ハイパフォーマンス
コンピューティング 62－8
（1996. 8. 28）

HPFにおけるBlock-Cyclic分割を含む配列の再分割

郷田 修

日本アイ・ビー・エム（株）東京基礎研究所

本論文ではblock-cyclic分割を含む配列の再分散(redistribution)について述べる．block-cyclic分割は負荷バランスと通信量の削減とを両立させるために行列計算等の分野で広く使われており，このためblock-cyclic分割を含む配列の再分散の効率化は重要な課題である．ここで述べる再分割ルーチンは配列分割に現れる規則性をも通信集合を求めるための計算量を少なくするとともに，この計算結果をもとにMPIを直接呼び出すことにより、バッファへのパック、アンパック時間と，バッファのために必要な記憶域を不要にする．

Redistribution of Arrays with Block-Cyclic Distribution in HPF

Osamu Gohda

IBM Japan, Tokyo Research Laboratory

This paper discusses the redistribution of HPF arrays with block-cyclic distribution. The block-cyclic distribution is commonly used in areas such as matrix computations to take advantage of reduced communication costs and load balancing among processors. The array redistribution algorithm presented here minimizes the cost for caluculating the communication set by considering the regularity of distributions, and reduces the communication and storage costs by directly mapping the culculated result into the MPI data types.

1 はじめに

近年, 分散メモリマシンの普及にともない, 標準プログラミング言語としてHPF (High Performance Fortran) [1] が注目を集めている. HPFではdistribute指示行等を使って,配列のプロセッサへの分割方法を指定するとともに, 実引数配列と仮引数配列に異なった分割を指定することにより, 副プログラムの境界で配列を再分割する事ができる.

一般に, HPFが使われるような分散記憶型の多重プロセッサ環境では, 計算の効率は配列の分割方法に大きく依存することが多く, またプログラムの各部分で好ましい分割は異なる. またPESSL [6] のような科学計算ライブラリをHPFプログラムから呼び出す場合にも, ライブラリが受け付ける分割に変更して配列を引き渡す必要がある. したがて配列の再分割の効率はプログラム全体の効率に大きく影響する.

本論文では, 再分割の中でも特に効率の問題が生じやすいblock-cyclic分割[1] を含む配列の再分割について議論する.

効率の観点からみてblockとcyclicだけを扱う場合と, block-cyclic分割を扱う場合では以下の点が異なる.

1. block-cyclic分割を含まない場合は, 必要とする通信は各プロセッサの組についてひとつの配列セクションだけである. 一方block-cyclic分割を含む場合は, 一般に必要とする通信は各プロセッサの組について配列中の複数個のセクションになる. また, これらの通信を求めるための計算も複雑になる.

2. block-cyclicについては各プロセッサの組で複数の配列セクションを交換するために, 通信の効率を考えて送受信の前後で配列, バッファ間でのパック, アンパック等を考慮する必要がでてくる.

本論文ではこれらの問題を考慮し, はじめに配列の再分割において, 各プロセッサ間で行われる通信の求めるための計算の量を, ブロックサイズや添字集合の規則性に着目して少なくする方法を示す. 次にこうして計算された結果がMPI [5] がサポートするデータ型にマップすることにより, 容易に, バッファ等を経由せずに直接配列領域間で送受信する方法を示す. また, この方法をRS/6000 SP/2上で実装した結果を示す.

[1]HPFにはblock-cyclicという分割の指定はないが一般にHPFのcyclic(1)をcyclic, cyclic(n)(n>1)をblock-cyclicと呼ぶことが多いのでこれに従う.

2 通信集合の計算

配列の再分割を行うために必要に通信を通信集合と呼ぶことにする. 配列の再分割では始めにこの通信集合を求め, これをもとに通信を行う. ここでは通信集合の計算法につしてのべる.

2.1 添字集合の表現

HPFでは配列をdistribute指示行等をつかってプロセッサ配列に分割するが, これを配列の各次元ごとに, 添え字の集合と対応するプロセッサ配列の上のインデックスをつ使って

$$S_A i = \{..., ([l_k : u_k : s_k], p_k), ...\}$$

のように表わすことにする. ここで$S_A i$は配列Aのi次元目に対する分割の集合, l_k, u_k, s_kはそれぞれ添え字の下限, 上限, ストライド, p_kはプロセッサインデックスである. これはi次元目の添字が$[l_k, u_k, s_k]$に含まれる配列要素は対応するプロセッサ配列上のインデックスp_kをもつプロセッサに割り当てられていることを示す.

たとえば再分割前の配列を$A(16,16)$, 再分割後の配列$B(16,16)$[2], これらの分割対象となるプロセッサ配列をそれぞれ$p(2,2)$, $q(2,2)$, Aの分割を(block,block),Bの分割を(cycylic,cyclic(4)) とすると分割集合は次のようになる. [3]

$$\begin{aligned} S_{A1} &= \{([0:7:1],0),([8:15:1],1)\} \\ S_{A2} &= \{([0:7:1],0),([8:15:1],0)\} \\ S_{B1} &= \{([0:14:2],0),([1:15:2],1)\} \\ S_{B2} &= \{([0:3:1],0),([4:7:1],1), \\ &\quad ([8:11:1],0),([12:15:1],1)\} \end{aligned}$$

2.2 添字集合の縮小

通信集合は, 前節に示した添字の分割集合をもとに計算するが, ここでは通信集合を求めるための計算量を削減するための分割集合の縮小法を提案する. なお

[2]分割前後の配列を区別するために別の名前を用いる

[3]配列およびプロセッサ配列の添字は0から始まるように正規化してあるものとする.

通信集合の計算は配列の各次元ごとに行うことができるので, 一般性を失うことなく一次元配列のみを扱う. また以下の議論では再分割前のブロックサイズおよびプロセッサ数をそれぞれm_1, p_1, 再分割後のブロックサイズとプロセッサ数をm_2, p_2とする.

分割集合の縮小は次の2つである.

1. rを$GCD(m_1, m_2)$とし, m_1, m_2をそれぞれ$m_1' = m_1/r$, $m_2' = m_2/r$に, 配列のサイズNを$N' = N/r$に変更する.

2. Nを$N' = MAX(N, LCM(m_1p_1, m_2p_2))$に変更する.

最初の縮小はブロックサイズが小さなblock-cyclic分割で有効である. たとえば例で用いている配列の場合, $GCD(m_1, m_2) = 4$となるから2次元目の添字集合を次のように縮小できる.

$$S_A = \{([0:2:1], 0), ([2:3:1], 1)\}$$
$$S_B = \{([0:2:2], 0), ([1:3:2], 1)\}$$

これはblock(8)からcyclic(4)への再分割を, block(2)からcyclic(1)への再分割として扱えるようにする. この例の場合, 縮小によって配列Bの分割集合の大きさは4から2に減少する. なおこの縮小は結果としてm_1またはm_2が1になる場合のみ意味があるのでm_1またはm_2が$GCD(m_1, m_2)$に等しい時のみ行う.

2番目の縮小は分割の繰り返しパターンが存在する場合に, 繰り返しの最初の部分だけを計算するための縮小である. これは再分割前と再分割後の分割がともにblock-cyclic(cyclic(1)を含む)の場合に有効である. たとえば, 配列サイズ$N = 12$, $p_1 = 2$, $p_2 = 2$, $m_1 = 2$, $m_2 = 3$とすると, 添字集合は次のようになるが:

$$S_A = \{([0:1:1], 0), ([2:3:1], 1),$$
$$([4:5:1], 0), ([6:7:1], 1),$$
$$([8:9:1], 0), ([10:11:1], 1)\}$$
$$S_B = \{([0:2:1, 0], [3:5:1], 1),$$
$$([6:8:1], 0), ([9:11:1], 1)\}$$

$LCM(p_1m_1, p_2m_2) = 6$であるから, これを次のように縮小できる.

$$S_A = \{([0:1:1], 0), ([2:3:1], 1),$$
$$([4:5:1], 0)\}$$
$$S_B = \{([0:2:1], 0), ([3:5:1], 1)\}$$

2.3 通信の計算

前節で求めた縮小された分割集合をもとに, 各プロセッサは再分割の際に他のプロセッサへ送られる添字集合(送信集合)C_sと, 他のプロセッサから受け取るべき添字の集合(受信集合)C_rを次のように求める. なおC_sとC_rはプロセッサインデックスと添字集合の組$([l:u:s], p)$からなり, C_sの場合pは送信先のプロセッサインデックス, C_rの場合は送信元のプロセッサインデックスである. またp_sは再分割前のプロセッサ配列上での自分自身のプロセッサインデックス, p_rは再分割後のプロセッサ配列上での自分自身のプロセッサインデックス, S_AとS_Bはそれぞれ再分割前と再分割後の分割集合である.

```
for each i_a in S_A
  for each i_b in S_B
    let i_a = ([l_a : u_a : s_a], p_a)
    let i_b = ([l_b : u_b : s_b], p_b)
    if p_b = p_r then
      C_r = C_r ∪
        ([l_a : u_a : s_a] ∩ [l_b : u_b : s_b], p_b)
    endif
    if p_a = p_s then
      C_s = C_s ∪
        ([l_a : u_a : s_a] ∩ [l_b : u_b : s_b], p_b)
    endif
  endfor
endfor
```

例えば前節の最後に示した分割集合の場合, プロセッサインデックスが0のプロセッサに対する通信集合は次のようになる.

$$C_r = \{(0, [0:1:1]), (1, [3:3:1])\}$$
$$C_s = \{(0, [0:1:1]), (1, [4:5:1])\}$$

配列の再分散においては, ここに示した通信集合の計算または類似の計算が必要となるが, 計算量は再分割前の分割集合の大きさと, 再分割後の分割集合の大きさの積に比例する. 従って分割集合の大きさを縮小できれば計算量は少なくなる. たとえば1000要素の配列を4つのプロセッサに分割するものとし, これをcyclic分割からcyclic(50)分割にするばあい, 縮小なしの場合cyclic側の分割集合の大きさは4となるが, cyclic(50)側の分割集合の大きさは20となる. 一方, 縮小を行う場合には(2)の縮小が適用できて, cyclic(50)側の分割

集合の大きさを4とすることができる. 従って計算量は約5分の1になる.

2.4 添字集合の展開

上記で行われた通信の計算は縮小された添字集合に基づいているので, これを展開して縮小前の添字集合にもどす. (ただしここで示す添字集合の展開は, 実際には通信集合をMPIのデータ型に変換するときに行う.)

添字集合の縮小で述べた(1)と(2)の方法で縮小された添字集合は以下のように, 元の配列に対する添字集合に展開する.

(1)の縮小に対する展開 縮小前のブロックサイズをm_1,m_2とし,$r = GCD(m_1, m_2)$とする. 通信集合の各要素$[l, u, s, p]$をつぎのように展開する.

$s = 1$のとき:

$$([lr, (u+1)r - 1 : 1], p)$$

$s = 1$のとき:

$$\{([kr : kr + r - 1 : 1], p) | \\ k = l, l + s, ..., u\}$$

(2)の縮小に対する展開 通信集合の各要素$([l : u : s], p)$を次の$n = \lceil N/N' \rceil$個の要素に展開する.

$$\{([l + rk : u + rk : s] \cap [0 : N - 1 : 1], p) | k = \\ 0, 1, ..., n - 1\}$$

3 通信の生成

上記通信の計算結果にもづいて以下の手順で通信を行う.

3.1 インデックスの変換

通信の計算においては配列の添字としてグローバルインデックス(配列宣言のインデックス)を用いるが, 各プロセッサはグローバル配列のうちの持ち分だけを連続した記憶量域に持っている. 従って通信ライブラリの呼び出しにを行う前にグローバルインデックスを各プロセッサごとのローカルインデックスに変換する.

3.2 実プロセッサへのマップ

これまでの段階では, 再分割に必要に通信は配列の各次元につて, その次元に対応するプロセッサ配列インデックス間で通信されるべき添え字の集合として計算されるが, 実際に通信ライブラリを呼ぶ前にこれを配列セクションと1次元のプロセッサ番号に変える. たとえば, あるプロセッサの配列の1次元目,2次元目の送信集合をC_{s1},C_{s2}を

$$C_{s1} = \{([1 : 3 : 2], 0), ([5 : 7 : 2], 1)\}$$
$$C_{s2} = \{([4 : 7 : 1], 1)\}$$

とする. これは配列セクション$[1 : 3 : 2][4 : 7 : 1]$をプロセッサ$(0, 1)$に,$[5 : 7 : 2][4 : 7 : 1]$をプロセッサ$(1, 1)$に送ることを意味するが, 再分割後のプロセッサ配列を$(2, 2)$とし, 行が最初に変化する順序で実プロセッサ0から3に割り当てるとすると最初のセクションを実プロセッサ2に, 2番目のセクションを実プロセッサ3に送ることになる.

3.3 MPIデータ型へのマップ

MPIでは基本的のデータ型(int, long, doubleなど)をもとに, これらを要素とす配列セクション(MPI_Type_vecotor)や構造体(MPI_Type_struct)を定義することができ, またこのようして定義した型を要素とする配列セクションや構造体も定義できる.

評価のための再分割の実装ではこれを利用して縮小した通信集合を展開し, バッファへのパックやアンパックをおこなわず直接配列から配列はデータを送るようにした. 通信集合が縮小形になっていると繰り返しパターンが明確になるので効率の良いMPIデータ型への変換が行える.

実際の通信を行うには, このデータ型を指定してMPIの送受信ルーチンを呼出す.

4 RS/6000 SP/2による評価

以上で述べた配列の再分割法を実装しRS/6000 SP/2上で評価した. 評価した項目は, 添字集合の縮小による効果, パック/アンパックを行わないことによる効果, およびいくつかの分割に対する再分割の効率の評価である. なお通信方法としてはノンブロッキングの送受信(MPI_Isend, MPI_Irecv)を使用した.

	(block,block)	(cyclic(50),cyclic(50))
全計算	0.03(3%)	0.28(18%)
縮小計算	0.04(4%)	0.06(6%)

表 1: 通信集合の計算時間の比較例

	通信時間	転送量
Pack/Unpack	0.23	26
直接呼出し	0.19	32

表 2: MPIを直接呼び出した場合の効果

4.0.1 添字集合の縮小

通信集合を求めるための計算量の問題は(block, block)から(cyclic, cyclic)といた再分割では, 実際の通信に要する時間に比べて無視できるが, たとえば(cyclic,cyclic)から(cyclic(50),cyclic(50))といた再分割の場合, 添字集合が大きくなることからこれを無視できなくなる. 一例として, 表1に2000x2000の配列を2x2のプロセッサ配列上で(cyclic,cyclic)から(block,block)および(cyclic(50),cyclic(50))に再分割したときの, 通信集合の計算に要した時間を示す. (計算時間にはblock-cyclic配列のグローバルインデックスをローカルインデックスに変換する時間も含む.) 括弧内は再分割に要した時間に対する計算時間の割合である. (block,block)への再分割ではどちらの方法でも全体の時間の数パーセントであるが, (cyclic(50),cyclic(50))の場合では大きく異なり, 添え字の縮小の効果が現われている.

MPI直接呼び出し 表2に(1)MPIを直接呼び出した場合と, (2)送信側でに配列セクションをバッファにパックしこれを受信側でバッファ受けアンパックした場合の効率の違いを示す. なお, パック, アンパックにはMPI_PackとMPI_Unpackを使用した. 例に示した再分割の場合, 各プロセッサは自分自身へのコピーを除いて約6Mbyteのデータの送受信をするが, MPIを直接呼び出した場合, 通信時間から換算すると約32Mbyte/秒の転送量となり, これはRS/6000 SP/2のプロセッサとスイッチ(HPS)を結ぶチャネルの転送速度(40Mbyte/秒)の約80%にあたる. 一方, パック/アンパックを行った場合転送量は約26Mbyte/秒になる. また再分割で使用される作業用の主記憶使用量は, MPIを直接呼び出す場合は通信の計算結果等を保持するための領域だけですむが, バッファを使った場合, すべてのプロセッサが同時にプロセッサ間で一度だけ送受信するようにした場合, 分割された配列の約2倍となる. これは, ここで示した例の場合6Mbyteとなる.

再分割の効率 表3は2000x2000の整数型(4バイト)配列を(*,cyclic)分割から(*,block)および(*,cyclic(50))分割に変更する場合の時間と通信の転送率(再分割に必要な通信バイト数を通信に要した時間で割ったもの)である. 使用できるプロセッサ数が最大16であったことと, メモリ参照パターンの影響を避けるため, 配列の2次元目のみを分割して測定した.

5 関連研究

HPF配列の再分割およびこれに関連した議論はこれまでにもいくつか行われており, 大部分がblock-cyclic分割を扱っている. 右辺と左辺が異なった分割を持つ配列代入は配列の再分割と類似しており, このときの通信の計算をclosed formとvirtual processorの概念を使って行う方法が[4]に示されている. [2]と[3]は, ともにblock-cyclic分割を含む配列の再分割を扱っている. [2]ではblockからcyclic, cyclic(x)からcyclic(kx)などの組み合わせに対して再分割のアルゴリズムを示しており, 一般のblock-cyclic間の再分割については各配列要素について送受信プロセッサを計算するよう提案している. [3]はcyclic(x)からcyclic(kx)の場合をおもに扱っており, Intel ParagonとIBM SP-1による結果を示している.

6 まとめ

以上, block-cyclic分割を含む配列の再分割法について述べた. 一般に配列の再分割では最初に必要な通信を計算し, そのあとこれに基づいて実際の通信を行う. 通信の計算に要する時間は, 実際の通信に要する時間に比べて無視できるように思われるが, 実現方法によってはblock-cyclic分割を含む場合はこれを無視

	(*,C)-(*,B)		(*,C)-(*,C(50))	
プロセッサ数	分割時間(秒)	転送率(MB/秒)	分割時間(秒)	転送率(MB/秒)
2	0.24	33	0.23	35
4	0.21	29	0.20	30
8	0.12	29	0.12	29
16	0.09	21	0.09	20

表 3: 再分割時間と通信の転送率

できない. 本論文で提案した方法によれば多くの場合について, これを十分小さくできる. また, 計算結果は直接MPIデータ型に変換可能であり, バッファへ一度コピーしてから送るといった手間を省くことができ時間と記憶域の節約になる. ここでは議論しなかったが, ここで述べた添字集合の縮小による通信の計算方法は, HPFのALIGN指示行が使われた場合にも比較的簡単に適用可能である.

参考文献

[1] High Performance Fortran Forum, "High Performance Fortran Language Specification (Version 1.0)," Rice University, May 3, 1993.

[2] R. Thakur, A. Choudhary, G. Fox, "Runtime Array Redistribution in HPF Programs," in Proceedings of IEEE Scalable Hight Performance Computing Conference, 1994.

[3] D. W. Walker, S. W. Otto, "Redistribution of Block-Cyclic Data Distributions Using MPI," ORNL/TM-12999, Ork Ridge National Laboratory, June 1995. S. D. Kaushik, C. H. Huang, J. Ramanujam, and P. Sadayappan, "Multi-Phase Array Redistribution: Modeling and Evaluation," in Proceedings of 9th International Parallel Processing Symposium, 1994.

[4] S. K. Gupta, S. D. Kaushik, S. Mufti, S. Sharma, C.-H. Huang, and P. Sadayappan, "On Compiling Array Expression for Efficient Execution on Distributed-Memory Machines," in Proceedings of 1993 International Conference on Parallel Processing.

[5] MPI: Message-Passing Interface Standard, Message Passing Interface Forum, June 1995.

[6] Parallel Engineering and Scientific Subroitine Library, Guid and Reference, IBM Corp. April 1995.