スティール評価法のための仮想並列マシンの設計と実現

伊藤貴康[†] 宮川伸也[†]

スティール評価法は実行効率の良い並列 Scheme 処理系の実現のために提案された並列評価法である．並列処理系の実現によく用いられる ETC は並列実行するタスクをすべて生成してから実行を開始するため，タスク過剰生成問題が発生し，逐次実行よりも遅くなることが多い．スティール評価法では，並列評価したい式はいったん SST と呼ばれるスタックに入れられ，空きプロセッサが発生したときのみ SST から式がスティール操作によって取り出されるタスク遅延生成法に基づいている．このため，スティール評価法は，タスク過剰生成問題が発生しない効率の良い並列評価法となる．スティール評価法におけるタスク生成が SST に対する操作によって行われることに着目して設計したのが，仮想並列マシン—SST マシン—である．SST マシンは，SST に対する push，pop，steal の基本命令に加えて，SST の flush-out やコピー操作などを行う命令を備えた，スティール評価法のためのタスク生成を行うタスク生成系である．スティール評価法に基づく並列処理系を，タスク生成を行う SST マシンとタスクの実行を行うタスク実行系に分離して実現することができる．SST マシンを UNIX 環境において移植性の高い形で提供できるようにするため，C 言語を用いた SST マシンライブラリとして実現した．共有メモリ型並列計算機上で実現した SST マシンライブラリの性能について報告するとともに，その応用例として実現した並列 Scheme 処理系の実験結果についても述べる．

A Virtual Parallel Machine for the Steal-Help Evaluation Strategy

Takayasu Ito[†] and Shinya Miyakawa[†]

Steal-Help Evaluation (SHE) has been proposed to implement an efficient parallel Scheme system. Eager Task Creation (ETC) is usually used as a parallel evaluation strategy to implement parallel systems, but ETC is not necessarily efficient, since it incurs the problem of excessive process creation. In SHE, expressions to be evaluated in parallel are pushed into a stack called "SST". Under a shared parallel architecture a processor called "home processor" evaluates an expression, obtaining it from the top of the SST, and when some idle processor is available, it steals an expression in the SST from its bottom for parallel evaluation. SHE is known to be an efficient parallel evaluation strategy that suppresses excessive process creation, based on lazy creation of concurrent tasks. We propose a virtual parallel machine called "SST machine" with operations on the SST to implement SHE. It is designed to equip "push", "pop", and "steal" instructions, together with additional instructions such as "copy", "save", "flush-out" and others. The SST machine can be used to realize SHE-based parallel systems, together with use of evaluators for expressions that reside in tasks created by SST machine operations. The SST machine instructions are implemented as C libraries to allow their use in various platforms. Then, we report their experimental results, including their use in implementing a parallel Scheme system.

1. はじめに

並列プログラミング言語には，並列実行を記述するための並列構文があり，その処理系は並列構文の引数式を実行するタスクを生成し，生成されたタスクがプロセッサによって実行されるように実現される．並列言語の処理系を実現するのによく用いられる ETC（Eager Task Creation）は，プログラム中の並列構文に遭遇すると，ただちに，その引数式を実行するタスクをすべてキューに生成する．並列 Lisp のように再帰的並列プログラムが許される並列言語の場合には，多数のタスクが生成され，タスク生成コストがプログラムの全実行コストに占める割合が大きくなるという問題が発生する．並列 Lisp における再帰的並列プログラムの並列実行に ETC を用いた場合，タスク生成によるオーバヘッドのため逐次実行よりも並列実行の方が遅くなることがたびたび発生し，これは ETC のタスク過剰生成問題として知られている．

† 東北大学大学院情報科学研究科
Department of Computer and Mathematical Sciences, Graduate School of Information Sciences, Tohoku University

スティール評価法（Steal-Help Evaluation）は，共有メモリ型並列計算機環境において，実行効率の良い並列 Scheme 処理系を実現するために提案された並列評価法である[8)~10)]．逐次 Scheme[14)] に並列構文を加えて設計された並列 Lisp 言語 PaiLisp の処理系[13)] に利用され，優れた実行性能を有する処理系の実現が行えることが示されている．

スティール評価法の基本的なアイディアは次のようなものである．並列構文の引数式として与えられる並列評価したい式は，SST（Stealable Stack）と呼ぶスタックに入れられ，ホームプロセッサは SST のトップ（top）から式を順次取り出して評価を進める．並列計算機環境における空きプロセッサは，ヘルパー（Helper）となって SST のボトム（bottom）から式をスティール操作によって取り出してホームプロセッサと同様の実行を行う．タスク生成が行われるのは，空きプロセッサがスティール操作を行ったときのみであるから，スティール評価法はタスク過剰生成問題が発生しない並列評価法となる．

スティール評価法は SST に対する操作によって実現されるタスク生成系と生成されたタスクの実行系に分離できる．このことに着目し，スティール評価法を実現するためのタスク生成系としての仮想並列マシン——SST マシン——を提案し，設計する．SST マシンは，SST に対する push，pop，steal の基本操作に加えて，SST の内容の除去・参照・保存の操作，SST の内容の flush-out，コピー操作を行う命令などを備える仮想マシンである．タスク実行系は，並列構文の引数式の実行系であり，それは引数式が書かれている言語の処理系——通常，逐次言語処理系——である．

本論文では，2 章において，スティール評価法とそのタスク生成系を実現する SST マシンの考え方について説明する．3 章では，SST マシンの設計と SST マシンによるプログラミング例を与える．4 章では，SST マシンを C 言語によって実現した SST マシンライブラリの概要について述べ，さらに，SST マシン命令のライブラリ関数の並列計算機での実験データと，SST マシンライブラリを用いて実現された並列 Scheme コンパイラの実験結果について述べる．5 章では，関連研究との比較や今後の課題について述べる．6 章は結言である．

2. スティール評価法と SST マシンの考え方

本章では，スティール評価法の考え方を述べたのち，スティール評価法による並列構文の実現を行うためのタスク生成系である SST マシンの基本的な設計方針について説明する．

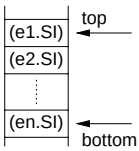

図 1 SST の構造
Fig. 1 A structure of SST.

2.1 スティール評価法の概要

スティール評価法のアイディアは，並列 Scheme の効率の良い処理系を共有メモリ型並列計算機上に実現するために提案された[8),9)]．

関数適用 $(e_f\ e_1 \ldots e_n)$ の並列化である並列関数適用 (**pcall** $e_f\ e_1 \ldots e_n$) のスティール評価法による並列実行について考える．なお，この式は，引数式$e_1, \ldots, e_n$を並列に評価し，すべての引数式の評価を終えてから，式 e_f を評価して得られた関数を引数式$e_1, \ldots, e_n$の評価値を用いて呼び出し関数適用を行うという意味を持つとする．**pcall** 式のスティール評価法による並列評価は，次のように行われる☆．**pcall** 式を評価したプロセッサはホームプロセッサとなり，引数式を$e_n, e_{n-1}, \ldots$の順に，式を正しく評価するための環境などからなる共有情報（Shared Information：SI）とペアにして SST のトップから入れる（push 操作）[**図 1** 参照]．その後，ホームプロセッサは，自身の SST のトップから式を取り出し（pop 操作）て評価する．空き状態のプロセッサがあると，それはホームプロセッサの SST のボトムから式と共有情報を取り出し（steal 操作）て，その式を共有情報に含まれる環境情報などを用いて評価する．引数式が評価されるたびに，その値をメモリ領域に保存し，（未評価の引数式の数に設定される）カウンタを 1 だけ減らす．カウンタが 0 になるとホームプロセッサが式 e_f を評価し，得られた関数をメモリ領域に保存されている引数式$e_1, \ldots, e_n$の評価値を用いて呼び出す．共有情報には，評価の対象となる式に現れる自由変数の値が束縛されている環境，未評価の引数式の数などが記録される．

上述のように，スティール評価法では，ホームプロセッサは，SST のトップから式を取り出して計算を進める．空きプロセッサが発生すると，それは式を SST

☆ (**pcall** $e_f\ e_1 \ldots e_n$) の ETC による並列実行の場合には，まず，$e_1, \ldots, e_n$を評価するタスクがすべてキューに入れられる．キューのタスク（$e_1, \ldots, e_n$）が順次空きプロセッサに割り当てられて実行され，それらの実行が終了してから e_f の評価が行われ，得られた関数が引数式$e_1, \ldots, e_n$の値を用いて呼び出され関数適用される．

のボトムから（ホームプロセッサと並行に）スティールし，その式を計算するタスクを生成し実行する．SSTにある未評価の式を評価するタスクの生成は空きプロセッサが発生するまで遅延されるため，タスクの遅延生成による並列処理が行われる．

ETCは，式を並列に計算するタスクをキューにすべて生成してからタスクをプロセッサに割り付ける評価法である．空きプロセッサが発生しない場合でもタスクをキューに生成するため，タスク生成のオーバヘッドが生じ，特に再帰的に多数の並列タスクを生成する再帰的並列プログラムの場合には，コストが大きくなり，逐次実行よりも遅くなることが多い☆．一方，スティール評価法によって再帰的並列プログラムの実行を行う場合，ホームプロセッサが再帰的に並列構文の処理を行い，空きプロセッサが発生したときにのみスティール操作によるタスク生成がなされ並列実行が行われる．空きプロセッサがなければ，ホームプロセッサによる逐次計算が行われる．スティール評価法による並列実行では，ホームプロセッサは逐次実行を進め，並列タスクの生成は空きプロセッサが発生するまで遅延される．したがって，過剰なタスク生成が生じない，効率の良い並列評価が行われる．なお，文献10)には，スティール評価法による並列実行が逐次実行よりも遅くなることがない条件も与えられている．

SSTを用いたスティール評価法によって**pcall**だけでなく，並列Schemeにおける**par**，**plet**，**pif**，**pcond**，**par-or**，**par-and**，**future**，**stealable**などの各種並列構文が実現できることが知られている[8),9),13)]．スティール評価法に類似したLTC（Lazy Task Creation）は**future** 構文にのみ有効な評価法である．LTCについては，5章でも言及する．

2.2 SST マシンの考え方

スティール評価法による並列実行を，SSTを用いて行うタスク生成系と，生成されたタスクを実行するタスク実行系に分離してとらえることができる．スティール評価法によるタスク生成系をSSTに対するpush，pop，stealなどの操作を有する仮想マシンとして設計したのがSSTマシンである．並列構文によって指定された並列タスクの生成は，SSTマシン命令を用いて，スティール評価法のためのタスク遅延生成を行うようにプログラムされる．並列構文の引数として現れる個々の式を計算するタスクを実行するのがタスク実行系である．タスク実行系は，引数式が書かれている言語の処理系——通常，逐次言語処理系——である．

図 2 SST マシンに基づく並列システム
Fig. 2 A parallel system based on the SST machine.

タスク生成系であるSSTマシンとタスク実行系を用いた共有メモリ型並列システムの構成を**図 2** に与えた．SSTマシンを用いてスティール評価法に基づくタスク生成が行われる．SSTマシンは，並列構文によって指定された並列タスクの生成を行う仮想並列マシンである．SSTマシンとタスク実行系からなるサブマシンがヒープ領域を共有する並列システムが図2に示したSSTマシンに基づく並列システムである．

再帰的並列プログラムの実行において現れる再帰的タスクの生成と処理も効率良く行うために，SSTマシンを次のような命令を備えたものとして設計する．

(1) SSTに対するpush操作，pop操作，steal操作を行う命令
(2) SSTのポインタ操作に関する命令
(3) SSTの要素の除去，参照，保存，コピー，ロックとその解除を行う命令
(4) SSTの要素の一括除去を行うflush-out命令

最後のflush-out命令は，**par-and**や**par-or**における投機計算を効率良く実現するのに有効であり，文献12)において示唆されたものである．

逐次言語に並列構文を導入して設計された並列言語の場合，並列構文によって指定されたタスク生成はSSTマシンを用いて実現され，生成されたタスクは逐次言語処理系によって実行される．この場合，逐次言語処理系がタスク実行系となる．スティール評価法による並列実行をタスク生成系とタスク実行系に分離して実現する方法は，共有メモリ型並列アーキテクチャにおける並列言語処理系の新しい実現法である．

以上の設計方針の下に，次章ではSSTマシンの詳細設計を与える．

3. SST マシンの設計

スティール評価法に基づくタスク生成を行う仮想並列マシンであるSSTマシンの命令の具体的な設計を行う．SSTを用いてタスクの生成を行うタスク生成系の命令の詳細設計を3.1節〜3.4節に与える．複数の

☆ フィボナッチ関数を**pcall** を用いて並列化した（pfib 20）をPaiLisp/MTのETCモードで，6プロセッサを用いて並列実行した場合には，21,890個のタスクが生成され，実行時間が0.767[sec]であるのに対し，1つのタスクの実行を行う逐次モードの場合には0.655[sec]という結果が報告されている[13)]．

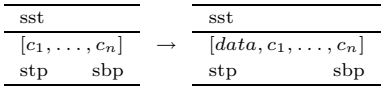

図 3 SST に対する push 操作
Fig. 3 A "push" operation on SST.

サブマシンが同時に 1 つの SST を操作しても不具合が生じないように，各命令は SST に対して不可分な操作を行うと仮定する．3.5 節には，SST マシン命令を用いたプログラミング例を与える．

3.1 SST に対する基本操作を行う命令

SST に対する基本操作として push 操作，pop 操作，steal 操作がある．ここでは，SST に対する基本操作を行う命令を与える．

SST は，図 1 のように式（e_i）と共有情報（SI）のペアが積まれたスタック構造をしている．以下の説明では，SST のトップの要素を c_1，SST のボトムの要素を c_n とする SST の構造を $[c_1, \ldots, c_n]$ と記し，[] は要素がない構造体であるとする．stp を SST のトップの要素が入れられている位置を指すポインタ，sbp を SST のボトムの要素が入れられている位置を指すポインタとする．また，SST マシン命令のオペランドを ⟨ と ⟩ で囲んで表記する．

SST に対する push 操作

SST に対する push 操作は，評価したい式を SST のトップ要素となるように置く操作であり，ホームプロセッサとなったサブマシンが行う操作である．push 操作を行う命令を次のように与える．

- SST_push(⟨sst⟩, ⟨data⟩)

この命令は，**図 3** に示す操作を行う．SST_push 命令は，この命令を実行しているサブマシンの SST である⟨sst⟩と push するデータ⟨data⟩をオペランドとし，⟨sst⟩のトップの上にデータ⟨data⟩を置き，新たに置かれたデータ⟨data⟩を指すようにトップポインタが指す位置を 1 つ上げて⟨sst⟩の状態を変える．

SST に対する pop 操作

SST に対する pop 操作は，ホームプロセッサが SST の式を評価するために SST のトップから式を取り出す操作であり，その命令を次のように与える．

- SST_pop(⟨sst⟩, ⟨reg⟩)

この命令は，SST に対して，**図 4** の操作を行う．SST_pop 命令は，この命令を実行しているサブマシンの SST である⟨sst⟩と pop 操作によって取り出した要素を代入するレジスタ⟨reg⟩をオペランドとする．⟨sst⟩の中に要素がある場合（図 4 の 1 の場合）には，トップの要素をレジスタ⟨reg⟩に入れ，⟨sst⟩からトップの要素を除いてトップポインタが指す位置を 1 つ下げる．

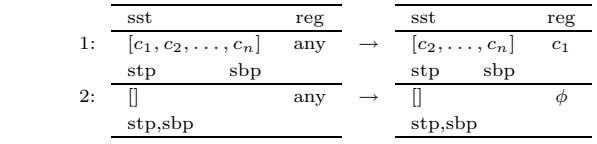

図 4 SST に対する pop 操作
Fig. 4 A "pop" operation on SST.

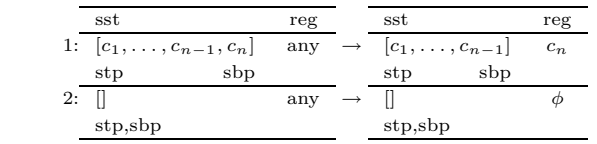

図 5 SST に対する steal 操作
Fig. 5 A "steal" operation on SST.

⟨sst⟩の中に要素がない場合（図 4 の 2 の場合）には，レジスタ⟨reg⟩を空にする．

SST に対する steal 操作

SST に対する steal 操作は，空き状態のプロセッサがホームプロセッサの SST のボトムから式を取り出す操作であり，その命令を次のように与える．

- SST_steal(⟨sst⟩, ⟨reg⟩)

この命令は，SST に対して，**図 5** の操作を行う．SST_steal 命令は，空き状態のサブマシン（ヘルパー）が実行し，スティールの対象となる他のサブマシンの SST である⟨sst⟩とスティールした要素を入れるヘルパーのレジスタ⟨reg⟩をオペランドとする．⟨sst⟩の中に要素がある場合（図 5 の 1 の場合）には，ボトムの要素をレジスタ⟨reg⟩に入れ，⟨sst⟩からボトムの要素を除き，ボトムポインタが指す位置を 1 つ上げる．⟨sst⟩の中に要素がない場合（図 5 の 2 の場合）には，⟨reg⟩を空にする．

3.2 SST ポインタに関する命令

SST のポインタを用いて SST マシン内にある任意の SST を操作できるようにするためには，対象とする SST とその要素を指すポインタの位置の情報が不可欠である．それらのペアを SST ポインタと呼び，SST ポインタを取り出す命令を次のように与える．

- SST_palloc(⟨sst⟩, ⟨reg⟩)

SST_palloc 命令は，⟨sst⟩の中に要素がある場合，トップ要素が入れられている位置 stp と⟨sst⟩のペア（stp . ⟨sst⟩）である SST ポインタをレジスタ⟨reg⟩に入れる．⟨sst⟩が空の場合，レジスタ⟨reg⟩を空にする．

3.3 SST の要素の参照・除去・保存・コピー・ロックを行う命令

SST の内容の参照・除去操作

SST にある式を評価の優先度が高い順に取り出し

て評価したり，評価が不要な式を除去したりすることを実現するために，SST に対する参照と除去の 2 種類の命令を導入する．SST の任意の位置の要素を参照する命令を次のように与える．

- SST_ref(⟨sst⟩, ⟨m⟩, ⟨reg⟩)
- SST_refp(⟨ssp⟩, ⟨reg⟩)

SST_ref 命令は，⟨sst⟩の要素が⟨m⟩個以上ある場合には，⟨sst⟩のトップから⟨m⟩番目の要素をレジスタ⟨reg⟩に入れ，⟨sst⟩の要素が⟨m⟩個より少ない場合には，レジスタ⟨reg⟩を空にする．

SST_refp 命令は，SST ポインタ⟨ssp⟩が (loc . sst) であるとき，位置 loc が sst のボトムポインタが指す位置以上でトップポインタが指す位置以下の範囲（以降は，この範囲を**有効範囲**と呼ぶ）にある場合，loc の位置にある要素をレジスタ reg に入れる．loc が sst の有効範囲外の場合，レジスタを空にする．

SST の要素を除去する次の命令を導入する．

- SST_del(⟨sst⟩, ⟨m⟩, ⟨reg⟩)
- SST_delp(⟨ssp⟩, ⟨reg⟩)

SST_del 命令は，⟨sst⟩の要素が⟨m⟩個以上ある場合，⟨sst⟩のトップから⟨m⟩番目の要素をレジスタ⟨reg⟩に入れ，その要素があった位置を空にする．⟨sst⟩の要素が⟨m⟩個より少ない場合，レジスタ⟨reg⟩を空にする．

SST_delp 命令は，SST ポインタ⟨ssp⟩が (loc . sst) であるとき，位置 loc が sst の有効範囲内にある場合，loc の位置にある要素をレジスタ⟨reg⟩に入れ，loc の位置の内容を空にする．loc が sst の有効範囲外の場合，レジスタ⟨reg⟩を空にする．

SST に対する保存操作

Scheme 言語のようなガーベジコレクション（GC）を備えた言語の場合，GC の処理において SST の要素をライブオブジェクトとして扱うときに，SST の任意の位置にある要素の参照や SST の任意の位置への保存が有用である．参照は，上述の参照命令で行える．保存命令として，次の命令を導入する．

- SST_save(⟨sst⟩, ⟨m⟩, ⟨data⟩)
- SST_savep(⟨ssp⟩, ⟨data⟩)

SST_save は，⟨sst⟩の要素が⟨m⟩個以上ある場合，⟨sst⟩のトップから⟨m⟩番目の位置にデータ⟨data⟩を入れる．

SST_savep は，SST ポインタ⟨ssp⟩が (loc . sst) であるとき，位置 loc が sst の有効範囲内にある場合，データ⟨data⟩を sst の loc の位置に入れる．

SST の内容のコピー操作

Scheme 言語では，ある時点の残りの計算を表すコンティニュエーションがファーストクラスのオブジェクトとして扱える．PaiLisp では Scheme の逐次コンティニュエーションを並列世界に拡張した $\mathcal{P}$-コンティニュエーション[11] を扱う必要がある．SST を用いて実現されるスティール評価法では，コンティニュエーションは関数呼び出しの戻り番地を保存するスタックの内容とホームプロセッサによって評価される SST の内容であると見なせる．

コンティニュエーションをファーストクラスとして扱うため，SST の内容のコピー（copy）と復元（restore）を行う次の命令を導入する．

- SST_copy(⟨sst⟩, ⟨reg⟩)
- SST_restore(⟨sst⟩, ⟨copy⟩)

SST_copy は，⟨sst⟩の内容のコピーを作成してレジスタ⟨reg⟩に入れる．SST_restore は，SST のコピー⟨copy⟩の内容を⟨sst⟩に復元する．

GC において，SST のコピーの要素をライブオブジェクトとして扱うときには，SST のコピーの要素の参照や SST のコピーへの保存が有用である．SST のコピーの参照・保存命令として次の命令を与える．

- SST_copy_ref(⟨copy⟩, ⟨m⟩, ⟨reg⟩)
- SST_copy_save(⟨copy⟩, ⟨m⟩, ⟨data⟩)

SST_copy_ref は，SST のコピー⟨copy⟩の要素数が⟨m⟩個以上ならば，⟨copy⟩のトップから⟨m⟩番目の位置にある要素をレジスタ⟨reg⟩に入れ，⟨copy⟩の要素数が⟨m⟩個より少なければ，レジスタ⟨reg⟩を空にする．

SST_copy_save は，SST のコピー⟨copy⟩の要素数が⟨m⟩個以上ならば，⟨copy⟩のトップから⟨m⟩番目の位置にデータ⟨data⟩を入れる．⟨copy⟩の要素数が⟨m⟩個より少なければ，データ⟨data⟩を無視する．

SST に対するロック操作

SST に対する参照命令が，除去などの他の命令と不可分に実行されるように，SST に対してロックを行う次の命令を導入する．

- SST_lock(⟨sst⟩)
- SST_lockp(⟨ssp⟩)

SST_lock 命令は，⟨sst⟩に対してロックをかける．

SST_lockp 命令は，SST ポインタ⟨ssp⟩が (loc . sst) であるとき，sst に対してロックをかける．すでにロックした SST に対してロック命令を実行した場合には，そのロック命令は無視される．

SST に対するロックを解除する次の命令を導入する．

- SST_unlock(⟨sst⟩)
- SST_unlockp(⟨ssp⟩)

SST_unlock 命令は，⟨sst⟩に対するロックを解除する．

SST_unlockp 命令は，SST ポインタ⟨ssp⟩が (loc .

sst) であるとき，sst に対するロックを解除する．ロックされていない SST に対してロックの解除命令を実行した場合には，そのロック解除命令は無視される．

3.4 SST の flush-out 命令

並列論理積式や並列論理和式，さらに，それらが入れ子状になっている場合に，これらの並列式を評価し，ある引数式が真または偽であることが分かると，残りの引数式を評価する必要がなくなる．スティール評価法に基づく評価では，引数式は SST に入れられるから，次のような場合が発生する．

- SST マシンが実行する計算が途中で強制終了され，その時点で SST のすべての式の評価が不要になった場合
- SST マシンが実行する計算がある計算のポイントまで戻され，それまでに入れられた SST の式の評価が不要になった場合

このような場合，SST から 1 つずつ式を除去するよりも，一括して除去した方が効率が良い．SST の要素を一括除去する次の flush-out 命令を導入する．

- SST_flush(⟨sst⟩, ⟨m⟩)
- SST_flushp(⟨ssp⟩)
- SST_flushall(⟨sst⟩)

SST_flush は，⟨sst⟩の要素数が⟨m⟩個以上の場合，トップから⟨m⟩個の要素を一括除去し，⟨sst⟩の要素数が⟨m⟩個より少ない場合，SST のすべての要素を除去する．

SST_flushp は，SST ポインタ⟨ssp⟩が (loc . sst) であるとき，loc が sst の有効範囲内にある場合，loc より上の全要素を一括除去する．loc が sst のボトムより下の位置である場合，sst の全要素を一括除去する．

SST_flushall は，⟨sst⟩のすべての要素を一括除去する．

3.5 SST マシンによるプログラミング例

図 2 の SST マシンに基づく並列システムのサブマシンが UNIX の下で動作する SST マシンとタスク実行系からなると仮定して，SST マシン命令を用いたプログラミングを 2.1 節で説明した並列関数適用**pcall** を例として考える☆．この例では，並列実行されるタスクは逐次 Scheme の式であるとし，式 exp を環境 env の下で逐次評価するタスク実行系 EVAL(exp, env) が使えるものと仮定する．

```
1:  obj_t pcall_args_ev(obj_t argl, obj_t env)
2:  {
3:    int c = length(argl), i;
4:    void* share_info[] = {env, c};
5:    obj_t tmp = argl; sobj_t obj[c];
6:    for (i = 0; i < c; i++) {
7:      obj[i].exp = tmp;
8:      obj[i].share_info = share_info;
9:      tmp = cdr(tmp);
10:   }
11:   while ((c−−) > 0) SST_push(SST[SM], obj + c);
12:   pcall_home_eval(share_info);
13:   return argl;
14: }
```

図 6 **pcall** 式による並列評価の初期化
Fig. 6 An initialization of **pcall**.

プログラムは C 言語の枠組みで書くが，いくつかの補助関数，構造体，型を追加して用いる．説明を簡潔にするため，大域変数や共有情報などに要求される不可分な操作についての処理や説明は省略している．

まず，(**pcall** e_f $e_1 \ldots e_n$) の引数式と共有情報を SST_push の繰返しによって SST に入れる初期化フェーズを与える．その後，ホームプロセッサが SST から SST_pop によって式と共有情報を取り出して EVAL によって評価するフェーズ，空きプロセッサであるヘルパーが式と環境を SST_steal によって取り出して EVAL によって評価するフェーズについて考える．

(1) **pcall** の初期化

pcall の引数式を，$e_n, e_{n-1}, \ldots$の順に，SST_push 命令を用いて SST に入れて行く操作が，**図 6** に与えた**pcall** の初期化のフェーズである．pcall_args_ev は，サブマシン SM の SST である SST[SM] に引数リスト argl の要素を SST_push によって入れていく C プログラムの例である．図 6 において，obj_t は SST マシンとタスク実行系がやりとりをするデータの型，sobj_t は SST に入れられる式 e_j と共有情報 share_info からなる構造体の型である．また，length(argl) は引数式の個数であるリスト argl の長さ，cdr(L) はリスト L から最初の要素を取り除いたリストを返すとする．

(2) **pcall** のホームプロセッサによる評価

ホームプロセッサは，SST_pop を用いて引数式と共有情報のペアである要素を取り出して EVAL によって逐次的な評価を進める．ただし，引数式には**pcall** のような並列構文が現れる可能性も許されるように EVAL の中からタスク生成系を呼び出せるものとする．

図 7 に示した pcall_home_eval は，サブマシン SM の SST である SST[SM] から要素（&objp：変数 objp のアドレス）を SST_pop によって取り出し，タスク実行系 EVAL を呼び出して評価を進める C プログ

☆ 並列 Scheme に現れる**par**，**plet**，**par-or**，**par-and** などの例については，文献 16) の SSTM ホームページを参照されたい．なお，引数式が副作用を起こす式の場合には，ETC やスティール評価法による並列実行の場合，逐次実行と異なる結果を与える可能性があるからプログラミングにあたって注意する必要がある．

```
1:  void pcall_home_eval(void **share_info)
2:  {
3:    sobj_t *objp; obj_t val;
4:    do {
5:      SST_pop(SST[SM], &objp);
6:      if (objp) {
7:        val = EVAL(car((*objp).exp), share_info[0]);
8:        set_car((*objp).exp, val);
9:      } else {
10:       while (share_info[1]) pcall_steal_ev();
11:       return;
12:     }
13:   } while (−−share_info[1]);
14:   return;
15: }
```

図 7 **pcall** の引数式のホームプロセッサによる評価
Fig. 7 Evaluations of arguments of **pcall** by a home processor.

```
1:  void pcall_steal_ev(void)
2:  {
3:    int i;
4:    sobj_t *objp; obj_t val, void **share_info;
5:    for (i = 1; i < SMN; i++) {
6:      SST_steal(SST[(SM + i) % SMN], &objp);
7:      if (objp) {
8:        share_info = (*objp).share_info;
9:        val = EVAL(car((*objp).exp), share_info[0]);
10:       set_car((*objp).exp, val);
11:       share_info[1]−−;
12:       break; }}
13:   return;
14: }
```

図 8 **pcall** の引数式のヘルパーによるスティール評価
Fig. 8 A steal evaluation of an argument of **pcall** by a helper.

ラムの例である．EVAL によって得られた結果は，set_car によって引数リストに上書きされ，そのたびに共有情報の未評価引数式の数が 1 つ減らされる（−−share_info[1]）．SST_pop によるこの処理は，未評価式の数が 0 でない間（while (−−share_info[1])）繰り返される．ただし，この繰返しの過程で，SST の式がヘルパーによってスティールされ，ホームプロセッサが SST からポップする式がない場合には，ホームプロセッサ自身もヘルパーとなり，他のサブマシンの SST から式を取り出してスティール評価を行う**図 8** の pcall_steal_ev を実行する．なお，図 7 において，car(L) はリスト L の最初の要素を返し，set_car(L, sym) はリスト L の最初の要素を sym に置き換える関数である．

(3) **pcall** のヘルパーによるスティール評価

空きサブマシンは，ヘルパーとなって（SMN−1）台ある他のサブマシンの SST から式と共有情報をスティール操作によって取り出し，タスク生成を行う．図 8 に示した pcall_steal_ev は，SST_steal によって ((SM+i) % SMN) 番目のサブマシンの SST から式を取り出して EVAL によって評価し，得られた値を引数リスト argl に上書きし，共有情報の未評価式の数を share_info[1]−− によって 1 つ減らすプログラムである．SMN はサブマシン台数であり，(_ % SMN) は SMN に関するモジュロ演算である．

以上のような評価の後，共有情報の未評価式の数が 0 になると，ホームプロセッサは argl にある引数式の値リストを，pcall_args_ev を呼び出したタスク実行系に返す．タスク実行系は，e_f を評価して得られた関数を引数の値リストを用いて呼び出し関数適用を行う．

4. C 言語による SST マシンライブラリとその利用

SST マシンは，スティール評価法を用いた並列言語処理系におけるタスク生成系を作成するのに有用な仮想並列マシンである．本章では，SST マシンを UNIX 上に C 言語を用いて実現した SST マシンライブラリについて説明する．SST マシンライブラリは，様々な UNIX プラットフォーム上で移植性が良いように設計・実装された，SST マシンの機能を提供する C ライブラリである．その概要を 4.1 節で説明した後，4.2 節で 3 つの並列計算機での実験評価データを与える．4.3 節では SST マシンライブラリを利用して作成された並列 Scheme コンパイラの実験結果を述べる．

4.1 SST マシンライブラリの概要

SST マシンを用いた並列システムは，図 2 に示したように，タスク生成系である SST マシンとタスク実行系からなるサブマシン間でメモリを共有する．UNIX を備えた共有メモリ型並列計算機上で，SST マシンに基づく並列システムを 1 つの UNIX プロセス上に実現し，各サブマシンの実現にスレッドを割り当てる．スレッドとして実現されたサブマシンは UNIX プロセスの実行環境を共有し，すべてのサブマシンがメモリ領域（ヒープ領域）を共有する．なお，SST が複数のサブマシンから同時に操作される可能性があるから，SST に対する排他制御が実装されている．

SST マシンの機能を UNIX 上での C ライブラリとして提供するため，SST 構造体を実現したうえで，SST マシン命令を C ライブラリとして実現した．空きプロセッサによるスティール操作を自動的に行うスケジュールされたスティール命令実行関数を導入しているのが 1 つの特徴である．

```
struct sst {
 data_t   *stp;     // top pointer.
 data_t   *sbp;     // bottom pointer.
 data_t   tslot[];  // slots pointed by a top pointer.
 data_t   bslot[];  // slots pointed by a bottom pointer.
 int      slots;    // number of slots.
 int      sm;       // name of a locked submachine.
 mutex_t mutex;} // mutex variable.
```

図 9 SST の C 構造体
Fig. 9 A C structure of SST.

(1) SST の実現

SST は，要素を入れるスロット列，トップポインタ，ボトムポインタからなる．SST のスロット列を C 言語の配列として実現し，トップポインタはトップの要素が入れられているスロットを指す C のポインタに，ボトムポインタはボトムの要素が入れられているスロットを指す C のポインタに対応させる．

SST のスロット列がオーバフローしたときには，そのスロット列と同じ長さの新しいスロット列を確保し，元のスロット列と相互リンクして拡張するように実現されている．SST のスロット列の長さは，なるべくリンクをたどることがないように，また，メモリの無駄使いにならないように，100 に設定しているが，プログラムの性質によって変更可能としている．

SST は sst_t 型の構造体として**図 9** のように実現されている．なお，sm はロック命令によって SST をロックしたサブマシンの名前であり，mutex は SST を排他的に操作するための排他変数である．図 9 の**data_t 型**は，式と共有情報の任意の型のペアを SST に入れることができるように，void ポインタ型としている．**mutex_t 型**は，排他変数の型である．

SST ポインタは，次の要素を持つ**ssp_t 型**の構造体として実現されている．

- 操作対象の位置を指すポインタ（loc）
- loc が指す SST のスロット列（slot[]）
- 操作の対象となる SST 構造体（sst）

SST_copy によって保存されたコピーを SST に復元するときのために，SST のコピーを次の要素からなる **csst_t** 型の構造体として実現している．

- SST の要素を保存するためのスロット列（slot[]）
- SST のトップポインタが指すスロット列（tslot[]）
- SST のトップポインタ（stp）
- コピーした要素の数（num）

SST_lock と SST_lockp を実行したサブマシンの名前は，図 9 の SST 構造体の sm フィールドに入れられる．sm フィールドにサブマシンの名前が入れられている限り，その SST に対する他のサブマシンによる操作は待たされる．sm フィールドに入れられたサブマシンの名前は，SST_unlock や SST_unlockp によって除去される．

(2) SST マシン命令のライブラリ関数

SST マシン命令を C ライブラリとして利用できるようにするため，各命令を C ライブラリ関数として実現した．SST マシン命令は，**int 型**の整数を返すようにし，SST に対する操作が成功したか失敗したかを返り値（1 または 0）により判別できるようにした．命令を実行したときに 0 を返すのは次の場合である．

- SST_push：SST のスロット列のオーバフロー時に新しい SST のスロット列の確保に失敗した場合．
- SST_pop，SST_steal：SST が空であった場合．
- SST_palloc：SST ポインタのためのメモリの確保に失敗するか SST が空であった場合
- SST に対する参照，除去，保存の命令：SST の有効範囲外の位置に対して，それぞれ，参照，除去，保存の操作を行った場合
- SST_copy：SST のコピーのためのメモリの確保に失敗した場合
- SST_copy_ref：SST のコピーに対する参照に失敗した場合
- SST_copy_save：SST のコピーへの保存に失敗した場合

上記以外の場合，返り値が int 型のすべての命令は，SST に対する操作に成功したとして 1 を返す．SST に対する操作が必ず成功し，その返り値が必要ない命令に関しては，返り値を**void 型**にする．

SST マシン命令のオペランドに，上述の型情報を加えて設定したライブラリ関数を**表 1** に与える．

(3) スティール操作とスケジューリング

SST_steal が実行されると，オペランドの SST のボトムから要素が取り出される．空き状態のサブマシンが，他のサブマシンの SST から式が得られるまでスティール操作を行うように，スケジューリングを行う必要がある．3.5 節の例では，空きプロセッサのスケジューリングを for ループを用いて記述した．このスケジューリングを簡単に記述するため，SST マシンに基づく並列システム sstm にあるサブマシンの SST から自動的にスティールを行う**表 2** の関数を導入する．SSTM_steal は，並列システム sstm の中の sst を有するサブマシン以外のサブマシンの SST から要素をスティールして 1 を返す．すべての SST が空の場合には，SSTM_steal はいずれかの SST からスティールできるまで待つ．一方，SSTM_checksteal は，SSTM_steal と同様の動作を行うが，すべての SST が空の場合，レジスタ reg に NULL ポインタを代入して 0 を返す．

表 1 SST マシン命令のライブラリ関数
Table 1 Library functions of the SST machine.

int SST_push(sst_t sst, data_t data)
int SST_pop(sst_t sst, data_t* reg)
int SST_steal(sst_t sst, data_t* reg)
int SST_palloc(sst_t sst, ssp_t* reg)
int SST_ref(sst_t sst, int m, data_t* reg)
int SST_refp(ssp_t ssp, data_t* reg)
int SST_del(sst_t sst, int m, data_t* reg)
int SST_delp(ssp_t ssp, data_t* reg)
int SST_save(sst_t sst, int m, data_t data)
int SST_savep(ssp_t ssp, data_t data)
int SST_copy(sst_t sst, csst_t* reg)
void SST_restore(sst_t sst, csst_t copy)
int SST_copy_ref(csst_t copy, int m, data_t* reg)
int SST_copy_save(csst_t copy, int m, data_t data)
void SST_lock(sst_t sst)
void SST_lockp(ssp_t ssp)
void SST_unlock(sst_t sst)
void SST_unlockp(ssp_t ssp)
void SST_flush(sst_t sst, int m)
void SST_flushp(ssp_t ssp)
void SST_flushall(sst_t sst)

表 2 スケジュールされたスティール命令実行関数
Table 2 Functions of scheduled stealing operations.

int SSTM_steal(sstm_t sstm, sst_t sst, data_t* reg)
int SSTM_checksteal(sstm_t sstm, sst_t sst, data_t* reg)

並列システム sstm に SST が$\mathrm{SST}_0, \ldots, \mathrm{SST}_{n-1}$の順に n 個存在し，SST_iを有するサブマシンが SSTM_steal を実行すると，次の処理が行われる．

(a) $\mathrm{SST}_{i+1}, \ldots, \mathrm{SST}_{n-1}, \mathrm{SST}_0, \ldots, \mathrm{SST}_{i-1}$ の順に，SST_steal を繰り返す．

(b) 要素をスティールできた場合には，その要素をレジスタ reg に入れ，実行を終了する．

ただし，すべての SST が空のとき，SSTM_steal における (a) の処理を中断する必要が生じる場合には SSTM_checksteal を用いる☆．

上記のスケジューリングは，すべての SST に均等にアクセスでき，スケジューリングコストが小さい簡便な方法である．なお，SST マシンに基づく並列システム sstm は，すべてのサブマシンの SST を保存する配列を持つ**sstm_t 型**の構造体として実現されている．

スティール評価法を用いた最初の処理系である PaiLisp/MT [13] では，SST に必ず排他的にアクセスし，すべての SST が空である場合には，一定の待ち時間を設け，再度 SST をチェックする方法を採用している．一方，上述の方法では，SST が空のときには（スティールによって SST が変化しないので）SST に対する排他処理のコストが削減されている．

☆ SSTM_checksteal を用いると，図 8 の 5 行目〜6 行目のスティール操作は次のように書ける．
SSTM_checksteal(sstm, SST[SM], &objp)
sstm は表 4 の実行管理関数 SSTM_init により確保された，n 個の SST を有する並列システムを表す構造体であるとする．

表 3 メモリ領域の解放に関する関数
Table 3 Functions for freeing memory area.

void SST_pfree(ssp_t ssp) SST ポインタ ssp のために確保したメモリ領域を解放する．
void SST_copy_free(csst_t copy) SST のコピー copy のために確保したメモリ領域を解放する．

表 4 SST マシンの実行管理に関する関数
Table 4 Functions for administration of the SST machine.

sstm_t SSTM_init() SST マシンに基づく並列システムの実行を開始するために，並列システムを表す構造体を確保して返す．
void SSTM_exit(sstm_t sstm) SST マシンに基づく並列システム sstm の構造体を解放し，並列システムの実行を終了する．
sst_t SST_alloc(sstm_t sstm) SST 構造体を確保・初期化して SST マシンに基づく並列システム sstm に登録し，SST 構造体を返す．
int SSTM_subcreate(void* start, void* a) start(a) を実行するサブマシンのスレッドを開始する．

(4) メモリ管理

SST ポインタを確保する SST_palloc と SST の内容をコピーする SST_copy は，メモリ領域から必要な大きさの構造体，または配列を確保する．SST マシンをコンパクトな C ライブラリとして実現するために，メモリ領域を解放する**表 3** の 2 つの関数を導入している．SST ポインタと SST のコピーは，表 3 の関数を用いて明示的に解放されなければならない．

(5) SST マシンの実行管理

表 4 に，SST マシンに基づく並列システムの実行を開始してから終了するまでの一連の実行を管理する主なライブラリ関数を与えた．共有メモリ型並列計算機環境に合わせた並列実行ができるように，SST マシンに基づく並列システムの実行開始・終了に加えて，サブマシンの追加・削除が行える関数を与えている．表 4 の関数のほか，スレッドの排他処理に関する関数も備えている．

4.2 SST マシンライブラリの実験評価

SST マシンライブラリは，**表 5** の共有メモリ型並列計算機上に実装されている．以降では，Enterprise4500 を EP，PRIMEPOWER600 を PP，TX7/AzusA を TX と略記する．

実装された SST マシンライブラリの各関数の基本コストを**表 6** に与える．SST に対する pop と steal

表 5 SST マシンライブラリが実装されている並列計算機
Table 5 Parallel machines on which the SST machine library is implemented.

マシン名	プロセッサ，メモリ，OS
Enterprise4500	UltraSPARC II 360 MHz×8, 8 GB, Solaris2.6
PRIMEPOWER600	SPARC64GP 450 MHz×8, 2 GB, Solaris8
TX7/AzusA	Itanium 800 MHz×16, 32 GB, RedHat Linux

表 6 SST マシンライブラリ関数のコスト
Table 6 Costs of functions of the SST machine library.

コスト名	関数名	コスト [μ sec]		
		EP	PP	TX
C_{push}	SST_push	0.387	0.134	0.266
C_{pop0}	SST_pop	0.055	0.017	0.038
C_{pop1}		0.314	0.113	0.248
C_{steal0}	SST_steal	0.055	0.017	0.039
C_{steal1}		0.329	0.118	0.256
C_{palloc}	SST_palloc	0.121	0.082	0.066
C_{ref}	SST_ref	0.056	0.033	0.050
C_{refp}	SST_refp	0.092	0.056	0.111
C_{del}	SST_del	0.316	0.124	0.232
C_{delp}	SST_delp	0.384	0.141	0.286
C_{save}	SST_save	0.082	0.038	0.045
C_{savep}	SST_savep	0.154	0.051	0.108
C_{copy}（※）	SST_copy	9.882	7.206	2.815
C_{rest}（※）	SST_restore	2.710	1.133	1.895
C_{cref}	SST_copy_ref	0.040	0.023	0.037
C_{csave}	SST_copy_save	0.037	0.022	0.035
C_{lock}	SST_lock	0.367	0.145	0.245
C_{lockp}	SST_lockp	0.391	0.161	0.273
C_{unlock}	SST_unlock	0.082	0.046	0.055
$C_{unlockp}$	SST_unlockp	0.096	0.047	0.058
C_{flush}	SST_flush	0.308	0.098	0.220
C_{flushp}	SST_flushp	0.314	0.127	0.287
C_{flusha}	SST_flushall	0.392	0.096	0.212
C_{pfree}	SST_pfree	0.046	0.035	0.037
C_{cfree}	SST_copy_free	2.472	1.677	1.772
C_{alloc}	SST_alloc	37.943	24.043	7.336
C_{init}	SSTM_init	1.693	1.007	0.441
C_{exit}	SSTM_exit	0.885	0.605	0.441
$C_{screate}$	SSTM_subcreate	185.976	179.432	119.845

(※)：100 個の要素をコピー，または復元するコスト

の操作に関しては，SST が空の場合には排他処理をせずにただちに実行を終了する実装にしており，そのコストをC_{pop0}とC_{steal0}とし，SST に要素があった場合に，要素を取り出すコストをC_{pop1}とC_{steal1}としている．C_{push}が，C_{pop1}やC_{steal1}などと比べて大きいのは，SST がオーバフローしたときに必要となるスロット列を拡張するコストを加算しているためである．

スレッドライブラリの排他処理のコストC_{mutex}は，EP が 0.28[μ sec]，PP が 0.05[μ sec]，TX が 0.20[μ sec] である．SST を不可分に操作するために，スレッドライブラリの排他処理を利用している命令のコスト（C_{push}，C_{pop1}，C_{steal1}，C_{del}，C_{delp}，C_{copy}，C_{rest}，C_{lock}，C_{lockp}，C_{flush}，C_{flushp}，C_{flusha}）には，C_{mutex}が含まれる．EP や TX の場合には，C_{mutex}の割合が大きい．さらに実行効率の良い SST マシン命令を実現するためには，スレッドの排他処理をなるべく使わない方法[4] が考えられるが，それは今後の課題である．

SST ポインタの確保と解放，SST に対する参照・保存，SST のコピーに対する参照・保存の操作は，排他処理を必要としないので，他の関数と比較して実行コストが小さい．SST の内容のコピー関数 SST_copy と復元関数 SST_restore は，実行時の SST の要素の数に依存する．表 6 には，SST の 100 個の要素をコピーするコストと SST に 100 個の要素を復元するコストを与えている．C_{copy}では，実行時に SST の要素の数の配列を確保する malloc 関数のコストが半分以上を占めている．malloc 関数のコストが小さい TX では，EP や PP よりもC_{copy}が小さくなっている．

SST の内容を flush-out する関数 SST_flush，SST_flushp，SST_flushall は，SST のトップポインタを指定した位置に移動して実現している．その結果，SST から 1 つの要素を除去する関数（SST_del，SST_delp）と同程度の効率が得られている．SST の連続した位置にある複数の要素を除去する場合，SST_del や SST_delp を用いるより flush-out 操作を用いた方が効率が良い．

SST を確保する関数 SST_alloc のコストC_{alloc}とサブマシンを立ち上げる関数 SSTM_subcreate のコスト$C_{screate}$は，表 6 の中で非常に大きい．これらの関数は，SST マシンの実行を開始するときに，サブマシンの数だけ実行されるが，その回数は少ないので，実行コストが大きくても問題は少ない．

4.3 並列 Scheme コンパイラでの実験結果

スティール評価法に基づく並列 Scheme コンパイラを，タスク生成系の実現に SST マシンライブラリを用い，タスク実行系として（C ライブラリとして実現した）Abelson-Sussman のレジスタマシン[1] を用いて作成した．対象とした並列 Scheme は，逐次 Scheme に（**pcall**，**pbegin**，**plet**，**par-and**，**par-or**，**future** などの）並列構文を導入した PaiLisp[13] である．リーダによって読み込まれた並列プログラムは，コンパイル部において SST マシンとレジスタマシンの C ライブラリを用いた C プログラムに変換され，それを C コンパイラによってコンパイルし実行する．並

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

図 10 pfib プログラム
Fig. 10 A pfib program.

表 7 ベンチマークの実行結果
Table 7 Results of executing benchmarks.

ベンチマーク	逐次 [sec]	SHE [sec]	ETC [sec]
pfib(25)	0.464	0.168	1.371
		(277)	(242,784)
ptarai(10,4,0)	0.554	0.156	0.527
		(918)	(203,322)
pqueen(9)	0.418	0.100	0.415
		(407)	(48,022)

列 Scheme コンパイラは，表 5 に示した共有メモリ型並列計算機上で動作している．本節では，並列構文として**pcall** と**par-or** を用いた簡単なプログラムの EP 上での実験結果について報告する．

(1) 並列化の効果

並列タスク生成を SST マシンライブラリによって実現した**pcall** を用いて並列化されたプログラム例として，**図 10** の pfib，文献 13) の ptarai，pqueen を用いた実験結果を**表 7** に与える．表 7 の「逐次」の欄は，並列プログラムから並列構文を除去して逐次実行した実行時間を示し,「SHE」の欄は，**pcall** を用いた並列プログラムを 8 台のプロセッサを用いてスティール評価法によって並列実行した場合の実行時間とスティール動作の回数（括弧内の数値）を表す．なお，比較のために ETC に基づく処理系も実現している．表 7 の「ETC」の欄は ETC による実行時間とタスク生成数（括弧内の数値）を示している．並列マシンによる実行では，負荷によるバラツキがあるので，プログラムを，それぞれ，1,000 回実行したときの平均値を実行時間として記している．

表 7 から,「SHE」の場合は,「逐次」よりも 2.7〜4.1 倍高速になっている．特に，pfib(25) では,「ETC」の場合には多数のタスクが生成され,「逐次」の場合より実行が遅くなるが,「SHE」の場合には「逐次」よりも 3 倍程度高速になっている．

各プログラムをプロセッサの台数を変化させて並列実行したときの「SHE」の実行時間 T_{par} と「逐次」の実行時間 T_{seq} の比 T_{par}/T_{seq} を**図 11** に与える．これらのプログラムに関しては，プロセッサ台数を増すにつれて台数効果が得られている．プロセッサ台数が 3 台以上になると，どのプログラム例でも「逐次」の場合より「SHE」の方が実行時間が小さい．これは，SST マシンライブラリを用いて実現されているタスク生成のオーバヘッドがタスク実行コストに比べて小さいことによると考えられる．

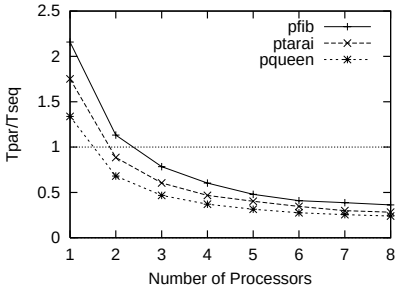

図 11 ベンチマークの台数効果
Fig. 11 Speedup-rates of benchmark programs.

表 8 flush-out の効果
Table 8 An effect of a flush-out operation.

引数	16	17	18	19
逐次	1.210	0.503	2.309	2.448
flush-out あり (a)	0.162	0.124	0.649	0.404
	(905)	(1071)	(1519)	(1621)
	[61]	[69]	[76]	[75]
flush-out なし (b)	0.183	0.145	0.896	0.599
	(0)	(0)	(0)	(0)
	[71]	[80]	[101]	[102]
(b) / (a)	1.13	1.17	1.38	1.48

[sec]

(2) flush-out の効果

SST の flush-out 操作は，**par-or** や**par-and** などの投機計算をともなう並列構文を用いたときに，その効果が現れる．そのようなプログラム例として，**par-or** を用いた一解探索の n クィーン問題を解く porqueen(n) を用いた☆．porqueen(n) は，**par-or** が入れ子状に用いられており，n クィーン問題の解の探索を並列に行い，1 つの解が得られた時点で，他に並列に行われている探索を強制終了させる並列プログラムである．

flush-out の効果を調べるため，flush-out 命令を用いて要素を一括除去する「flush-out あり」，および，SST の要素の除去命令を繰り返して 1 つずつ要素を除去する「flush-out なし」の**par-or** を用いた porqueen(n) の実行時間の比較を**表 8** に与えた．表 8 の「逐次」の欄は，porqueen プログラムの**par-or** を**or** に置き換えて逐次実行したときの実行時間を示す．porqueen(n) は解は非決定的であり，porqueen(n) を実行するたびに評価時間が異なるので，表 8 には 1 万回の実行を

☆ **par-or** は SST マシンライブラリを用いてスティール評価法によって実現されたものを用いている．

した平均実行時間と flush-out された SST の要素の数（括弧 ( ) の中の数値）およびスティール動作の回数（括弧 [ ] の中の数値）を与えた．なお，porqueen に与える引数によっても，flush-out の効果が変わるので，n を 16〜19 に変化させた．SST の中にある評価の必要がない式を一括除去する flush-out を用いた方がスティール回数が削減されている．その結果，「flush-out あり」が「flush-out なし」よりも実行時間が 1.1〜1.5 倍高速になっており，flush-out される要素の数が多いほど，flush-out の効果が大きくなっている．このことから，flush-out 命令が有用なものであることが知られる．

5. 関連研究と今後の課題

SST マシンのほかにも，様々な並列言語の処理方式や仮想マシンに関する研究が行われてきた．スティール評価法を用いた最初の処理系である PaiLisp/MT の場合，タスク生成系とタスク実行系という形に処理系が分離されておらず，逐次 Scheme 処理系を並列システムに拡張する実現法となっている☆．なお，SST マシンは，並列構文に応じてタスクの遅延生成を行い，タスク実行系が逐次 Scheme でなくても，スティール評価法を実現する手段を与えるものである．以下には，関連研究と今後の課題について述べる．

- LTC（Lazy Task Creation）は，ETC におけるタスクの過剰生成を抑制するいう観点から導入された評価法で，スティール評価法を発想する基になった．LTC は，Multilisp[6),7)] の処理系である Mul-T に LTQ（Lazy Task Queue）を用いて実装されている[15)] が，処理系をタスク生成系とタスク実行系とに分離する考えはなく仮想並列マシンという考え方も導入されていない．なお，LTC は**future** 構文にのみ有効な並列評価法であり，**pcall** 構文をはじめとする多様な並列構文を直接実現するのに利用できず，また，LTQ に対する flush-out 操作もない．SST は LTQ の役割も果たすように使用でき，LTC による**future** を SST マシンを用いて効率良く実現することも可能である[8)〜10)]．
- 仮想マシンを用いた並列言語処理系に関する研究としては，StackThreads[17)] や PVM/ESC[2)] がある．これらの研究では，タスク生成系とタスク実行系を兼ね備えた仮想マシンが設定され，それを用いた並列処理系の実現法が与えられている．SST マシンによる並列処理系では，タスク生成系である SST マシンとタスク実行系を分離して実現することができる．このような SST マシンによる並列処理系の実現法は新しい手法であると考える．
- StackThreads[18)] は，細粒度スレッドの効率の良いスケジューリングを実現するための C ライブラリとしても提供されている．StackThreads の C ライブラリは，C スタックを直接操作して並列処理を行うように実装され，効率の良いスレッド処理が実現されている．StackThreads のように C スタックを直接操作する実装ではないが，SST マシンライブラリでは，SST に対する flush 命令によって投機計算を効率良く行うことができ，また，SST に対する操作を柔軟に扱える SST マシン命令が備えられている．
- ネットワーク接続された異機種の計算機を単一の大きな並列計算資源に統合する PVM[5)] や MPI[3)] は，分散処理系の作成にあたり，プロセス間通信を行うライブラリを提供し，SST マシンライブラリよりも汎用的で低レベルな命令を備えている．しかし，プロセスはメモリアドレスを共有せずに通信を行うので，共有メモリ型並列計算機においては，マルチスレッドを用いた SST マシンライブラリほど効率良く並列処理を行えない．このため，共有メモリの利用が必要な言語の高性能処理系の実現には適していない．共有メモリ型並列計算機において並列処理系を実現するために導入された SST マシンを分散メモリ型並列計算機のサブマシンに利用することも考えられる．この場合，他の SST からのスティール操作をプロセス間通信を用いて実現することになるのでスティール操作のコストが大きくなる．スティールされる式の実行コストと通信コストを考慮に入れて通信方式を設計したうえで，分散環境における SST マシンを設計する必要がある．これは，今後の課題である．

6. おわりに

スティール評価法による並列処理は，SST に対する操作によって実現されるタスク遅延生成を行うタスク生成系と，生成されたタスクを実行するタスク実行系に分離できる．本論文では，スティール評価法を実現するためのタスク生成系としての SST マシンを提案し，その詳細設計を与えた．SST マシンは，スティール評価法が SST に対する操作によって実現されることに着目して設計された，タスクの遅延生成の実現に有用な仮想並列マシンである．SST マシンを UNIX 上

☆ PaiLisp/MT コンパイラは，PaiLisp/MT インタプリタの機能をランタイムシステムとして使用する簡便な方法をとっている．PP や EP に移植して比較したところ，4 章で紹介した SST マシンライブラリを用いたコンパイラの方が 5 倍程度も高速なものとなっている．

で C 言語を用いて実現した SST マシンライブラリの概要を述べ，その主なライブラリ関数の実行コストを 3 つの並列計算機上で計測し，比較した．また，SST マシンライブラリとレジスタマシンを用いて作成した並列 Scheme による実験結果についても報告した．これらの実験から，SST マシンとその C ライブラリは実用的にも有用であると考える．なお，SST マシンライブラリは，文献 16) の SSTM ホームページに公開され，利用可能となっている．

逐次言語に並列構文を導入して設計された他の並列言語のスティール評価法に基づく並列処理系も，タスク生成系である SST マシンとタスク実行系である逐次言語処理系を利用して実現できるが，UNIX 環境における実現を想定したツールを整備しておくと便利である．SST マシン命令をさらに効率良くサポートするために，アセンブラ言語による実装や，SST マシン命令を高速に実行できるハードウェアアーキテクチャを実現することも考えられる．これらの詳細な検討や具体化は今後の課題である．

謝辞 本論文の細部にわたってご指摘をいただいた担当委員，および，査読者に深謝の意を表する．

参 考 文 献

1) Abelson, H. and Sussman, G.J.: *Structure and Interpretation of Computer Programs*, MIT Press (1985).
2) Feeley, M. and Miller, J.S.: A Parallel Virtual Machine for Efficient Scheme Compilation, *Proc. 1990 ACM Conference on Lisp and Functional Programming*, pp.119–130 (1990).
3) Forum, M.P.I.: MPI: A Message-Passing Interface Standard, Technical Report (1995).
4) Frigo, M., Leiserson, C.E. and Randall, K.H.: The Implementation of the Cilk-5 Multithreaded Language, *Proc. ACM Conference on Programming Language Design and Implementation*, pp.212–223 (1998).
5) Geist, A., Beguelin, A., et al.: *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press (1994).
6) Halstead, R.H.: Multilisp: A Language for Concurrent Symbolic Computation, *ACM Trans. Prog. Lang. Syst.*, pp.501–538 (1985).
7) Halstead, R.H.: New Ideas in Parallel Lisp: Language Design, Implementation and Programming Tools, LNCS, Vol.441, pp.2–57, Springer (1990).
8) Ito, T.: Efficient Evaluation Strategies for Structured Concurrency Constructs in Parallel Scheme Systems, LNCS, Vol.1068, pp.22–52, Springer (1996).
9) Ito, T.: An Efficient Evaluation Strategy for Concurrency Constructs in Parallel Scheme Systems, *Advanced Lisp Technology in Japan, Information Processing Society of Japan*, Gordon&Breach (2000). (available from http://www.ito.ecei.tohoku.ac.jp/PaiLisp)
10) Ito, T.: A Sound Parallelization Framework for Parallel Scheme Programming, *Parallel and Distributed Computing for Symbolic and Irregular Applications*, pp.3–40, World Scientific (2000).
11) Ito, T. and Matsui, M.: A Parallel Lisp Language PaiLisp and its Kernel Specification, LNCS, Vol.441, pp.58–100, Springer (1990).
12) Ito, T. and Yuasa, T.: Notes on Parallel Evaluation Strategies and their Related Issues, *Parallel and Distributed Computing for Symbolic and Irregular Applications*, pp.82–97, World Scientific (2000).
13) 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).
14) Kelsey, R., Clinger, W., Rees, J., et al.: Revised^5 Report on the Algorithmic Language Scheme, *ACM SIGPLAN Notices*, Vol.33, No.9, pp.26–76 (1998).
15) Mohr, E., Kranz, D.A. and Halstead, R.H.: Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs, *IEEE Trans. Parallel and Distributed Systems*, Vol.2, No.3, pp.264–280 (1991).
16) SSTM ホームページ．http://www.ito.ecei.tohoku.ac.jp/SSTM/
17) Taura, K., Matsuoka, S. and Yonezawa, A.: StackThreads: An Abstract Machine for Scheduling Fine-Grain Threads on Stock CPUs, LNCS, Vol.907, pp.121–136, Springer (1994).
18) Taura, K. and Yonezawa, A.: Fine-grain Multithreading with Minimal Compiler Support — A Cost Effective Approach to Implementing Efficient Multithreading Languages, *Proc. ACM Conference on Programming Language Design and Implementation*, pp.320–333 (1997).

[註] UNIX は X/Open Company Limited の登録商標，Solaris，及び，Enterprise は米国 Sun Microsystems, Inc. の登録商標，PRIMEPOWER は富士通（株）の登録商標，TX7/AzusA は日本電気（株）の登録商標，C 言語は ANSI/ISO 9899 に規格されたプログラミング言語である．

(平成 14 年 7 月 22 日受付)
(平成 15 年 2 月 4 日採録)

伊藤 貴康（正会員）

1940 年生．1962 年京都大学工学部電気工学科卒業．現職，東北大学情報科学研究科教授．工学博士．本会理事，東北支部長等を歴任．現在，Information and Computation の Editor，Higher-Order and Symbolic Computation の Associate Editor，IFIP TC1 Chairman 等．専門分野，ソフトウェア基礎科学および人工知能．日本ソフトウェア科学会，電子情報通信学会，人工知能学会，ACM 各会員．情報処理学会フェロー，電子情報通信学会フェロー．

宮川 伸也（正会員）

1974 年生．1999 年東北大学大学院情報科学研究科情報基礎科学専攻博士前期課程修了．同年東北大学大学院情報科学研究科情報基礎科学専攻博士後期課程進学．2002 年東北大学大学院情報科学研究科情報基礎科学専攻博士後期課程単位取得退学．同年から東北大学大学院情報科学研究科助手．並列言語処理系に関する研究に従事．