JAERI-M
92-084

# 複数個のトランスピュータによる並列計算の検討
## ― 計算時間とオーバーヘッドの関係 ―

1992年6月

猪俣　新次・鈴木　勝男

日本原子力研究所
Japan Atomic Energy Research Institute

JAERI-Mレポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319-11茨城県那珂郡東海村）あて，お申しこしください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

JAERI-M reports are issued irregularly.
Inquiries about availability of the reports should be addressed to Information Division Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.



編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

# 複数個のトランスピュータによる並列計算の検討
## －計算時間とオーバーヘッドの関係－

日本原子力研究所東海研究所技術部
猪俣　新次・鈴木　勝男[+]

（1992年5月15日受理）

本報告は複数個のトランスピュータを用いて並列計算したときの計算時間の短縮とオーバーヘッドの増加について，2つの計算事例を対象として検討した結果を述べている。ハードウェアとしては，9個のトランスピュータを搭載したマザーボード(IMS B008)[1]を装着したパーソナル・コンピュータIBM PC/AT互換機を使用し，ソフトウェア開発環境としてはMS－DOS版occam 2ツールセットを使用した。トランスピュータ・ネットワークの形状はリング状に構成した。並列計算事例は，①四則演算及び6個の関数(sqrt, sin, cos, tan, exp, log)，②定積分計算の2つを取り上げた。両事例とも（計算時間）／（オーバーヘッド）の比が大きい場合にトランスピュータ数に比例した相対速度が得られるが，リング状ネットワークでのオーバーヘッドは並列化に伴う準備と後処理に要する時間と，トランスピュータ相互間の通信に要する時間（これはトランスピュータ数に比例して増加する）との和から成ることが明かとなった。

東海研究所：〒319-11　茨城県那珂郡東海村白方字白根2－4
[+]原子炉工学部

# Study on the Concurrent Calculation with a Multitransputer Network
## - Relation between Processing Time and Overhead -

Shinji INOMATA and Katsuo SUZUKI+

Department of Engineering Services
Tokai Research Establishment
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

(Received May 15, 1992)

A relation between the calculation processing time and the overhead of multitransputer network is studied in performing several concurrent calculations. Hardware configuration we used was a ring-type linked nine transputer network on an INMOS B008 mother board which was inserted into a slot of a personal computer compatible with IBM PC/AT. Further we used the occam2 toolset of MS-DOS version to write the calculation programs in the occam2 language.

On arithmetic operations, some trigonometric and transcendal functions and numerical integrations, the calculation time and overhead were measured. The results show that the relative calculation speed is proportional to the number of transputers when the ratio of calculation time to overhead is much greater than 1.0. And the overhead is consumed in inter-communicating among transputers for the pararell calculation set-up.

Keywords: Transputer, Occam2 Language, Multitransputer Network, Concurrent Calculation

---

+ Department of Reactor Engineering

# 目　　次

Contents

# 1. はじめに

マイクロ・プロセッサはコンピュータ・システムばかりでなく、さまざまな自動化システムに利用されている。マイクロ・プロセッサを使用したシステムの処理時間を短縮するには、高速なマイクロ・プロセッッサを使用すればよいが、1つのシステム内に複数個のマイクロ・プロセッサを使用して並列処理を行わせれば、さらに処理時間の短縮を計ることができる。

リアル・タイムに並列進行する現実の世界をモデル化して、それを複数個のマイクロ・プロセッサを用いたシステム上に実現しようとすると、つまりシーケンシャル・マシンを用いて並列処理システムを構築しようとすると、ハードウェアの複雑化やバスのボトルネックといった問題に直面する。したがって、このような問題に対応できるマイクロ・プロセッサとしては並列処理システムの構築に適したアーキテクチュアを有するものが要求される。トランスピュータは、はじめからこのようなシステムの構成要素として使用できるマイクロ・プロセッサとして開発された。トランスピュータは相互間のデータ転送にバスを使用せずに4本のシリアル・リンクを使用する。また、セントラル・プロセッシング・ユニット（CPU）、演算ユニット（FPU）、シリアル通信ユニットはそれぞれ独立並行に処理が実行されるように設計されていて、並列分散処理が必要とされる計測制御分野において、さまざまな応用が試みられている。

トランスピュータはリアルタイム並列処理システムに有効な機能を有しているとはいえ、プロセス（タスク）を複数個のトランスピュータに配分して処理を実行することに伴う準備や後処理に要する時間、さらにはトランスピュータ間の通信に要する時間（オーバヘッド）が、あらたに必要となり、これらによる処理時間の増加はさけられない（ここでは複数個のトランスピュータによる処理の並列化に伴って必要とされる余分な時間をオーバヘッドと称する）。そこで、今後の実用化システムへの応用に先だって、1）四則演算及び6個の関数（sqrt, sin, cos, tan, exp, log：これらの関数を$N^{88}$BASICにならって、以降、算術関数と称する）、2）定積分計算の2つの計算事例を9個のトランスピュータで構成したネットワーク上で実行し、処理時間の短縮とオーバーヘッドとの関係を検討した。並列化の手法には、事象の並列化、計算領域の並列化、アルゴリズムの並列化が考えられるが、ハードウェアの機能、および（トランスピュータ数／処理速度）比の評価が容易である点から演算領域の並列化を適用できる上記2例を取り上げた。

実験には9個のトランスピュータを搭載したマザーボード（IMS-B008）[1]を装着したパーソナル・コンピュータIBM-PC/AT互換機を使用した。本事例プログラムは並列処理言語occam2で記述し、ＭＳ－ＤＯＳ版occam2ツールセットを用いてプログラム開発を行った。巻末には付録としてoccam2ツールセットの使用例と本事例プログラムのソース・リストを掲載した。

# 2. 使用言語とシステム構成

## 2.1 使用言語の条件

本実験では各種の数値計算を複数個のトランスピュータ上で実行し、その処理時間を正確に測定することにより、トランスピュータの使用個数の増加がもたらす利点（処理時間の短縮）と欠点（オーバーヘッドの増加）に検討を加えた。トランスピュータ上で使用できる言語にはANSI-C, Pascal, FORTRAN, Ada, occam2があるが、本実験に適した言語として、次の観点からoccam2を採用した。

1) トランスピュータには、ハードウェア・タイマが内蔵されており、occam2からこのタイマを容易に参照できる。occam2からは６４マイクロ秒単位と１マイクロ秒単位のタイマを使用できるようになっているが、本実験では１マイクロ秒単位のタイマを使用した。
2) 計測制御の分野でトランスピュータを使用した場合には、ハードウェアの細部の操作や時間的な推移を微細に記述できる言語が適している。特に割り込み処理を容易に記述できる機能は不可欠である。
3) 処理時間とオーバーヘッドの計測した値は、出来るだけソフトウェアに依存した部分が少なく、ハードウェアに起因した値を得たい。そのためにはコード効率の良いコンパイラを使いたい。
4) トランスピュータが有する並列処理機能や通信機能を十分に引き出せるように設計されている言語が望ましい。

## ２．２ トランスピュータ・ネットワークの構成

マイクロ・プロセッサを使用したシステムの処理速度の向上は大きく分けて二つの方向から追求されている。一つは、より高速なマイクロ・プロセッサの採用であり、他はマルチ・プロセッサによる並列処理である。後者は複数のマイクロ・プロセッサを同時並列に走らせてシステム全体のスピードアップを計るという考え方である。この方法は、プロセッサ数が増加するにしたがってバスのキャパシタンスの増加によるバス・サイクルの低下と、バスの取り合い（バス・コンテンション）からスピードアップには本質的な限界があるばかりでなく、ハードウェアが複雑になり、並列処理プログラムの記述も困難になる。したがってマルチ・プロセッサ・システムを構成するのに適したプロセッサとしてはバスを共有せず、相互通信を必要とするプロセッサだけが同期を取り合って通信できるアーキテクチャを有するものが望ましい。

トランスピュータはこのような条件を満たしたプロセッサであり、マルチ・プロセッサ・システムを構成できる素子として下記の特徴がある。

1）　1個のトランスピュータは4対の通信リンクをチップ上に持っている。複数トランスピュータ間で通信を行うための共通メモリや外付け通信回路は不要である。通信リンクは20Mビット／secの全二重通信チャネルであって任意のネットワークを構成できる。

2）　リンク通信はプロセッサと独立に動作する。つまりプロセッサがプログラム実行中に送受信が並行して行われる。

3）　IMS T800はチップ上に64ビットの浮動小数点演算ユニットを内蔵していてプロセッサと演算ユニットは並列実行が可能である。

4）　通常のプロセッサではプロセスの切り替え（コンテキスト・スイッチ）はオペレーティング・システムが行うが、トランスピュータではハードウェア・スケジューラがプロセスの切り替えを行う。

5）　トランスピュータは各自が自分のメモリを有していて、お互いにメモリを共有することはない。

本事例ではパーソナル・コンピューIBM PC用マザーボードIMS B008上に9個のトランスピュータでリング状ネットワークを構成した。この形状に構成したのは各トランスピュータ上で走る計算プログラムが全く同一となるばかりでなく、トランスピュータ数に対する処理速度の評価も容易となるからである。ボードには10個までのトランスピュータ・モジュールを装着できるスロットがあるが、ボード上でホストとなるトランスピュータ（これをルート・トランスピュータと称する）は2個のスロットを占有するので、搭載できる最大数は9個である。ボード上のトランスピュータは直列にハードウェア接続されているばかりでなく、リンク・スイッチIMS C004をプログラムすることによりソフトウェア的に接続することもできる。ソフトウェア的に接続するにはツール（mms2.b4）[4]が必要であり、INMOS社から提供されている。その使用例については付録A4に述べる。

リンク・スイッチの接続はシステムの電源を投入した直後にmms2.b4を走らせて済ませておかなければならない。正しくリング状に構成されていないネットワークへ、リング状ネットワークで走るようにコンフィグレーションされたプログラムをロードしようとしても受け付けられずエラーとなる。mms2.b4は、ソフトワイヤ仕様を記述したファイルとハードワイヤ仕様を記述したファイルを入力として受け取る。mms2.b4はネットワーク上のトランスピュータを巡回して（虫がネットワークの上を這い回るのに似ているのでワーミングと称される）ソフトワイヤ仕様にもとづいた接続を行うとともに、ハードワイヤ仕様と実際のハードウェア接続が一致しているか否かをチェックする。

# 3. 並列計算時間とオーバーヘッドの検討

この章では、1）四則演算および算術関数、2）定積分計算の2つの事例を対象として並列計算をおこなわせ、処理時間の短縮とオーバヘッドの関係を検討する。まずはじめに1）の事例を単一のトランスピュータ上で実行させたときの計算回数と所要時間の関係を求め、これを以降の検討における1つの基準とした。計算時間の計測には、occamから参照できる1マイクロ秒タイマを使用した。

## 3．1　単一のトランスピュータによる計算

単一のトランスピュータによる計算時間計測プログラムのＣＲＴ画面を図3．1に示す。操作手順は四則演算あるいは算術関数の中から1つを選択する。同図では加算を選択している。これに続けて単精度か倍精度かを指定し、被加数、加数、計算回数を入力すると計算開始時刻（time1）、終了時刻（time2）、両者の差である計算時間（time.ope）、和（sum）および1回の加算に要した時間をマイクロ秒単位で表示する。このプログラムの詳細は付録Ａ5．1で述べる。

単一のトランスピュータ上で1回～１０万回の加算を実行させたときの計算時間を図3．2に、開平の計算時間を図3．3に示す。両図を見れば計算時間が計算回数に比例していることは明らかである。加算1回の付近での直線からのずれはタイマの分解能が1マイクロ秒であるためである。表3．1に対象とした四則演算と算術関数について、単一トランスピュータ上で１０万回の計算を実行させたときの1回当たりの計算時間を示す。

## 3．2　複数個のトランスピュータによる計算

### 3．2．1　複数個のトランスピュータへの計算の配分

1章において触れたように並列化の手法には下記の3種類が考えられる[5]。

1）　事象の並列化
2）　計算領域の並列化
3）　アルゴリズムの並列化

事象の並列化とは、複数の事象に、1つづつのトランスピュータを与えて、処理を実行させる方式である。計算領域の並列化とは、計算領域を分割し、分割された領域の計算を各トランスピュータに割り当てる方式である。アルゴリズムの並列化とは、計算のアルゴリズムを細分化して、それらに個々のトランスピュータを割り当てる方式である。

たとえば、よく知られたパイプライン処理は、この方式に属するものである。

ここで扱う二つの事例は、２）に属するものである。四則演算及び算術関数の並列計算では計算回数を分割して、その分割された回数の計算を個々のトランスピュータに配分して、並列化をおこなっている。定積分計算では積分領域を分割し、分割された一つの領域の計算を、一つのトランスピュータに割り当てて並列化をおこなった。ネットワーク上のトランスピュータにプログラムを配分した状態を図３．４に示す。

ネットワークに接続されたトランスピュータのなかでホスト役を担当しているトランスピュータはルート・トランスピュータと呼ばれ、下記の三つの役割を演じている。

１）　パーソナル・コンピュータとネットワークのあいだに存在して、両者の仲介役を務める。

２）　メイン・プログラムを実行する。

３）　他のトランスピュータとともに並列計算を実行する。

１）は、メイン・プログラムがパーソナル・コンピュータのＣＲＴ、キーボード、ディスクなどのシステム資源をホストCPU　80386経由で利用していることを意味している。したがって、いったんメイン・プログラムが走り出すと、システムのホストCPUがルート・トランスピュータに置き換えられたかのような動作をする。

### ３．２．２　トランスピュータ・ネットワークのオーバヘッド

単一のトランスピュータがある計算を繰り返し実行したときの所要時間は、１回の計算時間に計算回数を乗じた値になることは３．１に述べたが、複数個のトランスピュータの場合の所要時間は、このように単純には求められない。各トランスピュータに計算を配分するための前準備、計算開始・終了の通信、配分された計算結果の後処理などが必要となる。たとえば、加算の場合は被加数、加数、演算精度、計算回数を各トランスピュータへ送った後、計算開始指令と終了通知が必要である。

リング状ネットワークでは、ルート・トランスピュータ上のメイン・プログラムが、前準備と後処理を行う時間と、ネットワーク内トランスピュータ上のサブ・プログラムが計算条件、計算開始指令、終了通知を一巡させる時間とが必要である。これら両者の和が並列計算を行わせたために新たに生じるオーバヘッドである。

このオーバヘッドが生ずるメカニズムについては、次節３．２．３で述べるが、ここでは、加算の場合を取り上げてオーバヘッドの実態を見ておく。

トランスピュータ９個で構成されたリング状ネットワークにおいて、１回の加算を行わせる。この一回の加算はルート・トランスピュータ上の計算（サブ）プログラム内で実行され、その所要時間は１６１マイクロ秒である。単一のトランスピュータ上での加算一回の所要時間は３．１で見たように約１．４マイクロ秒であるから、このネットワークのオーバヘッドは約１６０マイクロ秒である。

図３．４にリング状ネットワークを、それぞれ９個～２個のトランスピュータで構成して、１回の加算を行わせたときの所要時間を示す。１個のトランスピュータが増える

とトランスピュータ数（ｘ個）との間に下記のような直線的な関係があることを示している。

$$y = 15x + 26$$

この関係は前準備・後処理に要する一定の時間（２６マイクロ秒）とネットワーク上のトランスピュータが１個増すごとに通信に要する時間が１５マイクロ秒ずつ増加することを意味している。この関係は他の演算や関数についても、アルゴリズムが同じであるならば成り立つと考えられる。

### ３．２．３　四則演算と算術関数の計算

この事例で対象とする演算と関数は３．１のものと同じである。ただし、ひとつの計算条件が追加された。この条件は、９個のトランスピュータから成るネットワーク上で並列計算を実行させるのに、９個のうちの何個のトランスピュータに計算を配分するかを指示するためのものであって、キィーボードから入力される。与えられた計算回数をネットワーク上のトランスピュータに配分して計算を実行させ、トランスピュータ数、計算時間、オーバーヘッドおよび計算回数の関係を実験により解明することを試みた。

まず計算の配分について述べる。ネットワーク上のすべての（９個の）トランスピュータに並列計算を実行させることができるように、計算プログラムを全トランスピュータにロードするようにした。オペレータが計算条件（計算種別、計算回数、計算精度、オペランドあるいは変数値）を与えると、メイン・プログラムはそれをプロトコルに適合したデータ型にまとめた後ネットワークへ送り出す。各トランスピュータはそれを受け取って、条件に即した計算を実行する。計算回数の配分は、たとえば、計算を配分されるトランスピュータ数を３個とすれば、ルート・トランスピュータから数えて連続した３個に計算回数を配分し、他のトランスピュータには０回の計算回数を与えている。１個当たりの計算回数を同一にできない場合、つまり（与えられた計算回数）÷（計算配分トランスピュータ数）に余りが生じた場合には、余り回数はルート・トランスピュータに負担させた。

このプログラムには下記の計算条件が必要である。

・演算種別：　メニュー画面に示したように１から１０までの整数値を各演算や関数に対応づけしてある。たとえば１は加算、２は減算・・・である。

・演算精度：　occamの実数には単精度（３２ビット）と倍精度（６４ビット）がある。１は単精度、２は倍精度である。

・計算回数：　与えられた計算回数である。この回数を指定された数のトランスピュータに配分する。要素が９個の３２ビット整数配列をとり、配列の添字０、１、２、・・・、８をトランスピュータ番号０、１、２、・・・、８に対応させている。

・オペランド（変数値）：　四則演算のオペランドあるいは関数に与える変数の値である。

これらの条件は連続したデータ列の形態でリンクを介して各トランスピュータへ送られる。リンクはoccamの要素としてはチャネルに対応する。チャネルにデータを送るには、そのデータの構造をoccamで宣言する必要がある。プロトコル宣言の基本型については、すでに２．２で述べた。ここでは、上記の条件データ用プロトコルの他に計算開始指令として一つの３２ビット整数と、計算終了通知を送らねばならないので混成プロトコルを使用している。PARAMと名づけた混成プロトコルの宣言を下記に示す。

```
PROTOCOL PARAM
  CASE
    packet; INT;INT;[num.of.transp]INT;[2]REAL64
    start; INT
    finish
:
```

計算条件データにはpacket,計算開始指令データにはstart,計算終了通知にはfinishというタグ（名札）をそれぞれ与えている。混成プロトコルを使用する通信ではデータの頭にタグを付けて送受信する規則になっている。図Ａ５．２にpacketのデータ構造を示す。

これらの条件は、ＣＲＴ画面のプロンプトに応じてオペレータがキィボードから入力する。図３．５に、このプログラムのメニュー画面を示す。図では実数３．０の開平を単精度（３２ビット）で、１０万回の計算を９個のトランスピュータに実行させている。このプログラムについては付録Ａ５．１で詳述する。

つぎにオーバーヘッドが生ずるメカニズムを認識しておく為に、計算時間を計測する部分のアルゴリズムについて述べる。

この計算時間計測プログラムは、ルート・トランスピュータ上で走るメイン・プログラム（ファイル名howfast1.occ)と全トランスピュータ上で走って計算を実行するサブ・プログラム（operatio.occ）に分かれている。両プログラムから時間計測と計算実行にかかわる主要部分を図３．６と図３．７ａ，ｂに示す。また、これらのフローチャートを図３．８に示す。

メイン・プログラム（ｍ）とサブ・プログラム（ｓ）を、フローチャートと対照させてトレースする。

ｍ１：　計算条件packetをチャネルoutへ送り出す。
ｓ１：　チャネルinputからの計算条件packetの到着を待つ。
ｓ２：　到着したら、それを用意してある変数に読みとり次のトランスピュータへチャネルoutputを介して送り出す。
ｍ２：　チャネルinへ計算条件packetがネットワークを一巡して戻って来るまで待つ。戻ってきたら、それを読みとって一巡したことを確認する。
ｍ３：　計算開始時刻を変数timeへ読みとる。

ｍ３：　計算開始時刻を変数timeへ読みとる。

ｍ４：　計算開始指令start（整数　０）をチャネルoutへ送り出す。

ｓ３：　チャネルinputからの計算開始指令startの到着を待つ。

ｓ４：　計算開始指令startは整数データを伴って来るので、そのデータを読みとった後、１を加えてチャネルoutput経送り出す。この整数値は計算条件packet内にある、各トランスピュータに配分された計算回数を納めた配列の添字として使用する。

ｓ５：　そのトランスピュータに配分された回数だけ計算（この場合は加算）を実行する。

ｍ５：　計算開始指令startが一巡するまで待つ。チャネルinへ戻って来たら、それを読みとって一巡したことを確認する。

ｍ６：　計算終了通知finish（タグだけ）をチャネルoutへ送り出す。

ｓ６：　計算終了後、前のトランスピュータからの計算終了通知finishがチャネルinputへ到着するのを待つ。

ｓ７：　チャネルinputからの計算終了通知finishを受け取ったら、それをチャネルoutputを介して次のトランスピュータへ送る。

ｍ７：　計算終了通知finishがチャネルinへ戻るのを待つ。

ｍ８：　チャネルinへ計算終了通知finishが戻って来たら、計算終了時刻をtime2へ読みとる。

以上の手順をネットワーク上の全トランスピュータが実行するのに要する時間は、３．２．２に述べたように約１６０μsecである。サブ・プログラムがｓ１からｓ７までを実行するに要する時間は約１５μsecであって、これにトランスピュータ数を乗じた積が、この１６０μsecに含まれる。これが計算を並列化したことによって新たに生じたオーバヘッドである。

このオーバヘッドが計算時間に対する計算回数の関係に、どのように影響するかを示したのが図３．９である。計算は単精度加算である。単一と記した曲線は２．２で述べた単一のトランスピュータによる結果であり、図３．２の曲線を転写したものである。２個、５個、９個と記した曲線は、それぞれ２、５、９個のトランスピュータでリング状ネットワークを構成し、２、５、９個のトランスピュータに計算を配分した場合を示す。９個のネットワークの場合、計算回数$10^2$回以下では計算時間はオーバーヘッド（１６０μsec）に埋もれてしまって一定である。$10^4$回を越えると計算時間はオーバーヘッドより大きくなって単一の場合の約１０分の１となり、９個のトランスピュータを並列に使用した効果が現れてくる。５個および２個のネットワークのオーバーヘッドはループが短くなった分だけ減少し、それぞれ１０１μsecおよび６２μsecである。このグラフは、四則演算のように一回当たりの所要時間が短い計算では、計算時間がネットワークのオーバーヘッドを大きく越える回数まで実行した場合に複数トランスピュータを使用した効果が現れることを示している。

対象とした６個の関数の中では、一回当たりの所要時間が最も長いものは、開平関数

である（表３．１）。単一のトランスピュータでの所要時間は単精度で６８μsec、倍精度で１０７μsecである。これらの値はオーバーヘッドに埋もれるほど小さくはないので計算回数が少なくても複数個のトランスピュータを使用した効果が現れる。図３．１０に９個のネットワーク上での、開平関数の計算時間対計算回数のグラフを示す。このグラフで単一と記した曲線は単一のトランスピュータの場合の曲線（図３．３）を転写したものである。この曲線は、９個のうちの１個のトランスピュータを使用した場合の曲線と、ほぼ一致している。これは、少ない回数でも計算時間がオーバーヘッドを越えるので計算時間に対するオーバーヘッドの影響は少ない。９個使用した場合には、$10^2$回付近から、１個を使用した場合の９分の１に計算時間が短縮されている。

９個のリング状ネトワークにおいて、並列計算の効果が現れる計算回数はいくらかを見るために相対速度対トランスピュータ数の関係を計算回数をパラメータとして示したのが図３．１１と図３．１２である。図３．１１は単精度の加算であり、図３．１２は単精度の開平である。ここで言う相対速度とは、９個のうちの１個を使用した時の計算時間を１、２、３、・・・、９個を使用した時の計算時間でそれぞれ割ったものである。したがって、９個を使用したときの相対速度が９であれば、１個だけ使用したときより９倍速くなったことを意味している。図３．１２で１０回、$10^2$回の曲線に凹凸が多いのは（計算回数）÷（トランスピュータ数）の余り回数をルート・トランスピュータに負担させているためである。余り回数を各トランスピュータに１回分づつ配分すれば、この凹凸は軽減できる。

四則演算と算術関数について、$10^5$回の計算を実行したときの１回当たりの所要時間を表３．１に示す。80286+80287のデータは文献[5]から借用した。

### ３．２．４　定積分計算

本事例では定積分を、計算領域を並列化する手法によって計算する。計算領域の並列化とは積分区間をいくつかの領域に分け、各領域ごとに１つのトランスピュータを割り当てて計算することである。

下記の積分を台形公式によって求めるとすれば、積分区間ａ，ｂにＮ個の分点をとって

$$\int_a^b f(x)\,dx \rightarrow \sum_{j=0}^{N-1} (x_{j+1}-x_j)\,\frac{f(x_{j+1})+f(x_j)}{2}$$

と表される。さらに、これをｍ個の領域に分ければ

$$=\sum_{k=1}^{m}\ \sum_{j=N_k}^{N_{k+1}-1} (x_{j+1}-x_j)\,\frac{f(x_{j+1})+f(x_j)}{2}$$

ただし、0=N1 〈N2 〈・・・〈Nm 〈Nm+1=N
となるので、m個のトランスピュータに

$$\sum_{j=N_k}^{N_{k+1}-1} (x_{j+1}-x_j)\,\frac{f(x_{j+1})+f(x_j)}{2}$$

を並列に実行させることになる。

事例として次の定積分を計算する。

$$\int_a^b f(x)\,dx$$

ここで　a=0.05, b=10.0
$f(x)=x^2 + 1/x^3 + \cos(e^{-x}) + \sin(x^{-4})$

である。

この計算は、前例と同様にルート・トランスピュータを含む９個までのトランスピュータ上で並列に実行される。キィーボードからは、単精度か倍精度かの選択、分点数、計算を分担するトランスピュータの個数を入力する。この入力に応答して計算開始時刻、終了時刻、積分値、計算時間が返される。積分計算のメニュー画面を図３．１３に示す。

各トランスピュータへの領域の配分は分点の番号で与え、各領域の計算結果はトランスピュータごとの計算結果を順次加算しながらネットワークを一巡するようにした。計算の開始指令、終了通知は前事例と同じである。したがってチャネル・プロトコルは図３．１４に示すように宣言した。packetのデータ構造はモード選択（INT）、精度選択（INT）、１区分の値（REAL64）、分点番号（[num.of.transp+1]INT）の順にとってある。sumは各トランスピュータの計算結果を順次、加算して廻すための変数であり（REAL32, REAL64)、start, finishは前事例と同じである。図３．１５に関数f(x)のグラフを、図Ａ５．７にpacketのデータ構造を示す。

分点数を１０$^5$個として積分計算をおこなった結果を表３．２に示す。この表でトランスピュータ数とはネットワーク上の９個のトランスピュータのうち積分計算を配分されたものの個数である。各積分値は倍精度では、上位９桁まで一致しているが単精度では上位３桁までしか一致していない。これは、分点間の１区間の値、すなわち、積分区間を１０$^5$で割った値の誤差が単精度では大きいためである。理論的には分点数を多くとれば積分値の誤差は小さくなるはずであるが、実際にはこの例に見るように、表現できる数値の最小値に限度があるため誤差は大きくなる。ちなみに分点数を１００にした場合

の積分値を表３．３に示す。単精度でも上位６桁まで一致している。この事実は、単精度では１区間の値を（ｂ－ａ）／$10^5 \leq 10^{-4}$より小さくすれば信頼できる桁が上位３桁以下になるであろうことを意味している。倍精度では、この場合も上位９桁まで一致しており、１区間の誤差は同等と考えられる。

本事例のオーバーヘッドの大きさは、分点数を１個にして、つまり１区間をａ＝０．０５からｂ＝１０．０にとれば、約１１１０μsecと求められる。この値は、前事例のオーバーヘッドよりかなり大きいが、これは並列計算を行うまでの前準備に要する時間が大きくなったためである。分点数をパラメータとして相対速度対トランスピュータ数の関係を図３．１６に示す。分点数を多くすれば（計算時間）／（オーバーヘッド）の比は大きくなり、計算を配分されたトランスピュータの個数に比例した効果が得られている。

# 4. おわりに

複数個のトランスピュータで構成されたリング状ネットワークで二つの並列計算を実行させ、処理時間の短縮とオーバーヘッドとの関係を明らかにした。オーバヘッドは処理を複数個のトランスピュータに行わせるための前準備・後始末に要する時間とトランスピュータ相互間の通信に要する時間の合成されたものであり、通信に要する時間は、リング状ネットワークでは接続されたトランスピュータの個数に比例して増加する。単一使用の場合に比較して、複数使用した場合の使用個数に対応した並列化の効果を得るには、個々のトランスピュータにおいて（計算時間）／（オーバーヘッド）＞＞1の条件が成り立つ必要がある。したがって、実際の使用に際しては効果の境界を念頭においてネットワークの構成やアルゴリズムを決めなければならない。

オーバーヘッドの大きさはネットワークの形状と並列処理のアルゴリズムに依存するので、種々の応用にトランスピュータを使いこなして行くには各種のネットワーク構成とそれに最適なアルゴリズムを集積していく必要がある。

今後は、パーソナル・コンピュータに挿入して使用されるA/D、D/A, DI/DOボードと複数トランスピュータとの間でデータのやり取りをするためのソフトウェア・インターフェースの開発を進めていくことを考えている。これは前述した並列化手法の一つである事象の並列化を可能にするものである。また、トランスピュータとI/Oデバイスとのソフトウェア・インターフェースには，iserverと称するソフトウェアがある。これはoccam2ツールセットの一つであって実行モジュールをロードし、そのプログラムの実行をスタート・ストップすると共にルート・トランスピュータにパソコンの資源を提供している。つまり、このソフトウェアは一般のオペレーティング・システムにおけるデバイス・ドライバの役割も分担している。このiserverにA/D, D/A等のインターフェースを追加する方法でもトランスピュータとのデータのやり取りが可能となる。このインターフェース・ソフトウェアの開発は、各種I/Oボードとトランスピュータをパソコン上で使用するための一般的な手法を確立することになろう。

最後に、原子炉制御研究室の藤井義雄氏は、INMOS社のトランスピュータのカタログを丹念に調査し、ソフト／ハード面での必需品の一切を用意した。加速器管理室の花島進氏にはoccam言語プログラムのデバッグで御協力を戴いた。原子炉制御研究室の島崎潤也氏には文献5)に関して貴重な御討論を戴いた。ここに記して感謝の意を表します。

# 文　　献

1) IMS B008 User guide and reference manual 1990 INMOS
2) A tutorial introduction to occam programming 1988 INMOS
（国井、守上訳：　並列処理言語occam2入門　1989 啓学出版）
3) occam2 Reference Manual 1988 INMOS
（井口、荒木、世古訳：　occam2　リファレンス　マニュアル　1990 啓学出版）
4) S708 USER GUIDE 1990 INMOS
5) 井門、辻：　トランスピュータを用いた並列高速演算システムの数値計算性能評価
原子力誌, **30**[11], 999(1988)
6) THE TRANSPUTER DATABOOK 1989 INMOS
7) IMS D7205: IBM/NEC PC Occam2 Toolset delivery manual 1991 INMOS
8) F editor: Preliminary version 1991 INMOS
9) occam2 toolset user manual-part 1: User guide and tools 1991 INMOS
10) occam2 toolset user manual-part 2: occam libraries and appendices 1991 INMOS
11) 木村、福島：トランスピュータによる並列処理　1990　海文堂
12) 山本、他：トランスピュータ入門　1990　日刊工業新聞社

表3.1　トランスピュータによる計算時間
$10^5$回実行したときの一回当たりの時間（$\mu$sec）

| 演算関数 | 単一のトランスピュータ | | 複数個のトランスピュータ：9個 | | *80286 +80287 |
|---|---|---|---|---|---|
| | 単精度 | 倍精度 | 単精度 | 倍精度 | |
| ＋ | 1.493 | 1.620 | 0.130 | 0.229 | 48 |
| － | 1.408 | 1.542 | 0.139 | 0.234 | 50 |
| × | 1.567 | 2.057 | 0.152 | 0.291 | 58 |
| ÷ | 1.800 | 2.527 | 0.194 | 0.344 | 66 |
| sqrt | 68.39 | 107.0 | 6.751 | 11.91 | 105 |
| sin | 15.98 | 33.75 | 1.401 | 2.952 | 505 |
| cos | 14.40 | 27.89 | 1.272 | 1.927 | 500 |
| tan | 18.51 | 36.45 | 1.601 | 3.102 | 375 |
| exp | 22.09 | 40.02 | 1.891 | 3.801 | 345 |
| log | 17.44 | 25.20 | 1.681 | 2.916 | 320 |

＊ 80286+80287のデータは文献５）から借用した

表3.2　定積分計算（分点数：$10^5$）
積分値と所要時間

| トランスピュータ数 | 単精度 | | 倍精度 | |
|---|---|---|---|---|
| | 積分値 | 所要時間 sec | 積分値 | 所要時間 sec |
| 9 | 543.50 | 1.318 | 543.532,259,88 | 2.372 |
| 8 | 543.50 | 1.483 | 543.532,259,85 | 2.671 |
| 7 | 543.51 | 1.695 | 543.532,259,83 | 3.059 |
| 6 | 543.49 | 1.975 | 543.532,259,78 | 3.574 |
| 5 | 543.47 | 2.372 | 543.532,259,82 | 4.296 |
| 4 | 543.49 | 2.983 | 543.532,259,83 | 5.379 |
| 3 | 543.55 | 3.947 | 543.532,259,83 | 7.184 |
| 2 | 543.66 | 5.886 | 543.532,259,83 | 10.796 |
| 1 | 543.69 | 11.750 | 543.532,259,83 | 21.628 |

表3.3　定積分計算（分点数：$10^2$）
積分値と所要時間

| トランスピュータ数 | 単精度 | | 倍精度 | |
|---|---|---|---|---|
| | 積分値 | 所要時間msec | 積分値 | 所要時間msec |
| 9 | 782.923,34 | 2.369 | 782.923,603,40 | 3.656 |
| 8 | 782.923,34 | 2.840 | 782.923,603,77 | 4.521 |
| 7 | 782.923,40 | 2.838 | 782.923,603,95 | 4.517 |
| 6 | 782.923,40 | 3.309 | 782.923,603,88 | 5.376 |
| 5 | 782.923,40 | 3.343 | 782.923,603,87 | 5.379 |
| 4 | 782.923,34 | 3.931 | 782.923,603,88 | 6.456 |
| 3 | 782.923,46 | 4.963 | 782.923,603,88 | 8.406 |
| 2 | 782.923,52 | 6.865 | 782.923,603,88 | 11.867 |
| 1 | 782.923,46 | 12.723 | 782.923,603,88 | 22.700 |

```
        How fast are the following operations performed on a transputer?

                  1)  +
                  2)  -
                  3)  *
                  4)  /
                  5)  sqrt
                  6)  sin
                  7)  cos
                  8)  tan
                  9)  exp
                 10)  log
                 11)  quit
        Select the number of the above arithmetic operators and the functio
 --: 1          ## addition
        1)single or 2)double--: 1
        augend--:1.0
        addend--:2.0
        number of addition--:100000
        time1=19970090
        time2=20119452
        time.ope=149362
        32-bit sum = 3.0
        time for a 32-bit addition = 1.49362004
press any key :
```

```
-File howfast:occ, State = Editing!, Language = Occam
        1 -- 32 bit operation
          ...  32 bit
        2 -- 64 bit operation
          {{{  64 bit
          SEQ
            PRI PAR
              SEQ
                clock ? time1
                SEQ i=0 FOR add.times
                  sum64 := augend64 + addend64
                clock ? time2
              SKIP
```

図3. 1　「四則演算と算術関数の計算」のメニュー画面および
計算時間計測プログラム（単一のトランスピュータ）

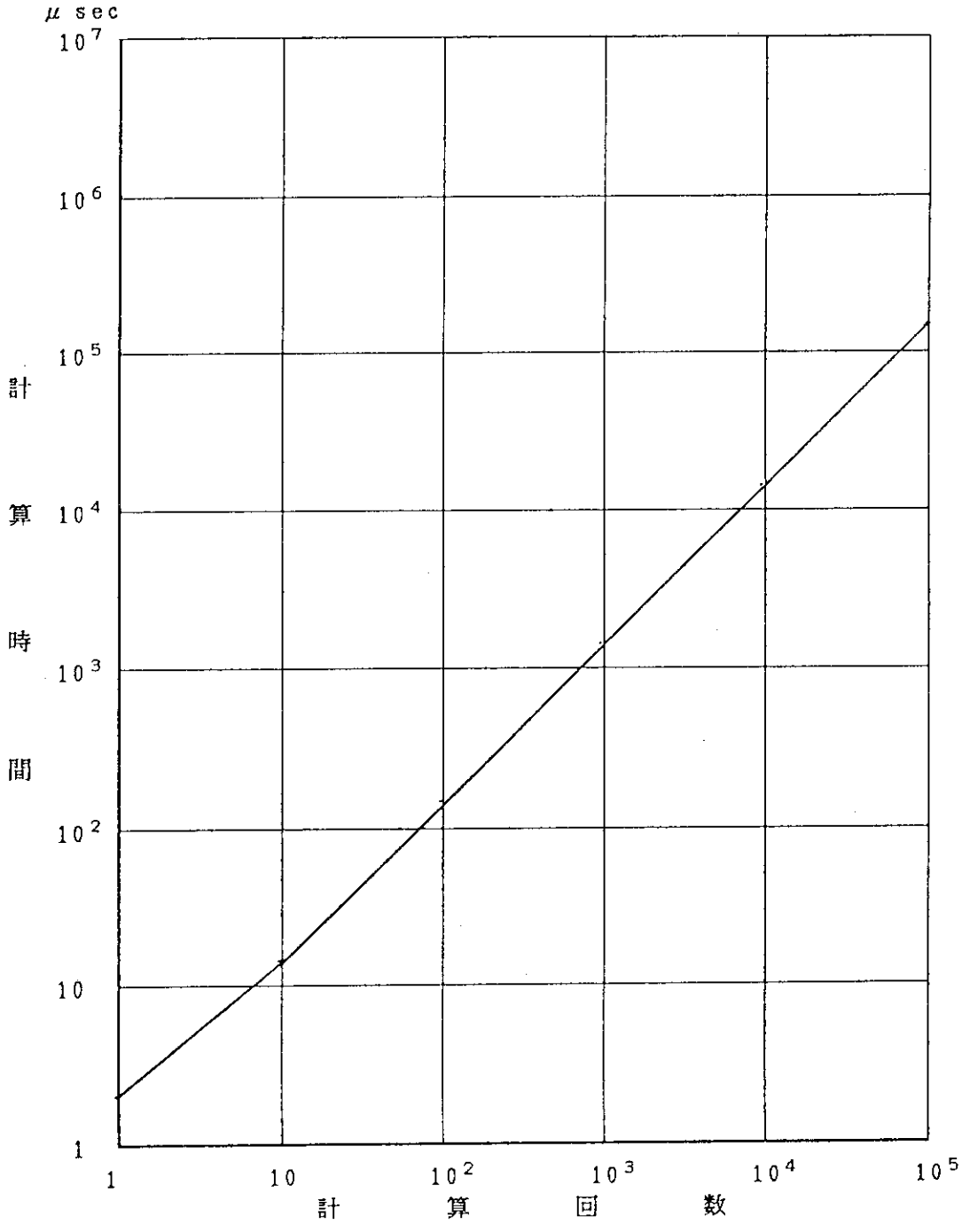



図3.2　単一のトランスピュータによる計算時間（加算）

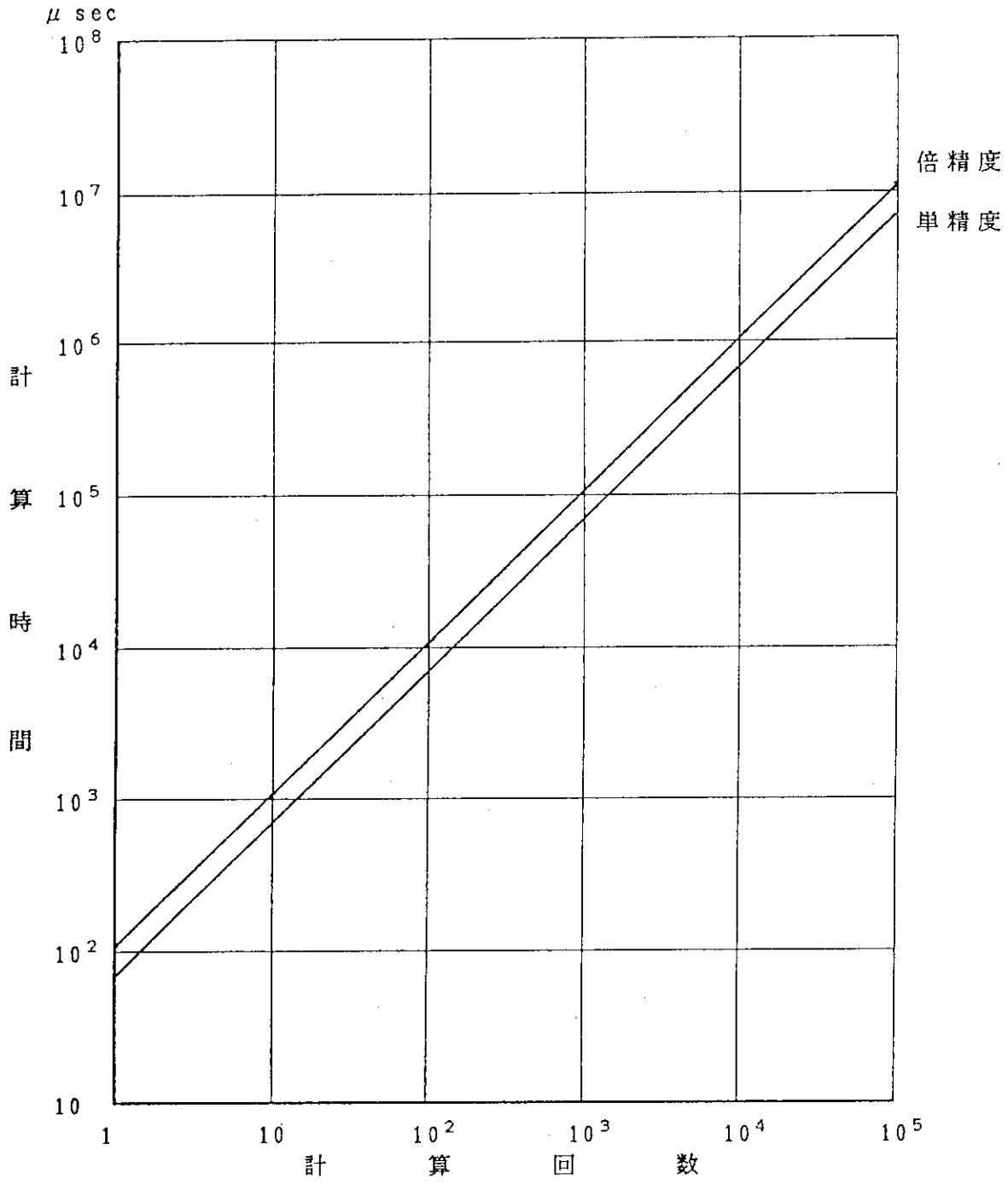



図3.3　単一のトランスピュータによる計算時間（開平）

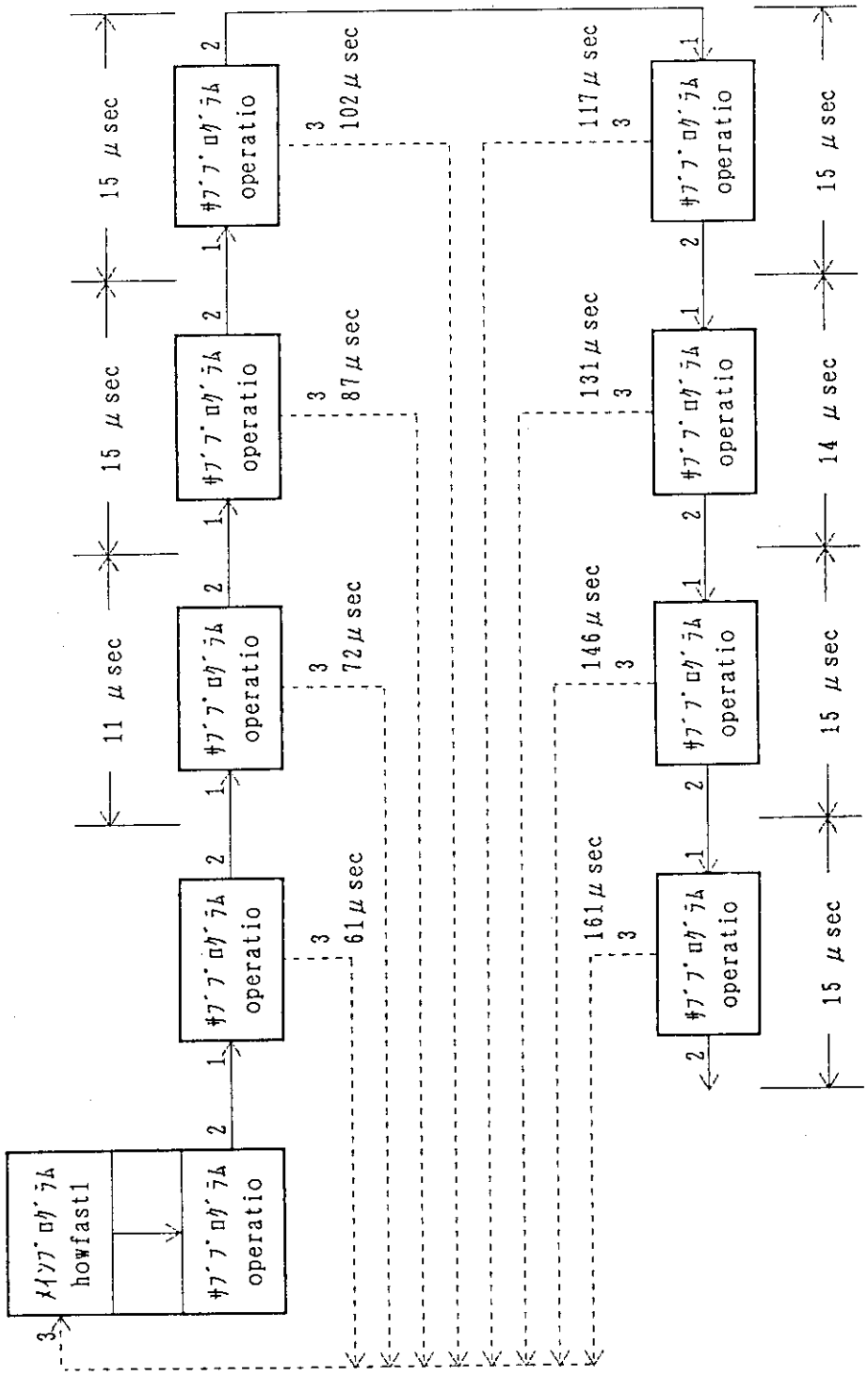



図3.4　トランスピュータが2，・・・，9個のときのネットワークのオーバーヘッド

```
        How fast are the following operations performed on the transputers?

                  1)  +
                  2)  -
                  3)  *
                  4)  /
                  5)  sqrt
                  6)  sin
                  7)  cos
                  8)  tan
                  9)  exp
                 10)  log
                 11)  quit
        Select the number of the above arithmetic operators and the functio
 --: 5          ## square root
        1)single or 2)double--: 1
        variable--:3.0
        number of operations--:100000
        number of transputers--:9
        time1=1329125450
        time2=1329890607
        time.ope=765157
        32-bit value = 1.7320509
        time for a 32-bit square root = 7.65156984
press any key :
```

図3.5　「四則演算と算術関数の計算」のメニュー画面
（複数個のトランスピュータ）

```
File howfast1.occ, State = Editing*!, Language = Occam
{{{  start an operation , get a start time and a stop time
out ! packet; ope.key;sng.dbl.key;repeat.num.t;operand
in ? CASE
  packet; ope.key.r;sng.dbl.key.r;repeat.num.r;operand.r
    clock ? time1
out ! start; 0(INT)
in ? CASE
  start; cpu.no
    out ! finish
in ? CASE
  finish
    clock ? time2
}}}
```

図3.6　メインプログラムの時間計測部分(howfast1. occ)

```
File operatio.occ, State = Editing, Language = Occam

  SEQ
    going := TRUE
    WHILE going
      SEQ
        {{{  input and output parameters
        input ? CASE
          packet; ope.key;sng.dbl.key;repeat.num;operand
            output ! packet; ope.key;sng.dbl.key;repeat.num;operand
        }}}
        {{{  get 'cpu.no', send 'cpu.no'
        input ? CASE
          start; cpu.no
            output ! start; (cpu.no + 1(INT))
        }}}
        {{{  operations
        CASE ope.key
          1 -- addition
            ...  addition
          2 -- subtraction
            ...  subtraction
          3 -- multiplication
            ...  multiplication
          4 -- division
```

図3.7 a　サブプログラム内のトランスピューター間通信(operatio. occ)

```
File operatio.occ, State = Editing, Language = Occam
        }}}
        {{{  operations
        CASE ope.key
          1 -- addition
            {{{  addition
            SEQ
              CASE sng.dbl.key
                1 -- 32-bit operation
                  SEQ
                    operand1.s := ( REAL32 ROUND operand[0] )
                    operand2.s := ( REAL32 ROUND  operand[1] )
                    SEQ i=0 FOR repeat.num[cpu.no]
                      result32 := operand1.s + operand2.s
                2 -- 64-bit operation
                  SEQ
                    SEQ  i=0 FOR repeat.num[cpu.no]
                      result64 := operand[0] + operand[1]
                ELSE
                  SKIP
              input ? CASE
                finish
                  output ! finish
            }}}
          2 -- subtraction
```

図3.7 b　サブプログラム内の加算実行部分 (operation. occ)

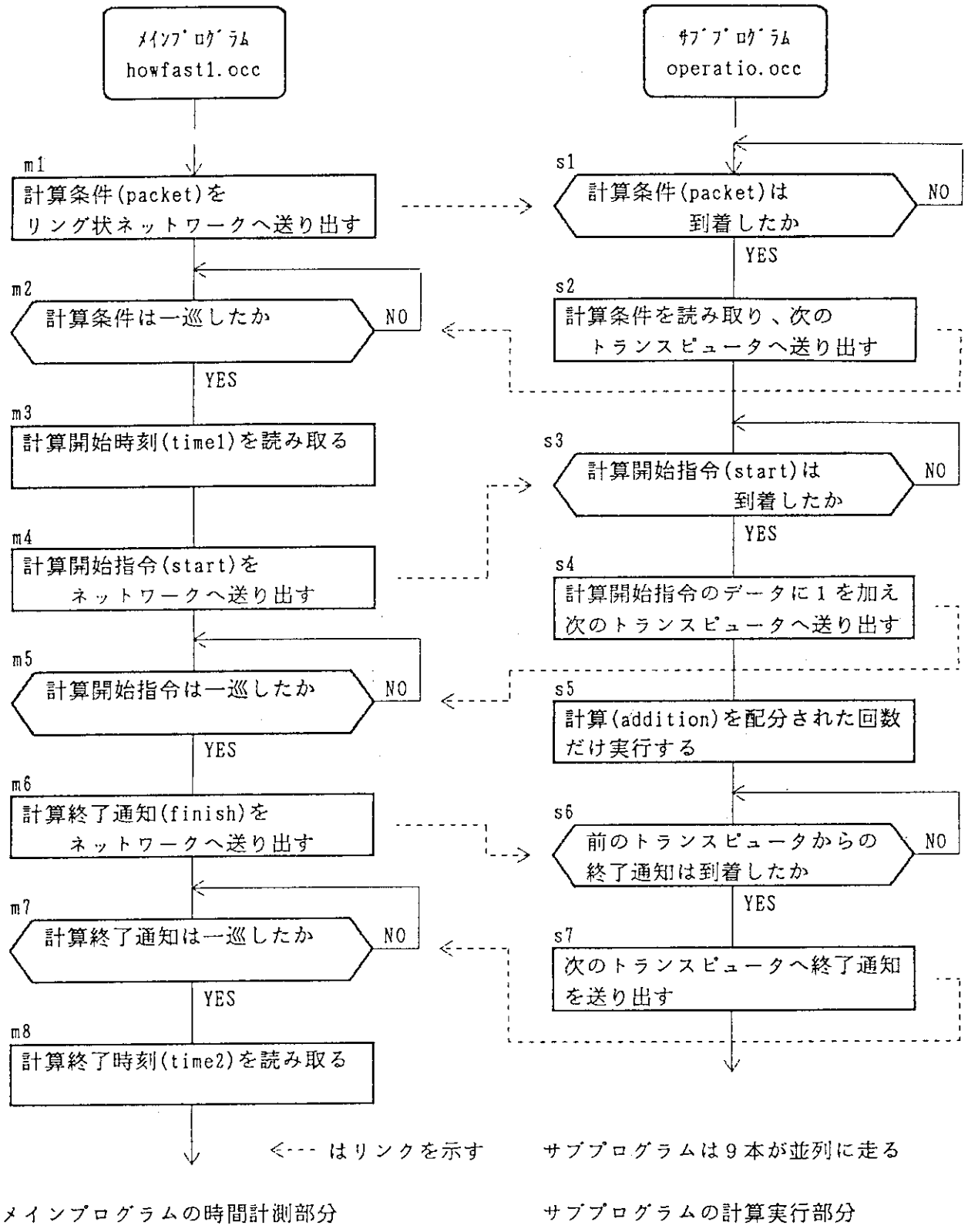



図3.8　ネットワークのオーバーヘッド発生状況

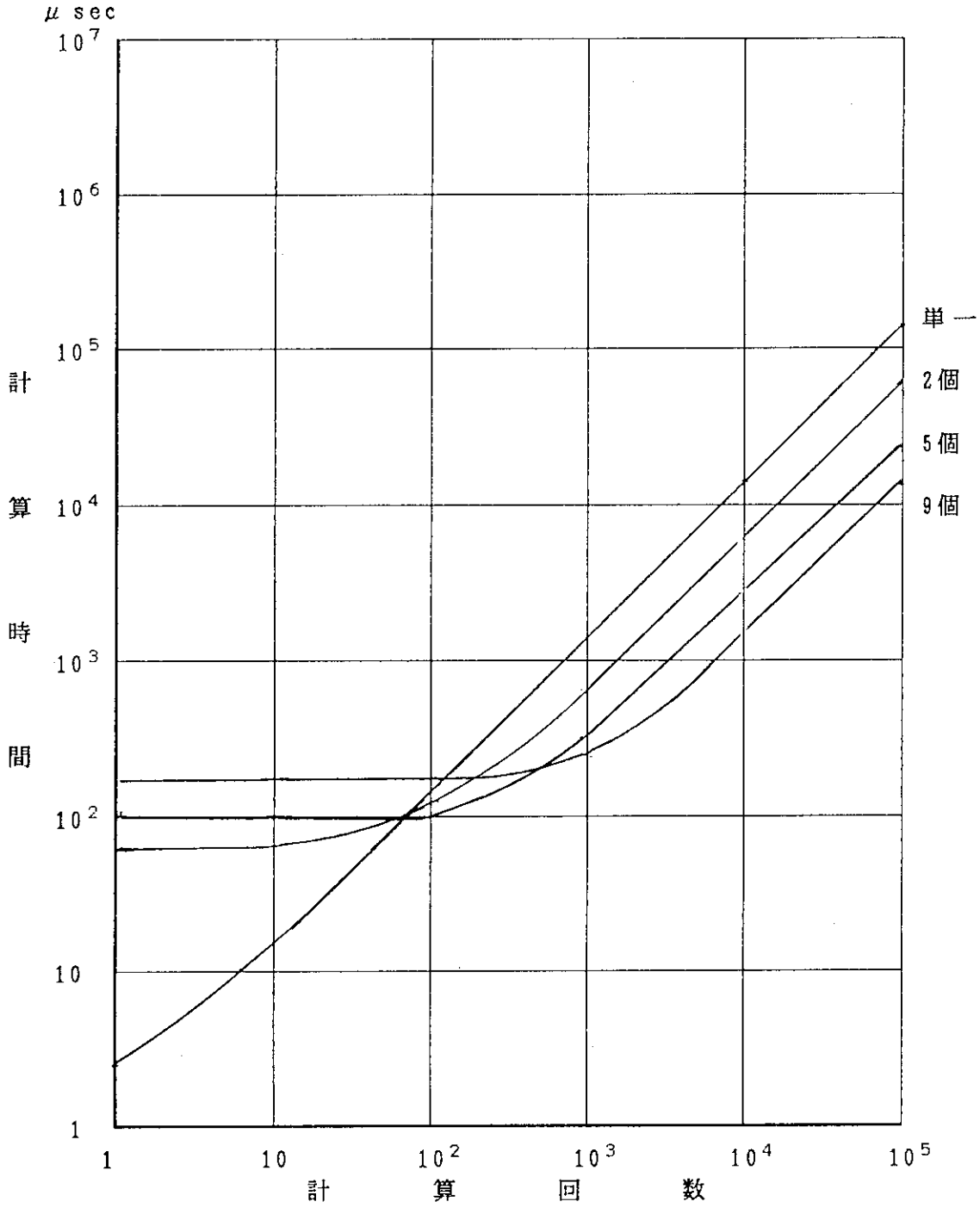



図3.9　複数個のトランスピュータによる計算時間（加算）

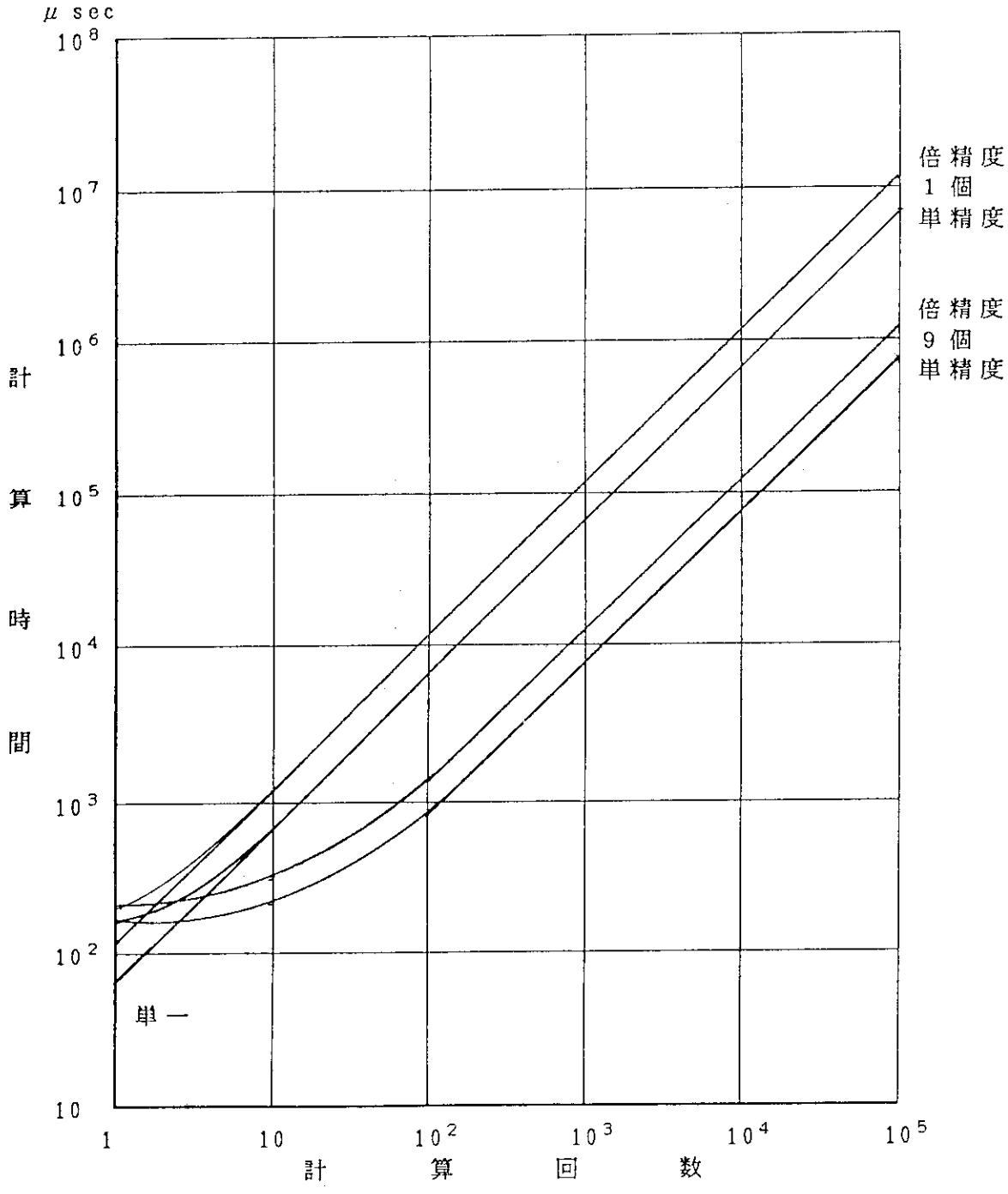



図3.10　複数個のトランスピュータによる計算時間（開平）

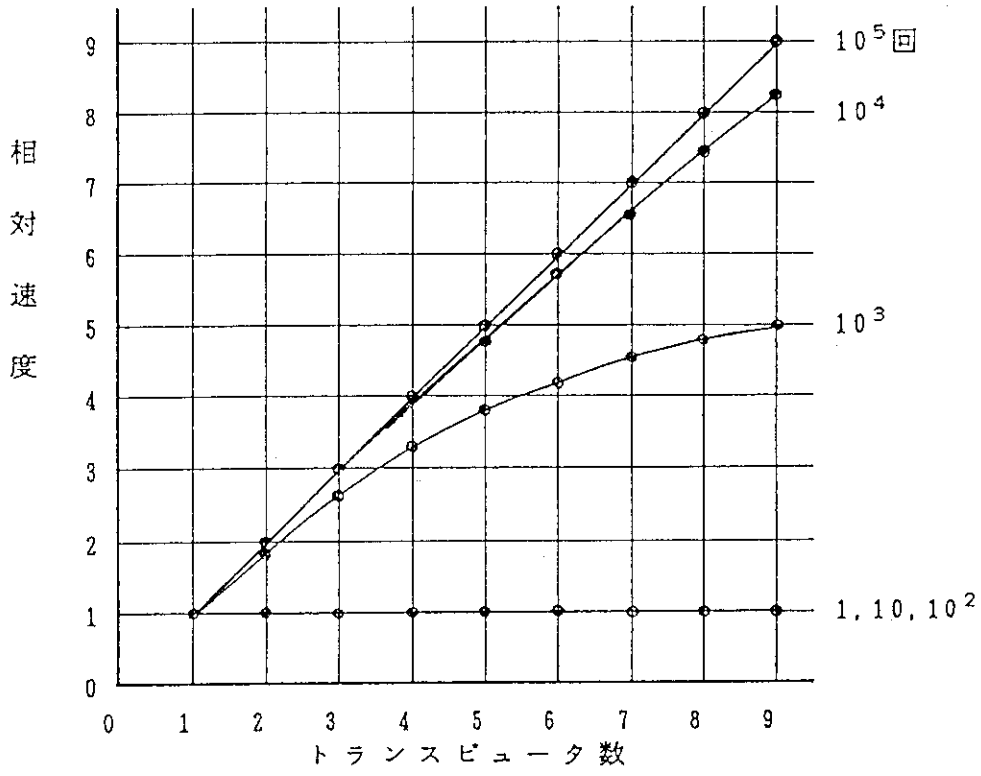



図3.11　計算回数による並列計算の効果（加算）

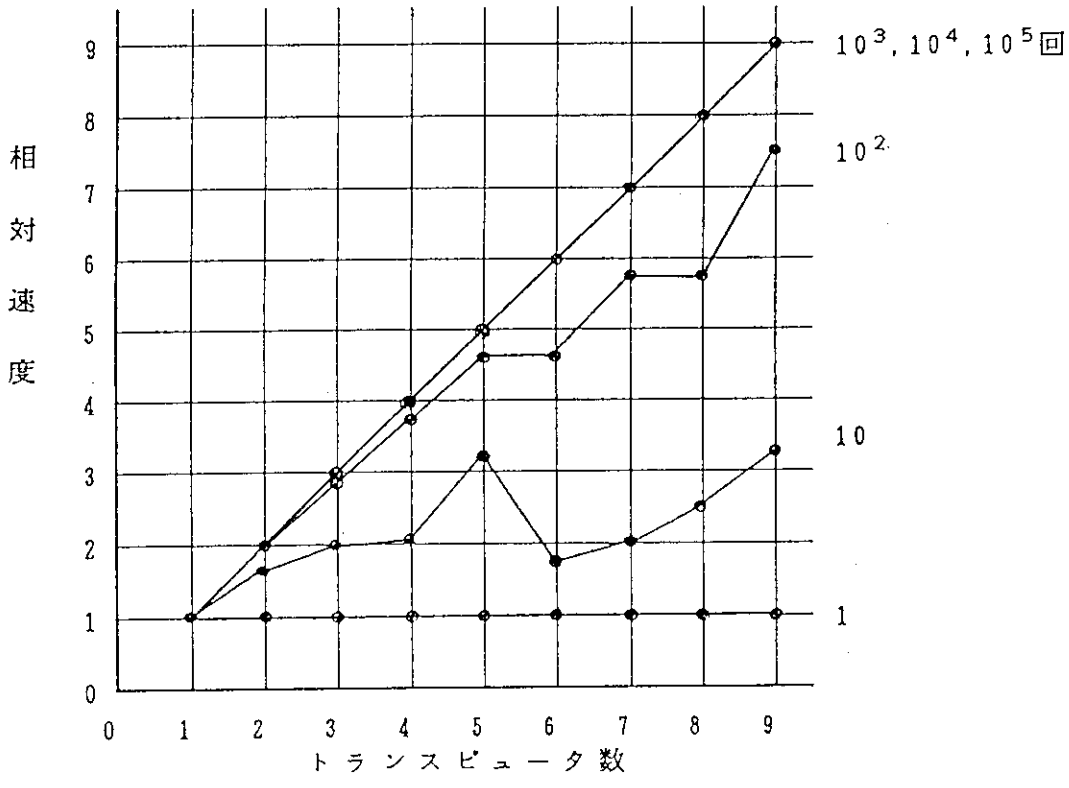


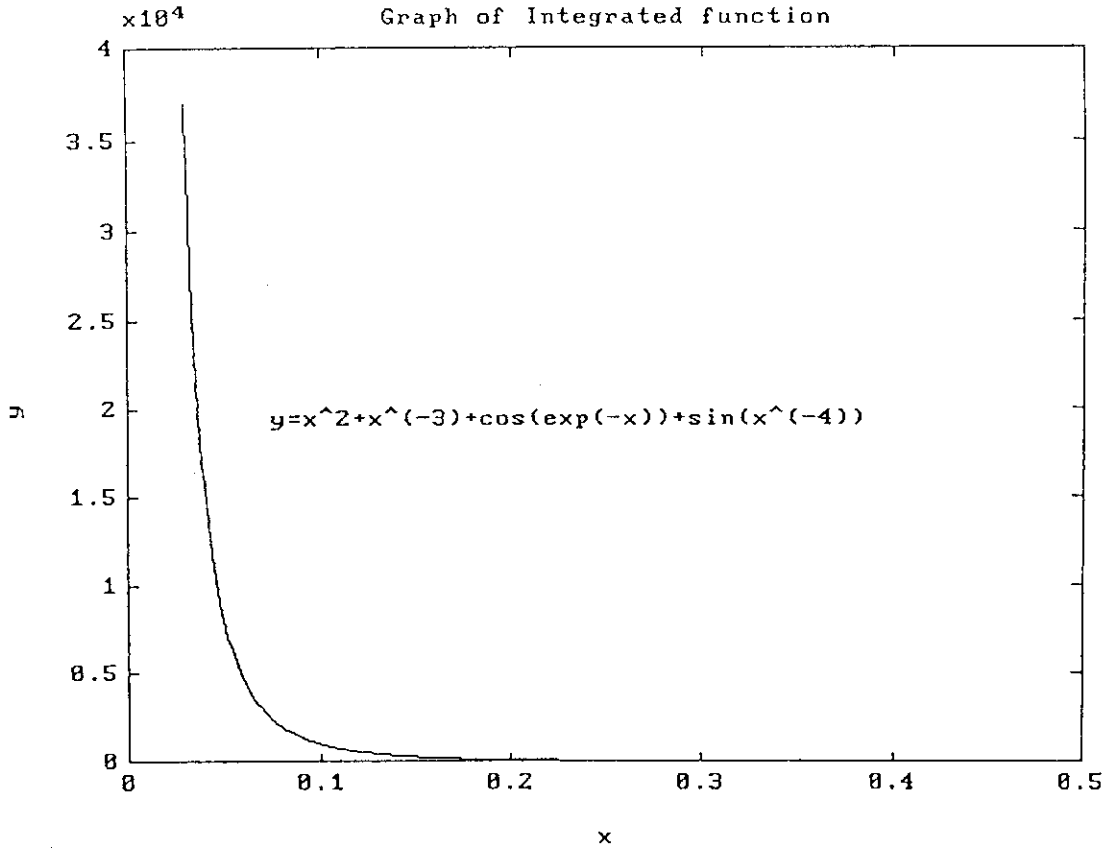

図3.12　計算回数による並列計算の効果（開平）

```
          Numerical integration of function f(x)

                    1)  integration
                    2)  quit

          Select the number of the above operation --: 1
          * precision  1)single or 2)double--: 1
          * number of divison on integral range--:100000
          * number of transputers(1, ... ,9)--:9
          ## integration is going on.....
          time1=49782089
          time2=51100637
          time in ticks=1318548
          32-bit integration= 543.508667
          time in uSEC = 1318548.0
press any key :
```

図3.13 「定積分計算」のメニュー画面

```
File integ.inc, State = Editing*, Language = Occam
{{{  integ.inc
-- nine transputers are installed, all for work
VAL num.of.transp IS 9:
PROTOCOL INTEG
  CASE
    packet; INT;INT;REAL32;REAL64;[num.of.transp + 1]INT
    sum; REAL32;REAL64
    start; INT
    finish
:
}}}  integ.inc
```

図3.14 「定積分計算」用チャネルプロトコルの宣言



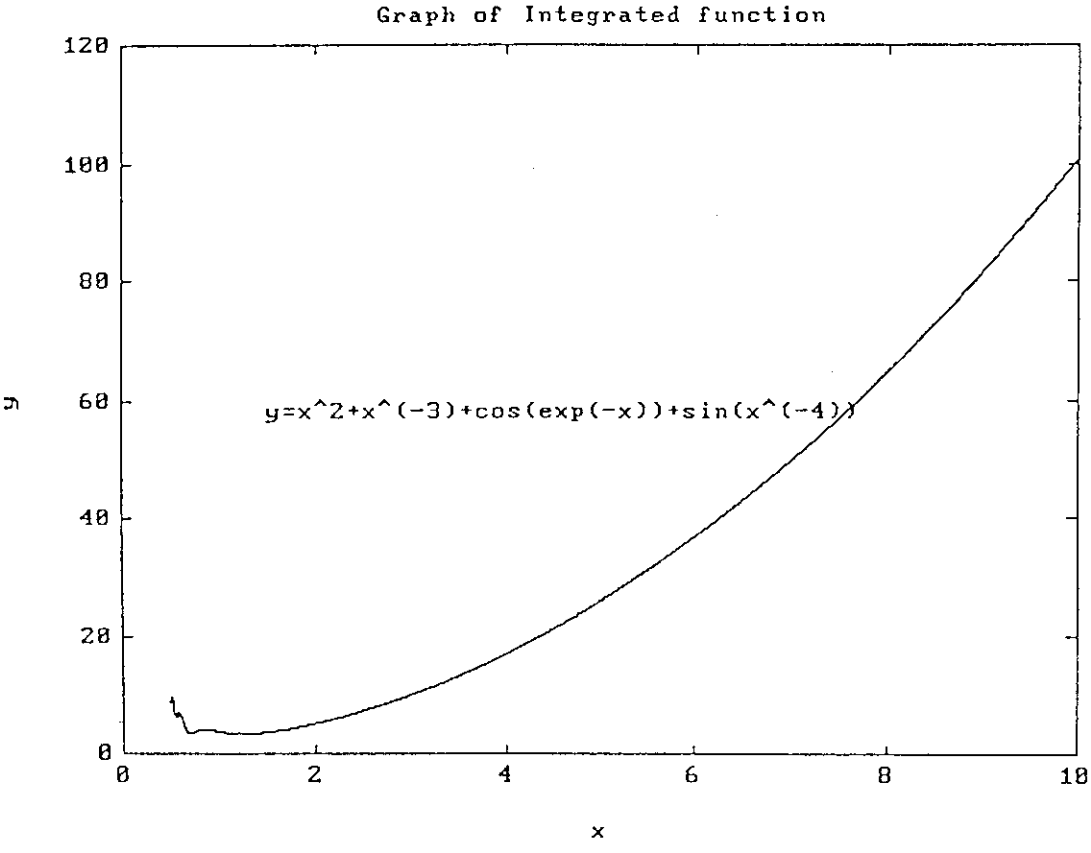




図3.15　関数 $f(x)=x^2+1/x^3+\cos(e^{-x})+\sin(x^{-4})$のグラフ

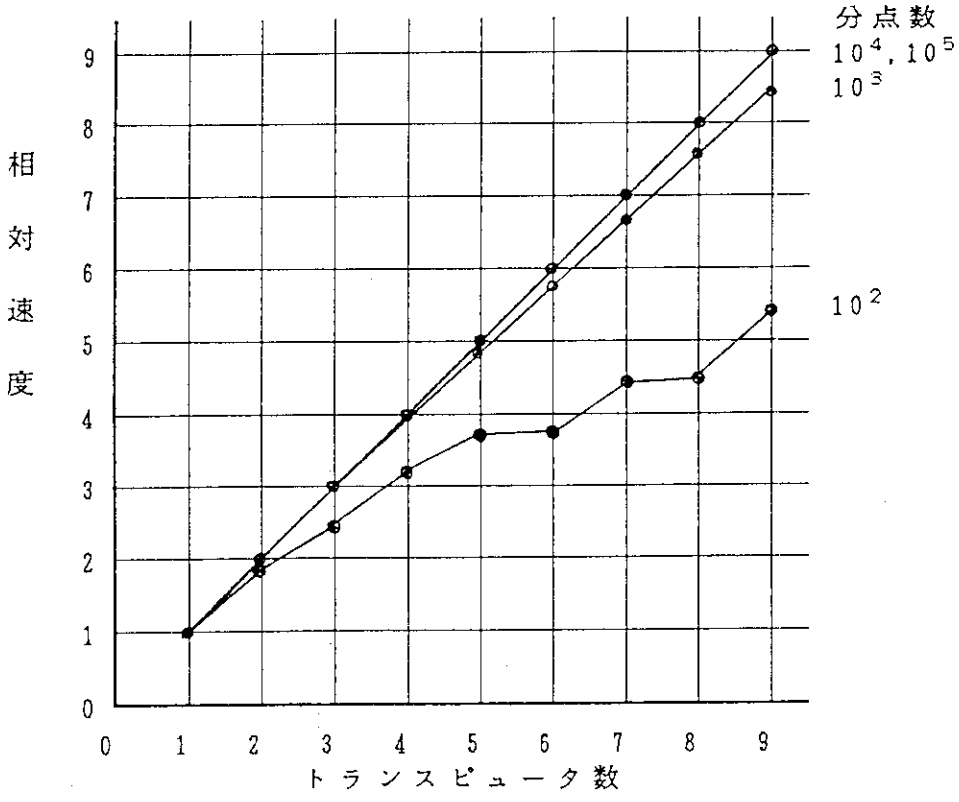



図3.16　分点数による並列計算の効果（定積分　単精度）

# 付　　録

トランスピュータを生産しているINMOS社から提供されている言語には，ANSI-C，Pascal，FORTRAN，Ada，occam2がある。occam2はoccamの最新版である。日本ではＣ言語人口が多い現状を反映してか、あるいは、なじみにくいoccamの習得が敬遠されてか、トランスピュータ言語においてもＣ言語の使用者が最も多いそうである。事実、複雑な構造型データを記述する能力においては、occamはPascalやCに劣る。しかし、事象の時間的な推移を微細に記述することが要求されるプログラム、たとえば、ハードウェア組み込みプログラムを記述するには、これらの言語のうちでoccamが適していると言えよう。

occamやトランスピュータについての解説書は少なく、TDS(Transputer Development System)については入門書[11)12)]があるが、occam2ツールセットについては未だ皆無であり、INMOS社が提供しているマニュアルに頼るしかない。

この付録では、occam2ツールセットのうち本研究で使用したツールについて使用経験にもとずいた解説をする。また、計算事例プログラムを掲載し、解説を加えた。

occam2ツールセットは、INMOS-Limited社の製品であり、マニュアル類は本文の文献欄に掲載した1)，2)，3)，4)，6)，7)，8)，9)，10)を参照した。

# A1　ツールセットのインストール

occam2ツールセット（D7205）[7]は１５個のツールから成るソフトウェアであって，IBM-PC/AT + MS-DOSあるいはNEC-PC + MS-DOS上で走る。インストールの手順についてはマニュアルに記述されているが、COMPAQ DESKPRO 386/25e(IBM-PC/AT互換期）へのインストール例について要点を示す。コマンド行末の'）'はリターンを意味する。

```
>a:)


>a:¥install  a  c )
   （ 以下ＣＲＴ上のメッセイジにしたがって次々とディスクを入れる。）
INSTALL COMPLETE


>cd  ¥d7205¥iserver )
>copy  isvrb04.exe  iserver.exe )


>mifes  config.sys )
    SHELL=COMMAND.COM  C:¥  /P  /E:1024
    FILES=20
    DEVICE=C:¥D7205¥ITERMS¥BANSI.SYS


>mifes  autoexec.bat )
    PATH=...;C:¥D7205¥ISERVER;C:¥D7205¥TOOLS;C:¥D7205¥ITOOLS;C:¥D7205¥ISPY
    SET  IBOARDSIZE=#200000
    SET  ISEARCH=C:¥D7205¥LIBS¥
    SET  IDEBUGSIZE=#200000
    SET  TRANSPUTER=#150
    SET  ITERM=C:¥D7205¥ITERMS¥PCBANSI.ITM

      インストールの確認
>cd  ¥mine )
```

```
>copy  ¥d7205¥examples¥oc¥simple.occ )
>imakef  simple.b8h )      (simple.mak と simple.l8hが生成される)
>make  -fsimple.mak )
    oc simple /t8 /h /o simple.t8h
    ilink /f simple.l8h /t8 /h /o simple.c8h
    icollect simple.c8h /t /o simple.b8h
>iserver  /se  /sb  simple.b8h )
Please type your name: Inomata )
Hello Inomata
```

インストール後のconfig.sysとautoexexc.batを次に示す。

```
C:¥>type config.sys
REM     1991/6/14
REM
DEVICE  = C:¥BIN¥DOS¥CEMM.EXE 1024 AUTO
DEVICE  = C:¥BIN¥DOS¥VDISK.SYS 1024 128 64 /E:8
DEVICE  = C:¥BIN¥DOS¥CACHE.EXE 1024 ON /EXT
DEVICE  = BIN¥DOS¥ANSI.SYS
INSTALL = BIN¥DOS¥SHARE.EXE
INSTALL = BIN¥DOS¥FASTOPEN.EXE C:=(100,10) /X
DEVICE  = C:¥D7205¥ITERMS¥BANSI.SYS
FILES   = 20
BUFFERS = 100,8 /X
SHELL   = COMMAND.COM C:¥ /P /E:1024

C:¥>
```

```
C:¥>type autoexec.bat
@ECHO OFF
PATH=C:¥BATCH;C:¥BIN¥DOS;C:¥BIN¥UTY;C:¥BIN¥MSC5;C:¥DV;C:¥D7205¥ISERVER;C:¥D72
TOOLS;C:¥D7205¥ITOOLS;C:¥D7205¥ISPY;C:¥NORTON
REM CHARSET MAIN=C:¥BIN¥DOS¥THINUS ALT=C:¥BIN¥DOS¥FONTUS
PROMPT $P$G
SET TMP=C:¥
SET HELP=C:¥BIN¥UTY
SET MIFES=C:¥BIN¥UTY
SET PKNO386=YES
SET IBOARDSIZE=#200000
SET IDEBUGSIZE=#200000
SET ITERM=C:¥D7205¥ITERMS¥PCBANSI.ITM
SET FTERM=C:¥D7205¥FY¥DATA¥PCFOLD.ITM
SET ISEARCH=C:¥D7205¥LIBS¥
SET TRANSPUTER=#150
REM MODE LPT1=LPT2
REM FR C: /SAVE
rem dosshell

C:¥>
```

# A2 フォールディング・エディタの常用コマンド一覧

occam2ツールセットにはフォールディング・エディタ[8]が標準で装備されている。このエディタはツールセット内のディレクトリ¥Fに収納されていて、Fエディタと略称されている。このエディタの特徴はソース・テキストを階層的に編集できるように作られている点にある。当初、使用経験の無いFエディタの修得を避けて、使い慣れたMIFESを用いて作業を始めたが，MIFESはoccam2の編集には極めて作業性が悪いことが分かり、Fエディタへ切り替えた。これは、occam2が字下げによってプロセスを区分しているのでPascalやCの字下げとは文法上の意味が異なり、厳格な規則が決められているからである。Occam2を使用するにはFエディタの修得は必須であるといえよう。

Fエディタを修得するにはマニュアルを読むよりも、Fエディタに付属しているファイル¥F¥DATA¥FTUTOR.OCCを読むことから入るのが最短コースである。このファイルは、それ自体がoccam2のソース・テキスト・ファイル(.occ)の形式を持っていて、Fエディタにかけて編集が可能なファイルにできている。このテキスト・ファイルにはFエディタの操作方法が系統的に記述されていて、CRT上に表示される説明を読みながら、指示にしたがって編集作業をおこなえば、エディタ・コマンドを理解できるようになっている。この演習が優れているのは、説明文やプログラム例がoccam2のソース・テキストであるので、演習での操作が即、実際のoccam2ソース・テキストの操作であるという点にある。

Fエディタがoccam2向きに作られているとは言え、シンタックス・チェックをしてくれないので、シンタックス・エラーの無いソース・テキストを作るには、occam2の文法に慣れる必要があるのは当然である。章末に使用頻度の高いコマンドを一覧表にまとめた。Fエディタにはメニューやアイコンが無いので作業効率を上げるには、コマンドを記したプレートをキィーボード上に置くような工夫が必要である。

Fエディタを使用するには、まず次のようにインストールする必要がある。

1） ¥D7205¥F¥TOOLS¥F.EXE を ¥D7205¥ITOOLS¥F.EXEへコピーする。

2） autoexec.batの中へ下記を加える。

```
SET FTERM=C:¥D7205¥F¥DATA¥PCFOLD.ITM
```

3） ファイル¥D7205¥F¥DATA¥PCFOLD.ITMの中に下記の下線部を加える。

startupfile "C:¥D7205¥F¥DATA¥fold.stp"

4） ファイル¥D7205¥F¥DATA¥FOLD.STPの中に下記の下線部を加える。

command "C:¥D7205¥F¥DATA¥fold.cmd"

## フォールディング・エディタのコマンド一覧

### カーソル移動

| | |
|---|---|
| [CURSOR DOWN] | カーソルを下へ |
| [CURSOR UP] | カーソルを上へ |
| [CUESOR LEFT] | カーソルを左へ |
| [CURSOR RIGHT] | カーソルを右へ |
| [(END OF) LINE→] | カーソルを行末へ。スクリーンの外まででると第一行目にpaningと表示される |
| [(START OF) LINE←] | カーソルを行頭へ |
| [WORD LEFT] | カーソルを一語左へ |
| [WORD RIGHT] | カーソルを一語右へ |
| [Backspace] | カーソルの左の一文字を消してその位置へ |
| [RETURN] | カーソルを行末に置いてリターンを押すと空行が挿入される。カーソルは字下げされた位置へ |

### 画面移動

| | |
|---|---|
| 「LINE DOWN] | カーソルを停止させたままテキスト全体を一行だけ上へスクロールする |
| [LINE UP] | カーソルを停止させたままテキスト全体を一行だけ下へ |
| [PAGE UP] | 一ページ分前（上）へ |
| [PAGE DOWN] | 一ページ分後（下）へ |

### 行移動・行コピー

| | |
|---|---|
| 「MOVE LINE] | テキストあるいはフォールドの一行を移動する。一回目の打鍵でその行を取り除きバッファへ入れる。カーソルを移動先へ合わせ二回目の打鍵で移る。 |
| [COPY LINE] | カーソル位置のテキストあるいはフォールドの一行を、その行の次にコピーする |
| [PICK LINE] | テキストあるいはフォールドを移動するために拾ってバッファへ入れる。次々と連続した行を拾うことができる |
| [PUT] | バッファの内容をカーソル位置へ吐き出す。復数行を一度に吐き出す |
| [BROWSE]<br>[COPY PICK] | テキストをread onlyにしておいて拾い読みする。[BROWSE]から抜けるには再度[BROWSE]を打鍵する[COPY LINE],[PICK LINE]は使えなくなる。[BROWSE][COPY PICK][BROWSE][PUT]の順でコピーできる |

### 行位置

| | |
|---|---|
| [LOCATE LINE] | 打鍵+[RETURN]でカーソル行の番号が表示される |

打鍵するとLine number? 100カーソルが100行目に移る

デリート

[DELETE] カーソルの下にある一文字を削除する

[DELETE→] カーソルの右にある一文字を削除する

[←DEL WORD] 左の一語を削除する

[DEL WORD→] 右の一語を削除する

[DEL TO EOL] カーソルから行末までを削除する

行削除と回復

[DELETE LINE] カーソル行を削除する。フォールドも削除できる

[RESTORE LINE] [DELETE LINE]で削除した行を元へ戻す。[DELETE LINE]の直後に使え

サーチ・リプレース

[NEW REPLACE] New replace string:fox新しく置換する語をバッファへ与える

[NEW SEARCH] Search for:dog新しくサーチする語をバッファへ入れる

[SEARCH] 現在のカーソル位置からdogをサーチしてｄで止まる。

[REPLACE] そのdogをfoxで置き換える。

フォールド生成と解除

[CREATE FOLD] 二回打鍵する。一回目でフォールドの始点を決め、二回目でフォールドの終点を決める。これは既に存在しているテキストをフォールドに閉じこめる。

...■が表示される。ここにコメントを入れる。

一回目の打鍵てカーソルの位置が新フォールドの行頭となるから、字下げも可能。

字下げは...の左側のスペースを調整してもよい

[OPEN FOLD]されたものをスペースを消して左へ移動させることは出来ない

{{{□□フォールドの始点へインサートはできない

State=Creatingが表示されているときはカーソル移動しか許されない

[REMOVE FOLD] フォールドを展開する

フォールドの操作

[TOP OF FOLD] フォールドの第一ページへ

[BOTTOM OF FOLD] フォールドの最終ページへ

[OPEN FOLD] カーソルをフォールド・ラインに合わせて打鍵すると、そのフォールドが開かれる。テキストが展開される

[CLOSE FOLD] 上で開いたフォールドを閉じる

[ENTER FOLD] 一つのフォールドが展開されて表示される

[EXIT FOLD]　展開されている一つのフォールドが閉じられる

キィーストローク・マクロ

[DEFINE MACRO]　二回続けて打鍵すると定義されていたマクロは消去される。一回目の打鍵以降のキィー入力をバッファに貯める。文字列をキィー入力してから二回目を打鍵する

[CALL MACRO]　打鍵ごとに上で入力されたテキストとして入力される

[SAVE MACRO]　定義されているマクロをフォールドに作り込んでおく。このフォールドはMacro keyと呼ばれる

[GET MACRO]　カーソルをMacro keyに合わせて打鍵するとMacro keyの内容がカレント・マクロになる

OSコマンド・Fコマンド

[COMMAND]　[COMMAND]コマンド[OBEY]の順でキィー入力する

[OBEY]　Fコマンドなら即時実行される。OSコマンドならオープン・ファイルはディスクへセイブされ、OSへ判断が任される

例

```
[COMMAND]print pcfold.hlp[OBEY]
[COMMAND]copy pcfold.hlp myhelp.hlp[OBEY]
[COMMAND]dir[OBEY]
[COMMAND]f pcfold.hlp[OBEY]
```

Fコマンドでは次のものが多用される

insert filename: カーソル位置へfilename内のテキストを挿入する

#include filename: あるファイルをエディット中に別のファイルを呼び出してエディットできる。[FINISH]で前のエディットへ戻る

ヘルプ

[HELP]　キィーボードの配置説明が表示される。読みにくくて使えない

終了

[FINISH]　現在のセッセションを終えて呼び出し元へ戻る

例

| | | | |
|---|---|---|---|
| Editor | [HELP] → | helpfile | エディタからヘルプへ |
| Editor | ← [FINISH] | helpfile | ヘルプからエディタへ |
| OS | f filename | Editor | OSからエディタへ |
| OS | ← [FINISH] | Editor | エディタからOSへ |

セイブ

[SAVE]　現在のテキストをファイルへセイブする

# A 3　occam 2 ツールの操作手順

occam2ツールセット[9)10)]には１５個のツールが含まれているが、ここでは本事例のプログラムをコンパイルしてから実行するまでに使用したツールに限って、その手順を述べる。本文3.2.2で述べた四則演算と算術関数のプログラム（ソース・コード・ファイルhowfast1.occ, operatio.occ ）を対象にツールを作用させる。howfast1.occはルート・トランスピュータ上にロードして走らせるメイン・プログラムであり、operatio.occはルート・トランスピュータを含めてネットワーク上のすべてのトランスピュータにロードして走らせるサブ・プログラムである。章末に掲載した手順の一例を参照しながら説明する。

１）、２）　先ず、テキスト・ファイルhowfast1.occ, operatio.occ をoccam2コンパイラにかける。ファイルのエクステンション.occは省略してよい。オプションにはトランスピュータの機種を指定する。この場合は、T800であるので、/800とし、続けてリターンをキィーインする。ソース・テキストは初期段階ではエラーを多く含んでいるので！ more を加えておくとエラー・メッセイジを読みながらコンパイルを続けることができる。

３）、４）　コンパイルした結果、エクステンション .tco が付いたオブジェクト・ファイルが生成される。これをリンカ ilink にかけてライブラリとのリンクをおこなう。ライブラリはエクステンション_.libを持つ。/f はリンカ・インクルード・ファイル（ .lnk ここではoccam8.lnk )を指定するオプションであって、T800シリーズ固有のインプット・ファイル名、リンカ・ディレクティブ、コメント等が記述されているテキスト・ファイルであって、供給されたツールセットに含まれている。　/t800 は１）、２）と同じくトランスピュータの機種指定である。

５）　これ以降はoccamだけにある独特な処理である。occonfはコンフィギュラと称して、通信シャネルとトランスピュータとをプロセスに割り付けるコードを生成し、ファイルhowfast1.cfb へ書き込む。

以上の操作を文法エラーが無くなるまで繰り返す。

６）　文法エラーを皆無にできたら、ilink（ 3),4) ）　の出力ファイル _.lnk あるいは _.c8h を５）の出力ファイル howfast1.cfb を使用して、最終出力であるブータブル・ファイル（ howfast1.btl ）にまとめるためコード・コレクタ icollect にかける。

7）　ブータブル・ファイルはネットワーク上のトランスピュータにロードされるコードが納められているファイルである。このコードをロードする際にはネットワークが正しく構成されていることを確認する必要がある。ispy4s はネットワークの接続状態を調査して、その結果をCRT上に表示してくれる。（図 A4.5）

8）　調査結果の表示を見て、もしネットワークが指定どうりに接続されていないなら、ネットワークを意図する形状（ここではリング状）に接続する必要がある。コマンド・ラインにあるb008soft, b008hard はそれぞれソフトウェア・リンク、ハードウェア・リンク、の仕様書である。リンク接続ソフトウェアmms2.b4は、これらの仕様にもとづいてリンクの接続とチェックを行う。

9）　8）でネットワークの接続を行った後は、正しく接続されているか否かを確認するため、再度ispy4sを実行させる。7）、8）、9）は、電源断かネットワークの変更をしない限り、一度だけ行えば良い。

10）　ホスト・サーバiserverはディスク上のブータブル・ファイルから実行コードを読みとって、それをネットワーク経由で個々のトランスピュータへロードし、実行を開始させる。オプション /se はエラー・フラグがセットされたとき実行を中断してMS-DOSへ戻るためのためのものであり、/sb は指定したファイルhowfast1.bt1からプログラムをロードすることを指定するためのものである。ホスト・サーバはコンソール・デバイスをもっていないルート・トランスピュータへホスト・コンピュータ（パーソナル・コンピュータCOMPAQ　DESKPRO　386／25e）の資源を提供する役割も担っている。
この段階では論理エラーを摘出し、Ｆエディタでソース・テキストを訂正した後、1）へ戻って繰り返す。

ソース・テキストを変更するごとに1）から6）までのキィー入力を繰り返す手間を省くにはUNIXでおなじみのmakeを使用すればよい。makeへの入力ファイルmakefile（ファイル名 _.mak ）を生成するためのツールがimakefである。makeはoccam2ツールセットには標準装備されていないので、本事例ではTurbo-Cのmakeを利用している。

以上の各ステップでの入力ファイルと出力ファイルのエクステンションの関係を図A3.1に示す。occam2ツールセットでは、その特殊性からエクステンションの種類は多いが、1）～10）に現れたエクステンションを知っておけば十分である。

## occam2 ツールセット操作手順の一例

```
1) >oc  howfast1  /t800  ( ! more )
2) >oc  operatio  /t800  ( ! more )
3) >ilink howfast1.tco hostio.lib streamio.lib convert.lib snglmath.lib
        dblmath.lib /f occam8.lnk /t800 ( ! more )
4) >ilink operatio.tco hostio.lib streamio.lib convert.lib snglmath.lib
        dblmath.lib /f occam8.lnk /t800 (! more )
5) >occonf  howhast1.pgm  ( ! more )
```

以上の操作を文法エラーが無くなるまで繰り返す

```
6) >icollect  howfast1.cfb  ( ! more )

7) >ispy4s
8) >iserver  /sb  mms2.b4  b008soft  b008hard
9) >ispy4s

10) >iserver  /se  /sb  howfast1.btl
```

ここで論理エラーが発生したら１）へ戻って繰り返す

makefileの利用

```
11) >imakef  howfast1.c8h        1),3)を連続実行するmakefileを生成する
12) >imakef  operatio.c8h        2),4)を連続実行するmakefileを生成する
13) >imakef  howfast1.btl        1)-6)を連続実行するmakefileを生成する
14) >make  -fhowfast1.mak        1),3)を連続実行する
15) >make  -foperatio.mak        2),4)を連続実行する
16) >make  -fhowfast1.mak        1)-6)を連続実行する
```

# A4　ネットワークの接続・点検ツール

IBM-PC/AT用マザーボードIMS-B008[1]には専用のネットワーク配線ソフトウェアmms2.b4と、ネットワークの配線状態を確認するためのネットワーク点検ソフトウェアispyがある。これらのツールはoccam2ツールセットとは別に提供されている。

## A4．1　ネットワーク接続ソフトウェア　mms2.b4

本事例のトランスピュータ・ネットワークはIBM-PC/AT用マザーボード（IMS-B008）上に9個のトランスピュータを装着して構成した。このボードはハードウェア接続とソフトウェア接続によりネットワークを構成できるように作られている。ハードウェア接続は図A4．1に示すようにプリント・カード上で接続されていて、接続の変更は不可能である。ソフトウェア接続はリンク・スイッチ（IMS-C004）をソフトウェアmms2.b4で設定することにより、さまざまな形状のネットワーク接続が可能である。

ハードウェア接続仕様はHardwire description language[4]で、ソフトウェア接続仕様はSoftwire description language[4]で記述したファイルを作成し、これをmms2.b4への入力ファイルとする。mms2.b4はハードワイヤ仕様にもとずいてボード上の実際の配線を点検するとともに、ソフトワイヤ仕様にもとずいてリンク・スイッチの設定を行う。

本事例では各トランスピュータ上に配置するプログラムを同一とするため、ネットワークの形状はリング状とした。このネットワークのハードワイヤ仕様ファイルと、ソフトワイヤ仕様ファイルの内容を図A4．2と図A4．3に示す。ソフトワイヤ仕様ファイルには、9個のトランスピュータを直線状に接続し、最後のトランスピュータ（SLOT 9）のリンク3を先頭（SLOT 0）のリンク3へリンク・スイッチを介して接続することが記述されている。つまり図A4．1の点線の接続を行うことをmms2.b4へ指示している。ハードワイア仕様ファイルにはマザーボードIMS-B008上の素子の個数とリンク端子の接続状態が記述されている。SIZESはT2(T212)、C4(C004)、SLOT（トランスピュータ用ソケット）、EDGE（ボードから外部への接続端子）の個数を定義している。T2CHANはT2とC4の接続を記述している。HARDWIREではトランスピュータのリンク相互の接続を指定している。C4が先頭に記された行はリンク・スイッチC004の端子までの接続を指定している。C004内部の接続は上述したソフトワイヤ仕様で指定する。この仕様の書き方についての詳細はS708 USER GUIDE[4]に述べられている。

mms2.b4を使うにはディストリビューション・ディスクIMS-S708B-2 B008 Support Softwareをコピーするだけでよく、特別なインストレーションのための作業は必要ない。MMS2.B4のコマンド・シンタックスは

```
>iserver  /sb  mms2.b4  softwire  hardwire
```

である。このコマンドを入力すると図Ａ４．４に示すメニューが表示される。mms2.b4へネットワークの接続を指示するには'S'をキィー入力する。mms2.b4はソフトワイヤ仕様にもとづいてリンク・スイッチC004の内部接続をおこなう。mms2.b4からMS-DOSへ抜け出すには'Q'をキィー入力する。メニューには１３個の選択枝があるが常用するのは、この二つだけである。

Ａ４．２　ネットワーク点検ソフトウェア　ispy

マザーボードIMS-B008上に構成されたネットワークの接続状態を調べるために、ネットワーク点検ソフトウェアispyが提供されている。ispyには、いくつかのツールが含まれているが本事例ではispy4sというスモール・モデル（６０ＫＢ）を使用した。

このソフトウェアにはドキュメントは無く、フロッピー・ディスクが一枚だけ供給される。README を読むだけで使用できる、簡単なツールである。電源投入直後に下記のコマンドを入力すると

```
>ISPY4S
```

図Ａ４．５に示す表が現れる。この表は同図に示すネットワークの接続状態を調べた結果を表している。このネットワークは電源投入直後のものであって、ボード上のハードウェアによる配線だけがされていることを示している。ここで、前項で述べたmms2.b4を走らせてソフトウェア接続を行った後、再度ispy4sを走らせると

```
>iserver  /sb  mms2.b4  boo8soft  b008hard
>ispy4s
```

図Ａ４．６に示す表がＣＲＴ上に表示され、同図に示すリング状ネットワークが構成されたことを確認できる。両図においてトランスピュータに与えられた番号（＃の下の数字、表の左端列）が図A4.5と異なっている。これはispy4sが点検作業を進めながら確定したものから番号ずけを行っているためらしい。

この表を見て、ネットワークの接続を確認した後にプログラムをロードすれば、つまり、

```
>iserver  /se  /sb  howfast1.btl
```

を入力すれば各トランスピュータにはプログラムが正しく配置され、実行が開始される。

# A５　並列計算のプログラム例

本文３．２．２で述べた並列計算の二つの事例、すなわち１）四則演算と算術関数および２）定積分計算のプログラムについて、各プログラムの構成概要、チャネル・プロトコル、複数個のトランスピュータへのプログラムの割り付けおよび通信リンクのチャネルへの割り付け、ソース・リストの解説等を述べる。

## A５．１　四則演算と算術関数

### １）　プログラム構成の概要

四則演算あるいは算術関数を複数トランスピュータ上で並列計算させるプログラムのうちユーザが記述したプログラムに限って説明する。occam2ツールセットに含まれているインクルード・ファイルやライブラリ・ファイル内のプログラムについては触れない。

本プログラムは下記の４個のファイルから構成されている。

1. howfast1.inc
   - ネットワークを構成しているトランスピュータの個数を定義している。
   - 通信リンクを介して転送されるデータの名前（タグ）と、そのデータ構造（プロトコル）を定義している。このファイルにおける諸定義はoccamに独特なものである。
2. howfast1.occ
   - 本事例のメイン・プログラムであり、計算や関数を選択するメニュー、計算の結果、計算時間を表示するマン・マシン・インターフェース等から成る。
   - 各トランスピュータへ与える計算種別、数値の精度、計算回数、オペランドあるいは変数値をネットワーク上のトランスピュータへ送り出す。
   - 計算時間を計測する。
3. operati.occ
   - このプログラムは各トランスピュータの上にロードされて、与えられた計算条件で計算をおこなう。
4. howfast1.pgm
   - _.pgmが付いたプログラムはコンフィグレーション・プログラムと呼ばれる。これはoccamに特有のプログラムであり、ネットワーク上のハードウェアとソフトウェアの対応づけを定義している。
   - 各トランスピュータの名前を定義し、その名前を用いて各トランスピュータのタイプ（ここではT800）、メモリ容量、トランスピュータ相互間のリンク接続を定義している。
   - 通信チャネルのプロトコルの定義、各プロセッサへのプロセスの割付をおこなっ

ている。

これらのファイルの相互関係を図Ａ５．１に示す。howfast1.incは３個の、ファイルhowfast1.occ, operatio.occ, howfast1.pgm にインクルードされて使われる。howfast1.occとoperatio.occはコンパイル、リンクされた後、howfast1.pgm内のプログラムにもとづいてトランスピュータへ割り付けられる。

２）　プログラムの要点

1. howfast1.inc

・定数定義ＶＡＬによりネットワークを構成しているトランスピュータの個数が９個であることを定義している。

・チャネルとはプロセス間の通信路のことであり、論理的な概念である。リンクとは物理的なトランスピュータ間の通信路のことである。したがってチャネルが二つのトランスピュータ間のリンクである場合と、一つのトランスピュータ上の共有メモリーである場合とがある。本プログラムでは、これら二種類のチャネルを使用していて、ルート・トランスピュータに配置されたhowfast1とoperatio間のチャネルだけが共有メモリー・チャネルである。

チャネルを使うには、先ずプロトコルを定義しておかなければいけない。occamのプロトコルとはチャネルへ送り出すデータのフォーマットの宣言であり、PascalやCの構造型データの宣言に似ている。ここでは使用するプロトコルにPARAMという名前を与えて、混成プロトコルを宣言している。タグはpacket, start, finishの３個があり、送り出し側はデータの頭にタグをつけて送り出し、受け取り側は先ず送られてきたタグを見て、宣言されている順にデータを受け取り、そのデータ型の変数に格納する。したがって送り側と受け側は初めからプロトコルを知っていなければならない。データはループを一巡して送られるので、howfast1.occとoperati.occは送り側であり、受け側でもあるから図Ａ５．１に示すように、両ファイルはhowfast1.occをインクルードしなければならない。ここでは３種類のプロトコルpacket, start, finishを宣言している。packetは指定された演算・関数を表す整数、実数値精度を表す整数、各トランスピュータに割り当てられた計算回数を格納するための９個の要素から成る整数配列、オペランドあるいは変数値を納めた２個の要素から成る倍精度実数配列で構成されている。startは計算開始指令であるとともに各トランスピュータの計算回数を納めた整数配列の添え字である。その値はルート・トランスピュータでは０であり、次のトランスピュータへ送られるごとにoperati.occ内でインクリメントされる。finishはデータを持たずタグだけであって、各トランスピュータが計算の終了を知らせ合うのに使う合図のようなものである。（図A5.2）

2. howfast1.occ

このファイルのソース・リストについては、プログラムの骨格を成す基本部分と、加算時間測定ルーチンだけを掲載する。他の演算ルーチンのプログラム構造は加算の場合と

同じであるので省略する。

Ｆエディタで作成されたソース・リストはプリント・アウトするとホールドがすべて展開されるのでプログラムの構造が見えにくい。そこでホールディング・エディタで作成したファイルの特徴を生かして、ホールドを１レベルづつ逐次展開したリストを示す。

・レベル１

宣言部とプログラム本体から成る。このレベルでは一般のプログラムとまったく同じ形態をしている。

・レベル２－ａ

宣言部はインクルード・ファイルとライブラリを宣言している。howfast1.incについては前述した。これ以外のファイルの使い方についてはoccam2ツールセット・マニュアルpart2に記述されている。

・レベル２－ｂ

プログラム本体部分は、計算時間測定ルーチンをadditionからlogarithmまでを定義した部分と、WHILEループでくくられた部分から成る。後者はメニューの表示と、選択された計算の時間測定を実行する。

・レベル３－ａ、ｂ

レベル２のWHILEループ内では、先ず図Ａ５．３に示すメニューを表示し、各演算・関数に対応した番号の選択を待つ。番号が選択された後に次々と現れるプロンプトに対して条件をキィーボードから与えていく。図Ａ５．３では加算が選択され、実数精度：単精度、被加数：1.0、加数：2.0、加算回数：１０$^5$回、トランスピュータ数：９個との条件を与えると、所要時間:13019μsec、１回の加算時間：0.13019μsecと表示される。

・レベル３－ｃ

ここでは四則演算と算術関数の実行時間を計測する１０個のルーチンaddition～logarithmが定義されているが、このうちadditionだけを展開する。

前述した加算の条件、すなわち精度、被加数、加数、加算回数、トランスピュータ数をキィボードから受け取る。PRI PARからSKIPまでは、occamで提供されている１μsec単位のタイマを使用して時間を計測している。与えられた条件を各トランスピュータへ送りだした後、加算開始指令を出し、開始時刻と終了時刻を読みとる。所要時間を表示するルーチンは単精度の場合と倍精度の場合に分けている。

・レベル４－ａ

加算条件をキィーボードから受け取るルーチンである。図Ａ５．３に示したプロンプトを順次、表示しながらオペレータへ条件の入力を促す。

・レベル４－ｂ

本文３．２．３に述べたチャネル・プロトコルPARAMで宣言されているデータ・フォーマットに適合するように入力された条件を整えている。与えられた加算回数とトランスピュータ数から各トランスピュータへ配分する加算回数を算出し、それを配列に入れる。加算が割り当てられないトランスピュータには加算回数０を与える。また、被加数、加

数、演算種別コード（加算は'1'である）を準備する。

・レベル４－ｃ

計算時間を計測している部分である。リング状に構成されたネットワークへチャネルoutを介して、packetというタグをつけたデータを送り出す。このデータはループを一巡して各トランスピュータに受け取られた後、送り出し元へ戻り、チャネルinから受け取られる。この時点の時刻をtime1へ読みとり、計算開始指令startをネットワークへ送り出す。タグstartに先導された整数は、ルート・トランスピュータを出発するときの値は０であるが、次のトランスピュータはそれを読みとった後インクレメントして、さらに次のトランスピュータへバトンタッチするという手順で次々と送られる。startが一巡したことを確認した後、終了問い合わせfinishを送る。finishが戻ってきて時刻をtime2へ読みとる。

・レベル４－ｄ

加算一回に要した時間を算出し、先に得たtime1, time2と、両者の差をCRTに表示する。

・レベル４－ｅ、ｆ

単精度、倍精度で計算したときの和と、加算一回当たりの所要時間を表示している。

3. operatio.occ

このファイル内のプログラムはネットワーク上の全トランスピュータにロードされて同時に実行される。

・レベル１

このプログラムはquitが入力されてMS-DOSへ戻るまで、WHILEループ内を繰り返し実行する。ループ内は次の三つの部分に分かれている。

(1) 前述したhowfast1.occレベル４－ｃにおいて送り出された条件パラメータを読みとって自分のメモリーへ格納した後、受け取った条件を直ちに次のトランスピュータへ送り出す。

(2) 計算開始指令を受け取ったら、それを次のトランスピュータへ渡す。

(3) 指示された計算を指示された回数だけ実行する。

・レベル２－ａ

チャネルinputを介して送られて来た計算条件を受け取って自分のメモリーへ格納し、直ちにチャネルoutを介して次のトランスピュータへ送り出す。この条件データはタグpacketが付けられたプロトコルにしたがって並べられたデータ列である。条件データ列がリング状ネットワークを一巡してメイン・プログラムに戻ると、次に、整数型データを伴った計算開始指令startが送られてくる。このデータは計算開始指令であるとともに、それを受け取ったトランスピュータの番号でもある。各トランスピュータは、これを計算回数を格納した配列の添え字として利用して、自分の計算回数を確定する。この番号には１が加えられ、チャネルoutputを介して次のトランスピュータへ送り出される。

・レベル２－ｂ

指定された演算・関数を表すコード'1'～'11'にしたがってＣＡＳＥで分岐され、そのコードに対応した計算が実行される。

・レベル３－ａ

演算・計算を実行するプロセスのうち加算プロセスだけを示す。オペランドは倍精度（６４ビット）で送られて来るので単精度を指定したときは倍精度を単精度に変換してから指定回数の加算を行う。計算終了の問い合わせfinishに対する応答は、計算が終了していればfinishを次のトランスピュータへ送ることによって行う。

4. howfast1.pgm

このファイルにはネットワークの定義、トランスピュータへのプロセスの割り付け、チャネルとリンクの対応づけ等が記述されたプログラムが納められている。これは、occamに固有のプログラムであって、occam2の付属機能であるコンフィグレーション・ランゲージ（configuration language）[9)10)]で記述されている。

・レベル１

このプログラムはハードウェア記述部（hardware description）とソフトウェア記述部（software description）から成る。

・レベル２－ａ

ハードウェア記述部が述べている意味は次のとうりである（図Ａ５．４）。

(1) ９個のトランスピュータにB008.t[0]～B008.t[8]という名前を与える。

(2) トランスピュータのタイプ（すべてT800）とメモリー容量を定義する。

(3) B008.t[0]のリンク０はHostlinkを介してHOSTへ接続することを宣言する。

(4) トランスピュータが持っている４本のリンクのうち図に示すリンクが相互接続に使われていることを宣言する。

・レベル２－ｂ

ソフトウェア記述部では、チャネル名とそのプロトコルの宣言、チャネルのリンクへの割り付け、およびプロセスへのプロセッサの割り付けを行っている。ハードウェア記述部の意味を図示した図Ａ５．４にソフトウェア記述部の意味を加えて図示すれば、図Ａ５．５になる。

## Ａ５．２　定積分計算

### １）　プログラム構成の概要

このプログラムはＡ５．４と同一のネットワーク上で実行される。したがってファイルの構成もＡ５．１と同一である（図Ａ５．６）。プログラムは次の４個のファイルから成る。

1. integ.inc

・ネットワークを構成しているトランスピュータの個数とチャネル・プロトコルを

定義している。

・プロトコルpacketは図Ａ５．７に示すデータ・フォーマットになっている。sumは各トランスピュータの積分値を加算しながらループを巡回して集めてくるためのものである。startとfinishは前例と同じである。

2. integbos.occ

・本プログラムのメイン・ルーチンである。メニュー、積分結果、所要時間の表示を行う。

・各トランスピュータへ与える積分区間、実数値精度、１区間の長さをネットワーク上のトランスピュータへ送り出す。

・計算時間を計測する。

3. integfol.occ

・与えられた積分区間について積分を行い、その結果を前のトランスピュータまでの部分和に加えて、それを次のトランスピュータへ送り出す。

4. numeinte.pgm

・ネットワークの形状は前例Ａ５．４と同形であるので、ハードウェア記述部はhowfast1.pgmと同一である。

・ソフトウェア記述部もプロセスの名前が異なるだけでＡ５．１と同じである。

２）　プログラムの要点

1. integ.inc

ネットワークに連結するトランスピュータの個数（num.of.transp）が９個であることを定義している。また、チャネル・プロトコルINTEGを宣言している。これは４つのプロトコルから成る混成プロトコルである。各タグとそのデータ構造を図Ａ５．７に示す。

2. integbos.occ

・レベル１

宣言部とプログラム本体から成る。

・レベル２－ａ

このプログラムに必要なインクルード・ファイルとライブラリを宣言している。前述したinteg.inc以外はシステムが提供しているファイルである。

・レベル２－ｂ

メイン・プログラムintegbosの定義である。チャネル・プロトコルSPはツール・セット内のhostio.inc[10]で定義されている。プログラム本体は、定積分計算を行う手続きintegrationの定義と、WHILEループ内での計算の実行とから成る。ループの内部では、先ず画面をクリアーし、メニューを表示する。次に選択された動作（積分計算あるいはＯＳへ戻る）を行う。

・レベル３－ａ

CRT画面をクリアーし、メニューを表示する。スクリーンへの通信チャネルのプロトコルＳＳはhostio.inc内で定義されている。ここで使用しているサブルーチンSS._はライブラリstreamio.lib[10]にある。

・レベル３－b

キィーボードからの動作選択コードを変数keyへ受け取る。keyの値が１なら積分計算を実行し、２ならWHILEループの条件をFALSEにしてＯＳへ戻る。

・レベル３－c

積分計算を行う手続き（procedure）の定義である。動作選択コード１を受け取ったときは積分計算をおこなう。計算は次の順序で進められる。

1. 計算の条件として実数値精度、分点数、計算を割り当てるトランスピュータ数を入力する。
2. 計算開始時刻を読みとった後、各トランスピュータへ配分する分点数とチャネルへ送り出すデータを準備する。
3. 計算のための準備が終了するとネットワーク上のトランスピュータは次々と計算を開始する。
4. 計算に要した時間を単精度の場合と倍精度の場合に分けて表示する。

・レベル４－a

手続きintegrationを呼び出し、積分計算を実行する。計算終了後、計算結果の積分値と所要時間が表示される。この後、任意のキィーをたたけばメニューへ戻る。

・レベル４－b

手続きintegrationの宣言部である。

・レベル４－c

積分計算の条件を入力する。実数値の精度、分点の個数、計算を分担するトランスピュータの個数を入力すると計算を実行中であるという表示をする。

・レベル４－d

計算開始時刻を読みとる。トランスピュータへ配分する分点の数、１区間の値を求め、部分和をクリアーし、動作コードを積分'１'とする。

・レベル４－e

計算条件の発送から計算終了時刻の読みとりまでの実行順序は次のとうりである。

1. 上記で準備した計算条件データつまりpacketに連なるデータをチャネルoutからネットワークへ送り出す。一巡してきた後、これをチャネルinから受け取る。
2. 計算開始指令startをチャネルoutへ送り出す。これも一巡して戻って来るのでチャネルinから受け取る。
3. 計算終了通知finishと部分和sumを送り出す。これを受け取ったトランスピュータは割り当てられた計算を終了した後sum32あるいはsum64へ自分の区間の積分値を加える。これらが戻って来たら受け取り、終了時刻をtime2へ読みとる。

・レベル４－f

計算開始時刻、終了時刻、所要時間をティックス単位で表示する。

・レベル４－ｇ、ｈ

単精度および倍精度の積分値と所要時間（μsec単位）を表示する。

2. integfol.occ

このプログラムはネットワーク上の全トランスピュータにロードされて、同時に実行される。実行される内容は、計算条件、計算開始指令、終了通知、積分値の受け渡しと積分計算である。

・レベル１

変数宣言をしてからループに入る。quitが入力されるとループから脱出してMS-DOSへ戻る。ループ内では計算条件と計算開始指令をリレーしたあと、計算条件に含まれているope.keyが'１'なら積分を実行し、２ならMS-DOSへ戻る。

・レベル２－ａ

定数の定義と変数の宣言である。

・レベル２－ｂ

積分条件、つまりタグpacketに連なるデータ列をチャネルinputから受け取り、チャネルoutputへ送り出す。

・レベル２－ｃ

計算開始指令startとともに整数値をチャネルinputから受け取り、それに１を加えてチャネルoutputへ送り出す。この整数値は計算条件のデータ列の中から各トランスピュータの積分範囲を取り出すのに使われる。

・レベル２－ｄ

積分計算を単精度あるいは倍精度で行う。

・レベル３－ａ、ｂ

単精度計算（ａ）、倍精度計算（ｂ）を行っている。あるトランスピュータに与えられた積分領域の下限におけるf(x)の値、および次の分点の位置を求めて繰り返し計算の準備を行う。与えられた分点数points.on.cpu[cpu.no]の値が０であれば、そのトランスピュータには計算が割り当てられていない。０でなければ与えられた分点数だけ繰り返し計算を行う。

計算が終了したらfinishが送られてくるのを待ち、部分和を集める変数sum32,sum64に計算結果を加え、それを次のトランスピュータへ送る。

・レベル４－ａ、ｂ

単精度（ａ）、倍精度（ｂ）で与えられた積分領域について台形公式による計算を行う。

4. numeinte.pgm

このファイルにはコンフィグレイション・プログラムが納められている。このプログ

ラムは、トランスピュータのタイプ、メモリー・サイズ、リンクへのチャネルの割り付け、トランスピュータへのプログラムの配置などを定義して、メイン・プログラムintegbosとサブ・プログラムintegfolがネットワーク上で並列実行できる環境作りを行うプログラムである。

・レベル1

このプログラムはハードウェア記述部とソフトウェア記述部から成り、occamの付属機能であるコンフィグレーション・ランゲージで記述されている。USEディレクティブにより、リンカーの出力ファイルintegbos.c8h, integfol.c8hが参照されているのは、これらのコードを最終出力ファイル（numeinte.btl）に組み込むためである。

・レベル2－a

ハードウェア記述部である。トランスピュータに名前（B008.t）を付けて、それをNODEとして宣言する。ルート・トランスピュータとホストＣＰＵ間のリンクの名前をHoatlinkと宣言する。各トランスピュータのタイプ、メモリ・サイズ、ネットワークの接続を定義する。

・レベル2－b

ソフトウェア記述部である。プロトコルSPはhostio.inc内で宣言されている。本プログラムで使用されている対ホスト間のチャネル（from.isv, to.isv）をハードウェア記述部で宣言されたHostlinkに割り付ける。プロセス間を結ぶチャネル（pipe）のプロトコル（INTEG）を宣言する。トランスピュータB008.t[0]にプロセスintegbosとintegfolを配置し、上で宣言したチャネルとの対応づけを行う。他のトランスピュータB008.t[i] i=1,...,8にはプロセスintegfolを配置し、チャネルとリンクの対応をつける。

# A 6　occam 2 言語の概要

一般のコンピュータ言語は、時間の経過にしたがってプログラムを一行ずつ実行するシーケンシャルな処理しか記述できない。このような言語で並列処理プログラムを記述するには、マルチ・タスク・オペレイティング・システムの機能であるシステム・コールを使用しなければならない。しかも、この場合の並列処理は、個々のタスクが時分割で実行される見かけ上の並列処理である。

トランスピュータ・ネットワークは、人間社会をモデル化していると考えることができる。たとえば一つの企業では一人一人の人間が、お互いに連絡を取り合いながら秩序ある企業活動を展開している。これは、トランスピュータ・ネットワーク上で複数の逐次処理が、お互いに通信しながら実行されていく状況（communicating sequential process）に対応している。occam言語はこのような状況をオペレイティング・システムの機能を借りずに言語レベルで記述できる。一方トランスピュータはoccamで記述されたとうりに並列処理を実行できるようなアーキテクチャに設計されている。（ Oxford大学のT.Hoareが提案したcommunicating sequential processモデルの思想にもとずいて先にoccamが開発され、この言語を実行するためのマイクロ・プロセッサとしてトランスピュータは設計された）。

ここではoccamの最新版であるoccam2が有する並列処理記述機能に重点をおいて、この言語の概要を述べる。

## 1）　基本プロセス

occamはアプリケーション・プログラムをプロセスの集まりとして記述することができる。各プロセスは並列に実行され、チャネルを通じて他のプロセスと通信する．occamプログラムにおいてもっとも簡単なプロセスは一つの処理である。処理とは代入、入力、出力のいずれかである。

代入の構文は

変数：＝式　　x：＝a＋b

と表され、変数に式の値を代入するものであって、一般の言語と同じである。：＝は代入記号である。

入力と出力はプロセス間の通信を記述するものであって，occamに独特な機能である。値は並列プロセスの間でチャネルを介した通信によって渡される。チャネルは二つのプロセス間で、バッファ無しの単一方向通信を提供する。チャネルはソフトウェア上の概念であって、チャネルに使用されるハードウェア資源は、一つのトランスピュータ内では共有メモリであり、トランスピュータ間ではリンクである。

入力はチャネルから値を受け取り、変数にその値を代入する。入力の構文を下記に示

す。？は入力記号である。

チャネル　？　変数　　keyboard　？　char

入力は、入力記号の左側のチャネルから値を受け取り、その値を右側の変数に代入する。右側の例はチャネルkeyboardからデータを受け取り、それを変数charに代入している。入力は値が受け取られるまで待つ。つまり相手プロセスから値が送られてくるまで受け取り側プロセスは待ち合わせをする事になる。このような値の受け渡しと待ち合わせを，UNIX上のＣ言語で記述するとすれば、共有メモリーとセマフォを使用してプログラマが記述しなければならないが，occamでは上記以外の文を書く必要はない。

出力は、式の値をチャネルへ送る。出力の構文を下記に示す。！は出力記号である。

チャネル　！　式　　screen　！　char

右の例は変数char内のデータをチャネルscreenへ出力する。出力も入力と同様に相手がデータを受け取るまで待ち合わせを行う。

２）　並列プロセス

上記の基本プロセスをコンストラクションと呼ばれる予約語によって結合したものは、より大きなプロセスとなる。並列プロセスを記述するための予約語としてPARがあり、並列に実行されるいくつかのプロセスを結合する。三つの名前付きプロセス（手続きprocedureに相当する）を結合したPAR構文を下記に示す。

```
PAR
  calculate(chann.in chann.out)
  observe(chann.in)
  display(chann.out)
```

この並列コンストラクションは三つのプロセスを結合していて、これらは同時に始動し、三つのプロセスがすべて終了したときに終了する。プロセスobserveはプロセスcalculateとチャネルchann.inを使って通信し、プロセスdisplayとプロセスcalculateはチャネルchann.outを使って通信する。各々のチャネルは、二つの並行プロセス間でバッファ無しの単一方向通信を提供する。UNIXのようにプロセスの生成、実行開始、終了、通信手段をプログラマが記述する必要はない。

３）　オルタネーション

オルタネーションは入力によってガードされた、いくつかのプロセスを結合する。オルタネーションは入力がレディーとなったガードに関連したプロセスを実行する。次の例について考える。

```
ALT
```

```
left ? packet
  left.process
right ? packet
  right.process
```

この例ではチャネルleftまたはrightから入力を受け取る。二つの入力のうちチャネルから先にデータを受け取ってレディとなった入力が選ばれ、それに関連したプロセスが実行される。もし、どちらのチャネルもレディーでなければ、オルタネーションはいずれかの入力がレディーになるまで待つ。もし、両方の入力がレディーならば、いずれか一方の入力とそれに関連したプロセスが実行される。occamでは割り込み入力ラインもチャネルとして扱うので、上の例でleftを割り込み入力チャネルに割り当てれば、割り込みが発生したときleft.processが実行される。

オルタネーションにおける複数入力のうちから特定の入力を一時的に除去するために、下記の例のように論理式を含ませることができる。

```
ALT
  left.enabled & left ? packet
    left.process
  right ? packet
    right.process
```

この例では入力leftの前に論理変数left.enabledが置かれている。もしleft.enabledが真ならば、入力leftはオルタネーションの考慮に含まれる。もしleft.enabledが偽ならば、入力leftは除かれる。

4）　チャネルとプロトコル

チャネルは二つの並列プロセス間でバッファ無しの単一方向ポイント・ポイント間通信を提供する。ポイント・ポイント間通信とは二つのプロセスの間だけで通信が可能という意味である。たとえばUNIXではプロセスにid番号が付けられていて，idを指定すれば任意のプロセスと通信できるが，occamのチャネルでは通信の相手は一つのプロセスに限られる。チャネルを介して渡されるデータの型と構造は、チャネル・プロトコルによって指定される。チャネルの名前とプロトコルはチャネル宣言によって指定される。

単一方向通信路であるチャネルにはシリアルにデータが転送されるので、どんなデータが、どんな順序で送られるかを宣言しておかなければならない。チャネルは、変数が宣言されるのと同じように宣言される。下記にもっとも簡単なデータ型のチャネル宣言を示す。

```
CHAN OF BYTE screen:
```

この例は，BYTE型のプロトコルを持つscreenと名付けられたチャネルを宣言している。つまり、チャネルscreenを介しての通信がBYTE型のデータでなければならないことを、宣言している。チャネルscreenを介しての出力は

```
screen ! 'H'
```

のように記述され、文字'H'をBYTE型チャネルscreenへ出力するという意味を表す。

プロトコルには、上記の単純プロトコルの他に、より複雑なデータ型が宣言できる。名前付きプロトコル、逐次プロトコル、混成プロトコル、アナーキック・プロトコルがある。本事例では混成プロトコルを使用している。

プロトコルとチャネルをUNIXとC言語に対比すれば、プロトコルは送受信する構造型データのC言語による宣言、チャネルは共有メモリへのデータ転送にそれぞれ対応していると言える。

5） タイマ

タイマは、いくつかの並列プロセスによって参照可能な時計を提供する。タイマはチャネルやデータ型のように基本型であり、チャネルや変数と同様の方法で宣言される。下記にタイマの宣言を示す。

```
TIMER clock:
```

この宣言はclockという名前で識別されるタイマを宣言する。タイマから時刻を読みとる構文を下記に示す。

```
clock ? time
```

この例は、時刻の値をタイマclockから入力し、整数型変数timeにその値を代入する。

タイマを用いた遅延も下記のように簡単に記述できる。

```
clock ? AFTER delay
```

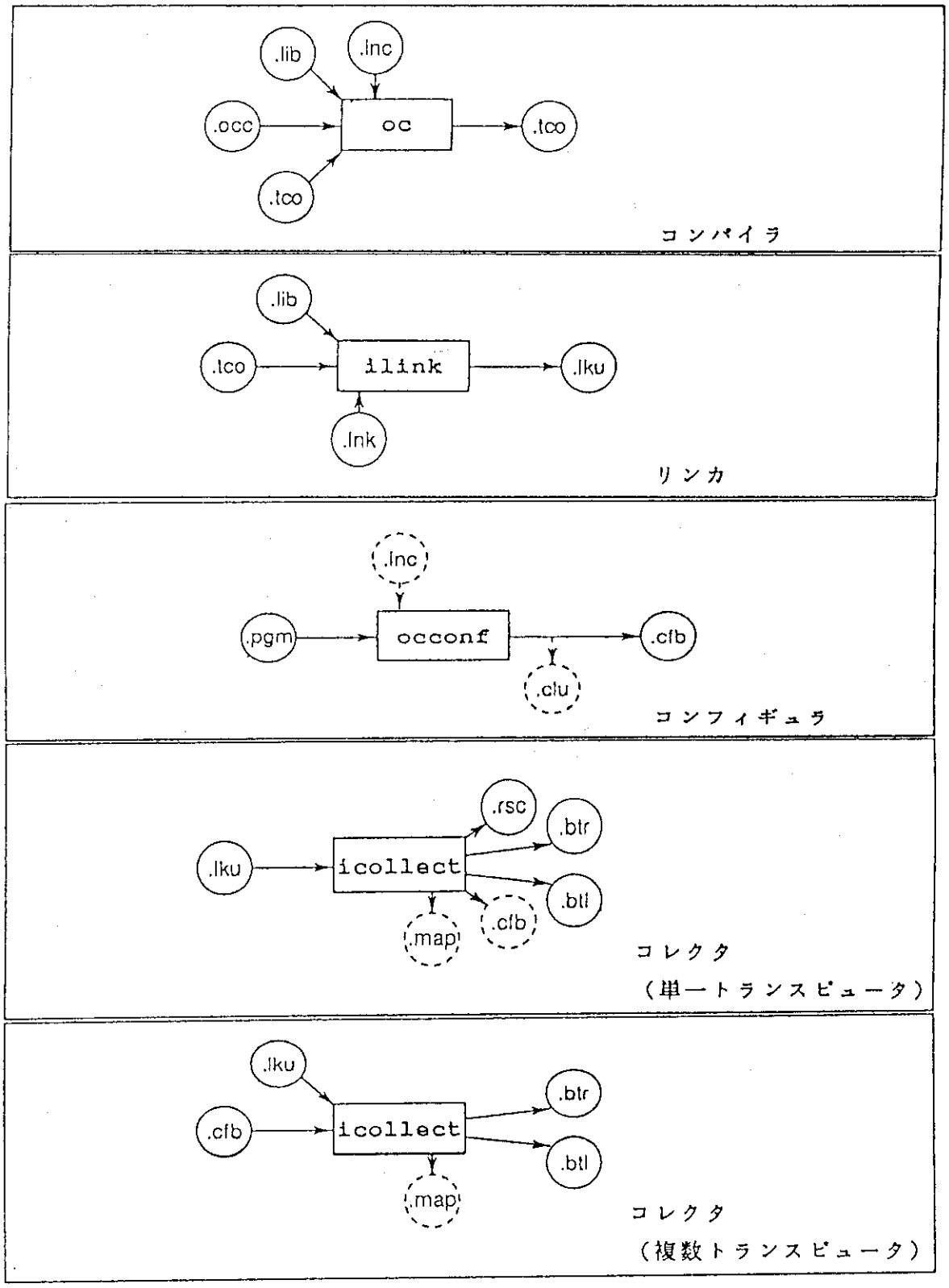
この入力はタイマclockの値がdelayの値を越えるまで待つ。つまりtimeがタイマの時刻であるならば、この入力は（time AFTER delay）が真になるまで待つ。delayの値は一定で変化しない。したがって現時点からdelayだけの遅延をつくるには下記のように記述すればよい。

```
SEQ
  clock ? now
  clock ? AFTER now PLUS delay
```

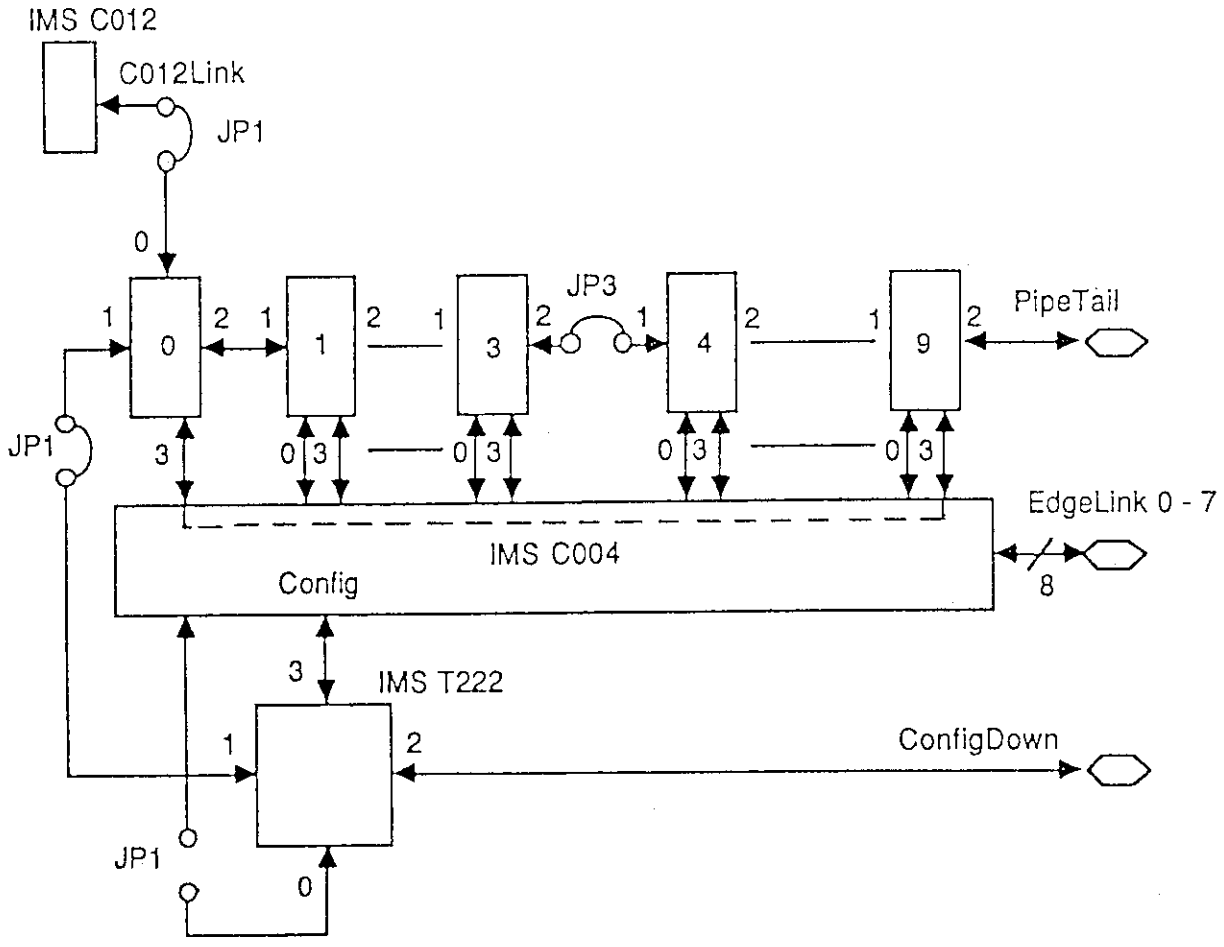
この例では、まず現在時刻を表す値を入力し、それを変数nowに代入する。次の遅延入力はclockからの入力の値がnow PLUS delayの値を越えるまで待つ。PLUSはモジュロ演算の加算である。

occamのタイマには1マイクロ秒単位のタイマと、64マイクロ秒単位のタイマがある。本事例では1マイクロ秒単位のタイマを使用した。UNIXでは、C言語で呼び出せるtimeというシステム・コールがあって現在時刻を知ることができるが、このタイマの時間分解能は1秒である。

以上，occam固有の機能をUNIXの類似した機能と対比しながら述べた。これらUNIXの機能は、C言語で呼び出せるシステム・コールとして提供されているのであって、C言語の機能ではない。occamではこれらの機能を言語レベルで記述できるのはトランスピュータのハードウェアがタスク・スケジューラ、タイマ、送受信デバイスを内蔵しているからである。




図A 3. 1　occam 2 ツールセットのファイル・エクステンション




図A 4. 1　マザーボードIMS B008の接続図

```
SOFTWIRE
 PIPE 0
   --SLOT 5,LINK 3 TO SLOT 0,LINK 3
   SLOT 9,LINK 3 TO SLOT 0,LINK 3
END
```

図A 4. 2　ソフトワイヤ仕様ファイル b008soft

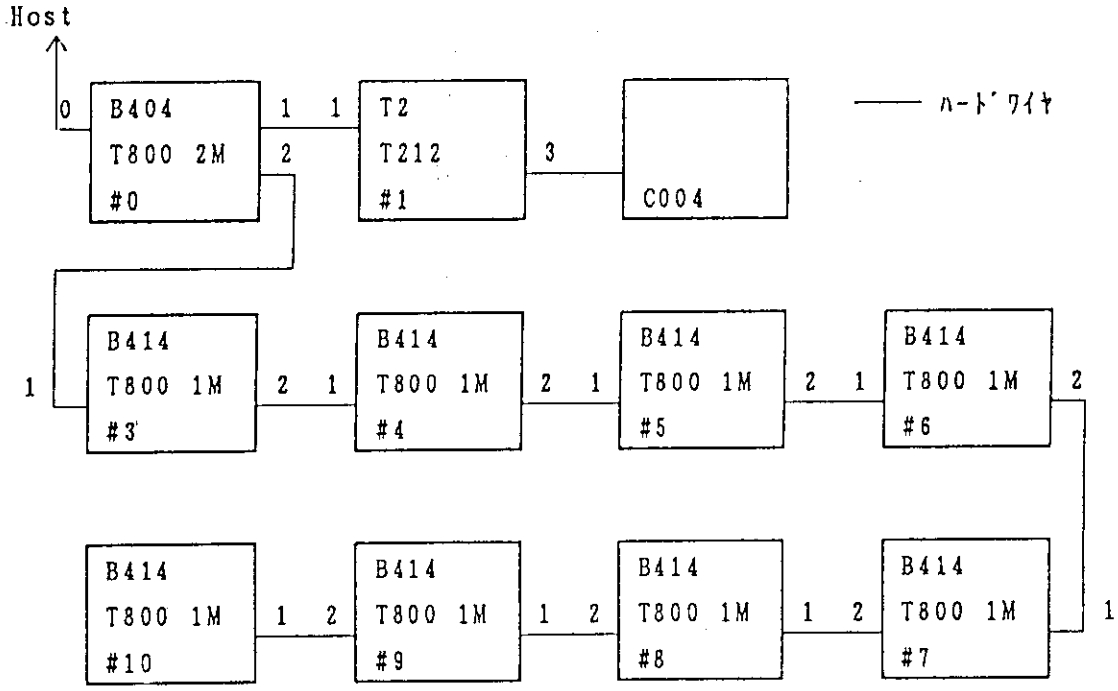
```
-- B008 harwire description
DEF B008
 SIZES
   T2 1
   C4 1
   SLOT 10
   EDGE 10
 END
 T2CHAIN
   T2 0, LINK 3 C4 0
 END
 HARDWIRE
   SLOT 0,LINK 2 TO SLOT 1,LINK 1
   SLOT 1,LINK 2 TO SLOT 2,LINK 1
   SLOT 2,LINK 2 TO SLOT 3,LINK 1
   SLOT 3,LINK 2 TO SLOT 4,LINK 1
   SLOT 4,LINK 2 TO SLOT 5,LINK 1
   SLOT 5,LINK 2 TO SLOT 6,LINK 1
   SLOT 6,LINK 2 TO SLOT 7,LINK 1
   SLOT 7,LINK 2 TO SLOT 8,LINK 1
   SLOT 8,LINK 2 TO SLOT 9,LINK 1
   C4 0,LINK 10 TO SLOT 0,LINK 3
   C4 0,LINK 1 TO SLOT 1,LINK 0
   C4 0,LINK 11 TO SLOT 1,LINK 3
   C4 0,LINK 2 TO SLOT 2,LINK 0
   C4 0,LINK 12 TO SLOT 2,LINK 3
   C4 0,LINK 3 TO SLOT 3,LINK 0
   C4 0,LINK 13 TO SLOT 3,LINK 3
   C4 0,LINK 4 TO SLOT 4,LINK 0
   C4 0,LINK 14 TO SLOT 4,LINK 3
   C4 0,LINK 5 TO SLOT 5,LINK 0
   C4 0,LINK 15 TO SLOT 5,LINK 3
   C4 0,LINK 6 TO SLOT 6,LINK 0
   C4 0,LINK 16 TO SLOT 6,LINK 3
   C4 0,LINK 7 TO SLOT 7,LINK 0
   C4 0,LINK 17 TO SLOT 7,LINK 3
   C4 0,LINK 8 TO SLOT 8,LINK 0
   C4 0,LINK 18 TO SLOT 8,LINK 3
   C4 0,LINK 9 TO SLOT 9,LINK 0
   C4 0,LINK 19 TO SLOT 9,LINK 3
   C4 0,LINK 20 TO EDGE 0
   C4 0,LINK 21 TO EDGE 1
   C4 0,LINK 22 TO EDGE 2
   C4 0,LINK 23 TO EDGE 3
   C4 0,LINK 24 TO EDGE 4
   C4 0,LINK 25 TO EDGE 5
   C4 0,LINK 26 TO EDGE 6
   C4 0,LINK 27 TO EDGE 7
   --C4 0,LINK 28 TO EDGE 8
   --C4 0,LINK 29 TO EDGE 9
 END
PIPE B008 END
```

図A 4. 3　ハードワイヤ仕様ファイル b008hard

```
INMOS Module Motherboard Configuring System
Toolset Version 3.1
OPTIONS :

(h)elp                      (r)eset subsystem
(s)et C004s                 (m)anual entry
(q)uit                      (d)iagnostics
(n)etwork mapper            (c)heck L1 file
(v)iew source file          (i)nitialise C004s
(l)ink numbers              (b)ootable file
(o)ccam table

Key command :
```

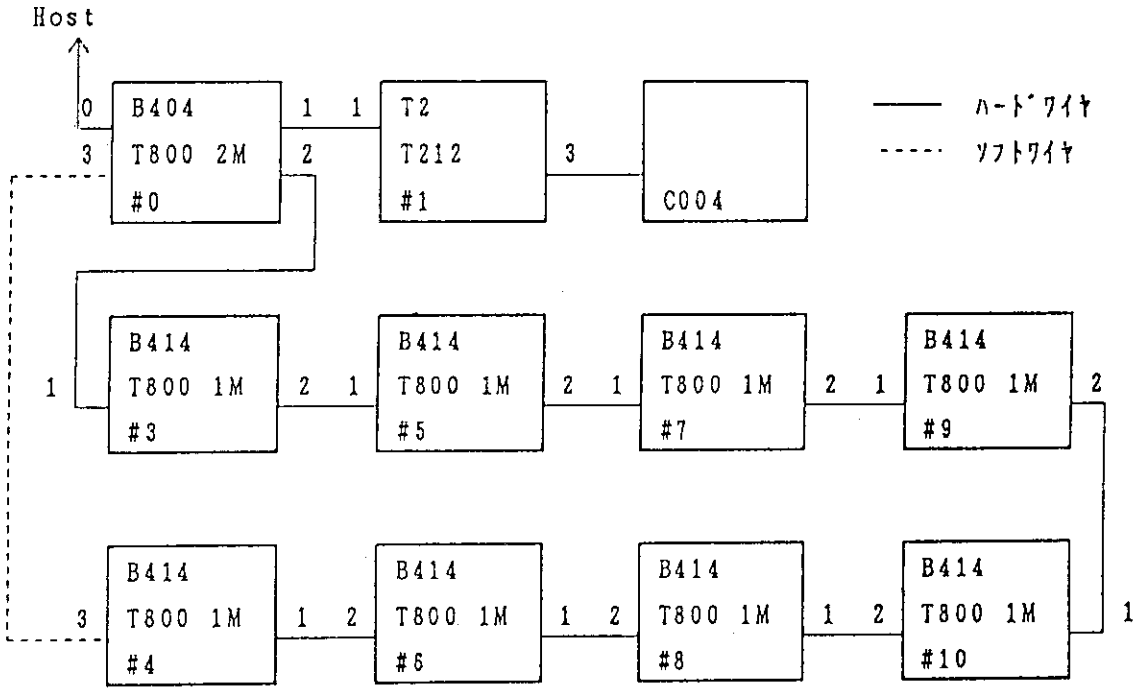
図A 4. 4　mms2. b4 のメニュー画面

```
C:¥D7205¥MULTI>ispy4s
Using #150 ispy 2.31
   # Part rate Mb Bt [  Link0  Link1  Link2  Link3 ]
   0 T800d-25 0.19 0 [   HOST    1:1    3:1    ...  ]
   1 T2    -20 1.75 1 [    ...    0:1    ...   C004 ]
   3 T800d-25 1.75 1 [    ...    0:2    4:1    ...  ]
   4 T800d-25 1.75 1 [    ...    3:2    5:1    ...  ]
   5 T800d-25 1.77 1 [    ...    4:2    6:1    ...  ]
   6 T800d-25 1.75 1 [    ...    5:2    7:1    ...  ]
   7 T800d-25 1.75 1 [    ...    6:2    8:1    ...  ]
   8 T800d-25 1.77 1 [    ...    7:2    9:1    ...  ]
   9 T800d-25 1.75 1 [    ...    8:2   10:1    ...  ]
  10 T800d-25 1.75 1 [    ...    9:2    ...    ...  ]

C:¥D7205¥MULTI>
```

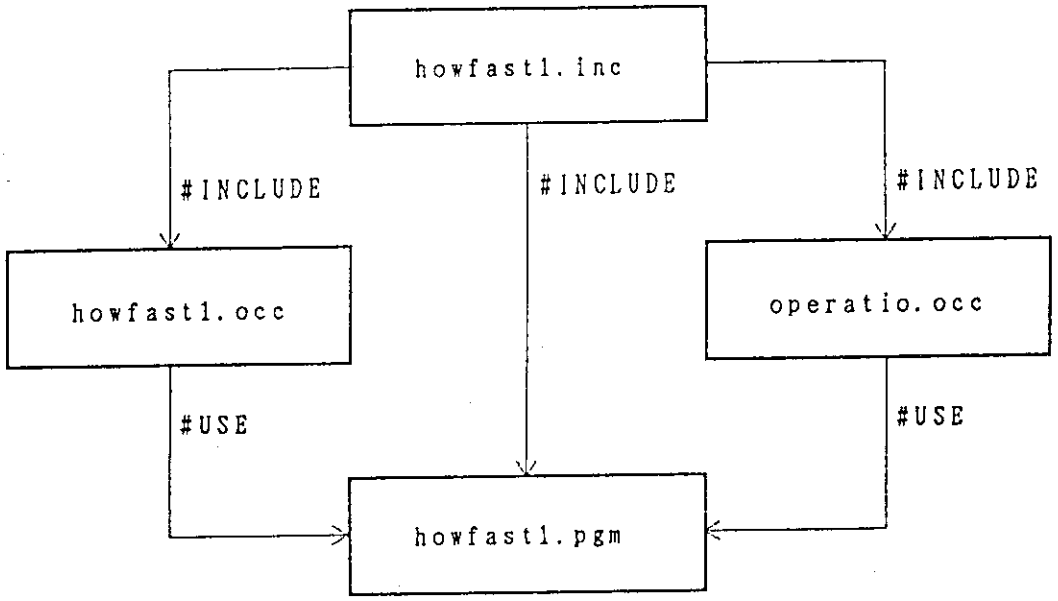



図A 4. 5　ispy4sが出力するリンク接続表とネットワーク
（電源投入直後）

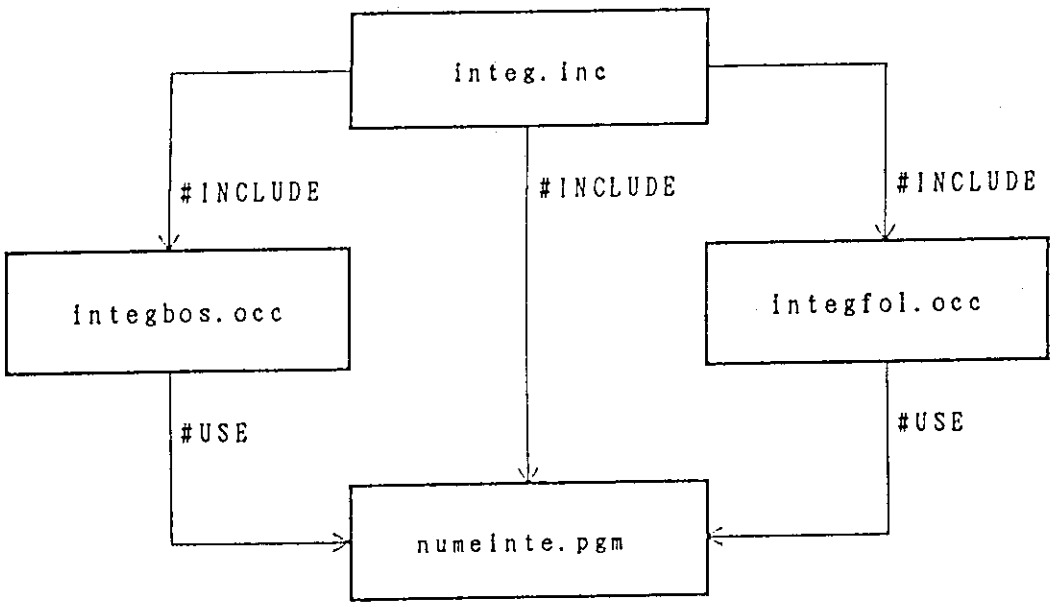
```
C:¥D7205¥MULTI>ispy4s
Using #150 ispy 2.3j
   # Part rate Mb Bt [  Link0  Link1  Link2  Link3 ]
   0 T800d-25 0.19 0 [   HOST    1:1    3:1    4:3 ]
   1 T2   -20 1.74 1 [    ...    0:1    ...   C004 ]
   3 T800d-25 1.75 1 [    ...    0:2    5:1    ... ]
   4 T800d-25 1.33 3 [    ...    6:2    ...    0:3 ]
   5 T800d-25 1.77 1 [    ...    3:2    7:1    ... ]
   6 T800d-25 1.75 2 [    ...    8:2    4:1    ... ]
   7 T800d-25 1.77 1 [    ...    5:2    9:1    ... ]
   8 T800d-25 1.75 2 [    ...   10:2    6:1    ... ]
   9 T800d-25 1.77 1 [    ...    7:2   10:1    ... ]
  10 T800d-25 1.75 2 [    ...    9:2    8:1    ... ]

C:¥D7205¥MULTI>
```




図A 4. 6　ispy4sが出力するリンク接続表とネットワーク
（mms2. b4 によるソフトワイヤ接続後）

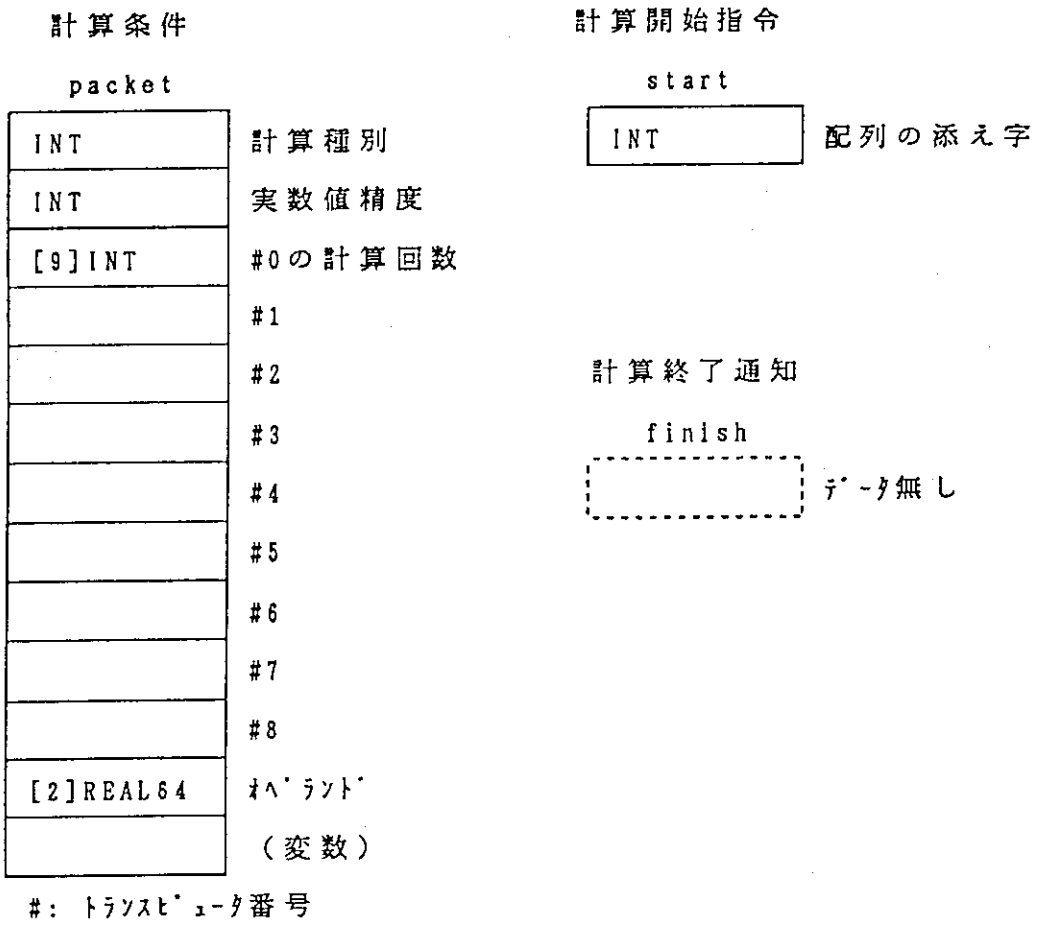



図A 5. 1　四則演算と算術関数：プログラムのファイル構成




図A 5. 6　定積分計算：プログラムのファイル構成

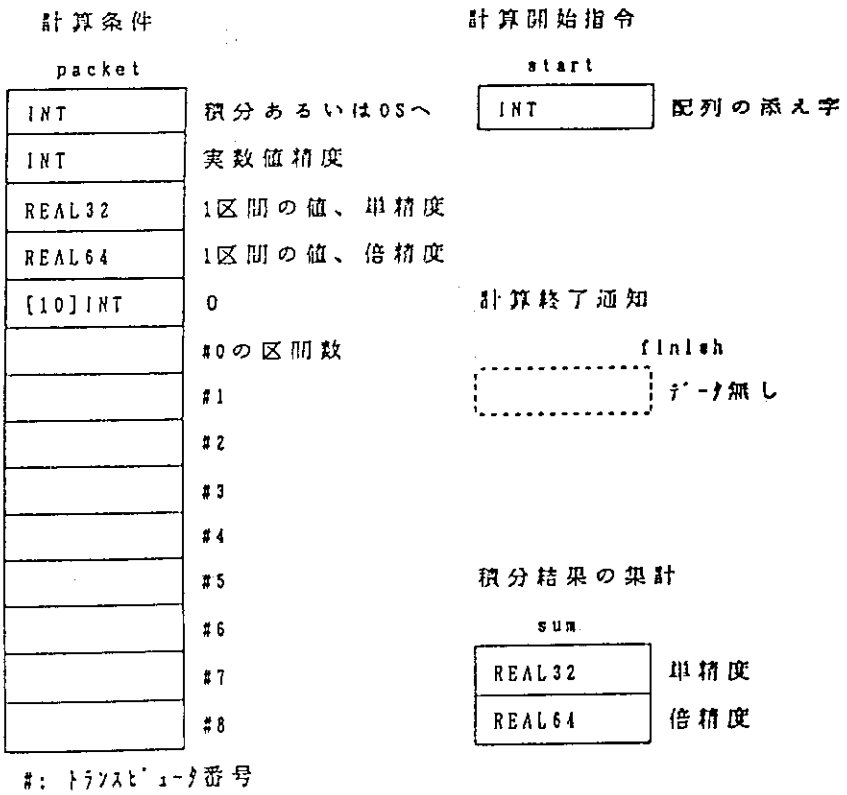
```
VAL num.of.transp IS 9:
--}}}
PROTOCOL PARAM
  CASE
    packet; INT;INT;[num.of.transp]INT;[2]REAL64
    start; INT
    finish
:
```



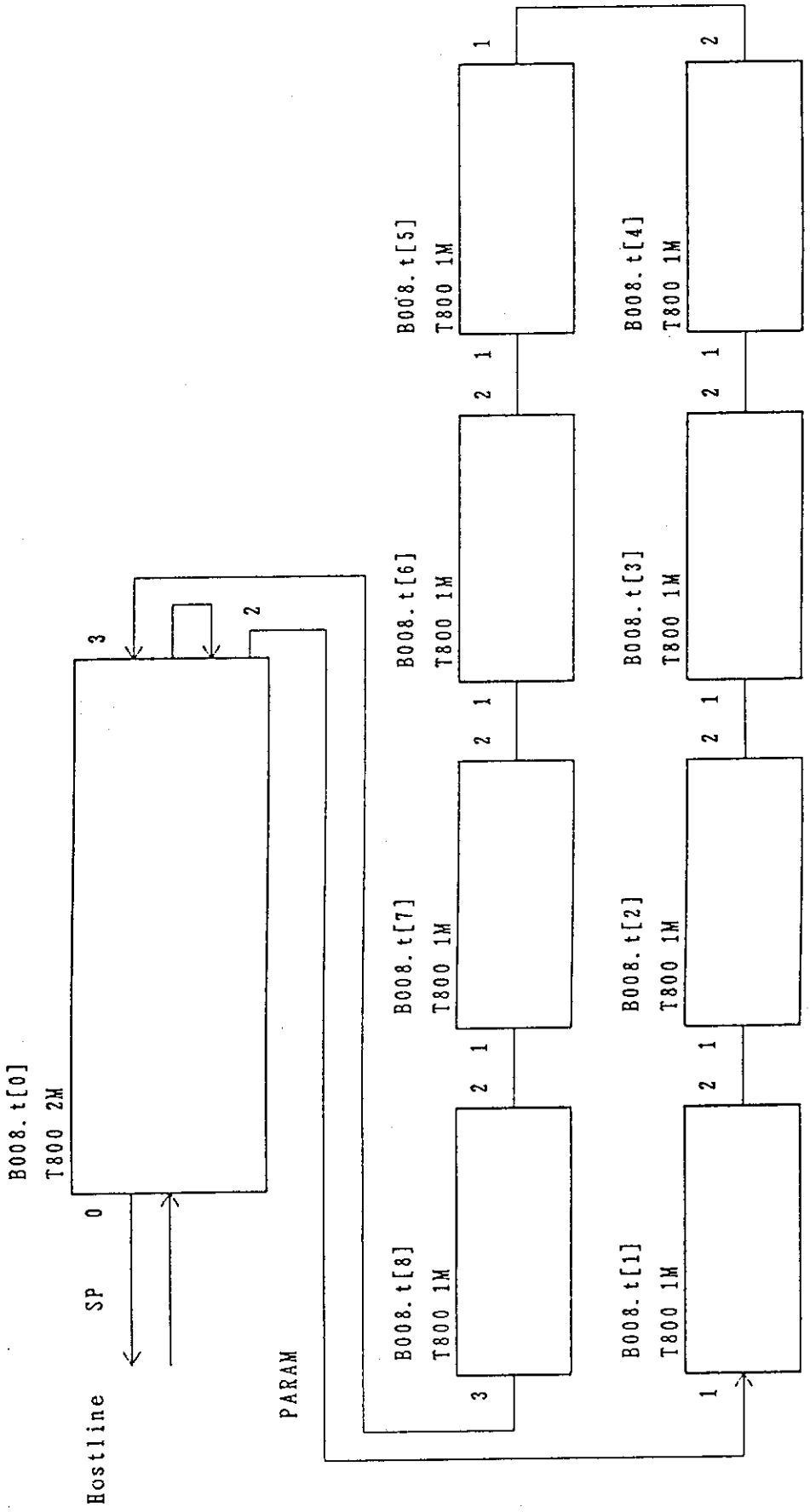

図A 5. 2　混成プロトコルPARAMのデータ構造

```
How fast are the following operations performed on the transputers?

                1)  +
                2)  -
                3)  *
                4)  /
                5)  sqrt
                6)  sin
                7)  cos
                8)  tan
                9)  exp
               10)  log
               11)  quit
Select the number of the above arithmetic operators and the functio
--: 1          ## addition
1)single or 2)double--: 1
augend--:1.0
addend--:2.0
number of additions--:100000
number of transputers--:9
time1=253041225
time2=253054243
time.ope=13018
32-bit sum = 3.0
time for a 32-bit addition = 0.130180001
press any key :
```

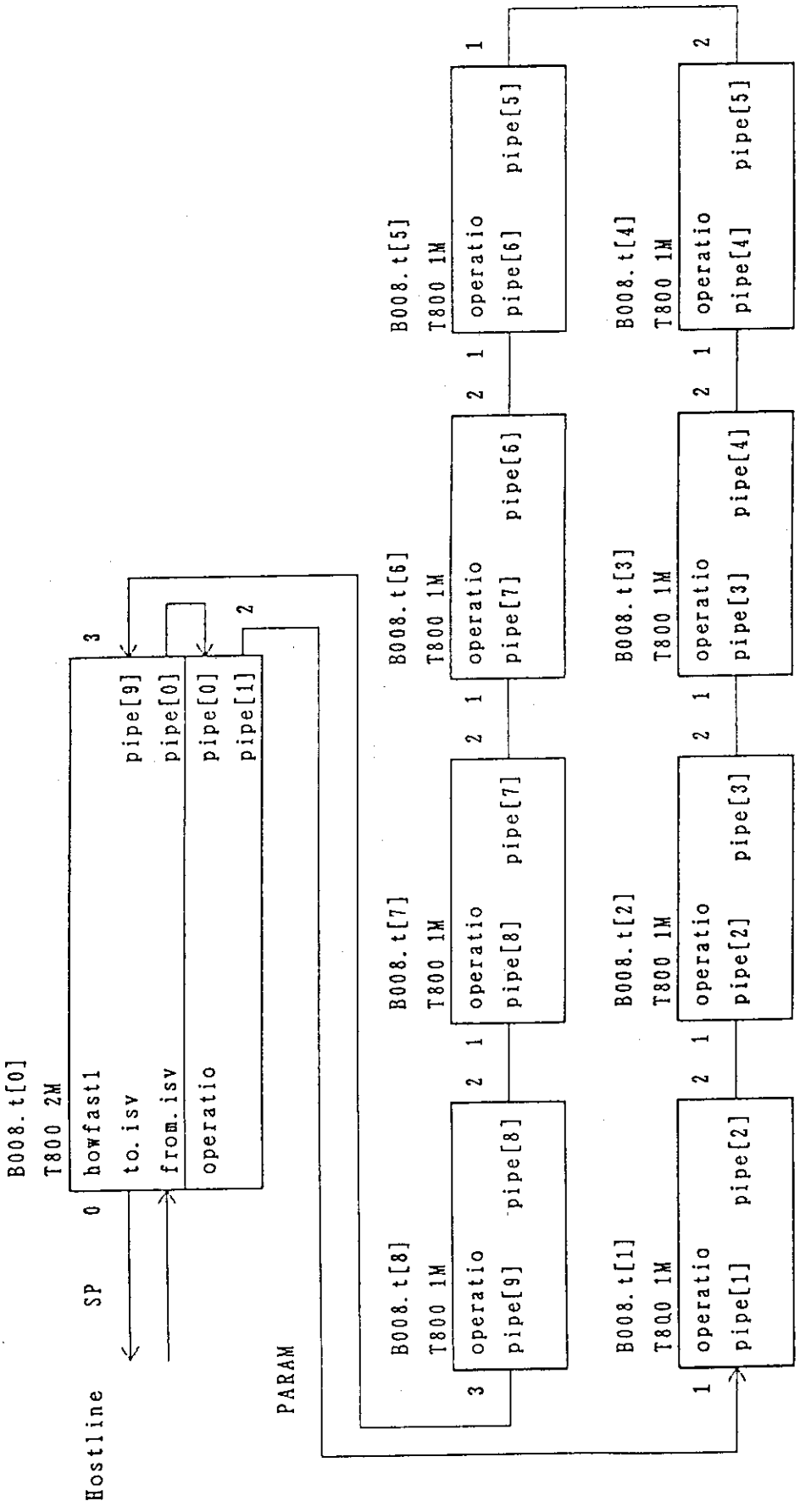
図A 5. 3　howfast1.occのメニュー画面
（四則演算と算術関数）




図A 5. 7　混成プロトコルINTEGのデータ構造




図A 5. 4　howfast1.pgmのハードウェア記述部




図A 5. 5　howfast1.pgmのハードウェア記述部とソフトウェア記述部

```
>>> howfast1.inc

{{{  howfast1.inc
{{{  number of transputers
-- five transputers are installed , one for debug
-- VAL  num.of.transp  IS  4:
-- five transputers are installed, all for work
-- VAL num.of.transp IS 5:
-- nine transputers are installed, all for work
VAL num.of.transp IS 9:
}}}
PROTOCOL PARAM
  CASE
    packet; INT;INT;[num.of.transp]INT;[2]REAL64
    start; INT
    finish
:

}}}  howfast1.inc

>>> howfast1.occ レベル 1

{{{  howfast1.occ
...  declarations
...  body of howfast1
:
}}}  howfast1.occ


>>> howfast1.occ レベル 2-a

{{{  howfast1.occ
{{{  declarations
#INCLUDE "howfast1.inc"
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#USE     "hostio.lib"
#USE     "streamio.lib"
#USE     "snglmath.lib"
#USE     "dblmath.lib"
}}}
...  body of howfast1
:
}}}  howfast1.occ

>>> howfast1.occ レベル 2-b

{{{  howfast1.occ
...  declarations
{{{  body of howfast1
PROC howfast1(CHAN OF SP from.isv, to.isv, CHAN OF PARAM in, out)
  BOOL loop, ierr:
  BYTE any, bres:
  INT key:
  ...  proc addition
  ...  proc subtraction
  ...  proc multiplication
  ...  proc division
  ...  proc squroot
  ...  proc sine
  ...  proc cosine
  ...  proc tangent
  ...  proc exponential
  ...  proc logarithm
  SEQ
    loop:=TRUE
    WHILE loop
```

```
    SEQ
      ...  clear the screen and display the menue
      ...  select a number, clear the screen, branch to the selected operat
  so.exit(from.isv, to.isv, sps.success)
```

>>> howfast1.occ レベル 3-a

```
      {{{  clear the screen and display the menue
      CHAN OF SS scrn:
      PAR
        so.scrstream.to.ANSI(from.isv, to.isv, scrn)
        SEQ
          ss.goto.xy(scrn, 0, 0)
          ss.clear.eos(scrn)
          ss.goto.xy(scrn, 10, 5)
          ss.write.string(scrn, "How fast are the following operations perf
          ss.write.nl(scrn)
          ss.goto.xy(scrn, 20, 7)
          ss.write.string(scrn, "1)  +" )
          ss.goto.xy(scrn, 20, 8)
          ss.write.string(scrn, "2)  -" )
          ss.goto.xy(scrn, 20, 9)
          ss.write.string(scrn, "3)  **" )
          ss.goto.xy(scrn, 20, 10)
          ss.write.string(scrn, "4)  /" )
          ss.goto.xy(scrn, 20, 11)
          ss.write.string(scrn, "5)  sqrt" )
          ss.goto.xy(scrn, 20, 12)
          ss.write.string(scrn, "6)  sin" )
          ss.goto.xy(scrn, 20, 13)
          ss.write.string(scrn, "7)  cos" )
          ss.goto.xy(scrn, 20, 14)
          ss.write.string(scrn, "8)  tan" )
          ss.goto.xy(scrn, 20, 15)
          ss.write.string(scrn, "9)  exp" )
          ss.goto.xy(scrn, 19, 16)
          ss.write.string(scrn, "10)  log" )
          ss.goto.xy(scrn, 19, 17)
          ss.write.string(scrn, "11)  quit" )
          ss.goto.xy(scrn, 10, 18)
          ss.write.string(scrn, "Select the number of the above arithmetic
          ss.write.endstream(scrn)
      }}}
      ...  select a number, clear the screen, branch to the selected operat
```

>>> howfast1.occ レベル 3-b

```
      ...  clear the screen and display the menue
      {{{  select a number, clear the screen, branch to the selected operat
      so.read.echo.int (from.isv, to.isv, key, ierr )
      CASE key
        1 -- addition
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## addition")
            addition(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        2 -- subtraction
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## subtraction"
            subtraction(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
```

```
        3 -- multiplication
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## multiplicati
            multiplication(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        4 -- division
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## division")
            division(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        5 -- square root
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## square root"
            squroot(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        6 -- sine
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## sine")
            sine(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        7 -- cosine
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## cosine")
            cosine(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        8 -- tangent
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## tangent")
            tangent(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        9 -- exponential
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## exponential"
            exponential(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        10 -- logarithm
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## logarithm")
            logarithm(from.isv, to.isv, in, out)
            so.write.string(from.isv, to.isv, "press any key :")
            so.getkey(from.isv, to.isv, any, bres)
        11 -- quit
          SEQ
            so.write.string.nl(from.isv, to.isv, "          ## quit")
            loop:=FALSE
        ELSE
          SEQ
            so.write.string.nl(from.isv, to.isv, "          Bad number. Try
            SKIP
            so.getkey(from.isv, to.isv, any, bres)
      }}}  end of branches
    so.exit(from.isv, to.isv, sps.success)
}}}
:
}}}  howfast1.occ

>>> howfast1.occ レベル 3-c

  {{{  proc addition
  PROC addition(CHAN OF SP from.isv, to.isv, CHAN OF PARAM in, out)
    TIMER clock:
    INT time1, time2, add.times, num.cpu, time.ope:
```

```
REAL32 augend32,addend32, time.add, sum32:
REAL64 augend64, addend64, sum64:
[num.of.transp]INT repeat.num.t, repeat.num.r:
[2]REAL64 operand, operand.r:
INT opes.per.transp, remain, ope.key, ope.key.r, cpu.no:
INT sng.dbl.key, sng.dbl.key.r:

SEQ
  ...  input the parameters
  PRI PAR
    SEQ
      ...  arrange the parameters
      ...  start an operation , get a start time and a stop time
    SKIP
  ...  display the time required for the selected operation
  CASE sng.dbl.key
    1 -- display the time required for a 32 bit operation
      ...  32 bit
    2 -- display the time required for a 64 bit operation
      ...  64 bit
```

>>> howfast1.occ レベル 4-a

```
INT opes.per.transp, remain, ope.key, ope.key.r, cpu.no:
INT sng.dbl.key, sng.dbl.key.r:

SEQ
  {{{  input the parameters
  so.write.string(from.isv, to.isv,"          1)single or 2)double--: " )
  so.read.echo.int(from.isv, to.isv, sng.dbl.key, ierr)
  so.write.nl(from.isv, to.isv)
  so.write.string(from.isv, to.isv, "          augend--:" )
  so.read.echo.real64(from.isv, to.isv, augend64, ierr)
  so.write.nl(from.isv, to.isv)
  so.write.string(from.isv, to.isv, "          addend--:")
  so.read.echo.real64(from.isv, to.isv, addend64, ierr)
  so.write.nl(from.isv, to.isv)
  so.write.string(from.isv, to.isv, "          number of additions--:")
  so.read.echo.int(from.isv, to.isv, add.times, ierr)
  so.write.nl(from.isv, to.isv)
  so.write.string(from.isv, to.isv, "          number of transputers--:")
  so.read.echo.int(from.isv, to.isv, num.cpu, ierr)
  so.write.nl(from.isv, to.isv)
  }}}
  PRI PAR
    SEQ
      ...  arrange the parameters
```

>>> howfast1.occ レベル 4-b

```
INT opes.per.transp, remain, ope.key, ope.key.r, cpu.no:
INT sng.dbl.key, sng.dbl.key.r:

SEQ
  ...  input the parameters
  PRI PAR
    SEQ
      {{{  arrange the parameters
      opes.per.transp := add.times / num.cpu
      remain := add.times Y num.cpu
      repeat.num.t[0] := opes.per.transp + remain
      SEQ i=1 FOR num.of.transp - 1
        repeat.num.t[i] := 0(INT)
      SEQ i=1 FOR num.cpu - 1
        repeat.num.t[i] := opes.per.transp
      operand[0] := augend64
      operand[1] := addend64
      ope.key := 1
```

```
        }}}
        ...  start an operation , get a start time and a stop time
      SKIP
    ...  display the time required for the selected operation
    CASE sng.dbl.key
      1 -- display the time required for a 32 bit operation
```

>>> howfast1.occ レベル 4-c

```
  INT opes.per.transp, remain, ope.key, ope.key.r, cpu.no:
  INT sng.dbl.key, sng.dbl.key.r:

  SEQ
    ...  input the parameters
    PRI PAR
      SEQ
        ...  arrange the parameters
        {{{  start an operation , get a start time and a stop time
        out ! packet; ope.key;sng.dbl.key;repeat.num.t;operand
        in ? CASE
          packet; ope.key.r;sng.dbl.key.r;repeat.num.r;operand.r
            clock ? time1
        out ! start; 0(INT)
        in ? CASE
          start; cpu.no
            out ! finish
        in ? CASE
          finish
            clock ? time2
        }}}
      SKIP
    ...  display the time required for the selected operation
    CASE sng.dbl.key
```

>>> howfast1.occ レベル 4-d

```
    PRI PAR
      SEQ
        ...  arrange the parameters
        ...  start an operation , get a start time and a stop time
      SKIP
    {{{  display the time required for the selected operation
    time.ope := time2 MINUS time1
    time.add:=(REAL32 ROUND (time.ope))/(REAL32 ROUND add.times)
    so.write.string(from.isv, to.isv, "          time1=")
    so.write.int(from.isv, to.isv, time1, 0)
    so.write.nl(from.isv, to.isv)
    so.write.string(from.isv, to.isv, "          time2=")
    so.write.int(from.isv, to.isv, time2, 0)
    so.write.nl(from.isv, to.isv)
    so.write.string(from.isv, to.isv, "          time.ope=")
    so.write.int(from.isv, to.isv, time.ope, 0)
    so.write.nl(from.isv, to.isv)
    }}}
    CASE sng.dbl.key
      1 -- display the time required for a 32 bit operation
        ...  32 bit
      2 -- display the time required for a 64 bit operation
        ...  64 bit
  :
```

>>> howfast1.occ レベル 4-e

```
    PRI PAR
      SEQ
        ...  arrange the parameters
        ...  start an operation , get a start time and a stop time
      SKIP
```

```
      ...  display the time required for the selected operation
      CASE sng.dbl.key
        1 -- display the time required for a 32 bit operation
          {{{  32 bit
          SEQ
            augend32:=( REAL32 ROUND augend64)
            addend32:=( REAL32 ROUND addend64)
            sum32 := augend32 +  addend32
            so.write.string(from.isv, to.isv, "          32-bit sum = ")
            so.write.real32(from.isv, to.isv, sum32, 0, 0)
            so.write.nl(from.isv, to.isv)
            so.write.string(from.isv, to.isv, "          time for a 32-bit ad
            so.write.real32(from.isv, to.isv, time.add, 0, 0)
            so.write.nl(from.isv, to.isv)
          }}}
        2 -- display the time required for a 64 bit operation
          ...  64 bit
  :
  }}}
```

>>> howfast1.occ レベル 4-f

```
    SEQ
      ...  input the parameters
      PRI PAR
        SEQ
          ...  arrange the parameters
          ...  start an operation , get a start time and a stop time
        SKIP
      ...  display the time required for the selected operation
      CASE sng.dbl.key
        1 -- display the time required for a 32 bit operation
          ...  32 bit
        2 -- display the time required for a 64 bit operation
          {{{  64 bit
          SEQ
            sum64 := augend64 + addend64
            so.write.string(from.isv, to.isv, "          64-bit sum = ")
            so.write.real64(from.isv, to.isv, sum64, 0,0)
            so.write.nl(from.isv, to.isv)
            so.write.string(from.isv, to.isv, "          time for a 64-bit ad
            so.write.real32(from.isv, to.isv, time.add, 0, 0)
            so.write.nl(from.isv, to.isv)
          }}}
  :
  }}}
```

>>> operatio.occ レベル 1

```
{{{  operatio.occ
#INCLUDE "howfast1.inc"
#USE     "sngimath.lib"
#USE     "dblmath.lib"
PROC operatio(CHAN OF PARAM  input, output)
  INT ope.key, sng.dbl.key, cpu.no:
  [2]REAL64 operand:
  [num.of.transp]INT repeat.num:
  BOOL going:
  REAL32 result32, operand1.s, operand2.s:
  REAL64 result64:

  SEQ
    going := TRUE
    WHILE going
      SEQ
        ...  input and output parameters
        ...  get 'cpu.no', send 'cpu.no'
        ...  operations
```

```
:

}}}  operatio.occ
```

>>> operatio.occ レベル 2-a

```
                                Editing, Language = Occam
  [2]REAL64 operand:
  [num.of.transp]INT repeat.num:
  BOOL going:
  REAL32 result32, operand1.s, operand2.s:
  REAL64 result64:

  SEQ
    going := TRUE
    WHILE going
      SEQ
        {{{  input and output parameters
        input ? CASE
          packet; ope.key;sng.dbl.key;repeat.num;operand
            output ! packet; ope.key;sng.dbl.key;repeat.num;operand
        }}}
        {{{  get 'cpu.no', send 'cpu.no'
        input ? CASE
          start; cpu.no
            output ! start; (cpu.no + 1(INT))
        }}}
        ...  operations
:

}}}  operatio.occ
```

>>> operatio.occ レベル 2-b

```
PROC operatio(CHAN OF PARAM  input, output)
  INT ope.key, sng.dbl.key, cpu.no:
  [2]REAL64 operand:
  [num.of.transp]INT repeat.num:
  BOOL going:
  REAL32 result32, operand1.s, operand2.s:
  REAL64 result64:

  SEQ
    going := TRUE
    WHILE going
      SEQ
        ...  input and output parameters
        ...  get 'cpu.no', send 'cpu.no'
        {{{  operations
        CASE ope.key
          1 -- addition
            ...  addition
          2 -- subtraction
            ...  subtraction
          3 -- multiplication
            ...  multiplication
          4 -- division
            ...  division
          5 -- square root
            ...  square root
          6 -- sine
            ...  sine
          7 -- cosine
            ...  cosine
          8 -- tangent
            ...  tangent
          9 -- exponential
            ...  exponential
```

```
          10 -- logarithm
            ...  logarithm
          11 -- quit
            going := FALSE
          ELSE
            SKIP
         -- end of branch

        }}}
:

}}}  operatio.occ
```

>>> operatio.occ レベル 3-a

```
        ...  get 'cpu.no', send 'cpu.no'
        {{{  operations
        CASE opc.key
          1 -- addition
            {{{  addition
            SEQ
              CASE sng.dbl.key
                1 -- 32-bit operation
                  SEQ
                    operand1.s := ( REAL32 ROUND operand[0] )
                    operand2.s := ( REAL32 ROUND  operand[1] )
                    SEQ i=0 FOR repeat.num[cpu.no]
                      result32 := operand1.s + operand2.s
                2 -- 64-bit operation
                  SEQ
                    SEQ  i=0 FOR repeat.num[cpu.no]
                      result64 := operand[0] + operand[1]
                ELSE
                  SKIP
              input ? CASE
                finish
                  output ! finish
            }}}
          2 -- subtraction
```

>>> howfast1.pgm レベル 1

```
{{{  howfast1.pgm
#INCLUDE "howfast1.inc"

...  hardware description

-- Code
#INCLUDE "hostio.inc"
#INCLUDE "howfast1.inc"
#USE "howfast1.c8h"
#USE "operatio.c8h"

...  software description
:

}}}  howfast1.pgm
```

>>> howfast1.pgm レベル 2-a

```
{{{  howfast1.pgm
#INCLUDE "howfast1.inc"

{{{  hardware description
--  network on B008 INMOS board
--  kilo, mega
VAL  K  IS 1024:
```

```
VAL M IS K*K:
[num.of.transp]NODE B008.t:
ARC Hostlink:
NETWORK
  DO
    CONNECT B008.t[0][link][0] TO HOST WITH Hostlink
    SET B008.t[0] (type, memsize := "T800", 2*M)
    CONNECT B008.t[0][link][2] TO B008.t[1][link][1]
    SET B008.t[num.of.transp - 1] (type, memsize := "T800", 1*M)
    CONNECT B008.t[num.of.transp - 1][link][3] TO B008.t[0][link][3]
    DO i=1 FOR num.of.transp - 2
      DO
        SET B008.t[i] (type, memsize := "T800", 1*M)
        CONNECT B008.t[i][link][2] TO B008.t[i+1][link][1]
:
}}}
```

>>> howfast1.pgm レベル 2-b

```
#INCLUDE "hostio.inc"
#INCLUDE "howfast1.inc"
#USE "howfast1.c8h"
#USE "operatio.c8h"

{{{  software description
CONFIG
  CHAN OF SP from.isv:
  CHAN OF SP to.isv:
  PLACE from.isv, to.isv ON Hostlink:
  [num.of.transp+1]CHAN OF PARAM pipe:
  -- allocate a cpu and a channel to processes
  PAR
    PROCESSOR B008.t[0]
      howfast1(from.isv, to.isv, pipe[num.of.transp], pipe[0])
    PROCESSOR B008.t[0]
      operatio(pipe[0], pipe[1])
    PAR i = 1 FOR num.of.transp-1
      PROCESSOR B008.t[i]
        operatio(pipe[i], pipe[i+1])
}}}
:

}}}  howfast1.pgm
```

```
>>> integ.inc

{{{  integ.inc
-- nine transputers are installed, all for work
VAL num.of.transp IS 9:
PROTOCOL INTEG
  CASE
    packet; INT;INT;REAL32;REAL64;[num.of.transp + 1]INT
    sum; REAL32;REAL64
    start; INT
    finish
:
}}}  integ.inc


>>> integbos.occ レベル 1

{{{  integbos.occ
...  declarations
...  body of integbos
:

}}}  integbos.occ

>>> integbos.occ レベル 2-a

{{{  declarations
#INCLUDE "integ.inc"
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#USE     "hostio.lib"
#USE     "streamio.lib"
#USE     "snglmath.lib"
#USE     "dblmath.lib"
}}}


>>> integbos.occ レベル 2-b

{{{  body of integbos
PROC integbos(CHAN OF SP from.isv, to.isv, CHAN OF INTEG in, out)
  BOOL loop, ierr:
  BYTE any, bres:
  INT key:
  ...  proc integration
  SEQ
    loop:=TRUE
    WHILE loop
      SEQ
        ...  clear the screen and display the menue
        ...  keyin a number, clear the screen, branch to the selected operati
    so.exit(from.isv, to.isv, sps.success)
}}}

>>> integbos.occ レベル 3-a

{{{  clear the screen and display the menue
CHAN OF SS scrn:
PAR
  so.scrstream.to.ANSI(from.isv, to.isv, scrn)
  SEQ
    ss.goto.xy(scrn, 0, 0)
    ss.clear.eos(scrn)
    ss.goto.xy(scrn, 10, 5)
    ss.write.string(scrn, "Numerical integration of function f(x)")
    ss.write.nl(scrn)
    ss.goto.xy(scrn, 10, 6)
```

```
    -- ss.write.string(scrn,
    -- "f(x) = x ** x + 1(/x ** x) +  cos(exp(-x)) + sin(x *** (-4))")
    ss.write.nl(scrn)
    ss.goto.xy(scrn, 20, 8)
    ss.write.string(scrn, "1)  integration" )
    ss.goto.xy(scrn, 20, 9)
    ss.write.string(scrn, "2)  quit" )
    ss.goto.xy(scrn, 10, 12)
    ss.write.string(scrn, "Select the number of the above operation --: ")
    ss.write.endstream(scrn)
}}}
```

>>> integbos.occ レベル 3-b

```
{{{  keyin a number, clear the screen, branch to the selected operation
so.read.echo.int (from.isv, to.isv, key, ierr )
CASE key
  1 -- integration
    ...   call integration
  2 -- quit
    SEQ
      so.write.nl(from.isv, to.isv)
      so.write.string.nl(from.isv, to.isv, "          ## quit")
      loop:=FALSE
  ELSE
    SEQ
      so.write.string.nl(from.isv, to.isv, "          Bad number. Try again."
      SKIP
      so.getkey(from.isv, to.isv, any, bres)
}}}  end of branches
```

>>> integbos.occ レベル 3-c

```
{{{  proc integration
PROC integration(CHAN OF SP from.isv, to.isv, CHAN OF INTEG in, out)
  ...  declarations

  SEQ
    ...  input the parameters
    PRI PAR
      SEQ
        ...  timer start, arrange the parameters
        ...  start an operation, get an end time
      SKIP
    ...  display the time required for an operation
    CASE sng.dbl.key
      1 -- display the time required for a 32 bit operation
        ...  32 bit
      2 -- display the time required for a 64 bit operation
        ...  64 bit

:
}}}
```

>>> integbos.occ レベル 4-a

```
{{{   call integration
SEQ
  -- check point
  --so.write.string.nl(from.isv, to.isv,
   --"enter into integration from here")
  so.write.nl(from.isv, to.isv)
  integration(from.isv, to.isv, in, out)
  so.write.string(from.isv, to.isv, "press any key :")
  so.getkey(from.isv, to.isv, any, bres)
}}}
```

```
>>> integbos.occ レベル 4-b

{{{  declarations
VAL a32 IS 0.05(REAL32):
VAL a64 IS 0.05(REAL64):
VAL b32 IS 10.0(REAL32):
VAL b64 IS 10.0(REAL64):
TIMER clock:
INT time1, time2, no.of.divs, num.cpu, time.ope:
REAL32 time.int, sum32:
REAL64 sum64:
[num.of.transp + 1]INT points.on.cpu, points.on.cpu.r:
REAL32 sector32, sector32.r:
REAL64 sector64, sector64.r:
INT divs.per.transp, remain, cpu.no:
INT ope.key, ope.key.r, sng.dbl.key, sng.dbl.key.r:
}}}


>>> integbos.occ レベル 4-c

{{{  input the parameters
so.write.string(from.isv, to.isv,
                "          ** precision  1)single or 2)double--: " )
so.read.echo.int(from.isv, to.isv, sng.dbl.key, ierr)
so.write.nl(from.isv, to.isv)
so.write.string(from.isv, to.isv,
 "          ** number of divison on integral range--:")
so.read.echo.int(from.isv, to.isv, no.of.divs, ierr)
so.write.nl(from.isv, to.isv)
so.write.string(from.isv, to.isv,
 "          ** number of transputers(1, ... ,9)--:")
so.read.echo.int(from.isv, to.isv, num.cpu, ierr)
so.write.nl(from.isv, to.isv)
so.write.string.nl(from.isv, to.isv,
   "          ## integration is going on..... ")
--so.write.string.nl(from.isv, to.isv, "end of input parameters")
}}}

>>> integbos.occ レベル 4-d

{{{  timer start, arrange the parameters
clock ? time1
divs.per.transp := no.of.divs / num.cpu
remain := no.of.divs Y num.cpu
SEQ I=0 FOR num.of.transp + 1
  points.on.cpu[I] := 0(INT)
points.on.cpu[1] := divs.per.transp + remain
SEQ i=2 FOR num.cpu - 1
  points.on.cpu[i] := points.on.cpu[i-1] + divs.per.transp
sector32 := (b32 - a32)/(REAL32 ROUND no.of.divs)
sector64 := (b64 - a64)/(REAL64 ROUND no.of.divs)
sum32 := 0.0(REAL32)
sum64 := 0.0(REAL64)
ope.key := 1
}}}


>>> integbos.occ レベル 4-e

{{{  start an operation, get an end time
out ! packet; ope.key;sng.dbl.key;sector32;sector64;points.on.cpu
in ? CASE
  packet; ope.key.r;sng.dbl.key.r;sector32.r;sector64.r;
                                                points.on.cpu.r

    SKIP
```

```
out ! start; 1(INT)
in ? CASE
  start; cpu.no
    out ! finish
out ! sum; sum32;sum64
in ? CASE
  finish
    in ? CASE
      sum; sum32;sum64
        clock ? time2
}}}
```

>>> integbos.occ レベル 4-f

```
{{{  display the time required for an operation
time.opc := time2 MINUS time1
time.int:=(REAL32 ROUND time.opc)
so.write.string(from.isv, to.isv, "          time1=")
so.write.int(from.isv, to.isv, time1, 0)
so.write.nl(from.isv, to.isv)
so.write.string(from.isv, to.isv, "          time2=")
so.write.int(from.isv, to.isv, time2, 0)
so.write.nl(from.isv, to.isv)
so.write.string(from.isv, to.isv, "          time in ticks=")
so.write.int(from.isv, to.isv, time.opc, 0)
so.write.nl(from.isv, to.isv)
}}}
```

>>> integbos.occ レベル 4-g

```
{{{  32 bit
SEQ
  so.write.string(from.isv, to.isv,
   "          32-bit integration= ")
  so.write.real32(from.isv, to.isv, sum32, 0, 0)
  so.write.nl(from.isv, to.isv)
  so.write.string(from.isv, to.isv,
   "          time in uSEC = ")
  so.write.real32(from.isv, to.isv, time.int, 0, 0)
  so.write.nl(from.isv, to.isv)
}}}
```

>>> integbos.occ レベル 4-h

```
{{{  64 bit
SEQ
  so.write.string(from.isv, to.isv,
   "          64-bit integration = ")
  so.write.real64(from.isv, to.isv, sum64, 0,0)
  so.write.nl(from.isv, to.isv)
  so.write.string(from.isv, to.isv,
   "          time in uSEC = ")
  so.write.real32(from.isv, to.isv, time.int, 0, 0)
  so.write.nl(from.isv, to.isv)
}}}
```

>>> integfol.occ レベル 1

```
{{{  integfol.occ
#INCLUDE "integ.inc"
#USE     "snglmath.lib"
#USE     "dblmath.lib"
PROC integfol(CHAN OF INTEG  input, output)
  ...  declarations
  SEQ
    going := TRUE
```

```
    WHILE going
      SEQ
        ...  Input and output parameters
        ... ' receive 'cpu.no', send 'cpu.no'
        CASE ope.key
          1 -- calculation
            ...  Integration
          ELSE
            going := FALSE
          -- end of branch
:
}}}  integfol.occ
```

```
>>> integfol.occ レベル 2-a

{{{  declarations
VAL a32 IS 0.05(REAL32):
VAL a64 IS 0.05(REAL64):
INT  ope.key, sng.dbl.key, cpu.no:
[num.of.transp+1]INT points.on.cpu:
BOOL going:
REAL32 sum32, part.sum32, sector32, x0.s, x1.s, fx0.s, fx1.s:
REAL64 sum64, part.sum64, sector64, x0.d, x1.d, fx0.d, fx1.d:
}}}
```

```
>>> integfol.occ レベル 2-b

{{{  Input and output parameters
input ? CASE
  packet; ope.key;sng.dbl.key;sector32;sector64;points.on.cpu
    output ! packet; ope.key;sng.dbl.key;sector32;sector64;
                          points.on.cpu
}}}
```

```
>>> integfol.occ レベル 2-c

{{{  receive 'cpu.no', send 'cpu.no'
input ? CASE
  start; cpu.no
    output ! start; (cpu.no + 1(INT))
}}}
```

```
>>> integfol.occ レベル 2-d

{{{  Integration
SEQ
  CASE sng.dbl.key
    1 -- 32-bit operation
      ...  single precision
    2 -- 64-bit operation
      ...  double precision
    ELSE
      SKIP
}}}
```

```
>>> integfol.occ レベル 3-a

{{{  single precision
SEQ
  x0.s := a32 +
        ((REAL32 ROUND points.on.cpu[cpu.no-1]) * sector32)
  x1.s := x0.s + sector32
  part.sum32 := 0.0(REAL32)
  --fx1.s := x0.s
```

```
  fx1.s := ((x0.s*x0.s) + (1.0(REAL32)/((x0.s*x0.s)*x0.s))) +
    (COS(EXP(-x0.s)) + SIN(POWER(x0.s, (-4.0(REAL32)))))
  IF
    points.on.cpu[cpu.no] <> 0(INT)
      ...  iterate calcuration
    TRUE
      SKIP
  input ? CASE
    finish
      output ! finish
  input ? CASE
    sum; sum32;sum64
      SEQ
        sum32 := sum32 + part.sum32
        output ! sum; sum32;sum64
}}}
```

```
>>> integfol.occ レベル 3-b

{{{  double precision
SEQ
  x0.d := a64 +
    ((REAL64 ROUND points.on.cpu[cpu.no-1]) * sector64)
  x1.d := x0.d + sector64
  part.sum64 := 0.0(REAL64)
  --fx1.d := x0.d
  fx1.d := ((x0.d*x0.d) + (1.0(REAL64)/((x0.d*x0.d)*x0.d))) +
    (DCOS(DEXP(-x0.d)) + DSIN(DPOWER(x0.d, (-4.0(REAL64)))))
  IF
    points.on.cpu[cpu.no] <> 0(INT)
      ...  iterate calcuration
    TRUE
      SKIP
  input ? CASE
    finish
      output ! finish
  input ? CASE
    sum; sum32;sum64
      SEQ
        sum64 := sum64 + part.sum64
        output ! sum; sum32;sum64
}}}
```

```
>>> integfol.occ レベル 4-a

{{{  iterate calcuration
SEQ i=points.on.cpu[cpu.no-1] FOR
    (points.on.cpu[cpu.no] - points.on.cpu[cpu.no-1])
  SEQ
    fx0.s := fx1.s
    --fx1.s := x1.s
    fx1.s := ((x1.s*x1.s) + (1.0(REAL32)/((x1.s*x1.s)*x1.s))) +
      (COS(EXP(-x1.s)) + SIN(POWER(x1.s, (-4.0(REAL32)))))
    part.sum32 := part.sum32 + (sector32 *
                    ((fx1.s + fx0.s) / 2.0(REAL32)))
    x1.s := x1.s + sector32
}}}
```

```
>>> integfol.occ レベル 4-b

{{{  iterate calcuration
SEQ i=points.on.cpu[cpu.no-1] FOR
    (points.on.cpu[cpu.no] - points.on.cpu[cpu.no-1])
  SEQ
    fx0.d := fx1.d
    --fx1.d := x1.d
```

```
    fx1.d := ((x1.d*x1.d) + (1.0(REAL64)/((x1.d*x1.d)*x1.d))) +
     (DCOS(DEXP(-x1.d)) + DSIN(DPOWER(x1.d, (-4.0(REAL64)))))
    part.sum64 := part.sum64 + (sector64 *
                        ((fx1.d + fx0.d) / 2.0(REAL64)))
    x1.d := x1.d + sector64
}}}
```

>>> numeinte.pgm レベル 1

```
{{{  numeinte.pgm
#INCLUDE "integ.inc"

...  hardware description

-- Code
#INCLUDE "hostio.inc"
#INCLUDE "integ.inc"
#USE "integbos.c8h"
#USE "integfol.c8h"

...  software description

}}}  numeinte.pgm
```

>>> numeinte.pgm レベル 2-a

```
{{{  hardware description
--  network on B008 INMOS board
--  kilo, mega
VAL  K  IS 1024:
VAL M IS K*K:
[num.of.transp]NODE B008.t:
ARC Hostlink:
NETWORK
  DO
    CONNECT B008.t[0][link][0] TO HOST WITH Hostlink
    SET B008.t[0] (type, memsize := "T800", 2*M)
    CONNECT B008.t[0][link][2] TO B008.t[1][link][1]
    SET B008.t[num.of.transp - 1] (type, memsize := "T800", 1*M)
    CONNECT B008.t[num.of.transp - 1][link][3] TO B008.t[0][link][3]
    DO i=1 FOR num.of.transp - 2
      DO
        SET B008.t[i] (type, memsize := "T800", 1*M)
        CONNECT B008.t[i][link][2] TO B008.t[i+1][link][1]
:
}}}
```

>>> numeinte.pgm レベル 2-b

```
{{{  software description
-- software deacription
CONFIG
  CHAN OF SP from.isv:
  CHAN OF SP to.isv:
  PLACE from.isv, to.isv ON Hostlink:
  [num.of.transp+1]CHAN OF INTEG pipe:
  -- allocate a cpu and a channel to processes
  PAR
    PROCESSOR B008.t[0]
      integbos(from.isv, to.isv, pipe[num.of.transp], pipe[0])
    PROCESSOR B008.t[0]
      integfol(pipe[0], pipe[1])
    PAR i = 1 FOR num.of.transp-1
      PROCESSOR B008.t[i]
        integfol(pipe[i], pipe[i+1])
:
}}}
```