構を実現するために，ベクターリターンを拡張して，すべての型について，問題のクロージャが既に輸出されり，サスペンド中の他のゴールによって評価中である場合のリターンエントリを加える．

このリターンエントリに戻ってくると，現在の階層の再実行用のゴールを作って，原因になったクロージャのサスペンドリストにゴールをつなぎ，さらに自分自身に ENTER するとサスペンドするようにする．このようにリターンを繰り返して，トップレベルの変数用リターンに至るので，次のゴールの実行に移る．

WaitVar していた輸入テーブルに値が送られてきた時や，ゴールの実行の結果トップレベルに戻ってきた時に，サスペンドゴールが継っている時はそれらを activate する．

4 評　価

Gofer 処理系に手を加え，処理系内部の中間形式から拡張された STG に基づいた AP1000 用の C 言語のコードを生成させる．Gofer の組み込み関数に対応する STG のライブラリが完全に揃っていないため，現在のところ小規模なプログラムしか試していない．

セル番号が 0 のプロセッサが Dialogue 型の main という名前の関数を評価し，途中spec で生成されたゴールが他プロセッサに移動していって，全体が並列に動作する．ホスト側のプログラムは，I/O の依頼をセルから受けて代行するだけの下請けになっている．

性能評価のために用いのは naiive な fibonacci プログラムと，それに次のように並列性を導入したプログラムである．

```
fib_sepc 0 = 1
fib_spec 1 = 1
fib_spec (n+2) = spec ((+) (fib_spec n))
                      (fib_spec (n+1))
fib_hy 0 = 1
fib_hy 1 = 1
fib_hy (n+2) = if (n>10) then
        spec ((+) (fib_hy (n+1)))
             (fib_hy n)
      else
fib_hy (n+1) + fib_hy n
```

粒度を変化させるための，n が小さくなると逐次実行するプログラムも試してみた．fib_hy では，並列実行から逐次実行に移る境界を 10 にしたが，この数字を変化させてみる．

表 1　実行結果 (fib 24)

プログラム	実行時間 (秒)	ゴール sum/max/min	サスペンド
fib	4.97	1/1/0	0
fib_spec	2.93	121394/9406/1	41701
fib_hy(2)	3.34	28658/2711/0	9538
fib_hy(4)	1.84	10947/992/0	3801
fib_hy(6)	1.50	4182/473/0	1502
fib_hy(8)	1.64	1598/212/0	573
fib_hy(10)	1.75	611/95/0	226
fib_hy(12)	1.64	234/39/0	93

これらを AP1000 の 1 台，64 台で実行した時の実行時間，ゴール数，サスペンド数は図 1のとおりである．適度な粒度でも，それほどスピードアップがはかられていないが，プロセッサあたりの最大 / 最小ゴール数からわかるように，負荷分散戦略が単純過ぎるため，特定のプロセッサにゴールの実行が集中してしまうためだと考えられる．

5 おわりに

関数型言語の逐次実行モデル STG を拡張することによって，細粒度並列プログラムを疎結合並列計算機の上で効率的に実行できることを示した．スケジューリングポリシーなど改良すれば，更に効率が上がることも期待される．

参考文献

[1] Bird, R. and Wadler, P.(武市正人 訳): 関数プログラミング，近代科学社, 1991.

[2] Jones, M. P.: An Introduction to Gofer, University of Oxford, 1991.

[3] Jones, SL, P.: Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine, Journal of Functional Programming 1992.

[4] 田中久美子, 岩崎英哉, 武市正人: 抽象度の高いアルゴリズム記述とプログラムの並列実行, 日本ソフトウェア科学会第 9 回大会論文集, pp. 1-4, 1992.

[5] 田中哲朗, 岩崎英哉, 武市正人: Committed Choice による関数プログラムの並列実行, 日本ソフトウェア科学会第 9 回大会論文集, pp. 85-88, 1992.

ファにデータがきているかチェックし，データが来ていたらそれを処理する．

ゴールの評価が終わると，ゴールから指されていたクロージャを update する必要がある． 一般の STG では型に応じた update 用のエントリを呼びだし側が用意するが，ゴールから呼び出される場合は評価した結果の型がどうなっているのかわからない．そのため，グローバル変数 RetInt, RetFloat の値と A スタックの中身を退避したクロージャを作り update する．

後で値が必要となって， このクロージャに ENTER すると，自分自身を update フレームに登録してから，グローバル変数 RetInt, RetFloat の値と A スタックの中身を戻して，トップレベルに戻ってきたのと同じ番号にベクターリターンする．

3.2 輸出入

疎結合型の並列計算機では，ゴールや値の受け渡しを通信によって行なう必要がある． ゴールや値はすべてクロージャ単位で通信するが，一つのクロージャを一度しか評価しないように，輸出テーブル，輸入テーブルを用いて，データの輸出入を行なう．輸出入は，info table のよあるテキスト領域のアドレスはすべてのプロセッサ上で同じだということを利用している．

- whnf
  WHNF に落ちている場合は， クロージャのコピーを作り輸出する． クロージャの中から別のクロージャが指されている時は深さを限定して再帰的にコピーする．
- それ以外
  輸出テーブルにクロージャを登録し輸出テーブルのアドレスを送る． 輸入側は輸入テーブルを作り，プロセッサ番号と輸出テーブルのアドレスを登録する．

ゴールの輸出の際には，WHNF 以外のクロージャも実体を輸出する必要がある． この場合は送り元のプロセッサに輸入テーブルを作ってから，その番号とクロージャの実体を送り，元のノードは輸入テーブルを指すように書き換える． 受けた側では，輸出テーブルを作り，その輸入テーブルへ値を送るというゴールをサスペンドキューにつなぐ．

輸入テーブルは，3 つの状態を持ち，それぞれ info で区別する (図 3)．

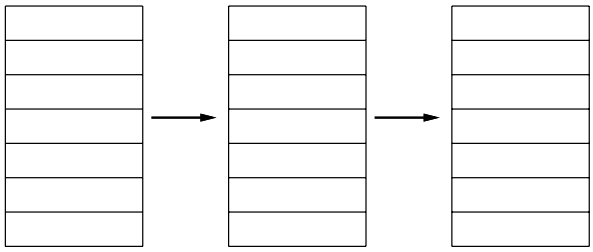

図 3 輸入テーブルの変化

1. ReadVar
   他プロセッサからの輸入されたクロージャのプロセッサ番号と，輸出テーブルのアドレスを内部に持つ． ここに ENTER すると，そのプロセッサに read_var の要求を出し，実行中のゴールをサスペンドリストに入れ，次の WaitVar の状態に移る．
2. WaitVar
   ReadVar に ENTER した場合と，ゴールを輸出した場合で，既に値の要求は出して，結果が返るのを待っている状態． ここに ENTER すると，実行中のゴールはサスペンドして，サスペンドリストにつなげられる．
3. GroundVar
   WaitVar の状態で，計算していたプロセッサから WHNF に落ちたクロージャが送られてきた時にこの状態になる．間接ノードと同値であり，次の GC の際に輸入テーブルから削除される．

クロージャの種類によって，輸出，輸入，ゴールの輸出などのふるいまいが違うが，これは info の中にそれぞれに応じたエントリを作ることによって区別する．

3.3 サスペンドと activate

実行中のゴールが， 輸入テーブルの ReadVar, WaitVar などの即座に評価できないクロージャに入ると，そのクロージャを他のコンテクストが評価するまでサスペンドする． 現在のスタックの状態を保存した新たなゴールを生成して，輸入テーブルのサスペンドリストにつなぐわけだが，細粒度の並列実行を考えるとサスペンドと activate は頻繁に繰り返されることが予想され，そのたびにスタックの内容全部をヒープとやりとりするのはコストがかかる．

そこで，評価しようとするクロージャのレベルごとの細かいゴールに分解し，各レベルごとにスタックの必要な部分だけをクロージャにいれる． この機

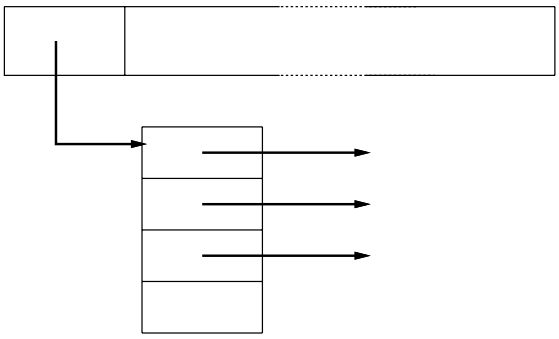

図 1　クロージャの構造

G-machine のように，カリー化された関数適用の構造をそのまま表現せずに, flat な構造で表している (Spineless) ので，関数定義を容易に得ることができる．また, 値と thunk を区別しないので, 特定の型の value を表すためのタグを必要としない (Tagless).

2.2　スタック

STG には A，B の 2 本のスタックがあり，A スタックは引数や実行結果の受渡しに，B スタックはコントロールと update のために用いる.

関数型言語ではループは末尾再帰や，末尾呼び出しの形で表現されるが，これを他の C の関数を呼び出してその結果の値を return する形に変換すると再帰が深くなりスタックを大量に消費してしまう．そこで，C の関数の末尾で他の C の関数に JUMP する形に変換する.

`if (f x) then 1 else 2` の`f x` のような末尾呼び出し以外の関数呼び出しも`f x` の評価後に実行すべきコードのアドレスをあらかじめ用意したスタックに積んでから JUMP すれば C 言語のスタックを用いるよりもスタック使用量が少なくてすむ．STG ではこの考えを更に進めて，ある型に属するコンストラクタの種類に応じた継続のアドレスの配列を B スタックに積む方法を用いている.

`if e1 then e2 else e3` において e1 は Bool 型だが，Bool 型は`False` と`True` の 2 種類のコンストラクタを持つので，B スタックにそれに対応する e3 と e2 への継続のアドレスの配列 (これはコンパイル時に作れる) のポインタを積んでから e1 を評価する．e1 を評価すると，最後に`False` か`True` のコードに入るので，B スタックから配列のポインタをポップし, 対応する継続アドレスに JUMP する.

このベクターリターンにより，if 文や case 文が効率的に実現できる．値の返却は, Int 型 Float 型はグローバル変数 RetInt, RetFloat に入れて行ない，それ以外の場合は A スタックに値を積む.

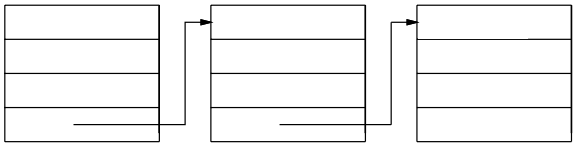

図 2　ゴールの構造

B スタックは update にも用いられる．　クロージャは，複数のクロージャから参照されることがあるが，計算は一度しか実行しないことが期待される．複数から参照される可能性があるかどうかは，ローカルな情報から導き出せるので，複数参照の可能性がある時は，そのクロージャに入った時に B スタックに自分自身を登録してから評価する.

評価した結果が WHNF になった時，B スタックの情報を元に対応するクロージャを update する．update もベクターリターンを利用して効率的な実現を行なっている.

3　並列実行のための拡張

組み込み関数 spec によって導入される並列性は動的なものなので，あらかじめプロセッサ毎に評価すべきクロージャを割り振っておくことはできない．また，並列度がプロセッサの数を超えることがあるので，単一のプロセッサで複数のコンテクストを実行する必要がある.

コンテクストはスタックに対応するが，スタックは大部分が使われないので最大コンテクスト数のスタックを用意するのはメモリの無駄である．そこで，実行中でないコンテクストのスタックはクロージャの形でヒープ中に持ち，評価すべきクロージャを指すゴールのキューを作る.

3.1　ゴール

`spec f x` を評価する時, x の部分を評価するためのゴールを生成して，他プロセッサに輸出する．輸入されたゴールはそのプロセッサのゴールキューに登録される．ゴールは図 2のような構造をしている.

各ゴールは，サスペンドするか評価を終了するまで中断せずに実行される．ただし，ゴールの実行中にも時々 (現在はクロージャのエントリで)，受信バッ

疎結合並列計算機用の関数型言語処理系

Parallel execution of functional programs on loosely coupled multiprocessor systems

田中 哲朗†
Tetsurou TANAKA

†東京大学工学部
Faculty of Engineering, University of Tokyo

概 要

関数型言語は言語中に逐次実行を陽に含まないので並列実行に向いていると考えられる．関数型言語を疎結合並列計算機上で実行する場合，共有メモリを前提にした逐次実行モデルをそのまま適用することはできない．本研究では，関数型言語の逐次実行のモデルである STG(Spineless Tagless G-machine) を元に関数型言語の並列実行のモデルを提案し，関数型言語 Gofer の処理系を疎結合並列計算機 AP1000 の上に実現する．

1 はじめに

関数型言語は並列性を内在するため並列実行に適していると言われる．内在する並列性をすべて引き出すと，不要な計算の発生や並列化のためのオーバヘッドの増大につながるので，並列性をコントロールする必要がある．

先行評価型の関数型言語では，関数適用の際に引数を並列に評価しても不要な計算が生じない．一方，遅延評価型の関数型言語では引数が必要になるまで評価しないことを前提としてプログラミングを行なうので，計算の無駄を抑えながら並列度を上げるためには並列性を陽に記述する必要がある．

並列性を導入するために，

```
spec f x = f x
```

を満たし，`f x` と `x` の計算を並列に実行する組み込み関数`spec` を用いる方法が提案されている．この方法は，言語そのものを変えずに済み，多種類の並列アルゴリズムの記述に十分な柔軟性を持っている ([4] [5])．

この方針に従って関数型言語 Gofer [2]を並列化し，疎結合並列計算機 AP1000 の上で並列に実行する処理系を実現する．効率的な処理系を実現するために，遅延評価型関数型言語の逐次実行モデルである STG(Spineless Tagless G-machine) [3]を拡張した並列実行モデルを用いる．

2 STG

STG は [3]で提案されている遅延評価型関数型言語の逐次実行モデルで，C 言語のような逐次言語に変換して効率的に実行できるように設計されている．

2.1 クロージャの構造

遅延評価を実現するために，

```
g1 x y = h (f x y)
g2 f x y = h z z
  where z = f x y
```

の`f x y` は，値が必要になった時に計算できる構造になっている必要がある．一般に WHNF(弱頭部正規形) になっている場合は値，それ以外を thunk と呼ぶが，STG では両方を区別せずにクロージャとして，図 1のような単一の構造で表す．

あるクロージャを評価する時には，大域変数`Node` にそのクロージャのアドレスを代入してから，info テーブルの 0 番目のエントリーへジャンプする．