

## ループによって運ばれる依存を有するループの並列実行時間の見積り

中西 恒夫<sup>†</sup>, 城 和貴<sup>†</sup>, Constantine D. Polychronopoulos<sup>‡</sup>,  
福田 晃<sup>†</sup>, 荒木 啓二郎<sup>†</sup>

†: 奈良先端科学技術大学院大学 情報科学研究科  
630-01 奈良県生駒市高山町 8916 番地の 5

‡: Center for Supercomputing Research and Development,  
University of Illinois at Urbana-Champaign  
1308 West Main St. Urbana, IL 61801, U.S.A.  
E-mail: nakasu-para@is.aist-nara.ac.jp

### 概要

ループは並列性の重要な源であり、またプログラムの実行時間の大部を占める。そのためこれまで、マルチプロセッサ用に様々なループのイタレーション単位（中粒度）での最適化技法が提案、使用されてきた。一方、スーパースカラ技術をはじめとする、近年のプロセッサ内部における細粒度並列処理技術の普及により、ループボディ内の最適化技法が重要になりつつある。ループ全体の並列実行時間の下限は、こうした異なる粒度での最適化手法の効果をも、統一かつ定量的に評価するのに有用な尺度であろう。ループ全体の並列実行時間の下限は、ループを完全に展開したコードの依存グラフのクリティカルパスのコストを求めることにより得ることができる。しかしながら、ループを完全に展開することは実用上極めて問題である。本稿では、問題を整数計画問題に帰着させ、このクリティカルパスのコストを、ループを一切展開することなく得る手法を提案する。さらに提案手法を実装、実際のプログラムに適用し、提案手法が実用的な時間でクリティカルパスのコストを与えることを示す。

## Estimating Parallel Execution Time of Loops with Loop-Carried Dependences

Tsuneo Nakanishi<sup>†</sup>, Kazuki Joe<sup>†</sup>, Constantine D. Polychronopoulos<sup>‡</sup>,  
Akira Fukuda<sup>†</sup>, Keiji Araki<sup>†</sup>

†: Graduate School of Information Science,  
Nara Institute of Science and Technology

8916-5, Takayama-cho, Ikoma-shi, Nara 630-01, Japan

‡: Center for Supercomputing Research and Development,  
University of Illinois at Urbana-Champaign  
1308 West Main St. Urbana, IL 61801, U.S.A.

E-mail: nakasu-para@is.aist-nara.ac.jp

### ABSTRACT

Loops are a rich source of parallelism and account for the largest part of program execution time, thus various kinds of loop restructuring schemes have been applied to loops. On the other hand optimization schemes for loop bodies increase their importance because of popularization of the fine-grain parallel processing technology such as the superscalar technology in recent years. The infimum of parallel execution time of a whole loop will be a reasonable criterion to evaluate the application effects of these optimization schemes with different granularity in a unified and quantitative manner. The infimum can be obtained by completely unrolling a given loop, generating a dependence graph (DAG) of the unrolled loop, and evaluating a critical path cost of the dependence graph. However, this method lacks scalability against the loop size. In this paper we propose a scheme to evaluate the critical path cost of the dependence graph of the completely unrolled code of a given loop without unrolling the loop at all by reducing the problem into an integer programming problem. Furthermore, we do an experimental implementation to apply the proposed scheme to actual programs and demonstrate that the practical computational complexity of our algorithm is extremely small and especially independent of the loop size.

## 1 はじめに

自動並列化コンパイラでは、スケジューリングあるいはリストラクチャリングにおいて、タスクとタスク間の実行順序に関する制約を表す依存グラフがよく用いられる。依存グラフの節点、枝には、しばしば該当するタスクの実行コスト（実行時間）、該当する依存に關与する通信の通信コスト（通信時間）が属性として与えられる。

基本ブロックでは依存グラフは非循環グラフとなる。この時、依存グラフのクリティカルパスのコストは、プロセッサ無限個時の並列実行時間の下限を与える。このため、クリティカルパスのコストはプログラムの並列実行時間を静的に見積もる尺度として有用であり、スケジューリングやプログラム分割において頻繁に用いられる [3]。

ループのイタレーションをひとつのタスクとした時に得られる依存グラフは非循環であり、クリティカルパスを定義することができる。この時得られるクリティカルパスのコストは、ループをプロセッサ無限個でイタレーション単位で並列実行（中粒度）した時の実行時間の下限である。一方、ループを完全に展開し、その各文をひとつのタスクとした時に得られる依存グラフも非循環となるため、クリティカルパスを定義することが可能であり、そのコストはループ全体をプロセッサ無限個で並列実行（細粒度）した時の実行時間の下限である。異なるイタレーションに属するタスク間に依存が存在するようなループは、中粒度での並列化のみならず細粒度での並列化も重要である。スーパースカラ技術をはじめとする、近年の命令レベル並列処理技術の普及・低価格化は、ループの中粒度および細粒度での並列実行を現実的なものとしている。ループ全体の並列実行時間の下限を静的に見積もることは、ループ最適化において極めて有用なことである。しかしながらループを展開することは明らかに実用上問題がある。

自動並列化コンパイラでは、ループリストラクチャリングにおいて、ループボディ部の依存グラフがよく用いられる。この依存グラフの大きさはループの繰り返し数に依存せず、単にループボディ内の文の数に依存するため、極めて実用的な大きさに収まる。しかしながら、この依存グラフは一般に循環であるため、クリティカルパスを定義することはできない。本稿では、後述するある条件を満足するループについて、この依存グラフのみから、ループを展開したコードの依存グラフのクリティカルパスのコスト、すなわちプロセッサ無限個時のループの並列実行時間の下限を算出する手法を提案する。提案手法は、問題を整数計画問題に帰着させ、単体法 (simplex method) と分枝限定法を適用する。2章ではループにおける依存グラフについて諸概念の整理を行う。3章では問題がいかに整数計画問題に帰着されるか、およびその正当性を説く。4章では提案手法の実装および評価について述べる。

## 2 ループにおける依存グラフ

本節では、ループにおける依存グラフに関する諸概念の整理を行う。依存グラフでは、タスクと節点、依存と枝が1対1の対応となるため、本稿では両者を適宜入れ換えて用いることがある。

### 2.1 ループタスクグラフ

ループボディ内のタスク間の依存関係を表現した依存グラフを、ループタスクグラフと呼ぶことにする。図 1(a) のプログラムのループタスクグラフは、図 1(b) のようになる。この例のように、一般にループタスクグラフは循環であり、クリティカルパスは定義できない。

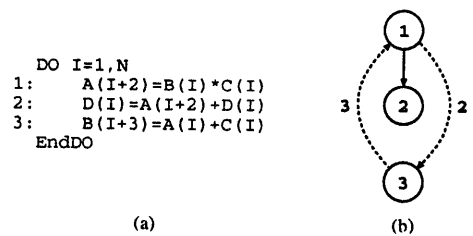


図 1: ループタスクグラフ

ループボディ内の依存は、ループ独立依存 (loop-independent dependence) とループによって運ばれる依存 (loop-carried dependence) の 2 種類に分類される [1]。前者は依存元タスクと依存先タスクがループの同じイタレーションの中に存在し、後者は異なるイタレーションの中に存在する。ループタスクグラフの自己ループは、常にループによって運ばれる依存の枝となることに注意されたい。依存元タスクの  $n$  個先のイタレーションに依存先タスクがある場合、その依存の依存距離は  $n$  であるという。ループ独立依存の依存距離は 0 である。図 1(b) の点線の枝はループによって運ばれる依存、その脇の数字は依存距離を表している。

ループタスクグラフからループによって運ばれる依存の枝を取り除いたグラフは、非循環グラフとなる。この時、算出されるクリティカルパスのコストは、ループのひとつのイタレーションをプロセッサ無限個で並列実行した際の実行時間の下限であり、ループ全体の並列実行時間の下限ではない。

### 2.2 展開されたループタスクグラフ

ループを展開したコードの依存グラフは、基本ブロックの依存グラフと同様であり、非循環グラフとなる。図 2は、この依存グラフとループタスクグラフの関係を図解したものである。

図2では、ループを展開したコードの依存グラフを、各イタレーションをひとつの層として、立体的に描いている。異なるイタレーションの同じ文に相当する節点は、それぞれの層で同じ位置に配置している。各層に現れるグラフは、ループタスクグラフから、ループによって運ばれる依存の枝を取り除いたものとなる。また、層と層の間にはループによって運ばれる依存の枝があり、その枝の始点と終点の高度差が依存距離に相当する。このグラフを展開されたループタスクグラフと呼ぶことにする。

展開されたループタスクグラフの真上から平行光線をあてた時に、展開されたループタスクグラフの真下に結ぶ像が、ループタスクグラフとなる。本稿では、ループタスクグラフの節点  $v$  に相当する、ループ展開後の節点  $vv_1, vv_2, \dots, vv_N$  を、 $v$  の投影元の節点<sup>1</sup>と呼ぶことにする。逆に、節点  $v$  を節点  $vv_1, vv_2, \dots, vv_N$  の投影先の節点と呼ぶことにする。また、枝、パスに対して同様の表現を用いる。

展開されたループタスクグラフのクリティカルパスのコストは、ループをプロセッサ無限個で並列実行した際の実行時間の下限を与える。

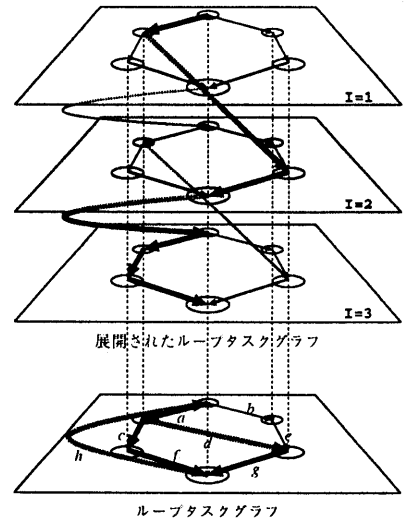


図 2: 展開されたループタスクグラフ

### 2.3 パスの影

図2は、展開されたループタスクグラフにおけるパス(図中太線)が、ループタスクグラフ上にどのように投影されるかを示している。この例のように、ループタスクグラフ上に投影された、展開されたループタスクグラフ上のパス  $p$  の像を、パス  $p$  の影と呼ぶことにする。

ループタスクグラフの枝にパスの影が何回重なるかを、その枝の影重複度と呼ぶことにする。例えば、図2のループタスクグラフの場合、枝  $a, b, c$  の影重複度はそれぞれ 2, 0, 1 である。

### 2.4 ループタスクグラフの前処理

問題の取り扱いを容易にするため、ループタスクグラフに次の前処理を加えるものとする。

まず、2つの仮想的な節点  $START, STOP$  を設ける。両者のコストはいずれも 0 とする。 $START$  から、ループタスクグラフのループによって運ばれる依存を削除した時に極小となる節点<sup>2</sup> に向けて、コスト 0、依存距離 0 の枝を張る。また、ループタスクグラフのループによって運ばれる依存を削除した時に極大となる節点<sup>3</sup> から、 $STOP$  に向けて、コスト 0、依存距離 0 の枝を張る。後に証明するが、この処理により、展開されたループタスクグラフにおけるクリティカルパスは、 $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とすることが保証され、問題の取り扱いが容易になる。

<sup>1</sup>一般的にはタスク  $u$  のインスタンスと呼ばれる。

<sup>2</sup>入る枝のない節点は極小であるという。

<sup>3</sup>出る枝のない節点は極大であるという。

## 3 整数計画問題への帰着

ループ全体のプロセッサ無限個による並列実行時間の下限は、展開されたループタスクグラフのクリティカルパスのコストにより与えられる。本節では、展開されたループタスクグラフのクリティカルパスのコスト算出の問題が、いかに整数計画問題へ帰着されるかについて述べる。

なお本稿では、記述を正確かつ簡潔にすべく、形式仕様記述言語  $Z$  の表記法を部分的に用いている。 $Z$  の詳細については文献[2]を参照されたい。また、付録付録Aに本稿で用いた  $Z$  の記号をまとめている。

### 3.1 表記と定義

本稿では、与えられたループタスクグラフを  $G = (V, E)$ 、 $G$  に対応する展開されたループタスクグラフを  $\tilde{G} = (\tilde{V}, \tilde{E})$  と表記する。 $V, E$  はそれぞれループタスクグラフの節点、枝の集合、 $\tilde{V}, \tilde{E}$  はそれぞれ展開されたループタスクグラフの節点、枝の集合である。その他、本稿で用いる記号の表記と定義を付録付録Bにまとめる。

本稿では、展開されたループタスクグラフ上のパスを、展開されたループタスクグラフの節点の列をもって定義する。また、パスのコストを、パスを構成する節点と枝のコストの総計と定義する。すなわち、パス  $p = \langle vv_1, vv_2, \dots, vv_n \rangle$  のコスト  $\Omega(p)$  は、形式的に式

(1) のように定義される。

$$\Omega(p) = \omega_{\tilde{V}}(vv_1) + \sum_{i=1}^{n-1} (\omega_{\tilde{E}}((vv_i, vv_{i+1})) + \omega_{\tilde{V}}(vv_{i+1})) \quad (1)$$

### 3.2 対象ループ

仮定 1 本稿で対象とするループは次の条件を満足するものとする。

1.  $\forall vv : \tilde{V} \bullet \omega_{\tilde{V}}(vv) = \omega_V(f_{\tilde{V}}(vv))$
2.  $\forall ee : \tilde{E} \bullet \omega_{\tilde{E}}(ee) = \omega_E(f_{\tilde{E}}(ee))$
3.  $\forall ee : \tilde{E} \bullet d_{\tilde{E}}(ee) = d_E(f_{\tilde{E}}(ee))$
4. ループの入れ子は一段。
5. ループの繰り返し数は定数  $N$ 。
6. ループボディ内に条件分岐がない。□

1-3は、展開されたループタスクグラフの任意の節点のコスト、枝のコストおよび依存距離は、ループタスクグラフ上の投影先の節点のコスト、枝のコストおよび依存距離に等しいことを意味している。

4の条件の緩和に関しては、検討において討議する。

### 3.3 パスの影の性質

クリティカルパスの影には次の性質がある。

性質 1 展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  のクリティカルパスは、ループタスクグラフ  $G = (V, E)$  の  $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とする。(証明略) □

本稿で問題としているのはクリティカルパスのコストの算出である。性質 1 により、我々は  $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とするパスのみを考慮の対象とすればよいことになる。

一方、仮定 1 の下では、一般のパスの影には以下の性質がある。

性質 2 展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  上に、 $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とする、パス  $p$  が与えられている。この時、枝  $e$  の測度  $\varphi_E(e)$  として枝  $e$  の影重複度を充てると、パス  $p$  のコスト  $\Omega(p)$  は式 (2) により算出することができる。(証明略) □

$$\Omega(p) = \sum_{(u,v) \in E} (\omega_E((u,v)) + \omega_V(v)) \varphi_E((u,v)) \quad (2)$$

性質 3 展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  上に、 $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とする、パス  $p$  が与えられている。この時、枝  $e$  の測度  $\varphi_E(e)$  として枝  $e$  の影重複度を充てると、次の関係式 (3)-(5) が成り立つ。(証明略) □

$$\sum_{e \in E} d_E(e) \varphi_E(e) < N \quad (3)$$

$$\begin{aligned} & \sum_{(START,v) \in E} \varphi_E((START,v)) \\ &= \sum_{(v,STOP) \in E} \varphi_E((v,STOP)) = 1 \end{aligned} \quad (4)$$

$$\sum_{(u,v) \in E} \varphi_E((u,v)) - \sum_{(v,u) \in E} \varphi_E((v,u)) = 0 \quad (5)$$

### 3.4 整数計画問題への帰着

ループタスクグラフ  $G = (V, E)$  の任意の枝  $e \in E$  には、測度  $\varphi_E(e)$  が定義される。本稿では、ループタスクグラフの全ての枝の測度への値の割り当ての方法を、測度定義と呼ぶことにする。

前小節では、展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  上に、 $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とするパスを描き、ループタスクグラフ  $G = (V, E)$  の任意の枝  $e \in E$  の測度  $\varphi_E(e)$  として、その時の枝  $e$  の影重複度を割り当てた場合、その測度定義は式 (3)-(5) を満足することを述べた。本小節では逆に、最初に式 (3)-(5) を満足するような測度定義が与えられた時、枝  $e$  の影重複度が測度  $\varphi_E(e)$  と等しくなるようなパスを、展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  上に描くことができるかどうかを考察してみる。

ここでループタスクグラフ  $G = (V, E)$  より、測度 0 の枝を除いた後に孤立節点を除き、さらに残っている任意の枝  $(u, v)$  に平行な  $(\varphi_E((u, v)) - 1)$  本の枝  $(u, v)$  を追加して得られるグラフ  $G'$  を考える。グラフ  $G'$  は次の性質を有する。

性質 4 グラフ  $G'$  の連結成分のうちひとつは必ず節点  $START$ 、 $STOP$  の両方を含む。(証明略) □

$START$ 、 $STOP$  を含む連結成分について考える。式 (5) より、グラフ  $G'$  の  $START$ 、 $STOP$  以外の節点の入次数と出次数は等しい。また式 (4) より、グラフ  $G'$  の  $START$  の出次数 (1) は入次数 (0) より 1 大きく、 $STOP$  の入次数 (1) は出次数 (0) より 1 大きい。したがって、 $START$ 、 $STOP$  を含む連結成分には  $START$  より  $STOP$  に至る Euler 路が存在する [4]。もし、グラフ  $G'$  が  $START$ 、 $STOP$  を含む連結成分のみでなる場合、次のアルゴリズム 1 により、式 (3)-(5) を満足するような測度定義より、ループタスクグラフ

$G = (V, E)$  の枝  $e \in E$  の影重複度が測度  $\varphi_E(e)$  と等しくなるようなパス  $p$  を、展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  上に確定できる。このパス  $p$  は、 $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とする。

#### アルゴリズム 1

1.  $i := 1$
2.  $\widehat{cur} := F_V(START)$  のうち最もイタレーション番号の若いもの
3.  $p := \langle \widehat{cur} \rangle$
4.  $cur :=$  Euler 路の  $i$  番目の節点
5.  $cur = STOP$  ならば、アルゴリズム終了。
6.  $v :=$  Euler 路の  $i + 1$  番目の節点,  $vv := (\widehat{cur}, vv) \in F_E((cur, v))$  なる  $\tilde{G}$  上の節点
7.  $p := p \hat{\cup} \langle vv \rangle$
8.  $cur := v, \widehat{cur} := vv, i := i + 1$
9. 4へ戻る。□

先に式 (3)–(5) を満足するような測度定義を与えた時、ループタスクグラフ  $G = (V, E)$  の任意の枝  $e \in E$  の影重複度が測度  $\varphi_E(e)$  と等しくなるようなパスは、一般に複数存在する。しかしながら、それらのパスのコストは、式 (2) により全て等しいことが保証される。

ここで、展開されたループタスクグラフ  $\tilde{G} = (\tilde{V}, \tilde{E})$  上の、 $START$  の投影元節点のひとつを始点とし、 $STOP$  の投影元節点のひとつを終点とするパスにより与えられる影重複度を測度とする時の、測度定義の集合を  $A$  とする。また、式 (3)–(5) を満足する測度定義の集合を  $B$  とする。性質 3 により  $A \subseteq B$  である。よって、クリティカルパスのコストを与える測度定義は、 $B$  の中から発見できるはずである。

以上の考察で、クリティカルパスのコストを求める問題は、式 (3)–(5) の規定する制約条件を満足する測度定義のうち、式 (2) の値を最大にし、かつグラフ  $G'$  が連結となるような（すなわち連結成分が唯一つとなるような）ものを探索する問題に帰着できる。全ての枝の測度の値は整数、式 (2) ならびに式 (3)–(5) は線形式であるので、問題は整数計画問題となる。

以下に整数計画問題の変数、目的関数、制約条件をまとめる。

変数 測度  $\varphi_E(e)$  ( $\forall e \in E$ )

目的関数 式 (2)

制約条件 式 (3), (4), (5)

### 3.5 アルゴリズムの詳細

前小節の整数計画問題を解くにあたり、本研究では単体法と分枝限定法を用いる。単体法で得られる解は一般に実数であり、単体法で得られた実数解近傍に存在する整数解を探索するために、本研究では分枝限定法を用いる。また、式 (3)–(5) を満足する測度定義のうち、パスを与えないもの（すなわちグラフ  $G'$  が非連結となるもの）を除くためにも分枝限定法を用いる。分枝限定法のアルゴリズムをアルゴリズム 2 に示す。単体法については、線形計画法に関する適当な文献を参照されたい。

分枝限定法では暫定解が定義されるが、本稿では暫定解を三つ組  $(C, \Omega, \varphi)$  と定義する。但し、 $C$  は制約条件の集合、 $\Omega, \varphi$  はそれぞれ、単体法によって得られた制約条件  $C$  の下での目的関数 (式 (2)) の最大値とその時の測度定義である。

#### アルゴリズム 2

1. 初期解  $(C_0, \Omega_0, \varphi_0)$  をリスト  $lis$  に挿入。但し、 $C_0$  は式 (3)–(5) のみからなる制約条件の集合である。
2. リスト  $lis$  より  $\Omega$  最大の暫定解  $(C_{cur}, \Omega_{cur}, \varphi_{cur})$  を取り出す。
3. 測度定義  $\varphi_{cur}$  の全ての測度が整数、かつ測度定義  $\varphi_{cur}$  の与えるグラフ  $G'$  が連結であるならば、アルゴリズム終了。
4. 測度定義  $\varphi_{cur}$  より整数でない測度  $val$  を持つ枝  $e$  を選択。
5.  $C_1 := C_{cur} \cup \{ \text{制約条件 } \varphi_E(e) \geq \lceil val \rceil \}$
6. 暫定解  $(C_1, \Omega_1, \varphi_1)$  をリスト  $lis$  に挿入。
7.  $C_2 := C_{cur} \cup \{ \text{制約条件 } \varphi_E(e) \leq \lfloor val \rfloor \}$
8. 暫定解  $(C_2, \Omega_2, \varphi_2)$  をリスト  $lis$  に挿入。
9. 2へ戻る。□

### 4 実装と評価

本研究では Livermore ベンチマークのカーネル 10 番（節点数 21, 枝数 46）とカーネル 13 番（節点数 19, 枝数 27）を用いて、計算時間の計測を行った。各文をひとつのタスクとし、その実行コストはアセンブラコードの行数とした。また、タスク間の通信コストは転送される変数の数とした。各ループのイタレーション数を 1000 から 10000 まで 1000 ずつ変化させ、計算時間の計測を行った。各計測は 100 回ずつ行い、計算時間はその平均を採った。

結果は図 3 に示す通りとなり、計算時間はループのイタレーション数に依存せず、提案手法がループの大きさに対し高いスケラビリティを有することがわかる。

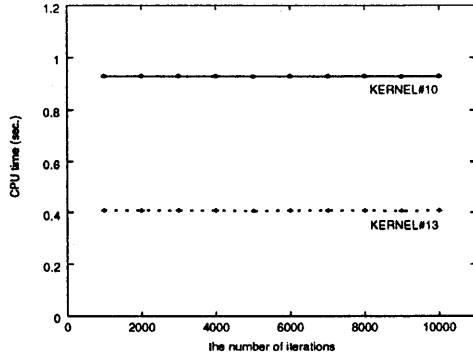


図 3: 計算時間

図 4 はカーネル 10 番のループタスクグラフとクリティカルパスの影(太線)である。節点の脇の数字はタスクの実行コストである。クリティカルパスは (START, 1, 2, 4, 6, 8, 10, 12, 14, 16, 4, 6, 8, ..., 12, 14, 16, 18, 19, STOP) となる。クリティカルパス長は繰り返し数 1000 回の場合 69032 となる。

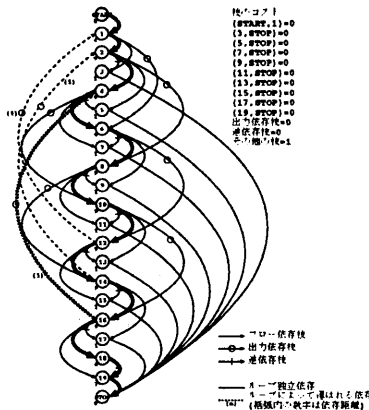


図 4: Livermore カーネル 10 番

## 5 まとめ

本稿では、ループを展開することなく、ループを展開したコードの依存グラフのクリティカルパスのコスト、すなわちプロセッサ無限個時のループの並列実行時間の下限を算出する手法を提案した。提案手法は、問題を整数計画問題に帰着させ、単体法と分枝限定法を適用する。計算量は最悪指数オーダーであるが、実装・評価の結果では十分実用的な時間で解が得られ

た。さらに計算時間はループのイテレーション数に依存せず、大規模なループに対しては本手法は極めて有効である。

クリティカルパスのコストは、コンパイル中に頻繁に計算されるケースが多い。本問題に特化した、整数計画問題を高速に解く専用アルゴリズムの開発が、今後の課題として挙げられる。

## 参考文献

- [1] Hans Zima and Barbara Chapman, *Supercompilers for Parallel and Vector Computers*, Addison Wesley, 1990.
- [2] J. M. Spivey, *The Z Notation A Reference Manual*, Prentice Hall, 1992.
- [3] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*, Prentice Hall, 1994.
- [4] 守屋 悦朗, 「コンピュータサイエンスのための離散数学」, サイエンス社, 1992.

## 付録 A 本稿で用いた Z の記法

$N$	0 以上の整数
$\{a, b, c\}$	集合の定義
$\{v : \text{type} \bullet \text{predicate}\}$	集合の一般的な定義
$\forall v : \text{type} \bullet \text{predicate}$	全称限量子
$PS$	集合 $S$ の巾集合
$a..b$	$a$ 以上 $b$ 以下の整数の集合
$\#S$	集合 $S$ の要素数
$S \setminus T$	集合 $S$ と集合 $T$ の差
$\text{seq } S$	集合 $S$ の元を要素とする列
$(a, b, c)$	列
$s_1 \ s_2$	列の連結
$X \rightarrow Y$	$X$ から $Y$ への全域関数
$X \leftrightarrow Y$	$X$ から $Y$ への部分関数
$X \twoheadrightarrow Y$	$X$ から $Y$ への全域全射

## 付録 B 記号の表記と定義

表記	定義 (型)
$N$	ループの繰り返し数 ( $N$ )
$\Omega(p)$	$G$ におけるパス $p$ のコスト ( $\text{seq } \tilde{V} \twoheadrightarrow N$ )
$\omega_V(v)$	$G$ の節点 $v$ のコスト ( $\tilde{V} \rightarrow N$ )
$\omega_E(e)$	$G$ の枝 $e$ のコスト ( $E \rightarrow N$ )
$d_E(e)$	$G$ の枝 $e$ の依存距離 ( $E \rightarrow 0..N-1$ )
$\varphi_E(e)$	$G$ の枝 $e$ の測度 ( $E \rightarrow 0..N$ )
$f_{\tilde{V}}(vv)$	$G$ の節点 $vv$ の $G$ における投影先の節点を返す関数 ( $\tilde{V} \rightarrow V$ )
$f_E(ee)$	$G$ の枝 $ee$ の $G$ における投影先の枝を返す関数 ( $E \rightarrow E$ )
$\omega_{\tilde{V}}(vv)$	$G$ の節点 $vv$ のコスト ( $\tilde{V} \rightarrow N$ )
$\omega_E(ee)$	$G$ の枝 $ee$ のコスト ( $E \rightarrow N$ )
$d_E(ee)$	$G$ の枝 $ee$ の依存距離 ( $E \rightarrow 0..N-1$ )
$F_V(v)$	$G$ の節点 $v$ の $G$ における投影元の節点の集合を返す関数 ( $V \rightarrow P \tilde{V}$ )
$F_E(e)$	$G$ の枝 $e$ の $G$ における投影元の枝の集合を返す関数 ( $V \rightarrow P E$ )