Vol.2013-HPC-141 No.15
2013/10/1

通信ライブラリの自動チューニングを支援する Hint API の提案

南里 豪志[1,2,a)]

概要：計算機の大規模化に伴い，通信ライブラリの性能に対する要求が高まっている．通常，通信ライブラリは，システムの基本性能や各通信命令に渡された引数といった，限られた情報しか参照できないため，ライブラリ内で十分なチューニングが行えない．そこで本研究では，プログラマが，プログラム中での通信ライブラリの使い方に関する情報を記述するための Hint API を提案する．プログラマは，この API を用いて，例えばプログラムの性能に大きく影響するカーネル領域の範囲についての情報を指定することが出来る．この情報を元に通信ライブラリでは，性能的に重要な通信に優先的に通信バッファ等の資源を割り当てる等の高度なチューニングが可能となる．本発表では，この API を MPI ライブラリに実装し，効果を検証する．

Proposal of Hint API to Help Auto-Tuning in Communication Libraries

Takeshi Nanri[1,2,a)]

Abstract: As the computer systems become larger and complicated, demands for the efficiency of the communication libraries are increasing. In the existing libraries, choices of their behavior are tuned according to the limited information available to them, such as the basic information about the system or the parameters specified as arguments for communication operations. Therefore, to provide information about how the program invoke these operations, an interface, Hint, is proposed. For example, this interface can be used to specify which invocation of communication operation is important to the performance of the program. With this information, communication libraries can apply aggressive auto-tuning techniques, such as, giving priority for using memories and network facilities on that operation to achieve high efficiency. A sample of this interface is implemented on an MPI library, and its effect is examined.

1. はじめに

複数の計算ノードをネットワークで接続した分散メモリ型の並列計算機は，高性能計算環境として広く用いられている．このアーキテクチャでは，個々の計算ノードの演算性能向上と，計算ノード数の増加によって計算速度の向上が図られてきた．一方，計算ノード数増加に伴う計算機全体の導入費用の増加を避けるため，メモリや通信機器等のハードウェア資源の量は，計算機の規模の増加に対してほとんど増加していないか，むしろ減少しており，この傾向は今後も続くと予想されている．これらのハードウェア資源は，理論的な計算能力には寄与しないものの，実際のアプリケーション性能には大きな影響を与える．また，計算に利用することの出来る電力に対しても，今後さらに厳しい制約が課せられることとなる．そのため，ハードウェア資源や電力を有効に用いた計算手法の開発が求められている [1]．

分散メモリ型並列計算機における並列計算では，プロセス間のデータ交換が必要となるため，通信ライブラリが重要な役割を果たしている．従来，通信ライブラリに対する主な要求としては，提供する通信機能の高度化と，各通信の遅延削減および帯域幅向上が求められてきた．しかし今後は，前述した背景により，通信ライブラリにおいてもハードウェア資源や電力を有効に利用する技術が重要とな

[1] 九州大学情報基盤研究開発センター
[2] JST CREST
[a)] nanri@cc.kyushu-u.ac.jp

る．資源に制約がある中で性能向上を追求するためには，資源の適切な配分が必須である．特に通信ライブラリにおいては，プログラム中で発行される各通信に対して，メモリや通信路をどのように配分するかによって性能が大きく変動する．そのため，プログラムの性能に大きな影響を与える通信に対して優先的に資源を割り当てるよう，通信ライブラリの挙動を調整することにより使用資源量に対する性能の向上を図ることが出来る．

一方，従来の通信ライブラリでは，通信への資源の割り当て等の調整は，システムの基本性能，通信関数に渡された引数，実行開始時に指定された環境変数等，限定的な情報に基づいて行われてきた．例えば一対一の送受信を行う際のプロトコルとして，多くの通信ライブラリでは，受信バッファを介した低遅延通信を行う Eager プロトコルと，送受信プロセス間で同期を取ることで受信バッファを用いない省メモリ通信を可能とする Rendezvous プロトコルを用意しており，通信のメッセージサイズに対する閾値によって使用するプロトコルを選択する．この閾値の初期値はシステム導入時にシステム性能や搭載メモリ量に応じて設定される．また，利用者は，環境変数や実行時オプションにより，実行毎にこの閾値を変更することが出来る．しかし，適切な閾値の設定にはプログラムの詳細な性能解析が必要となる．さらに，この手法では，プログラム中で特に重要な通信に対して優先的に通信資源を割り当てることは困難である．

そこで本報告では，通信ライブラリが自動的に適切な挙動を選択する自動チューニングを支援を目的として，プログラムに関する情報をプログラムからライブラリに提供するためのインタフェースを提案する．このインタフェースは，通信ライブラリの関数 `HINT` として定義する．これにより，プログラムの実行中の状況や，今後どのように通信ライブラリを用いるか，といった，環境変数や個々の通信関数の引数だけでは提供が困難だった情報をライブラリに渡すことを可能にする．例えば，プログラムの実行時間の大半を占めるカーネル領域について，この `HINT` 関数を用いて開始位置と終了位置を通信ライブラリに知らせることが出来る．この情報に基づいて通信ライブラリは，個々の通信命令呼び出しにおいて，それがカーネル領域内か否かで通信プロトコル選択の閾値を変えることが出来る．また，プログラマは具体的な閾値を指定するのでは無く，閾値の自動チューニングを支援するための情報を与えるだけで良いので，最適化のための性能解析等が不要で有る．

さらに本報告では，提案するインタフェースの有用性を評価するため，このカーネル領域の開始位置と終了位置をプログラム中で指定するための `HINT` 関数を MPI ライブラリ MVAPICH [2] 内に実装する．MVAPICH は，PC クラスタで広く用いられている高速インターコネクト InfiniBand [3] を主なターゲットとした MPI ライブラリである．MVAPICH には，InfiniBand の二つの転送モード Unreliable Datagram (UD) と Reliable Connection (RC) を選択的に利用可能なハイブリッドバージョンが提供されている．これらの転送モードにも通信プロトコルと同様に，使用メモリ量と通信速度のトレードオフが存在するため，通信の重要度に応じた適切な選択が必要である．そこで本報告における実装では，高速な RC 通信モードへの移行をカーネル領域中の通信に限定する．これにより，通信の重要度に応じた適切な資源配分の実現を図る．

本報告の構成は以下の通りである．まず 2 章で，本提案に関する技術的な背景として，一般的な通信ライブラリの実装技術を説明する．次に 3 章で，提案するインタフェースに対する要件と，それに基づいたインタフェースの定義を記述する．4 章では，このインタフェースの例として，カーネル領域指定のための `HINT` 関数を MVAPICH 内に実装し，実験により効果を確認する．5 章では，提案インタフェースに関連する研究を紹介する．

2. 通信ライブラリの実装技術

通信ライブラリは，プロセス間通信のための関数群，及びそれらを補助する関数群で構成される．現在利用可能な通信ライブラリの中で最も広く用いられているのは，MPICH [4]，Open MPI [5]，MVAPICH [2] のような，MPI 規格 [6] に準拠したライブラリである．これら以外にも，ARMCI [7]，GASNet [8] 等，様々な通信ライブラリが開発されている．

通信ライブラリの各関数の実装では，一対一通信や集団通信等を実現するためのアルゴリズムと，通信バッファやプロセス，デバイス等を管理するためのデータ構造について，様々な選択肢がある．例えば，送信関数と受信関数による一対一通信では，Eager プロトコルに基づいたアルゴリズム，もしくは Rendezvous プロトコルに基づいたアルゴリズムのどちらかが用いられることが多い．このうち Eager プロトコルでは，送信側が相手の受信関数発行を待たずに，相手の受信バッファに対してメッセージを送信する．受信側は，目的のメッセージの到着を待って，それを受信関数で指定されたメモリ領域にコピーする．このプロトコルは送受信間で同期を取るための通信が不要であるため，低遅延での実装が可能である．しかし，各プロセスが予め十分な大きさの受信バッファを用意する必要があるため，大きなメッセージの通信には適していない．一方 Rendezvous プロトコルでは，送信側は受信側での受信関数発行を待ってメッセージを送信するため，送受信プロセス間の同期のための遅延が必要となる．しかし，受信関数に指定されたメモリ領域に対して直接送信することが可能なので，受信側に別途受信バッファを用意する必要が無い．

このように，これらの二つのプロトコルは，遅延時間と使用メモリ量についてトレードオフの関係に有り，相互の

優劣は状況に依存する．そのため多くの通信ライブラリでは，それぞれのプロトコルに基づいたアルゴリズムを用意し，状況に応じて選択して使用する．同様の，アルゴリズムやデータ構造の選択が，通信ライブラリ内部の様々な場所で行われている．

アルゴリズム等の選択では，予め設定された閾値が用いられることが多い．例えば前述の送受信の場合，メッセージサイズの閾値が設定されており，通信対象のメッセージサイズがその閾値を下回る場合 Eager プロトコルに基づくアルゴリズムを，閾値以上の場合 Rendezvous プロトコルに基づくアルゴリズムを選択する．通常，この閾値は通信ライブラリ開発時に設定された初期値，もしくは通信ライブラリをシステムに導入する際のベンチマーク結果等に基づいて設定された値が用いられる．また，多くの通信ライブラリでは，これらの閾値について，利用者による変更を可能とするための環境変数や実行時オプションを用意しており，この場合，実行単位での調整が可能である．

3. 通信ライブラリにプログラム情報を提供するための Hint インタフェースの提案

3.1 プログラム情報による通信ライブラリの自動チューニングの可能性

2 節に記述したように，通信ライブラリの内部では様々な実装手段の選択が行われている．この選択に応じて，メモリや NIC，リンクといった資源が各通信に割り当てられ，それにともなって各通信の性能が変動する．特に今後は，各プロセスで使用可能な資源量に対する制約が厳しくなると予想されており，通信ライブラリが適切に実装手段を選択することによって資源の有効利用を図る自動チューニング技術の開発が求められている．

この自動チューニングにおいては，計算機やプログラムの状況に応じて内部を調整するため，十分な情報の収集が必要である．これに対し，既存の通信ライブラリで参照可能な情報としては，計算機の基本的な仕様と性能，プログラムに割り当てられた計算ノードの情報，プログラム実行開始時に利用者から指定された環境変数や実行時オプション，及び各通信関数に渡される通信相手やメッセージサイズ等の引数があげられる．そのため，通信ライブラリは個々の通信関数呼び出しについて，例えばプログラム中でどの程度重要なのか，といった情報を得ることが出来ないため，これらに対して均等に資源を割り当てることしか出来ない．

一方，プログラム中の通信関数利用に関する情報を通信ライブラリが参照できれば，より積極的な自動チューニングが可能となる．例えば，プログラムの性能に大きな影響を与える通信関数呼び出しを通信ライブラリが認識できる場合，通信ライブラリはその通信に対して優先的に大きな受信バッファを用意し，閾値よりも大きなメッセージに対して Eager プロトコルを使用することが出来る．その結果，限られた資源量の中で高い性能を得ることが可能となる．また，ある集団通信関数が同じ引数で十分な回数繰り返し呼び出されることを通信ライブラリが知ることが出来れば，多少の時間をかけて選択可能な実装手段を探索し，プログラムの状況に適したものを選択することが出来る．もしくは，プログラム中で動的にメモリを確保する命令が初期化部分のみに存在している場合，初期化処理の終了をライブラリに通知することにより，ライブラリはその時点での未使用メモリ領域全体を通信バッファとして使用できる．

```
int HINT(char *key, char *value);
```

図 1　HINT 関数の定義

3.2 Hint インタフェースの要件

本報告では，プログラムの情報を通信ライブラリに提供するための Hint インタフェースを提案する．このインタフェースを設計する際に設定した要件は以下の通りである．

広い適応性　本インタフェースは，通信ライブラリの自動チューニングを支援する多様なプログラム情報を扱える必要がある．

高い可搬性　プログラマの労力を最小限にするため，同じ情報が複数の通信ライブラリで共通に受理できるようなインタフェースとするべきである．

低い参照コスト　提供された情報は，ライブラリ中で頻繁に参照される可能性がある．そのため，参照コストの低い実装が可能となるようなインタフェースであるべきである．

修正の少なさ　情報提供のためのプログラムへの修正は可能な限り少なくなるようにすべきである．さらに，本インタフェースを提供するための通信ライブラリへの修正を少なくし，既存のライブラリへの導入を容易にすべきである．

3.3 Hint インタフェースの定義

前節で挙げた適応性や可搬性，修正の少なさ等の要件を満たすため，提案するインタフェースは関数 HINT として定義する．図 1 に HINT 関数の定義を示す．

この関数は，key と value の対を引数とする．指定された key に通信ライブラリが対応している場合，通信ライブラリは value を自動チューニングに利用する．もし指定された key に通信ライブラリが対応していない場合，その key は無視される．この関数返り値としては，key と value が正常に受理された場合 0 を，それ以外の場合 -1 を返す．

関数の実装は，通信ライブラリ，および対応するプログラム情報に依存する．key と value を通信ライブラリの挙動に反映させる方法として，例えば HINT 関数で指定された key と value の対を全てリスト構造やテーブルに格納し，必要に応じてその中から探索して参照する方法が考えられる．しかしこの方法の場合，参照の度に key を探索する必要があるため，参照コストが高くなり，頻繁に参照することが出来ない．一方，HINT 関数は引数として key と value という二つの文字列しか持たないので，各 key に対応したパラメータ変数を通信ライブラリ内部に用意し，HINT 関数では key に対応するパラメータ変数を value に応じて変更する，という方法も可能である．この場合，key の探索は HINT 関数呼び出し時の一回となり，渡されたプログラム情報の参照は，対応するパラメータ変数への参照に置き換えられるため，低コストでの参照が可能となり，頻繁にプログラム情報を参照することが出来る．

この関数を挿入する作業は，アプリケーションプログラマや数値計算ライブラリのプログラマの他に，Co-Array Fortran や Unified Parallel C のような高水準並列言語のコンパイラによる自動挿入も想定できる．また，MPI プログラムを解析し，HINT 関数を自動挿入する source to source のコンパイラを開発することも，技術的に十分可能であると考えられる．

4. Hint インタフェースの実用性の検証

4.1 検証の概要

本節では，提案した Hint インタフェースについて実用性を検証するため，通信ライブラリ上に実際にこのインタフェースを実装し，実装の容易さを確認する．この実装には，HINT 関数と，それによって得られるプログラム情報に基づいた自動チューニング機構が含まれる．また，実機での実験により，この自動チューニング機構の効果を調査する．

4.2 プログラムのカーネル領域情報取得のための HINT 関数

HINT 関数の実装として，今回の実験では，プログラムのカーネル領域の開始位置と終了位置を取得するものを実装した．ここでカーネル領域とは，プログラムの実行時間の大半を消費するプログラム領域とする．これにより通信ライブラリは，カーネル領域に含まれる通信に対して優先的に資源を割り当てて高速化することが可能となる．

カーネル領域に関する情報を提供する際の key としては “IN_KERNEL” を用いる．一方 value としては，カーネル領域開始を指示する場合 “1” を，カーネル領域終了を指示する場合 “0” を，それぞれ用いる．プログラム中に複数のカーネル領域が存在しても構わない．今回の実装では，通信ライブラリはカーネル領域の情報に関する整合性を確認しない．すなわち，全プロセスが同じ領域をカーネル領域として指定していることや，カーネル領域の開始と終了は一対一で対応しており，入れ子になっていないこと，等についてはプログラマの責任とする．

今回の実験で用いた実装を図 2 に示す．この実装では，通信ライブラリ内に新しいパラメータ変数 HINT_in_kernel を追加し，HINT 関数で値を変更している．具体的には，カーネル領域開始時にこの変数には 1 が格納され，終了時に 0 に戻る．このパラメータ変数は大域変数であるため，通信ライブラリの任意の箇所で，この変数の値によって現在カーネル領域内で実行されているか否かを確認できる．

```
int HINT_in_kernel = 0;
typedef struct HINT_item{
  char *key;
  int *v;
} HINT_item_t;
HINT_item_t Hint_table[]={
{"IN_KERNEL", &HINT_in_kernel},
{NULL,NULL}
}
int HINT(char *key, char *val)
{
  int i;
  i = 0;
  while (HINT_table[i].key){
    if (strcmp(key, HINT_table[i].key)){
      *(HINT_table[i].val) = atoi(val);
      return 0;
    }
    i++;
  }
  return -1;
}
```

図 2 カーネル領域情報を処理する HINT 関数の実装

4.3 InfiniBand の通信モード自動選択機構

前節で紹介したカーネル領域の情報を用いた通信ライブラリ自動チューニング機構の例として，本実験では，InfiniBand の通信モード自動選択機構を構築する．この機構は，カーネル領域内で通信に対してのみ，メモリを多く使用する高速通信モードの選択を許可することで，メモリの利用効率向上を図るものである．

InfiniBand の通信モードとは，InfiniBand によるインターコネクト上でデータを転送する際のプロトコルである．1 節で紹介したとおり，ほとんどの通信ライブラリでは，RC と UD のいずれか，もしくは両方を用いている．

RC は，通信するプロセス間に専用のコネクションを用意し，データを転送するモードである．このモードでは，データの到達性や到着順序が保証されていて使いやすい上，RDMA による高帯域幅の通信が可能であるため，多くの通信ライブラリで用いられている．しかし，一つのコネクション毎に送信キューや受信キュー等の確保が必要となるため，プロセス数が数十万となった場合，メモリが不足する可能性が高い．一方 UD を用いる場合，各プロセスは一対の送信キューと受信キューで多数のプロセスと通信することが可能である．そのため，通信ライブラリが使用するメモリ領域がプロセス数によらず一定となるため，大規模な並列計算環境への対応が容易である．しかし，UD ではデータの到達性や到着順序を保証するための処理も通信ライブラリが行う必要がある上，メッセージの転送には帯域幅が比較的低い send/receive による通信しか用いることが出来ない．

MPI ライブラリの一つである MVAPICH には，RC と UD を選択的に利用するハイブリッド機能が用意されている．この機能を用いる場合，プログラムの実行開始時には全ての通信が UD を用いる．その後，プログラム実行中に，あるプロセスから特定のプロセスに対する送信の回数が，予め設定されていた閾値を超えた場合，その送信側のプロセスは受信側のプロセスに対して RC 用のコネクション開設を要求する．コネクションの数が上限に達していなければ，この要求は受理され，コネクションが開設される．それ以降は，これらのプロセス間は RC を用いた通信が行われる．

一方，本実験で構築した自動チューニング機構では，RC コネクションの開設要求を，カーネル領域内で発生する送信に限定することにより，メモリの利用効率向上を図る．具体的には，送信処理の内部で `HINT_in_kernel` 変数の値でカーネル領域内か否かを判定し，カーネル領域で無ければコネクションの開設要求を発行しないよう，MVAPICH のソースコードを改変した．

4.4 実験概要

4.4.1 実験環境

構築した自動チューニング機構の効果を，九州大学情報基盤研究開発センターの高性能演算サーバ Fujitsu CX400 を用いた実験により検証した．計算機の諸元を表 1 に示す．今回の実験では，ノード数の増加によるメモリ消費量の変動を調べるため，1 ノード当たり 1 プロセスを割り当てて実行した．

この実験では，表 2 に示す 4 つのケースについて，それぞれプログラムを実行し，メモリ消費量と性能を比較した．RC-only は全通信で RC を使用するのに対し，UD-only は全通信で UD を使用する．HYBRID は，MVAPICH におけるハイブリッド機能を有効にした場合の標準の状態である．HINT は，4.2 で説明した `HINT` 関数と，4.3 節で説明した自動チューニング機構を実装した MVAPICH である．

表 1　実験環境

計算機	Fujitsu PRIMERGY CX400
CPU	Intel Xeon E5-2680 2.7GHz (2sockets x 8cores)
メモリ	128GB/node
インターコネクト	InfiniBand FDR
HCA	Mellanox MT4099
ノード数	1476 ノード中の 128 ノード
OS	RedHat 6.1 (kernel 2.6.32)
コンパイラ	gcc 4.4.6
MPI	MVAPICH2-1.9

表 2　実験に用いたケース

ケース	ライブラリ構築方法
RC-only	MVAPICH の標準の構築
UD-only	MVAPICH 構築時に `--enable-hybrid` を指定し，プログラム実行時に環境変数 `MV2_USE_ONLY_UD` を 1 に設定する．
HYBRID	MVAPICH 構築時に `--enable-hybrid` を指定し，実行時に環境変数 `MV2_USE_UD_HYBRID` と `MV2_HYBRID_ENABLE_THRESHOLD` をそれぞれ 1 に設定する．
HINT	カーネル領域を処理する `HINT` 関数とそれを用いた通信モード自動選択機構を実装した MVAPICH を使用し，構築時に `--enable-hybrid` を指定する．実行時は，環境変数 `MV2_USE_UD_HYBRID`, `MV2_HYBRID_ENABLE_THRESHOLD`, `MV2_UD_NUM_MSG_LIMIT` に，それぞれ 1 を設定する．ベンチマークプログラムでは，カーネル領域の開始位置と終了位置をそれぞれ `HINT` 関数で指定する．

4.4.2 実験に用いたプログラム

今回の実験では，科学技術計算の多くのプログラムにおける典型的な通信パターンを模擬した，図 3 に示すプログラムを用いる．これは，カーネル領域の処理と，その準備処理を繰り返すものである．通常，カーネル領域では，隣接通信等，限定的な相手との通信が行われることが多い．一方，カーネル領域の準備処理では，次のカーネル領域での計算に用いるデータを用意するため，全プロセスでの集団通信が行われることが多い．そこで，図 3 における `communication1` としては `MPI_Allreduce`，`MPI_Allgather`，`MPI_Alltoall` のいずれかの集団通信を 10 回繰り返す．集団通信における各プロセスの送信メッセージサイズは 64 バイトで固定する．一方 `communication2` としては，一つ前のランクからの受信と一つ後のランクへの送信によるシフト通信を行

う．各シフト通信でのメッセージサイズは 10KB とする．また，カーネル領域の繰り返し数 N は 10,000 とする．

```
1. main()
2. {
3.    ...
4.    for (r = 0; r < REPEAT; r++){
5.      communication1();
6.      for (i = 0; i < N; i++){
7.          communication2();
8.     }
9.    }
10.   ...
11. }
```

図 3　実験に用いたプログラムの構成

このように，カーネル領域での通信相手が少数のプロセスに限定され，かつ，カーネル領域以外の通信がプログラムの実行時間にほとんど影響しない場合，カーネル領域内の通信に対して優先的にメモリを割り当てて高速化することにより，使用メモリ量を抑えつつ通信速度の向上を図ることが出来る．そのため，図 3 のプログラムの場合，カーネル領域には 7 行目の communication2 のみが含まれるようにすべきである．しかし 7 行目の前後に HINT 関数を挿入すると，この関数の処理時間が，1 回の反復毎に 2N 回分必要となる．そこで，カーネル領域開始を指示する HINT 関数を 5 行目と 6 行目の間，終了を指示する HINT 関数を 8 行目と 9 行目の間にそれぞれ挿入することで，1 回の反復毎の HINT 関数の処理時間を 2 回とする．

4.5　実験結果

図 3 のプログラムにおける communication1 の集団通信として MPI_Alltoall を用いた場合のメモリ消費量を図 4 に示す．これより，UD-only，HYBRID および HINT におけるメモリ消費量が RC-only の場合よりも大きいことが分かる．特にプロセス数が小さい場合，RC-only のメモリ消費量は非常に小さい．これは，MVAPICH では実際に通信したプロセス対に対して RC のコネクションを開設するためである．このため RC-only では，プロセス数の増加に応じてメモリ消費量が増加する．

一方，UD-only，HYBRID および HINT は，標準の通信モードとして UD を用いる．UD では，他の全プロセスから到着するメッセージを一つの QP で処理するため，プロセス数によらず，予め十分な大きさのメモリを確保している．実行中，UD-only は RC コネクションを開設しないのに対し，HYBRID は，あるプロセスから特定のプロセスへの送信回数が閾値を超えると，RC コネクションの開設を

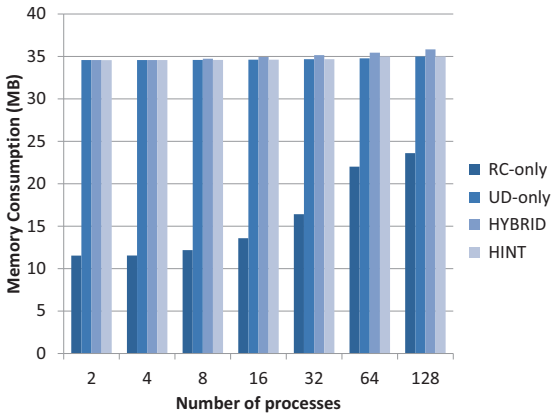

図 4　メモリ消費量 (MPI_Alltoall)

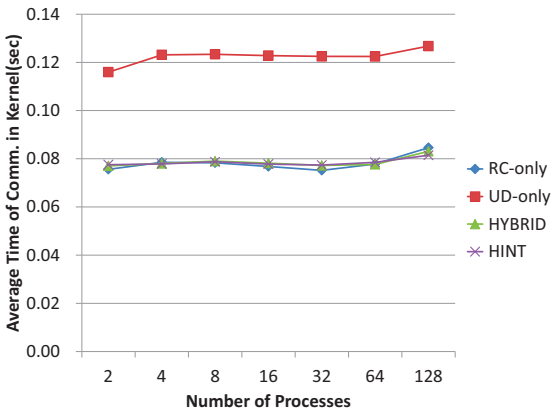

図 5　カーネル領域の平均実行時間 (MPI_Alltoall)

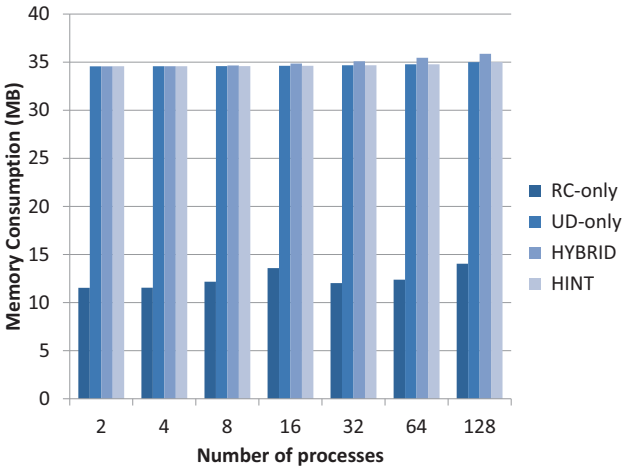

図 6　メモリ消費量 (MPI_Allreduce)

要求する．そのため，UD-only ではプロセス数によらずメモリ消費量は一定である．一方 HYBRID では，プロセス数に応じてメモリ消費量が増加する．

HINT も，実行中に RC コネクションを開設する．しかし，対象となるのはカーネル領域内の通信のみである．今回の実験で用いたプログラムでは，カーネル領域内の通信は隣接プロセスとのシフト通信のみであるため，UD-only

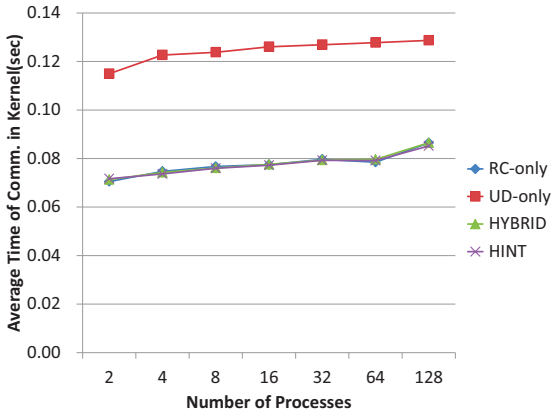

図 7　カーネル領域の平均実行時間 (MPI_Allreduce)

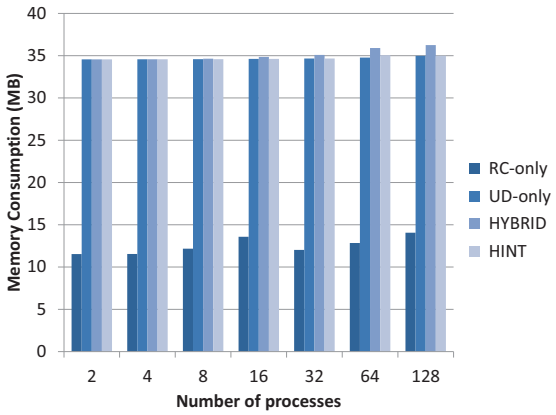

図 8　メモリ消費量 (MPI_Allgather)

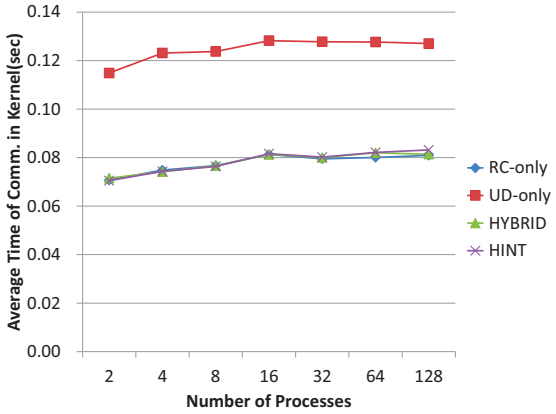

図 9　カーネル領域の平均実行時間 (MPI_Allgather)

と同様にメモリ消費量はプロセス数によらず一定である．

図 5 に，カーネル領域の一反復当たりの所要時間の平均値を示す．ほとんどの場合，UD-only は他のケースより 1.5 倍程度遅い．これは，UD では帯域幅の狭い send/receive による通信しか行えないのに対し，RC では高速な RDMA 通信が可能であるためである．

これらの結果から，HINT ではメモリ消費量を抑えながら，高い性能を維持できることが分かる．なお，今回は実験環境の制約により 128 プロセスを超えるプロセス数での実験が行えなかったため，HYBRID と HINT の相違が見られなかった．しかし，HYBRID，HINT とも，開設可能なコネクション数はメモリ容量によって制約される．プロセス数が十分大きくなると，コネクション数がこの制約に到達するため，コネクションを開設できなかったプロセス間の通信は低速となる．HYBRID では先に通信回数が閾値に到達したプロセス対からコネクションを開設するのに対し，HINT ではカーネル領域の通信に限定してコネクションを開設するため，プロセス数が大きくなった場合に通信性能に大きな差が生じると予想される．

図 6，7，8 および 9 に，図 3 のプログラムにおける communication1 の集団通信として MPI_Allreduce および MPI_Allgather を用いた場合のメモリ消費量と，カーネル領域の一反復当たりの平均所要時間を示す．カーネル領域の所要時間は，MPI_Alltoall の場合とほぼ同様である．一方メモリ消費量は，MPI_Allreduce や MPI_Allgather の場合，プロセス数の増加に対する増分が MPI_Alltoall の場合より少ない．これは，MPI_Alltoall のアルゴリズムは基本的に他の全プロセスと通信するため，RC-only の場合のコネクション数がプロセス数 P に対して $O(P)$ となるのに対し，MPI_Allreduce や MPI_Allgather のアルゴリズムは，プロセス間で仮想的な木構造を構築し，その枝に沿ってデータを転送するものであるため，RC-only の場合のコネクション数が $O(\log P)$ となるためである．その結果，プロセス数に対するメモリ消費量の増加が図 6，8 に示すとおりなだらかとなる．

4.6　考察

本報告における実験で実装した Hint インタフェースは，実装に伴う通信ライブラリへの改変が非常に小さかった．MVAPICH に追加した HINT 関数自体のソースコードは 100 行以下であり，それに伴って改変した MVAPICH のソースコードは 20 行以下であった．

このように少ない改変で実現できたのは，Hint インタフェースの定義が通信ライブラリと独立であり，しかも簡潔であったためである．例えば MPI で定義されている MPI_Info インタフェースを用いて同様の機能を実現するには，多数の MPI 関数を新たに定義するなど，大幅なソースコードの変更が必要となる．

また，Hint インタフェース利用のためのプログラムの改変は，カーネル領域を指定する Hint 関数の呼び出しの追加だけであった．一方 MPI_Info インタフェースを用いる場合，プログラマはこの情報を格納する構造体を準備し，値を設定し，MPI 関数の引数として渡す必要がある．

5. 関連研究

プログラムに関する情報を通信ライブラリに提供する手段として，MPI には MPI_Info インタフェースが定義されている [6]．これは，プログラム中で確保した構造体に，プログラムの情報を示す key と value の対を格納しておき，その構造体を関数の引数として渡すものである．そのため，前節で説明したとおり，新たにプログラムの情報を提供したい関数が出てきた場合，同様の関数で引数として MPI_Info の構造体を受理するものを別途定義する必要がある．また，プログラム情報は全て構造体の中に格納されるため，参照の度に目的の key を探索するコストが必要となる．

他にライブラリを最適化する手段としては，注釈言語とコンパイラによるものも提案されている [9]．これは，アプリケーションプログラムに応じてライブラリの実装を調整するために用いるもので，プログラマによる詳細な調整が可能である．しかし，本手法は各ライブラリについて専用の言語とコンパイラが必要である．そのため，多数のライブラリに適用するには多大な労力が必要となる．

アプリケーションプログラムに応じて通信ライブラリの挙動を調整する手法としては，プログラム中での通信関数の呼ばれ方を解析するコンパイラを用いる手法も提案されている [10]．これにより，ソースコードの再構成や追加となるコードの挿入を行い，通信の効率化を図ることが出来る．この技術は，HINT 関数の自動挿入にも応用可能である．

6. まとめと今後の課題

本報告では，通信ライブラリの自動チューニングを支援するためのプログラム情報を提供するインタフェース Hint を提案した．インタフェースの設計方針としては，簡潔で適応性が高く，実装が容易となるものを目指した．提案したインタフェースの実装例として，プログラム中のカーネル領域の情報を提供する HINT 関数と，その情報に基づいて InfiniBand の通信モードを自動選択する機構を実装し，提案インタフェースの有用性と効果を示した．

今後は，実装した通信モード自動選択機構の，より大規模な環境での効果を検証する．また，他の HINT 関数の実装を検討する．

参考文献

[1] Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Kumar, S., Lusk, E., Thakur, R. and Traff, J.L., “MPI on a Million Processors,” Recent Advances in Parallel Virtual Machine and Message Passing Interface Lecture Notes in Computer Science Volume 5759, pp 20-30, 2009.
[2] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE, http://mvapich.cse.ohio-state.edu/
[3] InfiniBand Trade Association. InfiniBand Trade Association. http://www.infinibandta.com
[4] MPICH, http://www.mpich.org/
[5] Open MPI, http://www.open-mpi.org/
[6] MPI Draft, http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
[7] ARMCI, http://www.emsl.pnl.gov/docs/parsoft/armci
[8] GASNet, http://gasnet.cs.berkeley.edu
[9] Guyer, S.Z. and Lin, C., “An annotation language for optimizing software libraries,” ,Proceedings of the 2nd conference on Domain-specific languages, pp. 39–52, 1999.
[10] Nguyen, T., Cicotti, P., Bylaska, E., Quinlan, D. and Baden, S.B., “Bamboo: translating MPI applications to a latency-tolerant, data-driven form”, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012.