計測自動制御学会東北支部第188回研究集会（2000.6.23）
資料番号　188－5

常微分方程式数値解法の並列アルゴリズム

Parallel Algorithms for the Numerical Solution of the Ordinary Differencial Equation

○川崎拓弥、奈良久

○ Takumi Kawasaki , Hisashi Nara

八戸工業大学

Hachinohe Institute of Technology

キーワード：並列アルゴリズム(Parallel Algorithm)、複区分予測子法、
PVM(Parallel Virtual Machine)

連絡先：〒031　八戸市妙字大開88－1　八戸工業大学電気電子工学専攻　川崎拓弥
E-mail：m00204@stud.hi-tech.ac.jp

1.　はじめに

近年のVLSI技術のめざましい発展に伴い、並列処理計算機の研究が盛んに行われており、高速演算が必要である科学用数値計算の分野で大いに期待されている。現在実用化されている科学計算用の高速計算機は更なる高速演算に向けて様々な努力が続けられているが、スピード向上の原理的限界に近づいていると考えられている。VLSI技術の発展により単一プロセッサの性能は著しく向上したが、これを多数用いる並列処理システムが、瞬間風速的には科学計算用高速計算機には及ばないが、総体的には単位時間にはるかに莫大な計算量を消化できるので、注目を浴びている。

実際、並列プロセッサを用いた様々な並列処理計算機システムが多数提案されている。しかしながら、このような並列処理計算機に対する数値計算アルゴリズムについては、行列演算などのベクトル演算以外については未だ十分には研究されていない。特に、高次代数方程式、常微分方程式や偏微分方程式などの数値解析手法は逐次近似のアルゴリズムを用いており、この種の並列処理アルゴリズムに関しては従来ほとんど研究されていなかった。本報告では、初期値問題として科学の分野のみならず広く用いられている常微分方程式を取り上げ、その並列数値計算アルゴリズムを示すとともに、従来のアルゴリズムとの比較検討した結果について述べる。

2.　一般的な常微分方程式の解法

常微分方程式は、熱伝導、力学系あるいは人口の増加などの様々な分野の数学的モデルとして用いられている。特に、$y_0 = y(x_0)$の条件の下に１階常微分方程式、

$$\frac{dy}{dx} = f(x, y) \quad \cdots (1)$$

を満足する特殊解$y(x)$を求める初期値問題は

各種の制御系において用いられ、その数値解法の高精度化と高速化が必要とされる。以下従来用いられてきた各種の逐次型の方法についてまとめておく。これらの方法は、後述する並列アルゴリズムの基礎をなすものである。

2.1 Euler 法

Euler 法は初期値問題の近似法の中で最も簡単なものである。Euler 法は精度が限定されており、比較的単純なため、誤差解析に関して多くの研究がなされている。(1) 式で示される初期値問題の数値解は、漸化式

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x,y)dx \quad \cdots (2)$$

で求めることが出来る。Euler 法は $f(x,y)$ が区間 (x_n, x_{n+1}) で十分滑らかだとして、(2) 式を

$$\left.\begin{aligned} y_{n+1} &= y_n + hf(x_n, y_n) \quad \cdots (3) \\ y_n &= y(x_n) \\ y_0 &= y(x_n) \\ h &= (x_{n+1} - x_n) \end{aligned}\right\} \quad \cdots (4)$$

で近似して求める解法である。

2.2 Taylor 展開法

(2) 式に Taylor 展開法を用い k 次以上の微分項を無視すれば、

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2} f'(x_x, y_n) + \cdots + \frac{h^k}{k!} f^{(k-1)}(x_n, y_n) \quad \cdots (5)$$

が得られる。(5) 式において $k=1$ とすれば Euler 法に一致する。

2.3 Runge-Kutta 法

(5)式は $f', f'', \ldots, f^{(k-1)}$ を含むために実際には非常に複雑である。Runge-Kutta 法は $f(x,y)$ の微分を求める代わりに (x_n, y_n) の近くの点の $f(x,y)$ を計算することにより間接的に Taylor 法の展開係数を求めた。4 次の Runge-Kutta 法は

$$y_{(n+1)} = y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \quad \cdots (6)$$

$$\left.\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= hf(x_n + h, y_n + k_3) \end{aligned}\right\} \quad \cdots (7)$$

で与えられ、常微分方程式で現在でもよく用いられている解法である。

3. 並列数値計算アルゴリズム

3.1 複区分予測子法

並列処理計算システムは様々な構成が考えられているが、ここではN個のプロセッサと共有メモリから構成されたシステムを考えている。

等間隔 h で並んだ分点、$x_1, x_2, \ldots, x_N$ に対する N 個の初期近似解 $y_1, y_2, \ldots, y_N$ が何らかの方法で求まっているものとする。(2) 式はパラメータ M を導入することにより

$$y_{n+1} = y_{n-M} + \int_{x_{n-M}}^{x_{n+1}} f(x,y)dx \quad \cdots (8)$$

となる。ここで M は正の整数または0である。(8) 式の $f(x,y)$ に対して Newton の後退補間多項式を用いると (8) 式は、

$$y_{n+1} = y_{n-M} + \int_{x_{n-M}}^{x_{n+1}} [f_n + \frac{\nabla f_n}{h}(x - x_n) + \frac{\nabla^2 f_n}{2h^2}(x - x_n)(x - x_{n-1}) + \cdots + \frac{\nabla^k f_n}{k! h^k}(x - x_n)\cdots(x - x_{n-k+1})]dx \quad \cdots (9)$$

となる。(9) 式をさらに変形すると次式が求まる。

$$y_{n+1} = y_{n-M} + \sum_{k=0}^{K} hP_k' \nabla^k f_n \qquad \cdots (10)$$

$$P_k' = \frac{1}{k!}\int_M^1 u(u+1)\cdots(u+k-1)du \qquad \cdots (11)$$

$$P_0' = 1$$

$$\nabla^k f_n = f_n - \binom{k}{1} f_{n-1} + \binom{k}{2} f_{n-2} - \cdots + (-1)^k f_{n-k} \qquad \cdots (12)$$

$$f_n = f(x_n, y_n)$$

(10) 式は更に、

$$y_{n+1} = y_{n-M} + \sum_{k=0}^{K} hP_k f_{n-k} \qquad \cdots (13)$$

$$P_k = \frac{1}{k!}\int_M^1 u(u+1)\cdots(u+k-1)du \qquad \cdots (14)$$

となり、(13) 式を複区分予測公式と呼んでいる。特に、M=3、K=2 の場合、(13) 式は、

$$y_{n+1} = y_{n-3} - \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \qquad \cdots (15)$$

となり、(15) 式を Milne の予測公式と一致する。

関数$f(x,y)$をk次のNewtonの前進補間多項式で近似すると、(8) 式は、

$$y_{n+1} = y_{n-M} + \int_{x_{n-M}}^{x_{n+1}} [f_n + \frac{\Delta f_n}{h}(x-x_n) + \frac{\Delta^2 f_n}{2h^2}(x-x_n)(x-x_{n+1})$$

$$+ \cdots + \frac{\Delta^k f_n}{k!h^k}(x-x_n)\cdots(x-x_{n+k-1})]dx \qquad \cdots (16)$$

となる。

(10) 式から (15) 式を導出した手法とほとんど同じ手法を適用すると、(15) 式に対応する式

$$y_{n-1} = y_{n+3} - \frac{4h}{3}(2f_n - f_{n+1} + 2f_{n+2}) \qquad \cdots (17)$$

が得られる。(15) 式と (17) 式は$M-n$の値が正になるか負になるかに従って使い分ける。

(15)・(17) 式の複区予測公式を前もって求めておいた初期近似解$\{y_n^{(0)}\}$に適用すると第1次の近似解$\{y_n^{(1)}\}$が求まる。$y_n^{(1)}$を求める場合、(15) 式の右辺は$y_{n-2}^{(0)}$と$y_{n-1}^{(0)} \equiv f_{n-1}(x_{n-1}, y_{n-1}^{(0)})$、$y_{n-2}^{(0)} \equiv f_{n-2}(x_{n-2}, y_{n-2}^{(0)})$および$y_{n-3}^{(0)} \equiv f_{n-3}(x_{n-3}, y_{n-3}^{(0)})$から成っている。同様に、$y_{n+1}^{(1)}$は、$y_{n-3}^{(0)}$、$y_n^{(0)}$、$y_{n-1}^{(0)}$および$y_{n-2}^{(0)}$から (15) 式に従って計算すればよい。$\{y_n^{(0)}\}$はすべて共有メモリに保存されているから、結局$y_n^{(1)}$と$y_{n+1}^{(1)}$は2個のプロセッサがあれば、並列に出来ることになる。同様の考察からn個のプロセッサがあれば$\{y_n^{(1)}\}$はすべて並列に計算できる。ただし、刻み幅hは$y_n^{(0)}$に対してはnhととる。

3.2 並列 Euler 法

$\{x_n\}$に対する初期近似解$\{y_n^{(0)}\}$を (3) 式より求める。

$$y_{n+1} = y_n + nhf(x_n, y_n) \qquad (n = 1,2,\cdots,N)$$

$$y_{n+1}^{(i+1)} = y_{n-M}^{(i)} + \sum_{k=0}^{K} hP_k f_{n-k}^{(i)} \qquad \cdots (18)$$

$$(n = M, M+1, M+2, \cdots N) \qquad \cdots (19)$$

$$f_{n-k}^{(i)} = f(x_{n-k}, y_{n-k}^{(i)}) \qquad \cdots (20)$$

が求まり、N 個のプロセッサにて独立に逐次近似計算することが出来る。(19) 式において$n-M<0$の場合は、

$$y_{n+1}^{(i+1)} = y_{n-M}^{(i)} + \sum_{j=0}^{J} hP_i' f_{n+1-j}^{(i)} \quad (n-M \geq 0)$$

の予測子修正子公式を適用する。 $\cdots (21)$

3.3 並列 Runge-Kutta 法

初期近似値$\{y_n^{(0)}\}$を2－3 で示した4次の Runge-Kutta 法により求める。

$$y_n^{(0)} = y_0 + \frac{nh(k_1 + 2k_2 + 2k_3 + k_4)}{6} \qquad \cdots (22)$$

ここで、k_1, k_2, k_3およびk_4は (7) 式のhを

nhに置き換えて求める。(22) 式により求めた $\{y_n^0\}$ を用いて (19)、(21) 式を用いてN個並列に逐次計算を行い微分方程式の数値解を求めていく。

4. 数値計算結果

3.2、3.3で示した並列アルゴリズムを用いて数値計算を行う。この並列アルゴリズムは、逐次近似手法を用いているため収束の判定を行う必要がある。従来の方法では予測子と修正子を用いて判定を行っていたが、並列計算では全体の収束判定を行わなければならない。

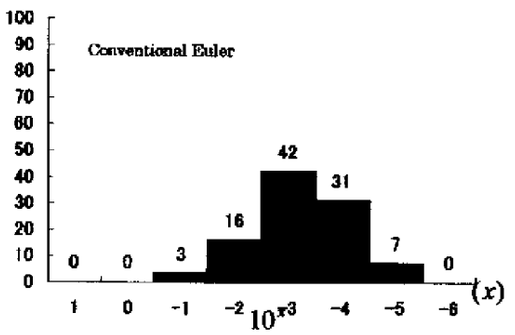
ここでは、相対誤差

$$\varepsilon_n^{(i+1)} = \left| \frac{y_n - y_n^{(i+1)}}{y_n} \right| \qquad \cdots (23)$$

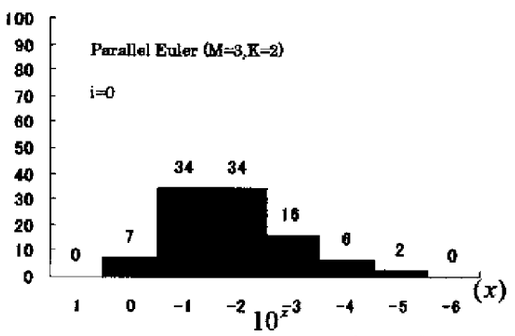
を定義し、$\varepsilon_n^{(i+1)}$ と $\varepsilon_n^{(i)}$ の分布変化を収束判定に用いる。実用的には、真値は未知であるので (28) 式の相対誤差を

$$\varepsilon_n^{(i+1)} = \left| \frac{y_n^{(i+1)} - y_n^{(i)}}{y_n^{(i+1)}} \right| \qquad \cdots (24)$$

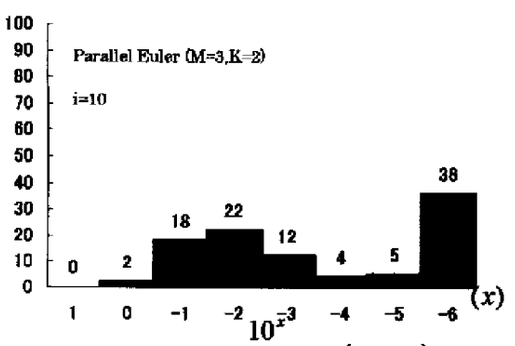
と定義し、$\varepsilon_n^{(i+1)}$ の分布変化を収束の判定に用いればよい。



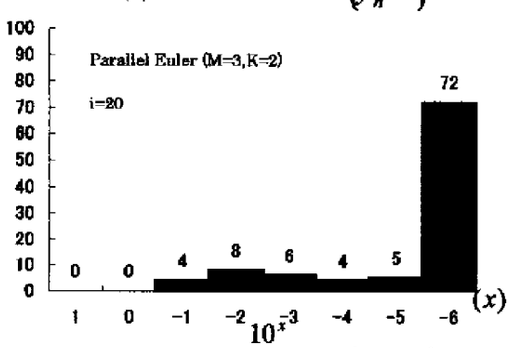
(a)Euler 法



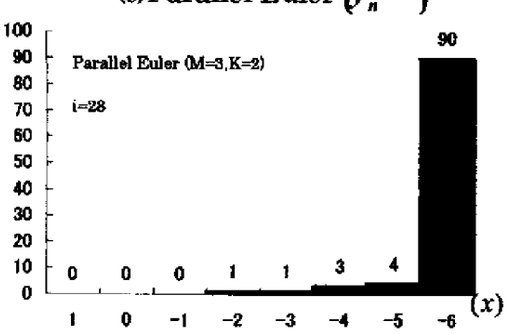
(b)Parallel Euler $\{y_n^{(0)}\}$



(c)Parallel Euler $\{y_n^{(10)}\}$

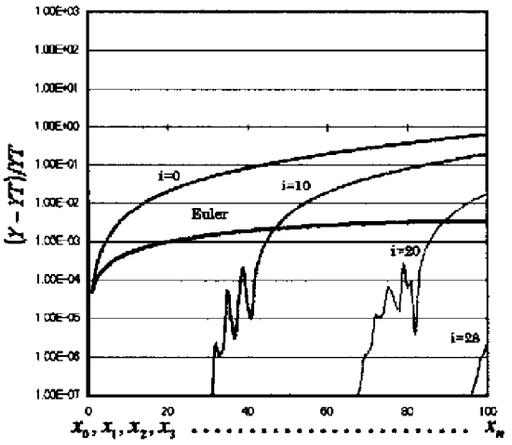


(d)Parallel Euler $\{y_n^{(20)}\}$

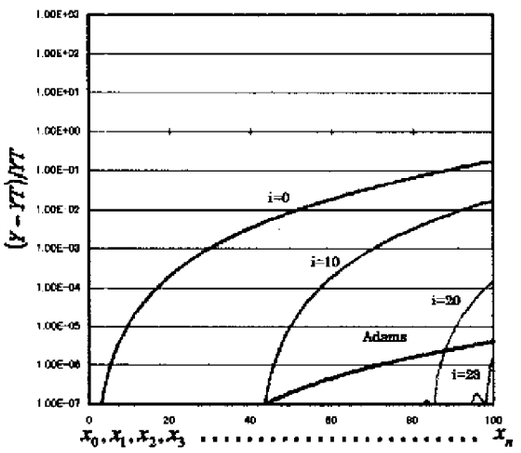


(e)Parallel Euler $\{y_n^{(28)}\}$

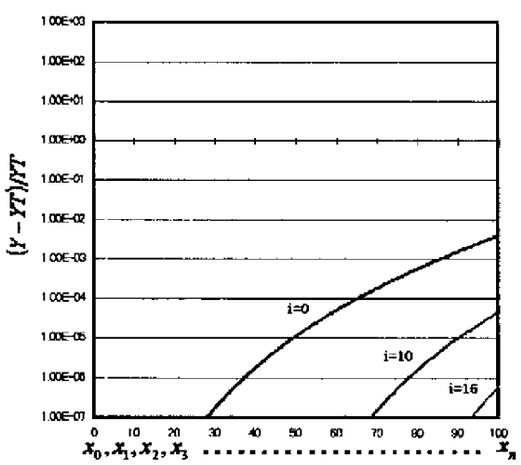
Fig.1 $y' = -\frac{1}{2y}, y(x) = sqr(1-x), y(0) = 1, h = 0.01, N = 99$ の各種数値解に対する区間相対誤差分布



(a)Parallel Euler



(b)Parallel Adams

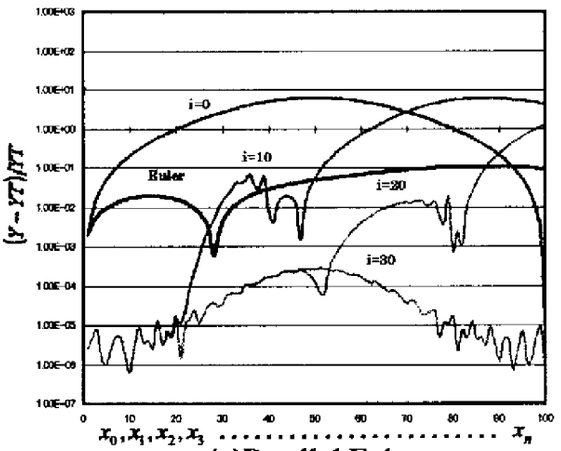


(c)Parallel Runge-Kutta

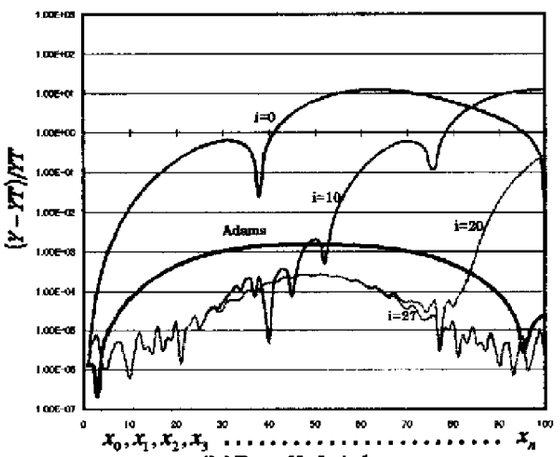
Fig. 2

$y' = -xy, y(x) = \exp(-x^2/2), y(0) = 1, h = 0.01, M = 3, K = 2$

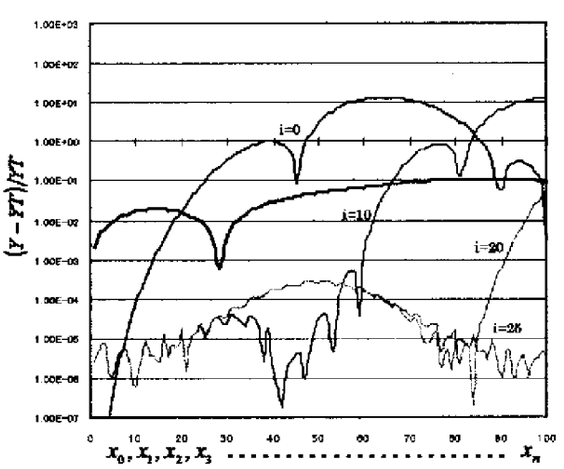
に対する空間相対誤差分布



(a)Parallel Euler



(b)Parallel Adams



(c)Parallel Runge-Kutta

Fig. 3

$y' = -y\sin x, y(x) = \exp(\cos x - 1),$

$y(0) = 1, h = 0.0628, M = 3, K = 2$

に対する空間相対誤差分布

4.1 区間相対誤差分布

図1は、$y'=-\frac{1}{2y}, y(0)=1$ (Euler 法) に対する数値解の区間相対誤差分布である。

図(a)は、従来の Euler 法に対する相対誤差分布である。

図(b)は、並列 Euler 法の初期近似解$\{y_n^{(0)}\}$の相対誤差分布である。

図(c)は、並列 Euler 法の第 10 近似解の相対誤差分布である。

図(d)は、並列 Euler 法の第 20 近似解の相対誤差分布である。

図(e)は、並列 Euler 法の収束後の相対誤差分布である。

従来の Euler 法では、10^{-2}～10^{-4}の範囲に相対誤差が多く分布している。$\{y_n^{(0)}\}$に関しては、10^{0}～10^{-2}の範囲に相対誤差があり、精度がかなり悪いが (24)・(26) 式の逐次近似法を用いると、$\{y_n^{(10)}\}$で10^{-1}～10^{-6}、$\{y_n^{(20)}\}$で10^{-2}～10^{-6}の範囲になり、$\{y_n^{(28)}\}$で収束し、10^{-6}に相対誤差が集中しているのがわかる。

4.2 空間相対誤差分布

次に並列 Euler 法と並列 Adams 法・並列 Runge-Kutta 法による収束の違いについて調べる。

図2は、$y'=-xy, y(0)=1$に対する各分点での相対誤差を表した空間相対誤差分布である。図6より各種解法による収束の違いについて述べる。

並列 Euler 法では、$i=28$で全点において10^{-6}以下、並列改良 Euler 法 (並列 Adams 法) では$i=23$で10^{-6}以下、並列 Runge-Kutta 法では、$i=16$で10^{-7}の分布を示す。

従来の方法と比べ、初期近似解はかなり悪いのが見てわかる。しかし、逐次近似法を用い、収束まで繰り返すことにより、相対誤差分布が10^{-6}以下に集中し、かなりの精度を得る事がわかる。

この例から見ると、並列 Euler 法と比べ、並列改良 Euler 法 (並列 Adams 法) で 1．2 倍、並列 Runge-Kutta 法では 1．8 倍早く計算できる事がわかる．これは、並列改良 Euler 法 (並列 Adams 法)、並列 Runge-Kutta 法の初期近似解に対する相対誤差が良いことに起因していると考えられる。

図3は、$y'=-y\sin x, y(0)=1$に対する各分点での相対誤差を表した空間相対誤差分布である。図3より各種解法による収束の違いについて述べる。

並列 Euler 法、並列改良 Euler 法 (並列 Adams 法)、並列 Runge-Kutta 法ともに$\{y_n^{(0)}\}$にあまり違いが見られない．このような場合には並列逐次近似回数も両者あまり違わなくなる。

また、単純には相対誤差分布は右上がりの曲線と期待されるが、この例では、相対誤差が複雑な分布を示し、現在のところその理由は明らかではない。

5. 各並列アルゴリズムの処理速度

シミュレーションで得られた処理速度を表1に示す。

Table.1 各種シミュレーションによる並列アルゴリズムの処理

初期値問題					処理速度		
$f(x,y)$	Initial value	$y(x)$	h	N	Parallel Euler	Parallel Adams	Parallel Runge
$y'=-xy$	$y(0)=1$	$\exp(-x^2/2)$	0.01	100	7.1	8.5	12.2
$y'=-1/2$	$y(0)=1$	$sqrt(1-x)$	0.01	99	2.4	2.6	5.8
$y'=-y\sin x$	$y(0)=1$	$\exp(\cos x-1)$	$2\pi/100$	100	6.8	7.2	7.5

実際の実行速度というのは、関数$f(x,y)$の計算処理やオーバーヘッドによる遅延時間など様々な要因を含んでるが、実行速度の大部分が関数$f(x,y)$の計算処理の時間を占めるために、簡単のために、従来の逐次処理計算で引用される$f(x,y)$の回数と、並列処理で1つのプロセッサにおいて引用される関数$f(x,y)$の回数の比を実行速度として処理速度を求めた。ここでは、逐次処理での誤差が10^{-6}以下に収束するまでの逐次処理の関数$f(x,y)$の計算回数と，並列処理での関数$f(x,y)$の計算回数の比率を求める事で何倍早くなったか求める事ができる。

表に3つの例題について各種解法（並列Euler法・並列改良Euler法（並列Adams法）・並列Runge-Kutta法）での処理速度を示した。この表からわかるように、一般に、並列Euler法、並列改良Euler法（並列Adams法)、並列Runge-Kutta法となるにつれて、処理速度が早くなるのがわかる。従来のRunge-Kutta法に比べると約6～12倍の処理速度が得られる事がわかる。

6. PVM

実際の並列計算機を用いて複区分予測子法等を用いた並列アルゴリズムを実行すべきであるが、あいにく、並列計算機は高価で、すぐ利用できない状態にある。そこで、仮想的な並列計算機を作り出すことが出来るPVMを利用しようと考えた。

PVMとはParallel Virtual Machineの略であり、アメリカのオークリッジ国立研究所(Oak Ridge National Laboratory)と、エモリー大学(Emory University)らの研究者らによる異機種分散計算の研究プロジェクトから生まれた並列プログラミングライブラリであり、世界中にかなりのユーザーがいる。

このPVMは、Web上でフリーウェアとして公開されており、LinuxやWindows上で動作させることが出来る。ネットワーク(LAN)で接続された複数の計算機を、あたかも1台の並列計算機のように利用することができるので、低コストで並列計算機を構築することが出来る。

図4に、PVMによる並列計算機の論理的構成図を示す。

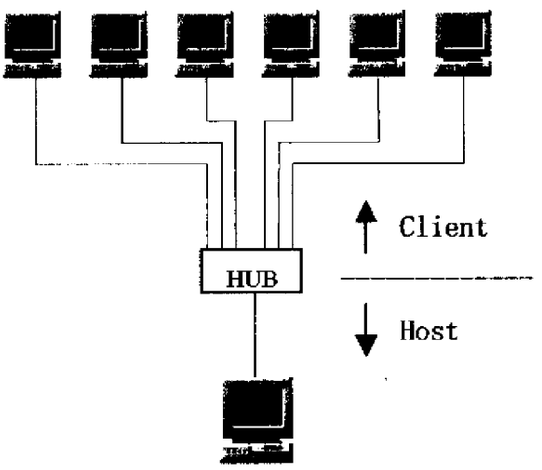


Fig.4 PVMによる並列計算機の論理的構成図

現段階では、PVMをLinuxにインストールして動作確認を行った段階であり、まだまだ、使いこなすレベルではないが、これからはこのPVMを使用して並列アルゴリズムの検証を行っていく予定である。

7．まとめ

VLSI技術の発展により、同一プロセッサを多数用いる並列処理計算機システムは高速演算が必要とされている科学数値計算の分野で大いに期待されている。本報告では、予測子法と逐次計算手法に基づいた常微分方程式の並列計算アルゴリズムを提案し、各種のシミュレーションを行い、この並列計算アルゴリズムの検討を行った。その結果、この並列計算アルゴリズムは従来の計算アルゴリズムに比べかなり

の高速化が実現されることと、数値解の精度も比較的高いことがわかった。今後は、PVM を使用して、シミュレーションで得られた処理速度が実際に正しいかどうか、検証していく予定である。また、数値計算における誤差分布の振る舞いは複雑であり、誤差伝播に関するより正確な解析は今後の問題である。

参考文献

1）奈良久・早川美徳・阿部亨：数値計算法，朝倉書店（1991）

2）玄光男：数値計算とデータ構造，共立出版株式会社（1994）

3）小沢一文：数値計算法，共立出版株式会社（1987）

4）湯淺太一・安村通晃・中田登志之：初めての並列プログラミング（1999）