2G-02

RDMA コードの置き換えによる隣接通信の高速化

野口貴希† 藤井昭宏† 田中輝雄†
工学院大学†

1 はじめに

大規模科学技術計算のプログラムでは,MPI 通信による並列化が利用され，特定の通信パターンが繰り返し現れ，通信時間の遅延が問題になっている.東京大学の FX10 は，RDMA インタフェース(Remote Direct Memory Access）を用いて通信データをリモートメモリに直接書き込むことで高速に通信可能なライブラリを提供している．RDMA は MPI とは大きく異なるインタフェースをもつため，MPI の Send，Recv による計算コード(MPI コード)から RDMA を用いたコード(RDMA コード）への変更に高い実装コストを必要とする．そこで，我々は MPI による特定の通信パターンを RDMA コードへ置き換える機構を提案する．これにより，ユーザは数行コードの追加により通信の RDMA 化が可能となる．本研究では疎行列ベクトル積(SpMV)を題材に隣接通信の高速化とコード変更量について評価した.

2 並列計算における通信

RDMA による通信では，受信メモリに直接データの読み出しや書き込みを行うことができる．ユーザは 0〜3 の NIC から使用する NIC を割り当てられる．[1] 関連研究として RDMA をもちいて通信の高速化を行うものに高機能バリア通信を利用した三次元ジョブ形状通信[2]や NIC を有効活用するため，メッセージを分割する手法がある．[3] 本研究では，通信の高速化を行うとともに，MPI の隣接通信処理を RDMA へ変換することまで対象とする．隣接通信において，送受信先のアドレスを記録することで，通信の際に，記録したアドレスを再使用することで，Send，Recv による MPI 通信に比べ高速な通信を可能としている.

Speedup of the adjacent communication by the replacement to the RDMA code
Takaki noguchi†, Akihiro Fujii† and Teruo Tanaka†
†Kogakuin University

3 関数置き換えによる RDMA 化

我々は，SpMV を行う際の通信パターンである隣接通信を題材に MPI 通信パターンに対応した RDMA 通信パターンを関数化した.

3.1 MPI の通信パターン

SpMV における隣接通信では，送信データをバッファへ代入し，受信する先頭アドレス，個数を指定する．その後 MPI_Irecv 関数を呼び出す．次に，送信する先頭アドレス,個数を指定し，MPI_Isend 関数を呼び出し通信後，MPI_Irecv の同期を行う．最後に，受信バッファから演算配列に代入し，MPI_Isend の同期を行う.

3.2 RDMA の通信パターン

上記の MPI 通信パターンに対応した RDMA 通信パターンは下のようになる． 転送用アドレス(MEMID)の登録を行う. 送信先に MEMID を送信(隣接通信の Recv 側)し，MEMID を受信(隣接通信の Send 側)後，同期をとり，MEMID のアドレスを取得する．上記を RDMA 通信の準備処理として関数化を行った.

送信データを送信バッファに代入し，送信する先頭アドレス，個数の指定し，RDMA 通信関数を呼び出し(RDMA_WRITE)を行う. その後，送信処理の終了確認と受信完了確認(末端データを参照)を行い，受信バッファから演算配列に代入を行う. 上記を RDMA 通信として関数化する.

最後に転送用アドレスとして登録した MEMID のフリー処理に関しても関数化する.

3.3 コードの置き換え

本研究の題材である SpMV において，MPI 通信部分に対して, RDMA 通信で新たに WR, WS, MEMID_array を変数とし，関数宣言により RDMA 通信を行う halo_setup，rdma_put という関数を呼び出すことで，RDMA 通信による隣接通信を行うものである.

4 数値実験

4.1 通信の高速化に対する性能評価

通信時間の高速化はついては，それぞれ以下３通りの通信方法で計測し性能評価を行った.

A)　1 対 1 通信
B)　行ブロック分割による SpMV
C)　Metis 分割による SpMV

A)では，あるプロセスは送信し，別のプロセスが受信する 1 対 1 通信における通信時間の計測．

B)C)では，x，y，z 方向にそれぞれ 100 個の立方体領域のポアソン方程式に基づく疎行列で 1 行あたり 27 前後の非ゼロ要素を持った SpMV における隣接通信時間の計測．

実験には，東京大学の FX10 スーパーコンピュータを利用し CPU は SPARC641IXFx，コンパイラは MPI，RDMA ともに mpifccpx， A)では，メッセージサイズは 100～100000 個，精度は double 型， ノード数はノード内通信（1 ノード）， ノード間通信（2 ノード）とした． B)では，問題領域を行ブロック分割した行列に，C)では Metis を用いて 3 次元方向に分割した行列とし，分割数を 6，12，48，96，192 とし，1 プロセスに対し 1 ノードを割り当てた．また 1 ノードあたり利用する NIC 数を 1NIC，4NIC の 2 通りで計測を行った．

各計測において以下の結果が得られた．

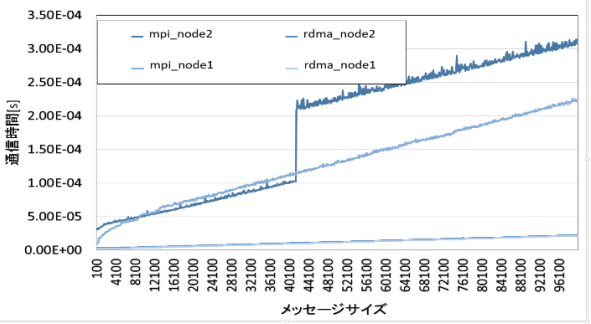

図 1　計測 A)における通信時間性能比較

表 1　B)の通信時間比較（ms）

proc	6	12	48	96	192
MPI	0.304	0.306	0.278	0.285	0.217
1NIC	0.310	0.320	0.291	0.293	0.225
4NIC	0.187	0.186	0.177	0.178	0.115

表 2　C)の通信時間比較（ms）

proc	6	12	48	96	192
MPI	0.436	0.428	0.301	0.296	0.293
1NIC	0.306	0.255	0.172	0.141	0.131
4NIC	0.275	0.217	0.122	0.095	0.086

A)において，ノード間，内どちらも MPI 通信と比較し，RDMA 通信では通信時間が削減されている．不連続な通信時間の増加は，FX10 のプロトコルによる影響であると考えられる．B)では，MPI 通信から RDMA 通信への置き換えによって 30%程度の通信時間が削減された．また，1 ノードあたりの使用 NIC を 4 本に増加することで 40%程度削減されている．C)では，置き換えによって 55%程度削減されている．B)と比べ置き換えによる通信時間は減少が見られるが，NIC の増加による通信時間の減少は見られなかった．

4.2　コード変更量の評価

MPI コードから RDMA コードへの置き換えでコード変更量について以下の結果が得られた．

```
1. for(i=0;i<loop;i++){
2.    MPI_Barrier(MPI_COMM_WORLD);
3.    time = MPI_Wtime();
4.    halo_exchange(x, neibpetot, neibpe,
            stack_import,stack_export,
            nod_import, nod_export, ws, wr);
5.    MPI_Barrier(MPI_COMM_WORLD);
6.    time = MPI_Wtime() - time;
7.     n_time[i] = time;
8.     spmv(val,col_ind, row_ptr, rowsize, x, y);
9. }
```

```
1.  FJMPI_Rdma_init();
2.  uint64_t *laddr, *raddr;
3.  int MEMID_array;
4.  raddr =
      uint64_t*)malloc(sizeof(uint64_t)*neibpetot);
5.  laddr2=
    (uint64_t*)malloc(sizeof(uint64_t)*neibpetot);
6.  MEMID_array =
    (int*)malloc(sizeof(int)*neibpetot);
7.  halo_setup(neibpetot, neibpe,
8.                stack_import, stack_export,
9.                MEMID_array, laddr, raddr, wr, ws);
10. for(i=0;i<loop;i++){
11.     MPI_Barrier(MPI_COMM_WORLD);
12.     time = MPI_Wtime();
13.     rdma_put(x, neibpetot, neibpe,
            stack_import, stack_export,
            nod_import, nod_export,
            laddr, raddr, ws, wr);
14.  MPI_Barrier(MPI_COMM_WORLD);
15.  time = MPI_Wtime() - time;
16.  n_time[i] = time;
17.  spmv(val, col_ind, row_ptr, rowsize, x, y);
18. }
19. Rdma_free(MEMID_array);
20. free(laddr,raddr,MEMID_array)
21. FJMPI_Rdma_finalize();
```

図 2　コードの変更点（左：MPI　右：RDMA）

赤文字となっている部分が MPI コードから RDMA コードへの変更点である．図 2 では隣接通信を halo_exchange 関数で行い，spmv 関数により，各プロセスで並列に処理を行っている．図内の 1. が RDMA を使用するための INIT 文である．2. 3. が RDMA 通信に必要な変数の宣言である．4. 5. 6. で宣言した変数のアロケート 7. で RDMA 通信準備関数の呼び出し，13. が RDMA 通信関数の呼び出し，19～20. で変数のフリー処理である．以上の手順によって，MPI 通信を RDMA 通信に置き換えている．

5　おわりに

MPI コードから RDMA コードへ通信関数の置き換えによって，RDMA 化が行える機構を提案した．これにより，SpMV の隣接通信に対して，最大で 55%の通信時間を削減した．

今後の課題として，隣接通信が多く現れるアプリケーションへ適用し，有用性を評価する必要がある．

参考文献

[1] 東京大学–FX10 スーパーコンピュータシステム (oakeaf-fx), http://www.cc.utokyo.ac.jp/system/fx10/

[2] 中尾昌広，佐藤三久，京速コンピュータ「京」における CGPOP Miniapp の性能評価　著：情報処理学会第 139 回 HPC 研究会，(2013)

[3] 森江善之，南里豪志，直接網において複数の通信デバイスを有効に使用する隣接通信アルゴリズムの提案　著：情報処理学会論文誌コンピューティングシステム，(2015)