島根大学教育学部紀要（自然科学）第25巻　1頁～6頁　平成3年12月

並列処理システムによる2次元積分の数値計算

福島　誠*

Makoto FUKUSHIMA
Numerical Calculations of Two-Dimensional Integral
with Parallel Processing System

1 まえがき

プロセッサ単体の演算処理能力が限界に近くなるにつれて，複数の中央処理装置（ＣＰＵ），またはマイクロプロセッサ（ＭＰＵ）を同時に使用する並列処理コンピュータが注目されている。これは並列処理を行うことによりプロセッサの数に比例した処理能力が得られることが期待できるからである。特に，高速な演算処理が要求されるスーパコンピュータ，あるいは画像処理システムではこのような並列処理（ベクトル処理）システムは既に一般化している。スーパコンピュータでは利用者が並列処理を考慮したコーディングをしなくても，指定した部分をコンパイラが自動的に並列処理用に変換してくれる自動ベクトル化コンパイラも用意されている。但し，スーパコンピュータの利用は現時点では特定の大型計算機センターの利用者に限られており，遠隔地からの利用は通信回線を経由して可能ではあるが，Ｉ／Ｏ関係の制約や利用料金のことを考えると研究室から常時利用するわけにはいかない。しかし，研究室でもパーソナルコンピュータを利用すれば並列処理システムの構築は可能である。本報告ではこのパーソナルコンピュータ用の並列処理システムとして，トランスピュータ（INMOS T-800）を使用したシステムを採用し，これを2次元積分の数値計算に適用して性能評価をした結果について報告する。このシステムには前述の自動ベクトル化コンパイラは用意されていないが，専用のコンパイラによるプログラムの並列化により，ＭＰＵ数に応じて数十 MFLOPS 程度の演算能力を得ることも可能である。

2　並列処理システム

2．1　並列処理アーキテクチャ

並列処理計算機のアーキテクチャとしてはフリンによる分類法として

(1) ＳＩＳＤ（Single Instruction Single Data stream）：単一命令単一データ流方式
(2) ＳＩＭＤ（Single Instruction Multiple Data stream）：単一命令複数データ流方式
(3) ＭＩＳＤ（Multiple Instruction Single Data stream）：複数命令単一データ流方式
(4) ＭＩＭＤ（Multiple Instruction Multiple Data stream）：複数命令複数データ流方式

の4種類に分類されている。[1] この分類によれば，スーパコンピュータなどの演算パイプライン方式はＳＩＭＤ方式となるが，データ流の考え方からＳＩＳＤ，あるいはＭＩＳＤとも考えられるので，単にパイプライン方式とされている。ここで利用するトランスピュータシステムは上記の分類に従えば，ＭＩＭＤ方式に相当すると考えられる。この方式は複数のプロセッサが互いに独立に動作し，通信機能により互いにデータのやりとりを行う方式である。この方式はメモリの共有の仕方により密結合マルチプロセッサと疎結合マルチプロセッサにさらに分類できる。前者はメモリを共有するのに対し，後者は個別にローカルメモリを有する。トランスピュータによるマルチプロセッサは後者に属しており，後述するようにプロセッサ間のデータは，20Mbpsのシリアルリンクと呼ばれる通信チャネルを介して行われる。この方式はプロセッサが比較的多くなっても，共有メモリ方式のようにメモリ競合の問題は起きない。

2．2　並列処理用ＭＰＵトランスピュータ

トランスピュータは英国の INMOSで開発されたRISCタイプのマイクロプロセッサで，現在までに 16bitと32 bit タイプが開発されている。ここで使用しているトランスピュータは T-800という32bit タイプである。トランスピュータという名称はトランジスタとコンピュータとの合成語であり，一つの部品（トランジスタ）でもあり，一つのコンピュータでもあるという意味であろう。

＊島根大学教育学部保健体育研究室

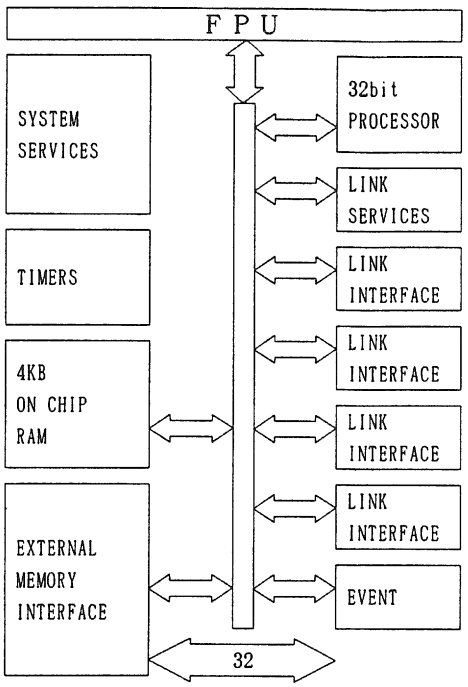

図1　T800のブロック図
Fig.1 Block Diagram of T800

図1はT-800のアーキテクチャのブロック図である。[2] 内部バス及び外部バスとも32bitであり，浮動小数点演算ユニット（FPU）を内蔵している。FPUのないタイプはT-400というタイプになる。図1から特徴的なことは，内部に4kbyteのon-chip RAM(SRAM)を用意していること，そしてプロセッサ間通信のための4チャネルのlink interfaceが用意されていることである。4kbyteのon-chip RAMは40nsecのSRAMで，キャッシュメモリとして使用されるのではなく，主記憶の一部として位置づけられており，プログラムコードが配置可能である。もしプログラムコード全てがこの内部RAMに配置されれば、より高速な処理が可能となる。link interfaceはそれぞれIn，Outのチャネルから構成されている。チャネルの通信速度は20Mbpsまたは10Mbps 2種類から選択できる。

トランスピュータの処理能力はメーカー公称値で，T-800(25MHz)では12.5MIPS，浮動小数点演算能力は，1.87 MFLOPSとなっている。この処理能力は汎用MPUのI-80486(25MHz),MC68040(25MHz)とほぼ同じレベルとも考えられるが，80486,68040はともにCISCタイプなので単純な比較は出来ない。

2．3　トランスピュータによる並列処理

複数のトランスピュータを使用して並列処理を実現するには専用の開発言語を使用しなければならない。トランスピュータの開発メーカであるINMOSからTDS(Transputer Development System)と開発言語のOCCAMが提供されている。しかし，このOCCAMは並列処理の記述が容易で効率がよいという利点がある反面，独自の言語という感じが強く，これまでに開発されてきた数学ライブラリ等のサブルーチン類が利用できないという欠点がある。また，科学技術計算では重要な複素数演算などの機能もサポートされていない。このような点を補うためにトランスピュータにはParallel FORTRAN,Parallel Cといった汎用言語も用意されており，これまで開発されたライブラリも，ソースプログラムを再コンパイルすることで利用できる。但し，マルチトランスピュータを利用した並列処理を行うには，ソースプログラムを書きなおす必要がある。

図2はトランスピュータを2個使用した並列処理システムのブロック図である。ホストコンピュータにはトランスピュータへの実行プログラムのロードと，トランスピュータからのI／O要求を処理するためのサーバプログラムが動作している。このプログラムはトランスピュータ上のfilterタスクを介してユーザプログラムとの間でのデータの受渡しを行う。図2ではユーザプログラムはtask 0，task 1，task 2という3種類のタスクに分割されており，並列処理用のソースコードを追加している。ここで，task 0はtask 1，task 2へデータを渡した後，task 1，task 2から計算結果を受取りホストコンピュータへ転送することが主な仕事になる。従って，task 1，task 2が実質的な並列処理を担当し，task 0の計算量は少ないので，図2のようなトランスピュータ2個のシステムでは，task 0，task 1を同一のトランスピュータに割り当てている。

ここで並列処理の簡単な例を考える。正の整数の1から$2N$までの総和を求め，その結果をSとすると，この計算は

$$S=\sum_{i=1}^{2N} i \qquad (1)$$

と定義できる。また，式(1)は

$$S=\sum_{i=1}^{N} i+\sum_{i=N+1}^{2N} i=S_1+S_2 \qquad (2)$$

のように分解できる。式(2)の右辺の第1項をS_1，第2項をS_2とすると，図2の並列処理システムではS_1をtask 1で，S_2をtask 2で計算させることになる。従ってトランスピュータの数がN個の場合は

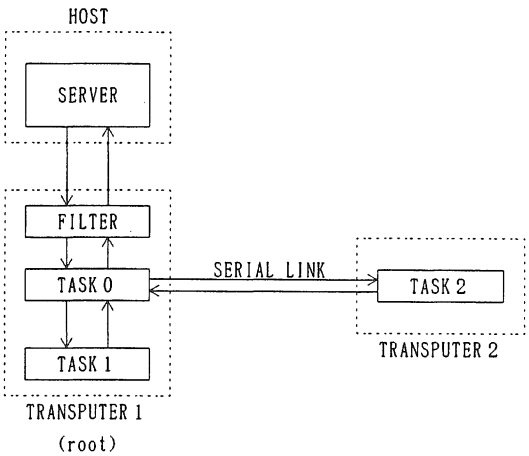

図2　トランスピュータ2個による並列処理
Fig.2 Parallel Processing with Two TRANSPUTERS.

$$S = S_1 + S_2 + \cdots\cdots + S_N \qquad (3)$$

のように式(1)を分解してそれぞれタスクに割り当てればよい。しかし，この方法ではタスクの数もN個作成しなければならない。もちろん式(1)のような単純な問題であればタスクの作成も簡単であるが，一般的にはタスクの数が増加するほどプログラムの変更は複雑になる。また，各トランスピュータのネットワークの形状を変更した場合には，その情報をタスクプログラムに反映しなければならない。従って，この方法は並列処理システムの接続形態に適した並列処理プログラムが作成できる利点があるが，ネットワーク形態を頻繁に変更するようなシステムでは，その都度プログラムを変更しなければならず不便である。

この問題を避けるためにParallel FORTRANではflood-fill 法[3]という並列処理法が用意されている。これは任意の数のトランスピュータに対応して，システム側で自動的に負荷分散を実行する処理方法で，トランスピュータの接続数，ネットワーク形態が変化しても原則的にソースプログラムを変更することなく並列処理が可能である。この方法では並列プログラムはmasterとworkerと呼ばれるタスクから構成されており，masterタスクはrootと呼ばれるトランスピュータにロードされ，workerタスクはrootを含む全てのトランスピュータにロードされる。この関係を図3に示す。workerはmasterから指示された範囲の計算を行い結果を返す。式(1)で考えればworker は

$$S_k = \sum_{i=P_{sk}}^{P_{ek}} i \quad (1 \leqq P_{sk} \leqq P_{ek} \leqq 2N) \qquad (4)$$

の計算を行う。masterではP_{sk}とP_{ek}の差が等しくなるように値を決めてworker に渡し，式(4)の結果を基に

$$S = \sum_{k=1}^{P} S_k \qquad (5)$$

の計算をすればよい。ここでPはパケット数で，計算の分割数を表している。Pの値は通常トランスピュータの数よりも大きくして，負荷分散をする場合に遊んでいるトランスピュータがないようにする。しかし，実際にはシステムのネットワーク形状により，rootトランスピュータから遠いトランスピュータでは命令が届く前に他のトランスピュータで実行されてしまい，期待した処理能力が得られない場合がある。これは各workerの負荷が軽く，トランスピュータが常に負荷待ちの状態になりやすいことが原因である。これ以外に flood-fill 法の欠点としては，利用者が直接プロセッサを指定出来ないため，個々の問題に対応した最適な並列処理プログラムが作成できないことが挙げられる。しかし，前述のようにソースプログラムの変更なしで，ネットワークの変更に対応できることから，汎用的な並列処理ライブラリには適した処理法と考えられる。

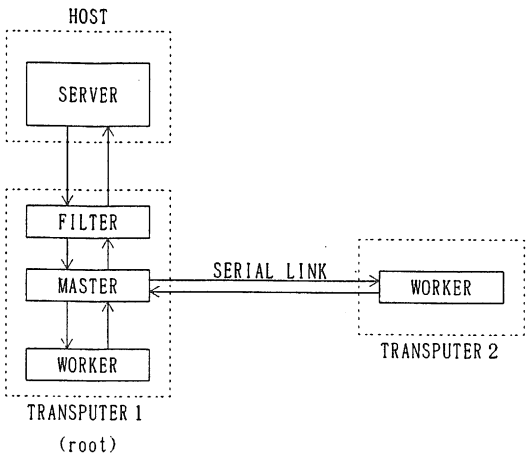

図3 Flood-fill 法による並列処理
Fig.3 Parallel Processing by Flood-fill Method.

3 並列処理による数値積分

3．1 優良格子点法

ここでは具体的な並列処理問題として，優良格子点（Good Lattice Point）法を用いた2次元数値積分の例を考える。GLP法による数値積分は数論的数値積分と呼ばれる方法の一つで，[4] 多次元数値積分法として最近注目されている。GLP法による数値積分は以下のように定義できる。即ち，一定の条件を満足するS次元の関数 $f(\mathrm{X})$ $(\mathrm{X}=(\mathrm{x}_1,\mathrm{x}_2,\cdots,\mathrm{x}_s))$の，$S$次元の単位超立方体 $I^s=[0,1]^s$ 領域での積分

$$\int_{I^s} f(\mathrm{x})d\mathrm{x} \tag{6}$$

は次式で数値計算できる。[4][5]

$$\frac{1}{N}\sum_{k=0}^{N-1} f\left(\left\{k\frac{g_1}{N}\right\},\cdots\cdots,\left\{k\frac{g_s}{N}\right\}\right) \tag{7}$$

ここで，$\{kg_1/N\}$ はkg_1/Nの小数部分を表す。また，N,g_s はあるきまった自然数で，2次元の場合

$$N=F_n \tag{8}$$

$$\{g_1,g_2\}=\{1,F_{n-1}\} \tag{9}$$

で与えられる。ここでF_n はフィボナッチ数である。

式(6)において適当な変数変換することにより，一般的に$N\to\infty$においてより精度の良い積分結果が得られる。但し，分点数Nを大きくしなければならないので計算時間が増加する。2次元の数値積分では，Nの値をフィボナッチ数に従って増加させて，εを相対誤差として

$$\left|\frac{I_n-I_{n-1}}{I_n}\right|<\varepsilon \tag{10}$$

であれば収束と判定する。ここで I_n は$N=F_n$ の場合の計算結果である。

3．2 GLP法の並列処理化

GLP法を並列処理化するには前述のようにパラレルタスク化，または flood-fill 法の2種類が考えられる。まずパラレルタスク化による並列処理を考える。トランスピュータ2個の場合には式(7)を

$$\frac{1}{N}\sum_{k=0}^{N-1} f(\boldsymbol{\alpha})=\frac{1}{N}\sum_{k=0}^{[N/2]} f(\boldsymbol{\alpha})+\frac{1}{N}\sum_{k=[N/2]+1}^{N-1} f(\boldsymbol{\alpha}) \tag{11}$$

$$\boldsymbol{\alpha}=\left(\left\{k\frac{g_1}{N}\right\},\ \cdots\cdots,\ \left\{k\frac{g_s}{N}\right\}\right) \tag{12}$$

のように分解して，それぞれ独立のタスクとして実行させればよい。ここで［ ］はガウス記号である。flood-fill 法では $[j]=[(N-1)/P]$ として

$$\frac{1}{N}\sum_{k=0}^{N-1} f(\alpha) = \frac{1}{N}\left\{\sum_{k=0}^{[J]} f(\alpha) + \sum_{k=[J]+1}^{2[J]} f(\alpha) + \cdots\cdots + \sum_{k=(P-1)[J]+1}^{P[J]} f(\alpha)\right\} \quad (13)$$

のようにP個のパケットに分解して実行する。但し，MOD(N-1,P)を(N-1)/Pの剰余とすると，MOD(N-1,P)≠0の場合，式(13)にループ回数MOD(N-1,P)のパケットが1個追加される。パケット数PはNの値に対応して変化させ

$$13 \leqq N < 100 \rightarrow P = 2$$
$$100 \leqq N < 1000 \rightarrow P = 10$$
$$1000 \leqq N \rightarrow P = 20$$

とした。これらのPの値は実測結果を基にして決定した。従ってトランスピュータの数とネットワークの接続形態が変化すれば，再検討する必要がある。

3．3　被積分関数

ここで使用する2次元積分は

$$\frac{1}{l_d}\int_0^{l_d}\int_0^{t} I(t')f(x,\ t-t')\,dt'dx \quad (14)$$

で与えられる。被積分関数 $I(t)$,$f(x,t)$ は

$$I(t) = -I_{inj}\cos(\omega t) \quad (15)$$

$$f(x,\ t) = v(t)g(x,\ t) \quad (16)$$

で与えられる。[6]　ここで

$$v(t) = v_0\exp(-t/\tau) + v_s \quad (17)$$

$$g(x,\ t) = \frac{1}{\sqrt{2\pi S(t)}}\exp\left\{\frac{-[x-M(t)]^2}{2S(t)}\right\} \quad (18)$$

$$M(t) = \int_0^t v(t')\,dt' \quad (19)$$

$$S(t) = 2\int_0^t D(t')\,dt' \quad (20)$$

である。これらの式はQWITTダイオードにおける量子井戸領域から，ドリフト領域へのキャリヤの注入現象を解析するための理論式である。ここで，I_{inj} はドリフト領域への注入電流振幅，l_d はドリフト領域長，ωは角周波数，$v(t)$　はキャリヤのドリフト領域の平均速度で，v_o は注入時の初速度，v_s は飽和速度，τは初速度の過度現象の時定数，$g(x,t)$ は拡散を考慮した場合のキャリヤの分布関数である。また，$D(t)$はキャリヤの拡散係数である。計算に使用したこれらの定数の値を表1，2に示す。

4　結果と評価

2個のトランスピュータ (T-800,20MHz, 外部メモリサイクル3)を使用した並列処理システムによる，2次元積分の実行時間の結果を表3，4に示す。ホストコンピュータはPC-286VEである。計算は表1，2に示した4種類のダイオードデータを使用し，$I(t)$の1周期内の21点について実行した。実行時間の計測はストップウォッチによる手動計測である。なお，並列処理システムの評価は以下のような項目について行った。

(1) 実行時間：プロセッサの数をmとした場合の実行時間をt_m とする。
(2) コスト：実行時間t_m と，使用したプロセッサの数mとの積　c ($=t_m \cdot m$)。
(3) 速度向上率：逐次アルゴリズム（プロセッサ数は1）と並列アルゴリズム（プロセッサ数はm）の実行時間の比　r ($=t_1/t_m$)。
(4) 効率：逐次アルゴリズムの実行時間に対する並列アルゴリズムのコストの割合を効率eとする。

$$e = \frac{t_1}{c} = \frac{t_1}{t_m \cdot m} = \frac{r}{m} \quad (21)$$

一般に効率 e はメッセージの送受信等によるシステムのオーバヘッドにより$e<1$となる。

表3，4の結果よりパラレルタスクによる結果が flood-fill 法による結果よりも効率が良いことがわかる。また，効率についてはさらに改良の余地があるかどうか検討することも必要である。

文　献

(1) 渡辺　勝正：“並列処理概説”，コロナ社，1991.
(2) “The transputer data book”，INMOS Ltd.
(3) 山本，中井，村上：“トランスピュータ入門”，日刊工業新聞社，1990.
(4) 杉原　正顯：“数論的数値積分法”，数理科学，238，pp.31-37(1983).
(5) 森　正武：“FORTRAN77数値計算プログラミング”，岩波書店，pp.205-214(1987).
(6) 福島　誠：“QWITTダイオードの注入キャリヤの拡散と過度現象効果”，信学論(C-II)，J73-C-II，3，pp.187-193(1990).

表1 計算に用いたダイオード定数（全ダイオード共通）

量子井戸領域長(lq)	7.5	nm
量子井戸領域のRF電圧(Vq)	0.0414	V
過度現象の時定数(τ)	8.0×10^{-14}	s
ドリフト飽和速度(v_s)	6.0×10^{6}	cm/s
注入時の初速度(v_o)	6.4×10^{7}	cm/s
拡散係数($D(t)$)	15.0	cm^2/s
ダイオードの寄生抵抗(Rs)	0.0	ohm

表2 各ダイオードの定数

ダイオードNo.		1	2	3	4
ドリフト領域長(l_d)	nm	246	181	110	96
注入電流振幅(I_{inj})	A	0.302	0.115	1.56×10^{-2}	6.09×10^{-3}
動作周波数($\omega/2\pi$)	GHz	60	90	200	300
断面積(S)	$\times10^{-8}cm^2$	2080	790	110	42

表3 実行結果（$\varepsilon = 10^{-3}$）

方法 / 項目	serial	parallel task	flood-fill
プロセッサ数(m)	1	2	2
実行時間(t_m, 秒)	$t_1 = 183$	$t_2 = 120$	$t_2 = 126$
コスト(c)	183	240	252
速度向上率(r)	1.0	1.525	1.452
効率(e)	1.0	0.763	0.762

表4 実行結果（$\varepsilon = 10^{-4}$）

方法 / 項目	serial	parallel task	flood-fill
プロセッサ数(m)	1	2	2
実行時間(t_m, 秒)	$t_1 = 341$	$t_2 = 220$	$t_2 = 238$
コスト(c)	341	440	476
速度向上率(r)	1.0	1.550	1.433
効率(e)	1.0	0.775	0.716