Vol.2017-HPC-160 No.7
2017/7/26

vGASNet:メモリ階層深化に向けたスケーラブルな低レイヤ通信ライブラリ

松宮 遼[1,a)]　遠藤 敏夫[1,b)]

概要：ポストペタ・エクサスケール時代において，DRAM の容量を超えるような超大規模な問題を解く需要がある．我々は，スケーラブルな低レイヤ通信ライブラリである vGASNet の開発に着手した．vGASNet は既存の低レイヤ通信ライブラリである GASNet を拡張したものである．vGASNet では計算ノード内の Flash SSD を DRAM よりも大容量なメインメモリとして，DRAM を SSD のキャッシュとして利用する．スケーラビリティ向上のために，vGASNet は分散ファイルシステムにて利用されている協調キャッシングと呼ばれる手法を取り入れている．協調キャッシングとは，キャッシュミス時に他ノードが持っているキャッシュがあればそれを利用するという手法である．我々は TSUBAME-KFC で 32 ノードを利用して vGASNet の性能を測定した．協調キャッシングを取り入れた vGASNet は，シーケンシャルアクセスにおいて 1 ノード利用時と比較して 6.78〜17.9 倍，ランダムアクセスにおいては 4.52〜17.7 倍の台数効果を観測した．本稿では，キャッシュ機構を中心として vGASNet の設計，実装および性能について議論する．

1. はじめに

ステンシル計算 [1,2] やグラフ処理 [3–5]，ソーティング [6] 等において，DRAM の容量を超えるような超大規模な問題を解く需要がある．我々は計算ノードに Flash SSD が備えられているクラスタマシンを仮定し，SSD を活用することで超大規模な問題を高速に解く手法について着目してきた．しかしながら，それらの手法を個別のアプリケーションに適用するのは，技術的に高度な知識を要する．そのため，様々な科学技術計算アプリケーションにおいて，SSD の活用による超大規模な問題を高速に解けるようにするには，専用のライブラリが必要となる．

SSD を利用して超大規模な問題を解くために，SSD を DRAM よりも低速だが大容量なメモリと見立てたライブラリやシステムは数多く存在する [7–12]．しかしながら，それらの研究の多くはクラスタマシンを仮定したものではない．クラスタマシンを仮定した既存研究においても，以下の 3 点全てを考慮したものではない．

- メモリ領域を確保する際，どのノードのメモリに確保するかを開発者が指定できない．つまり，データの局所性を考慮したプログラミングが困難となる点．
- Infiniband や Omni-Path といった低レイテンシ・高バンド幅な高速ネットワークの台頭により，ローカルの SSD よりも遠隔の DRAM の方が読み書きが高速となった点．
- 特定ノード上の SSD 上のデータに対する読み書きが集中した場合，そのノードの処理待ちによってスケーラビリティが低下する可能性が存在する点．

我々は深化するメモリ階層に向けた低レイヤ通信ライブラリの vGASNet を開発している．vGASNet では SSD と DRAM の双方を活用することで，大容量メモリを必要とする問題であっても高速な通信を行うことができる．また協調キャッシング [13] と呼ばれる手法によって負荷を分散させることで，vGASNet では高いスケーラビリティを保持している．

vGASNet は既存の低レイヤ通信ライブラリである GASNet [14] をベースとしている．GASNet は Partitioned Global Address Space（PGAS）といったデータの局所性を考慮したプログラミングが可能となるプログラミングモデルを想定している．そのようなプログラミングモデルを採用したランタイムとして，XcalableMP [15] や UPC++ [16]，Legion [17] 等がある．GASNet はそれらのランタイムの多くでノード間通信に利用されている．また Infiniband や Omni-Path といった高速ネットワークが利用可能な環境であれば，GASNet ではそれらを活用した通信を行うことができる．

vGASNet では SSD を大容量で低速なメインメモリとし

[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

てとして利用する．vGASNet は主に GASNet の片方向通信機能を拡張したものである．GASNet の片方向通信はローカル側とリモート側双方のメモリ領域を指定して行われる．vGASNet ではリモート側として指定されるメモリ領域は SSD 上に確保されたものと仮定している．またこの時 DRAM の一部はキャッシュとして利用される．SSD の持つ大容量と，DRAM の持つ性能を組み合わせて利用することで，大規模な問題を高速に解くことが可能となる．

vGASNet は，性能向上のために協調キャッシングと呼ばれる手法を取り入れた．協調キャッシングとは，あるノードが必要としているデータを，他のノードがキャッシュとして保持している場合，そのキャッシュを利用するというものである．協調キャッシングの導入により，データ転送によって単一ノードにかかる負荷を分散するだけでなく，キャッシュの転送元となるノードのネットワーク帯域を活用することで高スループットを得ることができる．

本稿では，vGASNet 設計や性能について，キャッシュ機構を中心に議論する．

2. GASNet

2.1 GASNet の概要

本章では，vGASNet のベースとなった GASNet について説明する．GASNet とは，Partitioned Global Address Space（PGAS）と呼ばれるプログラミングモデルによるランタイムライブラリのために開発された低レイヤ通信ライブラリである．

Infiniband や Omni-Path 等の高速ネットワークが利用できる環境では，GASNet は Verbs 等の通信 API を利用すること各種ハードウェアが提供する機能を活用して通信を行うことができる．

GASNet は Core API と Extended API の二種類の API によって構成されている．Core API は通信の初期化等の基本的な通信を行う関数や，Active Message [18] と呼ばれる形式による通信インターフェースを提供する．Extended API は片方向通信や Collective 通信といった高機能通信を提供している．

2.2 Core API

GASNet は Core API を用いることで，最低限の通信が可能となる．通信は `gasnet_init()` によって初期化され，`gasnet_exit()` によって終了処理が行われる．

Active Message とは，メッセージを片方向通信によってノード同士で送受信する通信形式である．メッセージの受信側は，事前に登録したハンドラ関数を呼び出してメッセージに対応する処理を行う．どのハンドラ関数を呼び出すかは，送信側のノードがメッセージのヘッダとして付加した情報に基いて決定される．

GASNet では，Core API によって提供されている `gasnet_AMRequestMedium0()` 等の関数を利用してメッセージを送信する．ハンドラ関数は `gasnet_attach()` によって登録される．

2.3 Extended API

GASNet の Extended API は片方向通信や Collective 通信等の高機能通信を提供している．現行の vGASNet は Collective 通信をサポートしておらず，片方向通信についても一部未対応である．ここでは，現行の vGASNet がサポートしている片方向通信に関するものについて述べる．

Verbs 等の一部の通信 API が提供する片方向通信では，通信の対象となるメモリ領域がページサイズにアラインメントされている必要がある．GASNet の片方向通信用関数で指定される遠隔ノードのメモリは GASNet によって確保された領域に収まっている必要がある．このメモリ領域は `gasnet_attach()` が呼び出されたタイミングで確保され，`gasnet_getSegmentInfo()` によってプログラムはその領域の先頭アドレスとサイズを認知することができる．

GASNet の片方向通信機能は，2 種類の関数群を呼び出すことによって実行される．1 つの関数群は Put である．Put はローカルのメモリ領域からリモートのメモリ領域に対してコピーを行うものである．片方向通信を行うもう 1 つの関数群は Get である．Get はリモートのメモリ領域からローカルのメモリ領域に対してコピーを行うものである．双方ともにローカルのメモリ領域の先頭アドレス，リモートのノード番号とメモリ領域の先頭アドレスおよびコピーする領域のサイズを関数の引数として指定する．

GASNet の片方向通信用関数には，通信の対象となるノードローカルのメモリ領域はアラインメントされていることを想定したもの（例：`gasnet_put()`）と，アラインメントされていない場合にも対応したもの（例：`gasnet_put_bulk()`）の二種類がある．

一方で GASNet の片方向通信機能はブロッキング通信を行うものと，ノンブロッキング通信を行うものの 2 種類にも大別される．その上ノンブロッキング通信には専用のハンドラが返ってくる関数（例：`gasnet_nb_put()`）と，ハンドラを返さない関数（例：`gasnet_nbi_put()`）の 2 種類に大別される．ハンドラが返ってくるノンブロッキング通信は `gasnet_wait_syncnb()` によって完了を待機することができる．ハンドラが返ってこないノンブロッキング通信は `gasnet_wait_syncnbi_all()` によって，それまでに行われた（ハンドラが返ってこない）ノンブロッキング通信の全てが完了するのを待機することができる．

3. vGASNet

3.1 vGASNet の概要

本章では，我々の開発したスケーラブルな低レイヤ通信ライブラリ vGASNet について述べる．vGASNet は GASNet

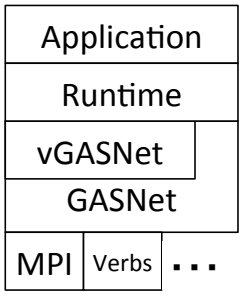

図 1　vGASNet のソフトウェアスタック

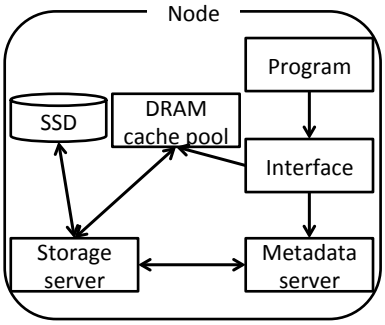

図 2　vGASNet の構造

の片方向通信を対象として，設計と実装に拡張を加えたものである．片方向通信の対象となるリモートのメモリ領域は SSD 上に存在し，GASNet の一部の関数と互換性のある関数（例 vgasnet_get()）によって読み書きを行う．

vGASNet を利用した場合のソフトウェアスタックを図 1 に示す．プログラムが片方向通信を利用する時は vGASNet の提供する関数を利用するが，Active Message の送受信は従来の GASNet が提供する関数を利用する．また vGASNet は内部的には GASNet を利用して通信を行う．このようにすることで，Verbs といった GASNet で利用可能な高速ネットワークを活用した通信は vGASNet でも利用可能となる．

図 2 は vGASNet の構造を表したものである．vGASNet は SSD やキャッシュを管理するストレージサーバと，データのメタデータを管理するメタデータサーバ，それらに対して指示を行うインターフェースによって構成されている．ストレージサーバは SSD や DRAM のキャッシュプールを管理している．メタデータサーバは SSD 上のページと，それらのページのキャッシュを持つノードを紐付ける役割を担っている．インターフェースによってそれら 2 種類のサーバを操作する．vGASNet を利用する各種プログラムは片方向通信を中心としたインターフェースを用いてそれらのサーバやキャッシュ，および SSD を透過的に操作する．

現行バージョンの vGASNet はスレッドセーフではない．GASNet はスレッドセーフで実行することを可能とする設定が存在する．この設定は GASNet を利用する各種ランタイムライブラリのコンパイル時に指定される．一方で，UPC++や XcalableMP といった GASNet を利用している複数のランタイムライブラリにおいて，そのような設定は現在利用されていないことが我々の調査で明らかとなった．我々はそのようなランタイムライブラリにて vGASNet が利用されることを想定している．

3.2　メモリの確保

vGASNet は計算ノードローカルの SSD を DRAM よりも低速かつ大容量なメモリとして利用する．より具体的には，SSD 上のファイルをメモリ空間と見立ててページ単位で読み書きを行う．このファイルは vgasnet_attach() したタイミングで O_DIRECT フラグをつけて open() される．

vgasnet_attach() が呼び出されると，gasnet_attach() を呼び出される．この時に GASNet 内部に確保されるメモリ領域のポインタを gasnet_getSegmentInfo() によって取得して，キャッシュ領域として扱う．

vgasnet_getSegmentInfo() によって vGASNet からプログラムに通知されるメモリ領域は，malloc() 等によって確保されるメモリとは重複しないことが望まれる．vGASNet の vgasnet_init() では，mmap() によって仮想メモリ空間を予約する．このとき確保される仮想メモリ空間の大きさは SSD に記録できる最大の容量とし，更に PROT_NONE フラグを立てることで読み書きを禁止とする．その上で，vGASNet の vgasnet_getSegmentInfo() は，その仮想メモリ空間の先頭アドレスと確保したサイズを返す．vGASNet が SSD 上に確保するメモリ領域を仮想メモリ空間上に確保することで，malloc() 等によって確保されるメモリ領域と区別することを可能としている．

3.3　キャッシュ

3.3.1　キャッシュ機構の概要

vGASNet では全てのノードが任意のページのキャッシュを持つ．また，各ページは複数のノードからキャッシュされうる．各ノードが持っているキャッシュの管理はそのノードのストレージサーバが行う．キャッシュプールは固定長のページ単位で区切られており，LRU キューによって管理されている．また各ノードのメタデータサーバは，そのノード上の SSD が保持しているデータのキャッシュの所在情報を持っている．

vGASNet では，あるノードにおけるデータアクセスがキャッシュミスした時に同一ページへのキャッシュを他ノードが保持していれば，データアクセスを行っているノードはそのキャッシュを利用する．このメカニズムは分散ファイルシステムで利用されているもので，協調キャッシングと呼ばれている．協調キャッシングによって，キャッシュミスによる SSD へのアクセス回数を減らすことが可能となる．その上で，ページの内容を送信するノードを分

散させることで，vGASNet はスケーラビリティを向上することができる．

3.3.2 ページの読み込み

あるノード Na があるページ P の内容を読み込もうとしてキャッシュミスしたとして，その場合の手順を以下に示す．

(1) ノード Na はページ P の内容を SSD 上に持つノード Nb のメタデータサーバに問い合わせを行う．

(2) Nb のメタデータサーバは，P についてのキャッシュの所在を確認する．
  (a) キャッシュを持っているノード Nc が存在すれば，Nc のストレージサーバに問い合わせを転送する．Nc は P のキャッシュの内容を Na のキャッシュに転送する．
  (b) キャッシュを持っているノードが存在しなければ，Nb は自身のストレージサーバにアクセスする．Nb のストレージサーバは SSD から P の内容を，Na のキャッシュに転送する．

(3) Na はキャッシュを受け取ると，Nb のメタデータサーバに対してキャッシュを得た旨の通知を行う．

Na，Nb，Nc 間で発生する問い合わせは Core API を用いた Active Message をベースに行われる．P の内容を転送する時のみ `gasnet_put()` による片方向通信が行われる．

通信の行き違いによって，Nc でキャッシュが破棄されたにも関わらず Nb から問い合わせが転送されることがある．そのような場合，Nc は Nb に対して転送された問い合わせを返却することで，再度キャッシュの所在情報を確認するよう促す．

メタデータサーバから問い合わせを転送するノードの決定方法は重要な課題である．もし，問い合わせを転送するノードが常に同一であったならば，そのノードに対して負荷がかかることになる．また，古くにキャッシュを取得したノードでは，そのキャッシュを破棄している可能性がある．そのような場合，先に述べた所在情報の再確認による追加の通信が発生する．

vGASNet ではメタデータサーバから問い合わせを転送するノードを，図 3 に示すスタックベースのデータ構造によって決定する．このデータ構造は，メタデータサーバが保持しており，各ページにつき 1 つずつ割り当てられている．あるノードがキャッシュを持ったことをメタデータサーバが通知されたとする．キャッシュされたページに対応したスタックに，ノードの情報がプッシュされる．キャッシュの読み込みが行われる時，このスタックの先頭を参照し，対応するノードに問い合わせを転送する．アクセスされた先頭の要素はスタックの先頭から末尾に移される．キャッシュを転送するストレージサーバがラウンドロビンに決定されるため，負荷を分散することができる．更に最近キャッシュを取得したノードから優先されるため，通信の行き違いも抑止される．

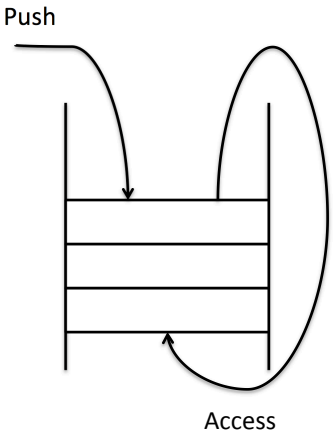

図 3 キャッシュ情報管理に利用されているデータ構造

どのノードもキャッシュしていない同一ページへのアクセスが短期的に行われた場合，SSD に対するアクセスが集中することがある．vGASNet では，どのノードもキャッシュされていないページへのアクセスでは手順（2）から（3）までの過程は不可分に行われる．

3.3.3 ページへの書き込み

あるノード Na が，あるページ P にデータを書き込みたいとした時の手順を以下に示す．

(1) 3.3.2 節と同様の手法で，Na のキャッシュプールにデータを転送する．しかし，Na が持つ P のキャッシュが最新であれば実際のデータ転送は実施されない．また Na に対してデータ転送完了の通知は行わず，代わりに Nb のメタデータサーバに対して通知を行う．

(2) Nb のメタデータサーバは，P のキャッシュを持つ Na 以外の全ノードに対して，キャッシュを無効化させる命令を発行する．

(3) 無効化命令の発行後，Nb のメタデータサーバより Na に対して P の内容を送信し終えた旨の通知をする．

(4) Na は通知を受け取った後に P のキャッシュにデータを書き込む．

更に false sharing に対応させるため，手順（1）から（3）までの過程は同一ページに対して不可分に行われる．

3.3.4 キャッシュの掃き出し

あるノード Na が，あるページ P のキャッシュを持っているとする．ここで P はノード Nb の SSD にオリジナルのデータがある存在するものとする．Na の持つ P のキャッシュが dirty であり，LRU によって P のキャッシュが掃き出しの対象となった場合の手順を以下に示す．

(1) Na は Nb にキャッシュを掃き出す旨の問い合わせを行う．発行される問い合わせにはキャッシュの内容そのものは含まれていない．

(2) Nb は発行された問い合わせを元に，Na からキャッシュの内容を読み込む．

(3) Nb は Na から取得した P のキャッシュの内容を SSD に書き込む．

(4) Nb は書き込み完了の旨を Na に通知する．

(5) Na が持っている P のキャッシュを無効化する．

Nb が Na が持つ P のキャッシュの内容を取得するときのみ `gasnet_get()` による片方向通信を使用し，その他の通信は Active Message による通信を行う．

Na の持つ P のキャッシュが dirty でない場合に，P のキャッシュが掃き出しの対象となった場合は，SSD に対するキャッシュの書き出しを行わない．上記手順のうち（4）と（5）のみを行う．

3.4 片方向通信

本節では vGASNet の片方向通信機能について述べる．GASNet と同様に，vGASNet でも `vgasnet_put()` のような片方向通信を提供する．vGASNet ではリモート側として指定可能なメモリ領域は，3.2 節で述べた SSD 用に擬似的に確保されたメモリ空間である．

vGASNet では片方向通信機能用の関数が呼び出されると，ハンドラをアサインする．このハンドラは `vgasnet_init()` によって通信が初期化されるタイミングで配列として確保される．vGASNet のハンドラは，それぞれの通信に必要となるキャッシュと紐付いている．`vgasnet_get_nb()` のようなノンブロッキングの関数では，ハンドラへのポインタを戻り値とする．

`vgasnet_get()` のように read only な操作である場合，まず該当部分のキャッシュを探索する．もし該当キャッシュが無効化されているならば，3.3.2 節で示した方法によってキャッシュを自ノードに転送する．転送が完了したならば，当該関数の引数として指定されたポインタに対してキャッシュの内容の指定された箇所をコピーする．指定された操作に関する全てのキャッシュに関するコピーが終わると，ハンドラのアサインを解除する．

`vgasnet_put()` のようにキャッシュに書き込む必要のある場合は，3.3.3 節で示した方法によってキャッシュを自ノードに転送する．転送が完了したならば，当該関数の指定されたポインタからキャッシュに書き込みを行う．指定された操作に関する全てのキャッシュに関するコピーが終わると，ハンドラのアサインを解除する．

表 1 TSUBAME-KFC 1 ノードあたりの性能

CPU	Intel Xeon E5-2620 v2
DRAM	64 GB
SSD	Intel DC S3500
Read BW	510 MB/s
Write BW	400 MB/s
Size	480 GB
File System	XFS
Network	Infiniband 4X FDR
OS	CentOS 7.3

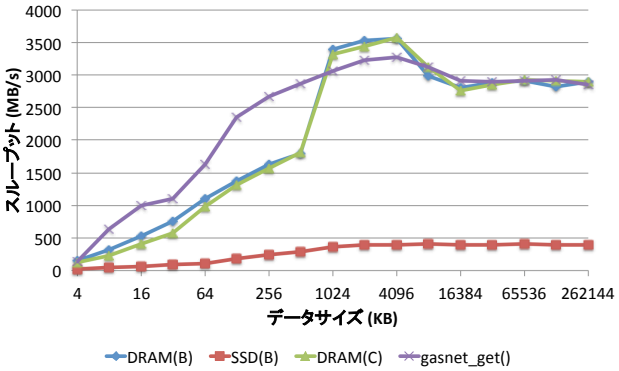

図 4 予備評価の結果

4. 性能評価

4.1 評価環境

我々は vGASNet を実装し，その性能を評価した．性能評価は TSUBAME-KFC/DL [19] 上で実施された．1 ノードあたりの性能を表 1 に示す．本実験にて我々は GASNet のバージョン 1.28.2 を選択した．キャッシュのプール領域は 1 ノードにつき 16 GB とした．

比較を行うために，協調キャッシングを行わない vGASNet の実装を用意した．この実装は，あるノードがページにアクセスする際，そのノードが保持しているキャッシュとそのページを SSD 上に保持するノードのキャッシュのみを参照する．SSD からページを読み込む場合はそのページを要求したノードだけでなく，その SSD を持っているノードにもキャッシュされる．一貫性を保証するため，バリア同期時や同一ページに対する他ノードの書き込みが発生したタイミングで，SSD にキャッシュの書き戻しを行う．

4.2 予備評価

vGASNet におけるページサイズを決定するために，3 ノードを用いた予備実験を行った．我々はまず vGASNet のキャッシュ読み込みを模したプログラムを GASNet を用いて作成した．ノード A がノード B の SSD 上にあるデータを要求したとする．更にそのデータはノード B とノード C にそれぞれキャッシュされていたとする．この時，ノード B のキャッシュからデータを取ってきた時のスループットを DRAM(B)，ノード C のキャッシュからデータを取っ

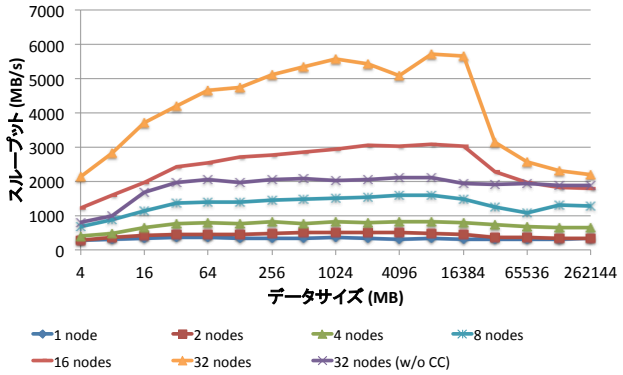

図 5　シーケンシャルアクセスの性能

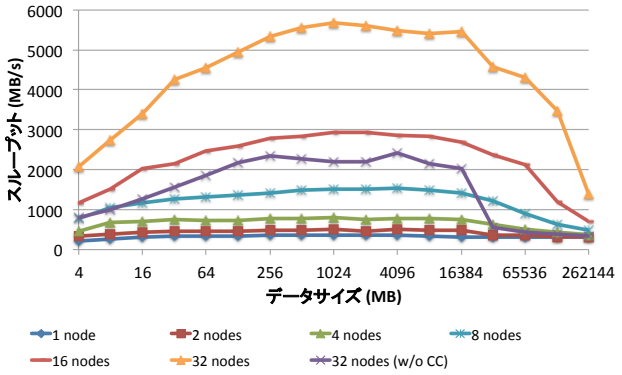

図 6　ランダムアクセスの性能

てきた時のスループットを DRAM(C) と表現する．比較のため，キャッシュミスをして SSD にアクセスした時のスループット SSD(B) と，gasnet_get() の性能も測定した．

DRAM(C) は SSD(B) と比較して，最小でも 4.8 倍，最大で 9.1 倍のスループットが観測された．これは SSD の帯域幅と比較して，Infiniband 4X FDR の帯域幅が大きいことによる．既存の協調キャッシングは HDD によって構成された分散ファイルシステムに対するものであった．この実験によって，SSD を用いて構成された通信ライブラリにおいても協調キャッシングが有効となりうることが示された．

DRAM(B)，DRAM(C) は，gasnet_get() と比較するとそれぞれ最大 50 %，64 %の性能低下が観測された．この時のデータサイズはともに 8 KB の時である．双方向的にデータの送受信を行っていることによって，通信に遅延が生じているためである．この時にノード間でやり取りされているメッセージのサイズは，送受信されようとするデータのサイズに依らず一定である．したがって，データサイズが大きくなると，メッセージの送受信によるオーバーヘッドは相対的に小さくなる．実際にデータサイズが 1 MB 以上となると，DRAM(B)，DRAM(C) のスループットと gasnet_get() のスループットに有意な差が見られなかった．

データサイズが 4 MB の時に DRAM(B)，DRAM(C) はスループットがそれぞれ最大となっており，それぞれ 3559 MB/s，3567 MB/s であった．この結果より，以降の実験における vGASNet のページサイズを 4 MB とした．

4.3　シーケンシャルアクセスの性能

我々は vGASNet におけるシーケンシャルアクセスの性能を評価するためのプログラムを作成した．このプログラムでは，rank0 上に put された，ただ 1 つのデータ領域に対して複数ノードから同時にアクセスされる．データアクセスは 4 MB 単位でシーケンシャルに vgasnet_get() されることで行われる．

図 5 はプログラム内で観測された vgasnet_get() の性能である．横軸はアクセスされたデータ領域のサイズを表し，縦軸はスループット（ノード数 × データ領域のサイズ / 転送時間）を表している．

1 ノード使用時と比較して 32 ノード使用時は最低でも 6.78 倍，最高で 17.9 倍のスループットが観測された．また協調キャッシングを行わない vGASNet の実装で 32 ノード動作させた時（32 nodes (w/o CC)）とは最低でも 1.17 倍，最高で 2.93 倍のスループットが観測された．協調キャッシングでは，ページの内容を転送するノードが分散される．このため，協調キャッシングを行う vGASNet では高いスケーラビリティを持つことができる．

一方で，データサイズが 16 GB を上回ると，vGASNet は性能を低下させてしまうという現象も観測された．原因は現在調査中ではあるが，LRU キューから追い出されたことによってキャッシュが無効化された時の挙動にあると推測される．vGASNet ではキャッシュが無効化されると，キャッシュされているページを SSD 上に持つノードのメタデータサーバに対して通知を行う．今回の実験においては，全てのノードにおいて同時多発的にキャッシュが溢れるために，rank0 に通知が集中してしまったことが性能低下に繋がった．

4.4　ランダムアクセスの性能

我々は vGASNet におけるランダムアクセスの性能を評価するためのプログラムも作成した．このプログラムでも，rank0 上に put された，ただ 1 つのデータ領域に対して複数ノードから同時にアクセスされる．しかしデータアクセスは 4 MB 単位でランダムに vgasnet_get() されることで行われる．

図 6 はプログラム内で観測された vgasnet_get() の性能である．1 ノード使用時と比較して 32 ノード使用時は最低でも 4.52 倍，最高で 17.7 倍のスループットが観測された．またシーケンシャルアクセスと同様に，データサイズが 16GB を超えると性能が低下してしまう現象が確認さ

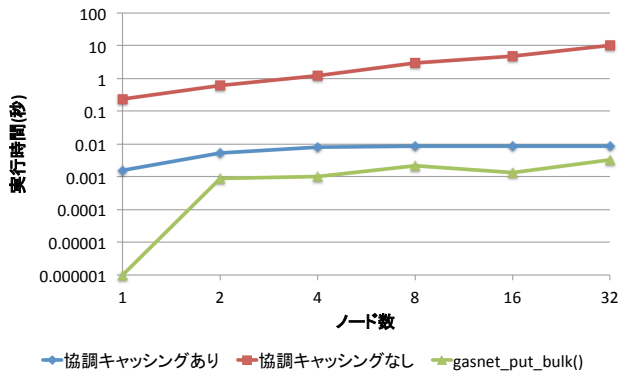

図 7　False sharing 発生時の性能

れた．

協調キャッシングを行わない vGASNet の実装で 32 ノード動作させた場合と比較して，協調キャッシングを行った場合はスループットは最低でも 2.28 倍となり，特にデータサイズが 16 GB を上回ると 4.04 倍以上となる．シーケンシャルアクセスと同様に，ページの内容を転送するノードが分散されていることが性能向上の要因の 1 つである．一方で，ランダムアクセスでは，協調キャッシングによって 1 ノードが利用できるキャッシュ空間が広がったことも性能向上の要因の 1 つとして挙げられる．本実験ではデータサイズが 16 GB を上回るとキャッシュアウトが起こる．そしてキャッシュアウトされたページへのアクセスが発生するとキャッシュミス回数が増大する．協調キャッシングによってキャッシュ空間が広くなると，キャッシュミス回数の増大が緩和される．

4.5　False sharing 発生時の性能

協調キャッシングの導入における，一貫性制御によって発生するオーバーヘッドを調査するために，我々は false sharing が発生するプログラムの作成を行った．このプログラムは，rank0 上の同一ページに対して各ノードから 1 バイトの書き込みを 10 回ずつ行う．書き込みを開始時から全てのノードから書き込まれ，バリア同期されるまでの時間を測定した．書き込みは `vgasnet_put_bulk()` を用いている．また，vGASNet を使用せず `gasnet_put_bulk()` によって書き込みを行うプログラムも作成し，比較した．

プログラムの実行時間を図 7 に示す．協調キャッシングを行わないバージョンの vGASNet では，最悪で (ノード数 $\times 10 + 1$) 回 SSD への書き戻しが発生する．一方で，協調キャッシングを行うバージョンの vGASNet では SSD への書き戻しが発生しない．これによって，協調キャッシングを行うバージョンの vGASNet では協調キャッシングを行わないバージョンと比較して計測時間を最大でも 0.89 %程度に抑えることができた．

`vgasnet_put_bulk()` は `gasnet_put_bulk()` と比較して，1 ノード使用時では 1000 倍以上の実行時間がかかっていることが分かった．vGASNet では最初の書き込みを行う際にキャッシュミスが発生し，SSD からの読み込みが発生したためである．32 ノード使用時には実行時間の倍率は 2.59 倍程度にまで小さくなっている．これはノード数を増やすと SSD からの読み込みによるオーバーヘッドが相対的に小さくなるためである．

5.　関連研究

我々が以前に発表した ComEx-PM [7] は既存の PGAS 向け低レイヤ通信ライブラリである ComEx [20] に互換性を持たせたまま，メモリ階層深化に対応させたものである．ComEx-PM は MPI を利用して通信を行うため，Infiniband や Omni-Path といった高速ネットワークを十分に活用できず，また協調キャッシングの導入も行っていない．

HHRT [2] はメモリ階層深化に向けた MPI 互換のライブラリである．vGASNet と同様に，HHRT も各計算ノードがローカルの SSD を持っているクラスタマシンを想定している．現状の HHRT は `MPI_Put()` 等の片方向通信機能をサポートしておらず，また協調キャッシングも行っていない．

SSD を DRAM よりも低速かつ大容量なメインメモリとして見立てたプログラミングモデルを提供しているライブラリは多数存在する [8–12]．これらのライブラリは，クラスタマシンを仮定したものではないため，協調キャッシングを利用していない．一方で，これらのライブラリで行われているメモリ管理手法における本研究への応用可能性は検討の余地がある．

Chen ら [21,22] は PGAS 言語である UPC のキャッシュ機構を提案した．また，Ferguson ら [23] は PGAS 言語である Chapel に対してを導入した．しかしながら，これらは超大規模問題に対応させたものではない．

Watkins ら [24] は，Legion [17] というランタイムライブラリを様々なメモリ階層に対応させた．彼らの研究では，開発者の与えたデータ配置情報をもとに，分散ファイルシステムや計算ノードローカルの SSD などを透過的に利用する方法を提案している．しかしながら，この研究での実験は，分散ファイルシステムと計算ノードローカルの DRAM のみを使用し，計算ノードローカルの SSD は利用していない．

協調キャッシングは Dahlin ら [13] によって提案され，多数の分散ファイルシステムへの導入が試みられている [25–31]．特に Sasaki らの研究 [30] では，ファイルの内容を保持する I/O サーバを計算ノードが請け負うことや，計算ノード間が Infiniband によって接続されていることを仮定しており，本研究との類似点が多い．

INFINISWAP [32] は，メモリ階層深化に向けたリモートページングシステムである．これは Linux カーネルの

メモリ管理部分を書き換えることで，各ノードの SSD と DRAM の双方をメインメモリとして扱うことを可能としている．INFINISWAP では自ノードの DRAM 容量を超える場合，他ノードの DRAM をキャッシュとして利用する．またノード間の通信は RDMA を利用しており，Infiniband のバンド幅を活用することができる．このようなリモートページングシステムで想定されているアプリケーションは 1 ノードを主体としたものであり，本研究のように多ノードを主体としたアプリケーションではない．

6. まとめ

我々はメモリ階層深化に対応したスケーラブルな低レイヤ通信ライブラリ vGASNet の開発を行っている．本稿では，vGASNet の設計と実装について，特にキャッシュ機構を中心に述べた．

vGASNet のキャッシュ機構は協調キャッシングを導入することで，性能向上を図っている．協調キャッシングとは，自ノードに存在しないキャッシュラインにアクセスする場合，他ノードがキャッシュしていればそれを利用するというメカニズムである．

1 ノード使用時と比較して，32 ノード使用時はシーケンシャルアクセスでは 6.78〜17.9 倍，ランダムアクセスでは 4.52〜17.7 倍の台数効果が実験によって観測された．また，協調キャッシングの効果によって false sharing に対する性能が向上したことも確認できた．

7. 今後の課題

分散ファイルシステムにおける協調キャッシングの研究では，様々な性能改善手法がこれまでに提案されている．例えば N-Chance Forwarding [13] は DRAM の空間効率を上げることで，キャッシュヒット率の向上を可能としている．Hint-based Cooperative Caching [28] はキャッシュの所在情報を分散させることで，キャッシュ問い合わせによる負荷を分散させることができる．vGASNet においてもこれらの手法を導入し，その効果を測定する余地は十分に考えられる．

SSD の読み書きが大きなオーバーヘッドとなっていることが分かっている．libaio を用いた非同期 IO を用いれば，SSD の読み書きを隠蔽できる可能性がある．我々は非同期 IO による高速化手法を vGASNet に組み込むことを予定している．

現行の vGASNet は全ての GASNet 関数に対応したものではない．しかしながら，UPC++等の一部 PGAS ランタイムライブラリの動作において最低限必要な関数は備わっている．vGASNet 上でこれらのランタイムライブラリを動作させ，アプリケーションベンチマークを動作させることを我々は計画している．また，必要があれば vGASNet 側の仕様を一部変更し，vGASNet と各種ランタイムのコ・デザインを試みる．

NVMe-over-Fabrics（NVMf）[33] は，DRAM を経由することなく遠隔ノードから直接 SSD 上のデータにアクセスする機能である．NVMf は iSCSI 等といった既存の遠隔ストレージプロトコルよりも低レイテンシ高スループットで通信ができるため，分散ストレージへの導入が期待されている [34]．vGASNet においても，SSD と遠隔ノードの間での通信において NVMf を導入する余地がある．

近年，GASNet を用いて開発されたランタイムライブラリ等において，タスク並列への応用が期待されている [17, 35–38]．vGASNet の応用としても，大規模データを扱うタスク並列アプリケーションの開発が考えられる．我々は，vGASNet と既存の PGAS ランタイムライブラリを利用した大規模データ用のタスク並列ベースのフレームワークの開発を検討している．

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] 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)* (2016).

[3] Iwabuchi, K., Sato, H., Yasui, Y., Fujisawa, K. and Matsuoka, S.: NVM-based Hybrid BFS with Memory Efficient Data Structure, *Proceedings of the 2014 IEEE International Conference on Big Data (Big Data '14)* (2014).

[4] Zheng, D., Mhembere, D., Burns, R., Vogelstein, J., Priebe, C. E. and Szalay, A. S.: FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs, *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15)* (2015).

[5] Iwabuchi, K., Sallinen, S., Pearce, R. A., Essen, B. V., Gokhale, M. and Matsuoka, S.: Towards a Distributed Large-Scale Dynamic Graph Data Store, *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW '16)* (2016).

[6] Sato, H., Mizote, R., Matsuoka, S. and Ogawa, H.: I/O Chunking and Latency Hiding Approach for Out-of-core Sorting Acceleration using GPU and Flash NVM, *Proceedings of the 2016 IEEE International Conference on BigData (BigData '16)* (2016).

[7] Matsumiya, R. and Endo, T.: PGAS Communication Runtime for Extreme Large Data Computation, *Proceedings of the 2nd International Workshop on Extreme Scale Programming Models and Middleware (ESPM2*

'16) (2016).

[8] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R. and Swanson, S.: NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories, *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)* (2011).

[9] Volos, H., Tack, A. J. and Swif, M. M.: Mnemosyne: Lightweight Persistent Memory, *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)* (2011).

[10] Wang, C., Vazhkudai, S. S., Ma, X., Meng, F., Kim, Y. and Engelmann, C.: NVMalloc: Exposing an Aggregate SSD Store as a Memory Partition in Extreme-Scale Machines, *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12)* (2012).

[11] Schwalb, D., Berning, T., Faust, M., Dreseler, M. and Plattner, H.: nvm_malloc: Memory Allocation for NVRAM, *Proceedings of the 6th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS '15)* (2015).

[12] Bhandari, K., Chakrabarti, D. R. and Boehm, H.-J.: Makalu: Fast Recoverable Allocation of Non-volatile Memory, *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '16)* (2016).

[13] Dahlin, M. D., Wang, R. Y., Anderson, T. E. and Patterson, D. A.: Cooperative Caching: Using Remote Client Memory to Improve File System Performance, *Proceedings of the 1st USENIX Conference on Operating System Design and Implementation (OSDI '94)* (1994).

[14] Lawrence Berkeley National Laboratory: GASNet Communication System, `https://gasnet.lbl.gov`.

[15] 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).

[16] Zheng, Y., Kamil, A., Driscoll, M. B., Shan, H. and Yelick, K.: UPC++: A PGAS extension for C++, *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS '14)* (2014).

[17] Bauer, M., Treichler, S., Slaugher, E. and Aiken, A.: Legion: Expressing Locality and Independence with Logical Regions, *Proceedings of the ACM/IEEE 25th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)* (2012).

[18] von Eicken, T., Culler, D. E., Goldstein, S. C. and Schauser, K. E.: Active Messages: a Mechanism for Integrated Communication and Computation, *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA '92)* (1992).

[19] Endo, T., Nukada, A. and Matsuoka, S.: TSUBAME-KFC: a Modern Liquid Submersion Cooling Prototype towards Exascale Becoming the Greenest Supercomputer in the World, *Proceedings of the 2014 IEEE International Conference on Parallel and Distributed Systems (ICPADS '14)* (2014).

[20] Daily, J., Vishnu, A., Palmer, B., van Dam, H. and Kerbyson, D.: On The Suitability of MPI as a PGAS Runtime, *Proceedings of the IEEE International Conference on High Performance Computing (HiPC '14)* (2014).

[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] Ferguson, M. P. and Buettner, D.: Caching Puts and Gets in a PGAS Language Runtime, *Proceedings of the 9th International Conference on Partitioned Global Address Space Programming Models (PGAS '15)* (2015).

[24] Watkins, N., Jia, Z., Shipman, G., Maltzahn, C., Aiken, A. and McCormick, P.: Automatic and Transparent I/O Optimization With Storage Integrated Application Runtime Support, *Proceedings of the 10th Parallel Data Storage Workshop (PDSW '15)* (2015).

[25] Xu, Y. and Fleisch, B. D.: NFS-cc: Tuning NFS for Concurrent Read Sharing, *International Journal of High Performance Computing and Networking*, Vol. 1, No. 4, pp. 203–213 (2004).

[26] Batsakis, A. and Burns, R.: NFS-CD: Write-Enabled Cooperative Caching in NFS, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 3, pp. 323–333 (2007).

[27] Annapureddy, S., Freedman, M. J. and Mazières, D.: Shark: Scaling File Servers via Cooperative Caching, *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI '05)* (2005).

[28] Sarkar, P. and Hartman, J.: Hint-based Cooperative Caching, *ACM Transactions on Computer Systems*, Vol. 18, No. 4, pp. 387–419 (2000).

[29] Hwang, I.-C., Jung, H., Maeng, S.-R. and Cho, J.-W.: Design and Implementation of the Home-Based Cooperative Cache for PVFS, *Proceedings of the 5th International Conference on Computational Science (ICCS '05)* (2005).

[30] Sasaki, S., Matsumiya, R., Takahashi, K., Oyama, Y. and Tatebe, O.: RDMA-based Cooperative Caching for a Distributed File System, *Proceedings of the 21st IEEE International Conference on Parallel and Distributed Systems (ICPADS '15)* (2015).

[31] 新井淳也，石川　裕，堀　敦史：RDMA をサポートするクラスタ向けの軽量協調キャッシング，情報処理学会研究報告，Vol. 2012-HPC-137, No. 27, pp. 1–8 (2012).

[32] Gu, J., Lee, Y., Zhang, Y., Chowdhury, M. and Shin, K. G.: Efficient Memory Disaggregation with INFINISWAP, *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)* (2017).

[33] NVM Express: NVM Express over Fabric Revision 1.0, `http://www.nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1_0_Gold_20160605-1.pdf` (2016).

[34] Guz, Z., Li, H., Shayesteh, A. and Balakrishnan, V.: NVMe-over-Fabrics Performance Characterization and the Path to Low-Overhead Flash Disaggregation, *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR '17)* (2017).

[35] Kumar, V., Zheng, Y., Cavè, V., Budimlì, Z. and Sarkar, V.: HabaneroUPC++: a Compiler-free PGAS Library,

Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models (PGAS '14) (2014).

[36] Jacquelin, M., Zheng, Y., Ng, E. and Yelick, K.: An Asynchronous Task-based Fan-Both Sparse Cholesky Solver, *CoRR*, Vol. abs/1608.00044 (online), available from ⟨http://arxiv.org/abs/1608.00044⟩ (2016).

[37] Grossman, M., Kumar, V., Budimlič, Z., and Sarkar, V.: Integrating Asynchronous Task Parallelism with OpenSHMEM, *Proceedings of the Workshop on OpenSHMEM and Related Technologies (OpenSHMEM '16)* (2016).

[38] Ozog, D., Kamil, A., Zheng, Y., Hargrove, P., Hammond, J. R., Malony, A., de Jong, W. and Yelick, K.: A Hartree-Fock Application using UPC++ and the New DArray Library, *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS '16)* (2016).