果を示す．

3.1 単純ステンシル計算（空間ブロッキング）の性能

図 8 (a) (b)に 2 ノード（256GiB 問題）から 72 ノード（9.2TiB 問題）までの単純ステンシル計算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（preload なし）と sms_preload_array 関数を用いて事前にフェッチした場合との違いを示す．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ことがわかる．

この主な原因を調査するために，preload 利用の有無のそれぞれの場合の，ステンシル計算における時間ステップ毎のバリア待ち時間総量（rank0 の値を利用）の成分を調査した．（図 8 (c) (d)）．バリア時間はノード毎に毎ステップ，異なるが，全体処理時間に占める rank0 の総バリア待ち時間の割合は一つの目安となる．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 により，計算部分の高速化が進んでいるかというと，図 8 (c) (d)の比較からわかるように，総計算時間成分の差は比較的小さく，高速化は見られるものの，全体の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とがわかる．この原因は，preload を用いない場合，毎回の各ノードの処理時間がノードによって非常にばらつき（すなわち segv によるページフェッチの待ち時間が一定でないと思われる），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．図 8 (c) (d)は，総計算時間から単純に rank0 のバリア時間を差し引いて示したものなので，全体でみると，各時間ステップでの rank0 の実行時間のばらつきは，相殺され，総計算時間成分は preload の有無で大きく差がないように見える．

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．

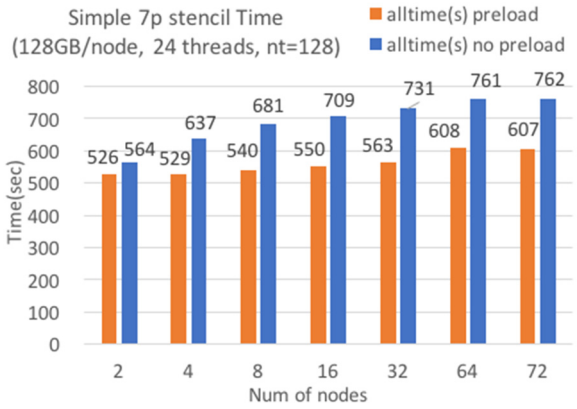

図 8 (a)　M-SMS 単純ステンシル計算　実行時間

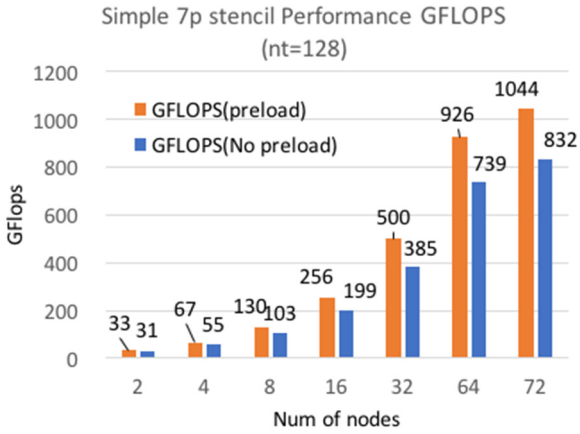

図 8 (b)　M-SMS 単純ステンシル計算　性能

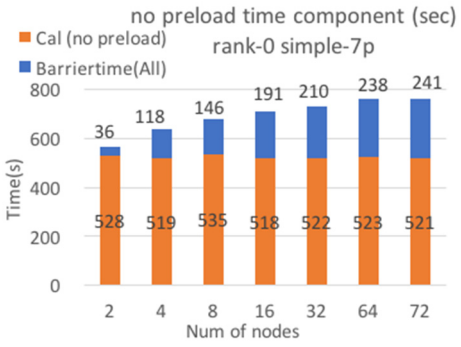

図 8 (c)　単純ステンシル preload なし　時間成分

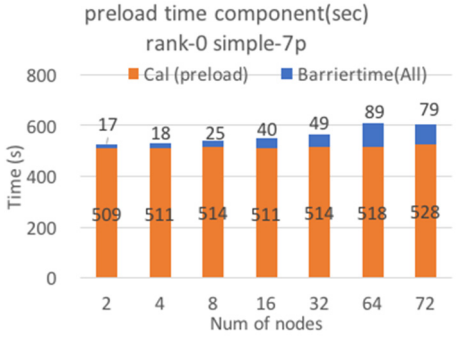

図 8(d)　単純ステンシル preload あり　時間成分

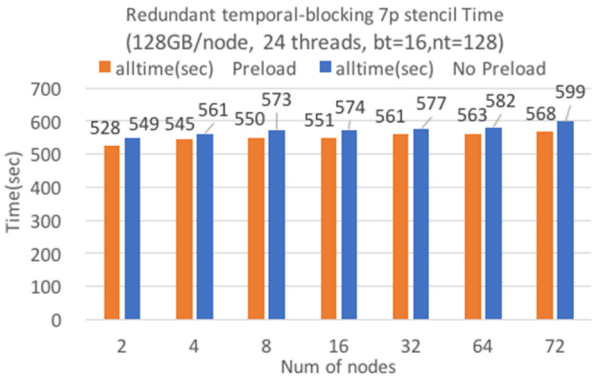

図 9 (a)　M-SMS 時間ブロッキングステンシル計算　実行時間

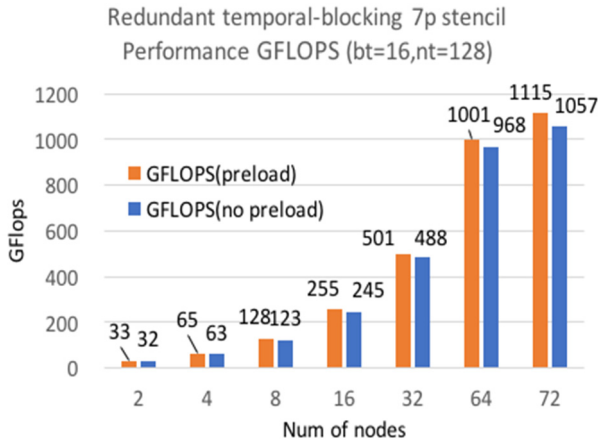

図 9 (b)　M-SMS 時間ブロッキングステンシル計算　性能

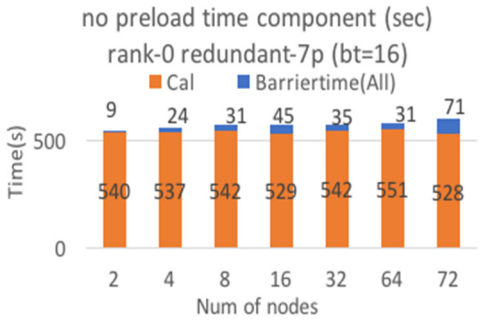

図 9 (c)　時間ブロッキングステンシル preload なし　時間成分

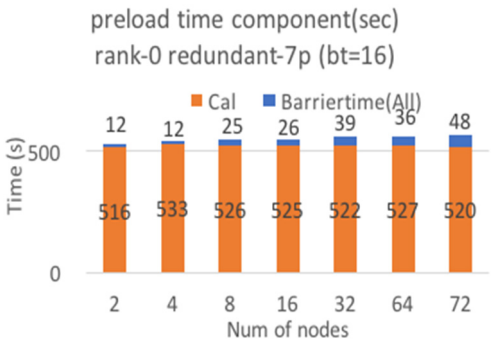

図 9 (d)　時間ブロッキングステンシル preload あり　時間成分

3.2 時間ブロッキングステンシル計算の性能

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．こちらは，時間ブロックサイズ(bt)が 16 の場合で，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．このため，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，以前 preload が有利である．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に比べるといずれも小さく，preoad の有無による短縮化率も小さい．

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る．

3.3 MPI プログラムとの性能比較

MPI プログラムと M-SMS 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，preload を用いると，M-SMS を用いた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った．

MPI も M-SMS 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，MP-SMS 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 措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，いずれも実行時間に影響はなかった．

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，M-SMS 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 M-SMS 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わかった．

今後，様々な通信サイズ，回数などを変えて，いつもこのような優位性が M-SMS にあるのかについては，検証していく予定である．

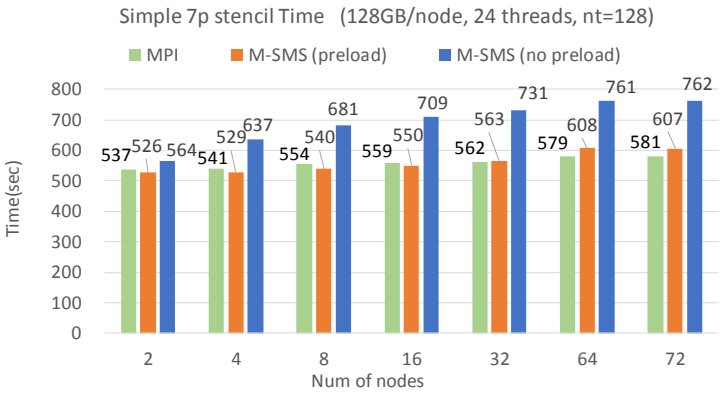

図 10 (a)　M-SMS と MPI との比較　単純ステンシル計算　実行時間

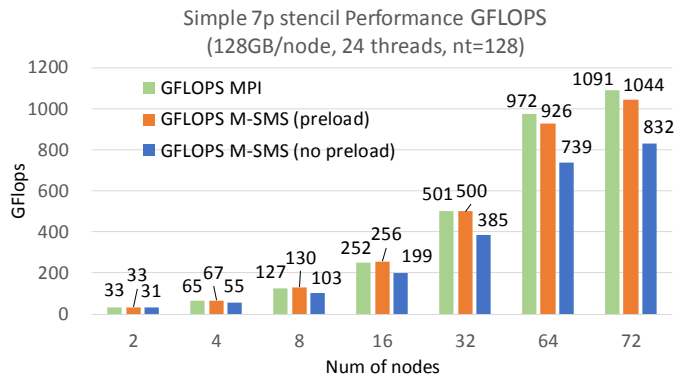

図 10 (b)　M-SMS と MPI との比較　単純ステンシル計算　性能

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，新たな機能を設計，実装し，複数スレッドを用いた通信，事前遠隔データフェッチ preload などを導入した．また，初期性能評価実験として，ステンシル計算を例に，preload の有効性を示した． 72 ノードという多数ノードを用いて，大規模データ（9.2TiB）を分散配置し容易にステンシル計算を記述して実行できることを示した．4 ノード利用時（両側近傍ノードとの通信が発生する最小ノード数）の性能と比べても，72 ノードの並列処理効率は 96.2%と優れている．

さらに，単純ステンシル計算では，MPI プログラムと比較しても，preload を利用した M-SMS の優位性が明らかになった．今回は，簡単化のため， z 方向分割という単純なデータ分割を用いたが，さらに細かい多数のデータ転送が発生する yz 方向への分割による性能も調査する予定である．

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