プログラミング 10－1
（1996．10．31）

分散メモリ型並列計算機を対象とする並列化トランスレータ

忠政 慎也　　小畑 正貴

岡山理科大学工学部

大規模な数値計算では、並列化による高速化が期待され、かつ、さまざまな研究がされている。しかし、並列プログラムの記述は、並列可能部分の認識、通信関数の導入など、あらゆる困難があり、ユーザには使いにくいのが現状である。そこで、我々は、分散メモリ型並列計算機を対象に、逐次プログラムから並列プログラムに自動的に変換する並列化トランスレータの作成を試みている。本稿では、現状での変換手法とその実装及びその評価について示す。

A Paralellizing Translator for Distributed Memory Machines

Shinya TADAMASA and Masaki KOHATA

Faculty of Engineering, Okayama University of Science

In large scale numerical analys, high speed computation by parallel processing is expected and various subjects has been studying. There are many difficult things in description of parallel programs, such as understanding paralellizable part and insertion of communication function. We are developing a parallelizing translator which translates sequential programs to parallel programs for distributed memory machine. This paper presents a scheme of translation and its implementation and its evaluation.

1 はじめに

近年、流体計算など数値計算を必要とするさまざまな分野から、並列処理により高速化が期待され、かつ、盛んに研究されている。しかし、並列プログラムは、逐次プログラムと比べて、並列化可能部分の認識や、通信関数の導入など、さまざまな困難が考えられる。特に、分散メモリ型並列計算機では、並列処理の効果がデータの分割配置によって決定されてしまうため、その認識も必要となる。そこで、我々は、分散メモリ型並列計算機を対象とし、逐次プログラムから並列プログラムへの変換を自動的行なえる並列化トランスレータの作成を試みている。

並列計算機には、Intel ParagonXP/S-5[2](以下、単にParagonと呼ぶ)を用いている。また、FORTRAN言語またはC言語に通信関数群NXライブラリを用いることで、並列プログラムを記述できる。現状では、FORTRAN言語を対象としている。

分散メモリ型並列計算機で並列実行を考える場合、ノードへのデータ分割配置が並列処理の効果を決定し、重要である。また、1ノード当たりのデータ量を最小限にするには、均等にデータを分割し各ノードに割り当てればよいが、これはノード間の通信を増やす可能性がある。一方、全データを各ノードに割り当てることにより通信が減る可能性があるが、より大規模な問題を扱うために1ノード当たりのデータ量は最小限にするのが望ましい。このように1ノード当たりのデータ量の削減(記憶領域の有効な利用)をとるか、通信時間の削減(高速化)をとるかの選択が強いられる場合がある。

現状では、特に計算時間がかかると予想する部分(以下、メインブロックと呼ぶ)から、データ量の大半を占める配列データの分割とそのノードへの配置について検討している。また、変換結果として得られる並列プログラムは、メインブロックの最外ループを分割した並列化を基本としている。

以下第2節では分散メモリ型並列計算機Paragonとその並列プログラムの特徴を示し、第3節では作成している並列化トランスレータの変換手順を概説し、第4節では行列の積を求める逐次プログラムを実際に変換したときの、その台数効果による評価を行ない、第5節でまとめる。

2 対象とした並列計算機とそのプログラムの特徴

対象としている並列計算機Paragonは、2次元メッシュのネットワークで形成された分散メモリ型の並列計算機である。並列記述には、FORTRAN言語にメッセージパッシングの通信関数群NXライブラリを用いて記述することができる。

NXライブラリの通信関数には以下のようなものがある。

csend() 同期型のデータ送信を行なう。

crecv() 同期型のデータ受信を行なう。

gsync() バリア同期をする。

また、Paragon のような分散メモリ型並列計算機で実行する並列プログラムの特徴としては、1．各ノードはローカルなメモリを持っているため、他のノードのデータを参照する場合、通信の記述が必要となる。2．データ領域の分割とその割り当てを明示しなければならない。等が挙げられ、ユーザにより並列プログラムの記述をするのは困難である [1]。

3 並列化トランスレータ

現状では、並列処理の効果が極めて高く得られるループをもつようなプログラムにおいて検討している。また、並列処理の効果を高くするような前処理は行なっていないので、得られる並列プログラムは逐次プログラムに依存した形となっている。

得られる並列プログラムの特徴としては、特に計算時間がかかると予測する部分(メインブロック)に並列処理の効果があらわれ、配列データを分割して割り当てても問題ない場合は、分割した大きさで宣言される。

以下に、現状での変換の流れ、及び、1ノードに割り当てる配列の大きさの決定手法を述べる。

3.1 変換手法

並列化トランスレータの変換方法は以下の流れとなる。メインブロックとは、最も計算時間がかかると推測したブロックであり、最も並列処理の効果を引き出すよう変換するブロックのことである。

入力と字句解析

FORTRAN ソースプログラムの入力をし、その入力プログラムを字句解析によりトークンに変換する。

実行するノード数をユーザが指定する。

ブロック設定

入力プログラムをブロックに区切る。現状では、1ループ(ネストループも含む)のみ、ループ外にある代入式のみとなるように区切っている。また、メインブロックは、ブロック内のループのイタレーション回数と代入式の代入と四則演算の数から推測している。

データ分割配置の決定

メインブロック内の配列データにおいて、配列要素間にイタレーションにまたがるデータ依存があるか否かを解析し、その解析結果からデータ分割配置を決定する。

メインブロックの並列化

決定したデータ分割配置に従って、メインブロック内の並列化変換を行なう。

その他のブロックの変換

メインブロック以外のブロックに関しては、データ分割配置とメインブロック内へのデータの流れ、または、メインブロック内からのデータの流れを損なわないことだけを注目し、変換する。

出力

内部表現を並列プログラムに変換し出力する。

3.2 データ分割配置の決定手順

メインブロック内の並列化変換に適したデータの分割配置を決定することを目的としている。

1. メインブロックの最外ループ変数(DO変数)に注目し、配列要素間にイタレーションにまたがるデータ依存が存在するか否かの解析を行なう。
2. さらに1の解析結果よりループ分割による並列化が可能と判断した場合、どのように並列化をすればよいか解析する。
3. 1,2の解析結果をもとに、1ノード当たりの配列の要素数やもとの配列のどの領域を持っているかなどを求める。
4. 上記からデータ分割配置の情報の確保と、配列宣言の変更を行なう。

具体的な流れとして、実際に入力に用いた行列の積の逐次プログラム(図1)とその変換結果(図2)について示す。

1. DO文より最外ループの変数IとIの領域を得る。
2. Iを参照しているのは、配列Aと配列Cの1次元目の添字である。
3. 配列Aと配列Cを参照している変数はない。
4. 配列Aを参照しているのは変数Sである。
5. 変数Sは自分を参照しているため、ループ分割により結果が変わる可能性がある。
6. 変数Sの値の流れを追うと、Sはループ I内で初期化され、配列Cに代入されるためループIの分割には問題ない。
7. ループJ、ループKからは、ループIの分割に依存しない。
8. 上記から、ループIはイタレーションで分割できることがわかる。
9. 配列Aと配列Cの1次元目の添字がIのみであるため、ループを分割した数と配列Aと配列Cの1次元目の要素数は同じでよい。
10. 配列Aと配列Cの宣言文を変更する。

```
      ...
      INTEGER A(16,16),B(16,16)
      INTEGER C(16,16),S
      ...
      DO 100 I=1,16
         DO 200 J=1,16
            S=0
            DO 300 K=1,16
               S=S+(A(I,K)*B(K,J))
300         CONTINUE
            C(I,J)=S
200      CONTINUE
100   CONTINUE
      ...
```

図 1: 行列の積を求める逐次プログラム

```
      ...
      INTEGER A(4,16),B(16,16)
      INTEGER C(4,16),S
      ...
      DO 100 I=1,4
         DO 200 J=1,16
            S=0
            DO 300 K=1,16
               S=S+(A(I,K)*B(K,J))
300         CONTINUE
            C(I,J)=S
200      CONTINUE
100   CONTINUE
      ...
```

図 2: 行列の積を求める並列プログラム

4 評価

実際に行列の積のプログラムを図1を変換し、並列プログラム2に変換した。得られた並列プログラムから配列 A,B,C のうちA,C の2個が1ノード当たりの要素数が配列データのノード数で割った値の宣言に変換された。また、得られた並列プログラムの有効性の評価として、行列の大きさを256x256、512x512に変えてParagonXP/S-5 上の実行時間を計測した (行列の計算を行なう部分のみ)。その計測結果から台数効果は図3となる。

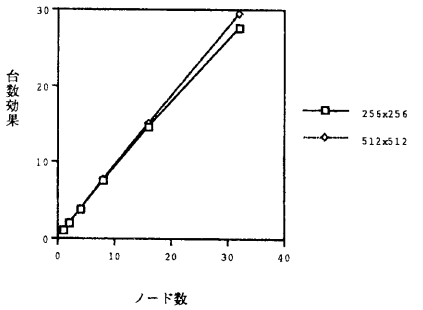

図 3: 行列積の台数効果

並列プログラムから、データ通信が必要なく最外ループも均等に分割できるので理論上台数効果は理想値となる。ここで、図3に台数効果落ち込んだ理由としては、計算時間に対しの時間計測のかかる時間が影響したと考えられる。

また、分割されなかった配列 B を分割した場合の並列プログラムを作成してみた。この場合、各ノードが他のノードが持つ配列 B のデータを参照するため、通信が必要

となる。ここで、データを受けとるための配列が必要となるが、その配列の大きさは行列データの１列を確保できるようにしている。この並列プログラムについても得られた並列プログラムと同様のデータ数でに実行時間を計測してみた。

トランスレータで得られたプログラムの実行時間と比較すると、配列Ｂを分割した場合、大幅に実行時間が長くなった。このことは前述でも述べたように配列Ｂの参照のためノード間で通信が起きるためである。このことから、並列化トランスレータにより、データ通信の少ない分割配置を決定した並列プログラムに変換できたことがわかった。

5　おわりに

配列データの分割領域の配置を決定させる自動並列化トランスレータを作成し、その変換手法と評価を述べた。現状では、いくつかのプログラムの入力をし、メインブロックという最も実行時間がかかると思われるブロックを設定し、そのブロックの最外ループ分割に適した配列要素の大きさを決定するように試みている。今後の課題として、より柔軟なデータ分割配置の決定、および、並列処理の効果を引き出すための逐次プログラムの前処理変換の追加、そして、さまざまな並列化変換手法の導入を考えている。

参考文献

[1] 本多弘樹:自動並列化コンパイラ, 情報処理,1993

[2] インテルジャパン:パラゴンプログラミングガイド,1995