オブジェクトの形状が定義可能な並列記号処理言語用核の設計と実現

岩崎英哉[†*] 竹内幹雄[†**]

本論文では，並列記号処理言語の処理系を実装する際の基礎となる“核”の設計・実現法について述べ，実際にトランスピュータのために開発したシステム TK800 について議論する．TK800 核は，利用者プログラムに対して，非同期スレッド間通信，スレッドの動的生成などの高レベルの機能を提供するが，中でも特に重要なのは，記号処理言語の処理系記述を念頭においた，ごみ集めを含むヒープ管理である．TK800 核は，オブジェクトへのポインタの形，いくつかの種類のオブジェクト本体の形状を既定義のものとして与え，さらに，利用者プログラム独自のオブジェクトを定義・利用する方法も提供する．それは多くの場合は，マクロを機械的に定義するだけですむ．ヒープ領域のごみ集めは，核が責任をもって行うので，利用者は，オブジェクトの割り当て，形状，ごみ集めなどのヒープ管理に煩わされることなく，言語処理系を構成することが可能である．本論文では，TK800 の内部構成，並列関数型言語の解釈実行系の TK800 を利用した記述についても述べ，さらに，設計・実現法についての考察も加える．

Design and Implementation of a Kernel with User-Definable Objects for Parallel Symbolic Processing Languages

HIDEYA IWASAKI[†*] and MIKIO TAKEUCHI[†**]

This paper proposes a small “kernel” named “TK800” on the Transputer network for implementing parallel symbolic processing languages. The TK800 kernel provides rather high level features such as asynchronous inter-thread communication and dynamic thread creation. One of the most significant features is its heap management with garbage collection, which is designed for symbolic processing languages. TK800 kernel predefines the formats of some commonly used object types. In addition, it provides object types which can be defined by the user program. To define an original object type, in most cases, the programmer has only to decide its kind and define some macros for the defined types mechanically. TK800 kernel has the responsibility to collect garbages in the heap area filled by predefined and user-defined objects. The internal configuration of the kernel and an implementation of a parallel functional language are also discussed.

1. はじめに

並列処理への関心が高まる中，Lisp・関数型・論理型など，いわゆる記号処理言語で書かれたプログラムの並列実行に関する研究が活発に行われている．一方ハードウェアとしては，INMOS 社のトランスピュータ[1]が，並列計算機としてだけでなく，さまざまな分野で幅広く利用されはじめており，記号処理的なプログラミングが重要な役割を果たしている．

本論文では，このような背景のもとで，並列記号処理言語の処理系を実現する際の基礎となる“核”として構築した TK800[2] (Transputer Kernel for T800 processor) について，その設計・構成を述べた後，主として記号処理的な側面に焦点をあてて議論する．ハードウェアとしては，はだかのトランスピュータネットワークを想定しているが，基本的な設計は他アーキテクチャにも適用が可能である．

以下に，本論文で用いる用語をいくつか定義する．トランスピュータ上の並列動作の各々を“スレッド”と呼ぶ．スレッドには，TK800 の核を構成する“核スレッド”と，利用者プログラムの実行を行う“利用者スレッド”の二種類がある．“リンク”とは，プロセッサどうしを物理的に接続する線を指し，“チャネル”とはスレッドどうしが通信するための論理的な通信路を意味する．チャネルの物理的実体は，同一プロセッサ

† 東京大学工学部計数工学科
Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo
* 現在，東京大学教育用計算機センター
** 現在，日本アイ・ビー・エム(株)東京基礎研究所

上のスレッド間通信では実メモリに対応し，隣りあうプロセッサ上のスレッド間通信ではリンクに対応する．“オブジェクト”とは，コンスセルなど，記号処理言語で扱うデータ構造でごみ集めの対象になるものを指す．また“TK800 システム”とは，TK800 核，ライブラリなどを含めたシステム全体を指す．

本論文では，第2章で研究の背景・環境について述べる．第3章では，利用者からみたシステムの構成について，第4章では，核の内部からみた構成について論じる．第5章では TK800 核の提供するヒープ管理，オブジェクトの定義，ごみ集めについて議論する．第6章では TK800 を利用した記号処理言語の実現例として並列関数型言語 pfc を示す．考察，他研究との比較などを含めた議論は第7章で行う．

2. 研究の背景

2.1 設計思想

トランスピュータは，相互にネットワーク状に接続し，並列処理を行うことを念頭において設計されたプロセッサであり，

- 2種類の優先度をもつスケジューラ；
- 他のトランスピュータとの通信のための四つのリンクインタフェイス；
- リンクを通じての同期式通信；

をハードウェアでサポートするという特徴をもつ．高優先度のスケジューリングには制御の横取りはない．一方，低優先度のスケジューリングには制御の横取りがあるが，実際は一定時間（2 ms）が経過した後，無条件分岐命令・チャネル入出力命令などコンテクスト切り替えを起こしうる命令のいずれかを実行する時に切り替わる．したがって，低優先度のスレッドが任意の時点で制御を取られるわけではない．

TK800 を実現するターゲットハードウェアとしてトランスピュータを採用した理由には，上で述べたようにハードウェアによる並列処理機能の支援があること，リンク接続によりいろいろな形の疎結合計算機が容易に構築できること，などがあげられる．

ところが，トランスピュータ上で作動するソフトウェアを記述するための言語やライブラリ[3]が提供する機能は，プロセッサそのものがハードウェアで提供する機能と直接に対応していることが多く，一般の利用者プログラムを記述するためにはレベルが低すぎる傾向がある．

たとえばスレッドどうしは，チャネルと呼ばれる構造を用いて通信するが，以下のような制限がある．

- 同期的な通信に限られる．すなわち，受信スレッドがメッセージを実際に受けとって，はじめて送信スレッドは実行を続行することができる．
- 通信は，同一プロセッサ上か，あるいは，直接リンクで接続されているプロセッサ上にあるスレッド間でのみ可能である．

このため，非同期的な通信，直接接続されていないプロセッサ上のスレッド間の通信は，それを支援するためのスレッドを生成して行わざるを得ない．

また，記号処理を行う処理系としての利用者プログラムの側面からみると，ヒープの管理はすべてのシステムに共通して必要な作業であるので，言語処理系ごとに同じようなコーディングを繰り返すことになりやすい，という問題点がある．しかもこの部分は，ごみ集め（Garbage Collection，以下 GC と呼ぶ）や相互排除など微妙なバグを含みやすく，システム開発においてネックとなりやすい．

以上のような背景から，TK800 は以下のような設計方針に基づいて開発を進めていった．

1. 高いレベルの基本機能の提供；
2. 並列記号処理システムのための共通と基盤と実験環境の提供；
3. ハードウェアの特徴を最大限に生かした実現．

1は，上で“レベルが低い”といった部分に関して，より抽象度が高く利用者プログラムにとって使いやすい機能の支援を意味する．2では，ヒープ管理・GC の機能を提供することによって，言語処理系を試作・評価する際に共通して用いることのできる尺度を目指す．さらに，並列言語処理系構成法・アルゴリズムの実験が容易に行うことができるシステムを目標としている．実際の実現にあたっては，3にあげたように，ハードウェアが支援する並列処理機能を有効に利用し，オーバーヘッドの少ない構成を目指している．

2.2 開発環境

本システムは，Sun Microsystems 社の SPARCstation 2 に仏 Archipel 社製トランスピュータボードを挿入したものと，オムロン社の Luna に PC 9801 用トランスピュータボード（いずれのボードもプロセッサは T800，20 MHz）を挿入したものの両者を用いて，開発・実験を行った．これらのボード上のトランスピュータは，ホスト計算機と交信を行うので，“ルートトランスピュータ”（または単に“ルート”）という．ルートの先にもトランスピュータをいくつか

接続し，ネットワークを構成している．

プログラムはすべてC言語で記述し，SPARCstation上で動くINMOS社製クロス開発システム[3]（クロスコンパイラなど）を用いて，トランスピュータ上で実行可能なファイルにする．これを，ホスト計算機上のサーバプログラムを通してトランスピュータにロードし，実行を進める．

3. システムの構成

3.1 全体構成

本システム全体の構成を**図1**に示す．

ルート用・非ルート用双方のTK800核，システムコールの関数などを含むライブラリファイルをコンパイル済みの形でシステムが提供する．記号処理言語処理系などの利用者プログラムはC言語で記述し，クロス開発システムを用いてコンパイルし，TK800のライブラリをリンクする．利用者プログラムは，ルート上・非ルート上で別々のものであったり，ひとつのトランスピュータ上に複数個用意することもある．さらに，利用者はトランスピュータネットワークの接続状態，利用者プログラムのプロセッサへの配置などをLispのS式風に記述したファイルも用意する．これを，TK800システムが用意した“コンフィグレーションファイル変換ツール”で，クロス開発システムが処理できる形に変換する．最後にクロス開発システムは，コンフィグレーションファイルの記述に従いTK800核・リンク済みの利用者プログラムをひとまとめにして，トランスピュータネットワークにロード・実行可能なオブジェクトを作る．これをホスト計算機上のサーバを用いて実際に実行することになる．

3.2 提供する機能

TK800は，利用者プログラムに対し，以下の機能を提供する．

1. 利用者スレッドの動的生成；
2. 任意の利用者スレッド間の非同期的通信；
3. 既定義オブジェクト・利用者定義オブジェクトとヒープ管理；
4. 位置透過な入出力；
5. 統計情報の集計；

1は，並列言語処理系を記述するための基本機能である．2については2.1節で述べたような事情から，核がルーティングを行い，スレッド間通信を支援する．通信相手スレッドは，識別子（トランスピュータネットワーク内で唯一の整数を与える）で指定し，1対1・多対1の通信を可能とする．3はTK800の大きな特長であり，次章以降で詳細に議論する．4は，ホスト計算機を介したファイル，ターミナルなどとの入出力を，任意のトランスピュータ上のスレッドに開放するもので，ルーティングを抽象化した高レベル機能，という点で2と共通している．5は，実現したTK800上の利用者プログラムの性能評価・性能改善を支援するものである．

4. 核の内部構成

図2に核の内部構成，および，利用者スレッドとの関係を示す．核は，スレッド（核スレッド）の集合体として記述し，核の構造のモジュール性を向上させた．核スレッドどうしは，チャネルを通して通信しあう．トランスピュータの場合はハードウェアがスケジューリングを行い，退避すべきコンテクストも小さく高速なコンテクスト切り替えが実現されている．核スレッドは高優先度で，利用者スレッドは低優先度で実行する．これにより，システムコールの受付，GCの処理（次章参照）を自然な形で行うことができた．

図2中で，Message Manager，Thread Manager，Memory Managerは，それぞれ，メッセージ送受信，スレッド管理，GCを含むヒープ・メモリ管理を行うスレッドである．Thread ReceiverとLink Receiver

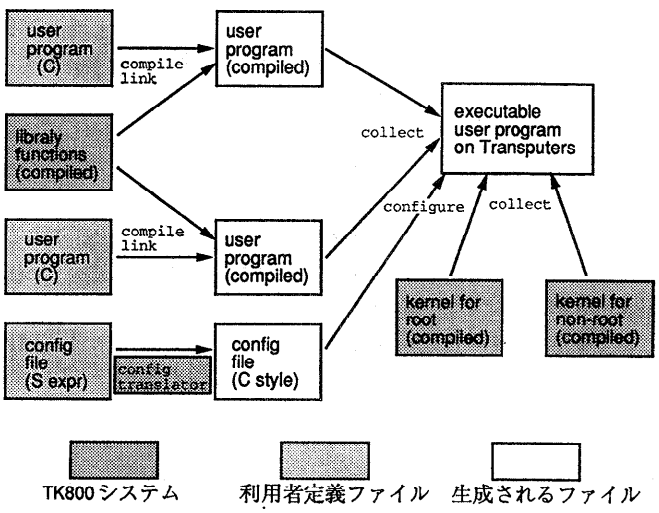

図1 TK800システムの構成
Fig. 1 Overview of the TK800 system.

は，それぞれ，利用者スレッド・隣接プロセッサからの要求を受け付け，要求の種類によってこれを適当なManagerに振り分ける．Thread Sender, Link Senderは，それぞれ，利用者スレッド，隣接プロセッサへパケットを送信するものであるが，この送信には低レベルの同期式通信しか利用できず封鎖する可能性があるので，送り先ごとに別々のものを用意している．

各利用者スレッドは，核（具体的には Thread Receiver）との通信のためのチャネルの対をもつ．利用者スレッドがシステムコールを発行するときはこのチャネル対の片方に要求メッセージを出力し，そのシステムコールが値を返すものならば，もう一方のチャネルから返事が到着するのを待つ．核スレッドは高優先度なので，低優先度の利用者スレッドがシステムコールの要求をチャネルに書き出すと，ハードウェアによって即座にコンテクストが切り替わり，核スレッドが動き出すことになる．これは，一般のプロセッサのオペレーティングシステム (OS) でシステムコールを行う際に，トラップ命令を発行して特権モードに移行していく過程に対応する．

TK 800 核は利用者スレッドのスケジューリングはハードウェアに一切を委ね，利用者スレッド管理によるオーバーヘッドを軽減した．Thread Manager が行う“管理”とは，利用者スレッドの識別子の割り当て，上記のチャネル対の管理などに限られる．

5. ヒープ管理

TK800 は，一定の形状に従ったオブジェクトが矛盾なく存在するヒープ領域を管理する．本章では，TK800 による（プロセッサ内部での局所的な）ヒープ管理について議論する．ここで，オブジェクト本体の大きさはすべて4バイト（1セル）の倍数とする．GC の際には，“ルート”と呼ばれる領域から到達可能なオブジェクトを“生きている”とみなす．

5.1 オブジェクトの割り当て

ヒープからのオブジェクトの割り当て，それに伴う GC などは，すべて利用者スレッドで実行されるライブラリの形で利用者に提供するという方法も考えられた．しかし，ある利用者スレッドが GC を始めた場合には，オブジェクト本体のアドレスが移動する可能性があるので，他の利用者スレッドは GC の終了を待たなければならない．低優先度のスケジューリングに制御の横取りがあることを考えると，これを利用者スレッドレベルで実現するのには，困難が伴う．

これとは反対のアプローチとして，オブジェクトの割り当てをすべて核に委ねる方法もある．しかしこの方法では，ヒープに余裕があれば多くの場合は GC を起動しないですむにもかかわらず，割り当てごとにシステムコールを発行しなければならず，記号処理プログラムの性質を考えると問題がある．

そこで TK 800 では，ヒープ領域の割り当て，および，オブジェクトの割り当てには核は関与せず利用者スレッド自身が行い，GC は核が行うという設計にした．利用者スレッドは，ヒープ領域が不足した場合になってはじめて GC のためのシステムコールを呼び，GC を核に委ねてその終了を待つ．

実際は，システムがオブジェクトの割り当てに関するライブラリを用意しているので，多くの場合利用者はその実現法を気にする必要はない．また，このライブラリを使う限りにおいては，はじめに述べた“一定の形状”が自動的に守られることになる．

5.2 オブジェクトの種類と定義

TK800 は記号処理言語処理系で頻繁に用いられるオブジェクトを“既定義”のものとして提供する．さらに，各々の利用者プログラムで，独自にオブジェクトを定義することもできる．

オブジェクトの型は，ポインタにタグ（ポインタタグ）を付けることによって実現した．この形のポイン

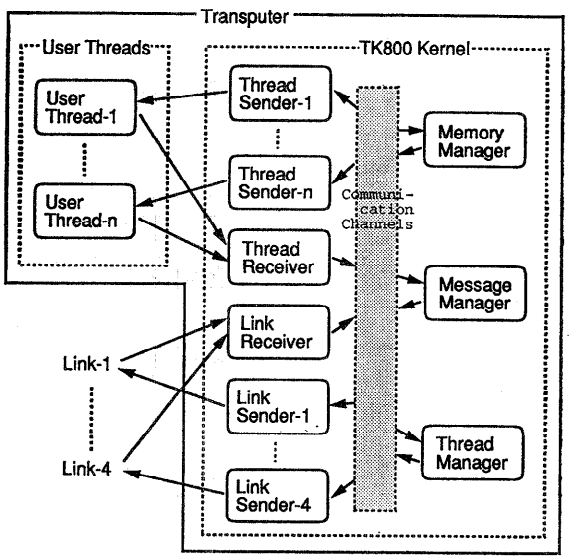

図 2 カーネルの内部構成と利用者スレッド
Fig. 2 Kernel threads and user threads.

タを“タグ付きポインタ”と呼ぶ．タグ付きポインタ（4バイト）の上位5ビットにタグを保持するので，タグの区別によって都合 32 種類のオブジェクトを用いることが可能である．また，後述する“Chunk”という構造を用いれば，さらに多くの種類のオブジェクトを扱うことができる．

図3に既定義のオブジェクトをいくつか示す．この図では，タグ付きポインタとオブジェクト本体の双方をあげた．ただし，整数はタグ付きポインタ中のタグを除いた部分に値を保持し，オブジェクト本体はない．図中，Cons・Vec・Chk・Int は，それぞれの型のポインタタグ値を表す．ここにあげたほかに，シンボル・ストリングなども既定義としている．

Chunk とは，最も一般的な形のオブジェクトであり，内部にはオブジェクトへの（タグ付き）ポインタと，それ以外のデータ（非ポインタ部，たとえば文字列本体，オブジェクト以外のデータへのポインタなど）を含む．非ポインタ部は，もちろん GC の対象にはならない．Chunk の中には，“オブジェクトタグ”と呼ばれる2バイトの領域があり，Chunk の種類を表す数を保持するのに用いる．これにより，65536 種類の異なる Chunk を扱うことができる．これはまた，ポインタタグが不足した場合に，新しい型を定義するのにも用いられる．

TK800 システムは，これら既定義のオブジェクトに対して，型の検査，オブジェクト本体中の各セルへのアクセス，オブジェクト自身の割り当て，その他のユーティリティをライブラリ関数・マクロとして提供している．

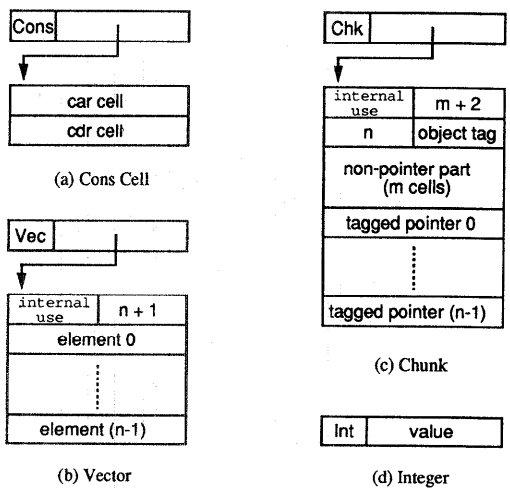

図 3 既定義オブジェクト
Fig. 3 Predefined objects.

現在のところ，9個のタグを既定義としているが，残り 23 個は，利用者プログラムごとの独自の型として定義して利用することができる☆．核は GC を正しく行う責任があるので，利用者が定義したオブジェクトの形状を知っていなければならない．TK800 は，GC が対処できるオブジェクトの形をいくつかに絞ることでこの問題を解決する．具体的には，あるタグ値を利用者定義の型として用いる場合に，タグ付きポインタ（からタグを除いた部分）に保持できるものは，以下の3種類のいずれかとする．

1. タグ付きポインタのみを固定個含むオブジェクト（固定長オブジェクト）へのポインタ；
2. タグ付きポインタのみを可変個含むオブジェクト（可変長オブジェクト）へのポインタ；
3. 単なるデータ（データオブジェクト）；

既定義のオブジェクトとの対比で説明すれば，固定長オブジェクトとはコンスセルのような種類，可変長オブジェクトとはベクタのような種類のものである．またデータオブジェクトには，整数のようにオブジェクト本体はなく，タグつきポインタ内にすべての情報が保持される．

さらに，利用者プログラムが未使用のタグ値に対して新しく定義する型のオブジェクトの形状が，上のいずれであるかを核に知らせるシステムコールも用意し，GC が正しく行われるようにした．

これら利用者定義オブジェクトについても，既定義のオブジェクトと同様に，型検査，オブジェクト本体中の各セルへのアクセス，オブジェクトの割り当てなどを行う汎用関数・マクロをライブラリとして提供する．その具体例は，第6章で示す．

上記以外の形状のオブジェクトを独自に定義する場合は，Chunk を用いる．TK800 は，このようにオブジェクトの形状に一定の条件を設けたが，実用上はこれで充分と考えている．また，どのオブジェクトも比較的単純な形なので，GC のアルゴリズムが複雑にならないという利点もある．

5.3 ごみ集め

現在のところ，一般的に用いられているトランスピュータボードは，局所的なメモリが 1M バイトから 4M バイトの範囲のものが多く，ここにプログラム

☆ 利用者プログラムは既定義のタグに対しても再定義することができる．

コード・データ・ヒープなどすべてを納めなければならない．したがって，ヒープを半分しか利用しない複写式のGCを行うのは不適当である．そこでTK800では，Morris[4]による印掃 (mark & sweep) と圧縮 (compaction) を基礎としたGCを採用している．アルゴリズムの関係上，ヒープ領域はアドレスの大きいほうから小さいほうへ向かって順番に用いる，という制約を設けるが，これが問題になることはほとんどないと考えている．

GCでは，オブジェクトの形状（固定長，可変長など）に応じた処理を行うので，それが既定義か利用者定義かは，GCの性能には影響しない．

5.1節で述べたように，制御を横取りされない高優先度の核スレッドでGCは行われる一方，利用者スレッドは低優先度である．低優先度スレッドは任意の場所で制御を奪われるわけではない（2.1節参照）が，コンテクスト切替えを起こしうる命令☆の実行タイミングには注意を要する．

“生きている”セルへの印付けを始めるためのルートはタグ付きポインタの配列であるが，実際はその先頭アドレスと末尾のアドレスをもつ構造体として表現する．TK800では，複数のルートからの印付けを許すことにした．GCを行うシステムコール (t_gc) へ与える引数は，ルートを表現する構造体へのポインタの配列と，使用中のヒープ領域へのポインタであり，生きているセルの圧縮終了後のヒープの先頭へのポインタを返す．

図4に，その典型的な使用法として，大きさnセルのオブジェクトのための領域をヒープから確保するライブラリプログラムの一部を示す．ここで，ヒープ領域は heaptop から heapbot の間であり，ヒープの使用中領域の先頭を heapp が指すものとする．また，chanpair は核との通信のためのチャネル対，roots は上述の（ルートを表現する）構造体へのポインタの配列を表す大域変数である．このプログラムは，heappの値をnだけ減らす部分にコンテクスト切替えを起こしうる命令を含まず，安全に作動する．

```
if ( heapp - n < heaptop )
        heapp = t_gc(&chanpair, roots, heapp, heapbot);
heapp -= n;
if ( heapp < heaptop )
        error("not enough heap collected"\n);
```

図4 ヒープから領域を確保するプログラムの一部
Fig. 4 Fragment of heap allocation program.

6. 記号処理言語の実現例

現在のところ，TK800を利用した記号処理言語処理系としては，関数型言語 pfc[5] の解釈実行系，および，Committed Choice 型言語 Fleng[6] の解釈実行系[2]がある．本章では疑似並列実行を行う pfc の実現を簡単に述べることによって，TK800利用による処理系構成の具体例を提示し議論を行う．

pfc はS式の構文を持つ．遅延評価を基本としているので，式の値は必要になるまで評価せず，また，一度評価を行えばその結果を再利用する．並列評価のための基本関数は2引数関数 spec[7] であり，(spec f x) は以下のような意味をもつ．

- 表示的意味：(f x) と同じ．
- 操作的意味：x と (f x) を並列に評価．

ここで示す実現例では，関数 spec によって指定された pfc における並列動作ひとつを，TK800の利用者スレッドひとつに割り当てて実行する．したがって，以下では pfc 内の並列動作に対しても“スレッド”という言葉を使う．

表1に，pfcで使うオブジェクトを示す．9種類のオブジェクトを pfc で独自に定義しているが，その例を**図5**に示す．固定長オブジェクトの例として三つ組を表す triple（タグ値は6），可変長オブジェクトの例として関数閉包を表す closure（タグ値は7），および，データオブジェクトの例として文字を表す char（タグ値は8）をあげた．#define で使っている

表1 pfc で使うオブジェクト
Table 1 Objects used in pfc interpreter.

型	定義方式	種類	
Integer	既定義		整数
Cons	既定義		コンスセル
String	既定義		識別子名
Vector	既定義		ベクタ
Char	利用者定義	データ	文字
Bool	利用者定義	データ	論理値
Quote	利用者定義	固定長	定数
Sym	利用者定義	固定長	識別子
Pair	利用者定義	固定長	二つ組
Triple	利用者定義	固定長	三つ組
Thunk	利用者定義	固定長	遅延評価の実現に使用
Closure	利用者定義	可変長	関数閉包
Internal	利用者定義	データ	内部的に使用

☆ チャネル入出力命令，無条件分岐命令が代表的なものであり，前者はTK800核へのシステムコールで，後者はCの繰り返し構文や else 節のある if 文などで用いられる．

ものは，TK800 システムが提供するライブラリで定義されている．たとえば，fixed_element(x, i) は，x で指されているオブジェクトを固定長とみなし，そのi番目の要素をとる汎用マクロである．また make_variant_2 は，要素数2の可変長オブジェクトを割り当てる汎用ライブラリ関数である．前章で説明したように，割り当て自体は利用者スレッド自身が行うが，GC を引き起こす可能性があるので，利用者スレッドから核へのチャネル対へのポインタも引数に含む．図5では，このチャネル対が ch で示されるものとした．

利用者定義オブジェクト内の各セルへのアクセスのマクロ，オブジェクト本体の割り当てのマクロなどは，ここで示したように，固定長・可変長・データオブジェクトそれぞれについての汎用ライブラリを使って定義しておくだけでよい．多くの場合は，このようにオブジェクトの実際の内部構造を気にする必要なく定義する．

各スレッドは，タグ付きポインタを積むためのスタック（“値スタック”と呼ぶ）を，制御スタック（Cの実行時スタック）とは別に用いて，pfc の式を評価する．このスタックに積まれているのは，GC のマーク付けのルートとしなければならない．したがって，各スレッドの値スタックの現在使用中の領域を，ルートひとつをあらわす構造体（5.3 節参照）で表現し，それへのポインタの配列を GC のシステムコールの引数として渡している．この方法は TK800 上で動作する言語処理系を実現する際の常套手段であり，TK800 が印付けのルートを複数許すことにした理由でもある．

同じ式を二回評価するのを避けるために，他スレッドによってすでに評価が開始された式の値を必要になったスレッドは，その値が求まるまで実行を中断しなければならない．また，値を求めたスレッドは，その値を待って中断しているスレッドを再開させなければならないが，これは TK800 による非同期式通信を用いている．非同期式通信は，この他にも並列評価を開始させるためのメッセージ，最終的な評価結果が得られた時点になってもなお実行を継続中のスレッドを回収するためのメッセージにも利用している．

pfc のような TK800 システム上での利用者プログラムは，“利用者スレッドは任意の場所では制御を横取りされない”というトランスピュータのハードウェアの性質を利用する．使用するCの構文によって，コンテクスト切替えを起こしうる命令が含まれるかどうかがわかる（5.3 節参照）ので，利用者プログラムは，きわどい部分や“過渡的な”状態がその命令をまたがないように気をつければ安全に作動する．その結果，陽な相互排除を避けることが可能となる．

```
/* triple */
#define T_TRIPLE          6
#define triplep(x)        tagp(x,T_TRIPLE)        /* type check */
#define triple_fst(x)     fixed_element(x,0)      /* first field of triple */
#define triple_snd(x)     fixed_element(x,1)      /* second field of triple */
#define triple_thd(x)     fixed_element(x,2)      /* third field of triple */
#define altriple(f,s,t)   make_fixed_3(ch,T_TRIPLE,f,s,t) /* allocation */

/* closure */
#define T_CLOSURE         7
#define closurep(x)       tagp(x,T_CLOSURE)       /* type check */
#define alclos0(f,n)      make_variant_2(ch,T_CLOSURE,f,n) /* allocation */
#define alclos1(f,n,a0)   make_variant_3(ch,T_CLOSURE,f,n,a0)
#define closure_fn(x)     variant_element(x,0)    /* function */
#define closure_arity(x)  variant_element(x,1)    /* arity */
#define closure_arg(x,i)  variant_element(x,(i)+2) /* i-th argument */

/* char */
#define T_CHAR            8
#define charp(x)          tagp(x,T_CHAR)          /* type check */
#define alchar(c)         put_tag(T_CHAR,c)       /* allocation */
#define char_code(x)      clr_tag(x)              /* character code */
```

図5 オブジェクト定義の例
Fig. 5 Examples of definitions for user defined objects.

7. 議論と考察

7.1 核としての側面

TK800 核は，これまで述べてきたような機能を利用者プログラムに提供する“ライブラリ”としての側面もある．しかし，核は利用者プログラムとは独立な実行主体*であり，トランスピュータ上にロードされれば高優先度で実行され，あたかも OS 核のような役割を果たす．このような視点にたてば，TK800 は，ファイルシステムなどはもたない“小さな OS 核”として捉えるのが適当であろう．

実際，5.3 節で述べたように，トランスピュータのメモリは多くないので，TK 800 核はなるべく小さくする必要がある．それにはコードを小さくすることはもちろんであるが，各核スレッドが使用する制御スタックの量を抑制するのも重要である．**表 2** に，核スレッドが使用する制御スタックの最大量を示す．(ここではクロス開発システムの提供するライブラリの関数が使用する分は入れていないので，実際の最大使用量はこれよりも少し多くなる．) 特に Memory Manager に関しては，GC での印付けを，再帰的に行わず**，逆転リンクを用いる[8]ことによってスタック消費をおさえている．

TK800 核が必要とするメモリ量を，利用者スレッドが 32 個の場合について**表 3** に示す．ここで，ルート用が非ルート用より多いのは，主としてホスト計算機との入出力処理の分である．動的データとは，核スレッドの制御スタックなど，動的に割り当てるデータを指す．また，ここではルート・非ルートの実メモリ量を，それぞれ 4M バイトとした．これを見ると，実メモリ全体における核の占有量はいずれも数%程度であり，負担は軽いと判断できる．

7.2 ヒープ管理とごみ集め

TK 800 を用いて，記号処理言語処理系などの利用者プログラムを記述するにあたって，独自に定義するオブジェクトに関しては，その形（固定長オブジェクトか可変長オブジェクトかデータオブジェクトか）を決めるだけでよく，あとは，機械的にマクロを用意すればそれで済む．もちろん，既定義オブジェクトに関しては何も準備をする必要はない．これは，プログラム開発の効率向上への TK800 の寄与を示している．

* 核と利用者プログラムそれぞれに独立した main がある．

** 再帰的な印付けでは制御スタックが 16K バイトでも不足する場合があった．

表 2 核スレッドの制御スタックの最大量
(単位：バイト)
Table 2 Maximum control stack usage of kernel threads (in bytes).

核スレッド名	割り当て量	最大使用量
Message Manager	512	164
Memory Manager	384	224
Thread Manager	384	124
Thread Receiver	128	48
Thread Sender	128	40
Link Receiver	128	48
Link Sender	128	24

表 3 核全体の必要とするメモリ量 (単位：キロバイト)
Table 3 Memory usage of TK 800 kernel (in K bytes).

	コード	静的データ	動的データ	合計	割合
ルート	35	27	8	70	1.7%
非ルート	20	24	8	52	5.1%

実際，前章で述べた pfc 処理系の場合，C言語のソースファイル全体で約 6600 行あるうち，オブジェクト・ヒープ管理に関する部分はヘッダファイル中など，わずか 270 行弱（約 4%）であった．また，新しいデータ型を pfc 処理系に後から導入するのも，GC の心配をすることなく，容易に行うことができた．

TK800 核による GC の処理時間は，ヒープ領域の大きさ，生きているセルの量に依存するが，pfc と Fleng の結果を総合すると，ヒープが 256K バイトの時，多くの場合で 0.4〜0.8 秒程度であった．しかし，場合によっては 1 秒を超えることもあり，この高速化は今後の課題でもある．

最近は，C言語の環境にも GC を導入するための，保守的 (conservative) な GC[9] に関する研究も行われている．保守的 GC では，タグ付きポインタという“疑い”のあるものから指されているオブジェクトは生きている可能性があるので，ごみとして回収せずそのまま保存しておく．しかし，TK800 核に保守的 GC を導入するのは，トランスピュータの実メモリ量との関連で，以下のような問題点がある．第一に，生きている可能性があるオブジェクトは，そのまま（場所を変えずに）保存しなければならないので，ヒープ領域の断片化を引き起こす．第二に，タグ付きポインタと“似た”ビットパターンのものが，本当にオブジェクト本体（の先頭）を指しているかどうかを判定しなければならない．そのためには，同じ大きさのオブジェクトだけを含むようにヒープを分割する，オブジェク

ト本体の先頭に印をつけて判別できるようにする，など，いろいろな方法が考えられるが，いずれもメモリの有効利用の見地から考えると不利である．

以上のような設計上の立場から，TK800では印掃・圧縮式のGCを採用した．そのため，利用者スレッドの制御が切り替わる可能性のある時点では，GCにおいて保護されるべきタグ付きポインタを印付けのルートにおく必要がある．pfcではこれを値スタック上におくことにしたが，Cの局所変数へのアクセスと比較するとオーバーヘッドがある．

GCの設計方針は，WeiserらによるPCR (Portable Common Runtime)[10]と大きく異なる．PCRは，汎用OS上の言語独立な基礎となるシステムで，Cなどの言語も対象とし保守的なGCを行う．また，PCRはOSと言語の実行時システムの中間に位置づけられているが，TK800は，はだかの計算機（トランスピュータ）上の核である点も立場を異にする．

7.3 スレッド間通信

TK800が提供する非同期的通信は，同一プロセッサ上のスレッド間であっても間に核が入るので，チャネルを直接使用する同期的通信と比較すると，当然のことながらコストが高い．したがってTK800は，通信コストがネックになるようなプログラムが同一プロセッサ上でチャネルを直接使うことは妨げず，状況に応じて使い分けてもらうという立場をとる．

ただしチャネルの場合は，占有的に利用するか，相互排除して利用するかしなければならない．TK800の提供する通信機能は，利用者スレッドへの通信路を多重化していることに相当するので，上のような心配の必要なく処理系の開発を行うことが可能である．

Barataら[11]も，通信を仮想化したシステムをトランスピュータ上で実現している．これはロボティクスでの応用を考えており，ルーティングを行う通信という点ではTK800と共通するが，リンク接続の状態をスイッチングハードウェアで動的に変更する機能も持つ．また，ヒープ管理機能は持たない点もTK800とは異なる．

8. おわりに

本論文では，トランスピュータ上の並列記号処理言語システム実現を支援するTK800について，その基本設計・構成とヒープ管理の面に焦点をあてて論じた．TK800の分散処理的な側面にはふれなかったが，これについては別の機会に報告したいと考えている．

pfcという言語自体，並列関数プログラミングの実験道具という位置付けで開発したものである．pfcの利用者から，並列化・非決定性のための基本関数の導入の要請を受けても，それを処理系に採り入れるのは容易であった．この場合も，TK800による高レベルな機能の寄与は大きかったと考えている．

GCは現在は一括方式で行っているが，応用プログラムによっては，逐次型 (incremental) のものも必要になることがあろう．世代別GC[12]も含めて検討する必要がある．また，プロセッサ間にわたるヒープ管理は現在実験中であるが，これを含めてTK800システムを改良・充実させてゆきたいと考えている．

参考文献

1) INMOS Limited : Transputer Instruction Set, Prentice-Hall (1988).
2) 竹内幹雄，岩崎英哉：Transputer ネットワーク上の記号処理用分散カーネル，第9回日本ソフトウェア科学会大会論文集，pp. 81-84 (1992).
3) ANSI C Toolset User Manual, INMOS Limited (1990).
4) Morris, F.L.: A Time- and Space-Efficient Garbage Compaction Algorithm, *Comm. ACM*, Vol. 21, No. 8, pp. 662-665 (1978).
5) Takeichi, M.: Evaluation Partial Order and Synchronization Mechanisms in Parallel Functional Programs, *The 40th IFIP WG 2.1 Meeting Record 629*, pp. 1-12 (1989).
6) Nilsson, M. and Tanaka, H.: FLENG Prolog —The Language Which Turns Supercomputers into Parallel Prolog Machiner, *Proc. 5th Logic Programming Conference* (*LNCS 264*), Springer-Verlag, pp. 170-179 (1986).
7) Burton, F.W.: Encapsulating Non-determinacy in an Abstract Data Type with Determinate Semantics, *Journal of Functional Programming*, Vol. 1, No. 1, pp. 3-20 (1991).
8) 高橋俊成：複合タグ方式のLisp処理系におけるガベージ・コレクタの実現とその問題点，情報処理学会論文誌，Vol. 30, No. 3, pp. 339-346 (1989).
9) Boehm, H-J. and Weiser, M.: Garbage Collection in an Uncooperative Environment, *Softw. Pract. Exper.*, Vol. 18, No. 9, pp. 807-820 (1988).
10) Weiser, M., Demers, A. and Hauser, C.: The Portable Common Runtime Approach to Interoperability, *Proc. 12th ACM Symposium on Operating System Principles*, pp. 114-122 (1989).

11) Barata, M. M., Cunha, J.C. and Steiger-Garção, A.: Transputer Environment to Support Heterogeneous Systems in Robotics, *Proc. 2nd International Conference on Applications of Transputers*, pp. 327-334 (1990).
12) Libermann, H. and Hewitt, C.: A Realtime Garbage Collector Based on the Lifetime of Objects, *Comm. ACM*, Vol. 26, No. 6, pp. 419-429 (1983).

(平成4年12月21日受付)
(平成5年6月17日採録)

岩崎 英哉（正会員）

1960年生．1983年東京大学工学部計数工学科卒業．1988年同大学院工学系研究科情報工学専攻博士課程修了．同年東京大学工学部計数工学科助手．1993年4月より東京大学教育用計算機センター助教授．工学博士．記号処理言語，関数型言語，並列処理システムなどの研究に従事．日本ソフトウェア科学会，ACM 各会員．

竹内 幹雄（正会員）

1966年生．1990年東京大学工学部計数工学科卒業．1993年同大学大学院修士課程修了．同年日本アイ・ビー・エム株式会社入社．現在東京基礎研究所勤務．並列処理，プログラミング言語，オペレーティングシステムに興味を持つ．日本ソフトウェア科学会，ACM 各会員．