ハイパフォーマンス
コンピューティング 52－1
（1994．7．22）

ストライドデータ転送機構を用いたコード生成

土肥実久, 林憲一, 進藤達也

(株) 富士通研究所 並列処理研究センター

並列化コンパイラにおいて、ストライドデータ転送機構をもつ DMPP 用にコード生成する手法について述べる。VPP Fortran の一括データ転送構文を用いて記述された実際のデータ転送パターンから、ハードウェアで用意されているストライドデータ転送機構を用いるために、ストライドデータ転送パターンを生成する方法について述べ、本手法によるコード生成を AP1000 上に実装したものを用い、その有効性を検証する。

Code Generation Using Stride Data Transfer Mechanism

Tsunehisa Doi, Kenichi Hayashi, and Tatsuya Shindo

Parallel Computing Research Center, Fujitsu Laboratories Ltd., Kawasaki, Japan

This paper describes the code generation in a parallelizing compiler with stride data communication for DMPPs. We discuss the method for extracting the stride data communication pattern from a more general block data communication pattern, and demonstrate the effectiveness of this method based on experiments using the AP1000 implementation.

1 はじめに

分散メモリ型並列計算機の言語として、プログラミングの容易さ、プログラム資産の利用の面から、グローバルアドレス空間のサポートと既存言語を基に拡張することが重要であると考え、我々は AP1000 上に VPP Fortran の処理系を実装している [8][9]。

従来、分散メモリ型並列計算機にこのような言語を実装する際には、メッセージパッシングによる送受信のペアを作る手法が使われてきている[2][3][4]。一方我々は VPP Fortran の実装でダイレクトリモートデータアクセス (DRDA) に基づく方式を採用した[8][10]。これによりコンパイル時にペアの求まらない不規則データアクセスを効率良く実装でき、バッファへのデータコピーのオーバーヘッドを削減することができる[6][10]ことにより、高速なデータ通信を行なうことが可能になる。

本論文では、DRDA においてデータを一括転送する上でのストライドデータ転送の重要性を述べ、VPP Fortran における各種の通信パターンからストライドデータ通信パターンを決定するコード生成法について述べる。

以下、2 章で VPP Fortran におけるデータ転送について述べ、3 章でコード生成の方法について述べたのち 4 章で評価を行なう。5 章で関連する研究について触れ、6 章でまとめる。

2 VPP Fortran におけるデータ転送

まず最初に VPP Fortran のメモリモデルについて述べる。VPP Fortran では図 1 に示す通り、グローバルメモリ空間とローカルメモリ空間の 2 種類のメモリ空間を持ったモデルを採用している。グローバルメモリ空間は必要であれば通信

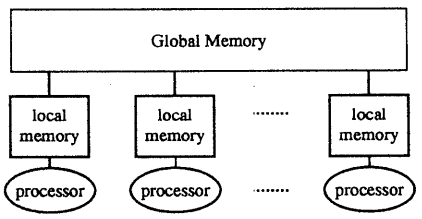

図 1 VPP Fortran のメモリモデル

を伴うが、どのプロセッサからもアクセスが可能である。一方ローカルメモリ空間はそのオーナのプロセッサのみアクセスが可能である。VPP Fortran における通信はグローバル空間においた変数 (グローバル変数) への代入 (write 転送)、もしくはグローバル変数の参照 (read 転送) といったグローバル変数へのアクセスの形として表現される。List 1 にグローバル変数のアクセス例を示す。

List 1 グローバル変数のアクセス例

```
!xocl processor P(10)
      real a(100)
!xocl global a(/(p))
      ......
      do i=1,100
        a(i) = ...        ! write
      enddo
      do i=1,100
        .. = ... + a(i)  ! read
      enddo
```

List 1 のように 1 要素づつグローバル変数に対してアクセスを行なうと、その度に通信が発生する。一般に分散メモリ型並列計算機の通信機構は一回の通信毎に立ち上がりオーバヘッドが存在し、転送すべきデータの総量が等しければ、通信の回数が少ないほど通信にかかる時間が少なくて済むという性質を持っている。したがって高い性能を得るためには一括データ転送機能が重要となる。VPP Fortran ではユーザが一括転送を指示できるように、2 種類の一括データ転送構文を持っている。次に一括データ転送構文について述べる。

2.1 VPP Fortran における一括データ転送構文

2.1.1 spread move 構文

spared move 構文は、グローバル変数とローカル変数との間で一括データ転送を記述するための構文である。spread move 構文は、List 2 に示

List 2 spread move の例

```
      real a(100),b(100)
!xocl global a(/(p))
!xocl local b(/(p))
      ....
!xocl spread move /(p)
      do i=1,100
        b(i) = a(i)       ! read
      enddo
!xocl end spread(r)
!xocl movewait(r)
      ....
!xocl spread move /(p)
      do i=1,100
        a(i) = b(i)       ! write
      enddo
!xocl end spread(w)
!xocl movewait(w)
```

すように VPP Fortran ディレクティブ **spread move** と **end spread** で囲まれたDO loop とループ内のグローバル変数とローカル変数との間の代入文として記述される。List 2 の例では 1 つめ

spread move ではグローバル変数 a からローカル変数 b への一括 read 転送を記述しており、2 つめの spread move ではローカル変数bからグローバル変数 a への一括 write 転送を記述している。

2.1.2 overlapfix 文

VPP Fortran では分割されたローカル変数を記述する際、図 2 に示すように境界領域に重なりを持たせるような分割を指定することができる。この領域を overlap 領域[5]といい、この領域の値をオーナの値で更新する操作を overlapfix 文で指示する。overlapfix 文の例を List 3 に示す。

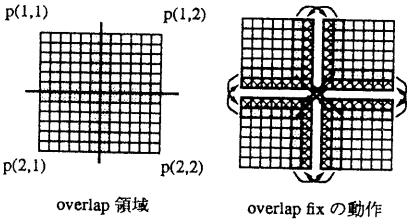

図 2 overlap 領域と overlap fix

List 3 overlapfix の例

```
!xocl processor p(2,2)
      real ga(12,12),a(12,12)
!xocl global ga(/(p.1),/(p.2))
!xocl local la(/(p.1,overlap=(1,1)),&
               /(p.2,overlap=(1,1)))
      equivalence(ga,la)
      ....
!xocl overlapfix(la)(o)
!xocl movewait(o)
      ....
```

2.2 DRDA におけるストライド転送の重要性

2.2.1 DRDA の利点

我々は VPP Fortran におけるデータ転送の実装法として DRDA を用いた実装を行なっている[8]。DRDA を用いることによって次のような点が有利である。

(1) コンパイル時に通信ペアの決まらない不規則データアクセスを効率良く実装できる。[8]

(2) 直接メモリに対する転送を行なう実装を採用することで、バッファへ (バッファから) のコピーのオーバーヘッドが削減できる。[6][10]

2.2.2 ストライドデータ転送の重要性

DRDA におけるデータ転送は、ローカル側メモリとリモート側メモリの間で直接行なわれる。転送対象のデータがローカル側・リモート側の両者でのメモリ上で連続領域に格納されている場合はその固まりを一つの転送命令で送ることが可能である (図 3(a))。しかし、どちらか一方の側で連続領域ではない場合には連続転送は行なえず、細切れの転送命令によって送らなくてはならなくなる。この典型的な例は 2 次元配列間のトランスポーズ転送がある (図 3(b))。

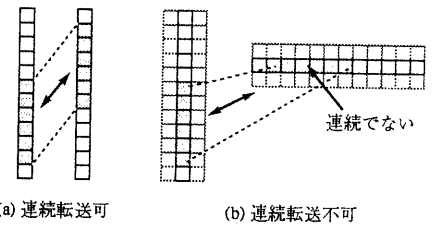

図 3 領域間の転送

先に述べたように、分散メモリ型並列計算機では通信立ち上がりオーバーヘッドの増大を避けることが高い性能を得るために重要であり、そのためにこのような連続でない領域間のデータ転送を行なえる機能が必要となる。このために我々はストライドデータ転送機能を利用している。

本論文ではストライドデータ転送として図 4 に示すように、一定間隔離れた同じ大きさの領域をアクセスする DRDA を想定する。図 4 で *Start* はストライドパターンの始まりのアドレスを示し、*Size* は転送単位の大きさ、*Stride* は次の転送単位が現れるまでの大きさを示し、*Count* は転送単位の数を示している。

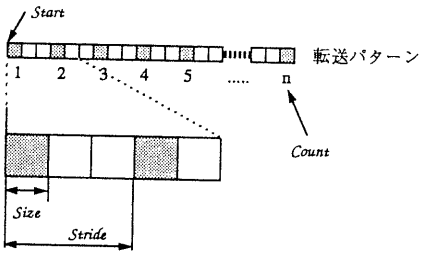

図 4 ストライドパターン

3 コード生成

本章では、プログラマにより記述された一括転送の構文から、前述のストライドデータ転送を用いた一括データ転送のコードを生成する手法について spread move 構文をとりあげて述べ

る。

3.1 spread move 転送

まず spread move 転送の場合について図5 にような例を用いて述べる。spread move は先に示し

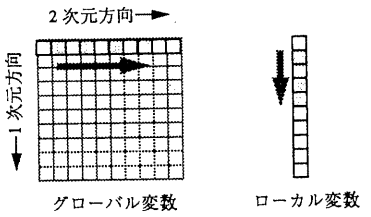

図5 spread move の動作

た通り do loop とその中にあるローカル変数とグローバル変数との間の代入文によって表現される。これは図 6 に示すように疑似的なリモート

```
      real g(10,10),l(10)
!xocl global a
      ....
!xocl spread move
      do i=1,4
        b(i+1) = a(2*i,1)
      enddo
!xocl end spread(r)
!xocl movewait(r)
```

```
do i=2,5,1
  b(i) = ...
enddo
```

```
do i=2,8,2
  ... = a(i,1)
enddo
```

ローカル側　　リモート側

図6 spread move の例

側ループとローカル側ループに分けて考えることができる。これらのループの初期値, 終了値, 増分値をそれぞれ *loopStart*, *loopEnd*, *loopStride* として分割形状の種類毎にストライドパターンの生成手法を述べる。

3.2 ストライドパターンの生成

BLock 分割と Cyclic 分割の場合について、ストライドパターン生成の手法を紹介する。ストライドパターン生成とは、一括データ転送を記述した spread move から図4のストライドパターンを決定するパラメータ *Start*, *Size*, *Stride* 及び *Count* を求めることである。これらのパラメータのうち *Size* と *Const* はグローバル側とローカル側で共通であるが、その他のパラメータはグローバル側とローカル側にそれぞれ存在する。今後の説明では G ないし L のサフィックスをつけて区別する。

なおグローバル変数の格納アドレス及びプロセッサはグローバルインデックス空間上の位置から一意に決定できるので、後述の式中で用いる $Start_G$ はグローバルインデックス空間上の位置を示すことにする。ローカル側の転送開始位置 $Start_L$ については $Start_G$ との関係で決定できるのでここでは述べない。*Size* はグローバル側とローカル側に共通な連続領域の大きさとする。また *blocksize* は対象次元方向にインデックスを 1 増加させるメモリ上でのサイズとする。

3.2.1 Block 分割の場合

対象次元に対して Block 分割されている場合

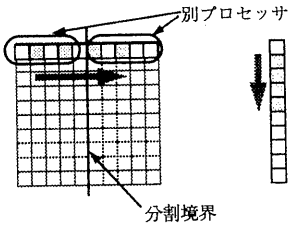

図7 Block 分割の場合

図 7 に示したようにアクセス対象となる領域は対象次元方向に連続でかつ分割境界でプロセッサが切り替わる。したがって求めるパラメータは以下のようになる。

$$Stride_G = loopStride_G \times blocksize_G$$

$$Stride_L = loopStride_L \times blocksize_L$$

$$Start_G = \begin{cases} loopStart_G & (1,2) \\ blockStart_G & (3,4) \end{cases}$$

$$Width_G = \begin{cases} loopEnd_G - loopStart_G + 1 & (1) \\ blockEnd_G - loopStart_G + 1 & (2) \\ loopEnd_G - blockStart_G + 1 & (3) \\ blockWidth_G & (4) \end{cases}$$

$$Count = \frac{Width_G}{loopStride_G}$$

ここで $blockStart_G$, $blockEnd_G$ は分割範囲、$Width_G$ は分割範囲の要素数を示している。(1) はループの範囲が一つの block 内に収まっている場合で、分割のない場合も含めて考えることができる。(2),(3),(4) は通常の場合でループの開始

位置を含むもの，ループの終了位置を含むもの並びにそれ以外の場合である。

3.2.2 Cyclic 分割の場合

対象次元に対して Cyclic 分割されている場合

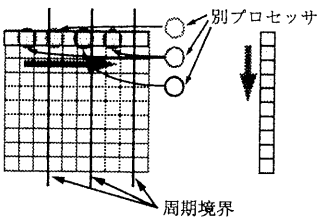

図 8 Cyclic 分割の場合

図 8 に示すように隣合う要素は別プロセッサになっている。分割方向のプロセッサ数を$procNum_G$とすると以下のようになる。

$$GCD_{CS} = GCD(procNum_G, loopStride_G)$$

$$LCM_{CS} = LCM(procNum_G, loopStride_G)$$

$$offset = \left\{ n \middle| \quad 0 \le n \le \frac{procNum_G}{GCD_G} - 1 \right\}$$

$$Start_G = loopStart_G + offset \times loopStride_G$$

$$Stride_G = \frac{LCM_{CS} \times blocksize_G}{Cycle_G}$$

$$Stride_L = \frac{procNum_G \times loopStride_L \times blocksize_L}{GCD_{CS}}$$

$$Count = \frac{loopEnd_G - Start_G + LCD_{CS}}{LCD_{CS}}$$

$GCD(A,B)$, $LCM(A,B)$ はそれぞれAとBの最大公約数、最小公倍数を示している。

4 評価

4.1 実験の目的と内容

この論文で述べた、ストライド機構を使用したデータ転送のコード生成を AP1000 上に実装し実験を行なった。この実験は以下の項目について調べることを目的としている。

(1) 本論文のコード生成法の効果

(2) ストライド転送の必要性

なお実験に使用したプログラムは NAS Parallel Benchmark の中から FT, SP, CG、SPECfp から tomcatv を VPP Fortran を用いて並列化したもの[11]である。

4.2 実験結果

実験結果としてNASPAR の CG, SP, FT の sample size と、SPECfp の tomcatv(Tm) について 16 プロセッサで実行した実行時間、総パケット数、平均パケット長を stride を使用したものと使用しなかったものについて表 1 にまとめた。

表 1 NASPAR-sample, tomcatv (16PE)

	実行時間	総パケット数	平均パケット長
CG_{no}	11.482	6240	700.0
CG_{stride}	11.491	6240	700.0
SP_{no}	15.935	244800	254.1
SP_{stride}	16.097	244800	254.1
FT_{no}	57.284	5181440	17.8
FT_{stride}	21.325	1009664	91.5
Tm_{no}	35.949	1645832	8.0
Tm_{stride}	22.894	101640	129.5

またストライド転送の効果が確認された FT と tomcatv について stride を使用した場合と使用しなかった場合のパケットサイズの分布を図 9, 図 10 に示した。

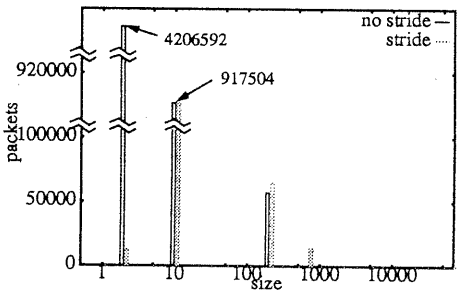

図 9 パケット分布 (FT-sample,16PE)

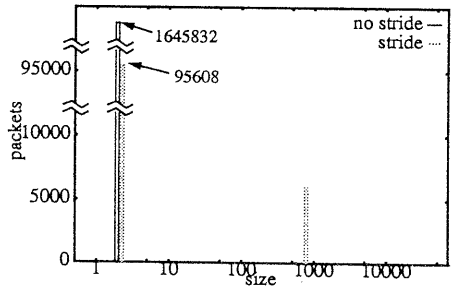

図 10 パケット分布 (tomcatv,16PE)

4.3 考察

本論文で提案するストライドを用いたコード生成は、用いない場合に比べて FT で約 520万回から 100 万回に、 tomcatv で約 160万 回から 10 万回にデータ転送回数を大幅に減少させること

ができた。また性能の比較では FT で約 57 秒から 21 秒へ、tomcatv では約 36 秒から 25 秒へと大きく性能を向上させることに役立っている。

今回実験に用いたプログラムではストライドデータ転送が現れるプログラムは FT と tomcatv の 2 つであり、必ずしもすべてのプログラムにおいてストライド転送が現れるわけではないことがわかったが、ストライドデータ転送を必要とする 2 つのプログラムについては、ストライド転送のサポートが性能を向上させることに大きく寄与できることが確認できた。

5 関連する研究

関連する研究としては Fortran-D[2] や Kali[3] などの VPP Fortran 同様の機能を実現している言語があるが、その実装にはメッセージパッシングを用いている。Split-C [7]は Active message[6] を用いて言語を実装しているが、本論文で述べたようなストライド転送については述べられていない。Gerndt の message vectorizing[5] では 1 次元の連続転送のみを対象に検出しているが、我々は多次元を対象にしている。なお今回は VPP Fortran の実装について述べたが、HPF[1] における異なる分散を持つ配列間の代入や再分散は同様にストライド転送を用いて扱うことができる。事実我々は、現在 FLoPS コンパイラプロジェクトにおいて HPF と VPP Fortran の両方を扱えるコンパイラを作成しており、HPF は内部的に VPP Fortran に変換されてコンパイルされる[12]。また AP1000+[10] の設計は本検討内容が反映されており、DRDA を現行の AP1000 上に実装するためにソフトウェアで実現されている機能の多くがハードウェア化され、性能向上がはかられている。

6 まとめ

本論文では DRDA でデータを一括転送する上でのストライドデータ転送の重要性を述べ、実際の転送パターンをストライドパターンにマッピングするコード生成手法についてのべ、実験結果によりその効果を示した。

7 謝辞

本研究に対して御指導、御支援をいただいている石井光雄氏、白石博氏、佐藤弘幸氏、池坂守夫氏、日頃有意義な議論をしていただいている岩下英俊氏、萩原純一氏、金城ショーン氏、AP1000 上の処理系の実装に際し貴重なアドバイスをいただいた並列センターアーキテクチャグループのメンバーに感謝致します。

参考文献

[1] High Performance Fortran Forum, "High Performance Fortran Language Specification Ver. 1.0," 1993

[2] S. Hiranandani, K. Kennedy, and C. Tseng, "Compiler optimizations for Fortran D on MIMD Distributed-Memory Machines," Proc. Supercomputing '91 pp.86-100, Nov. 1991

[3] Charles Koelbel, and Piyush Mehrotra, "Compiling Global Name-Space Parallel Loops for Distributed Execution," IEEE Trans. onParallel and Distributed Systems, Vol.2, Number 4, pp440-451, Oct. 1991

[4] Roland Rühl, and Marco Annaratone, "Parallelization of FORTRAN Code on Distributed-memory Parallel Processors," International Conference on Supercomputing pp.342-353, Jun. 1990

[5] Michael Gerndt, "Program analysis and transformations for message-passing programs," In SHPCC '92, Apr. 1992

[6] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," Computer Architecture News Vol.20, Number 2 pp.256-266, May. 1992

[7] David E. Culler, Andrea Dusseau, Seth Copen Goldstein, Arvind Krishnamurthy, Steven Lumetta, Thorsten von Eicken, and Katherine Yelick, "Parallel Programming in Split-C," Proc. Supercomputing'93 pp.262-273, Nov. 1993

[8] 進藤達也, 岩下英俊, 土肥実久, 萩原純一, "AP1000 を対象とした VPP Fortran 処理系の実現と評価," SWoPP '93 HPC 研究会, Vol.93 HPC 48-2, pp.9-16, Aug. 1993

[9] 岩下英俊, 進藤達也, 岡田信, "VPP Fortran: 分散メモリ型並列計算機言語," JSPP'94 Proc. pp.153-160, May 1994

[10] 林憲一, 土肥実久, 堀江健志, 小柳洋一, 白木長武, 今村信貴, 清水俊幸, 石畑宏明, 進藤達也, "PUT/GET インターフェースのハードウェアサポートによる並列プログラムの効率的実行," JSPP94 Proc. pp.233-240, May 1994

[11] 金城ショーン, 進藤達也, "VPP Fortran を用いた NAS Parallel Benchmark の並列化と AP1000 を用いた評価," In SWoPP '94, 1994

[12] 萩原純一, 金城ショーン, 土肥実久, 岩下英俊, 進藤達也, "HPF コンパイラの実装と AP1000 を用いた評価," In SWoPP '94, 1994