勉強会資料
並列言語、並列化コンパイラ

高洪波
平成１６年５月１０日

１　はじめに

近年，並列計算機は数値解析、ビジネス、人工知能などの科学分野において用いられる．これは一つにはハードウェアの著し進歩による，処理速度の高速化に負う所が大きい．しかし，並列計算機を最も効率良く動作させるためには、その上で動くソフトウェアの最適化が欠かせない．従来から使用していた逐次処理言語やコンパイラをそのまま用いただけでは，並列計算機においてはむしろ低速になりさえするのである．

このような現状におけるソフトウェア最適化のアプローチとしては、三種類ある．

1、 従来からの逐次処理言語を捨て、並列計算機により適合するような言語を新しく作る方法．
2、 今ある逐次処理言語に並列処理機能を追加するという折衷案．
3、 逐次処理言語で書かれたプログラムを、コンパイラが自動的に並列化するという方法．

２　並列言語

科学技術計算機の応用分野ではユーザは下記のような言語が利用でき、並列化コンパイラがソースプログラムを目的プログラムに変換する．

２．１逐次言語

Fortran が代表で、プログラムを一つずつ逐次実行される．多数のデータに対する処理は DO ループによる反復で記述される．このようなプログラムを並列プログラムに変換するのが並列化コンパイラである．もともと逐次処理を念頭においてプログラムが作成されており、ユーザにとっては負担はなく、連続性は満たされるが、自動並列化は非常に困難である．

２．２逐次言語の並列処理拡張

逐次言語をベースにしつつ、並列処理構造を取り入れて、ユーザ負担を少し強いりながら、並列化コンパイラの負担を大幅に削減する方式である．並列処理構造の導入には以下のようなものがある．

1、 ループ並列化の記述：DOALL 文によってすべての反復の並列実行を明示的に示す．

```
DOALL I=1,N
A(I+1)=A(I)+1.0
END DOALL
```

では、A(I)のすべての要素が一斉に１だけ加算され、A(I+1)に転送される．
通常の DO ループとは処理結果が異なる．

2、 ベクトル演算/配列演算の記述：たとえば，A,B,C がベクトルの場合、C=A+B

の表現によって、ベクトルの全要素間での加算を表す.

3、 プロセス生成・終了、同期操作の指定：FORK-JOIN などによってプロセスの生成や終了を明示的に示す.

4、 通信文の明示的利用：Send/Receive などプロセス間での通信を明示的に示す

5、 データ分割の指定：ベクトルや配列に対してそのデータ分割法を明示的に示す.

逐次言語の拡張には現在二つの大きな流がある.

1、 HPF のアプローチ：Fortran プログラム内でデータ分割をユーザに指定させ、所有者計算規則(owner computer rule)を用いて目的並列プログラムを生成する. 目的並列プログラムには SEND/RECEIVE 文を含みメッセージ交換型マルチプロセッサを対象としている． Fortran-D,HPF(High Performance Fortran),Vienna Fortran などが代表例である．データ分割指示などはコンパイラへの指示となっている．したがってこらの指示文を無視すれば、単一プロセッサの目的プログラムも生成できる.

2、 PVM のアプローチ：個々のプロセス間でのメッセージ通信のプロトコルを定めておき、これらを利用したメッセージ交換で並列処理を行う．各プログラムは C など通常の逐次言語で記述される． PVM,MPI や Linda などが代表例である．ネットワーク上のワークステーションにも適用される．手軽な並列処理の利用形態である.

HPF の概要

HPF は 1992 年に設立された HPFF(High Performance Fortran Forum)とよぶ団体で標準化作業が進められている．逐次処理型言語である Fortran を並列処理拡張したものである．分散メモリシステム上での並列化において最も重要なデータ分割をユーザに任せ、処理を owner-computes rule[1]に基づいて各プロセッサに割り当てる方式を採っている.

科学技術計算機における Fortran のこれまでの資産を活かすことがてきるという点、そして並列計算機の主な用途の一つが科学技術計算であることを考えると、このアプローチは非常に魅力的である.

２．３並列処理言語

初めから並列処理向けに設計された言語.

- メリット
  従来からの逐次処理概念による束縛がないため，コンパイラの構成が比較的容易になり、それ故に並列化を最も容易に追加できる.
- デメリット
  既存の資産（コード、プログラミング技術）が活かせない.

データフロー言語のような関数型言語や、メッセージ交換モデルをベースとした

[1] Owners-computes rule: 分割されたデータを保持するプロセッサ自身がそのデータをアクセスする文を実行するよう，コードを生成する策のこと

Occam などの例がある．しかし、上述したようなデメリートのため、まだ広く使われるには至っていない．

３　自動並列化コンパイラ

並列化コンパイラは、以下のような手順でプログラムを並列化する．

1、 データ依存、及び制御依存の解析
2、 並列実行可能な部分の検出
3、 プログラムを並列処理できる形へ変換
4、 並列実行可能な部分をプロセッサへスケジューリング
5、 スケジューリング結果に基づいた並列コードの生成

以下、これらを順にみていく

3.1　依存解析

データ依存には以下のものがある．

フロー依存　ある文で定義した変数を他の文が使う．
出力依存　ある文が定義した変数を他の文が再定義する．
逆依存　ある文が使用した変数を他の文が定義する．

これらの依存性がない、あるいは解消できたタスクは並列実行可能である．
また、制御依存とは、タスク間において条件分岐がある場合のことをいう．

3.2　プログラム変換

前節で解析したプログラムを、並列実行可能になるように変換する．ここでは、各種の変換手法を述べる．

３．２．１　Doall,Doacross

並列化手法のうち、最も一般的なものがループに対するものである．これは、ループには潜在的に並列可能性が大きい、またプログラム実行時間のうちループにおける時間の割合が高いなどの理由による．ここで述べる Doall,Doacross は、Do ループ各反復を異なるプロセッサに割り当てて実行する方法．

Doall

全ての反復が全く独立に計算可能な場合、つまりそれぞれの反復間に依存性がない場合に、各反復を異なるプロセッサに割り当てて実行する方法．

Doacross

Doall と同様に、各反復を異なるプロセッサに割り当てるのだが，ある反復 S1 が別の反復 S ２の結果に依存する場合に S1 は S2 の終了を待ち、その結果を得てから計算にかかる方法．当然 Doall よりも低速である．

3.2.2　スケジューリング

前節までに行った並列実行可能な箇所を各プロセッサに割り当てる．割り当て方は、コンパイル時に行う静的スケジューリングと、実行時に行う動的スケジューリングがある．静的な方法は動的な方法と比較して実行時のオーバーヘッド少ないが、ループ中での条件分岐などのためにプロセッサ毎に処理時間が大きく異なる場合に全体としてのバランスが崩れる可能性がある．

他方、動的な方法だと実行時の通信によるオーバーヘッドの問題がある．動的スケジューリングの例として、

セルフスケジューリング

共有メモリ上に処理済の反復数を記憶するカウンタ変数を置く．各プロセッサは一つの反復を終えるとカウンタをインクリメントし、次の反復を実行する．

チャンクスケジューリング

カウンタ変数を置くまではセルフスケジューリングと同様だが、各プロセッサに複数の反復を一度に割り当てる点が異なる．こうすることにより、カウンタ変数を参照するオーバーヘッドを減少させる．

ガイデッドセルフスケジューリング

チャンクスケジューリングを応用したもので、ループ処理の進行とともに、チャンクスケジューリングにおける割り当てる反復の数を小さくする工夫をする．カウンタ変数を参照するオーバーヘッドを減少させつつ、全体のバランスも保つ．

3.3 データ分散配置

下記のプログラムを四台のプロセッサで実行する場合を考えてみよう．

```
      DO 10 I=2,100
      DO 20 J=1,100
      A(I,J)=A(I-1,J)+1.0
20    CONTINUE
10    CONTINUE
```

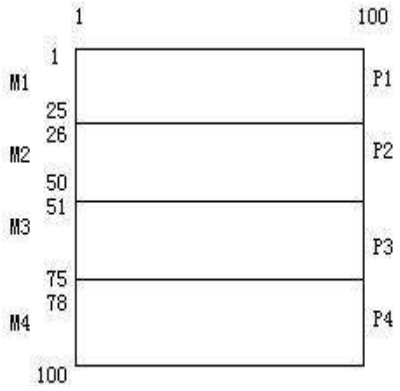

図１(a)

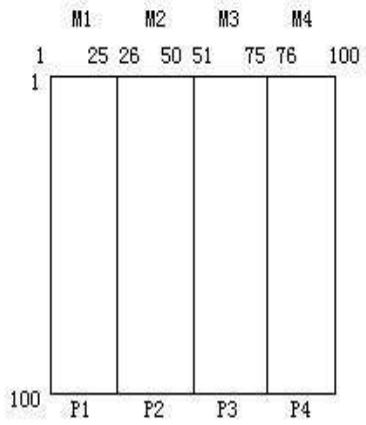

図１(b)

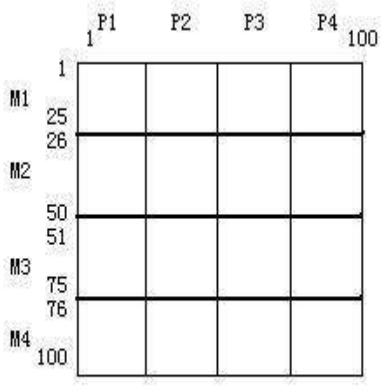

図１(ｃ)

図１ (a) のように配列 A を行方向で分割し、メモリ Mi に割付け、さらにその処理をプロセッサ Pi に割付けると、４台のプロセッサは並列処理は不可能となり、しかも境界で行方向データを３回通信する必要がある．同図（ｂ）のように、列方向で分割し、それらをメモリ Mi と Pi に割付けると、プロセッサ間に通信する必要がない．また、同図（ｃ）のように、行方向にデータ分割し、プロセッサ Pi のメモリ Mi に配置し、プロセッサ Pi は列方向のメモリ領域に対して演算を行う場合も考えられる．各プロセッサは並列動作可能である．しかし、たとえば、P2 の処理では M1,M3,M4 へのメモリアクセスの際に時間がかかる．このように、データ分割が性能に非常に大きく影響する．もちろん、データ配置を並列化コンパイラが自動的に行えるのが望ましい．しかし、非常に困難であるので、HPF ではユーザ指定となっている．

参考文献

［１］ 富田眞治，並列コンピュータ工学，昭晃堂，１９９６年

［２］ 並列言語，並列化コンパイラに関する勉強会資料，高山征大，２００２年