記　号　処　理 78－2
（１９９５．３．２４）

Ｃ言語とｆｕｔｕｒｅ関数

柳瀬龍郎
福井大学

Lispなどの言語において並列化を表すためにfuture構文がよく使用される。future構文はプロセスの生成と実行、およびその同期の手段を提供するが、Ｃ言語などの手続き型言語にはこれに相当する手段が見当たらない。

本研究ではＣ言語においてfutureに相当する機能を提供するefuture 関数を提案し、実際に小型のコンパイラとライブラリを制作しefuture 関数を実装した。さらに例題として書いた並行処理プログラムを、マルチプイロセッサシステムにおいて実行した結果について報告する。

C language and future function

Tatsurou Yanase
Faculty of Engineering
Fukui University, Fukui 910, Japan

The future construct is often used to introduce concurrency in a language such as Lisp.　A future creates a process and excute it, brings a methods for synchronizing.　But we have no mechanism same as future in procedural language

In this paper we introdece future mechanism in C language.　And we present the result of experiment on shared-memory multiprocessor system.

1　はじめに

マルチプロセッサシステムにおける並列処理をプログラミングするために様々な構文やメカニズムが提案されている。そのなかでもLisp言語などにおけるfuture構文〔1〕はアルゴリズムの記述においてきわめて自然に用いることができる。

手続き型言語においてもこのfutureのメカニズムを導入することが可能であれば並列処理のアルゴリズムの記述がきわめて容易になり、また作られたプログラムも可読性が高くなる。プロセス間でのメッセージの煩雑な遣り取りを頭の中でシミュレートしながらアルゴリズムを考えるより、futureのメカニズムを利用することにより自然なかたちで、例えばすでに考えられている洗練されたアルゴリズムを記述することができれば並列処理プログラミングも、より一層実用的になる。

本報告ではリストを実現するために一般に導入されるセル型の変数を使ってこの機能を実現し、またプロセスの爆発的生成を抑制するためのメカニズムを組み込んだefuture 関数を提案する。

2　future関数

Multilisp[1]にはfuture構文が用意されており、future構文

(future X)

はXの値に対するfuture（の値）をただちに返し、また並列にXを評価するタスクを生成する。このことにより、値の計算と、並行してその値の使用が可能となる。そしてXの評価で値が得られたら、その値はfutureと置き換えられる。

C言語においてこのような機能の関数を実現するためには、futureの返値に何らかの手段により属性を持たせることが必要になってくる。本報告ではリストを実現するために一般に導入されるセル型の変数を使ってこの機能を実現し、あわせてプロセスの爆発的生成を抑制するためのメカニズムを組み込んだefuture 関数を提案する。

すなわち、efuture 関数はその引数に対するefuture の値をもつセルへのポインタを直ちに返し、同時にその引数に対する評価を開始する。このとき、セルの属性は未評価となっている。評価が終了した時点で値がセルのｃｄｒ部に格納され、値が決定したことを示す属性が与えられる。属性はいまのところcar 部の上位ビットで表現する。　また実際に値の評価が完了していることの確認が必要になったなら、touch 関数で評価の完了と同期を取ればよい。またefuture 関数によってある一定以上のプロセスがすでに生成され、かつ並列に処理されているのであれば、引数をそのまま使用してｅfutureをよんだプロセス自信が関数を評価する処理に入る。次にefuture を使った簡単なプログラムの例をリスト1に示す。このプログラムの実行結果は6が出力される。

mainではfuture関数をよんでいる。このときの第1引数はsub の返値を格納するためのセルへのポインタのアドレスである。第2引数は <sub>、第3引数は＜１＞、第4引数は＜２＞であり、第3引数と第4引数は共に関数sub に渡される引数である。関数 subは二つの引数の和を返値とする関数である。

```
sub(n,m)int n,m; {
      return (n+m);
}
main() {
cell *p;
      future(&p,sub,1,2);
      printf("sum=%d¥n", touch(p)+3);
}
```

リスト１ efutureを使ったプログラム

ｆｕｔｕｒｅによる副作用は、プログラマにその処理がまかされておりｆｕｔｕｒｅでは全く関知しない。

次に、ここで想定しているマルチプロセッサシステムのブロック図を図１に示す。PE0はマスタと呼ばれ、外部とのインターフェースおよび簡単なモニタを備えている。PE1 ～PE17はマスタと区別してスレーブと呼ばれる。またどのPEも割り込み処理と、システムトラップルーチンを備えている。

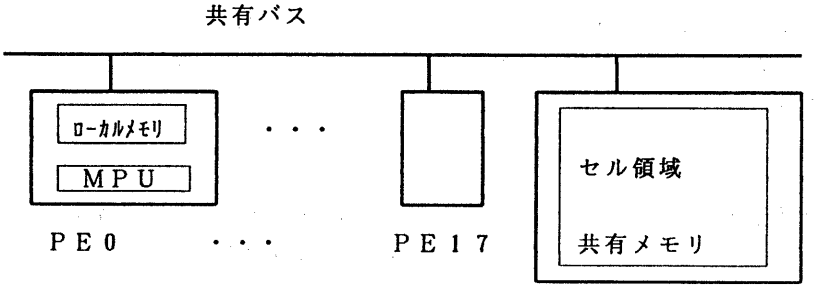


図１　部分共有メモリ型マルチプロセッサシステム

マスタはＷＳ（ワークステーション）と接続されており、Ｃ言語で記述されたプログラムはＷＳでコンパイルされ、マスタから見えるメモリ空間にロードされる。その後、マスタのイニシャライズルーチンにおいて、全てのスレーブにプログラムのロードイメージが転送される。スレーブはefuture によって関数の実行依頼があるまでアイドリング状態（無限ループ）で待つ。

efuture の返値を格納するためのポインタ変数には、

ｓｈａｒｅ　ｃｅｌｌ：共有メモリ内に領域が存在するグローバル変数

ｃｅｌｌ　　：ローカルメモリ内に存在するグローバル変数、および
　　スタック上に存在するオート変数

の２種類がそれぞれ存在する

プログラムが開始されると、マスタはｓｈａｒｅ変数およびグローバル変数のうちのセルポインタをＧＣルートに登録し、その後関数の実行につれて、オート変数のポインタを動的にＧＣルートに登録したり、はずしたりする。

3　セル型変数へのポインタの管理

セル型変数は当然ｇｃのための管理の対象になるが、まずグローバル変数としてポインタが宣言されていれば、main関数が実行される前にマーキングルートに登録される。

ちなみに、main関数はシステムの一般的な初期化を伴う関数から呼ばれる。

また関数内で宣言されるローカル変数であれば、実行制御がその関数内に移った場合に直ちにマーキングルートに登録される。また、その関数を脱出する際にはリターンコードが実行される直前にマーキングルートから外される。

また、関数の引数となっている場合、関数を呼び出した側でローカルかあるいはグローバル変数となっていることが前提となっており、従って返値がセルへのポインタであるような関数の評価値を直接、実引数として使うことを制限している。

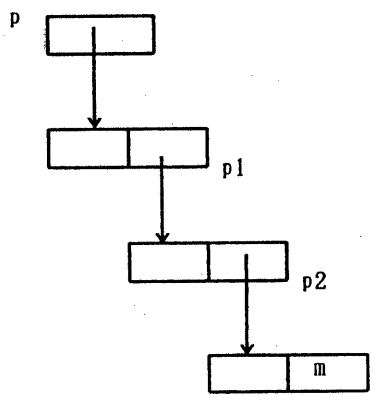
```
sub1(future( sub));
```

リスト２　futureの返値を直接引数として使用

すなわち、仮にリスト２に示すような方法によりsub への引数として積んでしまうと、セルにたいするＧＣが起こった場合にfutureの返値としてのセルがごみになってしまうので、このような方法を取らず、返値を格納するためのセルのポインタのアドレスを用意してefuture を呼ぶ。

```
cell  *p;
      future(&p, sub);
      sub1(p);
```

リスト３　futureの返値をポインタ変数に代入して引数とする




```
main() {
cell *p;
future(&p,sub);
     . . .
}
```

```
sub() {
cell *p1;
    future(&p1,sub2);
    return (p1);
 }
sub2()  {
cell *p2;
    future(&p2,sub3);
    return (p2);
 }
 sub3() {
 int m=3;
     return (m);
 }
```

図１　futureの中でfutureを呼んだ場合

リスト３のように宣言された変数に代入して引数とすることにしている。

以上の理由により、実引数として関数に引き渡されるセル変数へのポインタは常にマーキングルートに登録されていることが保証される。

futureの第一引数はfutureによって評価される関数の返値を格納するためのセルへのポインタのアドレスである。ここに格納されるアドレスはfutureによって返されるアドレスと同じものである。

4 efutureの処理メカニズム

efuture 関数はマルチプロセッサシステムにおいて実行されることを想定している。efuture 関数が実行されたときのメカニズムをおおまかにみると次のようになっている。

①まずefuture 関数を実行していないアイドル状態のプロセッサを探し、あいているプロセッサが見つかれば、引数を送って実行を依頼する。そして関数の返値を格納するためのセルへのポインタを返値としてefuture をreturnする。

②どのプロセッサもあいていなければ、自分で直接、引数の関数を実行する。得られた結果は、新しく用意されたセルのｃｄｒ部に格納され、評価済の属性が与えられてポインタを返値としefuture を抜ける。

どのプロセッサも空いていなければ、自分でefuture の引数の評価を実行するために、爆発的にefuture が発行されることがない。

つぎに詳細にこれを見ることにする。efuture の実現は図２に示すように、

1)アプリケーションレベル
2)ライブラリレベル
3)システムトラップレベル

の３つのレベルで実現している。 すなわち、アプリケーションレベルでefuture が呼ばれるとライブラリレベルからすぐにシステムトラップレベルに制御が移り、そこではまず

（１）アイドリング状態のプロセッサがあるかどうか調べ
（２）なければライブラリレベルにretureする。
（３）アイドリング状態のプロセッサがあれば
　　返値を格納するセルのアドレス、関数の相対アドレス、引数
　　をそのプロセッサにメッセージ送信し関数の開始を依頼する。（図１ａ参照）
　　その後、セルへのポインタをセットし、ライブラリレベルへ戻り、さらにアプリケーションレベルへ復帰する。
（４）関数の開始を依頼されたプロセッサは関数のアドレスと引数を受け取って、実行する
（５）実行が終了すれば、返値格納用のセルに返値を格納し、評価フラグを完了にセットしてアイドル状態に戻る。
（６）どのプロセッサもアイドル状態でなければ、ライブラリレベルにおいて、引数をスタックにつんで関数をコールする。
（７）返値をポインタで指しているセルにセットし評価フラグをクリア（評価済に）し

リターン
となる。

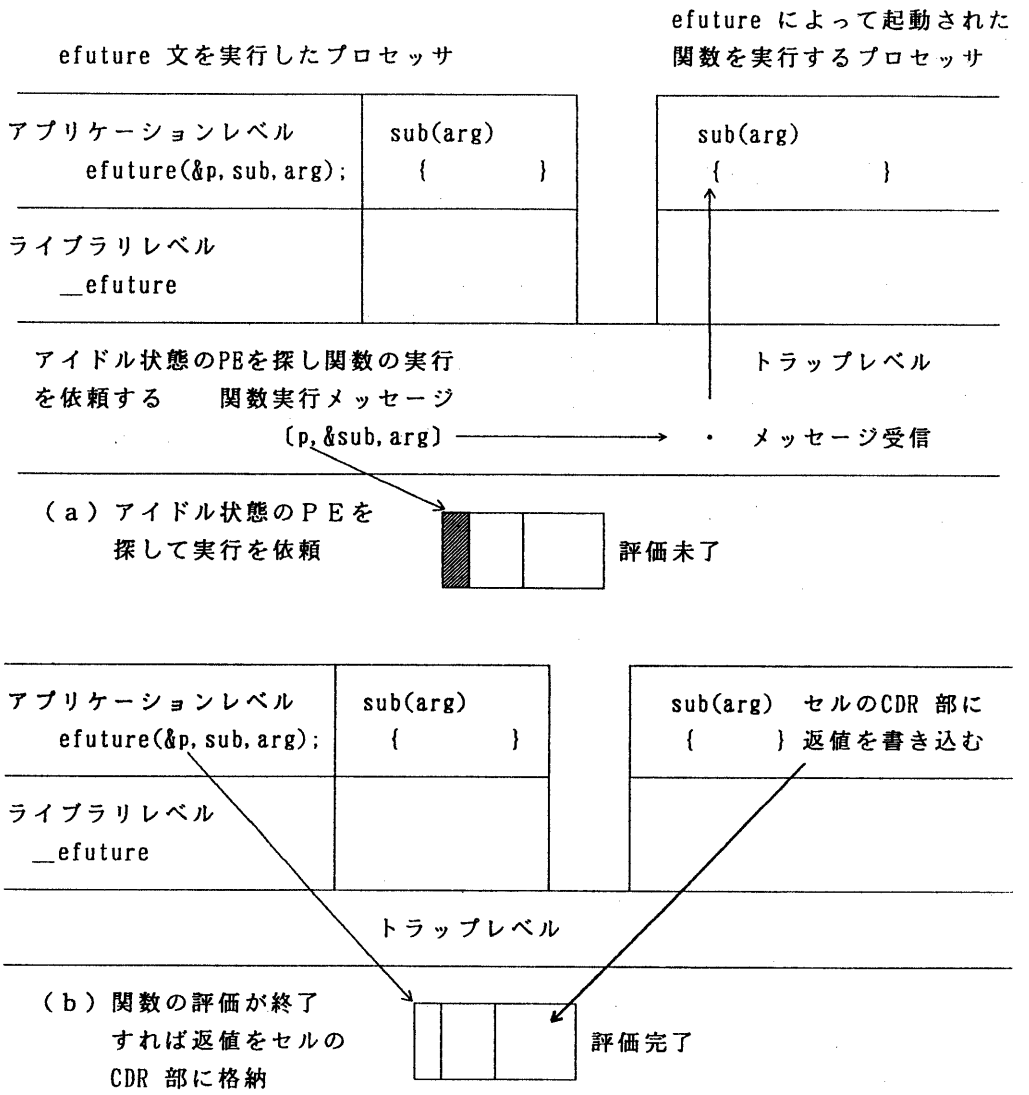


図2 efuture のメカニズム

5 GC

efuture を行おうとしたプロセッサが空のfreeリストを発見した場合、自分で主導権を持ってGCを起動する。GCは現在のところ、一括型マークアンドスイープ方式で実現されている。GCのステップを図3に示す。

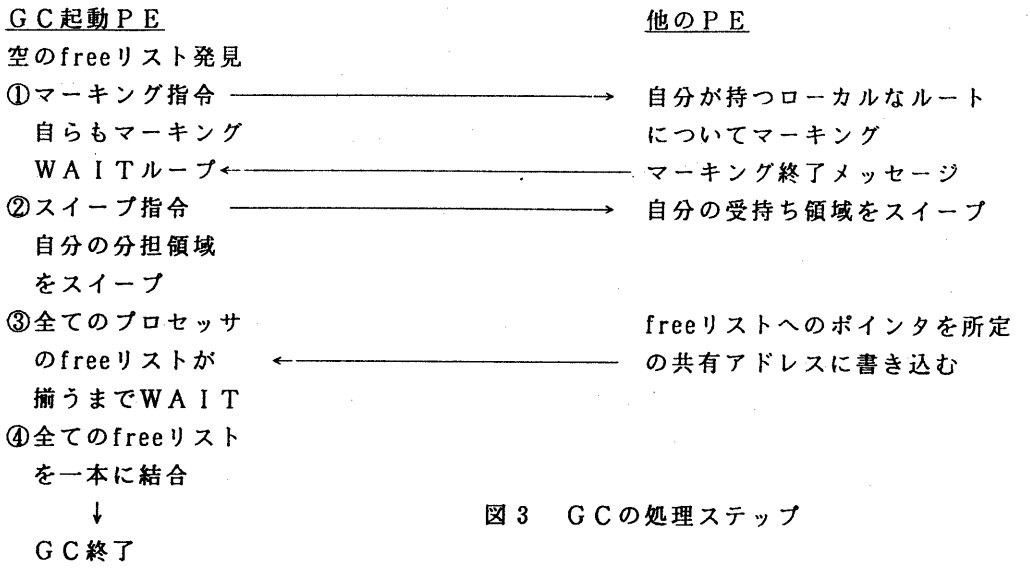


図３　ＧＣの処理ステップ

○efuture のためのライブラリおよびシステムトラップルーチンはすべてアセンブラでコーディング
○ＰＥ相互のメッセージ通信は共有メモリとＣＰＵ相互間の割り込みによって実現
○危険領域への出入りは、共有メモリ領域におかれたフラグに対するTest&Set命令によってセマフォを実現し、これを使用。

6　efuture とメッセージ通信

efuture は負荷の状態によって処理を、動的に他のプロセッサに請け負わせることを試みる。このために、他のプロセッサに環境を送り出す用意をする必要がある。しかもその結果、自分で実行する場合もありうる。この、自分で実行を行う場合、新しく取り揃えた環境は既に自分のところにある環境であるわけだから、無駄になるという欠点がある。しかし、プログラムの構造を大きく変えずに、逐次型のアルゴリズムを並列処理のためのアルゴリズムとして利用することができることが最大の特徴である。

一方、メッセージ通信を使った並列処理のプログラムは、計画的に生成された他のプロセッサ（プロセス）とメッセージをやりとりしながら、同期をとって処理を進める方式である。予め負荷の大きさの予測や負荷の分割が比較的容易な場合に有効な方法であり、それ以外の場合や、負荷の構造が不明なときなどにはアルゴリズムが究めて複雑になりやすい。

図４にefuture を使ってペントミノの箱ずめ問題を、解かせたときの時間、速度向上率、ｃｐｕ利用率を示した。但し、この問題の解き方は、木探索を複数のプロセッサで株分け方式で解いていくようになっており、efuture の使い方としてはあまり適切な例になっていないと思われる。

7　おわりに

並列処理のためのLispのfuture構文の有効性に着目し、C言語に同様な機能を提供する事を目的としefuture 関数を提案した。また、実際にC言語コンパイラおよびライブラリにおいてこれを実装し、並列処理プログラムを書いてマルチプロセッサシステムにおいて実行し、efuture 関数の有効性について検討した。

参考文献

[1] Halstead, Robert H. Jr.:Parallel Symbolic Computing,Computer,Vol. 19, No. 8, pp.35-43(Aug. 1986)

[2] 高橋尚子、中西正和、陽の並列性を導入したLispインタプリタNico Lisp,記号処理、64-5、1992

[3] 小田利彦、C言語におけるリスト処理のための記憶管理方式、66-1、1992

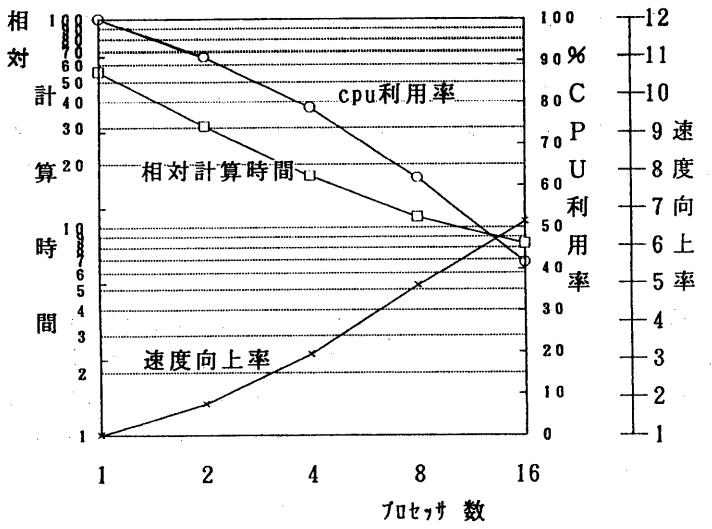


図4　ペントミノ箱ずめについて粗粒度な処理を使った場合