5T－2

超並列コンピュータ上のFortranにおける疎行列の扱いに関する提案

大谷 浩司† 安村 通晃†
†慶應義塾大学環境情報研究所

1 はじめに

科学計算においては、大規模な疎行列を扱う場合が多い。疎行列をFortran言語で扱う場合は、記憶領域および計算量の削減を行なうために、1次元の配列やその他の特殊な構造に詰め込んで格納する。しかしながら、このような方法をFortranレベルで記述した場合、データ構造が複雑化し配列の間接参照が増加するなどプログラムが複雑化する。そのため、コンパイラがその処理を把握する事が困難になり、コンパイラの最適化を阻害する。とくに、並列コンピュータにおいては配列のデータの配置やプロセッサ間通信のための解析をする際に大きな問題となる。そこで、HPF(High Performance Fortran)[1]の拡張としてデータ分割の宣言の際に、疎行列であることの宣言をし、プログラムの本体においては密行列と同様な記述を行なう方式(コンパイラ・パッキング方式と呼ぶことにする)を提案する。コンパイラは、疎行列であるという情報と、データ依存解析などによる情報を利用して各アーキテクチャに最適なデータ保持法を選択し、最適な実行コードを出すことができる。

以下では、まず、疎行列を詰め込んで記述する従来の方法の利点と欠点をのべて本方式の提案の理由について説明する。次に、本方式の記述法について述べた後、実際の例を示す。最後に人手によってCに変換しAP1000[2]において実行した結果を示し、本方式の有効性を示す。

2 疎行列の扱い

Fortranにおいて疎行列を表すのに、2次元の配列をそのまま使わずに1次元の配列などに詰め込んだ場合、記憶領域が少なくて済む、計算量を少なくできる、ベクタ長を長くできる、という利点がある。

しかし、一方、データ構造が複雑になり、間接参照が増えるために、プログラムが非常に複雑になる。このため、作成や維持改良が困難になる、人手による並列化をするのが困難になる、コンパイラの最適化や並列化を阻害する、移植性が悪くなる、という欠点が生じる。

このような欠点は、Fortranのプログラム・レベルでデータを詰め込んでしまうことから生じている。一方、利点はデータを詰め込むことと、演算を制限することから来ている。そこで、本方式ではコンパイラがデータを詰め込み、演算を制限するようにした。このことによって、プログラムを複雑化する必要がなくなり、利点を生かしつつ欠点をなくすことができる。本方式をコンパイラ・パッキング方式と呼び、従来のようにプログラマが陽にデータを詰め込む方式をプログラマ・パッキング方式と呼ぶことにする。

3 記述法

次のようにHPFでは、distributeデレクティブを使って各次元方向のデータ分割を指定することができる。

```
distribute a(cylic,block)
```

このdistributeディレクティブを拡張し、データ分割法に以下のものを指定できるようにする。

```
SPARSE_CYCLIC    : 疎でかつCYCLICに分割
SPARSE_BLOCK     : 疎でかつBLOCKに分割
SPARSE_COLLAPSED: 疎でかつ同一プロセッサに配置
```

さらに、帯行列等の疎行列の形やフィルイン(fill-in)の頻度なども指定できるようにした方が有用であると思われるが、今回は考えない。

4 例

では、実際の例を見てみよう。図1と図2は、それぞれ不完全コレスキー分解のプログラムを本方式と従来方式とで記述した例である。プログラムは、文献[3]のものを変形した。ふたつのプログラムを比べてみた場合、図2のプログラムは、図1のプログラムに比べて非常に複雑になっているのがわかる。このためdoループ内は簡単には並列化することができない。

次に、前進代入のプログラムを見てみる。図3と図4に、本方式と従来方式によるものを示す。さて、ふた

The proposal on manipulating sparse matrix in Fortran for massively parallel computers, Koji OTANI†, Michiaki YASUMURA†
†KEIO University Institute of Environmental Information

```
  do i = 1,k-1
    l(k,i) = l(k,i) - &
      sum(l(k,1:i-1)*l(i,1:i-1)*dr(1:i-1))
  end do
```

図 1: 本方式を使う場合の記述

```
  do mu = 1,na(k)
    i = ai(mu,k)
    mui = 1
    muk = 1
1   if (ai(mui,i) .gt. ai(muk,k)) then
      muk = muk+1
    else if (ai(mui,i) .lt. ai(muk,k)) then
      mui = mui+1
    else
      lv(mu,k) = lv(mu,k) - &
        dr(ai(muk,k))*lv(mui,i)*lv(muk,k)
      mui = mui+1
      muk = muk+1
    end if
    if ((ai(mui,i) .le. i-1) &
      .and. (ai(muk,k) .le. i-1)) goto 1
  end do
```

図 2: 従来方式による記述

つのプログラムを比べてみると複雑さにおいてはさほど変わりがない。しかし、図 3では、コンパイル時にdoループのデータ依存が後方依存であることが分かる。それに対し、4では配列が間接参照になっているためにコンパイル時にはデータ依存が全く分からない。

行列が密の場合は、このループは逐次実行するしかない。しかし、有限要素法などでの疎行列では並列に実行できる部分があることが分かっている。ところが図4の場合は、依存性が全く分からないために逐次実行をするコードを生成するしかない。また、後方依存であることが分かればループの各イタレーションを並列に実行できない場合でも、イタレーション中の一部の実行を重ねて実行するパイプライン計算を行なうことができる。

5 実行例

先の前進代入の例を人手によって C に変換しAP1000において実行した。疎行列の表現法としては、非ゼロ要素のみを記憶し、それに対応する添字を持つ方法を採用し、本方式ではさらに同一の添字を持つ要素をダブルリンクを使ってつないだ。表 1に実行に要

```
distribute ul(sparse_collapsed, cyclic)
do i = 1, n
  x(i) = ud(i)*(b(i) - &
    sum(ul(1:i-1,i)*(x(1:i-1))))
end do
```

図 3: 本方式による前進代入

```
distribute ul(*, cyclic)
do i = 1, n
  x(i) = ud(i)*(b(i) - &
   sum(ul(1:nzl(i),i)*x(il(1:nzl(i),i)))
end do
```

図 4: 従来方式による前進代入

表 1: 前進代入の実行時間 ($msec$)

行列の形	従来方式	本方式
4ランダム	2300	24
1%ランダム	14000	220
帯行列	2300	500

行列の大きさ 4096 × 4096

した時間を示す。単位は $msec$ である。行列の非ゼロ要素の出現の形は、1 行に 4 要素の帯行列 (表中では帯行列)、1 行に 4 要素の形がランダムな行列 (表中では 4 ランダム)、各行に非ゼロ要素が 1%出現するランダムな形の行列 (表中では 1%ランダム)、の 3 種類を試した。行列の大きさは 4096 行とした。

表を見ると帯行列での差は 4 倍ほどである。帯行列では並列に計算できるイタレーションはないため、この差は主にパイプライン計算の結果であろうと思われる。帯行列以外では、実行時間に 50〜100 倍の差があることが分かる。帯行列以外では、並列計算とパイプライン計算の両方の効果が出ているために大きな差となっている。

6 終りに

本稿では、Fortran 上で疎行列を扱う方法コンパイラ・パッキング方式を提案した。前進代入を、AP1000の上で実行した結果、従来の方式に比べて非常に高速に実行できることが分かった。今後の課題としては、その他の計算での効果の検証やフィルインがある場合などの検討が必要となる。また、コンパイラへの本方式の組み込みについても検討が必要である。

参考文献

[1] *High Performance Fortran Language Specification Version 1.0*, High Performance Fortran Forum, May 3, 1991

[2] H. Ishihata, T. Horie, and T. Shimizu, *Architecture for the AP1000 Highly Parallel Computer*, Fujitu SCIENTIFIC & TECHNICAL JOURNAL Spring 1993, Vol.29,No.1, pp.6-14

[3] 森 正武, "FORTRAN77 数値計算プログラミング", 岩波書店