リスト上の最大マーク付け問題を解く並列プログラムの導出

松崎公紀[†1] 胡振江[†1] 武市正人[†1]

最大マーク付け問題とは，入力として与えられたデータに対し，ある述語を満たし，かつ，重み和が最大となる要素集合を求める問題である．多くの最適化問題は最大マーク付け問題として定式化できることが知られている．本稿では，このうち，データがリスト構造を持つ場合を対象とする．リスト上の最大マーク付け問題の例としては，最大部分列和問題などが知られている．本稿では，最大マーク付け問題における述語の仕様から並列プログラムを導出する手法を提案する．並列プログラムの導出においては，並列計算においてよく利用される処理を抽象化した並列スケルトンを利用する．並列スケルトンを用いた並列プログラムの導出法についてはこれまでにも多くの研究がなされており，それらを利用する．本稿ではさらに，述語の仕様から並列プログラムのコードを生成するシステムについても述べる．

Derivation of Parallel Programs for Maximum Marking Problems on Lists

Kiminori Matsuzaki,[†1] Zhenjiang Hu[†1] and Masato Takeichi[†1]

The maximum marking problem is the problem of marking the entries of some given data structure in such a way that a given constraint is satisfied and the sum of the values associated with marked entries is as large as possible. We can formalize many optimization problems as instances of the maximum marking problems. In this paper, we show the derivation of parallel programs for maximum marking problems on lists, which include the well-known maximum segment sum problem. The derivation of parallel programs is based on skeletal parallel programming, where parallel programs are developed by compositions of abstracted parallel computational patterns called skeletons. In this paper, we also discuss a code generator for the maximum marking problems.

1. はじめに

最大マーク付け問題[5),19)]とは，入力として与えられる再帰データ X 中の述語 p を満たす要素集合の中で重み和が最大のものを求めるという問題である．再帰データ X と述語 p を適切に与えることにより，様々な最適化問題が最大マーク付け問題として定義される．たとえば，Programming Pearls[3)] の 1 つとしても有名な最大部分列和問題や，条件付きナップサック問題などは，最大マーク付け問題の例である．

これまで，最大マーク付け問題一般に対する逐次プログラムの導出法が複数研究されている[4)–6),19)]．しかし，並列プログラムの導出に関しては，最大部分列和問題などの具体的な問題に対しては研究されているが[1),2),8),11),18)]，一般的な最大マーク付け問題を解く並列プログラムの導出法はほとんど与えられていなかった．

効率の良い並列プログラムを導出することは一般に難しい．なぜならば，独立に計算することのできる複数の部分を見つけることが必要であるからである．さらに，効率の良い並列プログラムでは，良い台数効果（スケーラビリティ）を示し，並列化によるオーバヘッドが小さくなければならない．

本稿では，スケルトン並列プログラミング[7)] の手法を利用して，リスト上の最大マーク付け問題に対する並列プログラムの一般的な導出法を提案する．スケルトン並列プログラミングは，並列計算においてよく利用される計算パターンを抽象化した並列スケルトンを組み合わせることで並列プログラムを作成する手法である．これまでに，スケルトン並列プログラムの導出に関して多くの研究があり，本稿ではそれらの成果を利用する．

本研究の貢献は大きく次の 2 点である．

- リスト上の最大マーク付け問題を解くスケルトン並列プログラミングの導出法を提案する．

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

- 並列プログラムを導出するシステムを構築し，得られた並列プログラムの効率を実験によって確かめる．

本稿の構成は以下のとおりである．2 章では，基本的な表記法およびリスト上の並列スケルトンを導入する．3 章では，リスト上の最大マーク付け問題を定式化し，篠埜らによる逐次プログラムの導出法を示す．4 章では，篠埜らの手法によって得られる逐次プログラムからスケルトン並列プログラムを導出する．5 章では，並列プログラムを導出するシステム，特に，得られるプログラムの最適化について述べ，実験結果の報告する．最後に，関連研究について 6 章で述べ，まとめと今後の課題について 7 章で述べる．

2. スケルトン並列プログラミング

2.1 表 記 法

本稿では，関数型言語 Haskell に似た表記を利用する．本稿での関数の定義は，関数適用が空白で表現され，また，関数がカリー化されていることを除けば，数学的な関数の定義と同様に読むことができる．

2.1.1 関数，演算子，型

関数適用は空白によって表現し，引数には括弧をつけない．たとえば，$f\ a$ は $f(a)$ を表す．関数適用はカリー化されて左結合である．すなわち，$f\ a\ b$ は $(f\ a)\ b$ である．関数適用は結合順位が最も高いとする．

一般的な二項演算子として，$\oplus$，$\otimes$，$\ominus$ などを用いる．二項演算子はセクション化によって関数として扱い，$a \oplus b = (a\oplus)\ b = (\oplus)\ a\ b$ である．標準的な演算子のほかに，以下に示す演算子を用いる．演算子 $\uparrow$ は 2 つの入力のうち大きいものを返す．演算子 $\uparrow_f$ は，2 つの入力のうち関数 f の結果の大きいものを返す．演算子 $\circ$ は関数合成を表す．

真偽値の型を Bool（Bool $= \{\mathsf{T}, \mathsf{F}\}$）とし，数値の型を Num とする．

2.1.2 組

固定個の値をまとめたものを組と呼ぶ．組の要素は異なる型を持ってもよい．組は，たとえば $(1, 0.1)$ のように表記する．同じ型 D の要素 k 個からなる組に対しては，その型を D^k とも書く．

組の第 1 要素を返す関数を *fst*，組の第 2 要素を返す関数を *snd* とする．演算子 $(,)$ は，2 つの値を受け取りそれらの組を返す．演算子 $\triangle$ は複数の関数をまとめ，組を返す新しい関数を作る．たとえば，$(f \triangle g)\ x = (f\ x, g\ x)$ である．

2.1.3 リ ス ト

同じ型の要素からなる有限の列をリストと呼ぶ．リストは，空リストであるか，あるリストの先頭に要素を追加することで構成される．空リストは $[\,]$ で表現する．また，リスト x の先頭に要素 a を追加することを $(a : x)$ と表現する．また，型が α である要素から構成されるリストの型を $[\alpha]$ と表現する．リストの略記法として，$[1, 5, 2, 3]$ のような表記を用いる．

関数 *head* は空でないリストの先頭要素を返す．リストを操作する関数は，主に，パターンマッチ，もしくは，リストの内包表記によって定義される．

リストを操作する関数のうち，リスト準同型と呼ばれる特別な形を持つものを以下のように定義する．

定義 1（リスト準同型） 関数 f がある関数 g とある値 z によって

$$f\ [\,] = z$$
$$f\ (a : x) = g\ a\ (f\ x)$$

と定義されるとき，$f = (\![g, z]\!)$ と表記し，関数 f はリスト準同型であると呼ぶ． □

演算子 $\sum_\oplus$ は，入力のリストを演算 $\oplus$ によって畳み込む計算を行うもので，$\sum_\oplus [a_1, \ldots, a_n] = a_1 \oplus \cdots \oplus a_n$ と定義される．演算 $\oplus$ が単位元 $\iota_\oplus$ を持つならば，$\sum_\oplus = (\![(\oplus), \iota_\oplus]\!)$ である．

2.2 リスト上の並列スケルトン

並列スケルトン[7] とは，並列計算に頻出する計算パターンを抽象化したものである．本稿では，リストを操作する 5 つの並列スケルトン[20] を使用する．**図 1** に並列スケルトンの定義を示す．以下，各並列スケルトンの直観的な意味を例をあげて説明する．

並列スケルトン map $f\ x$ は，関数 f をリスト x の

```
map :: (α → β) → [α] → [β]
map f (a : x) = f a : map f x
map f [] = []
zipw :: (α → β → γ) → [α] → [β] → [γ]
zipw f (a : x) (b : y) = f a b : zipw f x y
zipw f [] [] = []
scanl :: (β → α → β) → β → [α] → ([β], β)
scanl f c [] = ([], c)
scanl f c (a : x) = let (x', c') = scanl f (f c a) x
                    in (c : x', c')
scanr :: (β → α → β) → β → [α] → ([β], β)
scanr f c [] = ([], c)
scanr f c (a : x) = let (x', c') = scanr f c x
                    in (c' : x', f c' a)
shift≫ :: α → [α] → [α]
shift≫ c [] = []
shift≫ c (a : x) = c : shift≫ a x
```

図 1 リスト上の並列スケルトンの定義
Fig. 1 Definition of parallel skeletons on lists.

すべての要素に適用する．たとえば，リストのすべての要素を 2 倍する計算は，

map $(2\times)$ $[1,5,2,3] = [2,10,4,6]$

となる．並列スケルトン zipw $f\ x\ y$ は，関数 f を使って，2 つの同じ長さのリスト x と y の対応する要素をまとめる処理を行う．たとえば，

zipw $(-)$ $[1,5,2,3]$ $[4,2,1,3] = [-3,3,1,0]$

である．並列スケルトン scanl $f\ c\ x$ は，蓄積変数 c を初期値としてリスト x の左から関数 f を使って畳み込んでいった途中の結果のリストおよび最終の結果の組を返す．たとえば，

scanl $(-)$ 2 $[1,5,2,3] = ([2,1,-4,-6],-9)$

である．並列スケルトン scanr $f\ c\ x$ は，逆に，蓄積変数 c を初期値としてリスト x の右から関数 f を使って畳み込んでいった途中の結果のリストおよび最終の結果の組を返す．たとえば，

scanr $(-)$ 2 $[1,5,2,3] = ([-8,-3,-1,2],-9)$

である．並列スケルトン $\mathsf{shift}_{\gg}\ c\ x$ は，リスト x の要素を 1 つ右にずらし，左端要素として c を代入したリストを返す[⋆1]．右端の要素は捨てられる．たとえば，

$\mathsf{shift}_{\gg}$ 2 $[1,5,2,3] = [2,1,5,2]$

である．

並列スケルトン scanl および scanr を効率良く並列に計算するためには，引数の関数 f に条件が必要である．本稿では，以下のとおり定義される準結合性を用いて条件を定式化する．

定義 2（準結合性） 二項演算子 $\ominus$ が準結合的であるとは，任意の a，b，c に対して

$$(a \ominus b) \ominus c = a \ominus (b \oplus c)$$

が成立するような結合的な演算子 $\oplus$ が存在することと定義する．この $\oplus$ を $\ominus$ の補演算子と呼ぶ． □

定義より，結合的な演算子 $\otimes$ は，$\otimes$ 自身を補演算子として準結合的である．

次に示す補題は，準結合的な演算子が存在すれば並列スケルトン scanl および scanr を並列に実装できることを示す．

補題 1 関数 f に対して，等式

$$f\ c\ a = c \ominus g\ a$$

を満たす関数 g，準結合的な演算子 $\ominus$ が存在するならば，関数 f を引数に取る並列スケルトン scanl および scanr を並列実装することができる． □

並列スケルトンの並列時間計算量について簡単に述べる．並列計算モデルとしては，共有メモリの EREW PRAM を考える．リストの長さを N，プロセッサ数を P，関数 f の逐次での計算時間を t_f とする．このとき，map f および zipw f は，$t_f N/P$ の時間で並列に計算することができる．また，scanl f および scanr f は，条件の補助関数を g，準結合的な演算子を $\ominus$，$\ominus$ の補演算子を $\oplus$ として，$(t_g + t_\oplus + t_f)N/P + (t_\oplus + t_\ominus)\log P$ の時間で並列に計算することができる．リストの長さと比較してプロセッサ数が十分に小さい場合において，scanl と scanr の並列計算のオーバヘッドは $(1 + (t_g + t_\oplus)/t_f)$ となる．なお，これらの 5 つの並列スケルトンは，分散メモリ計算モデルにおいても効率の良い実装が可能である．

2.3 準結合的な演算子の導出のための十分条件

前節で述べたように，並列スケルトン scanl および scanr は，それらの効率の良い並列実装のために準結合的な演算子を必要とする．本節では，そのような準結合的な演算子を導出するための十分条件を 2 つあげる[14),16)]．

まず，引数として与えられる関数の定義域の有限性に着目する．引数として与えられる関数の定義域が有限である場合には，その定義域のすべての要素が入力とした場合を列挙することで，準結合的な演算子を導出することができる．

補題 2（定義域の有限性） 集合 $\mathcal{D}$ が l 個の有限要素からなるとし，$\mathcal{D} = \{d_1, \ldots, d_l\}$ とする．ある型 $\mathcal{A}$ に対し，型 $\mathcal{D} \to \mathcal{A} \to \mathcal{D}$ を持つ関数 f を考える．このとき，$f\ c\ a = c \ominus g\ a$ を満たす準結合的な演算子 $\ominus$ と関数 g が存在する．

証明 上記の等式を満たす，準結合的な演算子 $\ominus$ とその補演算子 $\oplus$，関数 g は次のように与えられる．

$$g\ a = (f\ d_1\ a, \ldots, f\ d_l\ a)$$

$$c \ominus (e_1, \ldots, e_l) = \mathbf{case}\ c\ \mathbf{of}\ d_1 \to e_1; \cdots; d_l \to e_l$$

$$(e_1, \ldots, e_l) \oplus (e'_1, \ldots, e'_l) = (e''_1, \ldots, e''_l)$$

$$\mathbf{where}\quad e''_i = \mathbf{case}\ e_i\ \mathbf{of}\ d_1 \to e'_1; \cdots; d_l \to e'_l$$ □

次に，関数が組の値を計算し，その計算において環の 2 つの演算子が使用されている場合を考える．

定義 3（環の上で線形な関数） 代数 $\{\mathcal{D}, \oplus, \otimes\}$ を環とし，k を定数とする．型 $\mathcal{D}^k \to \mathcal{D}$ を持つ関数 f が適当な定数 $a_1, \ldots, a_k$ を用いて

$$f\ (d_1, \ldots, d_k) = (d_1 \otimes a_1) \oplus \cdots \oplus (d_k \otimes a_k)$$

と定義できるとき，関数 f は環 $\{\mathcal{D}, \oplus, \otimes\}$ の上で線形な関数であるという． □

組を計算する関数が，環の上で線形な関数の組に

⋆1 並列スケルトン $\mathsf{shift}_{\gg}$ は scanl を使って実現することもできるが，分かりやすさと効率のために導入する．

よって記述されるとき，その関数に対して準結合的な演算子を導出することができる．この導出には，行列積の結合性を利用する．

補題 3（環上の組） 代数 $\{\mathcal{D}, \oplus, \otimes\}$ を環とし，k を定数とする．型 $\mathcal{D}^k \to \mathcal{A} \to \mathcal{D}^k$ を持つ関数 f に対して，$f\ c\ a = (f_1\ a \mathbin{\triangle} \cdots \mathbin{\triangle} f_k\ a)\ c$ として，すべての $f_j\ a\ (j \in [1,k])$ が環 $\{\mathcal{D}, \oplus, \otimes\}$ の上で線型であるとする．このとき，$f\ c\ a = c \ominus g\ a$ を満たす準結合的な演算子 $\ominus$ と関数 g が存在する．

証明 上記の等式を満たす，準結合的な演算子 $\ominus$ とその補演算子 $\odot$，関数 g は，下記のように与えられる．ただし，$\times$ は環 $\{\mathcal{D}, \oplus, \otimes\}$ の上で定義される行列積とする．また，$\{a_{ij}\}$ は線型な関数 $f_j\ a$ の係数を縦方向に並べたものである．

$$
\begin{aligned}
&g\ a = \{a_{ij}\} \\
&c \ominus M = c \times M \\
&M \odot M' = M \times M'
\end{aligned}
$$

□

3. 最大マーク付け問題を解く逐次プログラム

本章では，本研究の対象であるリスト上の最大マーク付け問題を定式化し，それに対する篠埜らによる逐次プログラムの導出法を紹介する．

定義 4（最大マーク付け問題） 型が [Bool]→Bool である述語 p と型が [Num] である重みのリスト x が与えられる．このとき，リスト上の最大マーク付け問題 $mmp\ p\ x$ とは，p を満たすような長さが x と等しい型 [Bool] のリストのうち，T の要素に対応する重みの和が最大となるものを 1 つ返す問題である． □

述語 p を与えることにより，多くの最適化問題をリスト上の最大マーク付け問題として定式化することができる．例として，最終要素を含む連続した部分列のうち重み和が最大となるものを求める最大末尾部分列和問題 mts を考える．この問題は，以下のように定義される述語 p_{mts} を用いて最大マーク付け問題として定義できる．関数 all は型 [Bool] のリストのすべての要素が T であるかを判定する関数であり，関数 $last$ は F が 1 つ以上連続した後に T がリストの末尾まで連続しているかどうかを判定する．

$$
\begin{aligned}
&p_{mts}\ x = all\ x \vee last\ x \\
&\quad \textbf{where}\ all\ [\,] = \mathsf{T} \\
&\qquad\qquad all\ (a : x) = a \wedge all\ x \\
&\qquad\qquad last\ [\,] = \mathsf{F} \\
&\qquad\qquad last\ (a : x) = \neg a \wedge (all\ x \vee last\ x)
\end{aligned}
$$

右辺で呼び出す関数について閉じている（複数の）相互再帰関数は，組化変換[12] によって準同型にすることができる．たとえば，上記の p_{mts} は次のようにも定義できる．

$$
\begin{aligned}
&p_{mts} = accept_{mts} \circ (\!|g, z|\!) \\
&\quad \textbf{where}\ accept_{mts}\ (a, l) = a \vee l \\
&\qquad\qquad z = (\mathsf{T}, \mathsf{F}) \\
&\qquad\qquad g\ a\ (a', l') = (a \wedge a', \neg a \wedge (a' \vee l'))
\end{aligned}
$$

本稿では，最大マーク付け問題のうち，p_{mts} のように型 Bool の組を返すリスト準同型 $(\!|g, z|\!) :: [\mathsf{Bool}] \to \mathsf{Bool}^k$ を使って述語が $accept \circ (\!|g, z|\!)$ と記述することができるものを対象とする．

篠埜らは，一般的な再帰構造の上での最大マーク付け問題に対して，逐次プログラムの導出法を示した[19),21)]．導出の基本的なアイディアは次の 2 点である．まず，述語が型 Bool の組を返す準同型で書けることより，最大マーク付け問題を解くプログラムの計算は，重み和 w，状態 s，マークを付けたデータ構造 x' の 3 つ組を返す準同型として定義できる．次に，最適性の原理により，同じ状態を持つ 3 つ組のうち最大の重み和を持つものだけを求めればよい．

篠埜らによる手法によって導出されるリスト上の最大マーク付け問題を解く逐次プログラムを**図 2** に示す．図 2 のプログラムにおいて，新しく定義されるリスト準同型 $(\!|g', z'|\!)$ は 3 つ組を計算するものであり，関数 $eachmax$ は状態ごとに最大の重み和を持つものを取り出す計算を行う関数である．状態の数が定数であるため，得られたプログラムはリストの長さに線形な時間で計算を行うことができる．

4. 最大マーク付け問題を解く並列プログラム

本章では，篠埜らによる最大マーク付け問題に対する逐次プログラムをもとに，スケルトン並列プログラムを導出する．この並列プログラムの導出は，以下の 3 ステップからなる．

(1) 逐次プログラムを並列スケルトンを利用しやすい形に変形する．
(2) 逐次プログラムの計算を並列スケルトンの組合せで置き換える．
(3) 並列スケルトンが要求する準結合的な演算子を導出する．

4.1 逐次プログラムの変形

図 2 に示した逐次プログラムをそのまま効率良く並列化するのは難しい．並列化を困難にする要因は次の 2 点である．

- 状態（計算の途中の 3 つ組の第 1 要素）が型 Bool^k

$$
\begin{aligned}
&mmp\ (accept \circ (\![z, g]\!))\ x \\
&\quad = snd\ (\textstyle\sum_{\uparrow_{fst}} [(w, x') \mid (s, w, x') \leftarrow (\![z', g']\!)\ x, accept\ s]) \\
&\qquad \textbf{where}\ z' = [(z, 0, [\,])] \\
&\qquad\quad g'\ b\ y = eachmax\ [(g\ m\ s', w + w', m : y') \\
&\qquad\qquad\qquad\qquad \mid (m, w) \leftarrow [(\mathsf{T}, b), (\mathsf{F}, 0)], (s', w', y') \leftarrow y] \\
&\qquad\quad eachmax\ [\,] = [\,] \\
&\qquad\quad eachmax\ ((s, w, y) : xs) = eachmax'\ (s, w, y)\ (eachmax\ xs) \\
&\qquad\quad eachmax'\ (s, w, y)\ [\,] = [(s, w, y)] \\
&\qquad\quad eachmax'\ (s, w, y)\ ((s', w', y') : ys) = \textbf{if}\ (s == s')\ \textbf{then if}\ w > w'\ \textbf{then}\ (s, w, y) : ys \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ (s', w', y') : ys \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ (s', w', y') : eachmax'\ (s, w, y)\ ys
\end{aligned}
$$

図 2 篠埜らの導出によって得られる逐次プログラム
Fig. 2 Sequential program derived by the method by Sasano, et al.

$$
\begin{aligned}
&mmp\ (accept \circ (\![g, z]\!))\ x \\
&= \textbf{let}\ (x', (w'_0, \ldots, w'_{2^k-1})) = firststep\ x \\
&\qquad (w'_c, c) = \textstyle\sum_{\uparrow_{fst}} [(w_i, i) \mid i \in [0, 2^k), \overline{accept}\ i] \\
&\quad \textbf{in}\ secondstep\ (w_c, c)\ x' \\
&\textbf{where} \\
&\quad firststep\ [\,] = ([\,], (w'_0, \ldots, w'_{2^k-1}))\ \textbf{where}\ w'_j = \begin{cases} 0 & \textbf{if}\ j = \overline{z} \\ -\infty & \textbf{if}\ j \neq \overline{z} \end{cases} \\
&\quad firststep\ (a : x) = \textbf{let}\ (x', (w'_0, \ldots, w'_{2^k-1})) = firststep\ x \\
&\qquad\qquad\qquad \textbf{in}\ ((a, (w'_0, \ldots, w'_{2^k-1})) : x', (w''_0, \ldots, w''_{2^k-1})) \\
&\qquad\qquad\qquad \textbf{where}\ w''_j = \textstyle\sum_{\uparrow} [w'_i + q_{ij}\ a \mid i \in [0, 2^k)] \\
&\quad q_{ij}\ a = \begin{cases} a \uparrow 0 & \textbf{if}\ \overline{g}\ \mathsf{T}\ j = i \wedge \overline{g}\ \mathsf{F}\ j = i \\ a & \textbf{if}\ \overline{g}\ \mathsf{T}\ j = i \wedge \overline{g}\ \mathsf{F}\ j \neq i \\ 0 & \textbf{if}\ \overline{g}\ \mathsf{T}\ j \neq i \wedge \overline{g}\ \mathsf{F}\ j = i \\ -\infty & \textbf{if}\ \overline{g}\ \mathsf{T}\ j \neq i \wedge \overline{g}\ \mathsf{F}\ j \neq i \end{cases} \\
&\quad secondstep\ (w, c)\ [\,] = [\,] \\
&\quad secondstep\ (w, c)\ ((a, (w_0, \ldots, w_{2^k-1})) : x) \\
&\quad = \textbf{let}\ (m, w', c') = head\ [(m, w_i, i) \mid (m, a') \leftarrow [(\mathsf{T}, a), (\mathsf{F}, 0)], i \in [0, 2^k), w_i + a' = w \wedge \overline{g}\ m\ i = c] \\
&\qquad \textbf{in}\ m : secondstep\ (w', c')\ x
\end{aligned}
$$

図 3 状態を整数によって表現した 2 パスの逐次プログラム
Fig. 3 Two-pass sequential program that uses integers for states.

の組で表現されており，その値によって計算する関数 *eachmax* が存在すること．

- マーク付けの結果のリストを，重み和の計算と同時に生成していること．

これらの要因に対処するため，図 2 のプログラムを変形する．

まず，状態の表現を Bool^k 型の組から $[0, 2^k)$ の範囲の整数に変更する．組から整数への変換を行う関数を $bm2int_k$，その逆関数を $int2bm_k$ とする．組から整数への変換は 1 対 1 であれば任意のものが使えるが，ここでは組を二進数として読んだ数を使うことにする．たとえば，

$bm2int_2\ (\mathsf{T}, \mathsf{F}) = 2$

$int2bm_2\ 1 = (\mathsf{F}, \mathsf{T})$

である．

最大マーク付け問題の述語 $accept \circ (\![g, z]\!)$ に対して，状態を整数で表現した次の関数 $\overline{accept}$，$\overline{g}$，および，値 $\overline{z}$ を定義する．

$\overline{accept}\ s = accept\ (int2bm_k\ s)$

$\overline{g}\ b\ y = bm2int_k\ (g\ b\ (int2bm_k\ y))$

$\overline{z} = bm2int_k\ z$

これらを利用することにより，各状態に対応する最大の重み和を並べて表現することができる．

次に，動的計画法でよく利用されるテクニックである，2 パスのアルゴリズムへと変形する．まず，1 パス目では，リストの各要素に対して，各状態に対する最大の重み和の途中結果を保存する．次に，2 パス目では，最大の重み和を生成するようなマーク付けを，1 パス目で作成した表を逆順にたどることで求める．

図 3 に，これらの変形の結果のプログラムを示す．関数 *firststep* は 1 パス目の計算を行い，もとの値 a と各状態に対応する最大の重み和 $(w'_0, \ldots, w'_{2^k-1})$ の

組からなるリストを求める．関数 *secondstep* は，2 パス目の計算を行い，最終的に最大の重み和を与える途中状態 p' とマーク m をリストの先頭から順次求める．

4.2 並列スケルトンによる置き換え

図 3 で得られた 2 パスの逐次プログラム中の再帰関数を並列スケルトンによって実現する．

まず，1 パス目（*firststep*）について，x' と w'_i の計算は scanr によって実現できる．したがって，関数 *firststep* は，並列スケルトン scanr によって x' と w'_i を求めた後で，並列スケルトン zipw を使うことで実現できる．

$$
\begin{aligned}
&\mathit{firstsep}\ x\\
&= \mathbf{let}\ (x', (w'_0, \ldots, w'_{2^k-1})) = \mathsf{scanr}\ f_1\ z_1\ x\\
&\quad \mathbf{in}\ (\mathsf{zipw}\ (,)\ x\ x', (w'_0, \ldots, w'_{2^k-1}))\\
&\mathbf{where}\\
&\quad z_1 = (w'_0, \ldots, w'_{2^k-1})\\
&\qquad \mathbf{where}\ w'_j = \begin{cases} 0 & \mathbf{if}\ j = \overline{z} \\ -\infty & \mathbf{if}\ j \neq \overline{z} \end{cases}\\
&\quad f_1\ (w_0, \ldots, w_{2^k-1})\ a = (w'_0, \ldots, w'_{2^k-1})\\
&\qquad \mathbf{where}\ w'_j = \textstyle\sum_{\uparrow}[w_i + q_{ij}\ a \mid i \in [0, 2^k)]
\end{aligned}
$$

次に，2 パス目（*secondstep*）を並列スケルトンによって実現する．図 3 のプログラム中では，最大の重み和を実現するような状態とその重み和の両方を蓄積変数として渡している．実際には，状態だけを蓄積変数として使い，重み和については表を参照することで計算することができる．これを並列スケルトンを用いて実現するためには，1 つ左の要素における重み和があれば十分である．そこで，要素を 1 つ右にずらす並列スケルトン $\mathsf{shift}_{\gg}$ を使い，左の要素における各状態に対する最大の重み和のリスト x' を作る．

$$x' = \mathsf{shift}_{\gg}\ (w'_0, \ldots, w'_{2^k-1})\ (\mathsf{map}\ \mathit{snd}\ x)$$

このリスト x' ともとのリストをまとめたうえで，2 パス目の計算は，最大の重み和の与える状態を求める scanl と状態と元の値からマークを決定する zipw によって実現することができる．以下のプログラムにおいて，$(w'_0, \ldots, w'_{2^k-1})$ は，1 パス目において求めた各状態に対する最大の重み和である．また，関数 f_2，f_3 は同じ順序でリストを生成しなければならないことに注意が必要である．

$$
\begin{aligned}
&\mathit{secondstep}\ (w_c, c)\ x\\
&= \mathbf{let}\ x' = \mathsf{shift}_{\gg}\ (w'_0, \ldots, w'_{2^k-1})\ (\mathsf{map}\ \mathit{snd}\ x)\\
&\qquad x'' = \mathsf{zipw}\ (,)\ x\ x'\\
&\qquad (y, \_) = \mathsf{scanl}\ f_2\ c\ x''\\
&\quad \mathbf{in}\ \mathsf{zipw}\ f_3\ y\ x''\\
&\mathbf{where}\\
&\quad f_2\ c\ ((a, (w_0, \ldots, w_{2^k-1})), (w^l_0, \ldots, w^l_{2^k-1}))\\
&\quad = \mathit{head}\ [i \mid (m, a') \leftarrow [(\mathsf{T}, a), (\mathsf{F}, 0)],\\
&\qquad\qquad i \in [0, 2^k),\\
&\qquad\qquad w_i + a' = w^l_c \wedge \overline{g}\ m\ i = c]\\
&\quad f_3\ c\ ((a, (w_0, \ldots, w_{2^k-1})), (w^l_0, \ldots, w^l_{2^k-1}))\\
&\quad = \mathit{head}\ [m \mid (m, a') \leftarrow [(\mathsf{T}, a), (\mathsf{F}, 0)],\\
&\qquad\qquad i \in [0, 2^k),\\
&\qquad\qquad w_i + a' = w^l_c \wedge \overline{g}\ m\ i = c]
\end{aligned}
$$

並列スケルトンを用いた実装では $\mathsf{shift}_{\gg}$ や zipw による冗長な中間データ構造が生成されるが，これらのスケルトン並列プログラムは Emoto らによる最適化[9] などによって効率化することができる．

4.3 準結合的な演算子の導出

最後に，並列スケルトン scanl および scanr の引数の関数に対して準結合的な演算子を導出する．

まず，1 パス目の計算の中で使われている scanr について考える．並列スケルトン scanr の引数の関数 f_1

$$
\begin{aligned}
&f_1\ (w_0, \ldots, w_{2^k-1})\ a = (w'_0, \ldots, w'_{2^k-1})\\
&\quad \mathbf{where}\ w'_j = \textstyle\sum_{\uparrow}[w_i + q_{ij}\ a \mid i \in [0, 2^k)]
\end{aligned}
$$

は可換環 $\{\mathsf{Num}, \uparrow, +\}$ の上で線形な式によって計算される．したがって，補題 3 によって並列計算のための準結合的な演算子を導出することができる．特に，補題 3 における行列 $\{a_{ij}\}$ は，上記の $q_{ij}\ a$ を計算して得られる値を並べたものとなる．

次に，2 パス目の計算の中で使われる scanl について考える．並列スケルトン scanl の引数の関数 f_2

$$
\begin{aligned}
&f_2\ p\ ((a, (w_0, \ldots, w_{2^k-1})), (w^l_0, \ldots, w^l_{2^k-1}))\\
&= \mathit{head}\ [i \mid (m, a') \leftarrow [(\mathsf{T}, a), (\mathsf{F}, 0)],\\
&\qquad\qquad i \in [0, 2^k),\\
&\qquad\qquad w_i + a' = w^l_p \wedge \overline{g}\ m\ i = p]
\end{aligned}
$$

は定義は複雑であるが，蓄積変数 p は最大の重み和を与える状態であるので $[0, 2^k)$ の整数しかとらない．したがって，補題 2 によって並列計算のための準結合的な演算子を導出することができる．

4.4 並列化のまとめ

以上の導出をまとめることにより，最大マーク付け問題を解くスケルトン並列プログラムを得ることができる．導出されたスケルトン並列プログラムを**図 4** に示す．この並列プログラムにおけるパラメータ関数に具体的な定義を与えることで並列プログラムを得ることができる．ただし，図 4 のプログラムは，1 パス目の zipw および 2 パス目の map，$\mathsf{shift}_{\gg}$ および zipw の順序を入れかえ融合することにより簡略化してある．

$$
\begin{array}{l}
mmp\ (accept \circ ([g,z]))\ x \\
= \textbf{let}\ (x', (w'_0, \ldots, w'_{2^k-1})) = \mathsf{scanr}\ f_1\ z_1\ x \\
\qquad x'' = \mathsf{zipw}\ (,)\ (\mathsf{zipw}\ (,)\ x\ x')\ (\mathsf{shift}_{\gg}\ (w'_0, \ldots, w'_{2^k-1})\ x') \\
\qquad c = snd\ (\textstyle\sum_{\uparrow_{fst}} [(w_i, i) \mid i \in [0, 2^k), \overline{accept}\ i]) \\
\qquad (y, \_) = \mathsf{scanl}\ f_2\ c\ x'' \\
\quad \textbf{in}\ \mathsf{zipw}\ f_3\ y\ x'' \\
\textbf{where} \\
\quad z_1 = (w_0, \ldots, w_{2^k-1})\ \textbf{where}\ w_j = \begin{cases} 0 & \textbf{if}\ j = \overline{z} \\ -\infty & \textbf{if}\ j \neq \overline{z} \end{cases} \\
\quad f_1\ (w_0, \ldots, w_{2^k-1})\ a = (w'_0, \ldots, w'_{2^k-1})\ \textbf{where}\ w'_j = \textstyle\sum_{\uparrow} [w_i + q_{ij}\ a \mid i \in [0, 2^k)] \\
\quad g_1\ a = \{q_{ij}\ a\} \\
\quad \ominus_1 = \otimes_1 = \times \\
\quad q_{ij}\ a = \begin{cases} a \uparrow 0 & \textbf{if}\ \overline{g}\ \mathsf{T}\ j = i \wedge \overline{g}\ \mathsf{F}\ j = i \\ a & \textbf{if}\ \overline{g}\ \mathsf{T}\ j = i \wedge \overline{g}\ \mathsf{F}\ j \neq i \\ 0 & \textbf{if}\ \overline{g}\ \mathsf{T}\ j \neq i \wedge \overline{g}\ \mathsf{F}\ j = i \\ -\infty & \textbf{if}\ \overline{g}\ \mathsf{T}\ j \neq i \wedge \overline{g}\ \mathsf{F}\ j \neq i \end{cases} \\
\quad f_2\ c\ ((a, (w_0, \ldots, w_{2^k-1})), (w^l_0, \ldots, w^l_{2^k-1})) \\
\quad = head\ [i \mid (m, a') \leftarrow [(\mathsf{T}, a), (\mathsf{F}, 0)], i \in [0, 2^k), w_i + a' = w^l_c \wedge \overline{g}\ m\ i = c] \\
\quad g_2\ a = (f_2\ 0\ a, \ldots, f_2\ (2^k - 1)\ a) \\
\quad c \ominus_2 (d_0, \ldots, d_{2^k-1}) = \textbf{case}\ c\ \textbf{of}\ 0 \to d_0; \cdots; 2^k - 1 \to d_{2^k-1} \\
\quad (d_0, \ldots, d_{2^k-1}) \oplus_2 (d'_0, \ldots, d'_{2^k-1}) = (d''_0, \ldots, d''_{2^k-1}) \\
\qquad \textbf{where}\ d''_i = \textbf{case}\ d_i\ \textbf{of}\ 0 \to d'_0; \cdots; 2^k - 1 \to d_{2^k-1} \\
\quad f_3\ c\ ((a, (w_0, \ldots, w_{2^k-1})), (w^l_0, \ldots, w^l_{2^k-1})) \\
\quad = head\ [m \mid (m, a') \leftarrow [(\mathsf{T}, a), (\mathsf{F}, 0)], i \in [0, 2^k), w_i + a' = w^l_c \wedge \overline{g}\ m\ i = c]
\end{array}
$$

図 4 最大マーク付け問題を解くスケルトン並列プログラム．演算子 $\ominus_i$ は演算子 $\oplus_i$ を補演算子とする準結合的な演算子であり，$f_i\ c\ a = c \ominus_i g_i\ a$ を満たす（$i = 1, 2$）

Fig. 4 Skeletal parallel program for the maximum marking problems. For $i = 1, 2$, operator $\ominus_i$ is a semi-associative operator with its associative complement being $\oplus_i$ and equation $f_i\ c\ a = c \ominus_i g_i\ a$ holds.

本章の議論を以下の定理としてまとめる．

定理 1 リスト上の最大マーク付け問題 $mmp\ p$ において，述語 p が $p = accept \circ ([g, z])$ の形で定義され，かつ $([g, z])$ が型 $\mathsf{Bool} \to \mathsf{Bool}^k$ を持つとする．このとき，$mmp\ p$ を計算する効率の良い並列プログラムが存在する． □

5. 実　　装

本稿で示した導出法では，最大マーク付け問題の仕様をもとに，図 4 における値 $\overline{z}$，関数 $\overline{g}$，$\overline{accept}$，q_{ij} を代入することによって並列プログラムを導出することができる．したがって，スケルトン並列プログラムを導出するシステムを比較的容易に実現できる．

本章では，まず，スケルトン並列プログラムを導出するシステムの概観を示し，導出される並列プログラムの最適化について述べる．次に，いくつかの例について実験を行った結果を示す．

5.1 スケルトン並列プログラムの導出システム

システムの入力には，述語関数の定義を相互再帰関数として与える．たとえば，3 章に示した最大末尾部分列和問題の場合には，組化変換を適用する前の下記の入力を与える．

```
mts x = all x || last x;
all []    = T;
all (a:x) = a && all x;
last []    = F;
last (a:x) = !a && (all x || last x);
```

まず，システムは，入力から値 $\overline{z}$，関数 $\overline{g}$，$\overline{accept}$，q_{ij} を求める．最大末尾部分列和問題の場合には，$(all \vartriangle last)$ の組を 2 進数として読んだ $[0, 4)$ の整数を状態とし，$\overline{z} = 2$ および以下の関数を導出する．

i	0	1	2	3
$\overline{accept}\ i$	F	T	T	T
$\overline{g}\ \mathsf{T}\ i$	0	0	2	2
$\overline{g}\ \mathsf{F}\ i$	0	1	1	1

q_{ij} a	0	1	2	3
0	$a \uparrow 0$	a	$-\infty$	$-\infty$
1	$-\infty$	0	0	0
2	$-\infty$	$-\infty$	a	a
3	$-\infty$	$-\infty$	$-\infty$	$-\infty$

次に，不要な状態を削除することにより導出されるプログラムを効率化する．これらの最適化の詳細については，次節で述べる．最大末尾部分列和問題の場合には，状態 1 および 2 のみが必要な状態となる．

最後に，導出した値 $\overline{z}$，関数 $\overline{g}$，$\overline{accept}$，q_{ij} を使い，並列スケルトンを使ったプログラムを生成する．本システムでは，C++による並列スケルトンライブラリ「助っ人」[17] 上で動くコードを生成する．1 パス目の scanr に対するコードの生成において，Matsuzaki ら[16] によって提案された行列乗算を通じて定数である要素を除去する最適化を行っている．最大末尾部分列和問題に対して導出されるプログラムの一部を**図 5** に示す．

5.2 導出されるプログラムの最適化

導出されるプログラムのうち，scanr に対する準結合的な演算子が行列乗算であるため，状態の数を S とすると準結合的な演算子の計算時間は $O(S^3)$ となる．また，scanl および scanr における並列化のオーバヘッドは $(1+S)$ 倍となる．したがって，状態の数を減らすことにより，導出されるプログラムを効率化することができる．

本システムでは，**図 6** に示すような状態遷移のグラフを使い不要な状態を除去する．状態遷移のグラフにおいて，遷移は関数 $\overline{g}$ によって得られ，$\overline{z}$ を初期状態，関数 $\overline{accept}$ の値が T となる状態を受理状態と呼ぶ．なお，到達可能性のみを問題とするので，T と F の遷移を区別しない．不要な状態は，状態遷移のグラフにおいて次のいずれかの性質を満たすものである．

(1) 初期状態から遷移して到達できない．
(2) その状態から遷移していずれの受理状態にも到達できない．

性質 (2) については，遷移グラフの有向枝を逆転させたグラフ上で到達ノードを調べることで得られるが，受理状態は一般に複数あることに注意する必要がある．

最大末尾部分列和問題では，遷移グラフは図 6 である．このグラフから，性質 (1) を満たす状態は状態 3 であり，性質 (2) を満たす状態は状態 0 である．これらの状態を除去することにより，最大末尾部分列和問題においては状態 1 および 2 のみが必要な状態と

```
namespace mts {
  typedef int number;
  number NINF = INT_MIN;
  struct ntup {
    number w_1, w_2;
    ntup(number w_1, number w_2) {
      this->w_1 = w_1; this->w_2 = w_2;
    }
    ntup() {}
  };
  struct matrix {
    number w_1_2, w_2_2;
    matrix(number w_1_2, number w_2_2) {
      this->w_1_2 = w_1_2; this->w_2_2 = w_2_2;
    }
    matrix() {}
  };
                          (省略)
  SKETO_DEF_BINOP(f1, ntup, ntup, number,
    return ntup(
      max(x.w_1, x.w_2),
      plus(x.w_2, y)
    );
  );
  SKETO_DEF_UNOP(g1, matrix, number,
    return matrix(0, x);
  );
  SKETO_DEF_BINOP(om1, ntup, ntup, matrix,
    return ntup(
      max(x.w_1, plus(y.w_1_2, x.w_2)),
      plus(y.w_2_2, x.w_2)
    );
  );
  SKETO_DEF_BINOP(op1, matrix, matrix, matrix,
    return matrix(
      max(x.w_1_2, plus(y.w_1_2, x.w_2_2)),
      plus(y.w_2_2, x.w_2_2)
    );
  );
  ntup z1(NINF, 0);
                          (省略)
  dist_list<bool>* solver(dist_list<number>* x) {
    std::pair<dist_list<ntup>*, ntup> xp_w
       = gscanr(f1, z1, g1, om1, op1, x);
    dist_list<ntup>* tmp
       = shiftr(xp_w.second, xp_w.first);
    dist_list<aww_>* xpp
       = new dist_list<aww_>(x->get_global_size());
    zipwith(zipw1, xp_w.first, tmp, xpp);
    delete tmp; delete xp_w.first;
    zipwith(zipw2, x, xpp, xpp);
    int c = 0; number m = NINF;
    if (m < w.w_1) { c = 1; m = w.w_1; }
    if (m < w.w_2) { c = 2; m = w.w_2; }
    dist_list<int>* y
      = gscanr(f2, c, g2, om2, op2, xpp).first;
    dist_list<bool>* result = zipwith(f3, y, xpp);
    delete xpp; delete y;
    return result;
  }
} // namespace mts
```

図 5 導出されたスケルトン並列プログラムの一部
Fig. 5 Derived code of skeletal parallel program.

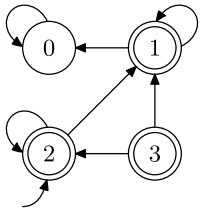

図 6 状態遷移グラフ．初期状態は 2，受理状態は二重丸で示される
Fig. 6 State-transition graph. The initial state is 2, and acceptable states are denoted by doubly-line.

なる．

5.3 例

5.3.1 最大独立要素和問題

まず，リストの連続する要素を選択してはならないという条件のもとで重み和が最大となるように要素を選択する，最大独立要素和問題を考える．この問題は，次のとおり定義される述語 `mis` による最大マーク付け問題である．関数 `ind` はリストの選択された要素が独立しているかどうかを返す関数，関数 `head` はリストの先頭の要素を返す関数である．

```
mis x = ind x;
ind []    = T;
ind (a:x) = ind x
          && (!a || (a && !(head x)));
head []    = F;
head (a:x) = a;
```

この述語の仕様から，値 $\overline{z}$，関数 $\overline{g}$，$\overline{accept}$ を次のように得ることができる．ただし，状態は (ind△head) の返す組二進数として読んだものを用いる．

$\overline{z} = 2$

i	0	1	2	3
$\overline{accept}\ i$	F	F	T	T
$\overline{g}\ \mathsf{T}\ i$	1	1	3	1
$\overline{g}\ \mathsf{F}\ i$	0	0	2	2

$q_{ij}\ a$	0	1	2	3
0	0	0	$-\infty$	$-\infty$
1	a	a	$-\infty$	a
2	$-\infty$	$-\infty$	0	0
3	$-\infty$	$-\infty$	a	$-\infty$

このようにして導出した関数 $\overline{g}$ および $\overline{accept}$ に対して最適化を行うことで，必要な状態は 2 および 3 となる．上記の関数のうち，必要な状態のみを取り出した関数の定義は下記のようになる．なお，関数 $\overline{g}$ の定義における $\perp$ は，その状態の生成を無視して良いことを示す．

i	2	3
$\overline{accept}\ i$	T	T
$\overline{g}\ \mathsf{T}\ i$	3	$\perp$
$\overline{g}\ \mathsf{F}\ i$	2	2

$q_{ij}\ a$	2	3
2	0	0
3	a	$-\infty$

5.3.2 最大部分列和問題

次に，Programming Pearls [3] の 1 つとしても知られる最大部分列和問題の例を示す．最大部分列和問題とは，1 つの連続する部分列の重み和のうち最大のものを求める問題であるが，ここではそのような部分列を求める問題と拡張したものを解く．

最大部分列和問題は次の関数 `mss` を述語とする最大マーク付け問題である．以下の定義において，関数 `none` はリストが 1 つも T を含まないかどうかを判定する関数，`seg` は 1 つの連続する部分列が選択されているかどうかを判定する関数，`head` はリストの先頭要素を返す関数である．

```
mss x = none x || seg x;
none []    = T;
none (a:x) = !a && none x;
seg []    = F;
seg (a:x) = (none x && a)
   || (seg x && (!a || (a && head x)));
head []    = F;
head (a:x) = a;
```

状態として (none △ seg △ head) の返す組を二進数として読んだものを用いる．値 $\overline{z}$ は 4 である．最適化を行った後に最終的に得られる関数 $\overline{g}$，$\overline{accept}$，q_{ij} は下記となる．

i	2	3	4
$\overline{accept}\ i$	T	T	T
$\overline{g}\ \mathsf{T}\ i$	$\perp$	3	3
$\overline{g}\ \mathsf{F}\ i$	2	2	4

$q_{ij}\ a$	2	3	4
2	0	0^*	$-\infty^*$
3	$-\infty$	a^*	a^*
4	$-\infty$	$-\infty$	0

このうち，状態 4 に対応する重み和はつねに 0 であり，最終的なコード生成においてその計算は除去される．また，上記の q_{ij} の行列のうち，$*$ が肩についた

表 1 最大部分列和問題に対する実験結果
Table 1 Experiment results for maximum segment sum problem.

	最適化あり		最適化なし	
P	時間（秒）	速度向上	時間（秒）	速度向上
1	8.52	1.00	51.42	1.00
2	7.50	1.14	263.33	0.20
4	3.80	2.24	132.05	0.39
8	1.88	4.53	66.38	0.77
12	1.26	6.78	44.20	1.16
16	0.94	9.07	33.13	1.55
24	0.69	12.29	22.11	2.33
32	0.65	13.05	16.73	3.07

位置の要素のみ値を計算すればよく，残りの要素の計算は省略される．この判定は，行列計算のシミュレーションによる最適化[16] によって行われる．

5.4 実　　験

得られたスケルトン並列プログラムの効率を調べるため，実験を行った．実験に使用したスケルトン並列プログラムは，最大部分列和問題を解くもので，最適化の有無で 2 通り導出した．データには，要素数 2^{24} である整数のリストを用いた．実行環境については，Pentium Xeon 2.4 GHz の CPU，2 GB のメモリを持つ計算ノードがギガビットイーサネットで接続された PC クラスタを使用した．またソフトウェアは，linux 2.4.21，gcc 4.1.1，mpich 1.2.7-p1 を使用している．並列スケルトンのライブラリ「助っ人」については，筆者らが開発中である非公開版を用いた[⋆1]．

実行時間と速度向上を**表 1** および**図 7** に示す．実行時間はデータの分配と収集の時間を含まない．また，速度向上については，各プログラムを 1CPU で実行した場合に対する比で表している．

まず，1CPU で実行した場合の計算時間についてみると，最適化を行うことでおよそ 6 倍速くなっている．これは，計算すべき状態数が 8 から 2 へと減ったことによるものである．次に，速度向上についてみる．並列スケルトン scanl および scanr は準結合的な演算子を要求し，一般に，導出した結合的な補演算子は元の計算と比較して計算コストが大きくなる．特に，今回導出した結合的な補演算子の計算コストは，おおよそ，元の計算コストに状態数をかけたものになる．この原因により，最適化した並列プログラムの方ではオーバヘッドが約 1.7 倍，最適化前の並列プログラムでは約 10 倍のオーバヘッドがかかっている．ただし，2CPU 以上では図 7 に見られるように，良い速度向上が得られている．実験結果より，本稿の導出法による並列プログラムは避けられない多少のオーバヘッドがあるものの良い速度向上を示し，また，最適化を行うことが重要であることが確かめられた．

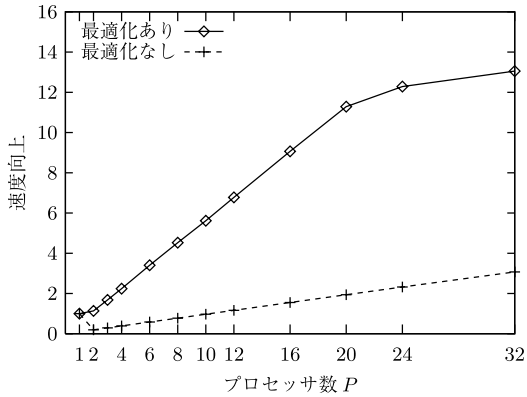

図 7 最大部分列和問題に対する速度向上
Fig. 7 Speedup against sequential program for maximum segment sum problem.

6. 関 連 研 究

最大マーク付け問題の具体的な問題である最大部分列和問題（リスト）および最大部分矩形問題（行列）については，これまで多くの研究がある[1),2),11),18)]．スケルトン並列プログラミングの枠組みにおいて取り組んだものとして，Cole による最大部分列和問題の導出[8)] や Hu らによる最大部分矩形和問題[11)] がある．また，最大マーク付け問題の木構造における具体的な問題に対する並列プログラムについては，He[10)] や Matsuzaki ら[15)] による導出がある．一方，一般的な問題の枠組みに対して並列化を議論した研究は少ない．最大重み和を求める並列プログラムの導出について，Kakehi ら[13)] のリスト準同型に基づく手法がある．

逐次プログラムの導出に関しては，これまでに非常に多くの研究がある．最大マーク付け問題に関して，古くは再帰構成グラフ（decomposable graph）上での単項二階論理に基づいたプログラム導出などが行われてきた[4),6)]．また，関数型言語による仕様に基づく逐次プログラムの導出には，本研究で基礎とした篠埜らによるもの[19),21)] のほか，Bird によるもの[5)] などがある．

本稿では，導出される並列プログラムを最適化するために，2 つの削除可能な状態の性質を 5.2 節で示した．篠埜ら[19),21)] のプログラム導出では，状態遷移に沿って計算を進めるため，性質 (1) の開始状態から到達できない状態は生成されない．一方，Bird[5)] による枠組みでは，遅延評価を行うことにより，性質 (2) の

⋆1 2008 年春の公開を目指している．

最終状態に到達することのない状態は生成されない．本稿での導出では，状態をすべて表として持つようにプログラムを生成するため，これらの不要な状態を明示的に削除する手法をとった．

7. ま と め

本稿では，リスト上の最大マーク付け問題について，その問題を解くスケルトン並列プログラムを導出する手法を与え，また，その導出を行うシステムについて述べた．スケルトン並列プログラムの導出は 3 ステップからなる．(1) 篠埜らによる導出法によって得られる逐次プログラムを変形する．(2) 逐次プログラム中の関数を並列スケルトンによって置き換える．(3) 得られた並列スケルトンの引数の関数に対して準結合的な演算子を導出する．また，得られたプログラムの台数効果および導出システムにおける最適化の効果について，実験によって確かめた．

今後の課題として，並列プログラムを導出できるクラスの拡大，および，行列や木などのより複雑なデータ構造に対する導出法の構築などがある．

謝辞 本研究の一部は，科学研究費補助金（基盤研究（B）17300005，若手研究（B）18700021）の助成を受けた．ここに記して謝意を表す．

参 考 文 献

1) Alves, C.E.R., Cáceres, E. and Song, S.W.: BSP/CGM Algorithms for Maximum Subsequence and Maximum Subarray, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, *Proc. 11th European PVM/MPI Users' Group Meeting*, LNCS, Vol.3241, pp.139–146 (2004).
2) Bae, S.E. and Takaoka, T.: Algorithms for the Problem of k Maximum Sums and a VLSI Algorithm for the k Maximum Subarrays Problem, *7th International Symposium on Parallel Architectures, Algorithms, and Networks* (*I-SPAN 2004*), pp.247–253 (2004).
3) Bentley, J.: *Programming Pearls*, 2nd edition, Addison-Wesley (1999).
4) Bern, M.W., Lawler, E.L. and Wong, A.L.: Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs, *Journal of Algorithms*, Vol.8, No.2, pp.216–235 (1987).
5) Bird, R.S.: Maximum Marking Problems, *Journal of Functional Programming*, Vol.11, No.4, pp.411–424 (2001).
6) Borie, R.B., Parker, R.G. and Tovey, C.A.: Automatic Generation of Linear-Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families, *Algorithmica*, Vol.7, No.5&6, pp.555–581 (1992).
7) Cole, M.: *Algorithmic Skeletons*: *Structural Management of Parallel Computation*, *Research Monographs in Parallel and Distributed Computing*, MIT Press (1989).
8) Cole, M.: Parallel Programming with List Homomorphisms, *Parallel Processing Letters*, Vol.5, No.2, pp.191–203 (1995).
9) Emoto, K., Matsuzaki, K., Hu, Z. and Takeichi, M.: Domain-Specific Optimization Strategy for Skeleton Programs, *Euro-Par 2007, Parallel Processing*, *Proc. 13th International Euro-Par Conference*, LNCS, Vol.4641, pp.705–714 (2007).
10) He, X.: Efficient Parallel Algorithms for Solving Some Tree Problems, *24th Allerton Conference on Communication, Control and Computing*, pp.777–786 (1986).
11) Hu, Z., Iwasaki, H. and Takeichi, M.: Formal Derivation of Parallel Program for 2-Dimensional Maximum Segment Sum Problem, *Euro-Par '96 Parallel Processing*, *Proc. 2nd International Euro-Par Conference*, Vol.I, LNCS, Vol.1123, pp.553–562 (1996).
12) Hu, Z., Iwasaki, H., Takeichi, M. and Takano, A.: Tupling Calculation Eliminates Multiple Data Traversals, *Proc. 1997 ACM SIGPLAN International Conference on Functional Programming* (*ICFP '97*), pp.164–175 (1997).
13) Kakehi, K., Hu, Z. and Takeichi, M.: List Homomorphism with Accumulation, *Proc. ACIS Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (*SNPD'03*), pp.250–259 (2003).
14) Matsuzaki, K.: Efficient Implementation of Tree Accumulations on Distributed-Memory Parallel Computers, *Proc. ICCS 2007*: *7th International Conference, Part II*, LNCS, Vol.4488, pp.609–616 (2007).
15) Matsuzaki, K., Hu, Z. and Takeichi, M.: Parallelization with Tree Skeletons, *Euro-Par 2003, Parallel Processing*, *Proc. 9th International Euro-Par Conference*, LNCS, Vol.2790, pp.789–798 (2003).
16) Matsuzaki, K., Hu, Z. and Takeichi, M.: Towards Automatic Parallelization of Tree Reductions in Dynamic Programming, *SPAA 2006*: *Proc. 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp.39–48 (2006).

17) Matsuzaki, K., Iwasaki, H., Emoto, K. and Hu, Z.: A Library of Constructive Skeletons for Sequential Style of Parallel Programming, *InfoScale '06*: *Proc. 1st international conference on Scalable information systems* (2006).
18) Perumalla, K.S. and Deo, N.: Parallel Algorithms for Maximum Subsequence and Maximum Subarray, *Parallel Processing Letters*, Vol.5, No.3, pp.367–373 (1995).
19) Sasano, I., Hu, Z., Takeichi, M. and Ogawa, M.: Make it Practical: A Generic Linear-Time Algorithm for Solving Maximum-Weightsum Problems, *Proc. 5th ACM SIGPLAN International Conference on Functional Programming* (*ICFP '00*), pp.137–149 (2000).
20) Skillicorn, D.B.: The Bird-Meertens Formalism as a Parallel Model, *Software for Parallel Computation*, NATO ASI Series F, Vol.106, pp.120–133 (1993).
21) 篠埜 功，胡 振江，武市正人，小川瑞史：最大重み和問題の線形時間アルゴリズムの導出，コンピュータソフトウエア，Vol.18, No.5, pp.1–16 (2001).

(平成 19 年 9 月 14 日受付)
(平成 19 年 12 月 21 日採録)

松崎 公紀（正会員）

1979 年生．2001 年東京大学工学部計数工学科卒業．2003 年同大学大学院情報理工学系研究科修士課程修了．2005 年同研究科博士課程中退．同年より同研究科助手，2007 年より助教となり現在に至る．博士（情報理工学）．並列プログラミング，アルゴリズム導出等に興味を持つ．日本ソフトウェア科学会会員．

胡 振江（正会員）

1966 年生．1988 年中国上海交通大学計算機科学系卒業．1996 年東京大学大学院工学系研究科情報工学専攻博士課程修了．同年日本学術振興会特別研究員を経て，1997 年東京大学大学院工学系研究科情報工学専攻助手，同年 10 月同専攻講師，2000 年同専攻助教授．2001 年同大学院情報理工学系研究科助教授．2007 年より同研究科准教授，現在に至る．博士（工学）．関数プログラミング，プログラム変換，構造化文書処理，並列プログラミング等に興味を持つ．日本ソフトウェア科学会，ACM 各会員．

武市 正人（正会員）

1948 年生．1972 年東京大学工学部助手，講師，電気通信大学講師，助教授，東京大学工学部助教授を経て 1993 年東京大学大学院工学系研究科教授（情報処理工学講座），2001 年より同大学大学院情報理工学系研究科教授，現在に至る．2003 年より日本学術会議会員，現在に至る．工学博士．プログラミング言語，関数プログラミング，構造化文書処理の研究・教育に従事．日本ソフトウェア科学会，日本応用数理学会，ACM 各会員．