Vol.2012-HPC-135 No.7
2012/8/1

軽量マルチスレッディング向け 大域アドレス空間ライブラリ

秋山 茂樹[1,a)]　田浦 健次朗[1,b)]

概要：分散メモリ型並列計算機における並列プログラミングの生産性を改善することを目的として，Partitioned Global Space (PGAS) モデルに基づくプログラミング言語およびライブラリが提案されてきている．一方で，PGAS モデルに基づくプログラミング環境は，データの配置を動的に変えることができないことや，データアクセス時にキャッシュが行われないことなど，大域的な軽量スレッドのノード間移動が発生するような軽量マルチスレッド環境において，必ずしも適したプログラミングモデルというわけではない．本論文では，localize と呼ぶ，軽量マルチスレッド処理系において有用な大域アドレス空間 API を提案し，localize に基づく大域アドレス空間ライブラリの実装手法について説明する．

Global Address Space Library for Lightweight Multithreading

Shigeki Akiyama[1,a)]　Kenjiro Taura[1,b)]

Abstract: In order to improve productivity of parallel programming on distributed memory machines, Partitioned global address space (PGAS) model is proposed. However, programming environments based on PGAS model are not always suitable for lightweight multithreading which supports thread migration between processes. For example, data placement cannot change dynamically, and cache support is limited. In this paper, we propose a global address space library supporting an operation, named 'localize', which is useful on lightweight multithread framework.

1. はじめに

現在主流の並列プログラミングモデルである MPI [1] は，並列計算機のハードウェアを単純に抽象化したものであり，プログラマは各プロセッサで行う計算処理と，計算に必要なデータを必要なプロセッサに転送する通信処理をそれぞれのプロセッサについて記述しなければならない．このようなプログラミングモデルにおいては，並列プログラムに対して，使用する並列計算機に関する知識に基づいた最適化を施すことができ，並列計算機の性能を最大限まで引き出すことができる一方で，使用する計算機固有の高速化を含む多くの高速化処理をプログラマが記述しなければならない．このことから，開発したプログラムがポータビリティに欠けることや，高速化のためのプログラム記述を数多く行わなければならないことなど，プログラム開発のコストが非常に大きいという問題が存在する．

この問題に対する一つの解決策として，cache-oblivious algorithm [2] と呼ばれる，問題を再帰的に小問題に分割して計算を行うアルゴリズムを軽量マルチスレッド処理系 [3], [4], [5] を用いて並列化する手法が提案されている [6]．cache-oblivious algorithm は，データアクセスの局所性を意識した再帰的な問題分割により，複数のメモリ階層 (L1/L2/L3 キャッシュ，メインメモリ) を効率良く利用することが可能なアルゴリズムである．我々の研究グループにおいては，MassiveThreads [7] と呼ぶ共有メモリ並列計算機向けの軽量マルチスレッド処理系を開発しており，実際に計算機の最大性能に迫る cache-oblivious な並列プログ

[1] 東京大学
The University of Tokyo
[a)] akiyama@eidos.ic.i.u-tokyo.ac.jp
[b)] tau@eidos.ic.i.u-tokyo.ac.jp

ラムを実装可能であることを確認している．

一方で，分散メモリ並列計算機においては，密行列積の cache-oblivious algorithm に対する実装手法が研究されており [8] ，実際に従来の手法を超える性能を達成しているものの，cache-oblivious algorithm を高性能に実行できる汎用的なランタイムシステムに関する研究はほとんどないのが現状である．

我々の研究グループでは，分散メモリ型並列計算機においてプロセスをまたがった動的負荷分散が可能な軽量マルチスレッド処理系と，キャッシュをサポートする大域アドレス空間 (Global Address Space, GAS) 処理系を組み合わせることによって，分散メモリ型並列計算機において，cache-oblivious algorithm を高性能に実行できる処理系を実現できるのではないかと考え，研究を行っている．

本研究では，その一環として，軽量マルチスレッド処理系からの利用を想定した大域アドレス空間ライブラリを提案する．提案するライブラリの特徴は，(1) 多くの PGAS 言語と異なり，C ライブラリとして実装されていること，(2)C ライブラリによる大域アドレス空間を扱う上での生産性を改善し，性能を向上させることを目的とした `localize` と呼ぶ大域アドレス空間操作を導入していること にある．この `localize` 操作を用いることによって，(a) プログラマによる明示的な通信の集約を簡単化でき，また (b) 大域アドレス空間におけるキャッシュ機能を提供しており，プロセス内で多数のスレッドが動作する軽量マルチスレッド処理系において，通信したデータをスレッド間で共有することができる．

本稿の構成は次の通りである．2 章では，提案する GAS ライブラリのプログラミングモデルについて述べ，3 章では，その実装手法について説明する．4 章では，提案する GAS ライブラリと関連する研究を比較し，5 章で，本稿をまとめる．

2. プログラミングモデル

2.1 設計指針

提案する GAS ライブラリは，(1)Lazy Task Creation [9] と同様のスケジューリング戦略および (2) プロセスをまたがった大域的なワークスティーリングが可能な軽量マルチスレッド処理系から利用することを想定して設計する．このような軽量マルチスレッド処理系によって記述された並列プログラムは，次の性質をもつ．

(1) ワークスティーリングが発生しない限り，スレッド生成を単なる関数呼び出しとするような実行順序でスレッドが実行される．
(2) 生成されるスレッドの数が多く，各スレッドが行う計算の粒度が小さい．
(3) 各スレッドは，プロセスをまたがって移動しうる．

これらの性質を考慮すると，軽量マルチスレッドから利用する場合において，GAS ライブラリは，性能の観点から次の要件を満たすことが望ましい．

- グローバルメモリアクセス時のオーバーヘッドが十分小さいこと
- 複数のスレッドが同じグローバルメモリ領域のデータを読み込む場合に，複数の読み込み時の通信を一つにまとめることができること
- あるプロセス上に存在するグローバルメモリ領域を別のプロセスに移動できること

また，提案するライブラリでは，プログラマからの利用しやすさを考慮して，次の要件を設定した．

- グローバルメモリを指すポインタについて，C と同様のポインタ演算が可能であること

2.2 実行モデル

提案する GAS ライブラリは，大域的なワークスティーリングをサポートする細粒度マルチスレッド処理系から利用することを想定しているが，GAS ライブラリ単体で利用する際には，実行モデルとして SPMD モデルを採用する．並列実行の単位はプロセスと呼び，各プロセスの仮想アドレス空間 (本研究の提供する大域アドレス空間ではなく，計算ノードのオペレーティングシステムの提供するもの) は共有されない．このため，プロセス間の通信は，ライブラリの提供する大域アドレス空間を介して明示的に行う必要がある．

2.3 メモリモデル

提案するプログラミングモデルにおいては，プログラマが扱うアドレス空間として，2 種類のアドレス空間を提供する．一つは，オペレーティングシステムが提供しており，各プロセスごとに存在する仮想アドレス空間である (このアドレス空間上に存在するメモリ領域をローカルメモリと呼ぶ)．もう一つは，本ライブラリが提供する，プロセス間でアドレス空間を共有することのできる大域アドレス空間である (このアドレス空間上に存在するメモリ領域をグローバルメモリと呼ぶ)．

グローバルメモリは，PGAS モデルを採用している多くのプログラミング言語やライブラリと同様に，その実体となるメモリ領域を各プロセスにわたって分散させることができる．分散の形としては，ユーザが指定したサイズのメモリ領域を単位としてブロックサイクリックに各プロセスに分散させるものとする (この分散の単位をページと呼ぶ)．

大域アドレス空間に対する操作としては，グローバルメモリの割り当て・解放 (alloc/free)，グローバルメモリのローカルメモリへのマップ・アンマップ (localize/unlocalize)，ローカルメモリに対する変更のグローバルメモリへの反映 (commit) を提供する (**図 1**)．

- **alloc**: `size` で指定されたサイズのグローバルメモリ

```
typedef uint64_t globalptr_t;
typedef struct {
  size_t index;
  size_t size;
} vector_t;

globalptr_t alloc(size_t size);
void free(globalptr_t p);
void *localize(globalptr_t p, size_t size,
               vector_t vectors[], size_t n_vectors,
               bool own);
void unlocalize(globalptr_t p, void *local_p);
void commit(globalptr_t p, size_t size,
            vector_t vectors[], size_t n_vectors,
            bool own);
```

図 1 Global Address Space API

領域を割り当て，そのグローバルメモリ領域へのポインタを返す．

- **free**: p で指定されたグローバルメモリ領域を解放する．
- **localize**: p から始まり，size で指定されたサイズを持つグローバルメモリ領域に対応するローカルメモリ領域を割り当て，vectors で指定される範囲のグローバルメモリ領域のデータをローカルメモリ上の同じ位置に読み込む．vectors で指定されなかった範囲のローカルメモリ領域の値は未定義である．同じプロセス上で，同じグローバルメモリ領域に対して localize が実行された場合には，先行する localize で指定したグローバルメモリ領域が，後続の localize で指定したグローバルメモリ領域を包含する場合に限り，対応するローカルメモリ領域が共有される．それ以外の場合における動作は未定義である．
- **unlocalize**: localize 関数を用いて割り当てられたローカルメモリ領域を解放する．
- **commit**: p および vectors で指定されたローカルメモリ領域のデータをグローバルメモリ上に書き込む．

localize および commit それぞれについて，指定したグローバルメモリ領域を所有するプロセスを変更する (この操作をページマイグレーションと呼ぶ) ためのオプションとして，own を提供する．ページマイグレーションは，プログラムに存在するデータアクセスの局所性を活用するための操作であり，特に，あるプロセスがある領域を頻繁に読み書きする際に，そのプロセス上に，グローバルメモリのブロックを配置することを可能にし，グローバルメモリアクセス時の通信を削減することができる．

alloc 操作によって返されるグローバルポインタは，C 言語のポインタと明確に区別されており，直接デリファレンスすることはできない．グローバルポインタの指す先のグローバルメモリ領域にアクセスする際には，localize 操作を用いて，一度ローカルメモリ上にマップする必要がある．

localize のもつ重要な性質を 2 つ説明する．

- localize/unlocalize は，一度 localize によってローカルメモリ上にマップされたデータを，他の localize 呼び出しによって通信なしで読むことができるという点において，通信結果のキャッシュを行っているものと見なすことができる．このキャッシュは，システムによって一貫性を保つことはなく，プログラマが，キャッシュとして存在するデータが最新であるかどうかを常に意識しながらプログラムを記述する必要がある．
- localize を用いると，グローバルメモリ上に離散したデータをグローバルメモリにおける位置と同じローカルメモリ上の位置に読み込むことができる．これに

よって，グローバルメモリから読む際の位置指定と，ローカルメモリからデータを読む際の位置指定を統一することができ，プログラマによる配列のインデックス変換にかかる実装コストを低減することができる．

3. 設計と実装

3.1 概要

提案する GAS ライブラリにおける各関数の動作の概要は次の通りである．

- **alloc**: システム全体で一意なグローバルアドレスの範囲を割り当てる．
- **free**: 割り当てたグローバルアドレスの範囲を再利用可能な状態にし，グローバルメモリの各ページに対応するメモリ領域を解放する．
- **localize**:
  - 指定した領域に対応するローカルメモリ領域が存在しない場合: 指定されたサイズのローカルメモリ領域を割り当て，指定したグローバルメモリ領域からそのローカルメモリ領域にデータを読み込む．そして，割り当てたキャッシュメモリ領域を返す．
  - 指定した領域に対応するローカルメモリ領域が存在する場合: 指定したグローバルメモリ領域と対応するローカルメモリ領域を返す．
- **unlocalize**: 指定したグローバルメモリ領域に対応するローカルメモリ領域を解放する．
- **commit**: 指定した位置のデータをローカルメモリ領域からグローバルメモリ領域に書き出す．

本実装では，グローバルメモリを管理するために，(1) グローバルアドレスアロケータ，および (2) グローバルメモリテーブルの二種類のデータ構造を使用する．

また，提案する GAS ライブラリでは，ページマイグレーションを提供している．ページマイグレーションを実現するためには，あるページがどのプロセスによって所有されているかを特定する必要がある．この点について，本実装では，各ページについて，そのページを所有しているプロセスを常に管理する「ホームプロセス」を用いて実現する．

3.2 グローバルアドレスの割り当て

グローバルメモリを割り当てる際には，グローバルアドレス空間における未使用領域を検索する必要がある．本実装では，この未使用領域を管理するために，グローバルアドレスアロケータと呼ぶデータ構造を採用している．

グローバルアドレスアロケータにおけるアドレス割り当ては，Linux などのオペレーティングシステムにおける物理メモリ管理に用いられる buddy system を用いて実装している．

グローバルアドレスアロケータは，基本的には，プロセス 0 が管理しており，アドレス割り当てを行う際には，プロセス 0 に割り当てを依頼する必要がある．しかし，アドレス割り当てを行うたび (すなわち，グローバルメモリ割り当てを行うたび) に，プロセス 0 との通信が発生するのは，細粒度のメモリ割り当てを行うプログラムにおいて性能低下の要因になりうる．

この問題に対して，本実装では，各プロセスについて一定のアドレス範囲をそのプロセス専用のアドレス範囲とし，一定サイズ以下のグローバルメモリについては，そのアドレス範囲からアドレスを割り当てる，という方法を取っている．この方法によって，一定サイズ以下のグローバルメモリ割り当てについては，プロセス 0 との通信なしで行うことができるようになる．

3.3 ホームプロセスの割り当て

3.1 節で述べたように，グローバルメモリにアクセスする際には，アクセスするページに対応するホームプロセスを特定する必要がある．これについて，本実装では，グローバルアドレスからホームプロセスを特定する方法を採用している．本実装において，ホームプロセスは，グローバルメモリの各ページについてプロセス 0 からラウンドロビンの順序で割り当てる．このようにホームプロセスを割り当てることによって，グローバルメモリアクセス時に与えられるグローバルアドレスに対し，いくつかの算術演算を適用するだけでホームプロセスを特定することができる．

3.4 グローバルアドレス変換

グローバルメモリにアクセスする際には，グローバルアドレスから，(1) アクセスするブロックを所有するプロセス，(2) ブロックに対応するメモリ領域のアドレス を得る必要がある．これらの情報を管理するため，本実装では，グローバルメモリテーブルと呼ぶデータ構造を使用する．

グローバルメモリテーブルは，各プロセスがそれぞれ独立して管理するデータ構造であり，グローバルアドレスからそのアドレスの示すページに関する情報を検索するためのテーブルである．グローバルメモリテーブルのエントリは次の情報を保持している．

- ブロックに対応するメモリ領域のアドレス (ブロックを所有しているプロセスのみが管理するデータ)
- ブロックを所有しているプロセスの ID (ホームプロセスのみのみが管理するデータ)
- キャッシュメモリ領域のアドレス
- キャッシュメモリ領域のサイズ

グローバルメモリテーブルは，Linux などのオペレーティングシステムで採用されているものと同様の，多段ページテーブルを用いて実装されている．

3.5 通信プロトコル

実装の基盤となる通信ライブラリとしては， GASNet

は，多くの分散メモリ型並列計算機において，高性能な Active Messages [10] と遠隔メモリアクセスを提供している GASNet [11] を採用している．

グローバルメモリアクセスを実装する上での注意点として，グローバルメモリアクセス中にページマイグレーションが発生しうるということがある．この問題に対処するため，提案するライブラリを実装するにあたって，次の二点を満たすように実装した．

- グローバルメモリへのアクセスを要求するプロセス (イニシエータ) から直接遠隔メモリアクセスを実行するのではなく，一度グローバルメモリアクセス要求をターゲットに送信し，そのページを所有するプロセス (ターゲット) が遠隔メモリアクセスを実行する．
- もう一つは，ターゲットにおいて，ホームプロセスからのグローバルメモリアクセス要求を (到着順ではなく) 送信順に基づいて処理する．

この二つの条件を満たすことによって，グローバルメモリアクセスを実行している最中に対象となるページのマイグレーションが発生しないようにすることができる．

まとめると，グローバルメモリへのアクセスは，次の手順で行われる．

(1) イニシエータは，アクセスする対象となるページのホームプロセスにデータアクセス要求を送信する．
(2) ホームプロセスは，データアクセス要求を受け取ると，ターゲットに要求を転送する．
(3) ターゲットは，イニシエータのもつキャッシュ領域とターゲットのもつページのメモリ領域の間で，メモリアクセスの種類に応じた遠隔メモリアクセスを実行する．
(4) ターゲットは，遠隔メモリアクセスを終えた後，イニシエータに対して完了を通知する．

3.6 現在の実装における制約

現在の実装では，グローバルメモリにおけるブロックのサイズはコンパイル時定数としているが，将来的には，各グローバルメモリについて任意のブロックサイズを選択できるようにする予定である．

4. 関連研究

分散メモリ型並列計算機において，仮想的な共有メモリを提供するプログラミング言語やライブラリは，数多く提案されてきている．特に，近年は，PGAS モデルに基づくプログラミング言語やライブラリとして，UPC [12] , XcalableMP [13] , Chapel [14] , X10 [15] , Global Arrays [16] などが提案されている．

UPC，XcalableMP，Chapel，X10 は，提案する GAS ライブラリの特徴である (1)C ライブラリとして実装されていること，(2) ページマイグレーションのサポート，(3) キャッシュのサポート を提供していない．また， Global Arrays については，(1) を満たすものの，(2)，(3) といった，軽量マルチスレッド処理系から利用する際に必要となる機能を提供していない．

その他に，仮想的な共有メモリを提供するプログラミング環境として，ソフトウェア分散共有メモリ (SDSM) システムが存在し，IVY [17] , Munin [18]，Treadmarks [19] など多数の処理系が存在する．このような SDSM システムにおいて，多くのシステムは，上記 (1)，(2)，(3) を提供しているものの，システムがキャッシュの一貫性を保つようなキャッシュシステムとなっており，ユーザが自由に (2)，(3) といった機能を制御できるようにはなっていない．

5. 現状と今後の課題

本稿では，分散メモリ型並列計算機において，仮想的な共有メモリを提供するライブラリを提案した．このライブラリは，分散メモリ型並列計算機において，大域的な動的負荷分散を行うような軽量マルチスレッド処理系からの利用を想定して設計されたものであり，`localize` と呼ぶ，通信を集約する機能，共有メモリのデータをプロセス上のメモリにコピーする際の，共有メモリにおけるインデックスとプロセス上のメモリにおけるインデックスを揃える機能，共有メモリに対するキャッシュ機能を同時に合わせもつ共有メモリアクセスインターフェースを採用している．

現在は，プログラミングモデルの細部は検討段階で，いくつかのプログラムにおいて実際にプログラムを記述し，十分に使用に耐えうるものであるか検証している段階である．また，実装については，本稿で述べた実装手法を基にプロトタイプ実装を進めている段階である．

今後の課題は，プロトタイプ実装を進め，ライブラリの性能評価を行うことや，プロセス内でマルチスレッド (特に軽量マルチスレッド) を利用する場合において，性能の良い実装を検討することである．また，軽量マルチスレッド処理系と組み合わせた場合の性能評価も行う必要がある．

謝辞 本研究は，JST CREST「高性能・高生産性アプリケーションフレームワークによるポストペタスケール高性能計算の実現」の助成を得て行われた．

参考文献

[1] Forum, M. P.: MPI: A Message-Passing Interface Standard, Technical report, Knoxville, TN, USA (1994).
[2] Frigo, M., Leiserson, C. E., Prokop, H. and Ramachandran, S.: Cache-Oblivious Algorithms, *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, Washington, DC, USA, IEEE Computer Society, pp. 285– (1999).
[3] Frigo, M., Leiserson, C. E. and Randall, K. H.: The implementation of the Cilk-5 multithreaded language, *Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation*,

PLDI '98, New York, NY, USA, ACM, pp. 212–223 (1998).
[4] Lea, D.: A Java fork/join framework, *Proceedings of the ACM 2000 conference on Java Grande*, JAVA '00, New York, NY, USA, ACM, pp. 36–43 (2000).
[5] Intel: *Intel® threading building blocks reference manual* (2009).
[6] Frigo, M.: Portable High-Performance Programs, Technical report, Cambridge, MA, USA (1999).
[7] 中島 潤，田浦健次朗：高効率な I/O と軽量性を両立させるマルチスレッド処理系，情報処理学会論文誌プログラミング（PRO），Vol. 4, No. 1, pp. 13–26 (2011).
[8] Bader, M.: Exploiting the Locality Properties of Peano Curves for Parallel Matrix Multiplication, *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Euro-Par '08, Berlin, Heidelberg, Springer-Verlag, pp. 801–810 (2008).
[9] Mohr, E., Kranz, D. A. and Halstead, Jr., R. H.: Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 2, pp. 264–280 (1991).
[10] 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, New York, NY, USA, ACM, pp. 256–266 (online), DOI: 10.1145/139669.140382 (1992).
[11] Bonachea, D.: GASNet Specification, v1.1, Technical report, Berkeley, CA, USA (2002).
[12] UPC Consortium: UPC Language Specifications, v1.2, Tech Report LBNL-59208, Lawrence Berkeley National Lab (2005).
[13] Lee, J. and Sato, M.: Implementation and Performance Evaluation of XcalableMP: A Parallel Programming Language for Distributed Memory Systems, *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pp. 413 –420 (2010).
[14] David Callahan, Bradford L. Chamberlain, H. P. Z.: The Cascade High Productivity Language, *High-Level Programming Models and Supportive Environments, International Workshop on*, Vol. 0, pp. 52–60 (2004).
[15] Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C. and Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing, *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, New York, NY, USA, ACM, pp. 519–538 (2005).
[16] Nieplocha, J., Harrison, R. J. and Littlefield, R. J.: Global arrays: a portable "shared-memory" programming model for distributed memory computers, *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, Supercomputing '94, New York, NY, USA, ACM, pp. 340–349 (1994).
[17] Li, K.: Shared virtual memory on loosely coupled multiprocessors, PhD Thesis, New Haven, CT, USA (1986).
[18] Carter, J. B.: Design of the Munin distributed shared memory system, *J. Parallel Distrib. Comput.*, Vol. 29, pp. 219–227 (1995).
[19] Amza, C., Cox, A., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W. and Zwaenepoel, W.: TreadMarks: shared memory computing on networks of workstations, *Computer*, Vol. 29, No. 2, pp. 18 –28 (online), DOI: 10.1109/2.485843 (1996).