3L－5

制御依存制約を排除したプログラム表現形式と細粒度並列計算機のためのオブジェクトコード最適化

小野田 亘利　高木 浩光　李 鼎超　石井 直宏
名古屋工業大学

1　はじめに

命令レベル並列処理の効率を高めるためには実行対象プログラムからいかに高い並列性を抽出するかが鍵となっており，大域コードスケジューリングが重要視されている．従来の大域コードスケジューリング技法としてトレーススケジューリング [1] やバーコレーションスケジューリング [2] などが知られているが，トレーススケジューリングでは正確な分岐予測が困難である場合に性能低下を招くことから，バーコレーションスケジューリングのようなコード移動を基本とした最適化手法が注目されている．

本稿では，「まずひとつのスケジュールを決定 (ソースプログラムの構造そのままにコンパイル) した後に基本ブロックを越えてコード移動させる」といった従来の小手先の最適化手法とは異なる，「一旦すべての配置可能性を中間コード上で表現した上でそこから最適解を求める」というスケジューリング手法について，基礎的考察を行なう．

2　制御依存制約を排除したプログラム表現形式

2.1　条件処理構造を越えた単一代入化

オブジェクトコードの大域的な最適化を系統的に行なうために，制御依存制約を排除したプログラム表現形式 FSD を用いる．FSD では制御依存制約 (条件値が計算されるまで，その条件に依存した計算のすべてが実行されないという先行制約) を排除するために，まず第一に，条件処理構造を越えた単一代入化を行なっている．図 1 は高級言語プログラムから FSD への変換の一例である．

```
a:=in();          (1  in   .   .)
b:=in();          (2  in   .   .)
c:=a*2;           (3   +   1  _2)
f:=a+b;           (4   +   1   2)
if (b) 0) {       (5   )   2  _0)
   e:=c+8;        (6   +   3  _8)
   c:=a+e;        (7   *   1   6)
} else {
   c:=b*2;        (8   *   2  _2)
   a:=c;          (9   :=  8   .)
   e:=a*c;        (10  *   9   8)
}
c:=e+5;           (11  +  (5   6  10)  _5)
if (c) b) {       (12  )  11   2)
   g:=a+c;        (13  +  (5   1   9)  11)
   c:=b+g;        (14  +   2  13)
} else {
   d:=b*2;        (15  *   2  _2)
   if (e) 0) {    (16  )  (5   6  10)  _0)
      g:=a-c;     (17  -  (5   1   9)  11)
      h:=f+g;     (18  +   4  17)
      b:=d*h;     (19  +  15  18)
   } else {
      b:=b*2;     (20  *   2  _2)
      g:=a*b;     (21  +  (5   1   9)  20)
   }
}
a:=g+b;           (22  +  (12  13  (15  17  21))
                          (12   2  (15  19  20)))
b:=c;             (23  :=  (12  14  11))
```

図 1　高級言語から FSD への変換

FSD は三番地表記 (U_i) を基本としており，複雑な式等は三番地表記の組に分解される．各 U_i (命令) は，

$$
\begin{aligned}
U_i &= \text{“(”}\ V_i\ \ op\ \ R\ \ R'\ \text{“)”}\\
op &= \text{“:=”} \mid \text{“+”} \mid \text{“−”} \mid \text{“*”} \mid \text{“/”} \mid\\
&\quad \text{“==”} \mid \text{“!=”} \mid \text{“<”} \mid \text{“<=”} \mid\\
&\quad \text{“>”} \mid \text{“>=”} \mid \text{“in”} \mid \text{“out”}\\
R &= V_j \mid S \mid .
\end{aligned}
$$

で与えられ，V は (計算される) 値を指すラベルである．S は，条件選択オペランド

$$S = \text{“(“}V_C\ \ R''\ \ R'''\ \text{“)”}\quad (V_C \text{は論理値を指す})$$

であり，V_C の論理値が真であるならば S=R''，偽ならば S=R''' を指すものとする．図 1 の例では，$U_{11}, U_{13}, U_{16}, U_{17}, U_{21}, U_{22}, U_{23}$ で条件選択オペランドが用いられており，これらにより同図左の if 文について条件処理構造を越えた単一代入化がなされている．

2.2　FSD のグラフ表現

FSD のグラフ表現を図 2 に示す．グラフ表現においては，U_i をノードとし，U_i が命令 U_j に先行する (U_j が U_i に依存している) 関係を方向付きエッジで示す (同図 (a))．条件選択オペランドは「選択子」により示す (同図 (b))．

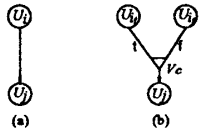

図 2　FSD のグラフ表現

U_j のオペランドとして，U_{i_t} または U_{i_f} が条件 V_C の論理値の真偽により選択されるとき，U_{i_t} の選択条件を V_C，U_{i_f} の選択条件を $\overline{V_C}$ と表記する．一般に，複数の選択子 $V_{C_1}, V_{C_2}, \ldots$ の連鎖により選択されるオペランド Sの選択条件は，各選択子での選択条件の論理積で示される．図 3 に 図 1 のプログラム例のグラフ表現を示す．

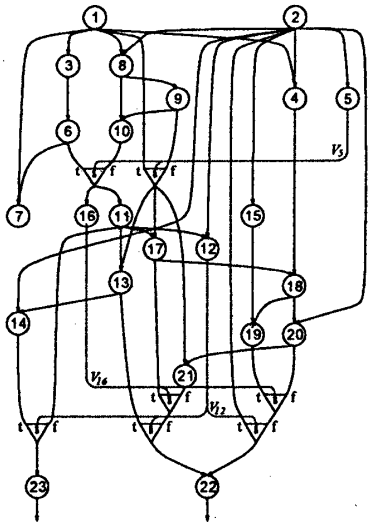

図 3　図 1 のグラフ表現

2.3　計算必要性条件

計算必要性条件は以下のように各命令ごとに定義される．

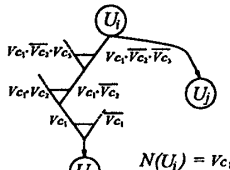

図 4　計算必要性条件

Intermediate program representation without control dependencies and its uses in optimazation
Nobutoshi Onoda, Hiromitsu Takagi, Dingchao Li, Naohiro Ishii
Nagoya Institute of Technology
Gokiso-cho, syowa-ku, Nagoya, 466, Japan

U_i が U_j に先行する関係を $U_i \prec U_j$とし，図3のように $V_{C_1} \cdot \overline{V_{C_2}} \cdot \overline{V_{C_3}}$ の選択条件となるような選択子を介した先行関係を $U_i \prec_{V_{C_1} \cdot \overline{V_{C_2}} \cdot \overline{V_{C_3}}} U_j$ と表記するものとする．ここで，Uの計算必要性条件を $N(U)$ とすると，$N(U_i)$ は，

$$\begin{aligned} N(U_i) &= C \cdot N(U_k) \vee N(U_j) \\ C &= V_{C_1} \cdot \overline{V_{C_2}} \cdot \overline{V_{C_3}} \end{aligned}$$

であり，一般に $N(U)$ は，

$$N(U) \triangleq \bigvee_{U \prec_C U_i} C \cdot N(U_i)$$

で与えられる．論理式 C を構成する論理値の少なくとも1つがまだ計算されていない時点では $N(U)$ の値は未定であるという．

計算必要性条件 $N(U_i)$ は，$N(U_i)$ へのデータ依存関係に基づいて V_i の値が使用される必要十分条件を表しており，$N(U_i)$ が真である場合 (偽でも未定でもない場合) には「U_i の計算が必ず実行されなければならない」ということを意味する (U_i が実行されるのは $N(U_i)$ が真のとき という意味ではない)．

FSDでは，プログラムの条件処理に関する構造を，条件選択オペランドに計算必要性条件を組み合わせることによって，制御依存制約なしに表現している．例えば図1のプログラムの場合，U_{11} のオペランドで使用される値は V_5 の値によって，U_6 または U_{10} のいずれかに決定されることを意味しており，U_6 の実行は V_5 が真のときにしか必要でないことを意味している．もし実行命令数を最小化する戦略でこれをオブジェクトコード生成するならば，V_5 が真のときに U_6 が実行されるようにスケジュールすればよいことになる．また実行命令数を最小化する必要がない場合には，V_5 の真偽に関わらず U_6 が実行されるようスケジュールしてもよいという可能性も持っている．

3 FSDを用いたオブジェクトコード最適化

3.1 本稿で対象とする実行アーキテクチャ

本稿ではごく一般的な細粒度並列プロセッサを対象アーキテクチャとする．すなわち，条件選択オペランドを直接処理できるような命令セットは持たないとし，条件値による命令の実行/不実行の制御はごく一般的な分岐命令により実現されるものとする．また，レジスタ数は無限とし機能ユニット (FU) は2台とする．このようなアーキテクチャを $M1$ とする．

3.2 オブジェクトコード生成と最適化

3.2.1 分岐命令によるブロック単位での条件実行

$M1$ においては分岐命令を用いることにより，ブロック単位で命令の実行/不実行を制御することができる．このブロック B を条件実行ブロックと呼ぶ．

FSDからオブジェクトコードへの変換には，機能ユニット割り当てやレジスタ割り当ての他に，条件実行ブロックへの命令の割り当てスケジューリングがなされる．このスケジュールに要求される条件は以下のものである．

U_iの条件実行ブロックへの配置条件 1
$N(U_i)$ が Tなら，U_i が必ず一回実行される

上記の配置条件を満たす全ての 命令配置の組み合わせにおいて，最も平均スケジュール長が短くなるようなものが，最適スケジュールである．

3.2.2 条件選択オペランドの実現

条件選択オペランドで表現された FSD から，レジスタ固定オペランドアーキテクチャ用のオブジェクトコードへの変換を，本稿では共通変数 (レジスタ) への書き込みにより実現する．このことにより，3.2.1の配置条件1に加え，次の制約が加わる．

U_i の条件実行ブロックへの配置条件 2
同一の条件ブロック B 内において各レジスタへの代入は1回のみ

4 コード生成結果と考察

4.1 最適スケジューリング結果

前節で示した条件実行ブロックへの割り当て条件を満たす範囲で，図1のプログラムを $M1$ を対象としてオブジェクトコードを生成し，スケジューリングを行なった結果を図5に示す．同図 (a) は，元のプログラムにおける各命令の出現位置 (if 文の) に直接対応する条件実行ブロックに配置した場合であり，(b) は最適に配置した場合である．

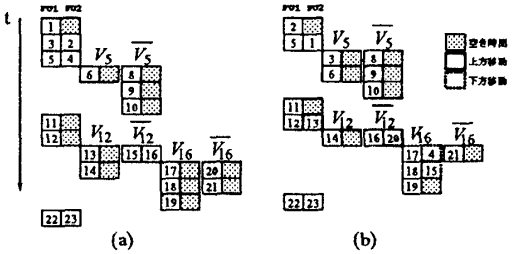

図5 図1のプログラムのスケジューリング結果

4.2 考察

図5のスケジュールでは (b) は (a) に対して平均スケジュール長が短縮されている．これは，

(1) 条件値によっては必ずしも実行の必要のない命令 U_4, U_{15} が，プログラマによって，条件値によらず実行される位置に書かれていた (いわば投機的な実行がプログラマによって指定されていた) ものが，(b) では計算必要性条件が真となる場合にだけ実行される位置に移っている．

(2) (a) では計算必要性条件が真となる場合にだけ実行される位置にある命令 U_{13},U_{20} が 必ずしも実行が必要でない条件でも実行される位置に移動している (いわゆる投機的コード移動がなされている)．

ことにより実現されている．

特にこの場合に重要なのは，U_{15} の下方移動によって機能ユニットに空き時間が生じたことによってクリティカルな命令 U_{20} をそこに投機的移動させることができたことである．

5 まとめと今後の課題

4.2 の考察で示したようなコード移動は，個々の移動そのものに関して言えば，パーコレーションスケジューリング [2] や古関らの手法 [4] をはじめとする基本ブロックを越えたコード移動スケジューリング方式でも行なわれるものと考えられるが，これらの手法では，局所的な観測からコード移動を繰り返し適用する方式を採っているため，極小解に陥りそこで止まってしまう可能性がある．

これに対し本稿で示した手法では，一旦すべての配置可能性を FSD モデルによって表現した上でそこから最適解を求めるという方法をとっているため，機能ユニット数有限の対象アーキテクチャの場合においても，十分な計算時間があれば必ず真の最適解を求めることができる．

今後の課題として，実用的な時間内に十分な近似解を得られるようなアルゴリズムを発見すること，また，これらのアルゴリズムがどの程度真の最適解に迫っているかを評価するために，GA, SA などを利用したより精度の高い近似解探索アルゴリズムの開発や，分枝限定法による全解探索の実験を行なうことなどが挙げられる．

参考文献

[1] J.R.Ellis, 'Bulldog: A Compiler for VLIW Architectures', The MIT Press, (1985).

[2] A.Aiken,and A.Nicolau, 'A Development Environment for Horizontal Microcode', *IEEE Transactions on Software Engineering*, vol.14,No.5,pp.584-594(1988).

[3] C.D.Polychronopoulos, 'Parallel Programing and Compilers', Kluwer Academic Publishers, (1988).

[4] 古関 聰, 小松 秀昭, 深澤 良彰, '命令レベル並列アーキテクチャのための大域的コードスケジューリング技法とその評価', 並列処理シンポジウム JSPP'94 論文集, pp.1-8(1994).