数理解析研究所講究録
1148巻 2000年 17-22

レベル付き依存グラフを用いた効率のよいソフトウェア・パイプライン化法について

三重大学工学部　武市雅俊（Masatoshi Takeichi）
三重大学工学部　大山口通夫（Michio Oyamaguchi）
三重大学工学部　太田義勝（Yoshikatsu Ohta）

1. はじめに

近年ＶＬＩＷやスーパスカラ、マルチプロセッサといった、複数の命令を同時に実行することのできるアーキテクチャが広く普及している。これらのアーキテクチャではコンパイラによって、並列実行度の高いコードを生成することが重要である。一般にプログラムの実行時間の多くはループの実行に費やされる。そのためループを最適化することで、多大な効果を期待することができる。

ループの最適化手法の中でも命令の並列性を抽出するための非常に重要な手法としてソフトウェア・パイプライン化法（SP法）がある。

SP法とは、ループを解析してパイプライン化し、並列実行度の高い新しいループに再構成しなおすことにより実行効率を高める手法であり（図１）、これまで多くの研究者によって盛んに研究されてきた[1～7]。その基本となる本質的なアイデアは、１個の基本ブロックを本体にもつ１重のループ(以後、単純ループと呼ぶ)に限定し、かつ資源（プロセッサ数,レジスタ数など）の制約条件を加味しない場合について提案されたSP法にあり、これまでの研究成果は、次の２つに分類することができる。

```
For(I=1;I<1000;I=I+1){
  A: a[I]=b[I-1]+3;
  B: b[I]=a[I]-4;
  D: c[I]=2*a[I];
  C: d[I]=b[I]-1;
}                          (a)元のループ
A(1)                    }  プロローグ部
B(1) D(1)               }
For(I=1;I<999;I=I+1){   }
C(I)   A(I+1)           }  カーネル部
B(I+1)   D(I+1)         }
}                       }
C(1000)                 }  エピローグ部
                           (b)SP化したループ
```

図１　ソフトウェア・パイプライン

まず一つは、実行サイクルが最適となるものである[1]。このとき新しいループ本体の命令数は、元のループ本体のより増加する場合があり、キャッシュ容量を越えるなどで効果が低減する場合がある。またカーネル部の作成に多大な時間を要する場合もある。

もう一つのアプローチは新しいループ本体の命令数の増加を許さない発見的スケジューリングである[3][4][5]。この方法では効率よくスケジューリングが完了するが、並列実行度を最大限引き出すことができるとは限らない。

本研究では、依存グラフにレベルの概念を導入したものを用いることにより、２つのアプローチのギャップを埋める、効率のよいSP法を提案する。

2. 準備

ここでは準備として本研究で用いる用語等について述べる。ループ本体のｉ（>0）回目の実行をｉ回目のイタレーションと呼ぶ。プログラム中の命令間にはある命令ｕの実行が終了しないと、ある命令ｖの実行が開始できないという依存関係が存在する。命令の依存関係を本手法いて大きく二つに分類する。同じイタレーション中に現れる２つの命令ｕとｖに依存がある場合、その依存はループ内依存と呼び、(u,v)と表記する。ｉ回目のイタレーション中に現れる

命令 u から i+j 回目のイタレーション中に現れる命令 v に依存がある場合、その依存をループ運搬依存と呼び、$[u,v]_j$ と表記する。但し $j \geq 1$。なお、j>1 であるループ運搬依存があるときは、(j-1) 回のループ展開を行うことにより j=1 のループ運搬依存に変更可能である。本研究では、すべてのループ運搬依存に対して j=1 と仮定して話を進める。

2.1. 依存グラフ

命令の依存関係を表す依存グラフは、各命令に対応する表す頂点の集合 V と、依存関係を表す有向辺の集合 E∪E'によって表現される(図2)。ここで、E はループ内依存の集合、E'はループ運搬依存の集合である。ループ内依存のエッジは実線で、ループ運搬依存は破線で表す。部分グラフ G=(V,E)は非循環有向グラフ(dag)となる。親を持たないノードを根と呼び、子を持たないノードを葉と呼ぶ。グラフ G に対し、次の関数を与える。ここで、$v \in V$。

- Des(v)は命令 v の子孫の集合
- dep(v)は命令 v の深さ
- hei(v)は命令 v の高さ
- L は G の高さ、即ち根から葉への路の長さの最大値。

dag において根は複数個存在しうる。高さ L の根を主根と呼ぶ。

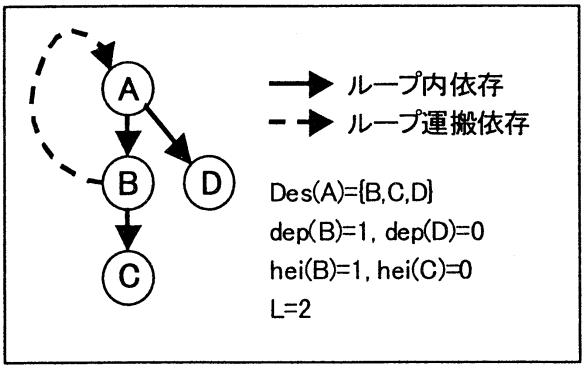

図 2 依存グラフ

2.2. スケジューリング

本稿では関数 $s_G: V \to N$ を dag G のスケジューリングと呼ぶ。ここで $N=\{0,1,2,\cdots\}$ である。G が明らか場合は省略する。

スケジューリング s は次の条件を満たすとき正当であると呼ぶ。

条件：$\forall (u,v) \in E \quad s(u) < s(v)$

これは全てのループ内依存を満たすということである。定義より、関数 dep は G の正当なスケジューリングである。

s(v)=0 を満たす命令 v が存在するとき、標準スケジューリングと呼ぶ。スケジューリング s に対して $m = \mathrm{Min}\{s(v) \mid v \in V\}$ とするとき、$s(v) = s(v) - m$ で定義される s を s の標準スケジューリングと呼ぶ。この新しいスケジューリング s を norm(s)又は s-m と表す。

スケジューリング s に対して、$h_s: V \to N$、$L_s \in N$ を次のように定義する。

$$h_s(v) = \begin{cases} 0 & v \text{が葉のとき} \\ \mathrm{Max}\{s(u) - s(v) \mid u \in \mathrm{Des}(v)\} & v \text{が葉以外のとき} \end{cases}$$

$$L_s = \mathrm{Max}\{s(v) + h_s(v) \mid v \in V\}$$
$$(= \mathrm{Max}\{s(v) \mid v \in V\})$$

h_s はスケジューリング s において命令 v から、命令 v の子で最も遅くスケジューリングされた命令の実行までにかかる時間を表す。L_s は s の実行にかかる時間を表す。

正当なスケジューリング s、s'に対して、その組（s, s'）が両立するとは、

$$\forall (u,v) \in E' \quad s(u) < s'(v)$$

が成立するときである。（s, s'）が両立するとき、s と s'はそれぞれ i 回目と i+1 回目のイタレーションのスケジューリングにすることができる。

2.3. レベル付き依存グラフ

本手法では、イタレーションごとにスケジューリングを行う。そのため命令間の依存関係とそのイタレーションのスケジューリングを同時に表現するために、依存グラフにレベルの概念を導入する。

依存グラフの各ノードにはレベルの属性を持たせる。レベルは各命令の実行順を表し、同じレベルをもつノードは、同じ時刻に実行されることを表す。図3は図2のグラフにおいて、ノード D に異なるレベルをつけたものである。こ

のようにレベル付き依存グラフを用いることで、同じ依存グラフに対しても異なるスケジューリングを表現することができる。

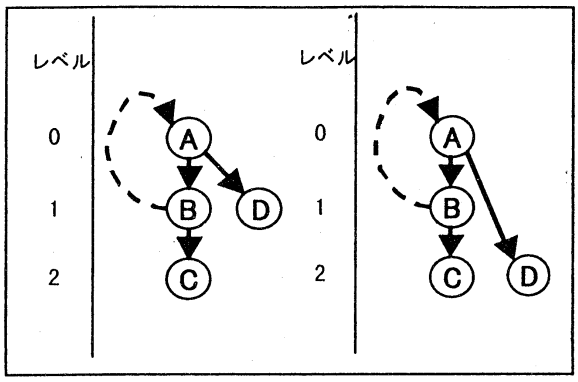

図 3　レベル付き依存グラフ

3. 従来の SP 法

ここでは従来提案されたＳＰ法のうち、アプローチの異なる手法を１つずつ簡単に述べる。

3.1. OLP 法

Aiken らによって提案された手法（OLP 法）[1]は最適スケジューリングを行うものである。各命令は可能な限り早い時刻にスケジューリングされる。スケジューリングが進むと、命令は幾つかのグループに分かれて出現するようになる。同じグループが出現するようになれば、グループ間にできた、命令が一つもスケジューリングされていない区間（ギャップ）を詰め繰り返しパターンを見つける。図４(ｂ)は図４(a)のループに OLP 法の適用を行った例である。

OLP 法ではリソースが無制限ならば実行時間が最適となるコードを得ることができる。しかしカーネル部の命令数が指数関数的に大きくなる場合があり、リソースが充分でない場合は、キャッシュ容量を上回るなどで効果が低減する。またパターンの発見までに多大な時間を要する場合もある。

3.2. URPR 法

Su らは URPR 法（UnRolling, Pipelining and Rerolling）という手法を提案している[4]。アルゴリズムはまずループを圧縮することから始まる。圧縮されたループに対し、展開数を決定するために次の値を計算する。

$$D=\mathrm{Max}\ \{dep(u)-dep(v)+1 \mid [u,v]\in E'\}$$

D は命令の移動を行わずに、ループ運搬依存を満足させる為に必要な値である。次にループ展開（unrolling）を展開数 K= L/D 回で行う。展開されたループはイタレーションごとに、グラフの形を変えることなく D ずつずらしてスケジューリングされる。そしてループの巻き戻し（rerolling）を行いカーネル部、プロローグ部、エピローグ部が作成される。リソース競合がなければカーネル部は高さ D となる。アルゴリズムの計算量は元のループ本体の命令数を n とすれば $O(n^2)$である。図４(c)は図４(a)のループに URPR 法の適用を行った例である。

4. 新しい SP 法

従来提案された SP 法は、計算時間や並列実行度、カーネル部の命令数増加などいくつかの問題を持っている。そこで本研究では、従来の SP 法を改善した新しい SP 法を提案する。

4.1. 遅延幅

本研究では各イタレーションの開始の間隔を遅延幅と呼ぶことにする。従来提案された SP 法において、遅延幅に相当する値は計算しない[1]、あるいは元のループに対して１回しか計算しない[3][4][7]。本手法ではより適切な遅延幅を求めるために、遅延幅はイタレーションのスケジューリングごとに新たに計算を行う。

本手法のスケジューリングでは各イタレーションはその高さを変えないようにスケジューリングを行い、アルゴリズムの効率化を図る。新しくスケジューリングされるイタレーションが高さ L_s を保存するため、スケジューリング s に対して、遅延幅 d_s は次式となる。

$$d_s=\mathrm{Max}\ \{s(u)+1+h_s(v)-L_s \mid [u,v]\in E'\}$$

ちなみに URPR 法の遅延幅は

$$\mathrm{Max}\ \{s(u)+1+(L_s-s(v))-L_s \mid [u,v]\in E'\} = s(u)-s(v)+1$$

で与えられる。

4.2. 次のイタレーションのスケジューリング

遅延幅の計算が終わると、それを用いて次のイタレーションをスケジューリングする。正当なスケジューリング s から次のイタレーション

のスケジューリングs'を次のように定義する。

$s'(v)=\mathrm{Max}(\{s(u)+1 \mid [u,v]\in E'\} \cup \{s'(u)+1 \mid (u,v)\in E\} \cup \{s(v)+d_s\})$

この新しいスケジューリングs'を next(s)で表す。

定理1　sを正当なスケジューリング、s'=next(s)とする。そのとき、次の(1)、(2)が成立する。

(1) s'は正当なスケジューリングである。

(2) 組（s, s'）は両立する。

定理1はs'の定義より明らかに成り立つ。

標準スケジューリングsから得られたs'=next(s)に対し、次の性質1～4が成立する。

性質1　$\forall v\in V$　$s'(v)+h_s(v)\leqq d_s+L_{ss}$

(証明)dep(v)に関する帰納法で示す

<u>vが根のとき（dep(v)=0）</u>

・$s'(v)=s(v)+d_s$のとき

L_sの定義より、$s(v)+h_s(v)\leqq L_s$

ゆえに、性質1を満たす。

・$s'(v)=s(u)+1$（但し$(u,v)\in E'$）のとき

d_sの定義より、$s(u)+1+h_s(v)-L_s\leqq d_s$

従って、$s'(u)+1+h_s(v)-L_s\leqq d_s$

ゆえに性質1を満たす

<u>vが根でないとき（dep(v) >0）</u>

・$s'(v)=s(u)+1$（但し$(u,v)\in E'$）のとき

d_sの定義より、$s(u)+1+h_s(v)-L_s\leqq d_s$

従って性質1をみたす

・$s'(v)=s'(u)+1$（但し$(u,v)\in E$）のとき

帰納法の仮定より、$s'(u)+h_s(v)\leqq d_s+L_s$

$s(u)<s(v)$より、$h_s(v)\leqq h_s(u)-1$

従って

$s'(v)+h_s(v)\leqq s'(u)+1+h_s(u)-1$

$=s'(u)+h_s(u)\leqq d_s+L_s$

ゆえに性質1を満たす

・$s'(v)=s(v)+d_s$のとき

L_sの定義より、$s(u)+h_s(v)\leqq d_s+L_s$

従って、$s'(u)+h_s(v)\leqq d_s+L_s$

ゆえに性質1を満たす。（証明終わり）

性質1より次の性質2が成立する。

性質2　$\forall v\in V$　$d_s\leqq s'(v)\leqq d_s+L_s$

従って、$L_{s'}\leqq d_s+L_s$

(証明)$s'(v)\leqq d_s+L_s$は性質1より、$d_s\leqq s'(v)$はs'の定義より明らか　(証明終わり)

性質2よりs'$-d_s$は正当な標準スケジューリングである。

スケジューリングの系列$S=<s_0, s_1,\cdots>$を次のように定義する。

（1）$s_0=dep$、$s_0'=s_0$

（2）$i>0$のとき、$s_i=next(s_{i-1}')$、$s_i'=norm(s_i)$

そのとき、次の性質が成立する。

性質3　$L=L_{s0}=L_{s1'}=\cdots$

(証明)$s_0=dep$より、$L_{s0}=L$が成立するのは定義より明らか。従って性質2より$d_s\leqq s'(v)\leqq d_s+L_s$

従って、$s_1'=s_1-d_s$より　$L_{s1'}\leqq L$

s_1'が正当なスケジューリングかつ$\mathrm{Max}(dep(v))=L$より$L_{s1'}\geqq L$　ゆえに$L_{s1'}=L$

同様の議論により、$i>1$においても$L_{s1'}=L$が成立する。　(証明終わり)

性質2と3より、頂点vが主根であれば任意の$i>0$に対して$s_i(v)=d_{si-1}$かつ$s_i'(v)=0$が成立する。

性質4　$\forall i\geqq 0$、$\forall v\in V$　$s_i'(v)\leqq s_{i+1}'(v)$

(証明)nextの定義より

$\forall v$　$s_{i+1}(v)\geqq s_i'(v)+d_{si}$

したがって、$s_{i+1}'(v)\geqq s_i'(v)$　(証明終わり)

定義　スケジューリングの系列$S=<s_0, s_1,\cdots>$が<u>単調増加</u>とは、

$\forall i\geqq 0$、$\forall v\in V$　$s_i(v)\leqq s_{i+1}(v)$

が成立するときである。

定理2　スケジューリングの系列$S'=<s_0', s_1', \cdots>$は単調増加である。但し$s_0'=dep$、$s_{i+1}'=norm(next(s_i'))(i>0)$である。さらに、

$\exists j\leqq L\cdot|V|$　$s_j'=s_{j+1}'$

(証明)単調増加は性質4より明らか、従って、スケジューリングsに対して$D_s=\Sigma_{v\in V}s(v)$と定義すると、$\forall i\geqq 0$　$D_{si'}\leqq D_{si+1'}$

性質3より$D_s\leqq L\cdot|V|$が成立する。ゆえに定理が成立する。（証明終わり）

4.3. アルゴリズム

本手法は以上に述べた関数を用いて次のアル

ゴリズムによってSP化をおこなう。

(開始)

1. 初期スケジューリング　s_0=dep
2. スケジューリングs_iに対して、
   - d_sを計算する。
   - next(s_i)を計算しs_{i+1}とする。

   norm(s_{i+1})≠norm(s_i)なら、next(s_{i+1})を新しいs_iとして2.の最初、等しければ3へ
3. カーネル部の作成
   - カーネル部の切り出し
   - indexの調整
4. プロローグ部、エピローグ部の作成

(終了)

本アルゴリズムの正当性は定理1により保証される。

次に本手法の時間計算量を考える。アルゴリズムの1では、ノード数をn、ループ内依存のエッジ数をeとすれば、O（n＋e）である。2において d_s の計算にはループ運搬依存のエッジ数をe'とすればO（e'）、next(s_i)の計算にO（e＋e'）かかり、この計算を定理2により最悪の場合L・n回繰り返すことになるのでm＝e＋e'とすればO（L・m・n）となる。ここでLは元のループにおける依存グラフの高さである。3、4についても同じ時間計算量で抑えることができる。以上より本手法の時間計算量はO（L・m・n）となる。

5. 比較

ここでは本手法と、OLP法、URPR法を図4の例で比較を行う。表1は各手法を適用した際のカーネル部の命令数と、繰り返し回数をk回としたときのカーネル部の実行時間を、プロセッサ数（PE）が4つの時、5つ以上の時に分けて示したものである。

	カーネル部命令数	実行サイクル数	
		4PE	5PE以上
元のループ	12	6k	6k
OLP法	36	3.33k	2.67k
URPR法	12	5k	4k
本手法	12	3k	3k

表1　比較

ここでは1つの例を示しただけだが、他の多くのループに対しても、本手法によって、OLP法のようにループの命令数を増やすことなく、URPR法以上の並列度のコードを生成することできる。

6. まとめ

本研究では、レベル付き依存グラフを用いて、カーネル部の命令数を増やすことなく、並列実行度も高いコードを、多項式時間の計算により得る新しいSP法を提案した。

参考文献

[1] A.Aiken and A.Nicolau, "Optimal Loop Parallelization", Proc. SIGPLAN '88 conference on PLDI, pp.308-317, 1988

[2] A.W.Appel and M.Ginsburg, "Modern Compiler Implementation", Chapter20, 1997,CAMBRIDGE UNIVERSITY PERSS

[3] B.R.Rau, "Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops", MICRO.27, pp.63-74, 1994

[4] B.Su, et al. "URPR−An Extension of URCR for Software Pipelining", MICRO.19, pp.104-108, 1986

[5] 武市雅俊、大山口通夫、太田義勝、"マルチプロセッサ向きループ最適化手法について"、電気関係学会東海支部連合大会 講演論文集、p.332、1999

[6] 村上智彰、"スーパスカラ・プロセサの性能を最大限に引き出すコンパイラ技術"、日経エレクトロニクス(No.521)、pp.165-185、1991

[7] 古関聰、他、"命令レベル並列アーキテクチャのためのループアンローリングおよびソフトウェア パイプライニング適用技法"、電子情報通信学会論文誌、Vol.J-80-D-Ⅰ、No.10、pp.824-835、 1997

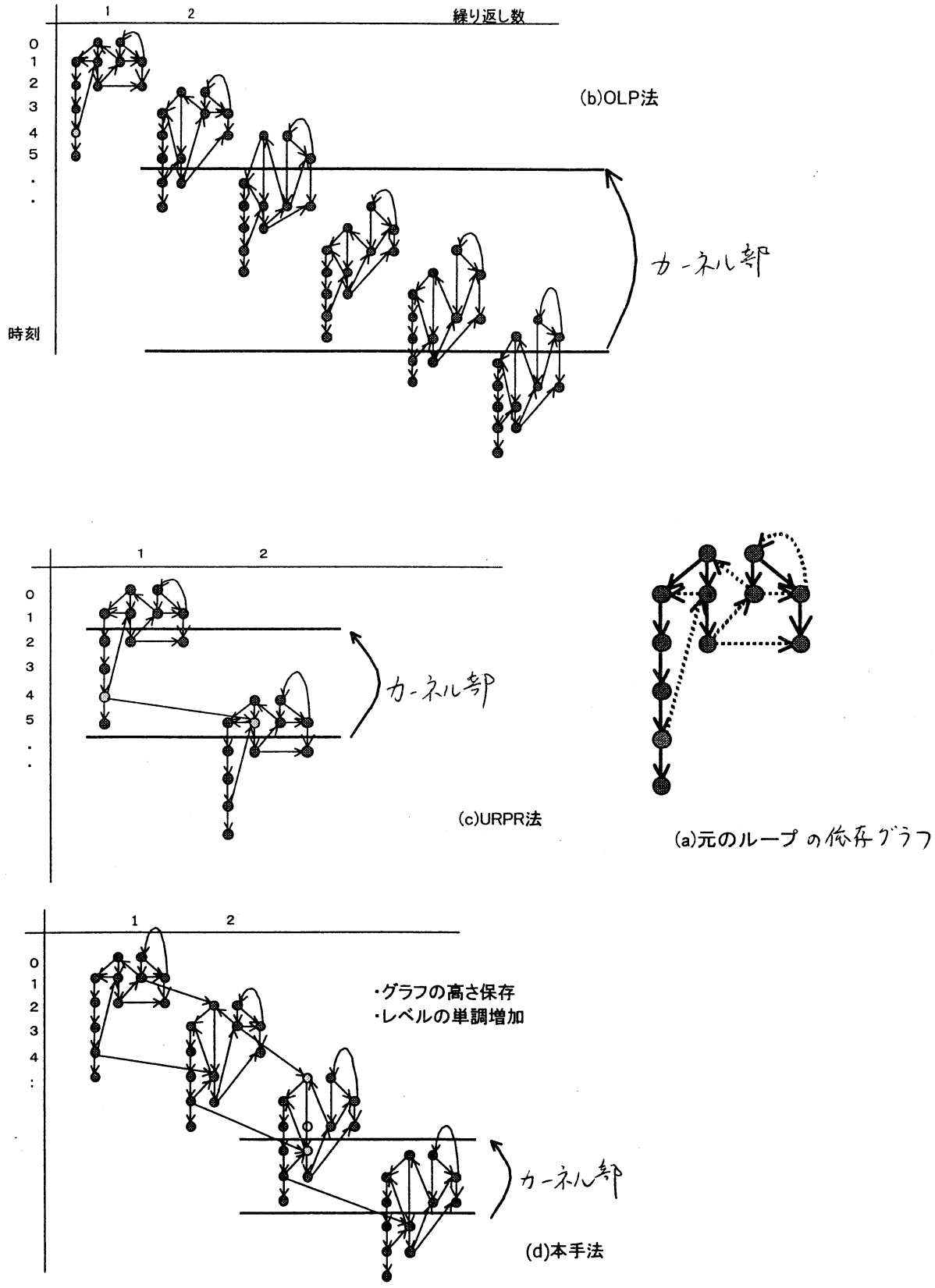
1
2
繰り返し数
0
1
2
3
4
5
時刻
(b)OLP法
カーネル部
1
2
0
1
2
3
4
5
カーネル部
(c)URPR法
(a)元のループの依存グラフ
1
2
0
1
2
3
4
・グラフの高さ保存
・レベルの単調増加
カーネル部
(d)本手法

図 4　SP 法の適用