勉強会資料
自動並列化コンパイラ、HPF

縣亮慶

平成10年 7月 14日

1 はじめに

１つのプログラムの処理を並列処理によって高速化する場合、以下に示す並列化のための作業が必要となります。

- プログラムをタスクに分割
- タスク間の並列性の検出・指示
- 各タスクのプロセッサへの配置
- プロセッサ間のデータ通信コード・同期コードの挿入

並列計算機の性能を十分に引き出すには、これらの指示をユーザが陽に記述する必要がありました。しかし、これだと、ハードウェアの性能を十分に利用できる可能性があるというメリットがある反面、

- 性能を追求する余り、ハードウェアに依存したポータビリティリーの無いコードとなってしまう
- プログラムの開発・デバッグが複雑なため、ユーザが並列化の全てを行うことは不可能である

などのデメリットがあります。

そこで、これらの作業を自動的に行う自動並列化コンパイラが望まれます。自動並列化コンパイラを使用すると、

- 既に存在する膨大な量のプログラムを、書き換える必要がない
- プログラムの生産性およびポータビリティーを損なわずに新規のプログラムを開発できる

などの利点があります。

以下、マルチプロセッサシステム上で単一のプログラムを並列処理するために必要な自動並列化コンパイラの諸技術について解説します。

2 並列化コンパイラの基礎技術

一般に並列化コンパイラはプログラムを以下のような手順で並列化します。

1. 並列性 (並列実行可能性) 抽出
   (a) データ依存、制御依存の解析
   (b) プログラム変換、並列実行可能な部分の検出
   (c) タスク生成 (タスク単位を決定)
2. データの分割、配置
3. 並列実行可能部分のプロセッサへのスケジューリング
4. スケジューリング結果に基づいた並列化コードの生成

2.1 データ依存解析

自動並列化を行なう際、コンパイラはまず命令間、ステートメント間、サブルーチン間、Do ループ間のデータ依存解析を行います。

データ依存解析は並列性抽出のためのもっとも基本的なプログラム解析の１つであり、データの定義・使用関係により生じる実行順序関係の制約を解析することを意味します。

データ依存には、以下の３種のものがあります。

1) フロー依存 :ある文で定義した変数を後続の文で参照する。

2) 出力依存 :ある文が定義した変数を後続の文が再定義する。

3) 逆依存 :ある文が参照した変数を後続の文が定義する。

これらは、並列実行の可能性を検出するために使用されます。

2.2 コントロールフロー解析

コントロールフロー解析とは、プロシージャ内の制御構造、プロシージャ間の呼び出し関係を解析し、グラフで表現するもので、主に粗粒度タスクの並列性抽出・スケジューリングに使われます。

これを利用したものとして、IBM の PTRAN, 早稲田大学の OSCAR などのプロジェクトがあります。

2.3 プログラム変換、並列性抽出

ループ

従来、並列化の主な対象となってきたのはループであり、ループに対する並列化技術が最も進んでいます。というのも、ループには多くの並列性が期待できる、ループ処理時間がプログラム実行時間に占める割合が大きい、解析が行いやすい、という理由があるからです。

ループを並列化する場合、Do ループの各イタレーションを異なるプロセッサに割り当てて処理する方法が一般的です。この方法でのループの並列処理は、Doall 処理と Doacross 処理の 2 種類に分けることができます。

* イタレーション : ループの繰り返しの 1 回分のこと

- Doall 処理
  ループの各イタレーションについて、それぞれ独立に並列実行をする。下の例では、外側のループが Doall 処理可能。Doall 処理可能なループを、Doall ループという。

- Doacross 処理
  ループの各イタレーションについて、同期を取りながら並列実行をする。下の例では、J=k のときの S1 の計算に、J=k-1 のときの S2 の結果 (A(I,k-1)) が必要なので、並列実行は Doacross 処理になる。

例:

```
DO I=1,N   !Doall ループ
  DO J=2,N !Doall ループではない
    C(I,J)=A(I,J-1)+B(I,J) ! S1
    A(I,J)=C(I,J)/2.0      ! S2
  END DO
END DO
```

種々のプログラム変換

- ループインターチェンジ (loop interchange)

```
DO I=1, N   ! Doall ループ
  DO J=2, N
    A(I,J) = A(I,J) + A(I,J-1)
  END DO
END DO
```

このプログラムは外側のループが Doall ループなので、I の各値毎に独立に並列実行できますが、このループを内側に持ってくれば、次のようにベクトル化することができます。

```
DO J=2, N
  DO I=1, N   ! Doall ループ
    A(I,J) = A(I,J) + A(I,J-1)
  END DO
END DO
```

ベクトル化後

```
DO J=1, N
  A(2:N,J) = A(2:N,J) + A(2:N,J-1)
END DO
```

ベクトル化と並列化

おおざっぱに言って、上の例のように最も内側のループを Doall ループにすることをを試

みるのがベクトル化です。逆に、最も外側のループを Doall ループにすることを試みて、最大の大きさの並列領域を作ろうとするのが並列化です。

ベクトル化したものはSIMD型の計算機で計算しやすく、並列化したものは MIMD 型の計算機で計算しやすくなります。

- ループ崩壊 (loop collapsing)
  繰り返し回数の少ないループは、そのぶん一度に並列実行可能なイタレーションが少ないですが、下のように 2 次元配列を 1 次元に変換して、2 重ループを 1 重ループにすれば繰り返し回数を多くできます。

```
REAL A(6,6), B(6,6), C(I,J)
DO I=1, 6
  DO J=1, 6
    A(I,J)=B(I,J)+C(I,J)
  END DO
END DO
      ↓
REAL A(36), B(36), C(36)
DO IJ=1, 36
  A(IJ)=B(IJ)+C(IJ)
END DO
```

- ストリップマイニング (strip mining)
  ループ崩壊とは逆に、繰り返し回数の多いループを多重ループに変換する手法です。

```
DO I=1, N
  C(I)=A(I)+B(I)
END DO
      ↓
DO I=1, N, 64
  DO J=I,MIN(I+63,N)
    C(I)=A(I)+B(I)
  END DO
END DO
```

- ループ融合 (loop fusion)
  複数のループを 1 つにまとめるもので、ループ制御のオーバーヘッドを減少させるために使われます。

```
DO I=1,N
  A(I)=B(I)*C(I)
END DO
DO I=1,N
  D(I)=A(I)+C(I)
END DO
      ↓
DO I=1,N
  A(I)=B(I)*C(I)
  D(I)=A(I)+C(I)
END DO
```

- ループ分割 (loop fission) ループ融合とは逆に、1 つのループを複数に分割するものです。Doacross ループから、Doall ループを分離したりするのに使われます。

```
DO I=1,N  !Doacross
  D(I)=B(I)+B(I-1)
  A(I)=A(I-1)+B(I)
END DO
      ↓
DO I=1,N  !Doall
  D(I)=B(I)+B(I-1)
END DO
DO I=1,N  !Doacross
  A(I)=A(I-1)+B(I)
END DO
```

2.4 データ分割配置の最適化

データの分割配置（分配）はプロセッサ間の通信回数と負荷分散に大きな影響を与えます。データの最適分割配置を見つける問題は NP 完全であることが知られており、実際の手法はヒューリスティックなものとなります。

後述する HPF では、自動的なデータ分割は行わず、プログラマに指定させるようになっています。

2.5　スケジューリング

並列性抽出の後、プロセッサへのタスクの割り当てとその実行順序を決定する必要があります。

スケジューリングには、コンパイル時に行う静的スケジューリングとプログラム実行時に行う動的スケジューリングがあります。静的スケジューリングでは実行時のオーバーヘッドが小さく、またプロセッサ間データ転送を最小化できるが、ループ中の条件分岐などで各プロセッサの負荷バランスが崩れる危険性があります。

オーバーヘッドの少ない動的スケジューリングとしては、以下のものがあります。

セルフスケジューリング
: 共有メモリ上に処理を終えたイタレーション数を記憶するカウンタを用意し、各プロセッサが1イタレーションの処理を終えるとそのカウンタを参照し、次に自分が処理すべきイタレーションを1つ決め、カウンタをインクリメントする方式。

チャンクスケジューリング
: セルフスケジューリングのように1イタレーション毎ではなく、事前に設定された数の複数イタレーションを1度に取ってくる方式。

ガイデッドセルフスケジューリング
: 各プロセッサが次に処理すべきイタレーションを決める際、その時点で残っているイタレーション数Rをプロセッサ数mで割った数、すなわち[R/m]個([x]はx以上の最小の整数)のイタレーションを次に処理するイタレーションとして獲得する方式。

HPF

3　目的

HPF(High Performance Fortran)は、以下の目的をもって設計された並列記述言語です。

- Fortranとの互換性
  HPFの指示文はすべてFortran90のコメントの形(つまり、!で始まる)をしているので、一部の点を除いてFortran90と下位互換性がある。
- データ並列プログラミング(単一スレッド、グローバルな名前空間、疎同期並列計算)をサポート
- SIMD/MIMD、共有メモリ/分散メモリのいずれでも性能を発揮
- さまざまなアーキテクチャの計算機に適用可能な標準的な言語を定義
- 他言語および他のプログラミングパラダイム(例えば、MPIを使用したメッセージ通信)とのオープンなインターフェースと相互引用性

4　HPFの概要

分散メモリシステム上の並列化においてもっとも重要となるのは、データの分散メモリ上への配置と、処理の各プロセッサへの分配です。並列化の良し悪しは並列性の抽出のみならず、いかにデータアクセスローカリティを高め、プロセッサ間通信を減らすかに依存します。このデータと処理の最適な分配をコンパイラが求められればいいのですが、現在の技術ではまだまだです。

そこでHPFでは、このデータ分割をユーザが明示的に指示し、処理の分配はコンパイラがowner-computes ruleなどの方法で行っています。

※ owner-computes rule: 代入文の左辺の変数を所有するプロセッサが計算を実行するように処理を分配する方法のこと。

4.1　データの整列、分散

まず、次の例を考えます。
例1：

```
DO I=1, N
  A(I) = B(2*I) + C(I+1)
END DO
```

このループの並列化において、通信のオーバーヘッドを減らすためにはA(I)とB(2*I)とC(I+1)が同一のプロセッサのメモリに割り付けられた方がよい。このために、異なる配列のどの要素とどの

要素を同一プロセッサのメモリに配置するかを記述するのが整列 (Alignment) です。

一方、

例 2 ：

```
DO I=2, N
 S = S + D(I) * D(I-1)
END DO
```

この例では、配列 D の連続する要素がなるべく同一プロセッサのメモリに配置された方が良い。このために、同一配列の各要素をどのように分散メモリにレイアウトするかを指定するのが分散 (Distribution) です。

HPF では、このように整列と分散の 2 段階の枠組みを用いて、配列データをユーザ定義の論理プロセッサ配列 （PROCESSORS 指示行で任意次元の矩形配列状のものを指定できる ）にマッピング (Mapping) します。

```
配列データ -----> 整列データ --->
          整列指示          分散指示

 --> プロセッサ配列 ----> 物理プロセッサ
                 処理系依存
```

5 HPF 言語

5.1 ALIGN 指示行

たとえば、例 1 の場合、次のように整列を指示することができます。

```
!HPF$ ALIGN A(I) WITH B(2*I)
!HPF$ ALIGN C(I) WITH B(2*I-2)
```

ここで A,C を被整列変数、B を整列対象変数、I をダミー変数と言います。

次元の異なる配列間の整列を指定するには、*を使います。

例 :`!HPF$ ALIGN A(I,*) WITH B(I)`

5.2 DISTRIBUTE 指示行

DISTRIBUTE 指示行は、配列またはテンプレートの各次元に対して、BLOCK,CYCLIC,*を指定することにより、配列データの分散メモリ上のレイアウトを指定します。

- BLOCK ブロック分割を行う。例えば A(1:30) をプロセッサ P1,P2,P3 に割り当てるとき、P1 に A(1:10), P2 に A(11:20) P3 に A(21:30) が割り当てられる。
- CYCLIC サイクリック分割を行う。例えば A(1:30) をプロセッサ P1,P2,P3 に割り当てるとき、P1 に A(1),A(4), A(7)... ,P2 に A(2),A(5),A(8)... というように割り当てられる。
- * これを指定した次元は、分割されない。

例 2 の場合、次のようにプロセッサ配列と分散を指示することができます。

```
!HPF$ PROCESSORS PROC(10)
!HPF$ DISTRIBUTE D(BLOCK) ONTO PROC
```

5.3 ループの並列性の指示

ループの並列実行を促進するため枠組みとして、FORALL 構文と INDEPENDENT 指示行があります。これらは、通常の DO ループよりデータ依存関係を弱める働きを持ちます。

参考文献

[1] 笠原博徳並列処理技術コロナ社

[2] 本多弘樹, 自動並列化コンパイラ 情報処理,Vol.34,No9,pp. 1150-1157,Sep 1993

[3] 妹尾義樹, HPF 言語の現状と将来 情報処理,Vol.38,No2,,Feb 1997