B-005

遠隔メモリページングにおけるマルチスレッド利用によるスワッピングプロトコルの改良
Improvement of swapping protocol by using multi-thread in remote memory paging

大浦 陽† 緑川 博子† 北川 健司†2 甲斐 宗徳†
Hikari Oura Hiroko Midorikawa Kitagawa Kenji Kai Munenori

1 はじめに

大規模科学技術計算などの大量のメモリを用いる応用プログラムがあるが，ハードウェア制限や消費電力などの問題から 1 つのコンピューターに搭載できる主メモリの容量には限界がある．そこで高速ネットワークを用いて複数のコンピューターをクラスタ上で繋ぎ，遠隔メモリページングを行うことで，仮想的な大容量メモリを提供する分散大容量メモリシステム(DLM)を開発してきた[1].

DLM はマルチスレッドプログラムをサポート[2]しており，OpenMP や pthread で書かれたユーザープログラムから利用することができる．現在，大規模データを処理するために，データをノードに分散して MPI により並列処理することが広く行われている．しかし，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プログラム・ライブラリには，分散メモリ型の MPI 並列プログラムに変換することが困難な場合，あるいは変換は可能であっても人的，時間的コストが高くて容易ではない場合がある．このため，処理時間が多少，長くなっても，余分な開発コストをかけずに，これまで，主メモリ容量の制限で実行不可能であった既存プログラムを大規模サイズの問題に適用したいという要求がある．この状況を解決する一つの選択肢として，DLM が提供する遠隔メモリページングが有効である．

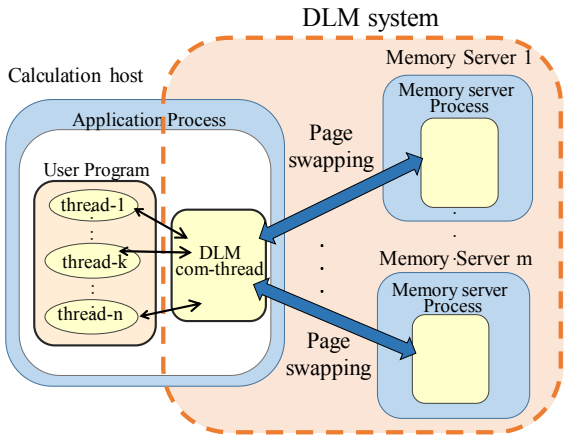

図 1.1 DLM の構成

DLM は 1 台の計算ノードと複数のメモリノードから構成されていて，計算ノードでユーザープログラムの実行を行い，メモリノードは計算ノードに収まらなかったデータを格納する．計算ノードはユーザースレッドとメモリノードとの通信を行う通信スレッドで構成されている．通信スレッドはユーザースレッドからの通信要求を要求キューによって管理する．計算ノードが保持していないデータにユーザープログラムがアクセスすると，Segv が発生する．これにより Segv ハンドラが起され，ユーザースレッドは内部要求キューにページ要求を入れ待つ．通信スレッドは内部要求キューからこのページ要求を取り出し，該当ページ保持するメモリノードを特定してページ要求を送信する．その後，メモリノードから該当ページを受け取り，計算ノード中にある他のページをメモリノードに送信してページ交換を行う．次に，全てのユーザースレッドを一時停止し，受信したページを該当するユーザーアドレス空間にコピーしてからユーザースレッドを再開させる．DLM の通信は独自に設定したページサイズ単位で行う．DLM の仮想的な大容量メモリは，DLM のアドレス空間をユーザーが宣言することで利用できる．DLM のアドレス空間は DLM ページ表によってページの格納場所などと共に管理される．

現在の DLM はマルチスレッドサポートだが，計算ノードとメモリノード間の通信を行う際，ユーザースレッドは通信スレッドに通信を依頼し，通信スレッドだけがメモリノードとの通信を行う．このため，通信待ち行列が生じてしまい，オーバーヘッドが非常に高くなっている．

本論文では，これまで 1 つのシステムスレッドに集中していた処理の負荷を軽減し，計算ノードとメモリノードの通信効率を向上させるための新たなページ交換プロトコルを提案する．2 つの通信プロトコルの設計と実装を行い，アプリケーションを用いた比較実験を行った．この結果，これまでのプロトコルの問題点を明らかにするとともに，新たに遠隔メモリページングにおける効率的なプロトコルを確立した．

2 受信専用スレッドを導入したスワッププロトコル

2.1 受信スレッド

計算ノードとメモリノード間の通信を行うスレッドは通信スレッドだけしかなく，通信スレッドがボトルネックになり，複数のユーザースレッドがページ要求を行ったとしても，内部要求キューを見て 1 件ずつ処理し，並列に処理することができなかった．この問題を解決するために通信スレッドの他に計算ノード内でメモリノードから送られてくるページや情報を受信する「受信スレッド」を DLM に取り入れた．

受信スレッドの役割は今まで通信スレッドが行っていた，メモリノードから送られてくるページやデータを受信し，外部要求キューに処理を入れる．通信スレッドが外部要求キューの中を見て，処理を行う．

† 成蹊大学理工学研究科理工学専攻 Graduate School of Science and Technology , Seikei University
†2 (株)アルファシステムズ Alpha Systems,Inc.

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

受信スレッドを導入することで計算ノードは各メモリノードへの要求とメモリノードからのデータの受信を同時に行えるようになり，ページ交換のオーバーヘッドが低下する．さらに 1 台のメモリノードへページ要求を出しながら他のメモリノードからページを受け取ることも可能になった．図 2.1 に受信スレッドを増設した DLM システムの概要を示す．

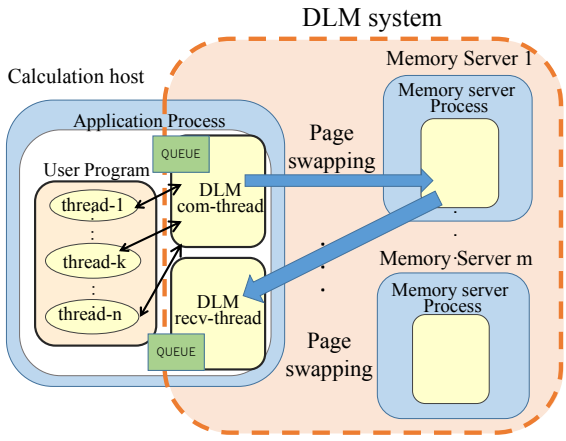

図 2.1 受信スレッドを含む DLM の構成

2.2 スワッププロトコル R58

受信スレッドを取り入れたスワッププロトコル R58 を以下に示す．改良を取り入れる前の DLM をスワッププロトコル R3 と表記する．

(1)ユーザースレッドは Segv を起こすとハンドラ内で内部要求キューに必要なページ番号を入れる

(2)通信スレッドが内部要求キューを見て，各メモリノードへページ要求を送信する

(3)メモリノードは受け取ったページ要求を処理し，計算ノードへページを送信する

(4)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

(5)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し，全てのユーザースレッドを一時停止させページをユーザーデータ領域にコピーする

(6)通信スレッドが外部要求キューを見て，計算ノードの保持するページの中，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

(7) ユーザースレッドを再開させる

2.3 プロトコル R58 のマイクロベンチマーク性能評価

受信スレッド導入によるスワッププロトコル R58 の性能を評価するため，図 2.2 のようなマイクロベンチマークを作成し，性能評価実験を行った．マイクロベンチマークは 2GiB の配列を DLM 領域に確保し，配列の全ての要素を要素番号で初期化した後，ページサイズ単位で配列にアクセスし，数値が正しく初期化されているかを確認する．すなわち，複数スレッドによる連続逐次書き込みに続き，DLM ページ単位の離散的な読み出しを行う．特に後半の読み出しは，複数スレッドによる DLM ページの要求が多量に発生する過酷なテストとなっている．

```
#define ENUM ((unsigned long int) (1L<<28))
int main(int argc, char *argv[]){
dlm_startup(&argc, &argv);
array = (unsigned long int *) dlm_alloc(
        sizeof(unsigned long int) * ENUM); //2GB alloc
#pragma omp parallel for
  for (i = 0; i < ENUM; i++) {
        array[i] = i;
  }
#pragma omp parallel for
    for (i = 0; i < ENUM; i+=(1L<<17)) {    //data access per 1MB
     if (array[i] != i) {
        return 1;
        }
}
dlm_shutdown();
return 0;
}
```

図 2.2 マイクロベンチマーク

CPU	Intel® Xeon® CPU E5-2687W 0 @ 3.10GHz 2CPU × 8core/node
Memory	64GB/node
Cache	L2 256KiB L3 20MiB
Network	Infiniband FDR
OS	CentOS 7.1.1503 (Core) Linux 3.19.5
Compiler	gcc version 4.8.3
MPILib	MVAPICH2 version 2.0.1

表 2.1 動作実験環境 1

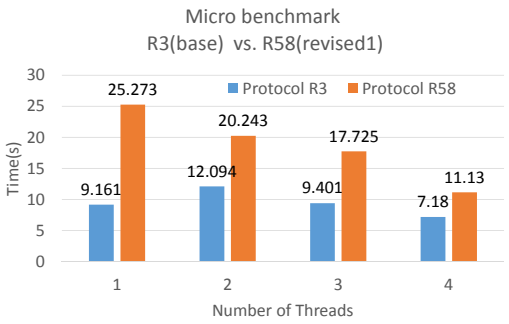

図 2.3 R58 と R3 のマイクロベンチマークの実行時間

動作実験環境を表 2.1 に示す．計算ノードの DLM 領域を 800MiB，メモリノードの DLM 領域を 6000MiB，メモリノード 1 台を使用し，ページサイズ 1MiB で実行した．

動作実験の結果を図 2.3 に示す．左棒が R3 の実行時間を表し，右棒が R58 の実行時間を表す．横軸が計算に使ったスレッド数を表し，縦軸は実行時間を表す．

スワッププロトコル R58 を導入すると，1 スレッドでの実行時間が改良前と比べ約 2.75 倍になることがわかった．実行時間に影響を与えるほどの遅延を発生させる場所を探るため，次に示す 5 つの計測個所を決め，プログラム中に

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

発生するすべてのページ要求に対し，各計測個所での平均を求めた．

A) Segv 発生直後から，通信スレッドが内部要求キューの中から呼び出すまで(A1-A2)

B) 通信スレッドがページ要求を出してから受信スレッドでページを受信するまで(A2-A3)

C) ページを受信し，外部要求キューへ入れてから，通信スレッドが呼び出すまで(A3-A4)

D) 通信スレッドがスワップページを送信してから，スレッドを再開させるまで(A4-A5)

E) スレッドが再開してから全てのページ交換処理が終了するまで(A5-A6)

図 2.4 はスワッププロトコル R58 における計測個所毎の

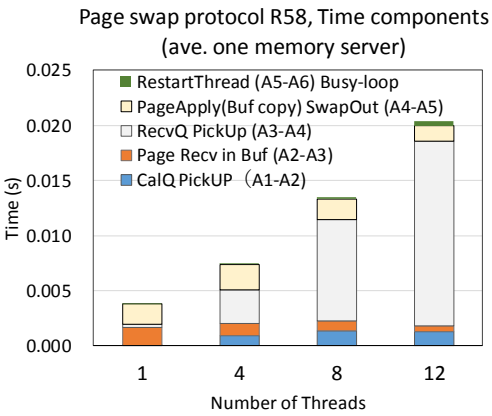

図 2.4 R58 における計測個所毎の実行時間

実行時間を実行スレッド毎にまとめたグラフである．横軸が計算に使ったスレッド数を表し，縦軸は実行時間を表す．

このグラフからスワッププロトコル R58 では，C)のメモリノードからページを受信し，外部要求キューに入れてから，通信スレッドに呼び出されるまでの時間が 1 スレッドではわずか 7% だったが，12 スレッドでの実行時では約 80% 占めることがわかった．さらに A) の通信スレッドが内部要求キューから要求を取り出すまでの時間もスレッド数が増えるにつれ増加している． これはスワッププロトコル R58 では，公平性のため通信スレッドが内部要求キューと外部要求キューの取り出しを交互に行うことを原則としており，両方のキューに要求数の偏りがある場合に効率的とは言い難い状況があった．これを緩和するために，キュー取り出しスケジューリングに関し幾つかの改善を試みたが，プログラム実行中に動的に変動するページ要求数と受け取ったページの割付要求数の変動にうまく適応させることが難しいことがわかった．このため，DLM の全体制御を通信スレッドに任せ，受信のみを受信スレッドが行うというスワッププロトコル R58 では，効率化が難しいことがわかった．

3　ページ交換専用スレッドによるプロトコル

3.1　スワッププロトコル R77

スワッププロトコル R58 のマイクロベンチマークを用いた性能評価実験の結果から，特に受信スレッドでページを受信してから，ページ割付要求を外部要求キューに入れた後，通信スレッドが外部要求キューを見て処理するまでに時間がかかっていたことが分かった．そこで，DLM のページ交換に関する処理を通信スレッドから受信スレッドに移すスワッププロトコル R77 を構築した．スワッププロトコル R77 の手順を以下に示す．

(1)ユーザースレッドは Segv を起こすとハンドラ内で，内部要求キューに必要なページ番号を入れる

(2)通信スレッドが内部要求キューを見て，各メモリノードへページ要求を送信する

(3)メモリノードは受け取ったページ要求を処理し，計算ノードへページを送信する

(4)受信スレッドはユーザースレッドを一時停止させ，ページをユーザー空間の領域に直接受け取る

(5)受信スレッドがページ表を見て計算ノードの保持するページの中，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

(6)ユーザースレッドを再開させる

スワッププロトコル R77 の特徴は受信スレッドがページを受け取る時，ページをバッファではなくメモリに直接受け取ることと，受信スレッドがページ交換を行うことである．これにより外部要求キューに処理をためずにページ交換を行うことができる．また，これまで DLM 内部データは通信スレッドのみをアクセス可能にしていたが，スワッププロトコル R77 では，受信スレッドによるアクセスも行われるため，内部データの排他制御機構も新たに組み込んだ．

3.2 プロトコル R77 のマイクロベンチマーク性能評価

スワッププロトコル R77 の性能を確認するために，スワッププロトコル R58 で用いたマイクロベンチマークによる評価実験を行った．スワッププロトコル R58 との比較のために以下の 4 つの計測個所を決め，各計測個所での平均を求めた．

A)ユーザースレッドの Segv 発生直後から，通信スレッドが内部要求キューの中から呼び出すまで(A1-A2)

B) 通信スレッドがページ要求を出してから受信スレッドでページを受信するまで(A2-A3)

C) 受信スレッドがユーザースレッドを一時停止させ，ページを受信し，スワップアウトページをメモリノードに送信し，ユーザースレッドを再開させるまで(A4-A5)

D) スレッドが再開してから全てのページ交換処理が終了するまで(A5-A6)

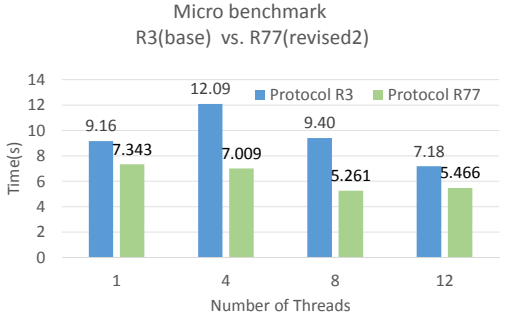

図 3.1 R77 におけるマイクロベンチマークの実行時間

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

図 3.1 は R3 と R77 の性能を示す．縦軸が実行時間，横軸が計算に使ったスレッド数を表す．R77 では，R3 に比べ，1 スレッドでは実行時間が 1.25 倍，12 スレッドでは約 1.3 倍の高速化に成功した．R58 の実行時間と比べると 1 スレッドでは 1/4，12 スレッドでは 1/3 の時間で実行することができる．

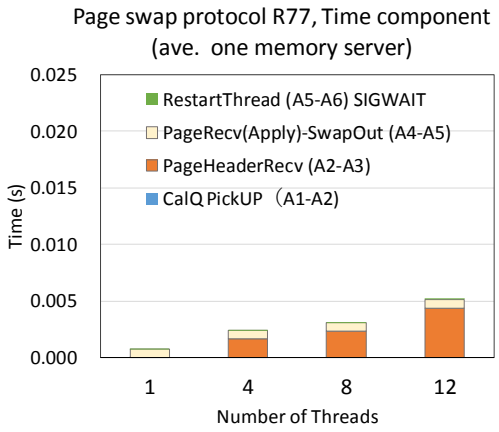

図 3.2 R77 における計測個所毎の実行時間

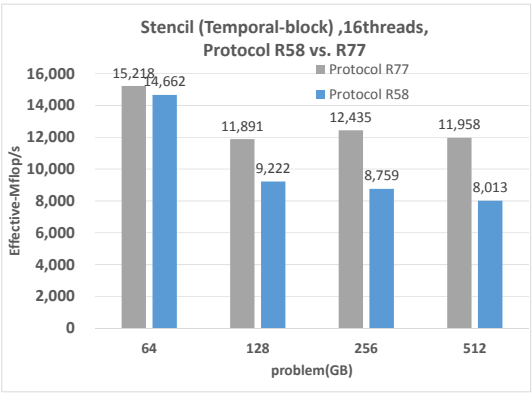

図 4.2 R58 と R77 の 7 点ステンシル計算の性能

図 3.2 は R77 における計測個所毎の実行時間を実行スレッド毎にまとめたグラフである．横軸が計算に使ったスレッド数を表し，縦軸は各時間成分を表す．このグラフから R77 では，R58 で最大 80%を占めていた外部要求キュー，内部要求キューの中での待機時間（図 2-4 の A3-A4）が無くなった．しかし B)の通信スレッドがページ要求を送ってから受信スレッドが受信するまでの時間が長くなっている．プロトコル B では，受信スレッドがページ交換を一つのメモリノードに対し連続して行うため効率的であるが，一方で，1 回のページ交換に必要なページ送受信をすべて受信スレッドが行うので，そこがボトルネックになっている．メモリノードが複数あった場合において，他のページがすでに送信されてきても，すぐには処理できないという状況が発生する．

CPU	Intel® Xeon® CPU E5-2687W v3 @ 3.10GHz 2CPU × 10core/node
Memory	128GB/node
Cache	L2 256KiB L3 25MiB
Network	Infiniband FDR
OS	CentOS 7.1.1503 (Core) Linux 3.19.5
Compiler	gcc version 4.8.3
MPILib	MVAPICH2 version 2.0.1

表 4.1 動作実験環境 2

4 アプリケーションによる性能評価

4 章では，実際のアプリケーションプログラムにおける改良プロトコルの効果を調査した．本実験ではテンポラルブロッキングを用いた 7 点ステンシル計算[2]と Stream[3]を使用した．動作環境は Stream では表 2.1 を使用し，ステンシル計算では表 4.1 を使用した．

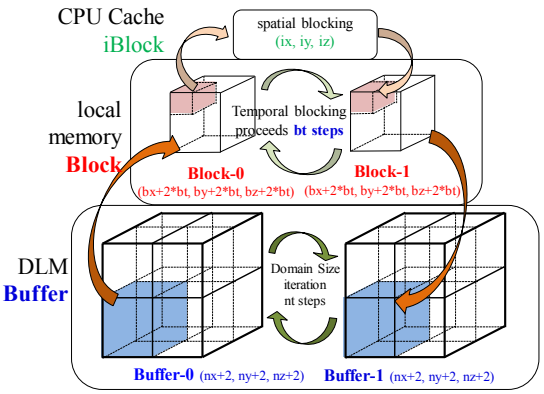

図 4.1 ステンシル計算での DLM のメモリレイアウト

4.1 テンポラルブロッキングを用いた 7 点ステンシル計算

7 点ステンシル計算は最も基本的な格子計算の一つで，更新する 1 点のデータとそれに隣接する 6 点のデータを用いて計算を行う．全ての点のデータを更新し，これを複数時間ステップ回数，繰り返す．図 4.1 のように計算ノードに収まる量のブロックを 2 つ確保し，バッファを計算ノードとメモリノードに確保する．

本実験では空間ブロッキングとテンポラルブロッキングを用いた 7 点ステンシル計算[2]を行った．問題サイズは 64GiB から 512GiB までを実行し，スレッド数は 16 スレッドで計測した．計算ノードの DLM 領域 120GiB，メモリノードの DLM 領域 120GiB，問題サイズ 64GiB から 256GiB まではメモリノード 3 台，問題サイズ 512GiB ではメモリノード 5 台を用いている．ページサイズ 1MiB で計測した．

図 4.2 はスワッププロトコル R58 とスワッププロトコル R77 を導入した DLM で実行した 7 点ステンシル計算の Effective Mflops/s と問題サイズの関係を示したグラフである．横軸が問題サイズ，縦軸が Effective Mflops/s を表す．

問題サイズ 64GiB ではどちらも全データが計算ノードに収まっているので性能差はない．データが計算ノードに収

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

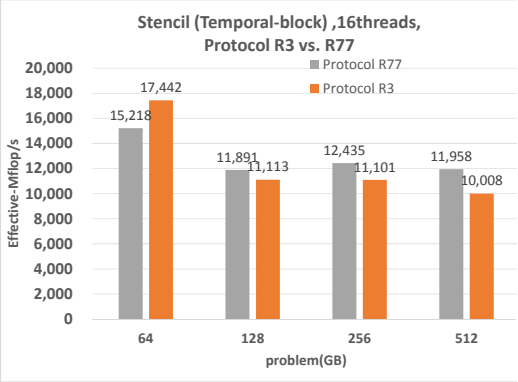

図 4.3 R3 と R77 の 7 点ステンシル計算の Effective Mflops/s

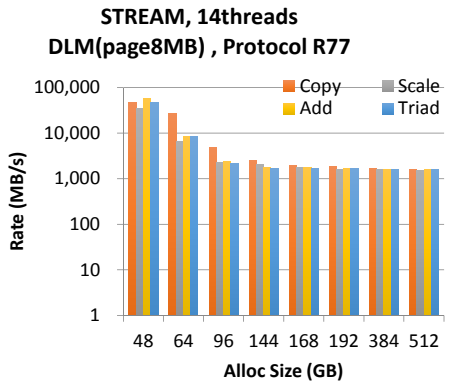

図 4.4 R77 の Stream 実行結果

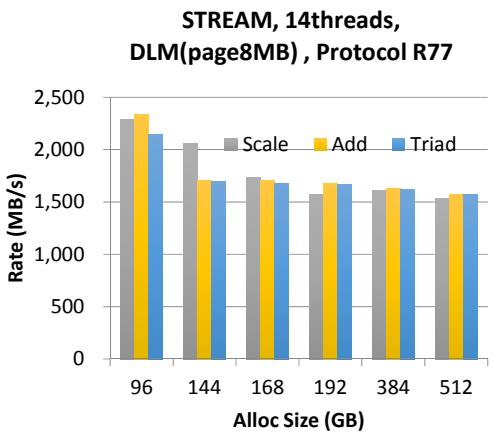

図 4.5 R77 の Stream 実行結果 2

まりきらない問題サイズ 128GiB 以上の計算では，R58 と比べ R77 は 1.3 倍から 1.5 倍の Effective Mflops/s を持つ．R58 では問題サイズが増えるにつれ徐々に性能が落ちていき，問題サイズ 64GiB の性能と比べると問題サイズ 512GiB では約 54%の性能しか出せていない．一方，R77 では問題サイズ 64GiB の性能と比べると問題サイズ 512GiB では約 78%の性能で実行できている．

図 4.3 は R3 と R77 で実行した 7 点ステンシル計算の Effective Mflops/s と問題サイズの関係を示したグラフである．横軸が問題サイズ，縦軸が Effective Mflops/s を表す．

問題サイズ 64GiB（遠隔メモリは未使用）ではプロトコル R3 はプロトコル R77 よりも 1.14 倍近く性能が優れている．遠隔メモリを利用する問題サイズ 128GiB 以上ではプロトコル R77 の性能が上回っている．問題サイズ 512GiB では，プロトコル R77 がプロトコル R3 に比べおよそ 1.2 倍の性能を持つ．プロトコル R3 では，問題サイズ 512GiB の性能が，問題サイズ 64GiB（ローカルメモリアクセスのみ）の性能と比べ 57%に低下している．これに対し，プロトコル R77 はプロトコル R3 の 78%の性能で実行出来た．これは，通信スレッドと受信スレッドでページ要求とページ交換を独立して行うことができるようになり，複数台のメモリノードとの処理を同時に進めることができるようになったためである．

4.2 Stream ベンチマーク

Stream はメモリアクセス帯域幅を調査するベンチマークである．Stream の演算で用いる 3 つの配列を DLM 領域で確保し演算を行うように改変した．計算ノードの DLM 領域を 56GiB，メモリノードの DLM 領域を 120GiB に設定し計算ノード 1 台，メモリノード 4 台，ページサイズ 8MB，計算に 14 スレッドを用いて計測を行った．3 つの配列の大きさの合計を 48GiB から 512GiB まで確保し計測を行った．48GiB の配列は，実際には，遠隔メモリを利用せずに計算ノードのローカルメモリアクセスのみになる．

図 4.4 はプロトコル R77 における Stream の結果を示す．縦軸は帯域幅を対数で表し，横軸は確保した配列の総量を表す．3 つ配列が DLM 領域に収まっている 48GiB では，処理の種類によって性能差が出ていないが，64GiB 以上では Copy を除く 3 つの処理では大きく影響が出ている．配列サイズ 64GiB の性能は配列サイズ 48GiB と比べ，59%まで性能が低下する．これはプロトコル R3 とプロトコル R58 と比べ最も性能低下が起きているが，性能としてはプロトコル R58 よりも 500MB/s 高い．

図 4.5 はプロトコル R77 における Stream の問題サイズ 96GiB 以上の結果を示すグラフである．縦軸が帯域幅，横軸が問題サイズを表す．配列サイズ 168GiB 以上の性能は 1500MB/s 以上を保つ．後述するプロトコル R3 とプロトコル R58 よりも通信効率が良いことがわかる．プロトコル

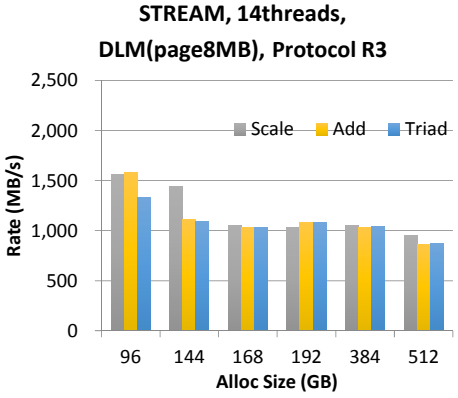

図 4.6 R3 の Stream 実行結果

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

R77 ではプロトコル R3 と比べると，配列サイズ 512GiB の性能では，約 600MB/s 優っている．

図 4.6 はプロトコル R3 における Stream の問題サイズ 96GiB 以上の結果を示したグラフである． 計算ノードが持つ DLM 領域の 3 倍の大きさの配列サイズ 168GiB 以上で実行すると性能は 1000MB/s を保つ．

図 4.7 はプロトコル R58 における Stream の結果を示す．ローカルアクセスのみの配列サイズ 48GiB において，Copy の性能はプロトコル R3 と同じ性能だったが，配列サイズ 64GiB においては 48GiB の場合の性能と比べ 67%の性能である．プロトコル R3 と R58 の性能低下を比較すると，配列サイズ 64GiB の性能はローカルメモリアクセスの場合（配列サイズ 48GiB の性能）の 80%で，プロトコル R58 ではメモリノードの通信で大きく性能を落とす．配列サイズ 168GiB 以上の性能は 600MB/s を保つ．

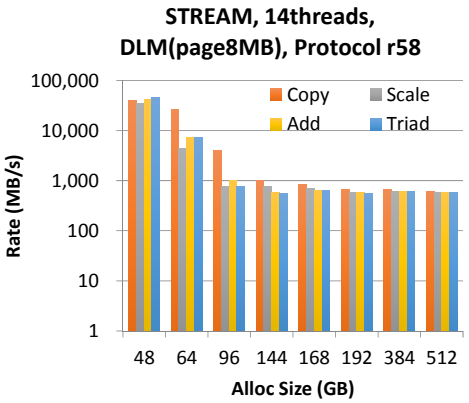

図 4.7 R58 の Stream 実行結果

5 dlm_get 関数と dlm_put 関数の設計

DLM は，Segv を利用して，DLM データに対しどのようなデータアクセスでも対応できるが，1 枚ずつページ交換を行うため，必ずしも効率的ではない場合がある．GA[5]などの PGAS ランタイムにはユーザーがノード間のデータ送受信を明示して，ローカル領域とグローバル領域のデータを操作する機能(get , put)がある．これらの機能を DLM に取り入れることにより，ローカルデータと DLM データ間のデータコピーという新たなデータ転送方式を実現できる． get,put を利用すると，4.1 で用いたステンシルのようなアルゴリズムでは，DLM データの一部をローカルデータにコピーし，ローカルデータに処理を行った後，ローカルデータの結果を DLM データに書き戻すという処理方式が可能となる．これにより計算ノード実行中には segv が発生せずに性能が向上する．データコピーであるため，ページ交換は不要で，DLM データをローカルノードにキャッシュする必要もない．これまで 1 ページ単位で行っているページ交換と異なり，複数ページやページサイズ未満で行うように実装することもできる．

以下に dlm_get と dlm_put の関数の概要を示す．dlm_get は DLM 大域データからローカルデータへ連続データのコピーする関数で，dlm_put はその逆である．いずれの関数も DLM データアドレス，ローカルデータアドレス，サイズを引数とする．dlm_get では，通信スレッドが DLM 連続データの所在を特定して複数のメモリノードへ要求を出し，受信スレッドがこれらのデータを集めて，指定したユーザー領域にコピーする．すべてのデータコピーが終了すると，dlm_get を返す．1 ページを超える大きな領域コピーも可能で，通信効率を高めることができる．dlm_put は，通信スレッドが DLM データを保持するメモリノードを特定して，ローカルデータを送信し，受信スレッドは，各メモリノードからの DLM データの更新の確認を受け取り，全領域のコピー終了後に dlm_put を返す．

さらにこの関数を組み合わせて，GA[5]で実装されているような DLM 大域データにある多次元データの部分ブロックを転送する get 関数を作ることもできる．この場合は，多次元の小ブロックを特定する始点，終点，ストライドなどを指定する引数をとる関数となる．

6 おわりに

本研究では，DLM の計算ノードとメモリノードの通信効率を向上させるために，新たに 2 つの通信プロトコルの設計と実装を行い，7 点ステンシル計算と Stream を用い評価した．その結果，受信プロセスを新たに導入して，ページ交換とページ要求の処理を独立して行うプロトコル R77 が，有効であることがわかった．ステンシル計算では主メモリの 4 倍の問題サイズに対して主メモリ内に収まる問題サイズの時の性能と比べ，78%の性能で実行することができた．Stream では，遠隔メモリアクセスバンド幅として約 1500MB/s の性能が得られることがわかった．

今後は，さらに効率化を図るために，複数ページをまとめて転送する方式，遠隔メモリに展開するデータと，ローカルメモリに固定されたデータ間での効率的なコピー操作（GET, PUT)なども，新たに開発し検証する予定である．

参考文献

[1] 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

[2] Hiroko Midorikawa, Yuichiro Suzuki, and Masatoshi Iwaida: "User-level Remote Memory Pagingfor Multithreaded Applications", proc. of IEEE/ACM International Symp. on Cluster, Cloud and the Grid ComputingCCGrid2013,pp.196-197,2013-5(DOI10.1109/CCGrid.2013.63)

[3] Hiroko Midorikawa, Hideyuki Tan and Toshio Endo:"An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations", Proceedings of the 2014 International Conference on High Performance Computing and Simulation (IEEE HPCS2014) (ISBN 978-1-4799-5311-0), pp.268-277, 2014-7 IEEE-HPCS2014

[4] John D. McCalpin :"STREAM: Sustainable Memory Bandwidth in High Performance Computers" http://www.cs.virginia.edu/stream/

[5] Nieplocha, Jarek. "Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit", International Journal of High Performance Computing Applications , (2006),20 (2): 203–231

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.