Vol.2016-HPC-155 No.31
2016/8/9

Flash SSDを含む多階層メモリを活用するPGASランタイムシステム

松宮 遼[1,a)]　遠藤 敏夫[1,b)]

概要：DRAM に収まりきらない（out-of-core な）データを高速に処理するための手法として，Flash SSD を活用するものがある．本稿では，既存の PGAS ランタイムである Global Arrays（GA）を変更することで，GA で作成されたプログラムが out-of-core なデータの処理を行えるようにする手法について述べる．提案手法では，1 ノードあたりに複数のプロセスを立ち上げる．これらのプロセスが持つメモリ領域を自動的に Flash SSD 上にスワップさせることで，out-of-core なデータを GA で処理できるようにした．更に我々は，提案手法を施した GA を用いて，2 次元 9 点ステンシル計算を行うプログラムと，行列積計算を行うプログラムを作成した．それらの性能評価を行った結果についても，本稿にて報告する．

1. はじめに

ポストペタ・エクサスケールと呼ばれる時代に突入し，並列計算機が処理するデータの大きさは増大し続けている．その一方で，トランジスタの密度の限界により，1 枚の DRAM に搭載できる容量は停滞している．つまり，並列計算機において DRAM の容量を超えるサイズの（out-of-core な）データを処理する必要がある．

半導体生産技術の向上により，HDD よりも高速かつ DRAM よりも大容量な Flash SSD が安価で市場に流通するようになった．Flash SSD を活用することで，ステンシル計算 [1] やソート [2] において out-of-core なデータを高速に処理させる手法が明らかとなっている．

Global Arrays（GA）[3]，XcalableMP [4]，X10 [5]，Unified Parallel C [6]，Titanium [7] などの Partitioned Global Address Space（PGAS）ランタイムによって高性能なプログラムを容易に開発できるようになった．PGAS ランタイムにはノードローカルな HDD や，分散ファイルシステム上のファイルを容易に操作するための入出力機能を備えたものがある [8, 9]．そのような機能をノードローカルな Flash SSD に対して適用することで，out-of-core なデータをプログラムに処理させることが可能となる．しかしながら，そのような機能を利用するには，Flash SSD に保存するデータ領域やタイミングをソースコード上でプログラマーが明示する必要がある．つまり，Flash SSD に保存するデータ領域やタイミングを明示させることなく，out-of-core なデータを処理できる PGAS ランタイムが求められている [10]．

我々は，Flash SSD を活用し，プログラムに out-of-core なデータを処理する PGAS ランタイムの開発を行った．本研究は新たな PGAS ランタイムを構築するものではない．我々は既存の PGAS ランタイムである GA に対して変更を加えた．具体的には，GA が利用している通信ライブラリである Communication runtime for Extreme scale（ComEx）[11] が Flash SSD を活用する ComEx-PM（ComEx for Post Moore era）の開発を行った．我々の目標は，プログラマーによって Flash SSD に対する入出力を明示しなくても，out-of-core なデータを高速に処理するプログラムを開発することが可能となることである．

本稿の構成を次に示す．2 章では背景として GA と，GA が利用している通信ライブラリの 1 つである ComEx について説明する．3 章では我々が開発している ComEx-PM の設計と実装について述べる．4 章で，ComEx-PM を導入した GA の性能評価について議論する．5 章で関連研究を紹介し，6 章でまとめと今後の課題について述べる．

2. GA

2.1 GA の仕様

本節では，GA の仕様について説明する．

GA は C/C++，Fortran および Python で提供される API を用いて操作する．

GA にはローカル領域とグローバル領域の 2 つのメモリ空間が存在する．ローカル領域は，プロセス毎に独立した

[1] 東京工業大学
Tokyo Institute of Technology
[a)] matsumiya.r.aa@m.titech.ac.jp
[b)] endo@is.titech.ac.jp

```
NDIM = 2;
l_a[2][2];
size_g_a[] = {8, 4};
g_a = NGA_Create(double, NDIM, size_g_a);
lo[] = {1,2};
hi[] = {2,3};
do_something(l_a);
NGA_Put(g_a, lo, hi, l_a);
lo[] = {0,1};
hi[] = {1,2};
NGA_Get(g_a, lo, hi, l_a);
```

図 1　GA を用いたソースコードの簡略な例

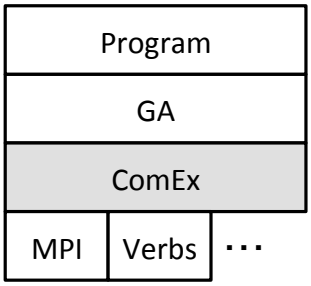

図 2　ComEx のソフトウェアスタック

メモリ空間である．ローカル領域上の空間は通常のプログラミングにおけるメモリ空間と同様に，malloc() 等の関数によって確保することができる．

グローバル領域は，全プロセス間で共有されているメモリ空間である．グローバル領域上の空間は NGA_Create() 等の API 関数によって確保することができる．確保された領域の実体は，各プロセスに分散されている．グローバル領域は 1 次元以上の行列として操作する．グローバル領域へのアクセスも NGA_Get()，NGA_Put() 等の API 関数によって行う．

GA はグローバル領域上の行列に対して行列積や LU 分解等の計算を行うための API 関数が存在する．本研究で述べる実験においては，GA_Dgemm() を用いてグローバル領域上の行列 2 つに対する行列積を計算する処理を行っている．

GA を用いたソースコードの簡略な例を図 1 に示す．まず NGA_Create() によって大きさが 8×4 の double 型の 2 次元配列 g_a をグローバル領域上に確保する．その後大きさ 2×2 のローカル領域の行列 l_a の内容を書き換えて，その内容を g_a[1:2][2:3] にコピーする．グローバル領域へのコピーは NGA_Put() によって行われる．次に g_a[0:1][1:2] の内容を l_a にコピーする．コピーは NGA_Get() によって行われる．

2.2　ComEx

ComEx とは，GA で利用されている PGAS ランタイム向け通信用ライブラリである．ComEx のソフトウェアスタックを図 2 に示す．PGAS ランタイムから要求を受け取ると，MPI，Verbs 等の低レベルのネットワークインターフェースを用いて通信を行う．

ComEx を利用する PGAS ランタイムは Put，Get，Acc の 3 種類の通信を ComEx に要求することができる．各要求は，ローカルプロセスの仮想メモリアドレス，リモートプロセスの仮想メモリアドレス，リモートプロセスのランクを，PGAS ランタイムより受け取る．Put はローカルプロセスの仮想メモリアドレスの内容をリモートプロセスの仮想メモリアドレスにコピーする．Get はリモートプロセスの仮想メモリアドレスの内容をローカルプロセスの仮想メモリアドレスにコピーする．Acc は指定されたリモートプロセスの仮想メモリアドレスの内容をアトミックに更新するものである．

ComEx による通信はパッキング通信に対応している．パッキング通信を用いることで，同一のリモートのランクに対する複数の通信を 1 度に集約することが可能である．現状の ComEx-PM の実装ではパッキング通信を行っていないため，詳細は省略する．ComEx が対応しているパッキング通信の仕様については，ComEx の前身である ARMCI [12] と近いものとなっているのでそちらを参照されたい．

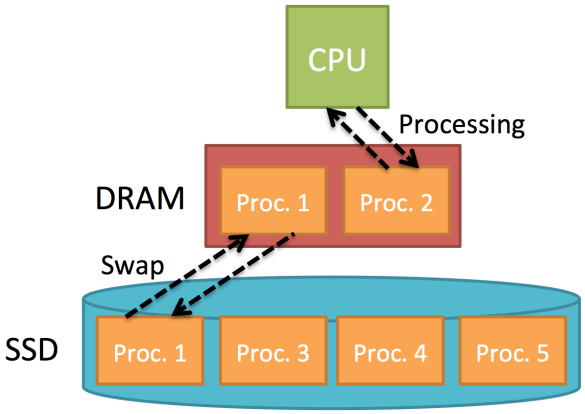

図 3　ComEx-PM の概略図

3.　ComEx-PM の設計と実装

3.1　ComEx-PM の概要

ComEx-PM の概略図を図 3 に示す．ComEx-PM では 1 ノードに複数のプロセスを立ち上げる．ここで 1 つのプロセスが利用するメモリ領域は，DRAM の容量に収まるものとする．

全プロセスが利用しているメモリ領域の合計が，DRAM の容量を超えるとき一部のプロセスが持つメモリ領域を Flash SSD にスワップアウトさせる．他プロセスとのブロッキング通信を行っているプロセスが，スワップアウトの対象となる．スワップアウトされたプロセスのブロッキ

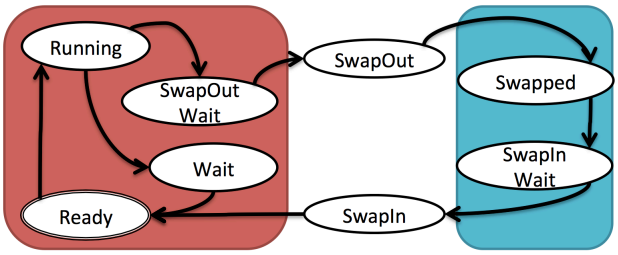

図 4　ComEx-PM におけるプロセスの状態遷移図

ング通信が完了し，DRAM が十分に空いていた場合，そのプロセスが持つメモリ領域は Flash SSD から DRAM にスワップインされる．

プロセス管理やメモリ領域管理，通信制御の実装は HHRT [13, 14] を利用している．ComEx-PM では HHRT に ComEx の実装を組み合わせた上で，メモリ管理やプロセス間通信に関する部分の実装に変更が加えられている．

3.2　プロセスの状態遷移

ComEx-PM において，各プロセスは図 4 にて示す状態遷移を起こす．赤はプロセスのメモリ領域が DRAM 上に，青は Flash SSD 上にあることを表している．

Running は現在実行中のプロセスを表す．ブロッキング通信や，他プロセスとの同期を行う時，DRAM の容量に余裕があれば Wait に，そうでなければ SwapOutWait に状態遷移する．

Wait はメモリ領域が DRAM 上に存在するが，他プロセスからの通信を待機している状態である．他プロセスからの通信が完了すると Ready に状態遷移する．

Ready はメモリ領域が DRAM 上に存在し，待機している通信もない状態である．全てのプロセスの初期状態でもある．1 つの Running なプロセスが Wait に状態遷移すると，Ready の状態にある全てのプロセスが Running になることを試みる．1 度に Running となることのできるプロセスはただ 1 つで，失敗したプロセスは Ready 状態を維持する．

SwapOutWait はメモリ領域が DRAM 上に存在するが，まもなく Flash SSD 上にスワップされる状態である．原則として，同時にスワップアウト・スワップインできるプロセスは 1 つとしている．あるプロセスによるスワップアウト・スワップインが完了すると，SwapOutWait なっている全てのプロセスはスワップアウトすることを試みる．スワップアウトが可能となれば，状態は SwapOut に遷移する．

SwapOut は Flash SSD 上にスワップアウトしている最中であることを表している．スワップアウトが完了すると，状態は Swapped に遷移する．

Swapped はメモリ領域が Flash SSD 上に存在している状態である．他プロセスからの通信が完了した場合は，状態は SwapInWait となる．

SwapInWait はメモリ領域が Flash SSD から DRAM へのスワップインを待機している状態である．先に述べた通り，現状の ComEx-PM では同時にスワップアウト・スワップインするプロセスはただ 1 プロセスのみである．そのプロセスがスワップアウト・スワップインが完了すると，SwapInWait なっている全てのプロセスはスワップインすることを試みる．スワップインが可能となれば，状態は SwapIn に遷移する．

SwapIn はメモリ領域が Flash SSD から DRAM にスワップインしている最中であることを表している．スワップインが完了すると，状態は Ready に遷移する．

3.3　スワップ機構

ComEx-PM では Flash SSD 上のファイルに対してスワップを行う．1 つのファイルは 1 つのプロセスに対して紐付けられている．このスワップ機構の方針として，OS に備わっているスワップ機構を利用する方法と ComEx-PM にスワップ機構を実装する方法の 2 つがある．前者の方法よりも，後者の方法の方が高速に処理できることが Midorikawa ら [1] によって明らかとなっているため，ComEx-PM では後者を採用している．プロセスが持つメモリ領域へのスワップアウトは，紐付けられたファイルへの書き込みとして行われる．スワップインは紐付けられたファイルからの読み込みとして行われる．

各プロセスのメモリ領域は，プロセス立ち上げ時やスワップインされるときに確保され，スワップアウトが完了すると解放される．この時，`malloc()` 等でメモリ領域を確保すると，スワップが発生した時に仮想メモリアドレスが変わってしまい，実行中のプロセスが正常に動作しなくなってしまう．そこで ComEx-PM では領域を確保するときに `mmap()`，開放する時に `munmap()` を利用する．`mmap()` にて領域を確保する際に `MAP_ANON` を指定することで，物理メモリ上にマッピングさせる．その上で `MAP_FIXED` も指定することで，スワップが発生した後でも同一の仮想メモリアドレスが利用できるようにした．

プロセス内にある全てのメモリ領域がスワップアウトされるわけではない．ComEx-PM では，スワップアウトされている間であっても他ノードとの通信を行う．本稿執筆時点での ComEx-PM の実装では，プログラムが `malloc()` および `calloc()` で確保した領域と，そのプロセスに実体のあるグローバル領域，原則的にはそれら 2 領域がスワップの対象となる．スワップアウトされている間に行う通信に必要なメモリ領域はスワップの対象としていない．プログラムが呼び出す `malloc()` 及び `calloc()` は，先ほど説明した `mmap()` によって確保された仮想メモリ空間を利用するように ComEx-PM によってフックされる．

3.4 プロセス間通信

本節では ComEx-PM においてプロセス間で発生する通信について述べる．

ComEx-PM のプロセス間通信は MPI によって行われる．MPI を用いた多くの PGAS ランタイムでは，MPI-3 の片方向通信を利用している [15, 16]．片方向通信を利用する場合，通信が開始してから通信が終了するまでの間，相手側のプロセスが持つメモリ領域が DRAM 上にある必要がある．しかしながら，自分以外のプロセスのメモリ領域が DRAM 上にあるかどうか把握することは困難である．

以上の理由より，ComEx-PM では片方向通信を利用せず，双方向通信を用いた手法をとっている．双方向通信を行うことにより，相手側のプロセスは自身のメモリ領域が存在するデバイスを把握した上で，プロセス間通信を制御できる．現状の ComEx-PM におけるプロセスは，自身の状態が Running，SwapIn，SwapOut の何れでもない時に，他のプロセスからの通信に対する応答を行う．

ComEx-PM において Put，Get，Acc が呼び出された時の通信フローを図 5 に示す．ここでは，P1 が行う P2 上のメモリ領域に対するアクセスを表す．

Put では，P1 が `MPI_Isend()` によって Put するデータや Put される先の仮想メモリアドレス等を P2 に送信する．P2 は `MPI_Iprobe()` によって通信を感知し，`MPI_Recv()` によってデータを受信する．この際，P2 のメモリ領域が DRAM 上にあれば，そのデータは直接メモリ上に書き込まれる．もし P2 のメモリ領域が Flash SSD 上にあれば，そのデータは事前に確保されたバッファを通して Flash SSD 上の P2 に割り当てられたファイルに書き込まれる．

Get では，P1 が `MPI_Isend()` によって Get するデータの仮想メモリアドレス等を P2 に送信する．P2 は `MPI_Iprobe()` によって通信を感知し，`MPI_Send()` によってデータを送信する．P1 は `MPI_Irecv()` によってデータを受信する．この際，P2 のメモリ領域が DRAM 上にあれば，そのデータは直接メモリ上のデータを送信する．もし P2 のメモリ領域が Flash SSD 上にあれば，Flash SSD 上の P2 に割り当てられたファイルをバッファに読み込む．ファイルの読み込みが完了すると，そのバッファの内容を送信する．

Acc では Put と同様の手法で P2 がデータを `MPI_Recv()` にてデータを受け取る．この時，P2 のメモリ領域の所在に関わらずバッファ上に受け取る．その後，P2 は P1 から指定された仮想メモリアドレスの場所のデータに対して指定された操作 (例えば，P2 のメモリ領域のデータとバッファ上のデータを加算して，P2 のメモリ領域上のデータに書き戻す操作) を行う．P2 のメモリ領域が Flash SSD 上にある時，この操作で発生する P2 のメモリ領域上の読み書きはファイルの読み書きとして行われる．

ComEx ではローカルプロセス内で完結する通信が要求される場合がある．そのような場合は，MPI による通信を行わない．`memcpy()` によるメモリコピーを行うことによって通信処理をしたとみなす．これによって MPI 通信によるオーバーヘッドを削減することが可能となる．

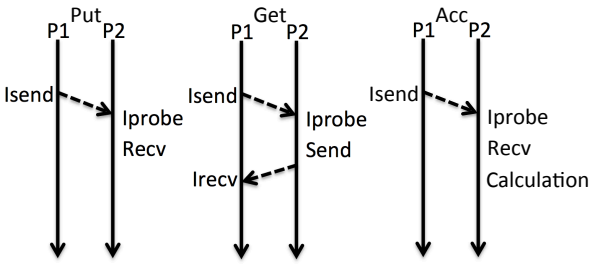

図 5 ComEx-PM が行う通信のフロー

表 1 評価環境

CPU	Core i7-6700K 4.0 GHz 4 cores 8 threads
DRAM	64 GB
SSD	Samsung 950PRO m.2 512 GB
OS	CentOS 7.2
File system of SSD	XFS
GA	5.4
MPI	MPICH 3.2
BLAS	LAPACK 3.4.2

4. 性能評価

ComEx-PM の性能を評価するために，我々は 2 つのプログラムを GA によって作成し，その性能を測定した．入力データはランダムなデータとして既にグローバル領域上に存在している．この状態から，出力データをグローバル領域に書き出すまでを実行時間とし，その Flops 値を求めた．評価環境を表 1 に示す．

作成したプログラムの 1 つは 2 次元の 9 点ステンシルである．大きさ $N \times N$ の倍精度浮動小数点型行列に対して，N を変化させつつ Flops 値を測定した．実際には，プロセス数 P に対して $N = 8192 \times \sqrt{P}$ として，P を変化させることで実験を行った．

Flash SSD を用いた out-of-core な問題サイズにおいてステンシル計算を行う場合, テンポラルブロッキング [17,18] という最適化手法が有用であることが分かっている [1,13,14]．テンポラルブロッキングとは，ステンシル計算において部分配列の処理を行うときに複数の時間ステップをまとめて計算するというものである．k ステップをまとめて計算することで，袖領域の通信回数を $1/k$ 回に抑えることができる．本実験では $k = 10$ としたテンポラルブロッキングが，テンポラルブロッキングを行っていない時との性能比較を行った．

9 点ステンシルの測定結果を図 6 に示す．テンポラルブ

ロッキング（TB）を行っていない場合と比較して，$k = 10$ としたテンポラルブロッキング適用時は最大 9.65 倍の性能改善が見られた．ComEx-PM を利用した GA においても，テンポラルブロッキングが有効であることを示す結果となった．$N = 8192$ の時は 1 プロセスでの実行であるため，MPI による通信が発生していない．したがって $N = 8192$ から $N = 32768$ の間で性能が大きく落ちているのは，MPI 通信によるオーバーヘッドによるところが大きいと言える．現在の ComEx-PM の実装は，パッキング通信を行っていないため，計算対象の行列 1 行につき 1 回の MPI 通信が発生している．パッキング通信を導入し，MPI 通信の回数を削減すれば性能が向上する可能性がある．「out-of-core」と記されている領域は DRAM の容量を超えた問題サイズである．この時，Flash SSD と DRAM の間でスワップが生じている．つまり $N = 13102$ から $N = 204800$ の間で性能が低下しているのは，SSD へのスワップによる影響である．しかしながら，ここでみられる性能低下は，手作業で out-of-core な問題サイズに対してステンシル計算を行った Midorikawa らによる実験結果 [1] よりも大きくなっている．調査の結果，その原因は下記のようなものであった．各プロセスの DRAM 上のメモリ領域が Flash SSD にスワップアウトされるタイミングは，ブロッキング通信を行うときである．これは ComEx-PM における当該部分の実装が HHRT をベースにしているためである．一方で我々の 9 点ステンシルプログラムでは，袖領域の通信のために，(最大で)8 個ずつの非同期通信と待ち合わせ処理（`NGA_NbWait()`）を呼び出している．`NGA_NbWait()` のために，本来必要な回数以上のスワップ処理が発生しており，そのコストが時間ブロッキングだけでは隠ぺいしきれなかったと考えられる．これを改善するために，このようなケースに対してスワップアウトを 1 度しか行わないような機構を，プログラムへの影響が少ないまま実現することが，今後の課題の一つである．

性能評価のために我々が実装したもう一方のプログラムは行列積（Dgemm）である．ここでは，GA の API 関数である `GA_Dgemm()` の性能を測定した．`GA_Dgemm()` はグローバル領域上にある 2 つの倍精度浮動小数点型行列の積をグローバル領域上の別の行列に書き込む．実験は，大きさ $N \times N$ である 2 行列の積を求める．ここでも，プロセス数 P に対して $N = 8192 \times \sqrt{P}$ として，P を変化させることで実験を行った．

行列積の性能測定結果を図 7 に示す．行列積では，out-of-core な問題サイズにしたときの性能低下が 9 点ステンシルのときほど大きくは観測されなかった．原因は調査中ではあるが，MPI ブロッキング通信の数がステンシル計算よりも少なく，スワップ機構によるオーバーヘッドが小さかったためと考えられる．

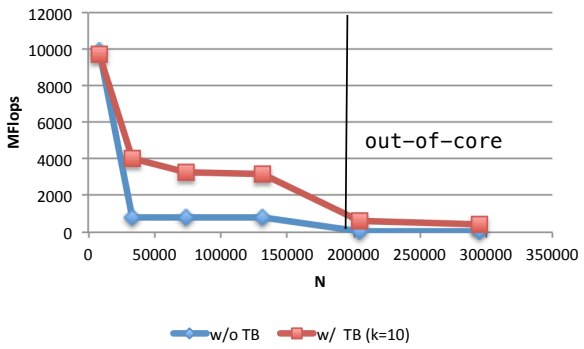

図 6　9 点ステンシルの性能

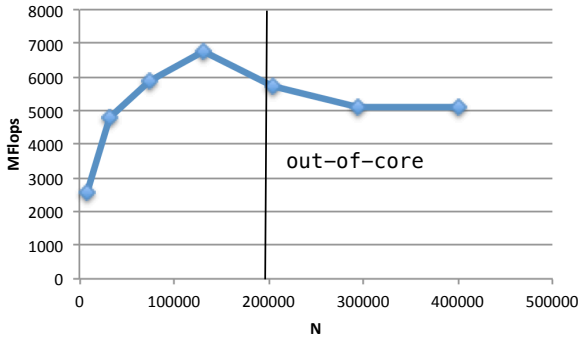

図 7　`GA_Dgemm()` の性能

5. 関連研究

本論文の共著者である遠藤 [13,14] は，MPI+CUDA で記述されたプログラムを，out-of-core なデータへの対応を容易に行うためのライブラリである HHRT を開発した．HHRT では本研究と同様に，Flash SSD を含めた多階層メモリを活用する．DRAM の容量に収まりきるメモリ領域を利用するプロセスを 1 ノード上に複数立ち上げ，ブロッキング通信が発生しているプロセスを Flash SSD にスワップアウトさせることで実現している．本研究は PGAS ランタイムである GA を対象としている．

DRA [8] は GA を out-of-core なデータに対応させるためのライブラリである．このライブラリが提供しているライブラリ関数を利用することで，グローバル領域のデータをノードローカルの Flash SSD に一時的に保存することができる．SSD に保存するタイミングや，どのデータを保存するかは開発者がソースコードに明示する必要があるという点で本研究と異なる．

Midorikawa ら [1] は，Flash SSD を用いて out-of-core なデータを対象としたステンシル計算を行う手法を示している．本研究で行った実験は，当該論文をもとに out-of-core に対応した GA におけるステンシル計算の性能を調査した．

PGAS ランタイムに対してキャッシュ機構を導入した

研究が複数の研究チームによって行われている．Miranda ら [19] は，Global Arrays をマルチスレッド環境上で実行するための Global Arrays の上位互換 Global Features を開発した．同一プロセス上の異なるスレッドによるアクセスを集約するために，キャッシュ機構を導入している．池上ら [20] は，細粒度の高い処理を行うための PGAS ランタイムである MassiveThreads/DM に対してキャッシュ機構を導入した．Chen ら [21, 22] は Unified Parallel C という PGAS プログラミング言語のランタイムにキャッシュ機構を導入している．彼らのキャッシュ機構では動的にプリフェッチを行う機構や，スレッド間で発生するキャッシュ競合を検出する手法を取り入れている．Su ら [23] は Java ベースの PGAS プログラミング言語である Titanium のコンパイラを拡張することで，これらのキャッシュ機構の導入により，複数のスレッドが近接した領域にアクセスするような処理を高速に実行することが可能となった．本研究においても，キャッシュ機構の導入による高速化手法を検討する必要がある．

Daily ら [11] は，GA のプロセス間通信に Progress Rank（PR）というプロセスを介入させることで，GA の高速化に成功した．1 つのプロセスは同一ノード上の 1 つの PR に紐付いている．あるプロセスに実体のあるグローバル領域への通信は，そのプロセスに紐付けられた PR が対応する処理を行う．本研究においても PR の導入を検討している．

6. まとめと今後の課題

本稿は，GA で作成されたプログラムが out-of-core なデータに対応できるようにする手法について述べたものである．提案手法では，通信ライブラリである ComEx を変更した ComEx-PM を実装することで，ノードローカルの Flash SSD を活用する．提案手法を取り入れた GA を用いて，2 つのプログラムの性能を評価した．その結果，次の 2 つが明らかとなった．

- ComEx-PM を利用した GA においてもテンポラルブロッキングはステンシル計算の性能を向上させることが可能である．
- ComEx-PM を利用した GA における行列積計算は，ステンシル計算を利用した場合よりも out-of-core なデータに対する性能低下が小さい．

ComEx ではパッキングによる通信に対応しているが，現在の ComEx-PM には導入していない．パッキングによって MPI のブロッキング通信の回数を減らし，スワップアウトによるオーバーヘッドを低減させることができると考えられる．そのため，パッキングによる通信への対応が今後の課題となる．また近年の並列計算機においては Burst Buffer の導入が進んでいる．そのため，Burst Buffer の活用もした PGAS ランタイムの開発も検討が求められる．更に我々は ComEx-PM と GPU などのアクセラレータとの融合についても考えている．この他，上述したスワップ回数の削減機構の導入，キャッシュ機構や PR の導入を検討する．それらの改善を行った上で，NWChem [24] などの，GA を用いて開発された実際のプログラムが out-of-core なデータを処理した場合の性能を評価していきたい．

謝辞

本研究は，科学技術振興機構戦略的創造研究推進事業（JST CREST）の研究課題「ポストペタスケール時代のメモリ階層の深化に対応するソフトウェア技術」の支援を受けている．

参考文献

[1] Midorikawa, H., Tan, H. and Endo, T.: An Evaluation of The Potential of Flash SSD as Large and Slow Memory for Stencil Computations, *Proceedings of the IEEE International Conference on High Performance Computing and Simulation (HPCS '14)* (2014).
[2] 佐藤 仁，溝手 竜，松岡 聡：GPU アクセラレータと不揮発性メモリを考慮した外部ソート，情報処理学会研究報告，Vol. 2016-HPC-154, No. 11, pp. 1–8 (2015).
[3] Nieplocha, J., Harrison, R. J. and Littlefield, R. J.: Global Arrays: A Portable "Shared-Memory" Programming Model for Distributed Memory Computers, *Proceedings of the IEEE/ACM Conference on Supercomputing (SC '94)* (1994).
[4] Lee, J. and Sato, M.: Implementation and Performance Evaluation of XcalableMP: A Parallel Programming Language for Distributed Memory Systems, *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW '10)* (2010).
[5] Ebcioglu, K., Saraswat, V. and Sarkar, V.: X10: Programming for Hierarchical Parallelism and Non-Uniform Data Access, *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, nd Applications (OOPSLA '05)* (2005).
[6] UPC Consortium: UPC Specifications, v1.2, Technical report, Lawrence Berkeley National Lab (2005).
[7] Yelick, K., Semenzato, L., Pike, G., Miyamoto, C., Liblit, B., Krishnamurthy, A., Hilfinger, P., Graham, S., Gay, D., Colella, P. and Aiken, A.: Titanium: A High-Performance Java Dialect, *Proceedings of the 1998 ACM Workshop on Java for High-Performance Network Computing* (1998).
[8] Nieplocha, J. and Foster, I.: Disk Resident Arrays: An Array-Oriented I/O Library for Out-of-Core Computations, *Proceedings of the IEEE Symposium on the Frontiers of Massively Parallel Computing (Frontiers '96)* (1996).
[9] 中村朋健，佐藤三久：並列 PGAS プログラミング言語 XcalableMP の入出力機能と Lustre ファイルシステムでの性能評価，情報処理学会研究報告，Vol. 2011-HPC-132, No. 36, pp. 1–9 (2011).
[10] Fürlinger, K.: Exploiting Hierarchical Exascale Hardware using a PGAS Approach, *Proceedings of the 3rd International Conference on Exascale Applications and Software (EASC '15)* (2015).
[11] Daily, J., Vishnu, A., Palmer, B., van Dam, H. and Kerbyson, D.: On The Suitability of MPI as a PGAS Run-

time, *Proceedings of the IEEE International Conference on High Performance Computing (HiPC '14)* (2014).

[12] Nieplocha, J. and Carpenter, B.: ARMCI: A Portable Remote Memory Copy Libray for Ditributed Array Libraries and Compiler Run-Time Systems, *Proceedings of the 1999 International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing Workshops (IPPS/SPDP '99 Workshops)* (1999).

[13] 遠藤敏夫：大規模・高性能演算のための多階層メモリの活用，情報処理学会研究報告，Vol. 2016-HPC-153, No. 14, pp. 1–7 (2016).

[14] Endo, T.: Realizing Out-of-Core Stencil Computations using Multi-Tier Memory Hierarchy on GPGPU Clusters, *Proceedings of the IEEE International Conference on Cluster Computing 2016 (CLUSTER '16) (to appear)* (2016).

[15] Hammond, J. R., Ghosh, S. and Chapman, B. M.: Implementing OpenSHMEM Using MPI-3 One-Sided Communication, *Proceedings of the 1st Workshop on OpenSHMEM* (2014).

[16] Zhou, H., Mhedheb, Y., Idrees, K., Glass, C. W., Gracia, J., Fürlinger, K. and Tao, J.: DART-MPI: An MPI-based Implementation of a PGAS Runtime System, *Proceedings of the 2014 International Conference on Partitioned Global Address Space Programming Models (PGAS '14)* (2014).

[17] Wolf, M. E. and Lam, M. S.: A Data Locality Optimizing Algorithm, *Proceedings of the 1991 ACM Conference on Programming Language Design and Implementation (PLDI '91)* (1991).

[18] Wittmann, M., Hager, G. and Wellein, G.: Multicore-Aware Parallel Temporal Blocking of Stencil Codes for Shared and Distributed Memory, *Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing Workshops (IPDPSW '10)* (2010).

[19] Chavarría-Miranda, D., Krishnamoorthy, S. and Vishnu, A.: Global Features: a Multithreaded Execution Model for Global Arrays-based Applications, *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud & Grid Computing (CCGrid '12)* (2012).

[20] 池上克明，田浦健次朗：分散メモリ環境上におけるタスク並列処理系 MassiveThreads/DM に対する共有メモリ環境上での模擬評価，情報処理学会研究報告，Vol. 2012-HPC-135, No. 5, pp. 1–7 (2012).

[21] Chen, W.-Y., Iancu, C. and Yelick, K.: Communication Optimization for Fine-Grained UPC Applications, *Proceedings of the 2005 International Conference on Parallel Architecture and Compilation Techniques (PACT '05)* (2005).

[22] Chen, W.-Y., Bonachea, D., Iancu, C. and Yelick, K.: Automatic Nonblocking Communication for Partitioned Global Address Space Programs, *Proceedings of the 21st Annual International Conference on Supercomputing (ICS '07)* (2007).

[23] Su, J. and Yelick, K.: Automatic Support for Irregular Computations in a High-Level Language, *Proceedings of the 19th IEEE Parallel and Distributed Processing Symposium (IPDPS '05)* (2005).

[24] Valiev, M., Bylaska, E., Govind, N., Kowalski, K., Straatsma, T., Dam, H. V., Wang, D., Nieplocha, J., Apra, E., Windus, T. and de Jong, W.: NWChem: A Comprehensive and Scalable Open-Source Solution for Large Scale Molecular Simulations, *Computer Physics Communications*, Vol. 181, pp. 1477–1489 (2010).