Vol.2017-HPC-160 No.36
2017/7/28

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

緑川 博子[†1]　北川健司[†2]　大浦 陽[†1]

概要：一つのコンピュータに搭載できる主メモリサイズには制限があるが，このサイズを超えるような大規模なデータを処理したいという要求が存在する．分散メモリ型の MPI プログラムへの変換が容易であれば，クラスタを用いて大規模な分散メモリを利用できるが，もともと共有メモリモデルで設計されたアルゴリズムや，既存のプログラム・ライブラリの中には，このような変換が難しい，あるいは変換は可能であっても，時間的，人的コストが高く容易ではない場合がある．これを解決する一つの選択肢として，ネットワーク接続された複数ノードの遠隔メモリを用いて大容量仮想メモリとして提供するシステム DLM が開発されている．本報告では，マルチスレッドユーザプログラムにおける非同期な遠隔データアクセス要求に対し，効率的に遠隔ページングを行うためのページ交換プロトコルを提案した．マイクロベンチマークと 4 種の応用プログラムによる評価の結果，従来プロトコルに対し，高性能化できるプロトコルを示した．

キーワード：遠隔メモリ，ページング，メモリサーバ，アウトオブコア，out-of-core，ページスワップ，スワッププロトコル，主メモリ拡張，分散メモリ，仮想メモリ，大規模メモリ

1. はじめに

近年，一つのコンピュータに搭載できる主メモリサイズは，以前に比べ大容量（たとえば 512GiB）になってきているが，ユーザが実際に利用できるシステムの搭載メモリ容量が様々な要因により制限されている場合，主メモリ容量を超えるような大規模サイズのデータを処理したいという要求が存在する．対象とする処理が，容易に分散メモリ型の並列処理（MPI など）に変換できる場合には，クラスタを利用して複数ノードの計算資源（CPU）と記憶資源(メモリ)の両方を得て高速化できる．一方，もともと共有メモリモデルで設計されたアルゴリズムや，既存のプログラム・ライブラリには変換することが難しい，あるいは変換は可能であっても，時間的，人的コストが高く容易ではない場合もある．この状況を解決する一つの選択肢として，高速ネットワークで接続された複数ノードの記憶資源（メモリ）を，遠隔ページングを用いて大容量仮想メモリとして利用するシステムが考えられる．遠隔メモリを用いると，主メモリだけを用いた場合に比べ処理時間は多少長くなるものの，これまで主メモリ容量不足により実行不可能であったプログラムを，余分な開発コストをかけずに，大規模サイズの問題に容易に適用して実行できる。

分散大容量メモリシステム DLM（Distributed Large Memory）[4-9]は，このような遠隔メモリ利用による主メモリサイズを超える処理（out-of-core 処理）を可能にするシステムである．DLM は，動的なスレッド生成や消滅を伴う OpenMP や pthread プログラムにも対応している[9]．DLM を利用するためのユーザプログラムの変更はほとんど不要で，dlm_startup(), dlm_alloc(), dlm_shutdown()の付加のみ，もしくは，専用 C トランスレータ利用する場合には[7]，遠隔データの変数宣言に dlm を付加するだけで利用できる．また，スレッド実装された汎用ライブラリ関数などもそのまま使用できる．DLM 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．

DLM では，ユーザデータが，計算ノードの主メモリにはいりきらない場合，あらかじめユーザがファイルで指定したメモリサーバノードのメモリに自動的に割り当てていく．ユーザプログラムは計算ノードで実行するが，計算ノードにないデータにユーザプログラムがアクセスすると，SEGV ハンドラが起動し，遠隔のメモリサーバから該当ページを取得し，代わりに計算ノードメモリにあるページをスワップアウトして交換する．ページ交換は，DLM 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，DLM ページ表により，どのページをどのメモリサーバが保持しているかを管理している．

本報告では, DLM における遠隔メモリサーバとのページ交換をより高性能化するため、ページ交換の実装上の改良を行った. ノード間のページ通信方式は, DLM だけでなく，ユーザレベル分散共有メモリシステムや PGAS モデルの実装基盤におけるデータ転送手法の予備評価としても意味があると考えている．現在の DLM は，汎用性，移植性の観点から，ノード間通信に広く利用されている MPI を用いている．MPI のマルチスレッドによる利用に関しては，未だ性能上，実装上，十分とは言えない状況もあるが，現時点での MPI 実装状況に対応しつつ，いくつかのプロトコルを提案，実装，評価し，問題点を明らかにして，従来手法を改良したので，報告する．

2. マルチスレッド対応 DLM システムの概要

図 1 は，マルチスレッドプログラムに対応した DLM システム DLM [9] (mDLM)におけるページ交換プロトコルの

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

概要を示す．DLM 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，遠隔データにアクセスするという制限を設けていない．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，メモリサーバ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，オーバーヘッドが高いと危惧されたが，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性能低下であることがわかった[9]．

現在の mDLM 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１の内部要求キューに登録され，起動時に自動生成された DLM 通信スレッドが内部要求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．ページ要求であれば，該当するページを持つメモリサーバにページ要求を送信，該当ページの受信，全スレッドのサスペンド，アドレス空間へのページの貼り付け，スワップアウトページの送信，ユーザスレッドの再開を行う．しかし，ユーザスレッド数が多くなると，これら一連の処理を単一の通信スレッドのみが逐次的に処理することは，性能上のボトルネックになると考えられる．このため，通信スレッドに加え，ページ受信専用の受信スレッドを増設することで，ページ交換の効率化を図ることとした．

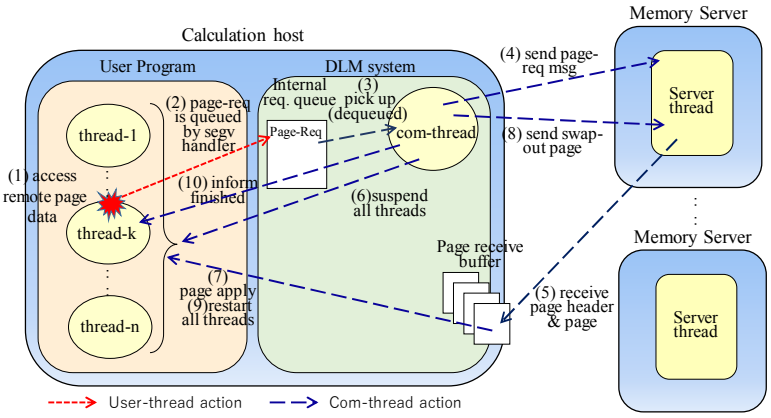

図 1　mDLM の従来のページ交換プロトコル r3

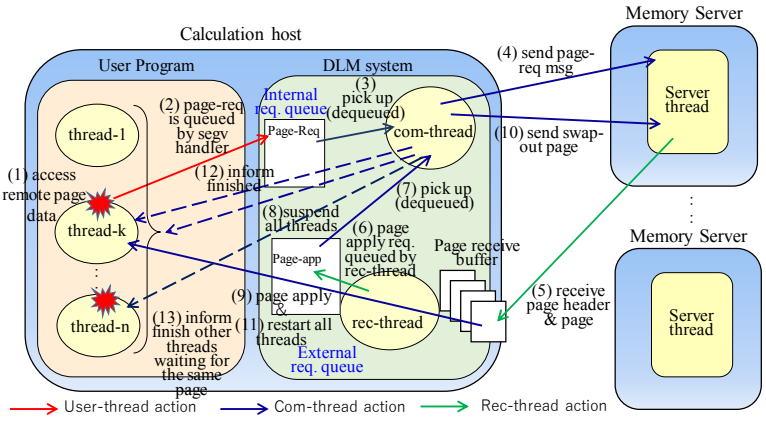

図 1　受信スレッド増設改良プロトコル r58

3. 改良型ページ交換プロトコル

レシーバスレッドを増設し，ページ交換の効率化を図るため，ここでは２つの改良プロトコルを提案，実装，評価した．

3.1 受信スレッド増設プロトコル r58

第一の改良プロトコル(r58 と呼ぶ)は，増設した受信スレッドにこれまで通信スレッドの担ってきた受信処理部分を任せ，通信スレッドの負担を軽減する．しかし，全体の制御，ページ表やユーザスレッド ID などの管理は，通信スレッドのみに行わせるというものである．DLM ページ表などの内部データを複数のシステムスレッドで扱うには，排他制御が必要になるため，単一スレッド（通信スレッド）による管理を優先した実装である．

このプロトコルは，マルチスレッド対応分散共有メモリシステム Multi-SMS[10]で実装したプロトコルを流用している．分散共有メモリでは，各ノードが，DLM におけるメモリサーバであり，かつ計算ノードであるため，他のノードから非同期に来るページ要求などに対応するため受信専用スレッドを作ることが MPI を使う上で必須であった．このページ交換プロトコル r58 の手順を図２に示す．

このプロトコルでは，ユーザスレッドからの要求（メモリ割り付けやページ要求など）を入れる内部要求キューとメモリサーバからの送信に対する処理要求を入れる外部要求キューの２つがある．全体を制御する通信スレッドは，この二つの要求キューを交互に見て，要求を処理する．一方，受信スレッドは，メモリサーバからの送信（メッセージヘッダーやページ本体）を受け取り，ヘッダーを外部要求キューへ入れ，ページ本体は，ページ受信バッファに受け取る．受け取ったページのユーザアドレス空間への貼り付け，スワップアウトページの送信などは，すべて通信スレッドが行う．

改良プロトコル r58 のページ交換手順を以下に示す．(番号)は図 2 の番号に対応している．

(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内部要求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，内部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 DLM 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) メモリノードは受け取ったページ要求から該当するページを特定し，メッセージヘッダーを付加して計算ノードへ送る．

(6) 受信スレッドがページ受信バッファにページを受け取り，ページ割付要求を外部要求キューへ入れる．

(7) 通信スレッドが外部要求キューからページ割付要求を取り出す

(8) 通信スレッドは，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．

(9) 受信バッファにあるページをユーザデータ領域にコピーする．

(10) 通信スレッドが DLM ページ表を参照し，計算ノードの保持するページのうち，一つをスワップアウトページとしてメモリノードへ送信する．

(11) 通信スレッドが，(8)で一時停止させたユーザスレッドを再開させる．

(12) 通信スレッドが，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．

(13) 通信スレッドは，(1)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場合には，このスレッドも起こす．

3.2 受信スレッド増設プロトコル r58 の性能評価

改良型プロトコルを評価するために，図 3 のような簡単なマイクロベンチマークプログラムを用いた．プログラムは，主メモリサイズを超えるサイズの配列を割り付け，配列全体を初期化（値の書き込み）後，各 DLM ページの先頭に対応する配列要素を離散的に読み出しアクセスし，内容をチェックするプログラムである．これを OpenMP により複数スレッドで並列に行う．初期化は配列要素すべてに対する連続書き込みアクセスとなるが，読み出しは，DLM ページ（ここでは 1MB）毎に 1 要素（double）を読み出すため，非常に頻繁なページ要求が全ユーザスレッドから発生する過酷なプログラムとなる．

この計測では搭載メモリ 64GiB 計算ノード(表 1 の crest4)を用い，DLM によるローカルメモリ利用上限を 800MB に制限し，メモリサーバ 1 台を用いて行った．本報告における評価実験で用いたサーバの仕様は表 1 に示す．

```
#define ENUM ((unsigned long ) (1L<<28))   //256M Elements
int main( )
{
  dlm_startup(&argc, &argv);
 // 2GiB array allocation
  array = (unsigned long *) dlm_alloc(sizeof(unsigned long) * ENUM);
// Initialization, array parallel write
#pragma omp parallel for
  for ( i = 0; i < ENUM; i++ )  array[i] = i;
// Array  parallel read per 1MB DLM page
#pragma omp parallel for
  for ( i = 0;  i < ENUM; i+=(1L<<17) )  // data read per 1MB
     if (array[i] != i )  return 1;
  dlm_shutdown();
}
```

図 3　評価用マイクロベンチマークプログラム

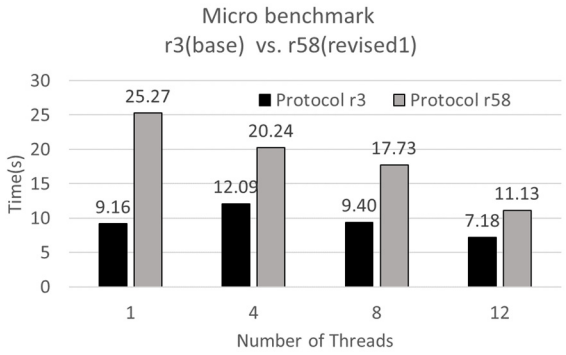

図 4　マイクロベンチ実行時間（従来 r3 と改良 r58）

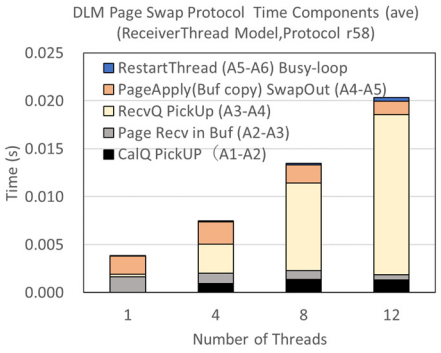

図 5　改良プロトコル r58 のページ交換の平均時間成分

図 4 は，このプログラムの実行時間（スレッド数 1－12）を，図 1 の従来の通信スレッドのみのプロトコル（r3 と呼ぶ）と受信スレッド増設改良プロトコル r58 で比較したものである．結果としては，従来プロトコル r3 に対し，改良プロトコル r58 は，優位性がないばかりか，性能が劣化する結果となった．

この原因を調査するために，r58 プロトコルにおいて，1 つのページ要求においてどの部分で時間がかかっているのかを，上記プログラム実行時の全パケットについてパケット毎に計算ノードで時間を計測し，各処理部分に要した最大，最小，平均時間を調査した．図 5 に，ユーザスレッド

による segv ハンドラの起動から，ページの獲得，ハンドラからのリターンまでに至るまでの各過程（A1-A6）の時間成分の平均を示す．この結果，メモリサーバからページを受け取った後（A3），受信スレッドがページ貼り付け要求を図２に示す外部要求キューに登録してから，それを通信スレッドが取り出す(A4)までの外部キュー滞在時間(A3-A4)が，ユーザスレッド数増加とともに，非常に増大していることがわかった．同様に，ユーザスレッドが SEGV を起こし，ページ要求を内部キューに登録（A1）してから，この要求が通信スレッドによって取り出されるまで（A2）の内部キュー滞在時間（A1-A2）も一定量，かかっている．すなわち，両方の要求キュー内で待たされる時間が問題となっている．

r58 では，内部と外部の要求を公平に処理するため，通信スレッドは，内外２つのキューから交互に要求を取り出して処理していたが，実際のプログラム実行では，最初の段階で，多くのページ要求が複数のユーザスレッドから内部要求キューに登録され，一方，外部要求キューにはまだほとんど要求がない状態となる．しかし，内部要求キューにある多くのページ要求をメモリサーバに出していくと，今度は，多くのページが一斉にメモリサーバから送信され，ページの貼り付け要求が外部要求キューに溜まる状態になる．この時，ほとんどすべてのユーザスレッドは自分の要求したページを待っている状態で，segv ハンドラ内で停止しており，新たな内部要求は起こらず，内部要求キューは空に近くなる．

このような変動を解決するため，キューに存在する要求数に応じて，通信スレッドの各キューの処理優先度を制御するなど，幾つかの処理スケジュール手法を試したが，プログラム実行中の内部要求と外部要求の偏りや変動を十分に制御することが難しいということがわかった．

3.3 ページ交換専任型プロトコル r77

第二のプロトコル(r77 と呼ぶ)は，第一のプロトコル(r58)の問題点であった外部要求キューの要求処理待ち時間の解消を図るため，DLM ページ表などのユーザスレッド ID 管理表などの DLM 内部データへのアクセスを受信スレッドにも許し，アプリケーション実行時に最も性能上のネックになるページ交換を受信スレッドが独立して行う方式とした．図６に r77 のページ交換プロトコルを示す．受信スレッドは，メモリサーバからページヘッダーを受信すると，ユーザスレッドのサスペンド，ページの貼り付け（これまでのページ受信バッファを廃止し，直接ユーザアドレス空間のデータ領域にページを受け取る），ページを送ってきたメモリサーバへのスワップページの送信，ユーザスレッドの再開など，r58 で通信スレッドが行っていた一連の処理をすべて行う．それ以外の外部要求処理（メモリ割り付けに対するメモリサーバの応答など）は，従来通り，外部要求キュー経由で通信スレッドが行う．このプロトコルでは，受信スレッドが，スワップアウトページの「送信」も担っており，名前に反し「受信のみを行うスレッド」ではなくなっている．

改良プロトコル r77 のページ交換手順を以下に示す．（番号）は図 6 の番号に対応している．

(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内部要求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，内部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 DLM 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) メモリノードは受け取ったページ要求から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 DLM ページ表を参照し，計算ノードの保持するページのうちの一つをスワップアウトページとしてメモリノードへ送信する．

(9) 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．

(10) 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

(11) 受信スレッドが，(1)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場合には，このスレッドを起こす．

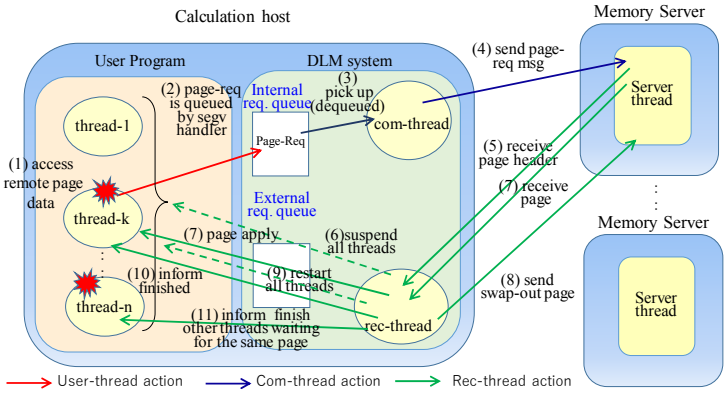

図 6　ページ交換専任型プロトコル r77

3.4 ページ交換専任型プロトコル r77 の性能評価

r77 プロトコルも，前述のマイクロベンチマークによる同様の評価を行った. 従来プロトコル r3 との比較を, 図 7, 図 8 に示す. 図 7 は全体実行時間であるが，従来型 r3 に比べ，r77 の実行時間は 57%~77%に短縮している．図 8 は，１つのページ交換にかかる各部分の時間成分の全パケットの平均を表している．r58（図５）で大部分を占めた外部要求キュー滞在時間（A3-A4）は，プロトコル上，なくなっており，内部キュー滞在時間（A1-A2）も非常に小さくなっていることがわかる．すなわち，ユーザスレッドの出したページ要求は即座に通信スレッドによりメモリサーバへ送られる．

一方で, r77 では, 通信スレッドがページ要求をメモリサーバに出してから，受信スレッドがメモリサーバからの応答（ページヘッダー）を受け取るまでの時間（A2-A3）が，r58 に比べ, 長くなっている. 通信スレッドがメモリサーバに送るページ要求は短いメッセージであるため，MPI の実装上，多くの場合，メモリサーバの受信との同期を待たずに次々と送られてしまう．しかし，実際には，メモリサーバはそれ以前に送られてきたページ要求に対するページ交換を受信スレッドとの間で行っており，次々と通信スレッドが送ってくるページ要求の実際の受信を未だおこなっていないと考えられる．このため，見かけ上，ページ要求を出してからページ受信までの時間(平均)が長くかかっているように見えるが，実際の計測値は最大，最小の振れ幅が大きく，この値が通信路とメモリサーバの性能を示すものではない．メモリサーバが多数台ある場合には，ページ要求が分散され，各メモリサーバが即座に通信スレッドからのページ要求に応答してページを送ることができると考えられるが，この場合にも，一つしかない受信スレッドがページ交換のネックになると考えられる．

4. 応用における改良プロトコルの効果

前節のマイクロベンチマークにおける各プロトコルの性能が，他の応用プログラムに対し，同様な効果をもたらすのかを調べることにした．ここでは，以下の 4 種のプログラムについて，プロトコルの違いによる性能差を調査した．用いたのは，(1) メモリアクセバンド幅計測用の stream ベンチマーク[1]，(2)テンポラブロッキングによる 3 次元 7 点ステンシル処理[2], (3)正方行列掛け算，(4)マルチスレッド実装の汎用ライブラリの使用例として fftw ライブラリ[3]による 3 次元フーリエ変換である．いずれもユーザプログラムは OpenMP によるマルチスレッドプロ

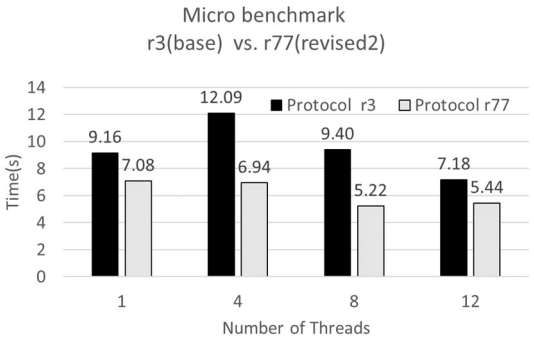

図７　マイクロベンチ実行時間（従来 r3 と改良 r77）

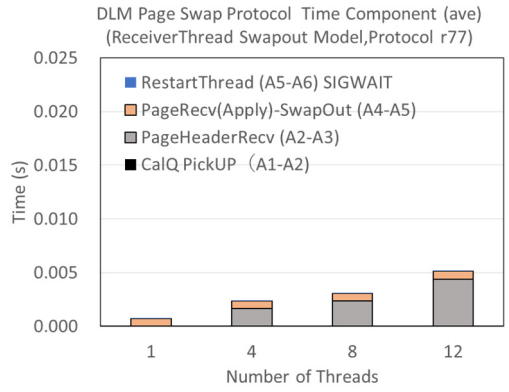

図８　改良プロトコル r77 のページ交換平均時間成分

グラムである． 計測環境は，表 1 に示す複数のサーバを用いた．搭載メモリと CPU は異なるが，ソフトウエア環境は同一である．

4.1 Stream ベンチマーク

遠隔メモリアクセスバンド幅を調査するため，Stream ベンチマーク[1]の性能を計測した．Stream で用いる３つの 1 次元配列（A, B, C）の合計サイズが 48GiB から 512GiB の場合の処理バンド幅（メモリアクセス性能）を調べた．実験には，搭載メモリ 64GiB の計算ノード(crest4)と，搭載メモ

Server Name	Sandy Bridge Server crest2,3,4	Haswell Server crest5,6,7,8	Broadwell Server crest9,10
Network	Infiniband single FDRx4 (56Gbps)		
Node CPU	Xeon E5-2687W (3.10GHz) 2 CPU x 8 Core	Xeon E5-2687W v3 (3.10GHz) 2 CPU x 10 Core	Xeon E5-2687W v4 (3.00GHz) 2 CPU x 12 Core
Node Memory	DDR3-1600, crest4: 8GiBx8 (64 GiB) crest3: 16GiB x8 or x12 (128 /196 GiB), crest2: 8GiB x 16 (128 GiB)	DDR4-2133 16GiB x 8 (128 GiB)	DDR4-2400 16GiB x 8 (128 GiB)
L1 cache	32 KiB	32 KiB	32 KiB
L2 cache	256 KiB	256 KiB	256 KiB
L3 cache	20 MiB	25 MiB	30 MiB
OS	CentOS 7.1.1503 (x86_64), kernel 3.19.5	CentOS 7.1.1503 (x86_64), kernel 3.19.5	CentOS 7.2 (x86_64), kernel 3.19.5
Compiler	gcc version 4.8.3 -O3	gcc version 4.8.3 -O3	gcc version 4.8.3 -O3
MPI	MVAPICH2 2.0.1_mrail_OFED2.4-1.0.4		

表１　性能計測に用いたサーバ

リ 128GiB のメモリサーバ 4 台を用いた．DLM が利用するメモリ容量の上限は，計算ノード 56GiB，メモリサーバ 120GiB に設定している．逐次アクセスのため，DLM ページサイズは 8MB を用いた．計算ノード（crest4）は 16 コアあるが，stream には 14 スレッド（OpenMP）を用いている．（DLM と MPI のシステムスレッドのために余裕をもたせている．）図 9 は，r77 プロトコルを用いた場合の試行回数の平均（最大ではない）処理バンド幅（縦軸は対数）を示す．左端の 48GiB データサイズの場合，遠隔メモリをアクセスせずに，100%ローカルメモリにアクセスしているので，計算ノード DRAM の性能を反映する（DDR3 8GiB x 8）．4 つの操作（COPY, ADD, SCALE, TRIAD）の平均は 45GiB/s（14 スレッド利用）に達している．

図 10, 11, 12 はプロトコル r3，r58，r77 のそれぞれにおいて，遠隔メモリの利用割合が多い場合（総データサイズに対するローカルメモリサイズの割合が 58%(96GiB)～11%(512GiB)）の性能を示す． 遠隔メモリの割合が一定以上になると，プロトコルやネットワークのバンド幅によって，性能がほぼ一定となる．512GiB サイズ（ローカルメモリ率 11%）の 4 操作の平均バンド幅（試行回数の平均で）は，r3 プロトコルで 976MB/s，r58 プロ後コルでは 501MB/s，プロトコル r77 では 1567MB/s となっている．新たに提案するプロトコル r77 は，従来プロトコル r3 の約 1.5 倍のバンド幅が得られる．

多くの応用プログラムでは，主メモリ（DRAM）とキャッシュを意識して，データアクセス局所性を高めたアルゴリズムを用いる．このため，この遠隔メモリとローカルメモリのそれぞれのメモリバンド幅の比がそのまま，応用プログラム実行時の性能比になるわけではない．

4.2 テンポラルブロッキングステンシル計算

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 3 次元データに対する近傍 7 点ステンシル計算において，遠隔メモリを用いた out-of-core 処理を行った結果を調査した．このアルゴリズムでは，２つのデータドメイン 3 次元格子データのうち，それぞれ一部の小ブロック（空間ブロックサイズ分）をローカルメモリにある小ブロックにコピーして，時間ブロックサイズ分の時間ステップのデータ更新を行い，結果を，元の全体 3 次元格子データに書き戻すアルゴリズム[2]になっている．このため，全体の大きな 3 次元格子データは dlm_alloc を用いて確保し（すなわち遠隔メモリにも展開するデータとなる），ローカルメモリにおける時間ステップ分の更新を行う小ブロック（64GiB）は通常の malloc を用いてローカルメモリに確保する手法をとっている．ただし，搭載主メモリのほとんどを小ブロックの割り当てに使うと，遠隔メモリのページキャッシュ領域がなくなるので，ここでは，小ブロックサイズを 32GiB 固定とし，256 ステップの更新処理のうち 128

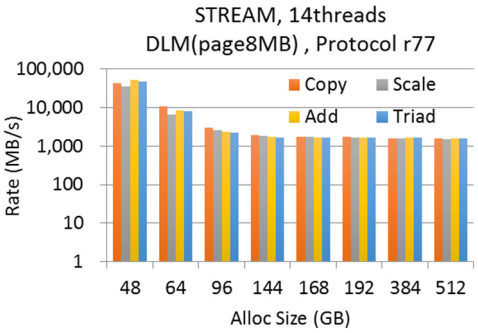

図 9　プロトコル r77 利用時の stream 性能

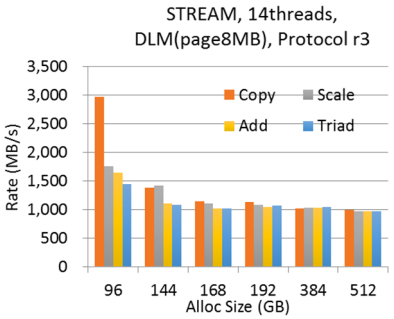

図 10　プロトコル r3 利用時の stream 性能

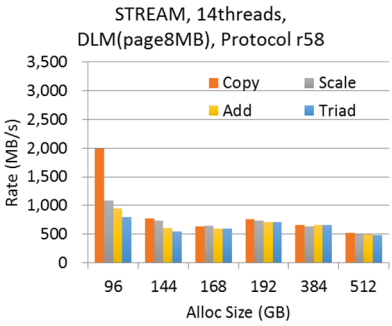

図 11　プロトコル r58 利用時の stream 性能

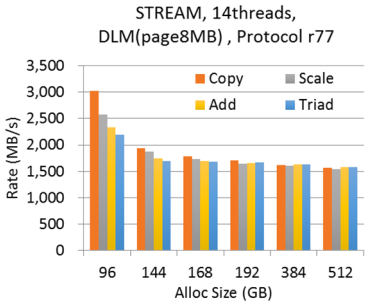

図 12　プロトコル r77 利用時の stream 性能

時間ステップを時間ブロックサイズとしている．（ここでは，搭載主メモリサイズに最適化した時間・空間ブロックサイズを採用しているわけではない．）問題サイズは 64GiB～512GiB で，128GiB のメモリを搭載したサーバを 5 台用い，うち 1 台を計算ノード(crest7)として使用した．DLM によるメモリ使用の上限は各ノード 120GiB に設定した．すなわち，計算ノード搭載メモリの約半分(64GiB)を空間・時間小ブロック用のローカルデータ領域として用い，残りの

56GiB 程度を DLM 遠隔ページのキャッシュエリアとして利用する．

３つのプロトコルを用いた場合の実行時間と処理性能(Effective Mflops)の比較を図 13 と図 14 に示す．左端の 64GiB 問題は，サイズが小さいので，事実上ローカルメモリのみにアクセスしてる．右端の 512GiB 問題は，計算ノード搭載メモリサイズの 4 倍サイズの問題で，遠隔メモリを利用して処理している．この時の処理性能は，ローカルメモリのみを使用した 64GiB 問題の処理性能の 68%~78%になっている．ただし，この値は，テンポラルブロッキングステンシルアルゴリズムの時間・空間ブロッキングサイズの最適化を行うとさらに小さくすることができる．図 13. 図 14 とも，問題サイズが大きくなるに従い，プロトコルの差は広がる．

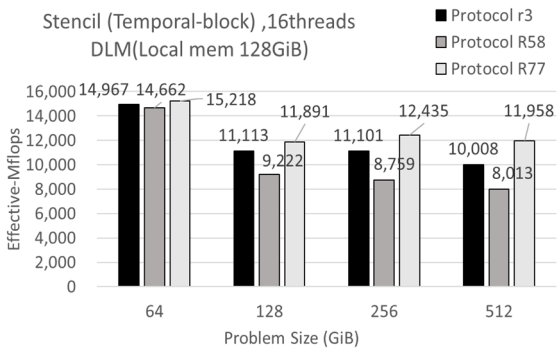

図 13　各プロトコル利用時の stencil 計算の性能

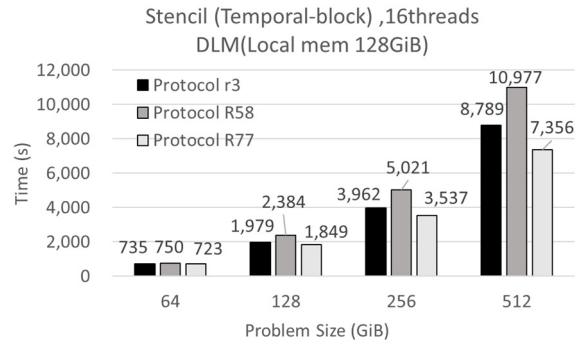

図 14　各プロトコル利用時の stencil 計算の実行時間

4.3 行列積

正方行列積の DLM 利用時の性能は，計測時間短縮のため，計算ノードにおける DLM のメモリ利用上限を問題サイズより小さく制限することにより，遠隔メモリの利用の割合を増やして計測した．搭載メモリ 128GiB の計算ノード(crest9)とメモリサーバ 1 台を用いているため，実際には実メモリサイズよりも小さい問題サイズでの計測となる．行列積（C=A*B）は，アクセス局所性を高めるためにブロッキングを行い（3 重ループを 6 重ループへ），行列 B は転置している．ローカルメモリ 100%利用時(通常実行)のにおいて，複数方式の性能を事前に調査し，最も効果のあった最外側ループのスレッド並列により，各スレッドの処理ブロックサイズがコア内 L2 キャッシュサイズに収まる方式を採用している．

図 15 は, 32K×32K の正方行列積(データサイズ 24GiB)を，ローカルメモリサイズ 12GiB～3GiB に制限した時の実行時間を示す．ローカルメモリ率（利用ローカルメモリサイズ/問題サイズ）12%（3GiB 利用時）の場合，改良プロトコル r77 は，従来プロトコル r3 に比べ，12%程度実行時間が短縮している．プロトコル r58 は，長時間かかると予想されるため，計測していない．

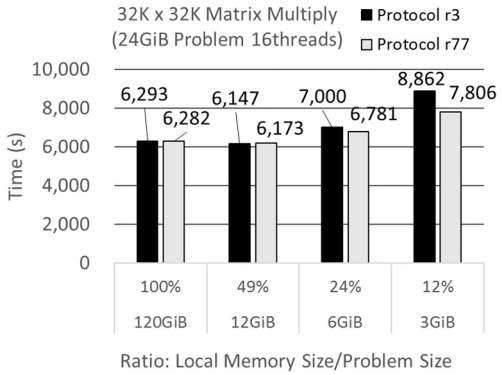

図 15　32K×32K 行列積（問題サイズ 24GiB）の実行時間

4.4 FFTW 利用による 3 次元フーリエ変換

FFTW ライブラリ（fftw-3.3.6-pl1 ）[3]を用いた 3 次元フーリエ変換プログラムの実行時間と性能を３つのプロトコルで計測した．FFTW は，OpenMP 版のライブラリ（libfftw3_omp.a）を利用している．２つの３次元 double 型配列を dlm_alloc でメモリに確保し, 3dfftw の関数の引数として渡し計算するプログラムである．ユーザは，fftw 関数内部のスレッド利用の詳細はブラックボックスのまま，out-of-core 処理に用いることができる．

問題データサイズは，64GiB~1TiB である．搭載メモリ 128GiB の計算ノード(crest2)を用い，最大 8 台のメモリサーバ（搭載メモリ 64GiB~192GiB）を利用し，計測した．各ノードの DLM メモリ利用上限は，計算ノードは 120GiB, メモリサーバは搭載メモリ容量に応じて 55GiB~180GiB とした．

図 16，17，18 に，３つのプロトコル r58，r3，r77 の実行時間と性能（Mflops）をそれぞれ示す．ユーザプログラムは 14 スレッドで実行している．プロトコル r58 の性能は，遠隔メモリを利用すると極端に低くなる．プロトコル r77 は従来プロトコル r3 に比べ，5%から 37%程度性能があがっている．プロトコル r3 の 64GiB 問題のときの性能を基準にした時の r77 の相対性能を図 19 に示す．64GiB 問題では遠隔メモリアクセスが行われていないにもかかわらず，プロトコルに性能差が出ている点が明らかにはなっていな

い．問題サイズごとの典型的なスワップ回数を r77 について図 20 に示す．ページ交換回数（スワップ数）はプロトコル毎に非常に大きな差があるわけではない．

fftw は，連続アドレスアクセスの多い他の 3 つの応用とは異なり，内部で複数スレッドによる離散的なデータアクセスが多量に生じていると考えられる．大規模サイズの fftw を複数スレッドで実行する場合，1 つのプロセスのユーザアドレス空間上の割り付け領域の最大数が，規定値の 64K 個（max_map_count，OS パラメタ）では足りない．このため，この実験では，256GiB から 1TiB の問題を実行する時，/proc/sys/vm/max_map_count のファイルに 70K~360K の値を設定して，領域数を大きくしている．

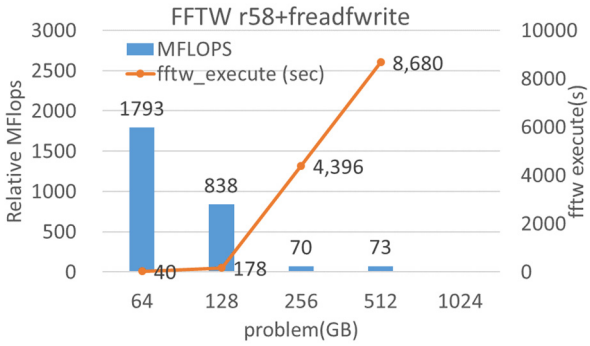

図 16　fftw 実行時間と性能　プロトコル r58

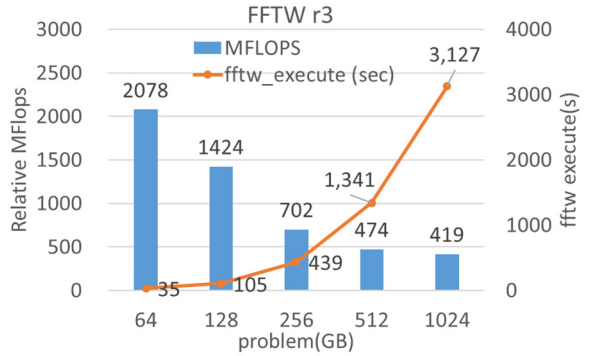

図 17　fftw 実行時間と性能　プロトコル r3

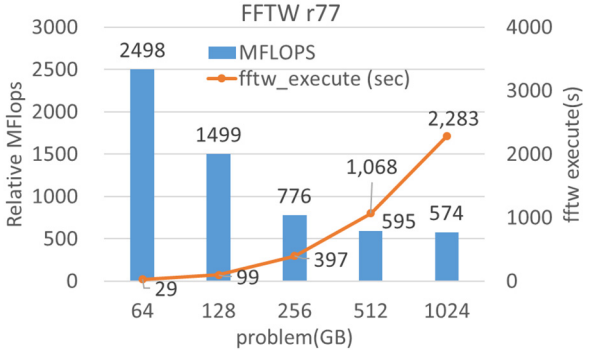

図 18　fftw 実行時間と性能　プロトコル r77

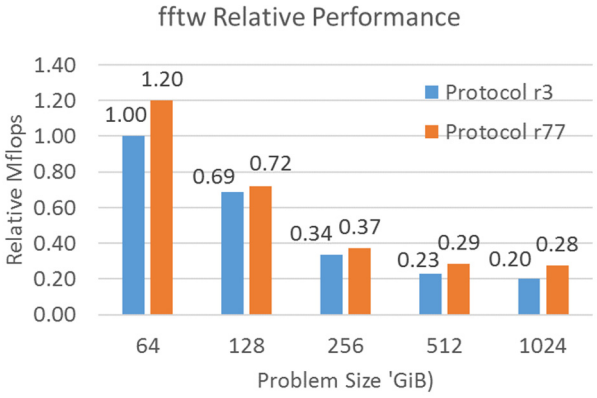

図 19　従来プロトコル r3 に対するプロトコル r77 の性能

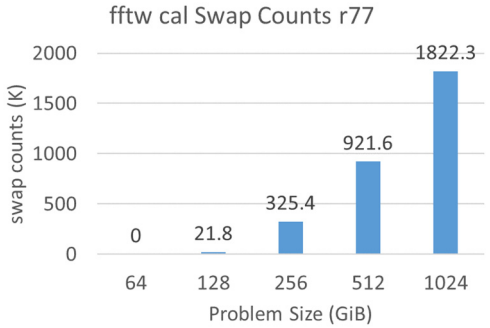

図 20　fftw 関数実行中のスワップ回数（プロトコル r77）

5. 終わりに

本報告では，マルチスレッドユーザプログラムにおける非同期な遠隔データアクセス要求に対し，効率的に遠隔ページングを行うためのページ交換プロトコルを提案し，マイクロベンチマークと 4 種のプログラムによる評価を行った．システムスレッドが 1 つ（通信スレッド）であった従来のプロトコルに対し，新たなシステムスレッド(受信スレッド)を導入し，メモリサーバへのページ要求とページ交換を独立，かつ並列的に行うことにより，効率的なページ交換プロトコルを実現できた．

ここで対象とした mDLM システムは，計算ノードとメモリサーバのメモリを同一に扱うフラットモデルである．フラットモデルは，サーバ台数が少なくても，効率よく大規模メモリを構成することができる利点がある．このモデルでは，ページの所在を固定しておらず，ページ交換の度にページは様々なメモリサーバへ移動する．フラットモデルの利点は，1 つのページ交換の際に対象となるメモリサーバが一つで，連続してスワップインとスワップアウトの送受信ができるため MPI のメッセージランデブーのために待たされる時間がほとんどない．このため，プロトコル r77 では，ページ交換は一つの受信スレッドが続けて行うようにしている．

一方，DLM には，メモリサーバ全体で提供する大規模メ

モリのキャッシュとして計算ノードのメモリを位置づける，階層モデル型の DLM システムがある．この階層モデルでは，各ページの所在は割り付け時に一つメモリサーバに固定されており，計算ノードでアクセスがあると，キャッシュページとして，計算ノードに転送され，スワップアウトされる際には，必ず所定のメモリサーバに返される．階層モデルでは，１つのページ交換に，通常，スワップインとスワップアウトの２つのメモリサーバがかかわる．これにより，スワップインページ（ページ要求）とスワップアウトページの２つのメッセージが独立に各メモリサーバに送られるため，複数メモリサーバによる並列性はあがるものの，２つの DLM システムスレッドが独立にページ要求とスワップアウトページの送信を行うと，メモリサーバ側の送受信の順序などにより，受信スレッドによるページ受信とページ送信（ページ交換は MPI 実装上，同期的送受信になる）の性能は影響をうける．このようなモデルにおける最適プロトコルは，今回，提案したものとは少し異なってくる．階層モデル型システムに関しても，複数のプロトコルを構築，評価しており，効率的プロトコルを今後，提案していく予定である．

参考文献

[1] Stream ベンチマーク　https://www.cs.virginia.edu/stream/
[2] H. Midorikawa, H.Tan, T.Endo,”An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations”, IEEE The 12th International Conference on High Performance Computing & Simulation, HPCS2014, pp.268-277, (2014.7)
[3] FFTW ライブラリ　http://www.fftw.org/
[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] 齋藤和広，緑川博子，甲斐宗徳："ユーザレベル実装遠隔メモリページングシステムにおけるページ置換アルゴリズムの評価"，情報処理学会、ハイパフォーマンス研究会 Vol.2010-HPC-125, No.9, pp.1-6, (2010, 6)
[7] S. Yoshimura, H.Midorikawa; "A C Compiler for Large Data Sequential Processing using Remote Memory", proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp.198-202 , (2011.8)
[8] H. Midorikawa, J.Uchiyama: " Automatic Adaptive Page-size Control for Remote Memory Paging", proc. of 13th IEEE/ACM International Symp. on Cluster, Cloud and the Grid Computing CCGrid2012, pp.694-696, (2012.5)
[9] 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)
[10] 緑川博子,岩井田匡俊："マルチスレッド対応型分散ン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)