プログラミング 18－23
(1998. 3. 24)

PaiLisp プログラミングのための視覚的ツールの試み

三森 哲　　伊藤 貴康

東北大学 大学院 情報科学研究科

PaiLisp は，Scheme に豊富な並列構文を加えた並列 Lisp 言語であり，スティール評価法を用いた効率のよい処理系 PaiLisp/MT が作成され，様々な応用に使われつつある．式の構造や実行コストを考慮に入れ，実行効率のよい PaiLisp プログラムの作成を支援する視覚的ツール PaiVisualizer を設計・試作したので，その概要を報告する．PaiVisualizer は，PaiLisp プログラムのグラフ的視覚化機能，PaiLisp プログラムに現れる式の実行コストを求める機能，実行コストと構造に基づいてプログラムをリダクションする機能などを備えるものである．PaiVisualizer を用いて作成された並行プログラムの実行例についても述べる．

A Visualization Tool for PaiLisp Programming: PaiVisualizer

Satoshi Mitsumori　　Takayasu Ito

Department of Computer and Mathematical Sciences,
Graduate School of Information Sciences,
Tohoku University

PaiLisp is a parallel Lisp language with a rich set of concurrency constructs, and its efficient multi-threaded interpreter (PaiLisp/MT) based on the steal-help evaluation strategy is used for various applications. An interactive visualization tool, called PaiVisualizer, is designed and implemented for developing efficient PaiLisp programs. PaiVisualizer consists of the following mechanisms: (1) visualization of PaiLisp programs in a graphical form, (2) calculation of evaluation costs of PaiLisp expressions, (3) reduction mechanism of PaiLisp expressions into efficient ones. Some experimental results are also shown, using PaiLisp/MT on a multi-processor DEC7000.

1 はじめに

PaiLispは，豊富な並列構文を有する並列Lisp言語[6]であり，スティール評価法[5]を用いた効率のよい処理系PaiLisp/MTが作成され[7]，様々な応用に使用されている．PaiLispには逐次Scheme構文の殆ど全てに対応する並列構文が用意されているため，逐次Schemeプログラムの並列化は容易である．しかし，PaiLispプログラミングには，再帰と並列性を考慮する必要があり，効率のよいPaiLispプログラムの作成は必ずしも容易ではない．

効率のよいPaiLispプログラムの作成を可能とする視覚的プログラミング環境の構築を目指して，PaiVisualizerという視覚的ツールの設計と試作を行ったので，その概要を報告する．

PaiVisualizerは，PaiLispプログラムのグラフ的視覚化機能，式の実行コストを求める機能，実行コストと式の構造に基づき並列プログラムをリダクションする機能を具備するシステムである．

次のFibonacci関数のプログラムを考える．

```
(define (fib n)
  (if (< n 2) n (+ (fib (- n 1))
                   (fib (- n 2)))))
```

この並列化として次の2つのものが考えられる．

```
(define (pfib n)
  (if (< n 2) n (pcall + (pfib (- n 1))
                         (pfib (- n 2)))))
(define (ppfib n)
  (if (pcall < n 2) n
      (pcall + (ppfib (pcall - n 1))
               (ppfib (pcall - n 2)))))
```

並列関数適用 **pcall** の多用により ppfib の方がpfibよりも並列度は高いが，実際に実行するとプロセス生成に伴うオーバーヘッドのため，pfibの実行の方が速い．一方，fibから，複数の引数を持つ(+, −, <のような)関数は全て**pcall**で並列化し，ppfibを得ることは容易である．PaiVisualizerを用いると，ppfibのように過剰並列化されたプログラムから実行効率のよいpfibのような並列プログラムを対話的に作成できる．なお，PaiVisualizerは，MultilispのVista[4]や，Progragh[2]等の視覚的ツールと異なり，効率のよい並列プログラム作成の直接的な支援を目指したツールである．

2 PaiVisualizer 設計の基本方針

逐次プログラムを並列化して実行しても評価法によってはかえって遅くなってしまう場合がある．例えば，ETCを用いて(pfib 18)を計算しても(fib 18)の逐次実行より遅くなる．(pfib 18)のETC評価の場合，8000個以上もの多数のプロセス生成によりトータルコストが大きくなるためである．この問題は，スティール評価法によって対処できる．PaiLisp/MTのスティール評価法を用いると(pfib 18)のプロセス生成数は約60個に抑制される[5, 7]．

効率のよいPaiLispプログラムの作成にも，並列評価におけるプロセス生成コストを考慮に入れる必要がある．PaiVisualizerは，この点を考慮に入れて設計され，効率のよいPaiLispプログラムを開発するため次のような機能を有する．

1. 図的表示機能：PaiLispプログラムをグラフ表示で視覚的に表示でき，また，視覚的要素を用いてプログラム作成をすることができる機能．
2. コスト計算機能：式やプログラムの評価コストを求める機能．
3. リダクション機能：式の評価コストを基に，グラフ表示されたPaiLispプログラムから過剰な並列性を除去し，リダクションできる機能．

2.1 PaiLisp 構文と処理系の基本的評価コスト

PaiVisualizerが対象とするPaiLispは，Schemeの逐次構文に加えて次のような並列構文を有する．

(**pcall** $f\ E_1 \cdots E_n$) | (**pbegin** $E_1 \cdots E_n$)
| (**pif** $E_1\ E_2\ E_3$) | (**future** E)
| (**par-and** $E_1 \cdots E_n$) | (**par-or** $E_1 \cdots E_n$)
| (**plet** $((x_1\ E_1) \cdots (x_m\ E_m))\ E_{m+1} \cdots E_n$)
| (**pcond** $((E_{11}\ E_{12}) \cdots (E_{n1}\ E_{n2}))$)

その他，**par**，**pletrec**，**pcond#**，**call/pcc**などの並列構文がある(文献[5, 6, 7]を参照)．

共有メモリ型並列計算機DEC 7000上のPaiLisp/MTインタプリタは，実行時のプロセス生成が少ないスティール評価法を用いた効率のよい処理系である．この処理系について次のような基本的評価コストが知られている．

- 構文識別のコストや基本式(定数，変数，クオート式)の評価コスト：1〜2μsec.
- 基本関数(cons,car,cdr,+,-,=,null?,eq?など)の評価コスト：2μsec.
- 変数束縛のコスト：2μsec.
- ラムダクロージャ生成コスト：2.2μsec.
- 関数適用のコスト $Cost[fun\text{-}apply]$：5μsec.
- スティール評価によるプロセス生成コスト $Cost[p\text{-}create]$：70μsec.

2.2 効率のよい並列プログラム作成の基本方針

関数適用($f\ e_1 \cdots e_n$)の並列化は(**pcall** $f\ e_1 \cdots e_n$)と表すことができる．ここに，**pcall**構文は引数式を全て並列に評価し終えてからfの評価を行い，その値を引数式の値に適用するという意味を持つ．($f\ E_1\ E_2$)の評価コストは，次のようになる．

$Cost[(f\ E_1\ E_2)]$
$\Rightarrow\quad Cost[f] + Cost[fun\text{-}apply] + Cost[E_1] + Cost[E_2]$

(**pcall** f E_1 E_2) において E_2 がスティールされ並列評価された場合には次のようになる．

$$Cost[(\textbf{pcall}\ f\ E_1\ E_2)]$$
$$\Rightarrow\ Cost[\textbf{pcall}] + Cost[f] + Cost[fun\text{-}apply] + Max[Cost[E_1], Cost[p\text{-}create] + Cost[E_2]]$$

$Cost[\textbf{pcall}]$ は $Cost[p\text{-}create]$ よりもかなり小さいので，E_1, E_2 の評価コストが共にプロセス生成コストより大きい $Min[Cost[E_1], Cost[E_2]] > Cost[p\text{-}create]$ の場合には，並列評価の方が逐次評価よりも速くなる．一方，E_1, E_2 の評価コストがプロセス生成コストよりも小さい場合には，**pcall** で並列評価してもプロセス生成コストにより逆に遅くなってしまう．

PaiLisp/MT インタプリタの場合，プロセス生成コスト $Cost[p\text{-}create]$ は 70μsec である．他の並列構文においても **pcall** と同様にプロセス生成コストを考慮する必要が生じる．そこで効率のよい並列プログラムの作成は次の方針で行われる必要がある．

- 並列構文で書かれた式の並列評価においてプロセス生成を伴う引数式の評価コストがプロセス生成コストより大きければ並列評価をし，引数式の評価コストがプロセス生成コストより小さければ並列構文を取り除き逐次評価をした方がよい．また，逐次プログラムが与えられたとき，それが複数の式に分割できてそれらのコストがプロセス生成コストより大きければ並列構文によって (可能であれば) 並列化を行う．

3 PaiVisualizer の視覚化機能

PaiVisualizer を用いた PaiLisp プログラムの作成には，次の 2 つのアプローチが考えられる．

- Scheme プログラムを作成し，次いで PaiLisp の並列構文を用い過剰気味の並列プログラムに変換する．その後，PaiVisualizer を用いて効率のよい PaiLisp プログラムに改良する．
- PaiLisp の豊富な構文と対応する視覚的表示を利用して，並列の PaiLisp プログラムの作成，改良の際に，PaiVisualizer を用いる．

PaiVisualizer は，このようなプログラミング過程の各ステップを視覚的に表示し，表示部分の指定，コスト情報の計算と付与の指示，リダクションの適用の指定などが行えるように設計されている．

3.1 PaiVisualizer における表示と実装環境

PaiVisualizer は，並列プログラムに対する次のような 3 次元カラー表示システムである．

1. カラーマルチウインドウを用いて PaiLisp プログラムの構造を 3 次元的にグラフ表示できる．
2. プログラムに現れる定数，変数，関数などを視覚的アイテムで表示し，プログラムの逐次的な実行と制御の流れを横方向 (X 方向)，条件分岐を奥行き方向 (Y 方向)，並列性を縦方向 (Z 方向) の矢印等で表示する．
3. 実装には，Scheme をベースとする PaiLisp との親和性を考慮して，Scheme に対して Tk を扱えるように拡張した STk[3] を利用した．STk に対し，3 次元表示を行うため 3 次元配列のベクタを扱えるように拡張し，STk+(3 次元ベクタとそのためのプリミティブ関数) を実装言語とした．
4. プログラムのグラフ表示に図的要素を用意し，それらを組み合わせて構文要素を表示し，抽象化のレベルに応じて階層的にプログラムのグラフ表示をできるようにした．
5. 作成された PaiLisp プログラムの処理系は，6 プロセッサの共有メモリ型並列計算機 DEC7000 上の PaiLisp/MT インタプリタである．

3.2 プログラムの視覚的表示

プログラムの厳密な記述は，プログラムテキストとして与えられ，それが処理系に読み込まれて実行される．このことを念頭に PaiVisualizer の視覚的表示機能の設計を行った．

1. プログラムテキストとプログラムの視覚的表示の対応を取り易くするために，表示用構成要素を数多く用意し，構文の視覚的表示が自然に行え，拡張性にも対応できるようにした．
2. プログラムを，式や演算，データと制御の流れ，データの記憶，スコープとブロックの明示，更に，それらの合成によって表示する観点から，構成要素の設計を行い，プログラムを構成要素間を結ぶグラフ的表示で与えた．
3. カラーマルチウインドウを用いて，プログラムテキストそのものの表示，プログラムのグラフ的表示，プログラムのスケルトンとそれへのコスト情報付与の表示をできるようにし，式や構文の階層的なグラフ的表示，マウス操作による各要素へのアクセス等も可能にした．また，構成要素やデータや制御の流れは状態により色を変化させうるようにした．

(1) 構成要素

構成要素は，PaiLisp の式の構造と実行の流れを図的に表示するという観点から設計し，プログラムは拡張されたデータフローグラフとして表示する．プログラムの詳細を容易に表示するため，30 余種類の構成要素を用意している．

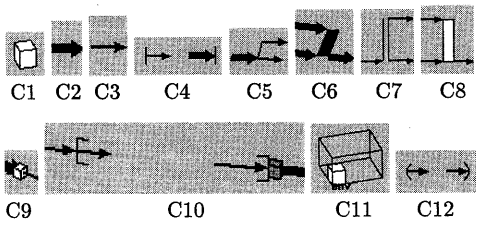


図 1: プログラムの視覚的表示の構成要素

図 1に構成要素のいくつかを示した．C1 は式や演算を表すアイテムである．C2 はデータと制御の矢印の方向への流れを表す．C3 は制御が矢印の方

向に移ることを表す．C4 はブロックの始めと終りを表す．C5 は条件分岐の始点，C6 は条件分岐の終端を表す．C7 は並列構造の開始，C8 は並列構造の終端を表す．C9 は流れて来たデータの一時記憶，C10 は関数適用に使用される略記法である．C11 は変数束縛におけるスコープを表し，C12 は内部のアイテムがクロージャの仮引数であること表す．その他，**quote**，**future** ，**call/cc** などを表示するための構成要素も用意されている．

(2) 構文の表示

いくつかの構文のグラフ的表示について説明する．

1. 式の表示：PaiLisp の式である定数，変数，演算，基本式，定義式などは，図 1の C1 を用い，それに式やその名前を付与して表示する．
2. (**begin** $e_1 \cdots e_n$) の表示：`(begin e1 e2 e3)` を例に取ると図 2のように表示される．`e1` を計算の後に制御が `e2` の計算に移り，`e2` の計算の後に `e3` の計算がされ，その値 (`e3` の値) がデータとして返されることを表示している．

図 2: **begin** の表示

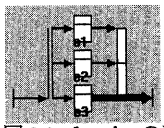
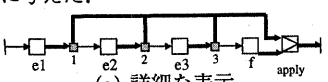


図 3：**pbegin** の表示

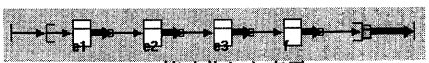
3. (**pbegin** $e_1 \cdots e_n$) の表示：`(pbegin e1 e2 e3)` を例に取った表示を 図 3に与えた．この場合，`e1`，`e2`，`e3` が並列評価され，それらの評価が全て終了してから `e3` の値がデータとして返されることを意味している．

註 **pbegin** の引数式$e_1 \cdots e_n$が「副作用」を有する場合には，(**begin** $e_1 \cdots e_n$) が (**pbegin** $e_1 \cdots e_n$) に変換できる訳ではなく，安易にこのような変換を行っても逐次と並列で意味が変わってくることに注意する必要がある．プログラムの意味については，原則としてユーザが責任を持つ必要がある．なお，PaiVisualizer には，副作用が生じる式に色を付けて注意を促す機能もある．

4. (f $e_1 \cdots e_n$) の表示：`(f e1 e2 e3)` の表示を図 4(a) に与えた．



(a) 詳細な表示



(b) 簡略化した表示

図 4: `(f e1 e2 e3)`

`e1` の値が計算され，その値が一時記憶に置かれ，その後 `e2`，`e3` についても同様の処理がされた後，`f` の値が計算されその値が一時記憶に置かれた `e1`，`e2`，`e3` の値に適用されることを表示している．しかし，Lisp で度々現れる関数適用にこの表示を用いるのは煩雑になるので，通常は，より簡略化した図 4(b) によって表示する．

5. (**pcall** f $e_1 \cdots e_n$) の表示：`(pcall f e1 e2 e3)` の表示を図 5に与えた．この表示は簡略化した表示であり，図 4(a) のような詳細な表示を与えることも可能である．

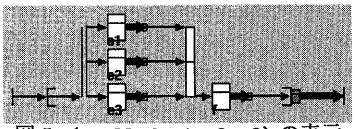


図 5: `(pcall f e1 e2 e3)` の表示

6. (**if** e_1 e_2 e_3) の表示：式 e_1を評価し，e_1の値が偽でなければ e_2の評価値を返し，偽であれば e_3の評価値を返すことを図 6のように表示する．

図 6: **if** の表示

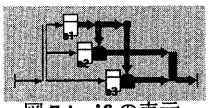


図 7：**pif** の表示

7. (**pif** e_1 e_2 e_3) の表示：**pif** の場合には，式 e_1, e_2, e_3が並列評価され，e_1の値によって実行中の不要な式の計算を強制終了させることを図 7のように表示させている．
8. (**plet** $((x_1\ e_1) \cdots (x_n\ e_n))$ $E_1 \cdots E_m$) の表示：式$e_1 \cdots e_n$が並列評価された後，それらの値が$x_1 \cdots x_n$に束縛され，環境を拡張してからそのスコープの下で$E_1 \cdots E_m$が並列評価され，E_mの値が返されることを図 8のように表示している．

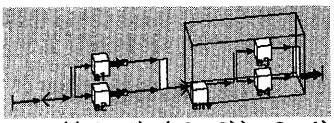


図 8: `(plet ((x1 e1) (x2 e2)) e3 e4)` の表示

(3) 視覚化の例

構文を組み合わせた式の表示例を図 9に示す．

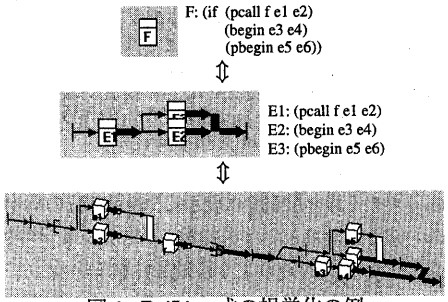


図 9: PaiLisp 式の視覚化の例

この図のように，PaiVisualizer では，式の抽象レベルを変化させた表示ができるので，ユーザの望まない視覚的情報を抑制することができる．

4 PaiVisualizer のリダクション機能

PaiVisualizer のリダクション機能は，PaiLisp プログラムのスティール評価法を用いた PaiLisp/MT

による実行を前提として，式のコストとプログラムの構造をもとに効率のよい並列PaiLispプログラムを作成する機能である．PaiVisualizerには，式のコスト計算機能，コストスケルトンを求める構造保存型リダクション，並列スケルトンを求めて過剰並列化を除去するコストリダクションの機能が具備されている．

4.1 式のコスト計算機能

グラフ表示されたプログラムに現れる式のコストを計算し，視覚構成要素に付与する機能である．式や演算のコストは，2.1節で述べたような基本的評価コストと，式の評価順序を考慮して求められる．各構文のスティール評価におけるコストは，以下の近似的なコスト計算規則を用いて計算される．

$$
\begin{aligned}
&Cost[(f\ E_1 \cdots E_n)]\\
&\quad \Rightarrow\ Cost[E_1] + \cdots + Cost[E_n] + Cost[f]\\
&\qquad\quad + Cost[fun\text{-}apply]\\
&Cost[(\mathbf{begin}\ E_1 \cdots E_n)]\\
&\quad \Rightarrow\ Cost[\mathbf{begin}] + Cost[E_1] + \cdots + Cost[E_n]\\
&Cost[(\mathbf{if}\ p\ E_1\ E_2)]\\
&\quad \Rightarrow\ Cost[\mathbf{if}] + Cost[p]\\
&\qquad\quad + (if\ p\ then\ Cost[E_1]\ else\ Cost[E_2])\\
&Cost[(\mathbf{pcall}\ f\ E_1 \cdots E_n)]\\
&\quad \Rightarrow\ Cost[\mathbf{pcall}] + Max[Cost[(\mathbf{pcall}\ f\ E_1 \cdots E_{n-1})],\\
&\qquad\qquad\qquad\qquad Cost[p\text{-}create] + Cost[E_n]]\\
&\qquad\quad + Cost[f] + Cost[fun\text{-}apply]\\
&Cost[(\mathbf{pbegin}\ E_1 \cdots E_n)]\\
&\quad \Rightarrow\ Cost[\mathbf{pbegin}]\\
&\qquad\quad + Max[Cost[(\mathbf{pbegin}\ E_1 \cdots E_{n-1})],\\
&\qquad\qquad\quad Cost[p\text{-}create] + Cost[E_n]]\\
&Cost[(\mathbf{pif}\ p\ E_1\ E_2)]\\
&\quad \Rightarrow\ Cost[\mathbf{pif}] + Max[Cost[p] + Cost[kill],\\
&\qquad\qquad\qquad\qquad (if\ p\ then\ Cost[E_1]\ else\ Cost[E_2])\\
&\qquad\qquad\qquad\qquad + Cost[p\text{-}create]]
\end{aligned}
$$

4.2 構造保存型リダクション機能

コスト計算機能によってコスト情報が付与されたグラフ表示を条件分岐，並列構造などの制御構造は保存しつつ，簡約化した表示へ変換するリダクションであり，この結果得られたグラフ表示をプログラムの**コストスケルトン**と呼ぶ．

`fib`，`pfib`，`ppfib`のグラフ表示をすると，それらが再帰的にグラフ表示に現れるが，このように再帰が現れた場合にはとりあえず"Large"という記号値を与える．再帰的に関数呼び出しがある場合でも，`fib`，`pfib`，`ppfib`のように引数の値が小さければ大きなコストにはならない(例えば，nが5以下であれば`(fib n)`の評価コストの方が`(pfib n)`の評価コストより小さい)．また，`(+ x1 x2 ⋯ x70)`の場合，x_iが変数ならば，変数の評価コストは小さいから，`(pcall + x1 x2 ⋯ x70)`のように並列化するのは無益である．しかし，`(+ (+ x1 ⋯ x35) (+ x36 ⋯ x70))`に変形すると，`(pcall + (+ x1 ⋯ x35) (+ x36 ⋯ x70))`は意味のある並列化となる．効率のよい並列プログラムを作成するため，PaiVisualizerはこのような解析が行えるシステムとして設計されている．

4.3 コストに基づくリダクション機能

コスト情報を用いて並列スケルトンを求めた後，過剰並列化を除去するリダクションである．例えば，2章で**pcall**を例に述べた理由から，並列構文の引数式の全ての評価コストがプロセス生成コストよりも小さい場合には，並列化を行う必要がない．

- 並列スケルトン：コストスケルトンを対象とし，式を表わす構成要素の逐次実行の累積コストとプロセス生成コスト(70μsec)との比較を行う．累積コストがプロセス生成コストよりも小さい場合は表示を行わず，大きい場合には逐次実行された(複数の)構成要素を1つのアイテムとして表示する．このようにして得られた表示を**並列スケルトン**と呼ぶ．
- 過剰並列化の除去：並列スケルトンを対象として並列開始記号の引数アイテムのコストがプロセス生成コスト(70μsec)より小さいときには，並列化されている部分を逐次プログラムに変換し，コスト比較をした上で逐次プログラムの方が好ましければ逐次版を採用する．また，再帰関数でコストが"Large"と表示されている場合については，その引数の性質を考えても確かに"Large"であるか否かを検討し，並列評価の意味がないときには逐次プログラムに変換するものとする．

註 並列構文が複数の引数式を取る場合，2つ以上の引数式の評価コストがプロセス生成コストよりも大きければ並列化されたままにしておき，そうでないときには逐次に戻し，並列化された式とのコスト比較の上で有利な方を選択する．

- コストの大きい式の並列化の検討：並列スケルトンにおいて式の評価コストがプロセス生成コストの2倍以上の大きさのときには，4.2節で述べたように，分解して並列実行を行える可能性があるか否か吟味し，並列化の意義がある場合には，然るべき並列構文を選択して並列化する．

PaiVisualizerを用いたPaiLispプログラミングは，上記のような視覚的機能とリダクション機能を利用することによって，効率のよい並列プログラム作成という形で進められる．

5 PaiVisualizerの使用例

Fibonacci関数とNQueen問題の解の数を求める逐次プログラムについて，複数の引数を持つ関数を全て**pcall**によって過剰並列化する．その後，得られた並列プログラムに対してPaiVisualizerを用いて効率のよいプログラムを作成した例を示す．

[1] Fibonacci 関数の並列化

図10は ppfib に対するコストスケルトンを示し，それを基に求めた並列スケルトンを図11に示した．図11から過剰並列化を除去して得られたリダクション結果が図12である．

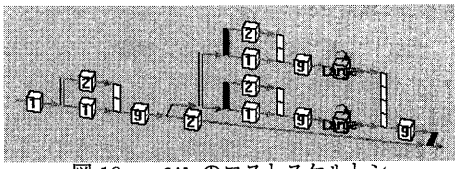


図 10: ppfib のコストスケルトン

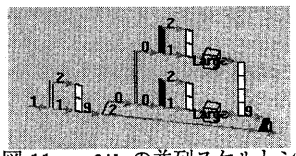


図 11: ppfib の並列スケルトン

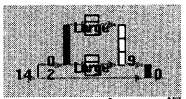


図 12: リダクションによって得られた pfib

表1に，pfib，ppfib の $n = 18$ のときの ETC とスティール評価法 (**SHE**) による実行時間とプロセス生成数を示した．fib については逐次実行のコストである．この表からスティール評価による pfib が最もよいことが知られる．

表 1: Fibonacci プログラムの評価結果 [sec]

評価方式	(fib 18)	(pfib 18)	(ppfib 18)
ETC	0.25	0.31	1.50
(プロセス数)	(0)	(8360)	(41802)
SHE	0.25	0.06	0.11
(プロセス数)	(0)	(59)	(88)

[2] NQueen の解の数を求めるプログラムの並列化

図13は，過剰に並列化されたプログラム ppqueen に対する図的表示を示し，それに対するコストスケルトンを図14に示した．図14をリダクションにより改良し，得られた結果が図15である．

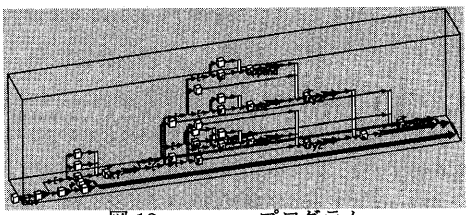


図 13: ppqueen プログラム

表2にプログラムの評価結果を示す．この表から スティール評価による pqueen が最もよいことが知られる．

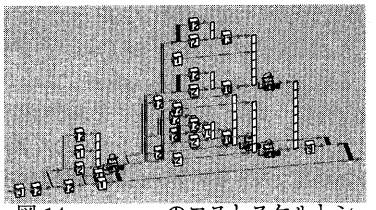


図 14: ppqueen のコストスケルトン

図 15: リダクションによって得られた pqueen

表 2: NQueen プログラムの評価結果 [sec]

評価方式	(queen 8)	(pqueen 8)	(ppqueen 8)
ETC	1.89	0.44	2.04
(プロセス数)	(0)	(4112)	(62568)
SHE	1.89	0.37	0.45
(プロセス数)	(0)	(62)	(144)

6 おわりに

PaiLisp プログラムを視覚化し，コストを考慮したリダクション機能を用いて効率のよい PaiLisp プログラムの作成を支援する PaiVisualizer という視覚的プログラミングツールについて報告した．PaiVisualizer には，PaiLisp の全ての PaiLisp 構文の視覚表示が具備され，各スケルトンを求めるリダクション規則も与えられているが，紙面の都合でその多くを省略した．なお，**call/cc** や **call/pcc** の扱い，プログラム変換機能の充実などは今後の課題である．

[本研究は，重点領域 (課題番号 09245102) の一環として行われたものである．]

参考文献

[1] W. Clinger, J. Rees (ed): Revised^4 Report on the Algorighmic Language Scheme, (1991).

[2] P. Cox, H. Glaser, B.Lanaspre.: Distributed Prograph, LNCS 1068, pp.128-133, Springer, (1995).

[3] E. Gallesio: STk Reference Manual ver 3.1, (1996).

[4] R. Halstead, Jr.: Understanding the Performance of Parallel Symbolic Progarams, LNCS 1068, pp.81-107, Springer, (1995).

[5] T. Ito: Efficient evaluation strategies for structured concurrency constructs in parallel Scheme systems, LNCS 1068, pp.22-52, Springer, (1995).

[6] T. Ito, M. Matsui: A parallel Lisp language PaiLisp and its kernel specification, LNCS 441, pp.58-100, Springer, (1990).

[7] 川本, 伊藤: "スティール評価法を備えた PaiLisp システムの実現とその評価", 情報処理学会論文誌, Vol.39, No.3, (1998)(掲載予定).