Cプログラムのループ並列化に関する一手法

5G－2

朝井 義久　塩田 佳明　渋沢 進

茨城大学工学部情報工学科

1　はじめに

Fortranプログラムでのループでは制御変数が1回の繰り返しで1増えるだけというような定数値の増減だけであったので，これを繰り返し回数として利用して並列化を行なうことができた[1]．しかし，Cプログラムでは制御変数が存在しないものや，存在しても複雑な変化をするようなものがあるために並列化が困難となっている．

そこで，制御変数の概念をより拡張した拡張制御変数を提案し，それを利用することにより並列化することを考えた．また，ループの並列化を可能にするためのループの条件式の条件や，逐次処理した際の変数の値と同一なものにするための後処理についても考察した．

2　ループについて

2.1　ループの並列化

ループの並列化はループ本体内の一連した文を表すイタレーションをいくつかのプロセッサに割り当てて，それらを同時に実行することによって行なわれる．そのため，イタレーション内での依存関係はループの並列化では無視することができる．

問題となるのは，異なるプロセッサに割り当てられるイタレーション間の依存関係である．そして，すべてのイタレーションでイタレーション間依存が存在しない時に並列化が可能となる[2]．

2.2　イタレーションの割り当て

イタレーションの割り当て方法はいろいろあるが[1][3]，任意の連続した2つのイタレーションの実行開始の時間差ができるだけ少なく，かつ平均的である以下の割り当て方法を本研究では利用する．

$$\{\text{プロセッサ}\,i\,\text{に割り当てられるイタレーション}\} = \{x \mid x \bmod n = i\}$$

ここでxはイタレーション番号，nはプロセッサ数．

例:プロセッサ数が4のとき（プロセッサ番号はP0～P3）

プロセッサ		イタレーション番号
{P0}	=	{ 0, 4, 8, 12, ⋯ }
{P1}	=	{ 1, 5, 9, 13, ⋯ }
{P2}	=	{ 2, 6, 10, 14, ⋯ }
{P3}	=	{ 3, 7, 11, 15, ⋯ }

A Method of Loop Parallelization for C Programs
Yoshihisa ASAI
Yoshiaki SHIOTA
Susumu SHIBUSAWA
Faculty of Engineering, Ibaraki University
Hitachi, Ibaraki 316, Japan

2.3　3つのループ文について

C言語にはfor, while, do～whileの3つのループ文が存在するが，これらは以下のように変換することによりwhileループに変換することができる．

do ～ while ループ

```
do                          文
    文          =>      while( 条件 )
while( 条件 )               文
```

for ループ

```
for( 初期化 ; 条件 ; 再初期化 )
            文
                        初期化
                        while( 条件 ) {
                =>          文
                            再初期化
                        }
```

3　拡張制御変数

Cでは，Fortranのように明確な制御変数は存在しないが，イタレーションごとに規則的に変化する変数は存在することが多い．そして，それらは直前のイタレーションに依存するため，連続したイタレーションを別々のプロセッサに割り当てても，正しく並列化が行なわれない．しかし，規則的に変化する変数に対しては，現在の変数の値と繰り返したイタレーション回数から，任意のイタレーションの値を求めることができる．このような性質を持つものを拡張制御変数と呼ぶことにする．

この節では拡張制御変数をさらに3つに分類したものを提案する．以下ではこれらについて代入文の形式を定義し，イタレーションをn回繰り返した後の値を求める計算式を示す．

3.1　自己依存変数

ある変数iへの代入文の右辺がその変数のみの関数であるもの．

- プログラムでの代入文の形式

  $$i = f(i)$$

  （$f(i)$はi以外の変数を含まない関数）

- イタレーションをn回繰り返した後の値

  直接計算式から求めることはできないが，この変数は他の変数に依存しないのでループから代入文だけを抜き出してn回実行すればよい．

3.2 規則変化変数

ある変数iへの代入文の右辺がその変数の1次式であるもの.

- プログラムでの代入文の形式

$$i = ai + b \quad (a, b \text{は定数})$$

- イタレーションをn回繰り返した後の値i_n

$$i_n = a^n i + \sum_{k=0}^{n-1} a^k b$$

3.3 定変化変数

ある変数への代入文の右辺がその変数を加減したもので4節での単調変化する式の要素となることができるものである.

- プログラムでの代入文の形式

$$i = i + b \quad (b \text{は定数})$$

- イタレーションをn回繰り返した後の値i_n

$$i_n = i + nb$$

4 ループの条件式

逐次のループでは一度条件を満たさなくなった時点で, たとえ次のイタレーションで条件を満たしていたとしても繰り返しを終了する. ループを並列化すると, プロセッサにより実行しないイタレーションが存在するために, 並列化を可能にするためには一度条件を満たさなくなったらそれ以降も条件を満たさないような条件式であれば良い. つまり, ループの条件式が以下の2つの条件を満たせばよい.

1. 関係演算子が$\leq, <, >, \geq$のいずれかであること.
2. 条件式の右辺と左辺が両方とも,

$$c_1 i_1 + c_2 i_2 + \cdots + c_n i_n + c_{n+1}$$

の式で表される単調変化する式であること. ただし, $i_1, i_2, \cdots, i_n$を定変化変数とし, $c_1, c_2, \cdots, c_{n+1}$を定数とする.

5 後処理

この処理はループの実行後の変数の値を逐次処理後と同じにするものである. これは, プロセッサ間で共有している変数には必要ないが, 拡張制御変数のようにプロセッサによって異なる値を持っているものには必要である.

e_1, e_2を単調変化式としプロセッサ数をnとしたとき, ループ繰り返し条件を$\boxed{e_1 \text{関係演算子} e_2}$とすると,

$$\begin{cases} \max(h_1, h_2, \cdots, h_n) & (\text{関係演算子が} >, \geq \text{のとき}) \\ \min(h_1, h_2, \cdots, h_n) & (\text{関係演算子が} <, \leq \text{のとき}) \end{cases}$$

を満たすh_kをもつプロセッサkが逐次処理後の変数と同じ値を持っている. ただし, h_kをプロセッサkの$e_1 - e_2$の値とする.

6 変換様式

以下に示すのは, idをプロセッサ番号, nをプロセッサ数としたときの2.2節のイタレーション分割法でのそれぞれのプロセッサが実行するプログラムである.

このイタレーション分割法では, まずi番目のイタレーションを実行し, それ以降はnイタレーションずつ進めて実行していくものである. しかしプログラム内の"文"を実行することにより自動的に1イタレーション進むので, 残りの$(n-1)$イタレーションを最後に進めればよい.

```
拡張制御変数を id イタレーション進める
while( 条件式 ) {
  文
  拡張制御変数を (n-1) イタレーション進める
}
```

すべてのプロセッサでの処理が終った後に後処理を行ない, 正しい拡張制御変数の値にする.

変換例:

```
                          i=i+id*1;
while( i<N ) {            while( i<N ) {
   a[i]=b[i]*c[i]; =>        a[i]=b[i]*c[i];
   i++;                      i++;
}                            i=i+(n-1)*1;
                          }
```

7 おわりに

本研究で新たに拡張制御変数という概念を導入することにより, 明確な制御変数の存在しないCプログラムのループについても拡張した制御変数として見つけ出すことができるようになったため, より効率的な並列化が行なえるようになった.

今後の課題として以下のようなものが挙げられる.

- ループより大きな関数単位での並列化手法についての考察
- 本研究での手法を用いて, 逐次プログラムから並列プログラムへ自動的に変換するトランスレータの開発

参考文献

[1] 本多弘樹: マルチプロセッサのためのコンパイラ技術, 情報処理, Vol.31, No.6, pp.744-752(Jun.1990).

[2] 菊池純男: 並列化支援システム, 情報処理, Vol.34, No.9, pp.1158-1169(Sep.1993).

[3] 本多弘樹: 自動並列化コンパイラ, 情報処理, Vol.34, No.9, pp.1150-1157(Sep.1993).