計算機アーキテクチャ 119-31
(1996. 8. 29)

式の分割による並列化アルゴリズム ESH

岩根雅彦[†] 濱田智雄[†,☆]
小島和広[†] 松田孝史[†]

式の変数の定義される時間，変数の使用可能時間と演算優先順位に注目した変数レベルの並列化のためのアルゴリズム ESH を提案する．ESH では同一優先順位をもつ変数間の演算を一つにまとめて未決定ノードとし原始プログラムから決定ノードと未決定ノードの混在した不完全タスクグラフを生成する．このタスクグラフからノードを取り出したとき未決定ノードであれば変数の使用可能時間の早い順に変数間の2項演算をプロセッサに割り当てて決定ノードを生成する．決定ノードであれば DSH によってスケジューリングする．すべてのノードが決定ノードになったときスケジューリングは完了する．ESH では複数の文間に内在する並列性が抽出できるだけでなく，最適な計算木が生成される．5つの簡単なプログラムを用いて ESH を細粒度マルチプロセッサ MSBM 上で評価した．その結果，すべてのプログラムで速度向上比は ESH≧THR と DSH の併用≧THR と GSH の併用であった．ESH の THR と DSH の併用に対する速度向上比の実測値の平均は 1.19 であった．一方 DSH の GSH に対する速度向上比の実測値の平均は 1.004 であった．このことから ESH の有効性が確認できた．

Expression Scheduling Heuristic Algorithm based on partitioning an expression

MASAHIKO IWANE,[†] TOMOO HAMADA,[†,☆] KAZUHIRO OSHIMA[†]
and TAKASHI MATSUDA [†]

The ESH algorithm schedules dyadic operations which derived from an incomplete task graph that is in consideration of the operation precedence and definition-time, acquisition-time of the variables and create a complete task graph. In ESH algorithm, an incomplete task graph is generated from a basic block which extracts from a primitive program and composed decidednodes and undecidednodes. A decidednode is an expression which includes a dyadic operation. An undecidednode is an expression which includes the same precedence operations in the expression. In this algorithm, a node is derived from an incomplete task graph and if it is an undecidednode then this algorithm generates decidednodes in consideration of acquisition-time of the variables and schedules it with DSH algorithm else if it is a decidednode then it is scheduled with DSH algorithm. When all nodes transformed into decidednode, this algorithm is completed. The ESH algorithm extracts not only parallelism that extends to expressions but also generates optimum calculation tree. We evaluated the ESH algorithm with five examination programs. There programs tested on the MSBM(Multiple Static Barrier Management MIMD). The results of the test, the speedup ratio of all of the examinations is ESH≧THR+DSH≧THR+GSH. The average of speedup ratio is 1.19 on ESH in contrast to THR+DSH. But, the average of speedup ratio is 1.004 on THR+DSH in contrast to THR+GSH. Because of these things, we confirmed effectiveness of the ESH algorithm.

1. はじめに

並列計算機をもちいてプログラムを並列化し処理速度の向上をはかる場合にはプログラムの意味的同一性を保持した上で並列実行可能な部分を最大限に抽出し並列実行する必要がある．とくにプロセッサ数にほぼ比例した速度向上率が得られる doall ループやベクトル演算などをプログラムから抽出することは，プログラム全体の実行速度向上に大きく寄与する．しかし一般的なプログラムには逐次実行の代入文や分岐文も数多く存在する．このような部分も並列化することによって一層の速度向上が期待できる．

逐次実行の代入文で構成された基本ブロックを並列化するための従来の静的スケジューリング手順は，まずプログラムをプロセッサ割り当ての最小単

† 九州工業大学工学部電気工学科
Department of Electrical Engineering, Faculty of Engineering, Kyushu Institute of Technology
☆ 現在，松下電器産業株式会社 半導体開発部門
Presently with Matsusita Electric Industrial Co.,Ltd.

位に分割し，つぎに依存解析ののちにこの最小単位をノードとするタスクグラフを作成する．そしてスケジューリングアルゴリズムによってノードをプロセッサに割り当てる．このアルゴリズムとしてGeneral List Scheduling Heuristic(GSH)[1)2)]，Duplication Scheduling Heuristic(DSH)[2)]，Trace Scheduling(TRS)[3)4)]などがある．これらのアルゴリズムではプロセッサの割り当て単位であるノードの大きさは本質的に自由である．VLIW計算機やスーパースカラ計算機ではこのアルゴリズムを機械命令レベルに適用[1)4)]し，OSCARなどでは文レベルに適用[5)6)]している．また代入文を2項演算に分解して変数レベルで並列化するアルゴリズムにTree Height Reduction(THR)[7)]がある．これは式の計算木を完全2分木に近づけて並列実行可能性を最大にすることによって並列化する．この並列化は代入文ごとに独立に行われる．しかし基本ブロックには通常複数の代入文が存在するため，これらの並列実行性を考えるとTHRによる変数レベルの並列化は最適とは言えない．

そこで，代入文の式の変数の定義される時間と演算優先順位に注目したExpression Scheduling Heuristic(ESH)を提案する．ESHでは同一優先順位をもつ変数間の演算すなわち部分式を1つのノードとして式を分割する．そして依存解析をおこなってタスクグラフを作成する．タスクグラフからノードを取り出して，変数の定義時間の早い順に部分式を2項演算に分解してプロセッサに割り当てる．

本論文では，ESHに必要な基本概念について述べ，次にESHスケジューリングアルゴリズムの概要について述べる．そしてESHにより並列化した例題プログラムをMSBMマルチプロセッサ[8)]で実行する．その結果を考察してESHの評価をおこなう．

2. 基本概念

2.1 前　提

一般に手続き型言語で記述されたプログラムは代入文，制御文などで構成される．このような文系列で，その出口を除いて分岐文がなく，かつその入口を除いて外から分岐されることのない部分を基本ブロックとよぶ．基本ブロックには手続き呼び出しは含まれず代入文のみで構成される．基本ブロック内代入文系列を変数レベルでの2項演算を基本とした並列化を考える．並列化によって生成される変数を一時変数，一方代入文中に明示的に与えられた変数を恒久変数，一時変数および恒久変数をさすときは単に変数とよぶ．べき乗は並列化の観点から乗算に置き換え，代入文の式の演算の優先度を規定する括弧は必要な個所にのみに許す．並列化における括弧の付け替えは演算精度に関係するので，代入文で記述されたものをそのまま使用する．

2.2 タスクグラフ

基本ブロックを並列化するにあたって基本ブロック内の変数の定義と参照を表す有向非循環グラフであるタスクグラフを導入する．タスクグラフのノードは単項演算子を伴った変数の代入文，2変数演算の代入文および演算優先順位が等しい演算子からなる多変数演算の代入文であり，エッジはノード間の依存関係を表す．依存関係には真依存，逆依存，出力依存があり，タスクグラフではこれらの依存関係を基本的にエッジにr/w，w/r，w/wを付加して表すが，真依存関係が満足されれば他の依存関係も同時に満足される場合が多いこと，逆依存，出力依存関係は変数のリネーミング等により回避できることから真依存関係を表すエッジには何も付加せずに，とくに逆依存，出力依存関係を注意する必要があるときのみw/r，w/wを付加したエッジを追加する．

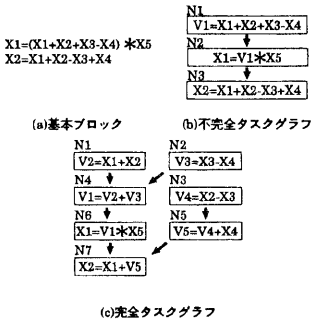

図1　タスクグラフ
Fig. 1　Example of task graphs

ノードが3変数以上の多変数演算の代入文であるとき未決定ノードとよび，2変数以下の演算の代入文のとき決定ノードとよぶ．また決定ノードのみで構成されるグラフを完全タスクグラフ，未決定ノードを含むグラフを不完全タスクグラフとよぶ．このようなタスクグラフにおいて，ノードN_iからノードN_jに向かうエッジがあるときノードN_iをノードN_jの親ノード，ノードN_jをノードN_iの子ノードとよぶ．タスクグラフ中で親ノードをもたないノードを入口ノード，子ノードをもたないノードを出口ノードとよぶ．ノードN_iの代入文を実行する時間をノード実行時間$T_e(N_i)$で表し，未決定ノードのノード実行時間はその演算を逐次実行したときの実行時間をとる．ノードN_iから出力ノードの間の経路で，経路に存在するノードの実行時間の和の最大値をノードN_iの優先度とよぶ．図1(a)は基本ブロック，(b)は基本ブロックに対応する依存解析後の不完全タスクグラフ，(c)は不完全タスクグラフをESHによってスケジューリングした結果の完全タスクグラフを示す．図1(b)のノードN1，N3は未決定ノード，(b)のノードN2と(c)のすべてのノードは決定ノードである．また変数X_iは恒久変数，V_iは一時変数である．

2.3 2項演算とタプル表現

演算優先度が等しい演算子で構成される未決定ノードの代入文では右辺の式の評価順序に拘束はない．すなわち，結合則，可換則をどのように適用してもかまわない．基本ブロックの実行開始時間を0としたときに，式にあらわれる変数 X_i がプロセッサ PU_j で定義される時間を変数 X_i の定義時間 $T_d(X_i, PU_j)$ とよぶ．変数 X_i がプロセッサ PU_j で定義されてプロセッサ PU_k で参照可能となる時間を変数 X_i の取得時間 $T_g(X_i, PU_k)$ とよび次式で与える．

$$T_g(X_i, PU_k) = T_d(X_i, PU_j) \quad (j=k) \quad (1)$$

$$T_g(X_i, PU_k) = T_d(X_i, PU_j) + T_c(PU_j, PU_k) \quad (j \neq k) \quad (2)$$

ここで，$T_c(PU_j, PU_k)$ はプロセッサ PU_j とプロセッサ PU_k との通信時間である．すると取得時間の小さい変数に対する演算から順次評価していけば代入文の実行が最も早く完了する可能性が高い．また基本ブロックの代入文を図1で示したように決定ノードと未決定ノードに分解していくので負符号または否定演算子を伴った変数は決定ノードとして扱われる．それゆえ未決定ノードには負符号，否定演算子を伴った変数は含まない．

式3に示した未決定ノードの代入文において変数 X_0 の前に形式的な空なる演算子 $OP_0(\equiv \phi)$ をおけば演算子 OP_i と変数 X_i を組として取り扱える．

$$X_n := X_0 OP_1 X_1 \cdots OP_i X_i \cdots OP_j X_j \cdots OP_m X_m \quad (3)$$

取得時間の最小と次に小さい2つの変数 X_i，X_j の演算を考えると変数 X_i，X_j に対する式3の部分式は式4となる．

$$OP_i\ X_i\ OP_j\ X_j := OP_q\ V_q \quad (4)$$

$$V_q := (X_i\ OPr\ X_j) \quad (5)$$

このとき演算子 OP_i，OP_j，OP_q，OP_r には**表1**に示した関係がある．

表1 演算子の変換則
Table 1 Rules of operations

	OP_i	OP_j	OP_q	OP_r
空を含む演算	ϕ	α	ϕ	α
	α	ϕ	ϕ	α
算術演算	+	+	+	+
	+	−	+	−
	−	+	−	−
	−	−	−	+
	×	×	×	×
	×	÷	×	÷
	÷	×	÷	÷
	÷	÷	÷	×
論理演算	∧	∧	∧	∧
	∨	∨	∨	∨

$\alpha = \{+, -, \times, \div, \wedge, \vee\}$

式3は変数が一つ減って式6となって式5で表された決定ノード N_q が生成され，このノードの演算実行時間は OP_r によって規定される．

$$X_n := X_0 OP_1 X_1 \cdots OP_q V_q \cdots OP_m X_m \quad (6)$$

未決定ノードの入力変数に，演算子 OP_i，変数 X_i，変数 X_i の取得時間 $T_g(X_i, PU_k)$ をタプル $(OP_i, X_i, T_g(X_i, PU_k))$ として与える．2つのタプルを用いて未決定ノードから決定ノードを生成する．生成された決定ノードの一時変数にも同様のタプル $(OP_q, V_q, T_g(V_q, PU_k))$ を与える．なお，一時変数 V_q の定義時間 $T_d(V_q, PU_k)$ は次式で与えられる．

$$T_d(V_q, PU_k) = max(T_g(X_i, PU_k), T_g(X_j, PU_k), T_s(PU_k)) + T_e(N_q) \quad (7)$$

ここで，$T_s(PU_k)$ はプロセッサ PU_k での実行開始可能時間である．これらのタプルと式3〜7によって統一的に完全タスクグラフが生成できる．**図2(a)** は未決定ノード，(b) は生成された決定ノードからなる完全タスクグラフを示す．図2において入力変数の取得時間は0，加減演算時間を10とした．

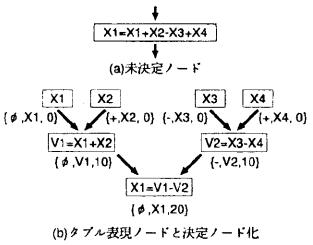

図2 未決定ノードの分解と生成
Fig. 2 Partitioning and generating of undecidednode

3. ESHアルゴリズム

3.1 概　　要

ESHでは基本ブロックを構成する代入文を式の評価順序に従って決定ノードと未決定ノードが混在する不完全タスクグラフを作成する．つぎに不完全タスクグラフのすべてのノードにノード優先度をつける．親ノードがすべてプロセッサ割当を終了したノードのなかでノード優先度の高いノードから順に，そのノードが決定ノードであればDSHによりスケジューリングをおこなってプロセッサに割り当てる．未決定ノー

```
algorithm ESH(BB,GC)
  input:  BB /* 基本ブロック */
  output: GC /* ガントチャート */
  (1)基本ブロックからタスクグラフTGを作成.
  (2)TGの各ノードのノード優先度を計算.
  (3)while(TG≠空) begin
  (4)   TGで親ノードの全てがスケジューリング済の
        ノードで最もノード優先度が高いものを
        選びCNとする.
  (5)   if(CNは決定ノード?) then
  (6)    RESNODE(TG,GC,CN) /*未決定ノードを
                             決定ノードに分解 */
        else
  (7)    DSH(TG,GC,CN)    /* 決定ノードをDSHで
                             スケジューリング */
        endif
  (8)タスクグラフを更新する.
  (9)end while
end algorithm
```

図3 ESHスケジューリング
Fig. 3 Algorithm of ESH scheduling

ドであれば決定ノードを生成しながらプロセッサ割当をおこなう．プロセッサ割当はガントチャートで表現する．図3にESHのスケジューリング概要を示す．図3のDSH(TG,GC,CN)はタスクグラフTG，ガントチャートGC，スケジュールの対象ノードCNを入力として，DSHによって対象ノードCNをプロセッサに割り当てる．その結果をガントチャートGCに書き込む．RESNODE(TG,GC,CN)は未決定ノードのスケジューリングをおこなう．

GSHでは与えられたタスクグラフのノードをそのままの形でスケジューリングするが，ESHではノードを生成してタスクグラフを再構成しながらプロセッサに割り当て，タスクグラフが完成したときにはスケジューリングが完了する．

3.2 未決定ノードの並列化

式3に示した未決定ノードの代入文を式4によって式5，6で示した2つの代入文に分解する．まず式3の入力変数の取得時間をプロセッサごとに算出する．1つのプロセッサPU_kにおける入力変数の取得時間の最も小さい2変数X_i, X_jの取得時間を$T_g(X_i, PU_k)$，$T_g(X_j, PU_k)$とする．そのプロセッサで式5の代入文を実行したときの一時変数V_qのそのプロセッサでの取得時間$T_g(V_q, PU_k)$をDSHでスケジューリングすることによって算出する．すべてのプロセッサについてそのプロセッサでの取得時間$T_g(V_q, PU_k)$を求め，そのなかで最小の取得時間$T_g(V_q, PU_s)$が得られるプロセッサPU_sに式5で生成された決定ノードを割り当てる．そして未決定ノードの代入文を式6に変更する．この操作を式6が2変数になるまで繰り返す．図4に未決定ノードから決定ノードの生成アルゴリズムを示す．

```
algorithm RESNODE(TG,GC,UN)
   input: TG /* タスクグラフ */
   GC /* ガントチャート */
   UN /* 未決定ノード */
   output: GC   /* 未決定ノードUNが決定ノードに
                   分解されてスケジューリング
                   された結果のガントチャート */
   (1)for j:=1 to no_of_PU do begin
        / *PUの数まで繰り返す */
   (2)  未決定ノードUNの入力変数X0,X1,…,Xmの
        プロセッサPUjにおける取得時間を計算して
        タプルを作成.
        TSj={(OP0,X0,Tg(X0,PUj)),
     (OP1,X1,Tg(X1,PUj)),…,(OPm,Xm,Tg(Xm,PUj))}
   (3)end for
   (4)k:=no_of_var  /* 未決定ノードの入力変数の
                       数をkに設定 */
   (5)while(k≧2)   begin
   (6)  i:=no_of_var-k+1
   (7)  for j:=1 to no_of_PU do begin
   (8)    TSjのなかで取得時間の最小のタプルと次
          のタプルをTP1,TP2とする.
   (9)    TP1,TP2から決定ノードNEW_NODEk
          (一時変数Vkを定義する代入文)と
          そのタプルTPkを候補として生成しTGの
          更新.
   (10)   DSH(TG,GC,NEW_NODEk)
   (11)   Vkの取得時間をTg(Vk,PUj)とする.
   (12)   GCとTGを(9)の実行前に戻す.
   (13) end for
   (14) min(Tg(Vk,PU1),…,Tg(Vk,PUno_of_PU))
        をTg(Vk,PUs)とする.
   (15) このTg(Vk,PUs)に対応する決定ノード
        NEW_NODEkをTGに追加し，未決定ノードUN,
        TGを更新.
   (16) for j:=1 to no_of_PU do begin
   (17)   決定ノードNEW_NODEkの入力変数に対応す
          るタプルをTSjから除去し一時変数Vkの
          タプルを追加.
   (18) end for
   (19) k:=k-1
   (20)end while
end algorithm
```

図4 未決定ノードの分解
Fig. 4 Algorithm of partitioning undecidednode

4. 実　　験

4.1 例題プログラムと実験結果

図5に示した5つ簡単なプログラムを，文レベルのDSHによる並列化，THRとGSHによる変数レベルの並列化，THRとDSHによる変数レベルの並列化，ESHによる変数レベルの並列化をおこなった．評価にあたっては包含検索機能付き機能メモリによるハードウェアバリア同期機構(MCAMU)とマルチキャスト機能付きのグループ共有メモリ(GSM)をもった細粒度マルチプロセッサMSBM[8]を用いた．

また，MSBMの特性に合わせてDSHを修正(修正DSH)した．修正DSHとDSHの相違は次の2点である．

```
void Ex1{
  int x1=1,x2=2,x3=3,x4=4;

  x1=x1+x2+x3-x4;
  x2=x1+x2-x3+x4;
}
void Ex2{
  float x1,x2;
  float a=1.1,b=1.2,c=1.3,d=1.4,e=1.5,f=1.6;

  x1=a*b/c+d-e+f;
  x2=a+b-c+d/e*f;
}
void Ex3{
  int i,n6,j=1,k=2,m=3,e1[100],e2[100];

  j=j*(k-j)*(m-k);
  k=m*k-(m-j)*k;
  m=(m-k)*(k+j);
  e1[m-1]=j+k+m;
  e2[k-1]=j*k*m;
}
void Ex4{
  float x1=1.1,x2=1.2,x3=1.3,x4=1.4,x5=1.5,X6=1.6;
  float t=0.2;

  x1=t*(x1+x2+x3+x4+x5-x6);
  x2=t*(x1+x2+x3+x4-x5+x6);
  x3=t*(x1+x2+x3-x4+x5+x6);
  x4=t*(x1+x2-x3+x4+x5+x6);
  x5=t*(x1-x2+x3+x4+x5+x6);
  x6=t*(-x1+x2+x3+x4+x5+x6);
}
void Ex5{
  int i=1;
  float s,u,v,w,x,one=1.0;
  float A1=2.1,A2=2.2,A3=2.3,A4=2.4,A5=2.5,A6=2.6;
  float B1=3.1,B2=3.2,B3=3.3,B4=3.4,B5=3.5,B6=3.6;

  u=i*x;
  w=u*u;
  v=u*A6*w*w*w*w*w*w+A5*w*w*w*w*w+
    A4*w*w*w*w+A3*w*w*w+A2*w*w+A1*w+one;
  s=s+v/(w*w*w*w*w*w*B6+w*w*w*w*w*B5
     +w*w*w*w*B4+w*w*w*B3+w*w*B2+w*B1+one);
}
```

図5 例題プログラム
Fig. 5 Example programs

(i) DSHでは明示的な同期命令の実行を想定おらずかつノード間の依存関係では真依存のみを取り扱っていたが，ESHではノード間に真依存関係があればバリア同期の挿入と通信を考慮し，逆依存および出力依存関係にはバリア同期を挿入した．なおバリア同期の挿入はZaafraniのアルゴリズム[9]を用いた．

(ii) DSHのTask Duplication Processで複製された元のノードの出力変数を参照する子ノードが存在しなくなることがある．ESHではこのようなノードをガントチャートから取り除いた．

MSBMがマルチプロセッサであることから一時変数の共有変数(一時共有変数)および恒久変数を共有メモリに，一時変数の局所変数(一時局所変数)を局所メモリに割り当てた．また**表2**に静的スケジューリングの基本データである四則演算実行時間，配列アドレス計算時間，バリア同期命令実行時間，プロセッサ間通信時間を示す．

表2 スケジューリング基本データ
Table 2 Basic data of Scheduling

加算	整数演算	4.67
	浮動小数点演算	18.50
減算	整数演算	4.67
	浮動小数点演算	19.56
積算	整数演算	10.94
	浮動小数点演算	19.56
商算	整数演算	12.61
	浮動小数点演算	25.00
配列計算	配列のアドレス計算	3.22
同期	バリア同期命令実行	1.33
通信	整数通信	1.00
	浮動小数点通信	1.39

単位は単位時間である．

図5のプログラムを最大4台のPUで並列化した速度向上比の予測値と実測値を**表3**に示し，**図6**に実測値をグラフで示した．縦軸に文レベルのDSHによる並列化を1として正規化した速度向上比を示す．

表3 速度向上比の予測値と実測値
Table 3 Estimation and measurement of speedup ratio

		スケジューリングの種類		
		THR+GSH	THR+DSH	ESH
Ex1	予測値	1.25	1.25	1.45
	実測値	1.25	1.25	1.45
Ex2	予測値	1.20	1.20	1.55
	実測値	1.19	1.19	1.52
Ex3	予測値	1.28	1.32	1.32
	実測値	1.27	1.30	1.30
Ex4	予測値	1.57	1.57	2.38
	実測値	1.54	1.54	2.32
Ex5	予測値	2.72	2.72	2.74
	実測値	2.60	2.60	2.60

5. 考　察

表3より，すべてのプログラムにおいて速度向上比は予測値，実測値図ともESH≧THR+DSH≧THR+GSHであった．また予測値が実測値より若干よい結果になっているが，演算時間のデータによるばらつき，バリア同期の成立時間のばらつきによるものである．

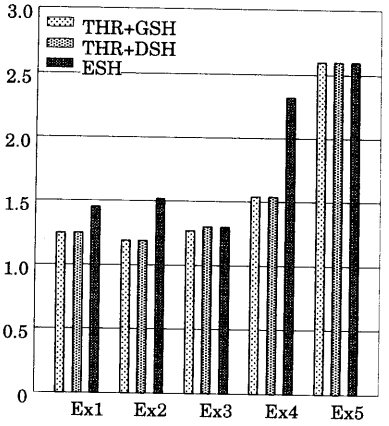

図6 MSBMにおける実測結果
Fig. 6 Speedup ratio of example programs on MSBM

Ex1では，2つの文に真依存関係が成立しているので文レベルの並列化はできなかった．THR+GSHとTHR+DSHとは同じスケジューリング結果となった．THR+DSHとESHのガントチャートを**図7**に，タスクグラフを**図8**に示す．

第1文のスケジューリングはTHR+DSH，ESHとも同一である．THR+DSHでは第2文はこの文単独で計算木の高さが最低になるようにスケジューリングされるが，ESHでは2つの文全体の計算木の実行時間が最

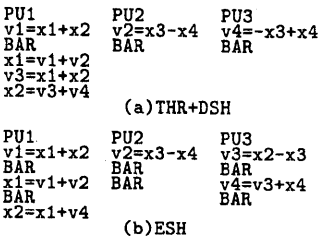

図7 THR+DSHおよびESHによるガントチャート
Fig. 7 Gantt Chart of THR+DSH and ESH

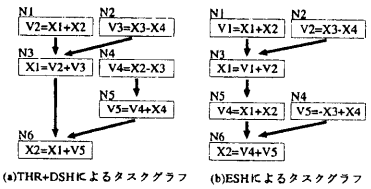

図8 Ex1のタスクグラフ
Fig. 8 Task graph of Ex1

短になるように第2文がスケジューリングされる．この結果，バリア同期はTHR+DSHでは1個所，ESHでは2個所必要であるが，バリア同期の実行時間およびプロセッサ間通信時間が演算実行時間に較べて数倍以上短かいのでESHの方が高速となった．THR+DSHに対するESHの速度向上比の実測値は1.16であった．

Ex2では，文レベルの並列化は2つの文に依存関係がないので2台のPUで独立に実行できた．Ex1と同様にTHR+GSHとTHR+DSHは同じスケジューリング結果であった．THR+DSHとESHはともに第1文で2台のPU，第2文でも2台のPUを使用した．しかし，THR+DSHでは各文の計算木の生成時に演算時間を考慮せずに計算木の高さのみに注目したために，ESHとの差が生じた．バリア同期はTHR+DSH，ESHとも1個所必要であり，プロセッサ間通信のオーバーヘッドも同じであるが，演算時間の考慮の有無がそのまま速度向上比に表れ，THR+DSHに対するESHの速度向上比の実測値は1.28であった．

Ex3では，文レベルの並列化は第4文と第5文には依存関係がないのでこの部分のみ2台のPUで実行できた．THR+DSHでは4台のPUによりスケジューリングを行い，THR+DSHおよびESHでは4台のPUによりスケジューリングを行い同じ結果を得た．THR+DSHはバリア同期は3個所必要であるが，第3文の(m−k)を複製することによってTHR+GSHよりもバリア同期が1個所少なくなりプロセッサ間通信も1回少なくなっている．THR+DSHのTHR+GSHに対する速度向上比の実測値は1.02であり，これは複製による効果である．一方THR+DSHとESHとは同じスケジューリング結果となった．これは各文の未決定ノードが3項演算以下であるために決定ノード化する自由度がなかったことによる．THR+DSHに対するESHの速度向上比は当然1.00である．Ex4では，各文が順次真依存関係があるので文レベルの並列化はできない．THR+GSHとTHR+DSHは4台のPUによる同じスケジューリング結果を得た．ESHも4台のPUを使用した．ESHのTHR+DSHに対する速度向上比の実測値は1.51であったが，これはEx1と同じ理由による．Ex5では，Ex4と同様に文レベルの並列化はできない．THR+GSHとTHR+DSHは4台のPUによる同一のスケジューリング結果を得た．ESHも4台のPUを使用し，THR+DSHに対するESHの速度向上比の予測値は1.01であった．ESHのガントチャートはTHR+DSHと全く異なっており多項演算の未決定ノードが存在するにも拘わらず同一変数の乗算であるのでTHR+DSHでは変数の取得時間の影響はない．ESHのTHR+DSHに対する速度向上比の実測値は1.00であった．

以上のことから，ESHは，1つの文の計算木の実行時間が最短になるのみならず，複数文全体の計算木の実行時間が最短になるようにスケジューリングされる．ESHは細粒度並列化のアルゴリズムとして有効であることが確認された．

6. むすび

変数レベルの並列化アルゴリズムESHについて述べた．ESHは複製というDSHの利点を活用しつつ，決定ノードと未決定ノードが混在した不完全タスクグラフに対して変数の取得時間に基づいて2項演算をプロセッサに割り当てる．すべてのノードが決定ノードになったときすなわち完全タスクグラフが生成されたときスケジューリングが完了する．これによって複数の文間で内在している並列性を抽出し，計算木を最適に生成していくことができる．

5つの簡単なプログラムを用いて，3つのスケジューリングアルゴリズムすなわちESH，THRとGSHの併用，THRとDSHの併用を評価した．これらすべてのプログラムにおいて速度向上比はESH≧THRとDSHの併用≧THRとGSHの併用であり，ESHの有効性が確認できた．今回の実験では細粒度マルチプロセッサMSBMを用いて行ったが，インオーダ命令発行スーパースカラプロセッサやVLIWプロセッサにおける静的スケジューリングアルゴリズムとして使用すれば速度向上比がさらに上昇すると思われる．

今後の課題は，このアルゴリズムをスーパースカラプロセッサ，VLIWプロセッサ，細粒度マルチプロセッサ用コンパイラに実装することである．

謝辞 本研究にご助力頂いた（株）東芝日野工場 木村 勝彦氏に深謝します．

参考文献

1) M.Johnson(村上和彰監訳)：スーパースカラプロセッサ，日経BP出版センター(1994)
2) H. El-Rewini,T. Lewis,H. Ali:Task Scheduling in PARALLEL and DISTRIBUTED SYSTEMS, Prentice Hall(1994)
3) Joseph A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction", IEEE Transaction on Computers, Vol.C-30, No.7, July 1981
4) 富田眞治: 並列計算機構成論, 昭晃堂 (1987)
5) 笠原博徳: 並列処理技術, コロナ社 (1991)
6) 尾形航他: スタチックスケジューリングを用いたマルチプロセッサシステム上での無同期近細粒度並列処理, 情報処理論,Vol.35,No.4,pp.522-531,Apr.,1994
7) 村岡洋一: 超並列処理コンパイラ, サイエンス社 (1990)
8) 岩根雅彦他: 細粒度マルチプロセッサMSBM, 情報処理論,Vol.37,No.6,pp.1196-1205,Jun.,1996
9) A.Zaafrani,H.G.Diets,M.T.O'Keefe:Static Scheduling for Barrier MIMD Architectures, Int. Conf. Parallel Processing,pp.II.187-II.194,1990