1U-03 並列コンティニュエーション呼び出しによる並列 Scheme コンパイラの実現*

宮川 伸也　　伊藤 貴康†

東北大学大学院情報科学研究科‡

1 並列 Scheme 言語 PaiLisp とその核言語

PaiLisp は，図 1に示すように Scheme に豊富な並列構文を加えた並列 Scheme 言語である．Scheme に 4

E ::= *Scheme expression* |
(**spawn** E) | (**exlambda** $(x_1 \cdots x_n)$ $e_1 \cdots e_m$) |
(**suspend**) | (**call/pcc** E) | (**call/pep** E) |
(**pcall** E_f $E_1 \cdots E_n$) |
(**pbegin** $E_1 \cdots E_n$) | (**par** $E_1 \cdots E_n$) |
(**plet** $((x_1\ E_1) \cdots (x_m\ E_m))$ $E_{m+1} \cdots E_n$) |
(**pletrec** $((x_1\ E_1) \cdots (x_m\ E_m))$ $E_{m+1} \cdots E_n$) |
(**par-and** $E_1 \cdots E_n$) | (**par-or** $E_1 \cdots E_n$) |
(**pcond** $(E_{11}\ E_{12}) \cdots (E_{n1}\ E_{n2})$) |
(**pcond#** $(E_{11}\ E_{12}) \cdots (E_{n1}\ E_{n2})$) |
(**pif** $E_1\ E_2\ E_3$) | (**future** E) | (**stealable** E)

図 1: PaiLisp の構文

つの基本的並列構文 **spawn**, **exlambda**, **suspend**, **call/pcc** を加えたコンパクトな言語は PaiLisp-Kernel と呼ばれる核言語であり，この核言語を用いて PaiLisp の全ての構文が記述できることが文献 [1] に示されている．このことを利用して ETC (Eager Task Creation) 評価法の下で PaiLisp の処理系を構成する方法が文献 [2] に与えられている．このとき図 1の各種の並列構文による制御メカニズムを核言語によって記述·実現する上で中心的な役割を演じるのが **call/pcc** 構文によって捉えられる並列コンティニュエーションである．並列コンティニュエーション $\mathcal{P}$-Continuation は，Scheme の **call/cc** によって捉えられる逐次コンティニュエーションの拡張となっており，逐次コンティニュエーションでは実現できなかったプロセスの kill や resume などの並行プロセスの制御を容易に記述·実現することができる．文献 [2] では，PaiLisp-Kernel を基に並列コンティニュエーションを利用したコンパクトな PaiLisp インタプリタが与えられている．しかし ETC を並列評価法として用いる限り，プロセス過剰生成問題のため効率のよい処理系の実現は望めない．本稿では，プロセス過剰生成問題を解決できる文献 [3,4] のスティール評価法を用い，また核言語と並列コンティニュエーションによる強力な制御機能を利用したコンパクトで効率のよい PaiLisp コンパイラ PaiLisp/PCC とその実験結果の概要について述べる．

2 スティール評価法と核言語アプローチによる PaiLisp コンパイラ PaiLisp/PCC

PaiLisp-Kernel の構文 **spawn** は ETC によるプロセス生成を記述する構文であり，スティール評価法を用いる場合には **stealable** を用いた核言語を考えることができる．しかし，本稿では，コンパイラ作成の観点から，**stealable** より低レベルな核言語を用いる．

[1] スティール評価法とその核言語

スティール評価法は実装的観点からは SST (Stealable Stack) に式を積み，ホームプロセッサは SST のトップから式を取って評価をし，アイドルプロセッサは SST のボトムから式をスティールして実行·評価する方法であると見なすことができる．このような観点から SST に対する次のような操作を導入する．

(**mk-stealable** E): 式 E と共有オブジェクトを SST に積む．
(**mk-disable** E): 式 E が SST のトップにある場合にポップして取り除く．

また，排他処理のため **exlambda** よりも低レベルな構文として **lock** と **unlock** を用いる．すなわち，スティール評価法によるコンパイラ実現の観点から次のような核言語 $\mathrm{PK_{steal}}$ を用いる．

$\mathrm{PK_{steal}}$ = Scheme + {**mk-stealable**, **mk-disable**, **call/pcc**, **lock**, **unlock**, **suspend**}

この $\mathrm{PK_{steal}}$ を用いて，スティール評価法に基づく PaiLisp 並列構文の記述を文献 [1] において PaiLisp-Kernel を用いて行ったと同様の仕方で与えることができる．$\mathrm{PK_{steal}}$ による各並列構文の制御メカニズムの記述において中心的な役割を果たすのが **call/pcc** によって捉えられる並列コンティニュエーションである．

[2] スティール評価法と核言語によるコンパイラ実現法

スティール評価法と核言語 $\mathrm{PK_{steal}}$ に基づくコンパイラの実現は次のように行われている．

(1) $\mathrm{PK_{steal}}$ の Scheme のコンパイラは筆者の一人 (伊藤) によって提案されている CC-Style(Continuation Calling Style) という方法を用いて実現されている．CC-Style は CPS(Continuation Passing Style) の一つの実現法であり，Scheme コンパイラは CPS に基

---

*An Implementation of a Parallel Scheme Compiler using $\mathcal{P}$-Continuation and the Steal-&-Help Evaluation Strategy

†Shin-ya Miyakawa, Takayasu Ito

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

づくコンパイラと同じく効率のよいコンパイラである．compiled object codes は Abelson-Sussman のレジスタマシンの命令コードである．

(2) スティール評価法のための核言語の並列構文については，それらの実現のためレジスタマシン命令を拡張して並行的タスク生成を行うコードを生成している．**mk-stealable** と **mk-disable** については図2に示す拡張レジスタマシンのコードが生成される．

```
   C[(mk-stealable E)]
       ⇒      (goto label2)
1 :  label1   C[E]
2 :  label2   (make-stealable-object label1 acc)
3 :           (push-SST acc)

   C[(mk-disable S)]
4 :    ⇒      C[S]
5 :           (try-pop-SST acc acc)
```

図 2: 拡張レジスタマシンのコードへのコンパイル

(3) 拡張レジスタマシンのコードに対して C 言語のプログラムを生成し，そのC言語プログラムをC言語コンパイラによってコンパイルして実行する．

以上のようにしてスティール評価のための核言語の PK_{steal} コンパイラは作成されている．

(4) PaiLisp 並列構文のコンパイルは次のように行っている．図 1に示した PaiLisp の各種の並列構文の核言語 PK_{steal} による記述を与える．その結果を上述の PK_{steal} コンパイラによって実行する．PaiLisp の並列構文の PK_{steal} による記述を与えるときに並列コンテイニューションが重要な役割を演じる．例えば，並列論理和演算 **par-or** において引数式のどれかが non-NIL の値を返したときに，他の引数式を評価するプロセスを並列コンティニューエションを用いたプロセスの KILL 操作によって kill するように記述する．並列論理積演算 **par-and** の実現においても並列コンティニュエーションを用いた KILL 操作が使われる．なお，PK_{steal} のコンパイラの作成と，PK_{steal} に基づく PaiLisp の記述の過程で各種の最適化が行われている．例えば，PK_{steal} による PaiLisp の記述には，**call/pcc** でなく **call/pep** でよいことが知られているが，この事を取り入れたコードの最適化もされている．

3 PaiLisp/PCC の実験評価

PaiLisp/PCC は 6 プロセッサの共有メモリ型並列計算機 DEC7000 上に実現されており，このシステムを用いて評価実験を行った．

(1) 同じマシン上にスティール評価法を用いた PaiLisp/MT コンパイラも実現されており，PaiLisp/PCC コンパイラとの比較実験を行った結果の一部を表 1 に示す．Fibonacci 数を計算するプログラムの並列化

表 1: PaiLisp/PCC と PaiLisp/MT の比較

	本コンパイラ	PaiLisp/MT
pfib	0.287	0.527
ptarai	0.319	0.691
pqueen	0.514	1.432
pmsort	0.036	0.067

[sec]

版 *pfib*，たらい回し関数の並列化版 *ptarai*，8 クイーン問題の解を求めるプログラムの並列化版 *pqueen*，マージソートプログラムの並列化版 *pmsort* についての結果である．これらのプログラムは文献 [4] に用いられたものと同じであり，並列化には関数適用の並列化を行う **pcall** を用いている．表 1から知られるように PaiLisp/PCC コンパイラは PaiLisp/MT コンパイラより約 2 ～ 3 倍高速になっている．

(2) 図 3は **par-or** を用いた並列化の例 (*prolog*)，**future** を用いた並列化の例 (*hou*)，**par** を用いた並列化の例 (*travels*) に対する PaiLisp/PCC の台数効果を示している．

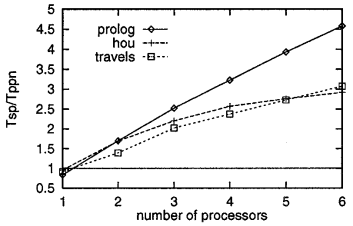

図 3: 中規模プログラムの台数効果

4 おわりに

PaiLisp/PCC は核言語 PK_{steal} のコンパイラを基にしたシステムであるからコンパクトなコンパイラであるが，最適化が行い易い CC-Style 法をベースとしているので，PaiLisp/MT よりも高性能な処理系となっている．核言語 PK_{steal} はスティール評価法に基づくコンパクトなコンパイラを簡便に作成する観点から設定されたが，核言語の見直し，並列コンティニュエーション呼び出しによる並列コンパイラの作成法の体系化について検討を進めている．

参考文献

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

[2] T. Ito, T. Seino, *P-Continuation Based Implementation of PaiLisp Interpreter*, LNCS, Vol. 748, pp. 108-154, Springer (1993).

[3] T. Ito, *Efficient evaluation strategies for structured concurrency constructs in parallel scheme system*, LNCS, Vol.1068, pp22-52, Springer (1996).

[4] 川本真一，伊藤貴康，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, pp.692-703, 1998.