非粘性バーガース方程式の数値計算の PVM による分散処理について

平成 14 年 2 月 15 日

情報電子工学科 竹野研究室
藤井 学

目 次

概要

コンピュータ同士を接続する際の方式、手段にはどのようなものがあるのか。また、その通信手段がどのようなものか。それぞれにどのような特徴があるのかを自らの使用した一般に使われるスター型を含め、述べた。その上で、接続したコンピュータ同士を分散処理させる為に必要なソフトウェアとして、比較的に手に入り易く、様々なコンピュータに使用可能な PVM というものを挙げ、さらに、その特徴とそれを利用する事で非粘性バーガース方程式の差分化した式を分散化させて計算させ、どのくらいのマシン数や仕事の数ならば良いかを検討した。更に、結果よりマシンが多ければ良いという訳ではなく、バランスが重要であると分かった。よって、そのバランスをとる事やその他の改善点を今後の課題として挙げた。

1　はじめに

数値計算では多くの場合、計算に時間がかかる。そのような場合に分散処理を使用する事で計算時間を短縮できれば良い。コンピュータをもし複数持っている場合、それらを接続する事はできるのか。どのようなマシンが接続できるのか。

本稿ではコンピュータを複数個接続したい場合に、どのような特徴があるのか、実際に接続してみた場合のものと、理論のみのものについて併せて考察する。

また、実際に接続し、同時に使用して分散処理をさせる場合に、どのような手段があるのかを、分散化ソフトウェアと使用した上で確かめる。

また、それを使用し保存則方程式を分散化すると、どのようなアルゴリズムになり、どのようなプログラムになるかを述べ、解決方法を模索する。

2　コンピュータの接続

2.1　コンピュータ同士の接続網について

並列処理コンピュータの接続網についての細かい方式を説明する。今回、ここに挙げる方式は実際に使用するものではないが、(実際に使用する方式は 2.2 でとり挙げる) その特徴を示す。これらは、仲介になるものを介さずに接続する方法を挙げる。

完全結合網
ライン、リング
メッシュ
ツリー
ハイパーキューブ

等があり。特に考える必要はないと思うが、ノード (コンピュータ) 同士の最も遠い接続同士の距離を直径と呼ぶ事にする。通信距離に制限がある場合に有効であると考えられる。

2.1.1　完全結合網

Fig,2.1 より、これは、各ノードが他の全ての node に対してリンクを持っており、n 台のノードがあるとすれば、他の $n-1$ のノードに $n-1$ 本のリンクを持っている。

よって全体で $\frac{n(n-1)}{2}$ 本のリンクを持っており、これは n が増えれば増える程、累乗される為膨大な数字になり現実ではそんなにケーブルもコンピュータに端子も存在しないので n が小さい時にしか実行出来ない。しかし、これが接続網で最も理想だと考えられる。

2.1.2　ライン、リング

ラインは Fig,2.2 のように一列に並べたノード同士を一つずつ繋いでいく。更に両端を繋げばリングになる。

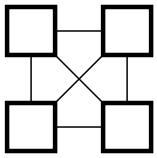

Fig. 2.1 完全結合網

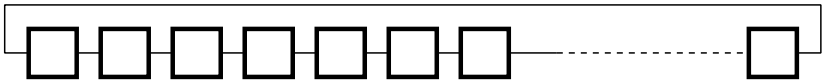

Fig. 2.2 ライン、リング

よって直径はラインでは $n-1$ でリングでは双方向より通信できる為 $\frac{n}{2}$ となる。

2.1.3 メッシュ

Fig,2.3 のように二次元平面にノードを敷き詰め、四方向にリンクを延ばした形がメッシュである。

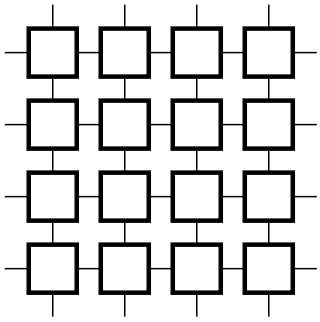

Fig. 2.3 メッシュ

$\sqrt{n} \times \sqrt{n}$ のメッシュの直径は横に $(\sqrt{n}-1)$ 縦に $(\sqrt{n}-1)$ 進むので $2(\sqrt{n}-1)$ となる。リンクは縦横反対側に接続してもよいので、そうすると全てのノードに 4 本ずつのリンクができる為、直径は $2n$ となる。これはトーラスとも呼ばれる。

2.1.4 ツリー

Fig,2.4 のように二分木結合網がツリーである。各ノードはその下のノードに 2 つのリンクを持つので結合網は最初のノードから扇状に広がっていく。

最初のノードを 0 段目として一段ずつ倍になっていくので j 段目では 2^j のノードがあ

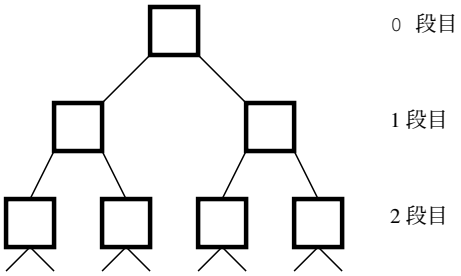

Fig. 2.4 ツリー

る。よって、全体では自分の数えている段の上には必ず $2^j - 1$ 台あるので $n = 2^{j-1} - 1$ となる。そして直径は最初の段のノードから別方向に降りていった一番奥のノード同士で、j 段には j 本のリンクがあるので $2j$ となる。

2.1.5 ハイパーキューブ

Fig,2.5 のように立方体の形をとっているのが三次元ハイパーキューブである。

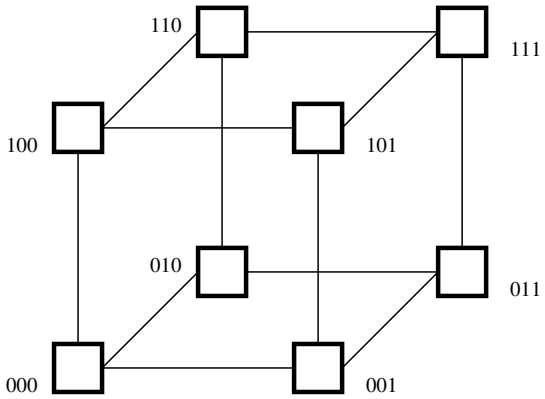

Fig. 2.5 ハイパーキューブ

これは三次元のものに限らず、何次元でも存在する。が、実際繋ごうとすると五次元程度から思考する事が困難になってくる。例に四次元ハイパーキューブを挙げる。

これを解決する方法としてノードことに割り当てられるアドレスを d 次元では d ビットの二進数のアドレスに割り当てていくと、例えば三次元で 000 は隣り合う 001 010 100 に接続し 111 は 011 101 110 に接続される。よって 1 ビットだけ違うアドレスを持つ node に接続すればよいわけである。すると、五次元では 11111 というノードに 11110 11101 11011 10111 01111 というノードを接続すればよい事になり、接続も容易になる。

ハイパーキューブは直径が $\log_2 n$ となる。

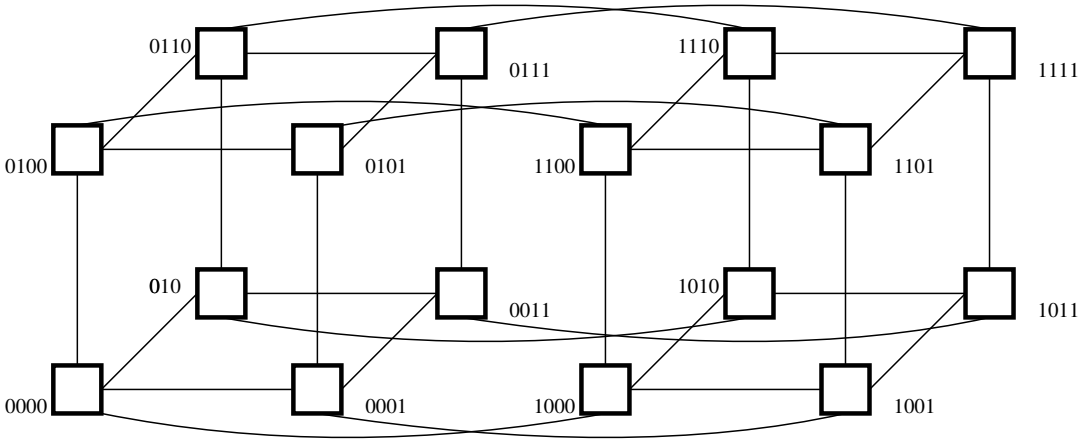

Fig. 2.6 四次元ハイパーキューブ

2.1.6 考察

これらの事から、あらゆる面で完全結合がもちろんよいが、リンク数のみならライン型が適している。リンクを一つ増やすだけでリングになる為、思い切ってラインにすることもよい。ノードごとに必要な端子の数も 2 つと、少なくて済む。しかし、通信が他のどれよりも遅くなる為、コストの面でしか利点は無い。直径で考えるならば、ハイパーキューブ型が有利である。

2.2 LAN の接続方式について

コンピュータ同士を LAN で接続する場合、

バス型
リング型
スター型

との方式がある。今回はスター型を使うがその他の方式との比較をする。

2.2.1 バス型 LAN

バス型はバスと呼ばれる 1 本のケーブルに端末を接続する方式で、ケーブルの端には終端抵抗が取り付けてあり、信号が反射して雑音になるのを防いでいる。

バス型は、一本の伝送路を中心とした直線的な形状の配線形態であり、これに情報処理機器をぶら下げる配線形態 (Fig,2.7) である。(図の node はそれぞれノード一つ一つを指す)

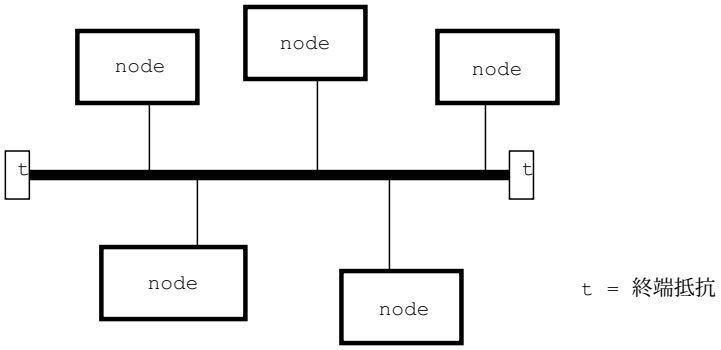

Fig. 2.7 バス型 LAN

以前はこのバス型が主流だったが、現在はコストやレイアウトの自由度の制約から、スター型に主役の座を受け渡した形になっている。しかし、この方式を利用するケーブルはノイズに強いため、フロア間等を接続する際に使用されている。

2.2.2 リング LAN

先程と同じバスと呼ばれる環状の 1 本のケーブルに端末を接続する方式で、他の方式に比べケーブルの総延長を長くすることが容易で、広域ネットワークにもリング型の接続形態のものがある。

リング型はケーブルでループを形成し、そのリング上に機器を接続する。

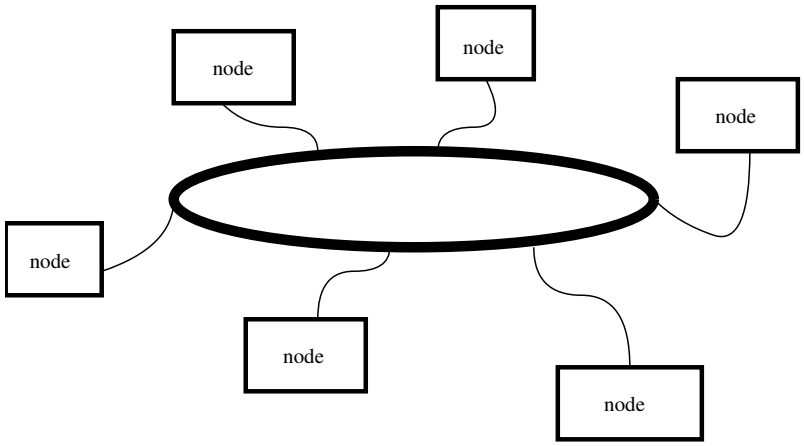

Fig. 2.8 リング型 LAN

トークンと呼ばれる論理的なデータを循環させることにより機器同士の通信を行なっており、データのぶつかり合いがないため高速な通信が可能である。

2.2.3 スター型 LAN

スター型は中心となる通信機器を介して端末を相互に接続する方式で他の接続方式よりも配線の自由度が高いのが特長である。

スター型は、HUB という集線装置を中心に、コンピュータを放射線状に配線する。形状が星状のためスター型と呼ぶ。

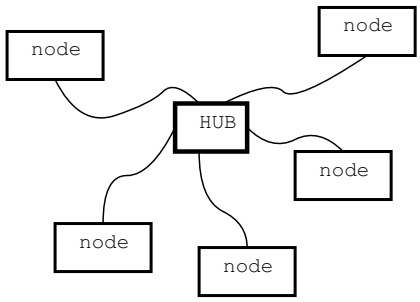

Fig. 2.9 スター型 LAN

設置のための工事および設置後の変更が最も容易な形式である。障害に対しても、ケーブルの断線で影響を受けるのは、そのケーブルに配線されていた機器のみであり、ノードごとに問題のある個所を切り分けることが可能である。

ハブは最大で直線で繋いだ場合 4 台であるが、それはこれ以上では信号がノイズに埋もれてしまうからである。よって、

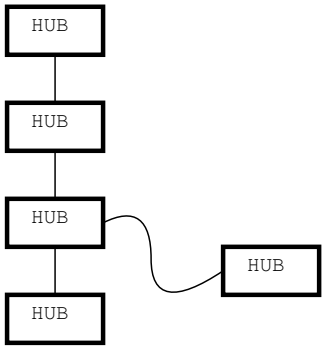

Fig. 2.10 スター型 LAN

Fib,2.10 のような接続も可能である。HUB の通信方式には CSMA/CD という方式が使用されている。これは、LAN で利用される通信方式の一つで、データを送信したいノードはケーブルの通信状況を監視し、ケーブルが空くと送信を開始する。このとき、もし複数のノードが同時に送信を開始するとケーブル内でデータが衝突するので両者は送信を中止し、ランダムな時間待って送信を再開する。この方法に従うと、1 本のケーブルを複数のノードが共有して、互いに通信することができる。

CSMA/CD 方式の場合、衝突の発生による伝送効率はネットワークの使用率に左右される。端末の数が多ければ多いほど衝突の確立は高くなり、効率が低下してしまう。

3　計算の分散と、命令方式

3.1　計算の分散方法の簡単な例

最も分散計算がしやすいのは、独立した部分部分が他の計算結果を必要としない場合である。よって今回はそれを使用するが、具体的に言うと、

例えば積分の計算が挙げられる。これは区間の x 軸とグラフの間の面積を求めるものなので区間をノードの数に分割し、それぞれが与えられた区間の y 軸を高さとした長方形の面積を求め、最後にその総和を出せばよい。

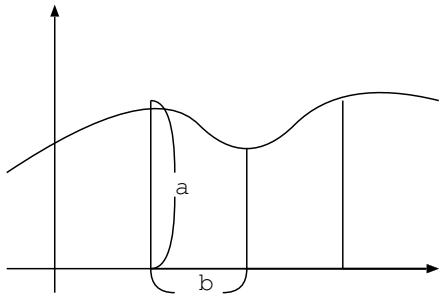

Fig. 3.1　高さを a、幅を b とおいてみる

これは、最初に親が作業する数にそれぞれの区間を分けるという作業を行ない、これと元々の関数を各ノードに送る。各ノードは受け取った部分のみを計算する為、他のノードの計算結果を必要としない。よって、それぞれが独立するので分散化は最も容易にできる。

よって今回も基本的にはこのように独立した分散を行うが、隣同士のノードでその境界にあるデータをやりとりしなくてはならず、その方法は第 5 章で説明する。

3.2　命令方式

分散処理をする命令とデータの状態によって、その方式が分かれる。それぞれ SIMD 方式と MIMD 方式と呼び、今回使用するのは SIMD 方式である。こちらの方が、PVM のプログラムを書く事に適している。何故なら、元々、PVM ではノードの数を意識せずにプログラムを組む事が出来るからである。((4.1) 参照)

3.2.1　SIMD 方式

(single instruction stream multiple data stream) で単一命令ストリーム複数データストリーム。これは命令が複数のプロセッサに送られ、それぞれのプロセッサはそれぞれの

データに対して命令を実行する。それぞれのデータは親プログラムの中で指定し、命令と同時に送信する。

具体的には、例えば $a * b$ という計算があるとする。b にはデータを入力するものとして、この b は親が持っている多数の配列であるデータ c を、親がある程度の数に分割する。

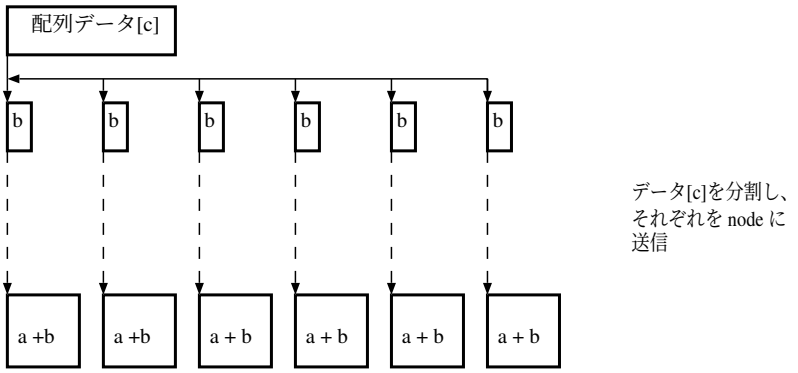

Fig. 3.2 SIMD 方式

(Fig,3.2) では受け取ったデータをそれぞれの node は b として $a + b$ を実行している。子は送られて来たデータに対して同じ命令を実行するだけである。

3.2.2 MIMD 方式

(multiple instruction stream multiple data stream) で複数命令ストリーム複数データストリーム。これは各プロセッサごとにそれぞれのプログラムを持ち、それぞれのデータに実行される。

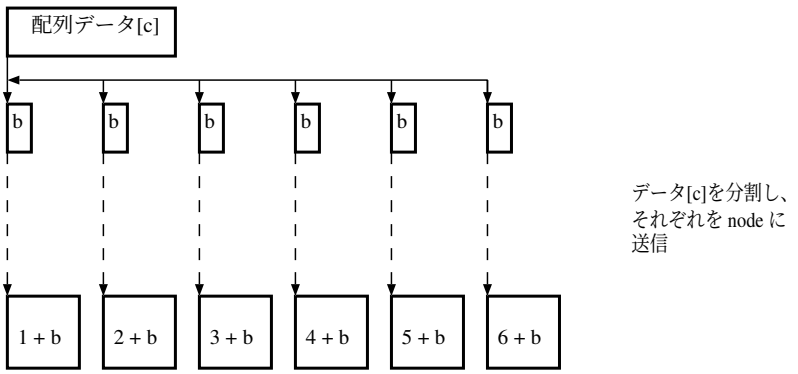

Fig. 3.3 受け取ったデータをそれぞれの node は b として、更にそれぞれの命令を実行

(Fig3.3) では受け取ったデータをそれぞれの node は b として、更にそれぞれの命令を実行している。よって、このそれぞれ違う命令を指定する分、手間がかかる。だが、これ

により SIMD よりも自由度が増す。しかし、各プロセッサに対してプログラマが命令を配分していかなければならず、プロセッサ同士の同期や協調動作をいかに行なわせるかはプログラマーに委ねられる。

4 PVM

では実際の分散処理を請け負うソフトウェア (以後 PVM) の説明をする。実際に PVM はネットワークの接続が完了した後に走らせるプログラムである。

4.1 PVM の概要

PVM((Parallel Virtual Machine) 仮想並列計算機) は、ネットワークで接続された異機種のコンピュータを接続し、メッセージパッシングによって、並列計算を行うためのソフトウエアであり、動作するマ シンの種類が多いこと、ftp や電子メールで比較的容易に入手できることもあって、広く利用されるようになってきている。複数マシンによる並列処理を実現でき、逐次コンピュータ、ベクトルコンピュータ、並列コンピュータを使用できる。アプリケーション・マシン及びネットワーク・レベルでの異機種間利用を実現、標準使用のメッセージ通信方式を使用でき、整数及び浮動小数点数の違いを吸収する、という特徴がある。

使用するには、PVM を使用したい全てのコンピュータに PVM をインストールし、その全てのコンピュータで PVM を起動させるが、起動は一台のコンピュータで PVM を起動する際にオプションで全てのコンピュータの PVM を一度に起動できる。

プログラムは、直接命令を実行させるコンピュータ (親) に全てのコンピュータを制御するプログラムを置く、そして残りのコンピュータ (子) に実行したいプログラムを置いておき、親が実行させる事で並列処理をする。

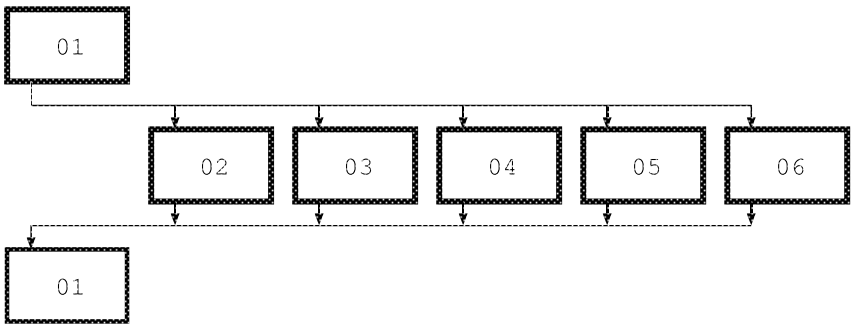

Fig. 4.1 01 を親として、02 から 06 までを子とした場合

4.2 具体的な PVM の例

具体的にどのように使うかは簡単なプログラムを追いながら説明したい。

PVM をインストールした際に付属してきた、PVM ノードのタスク ID を表示する hello プログラムを実行した。hello は親と子を nolm01 と nolm03 とするとそれぞれ、

```
hello
```

と

```
hello_other.
```

をインストールしておく。そして、親より実行すると、

```
nolm01 1% hello
i'm t40003
from tc0001: hello, world from nolm03
nolm01 2%
```

と表示されて終る。この経過がどうなっているのか文末に親、子の同時進行フローチャートを示す。ここではその部分部分を追って検討していく。先頭より親プログラムを見ると、(太枠は通信発生)

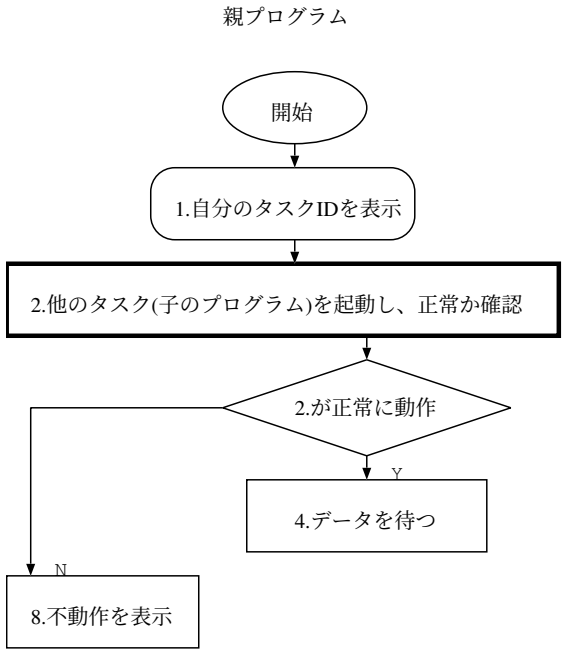

Fig. 4.2 通信が起き、正常か判別するまでの親プログラム

(Fig,(4.2)) これを見ていくと、始めにタスク ID を表示し、2 で通信が発生し、他のタスクを起動する。(タスク ID= 仕事別の識別番号) 他のタスクを起動する際に起動するファイル名を指定する。他タスクが正常に起動した場合はそのままデータの受信を待つが、そうでない場合は不動作を出力する。

そして、子のプログラムが起動すると、

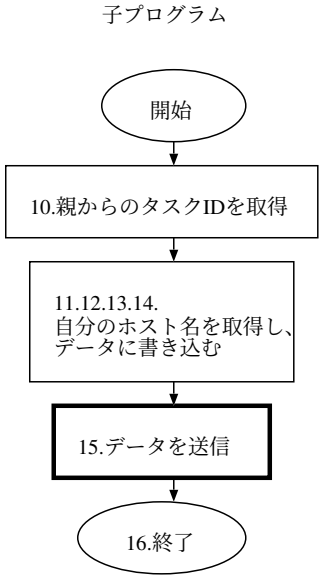

Fig. 4.3 子がデータを受取り、仕事をして送信する。

(Fig,(4.3))10 で親プログラムからのタスク ID を受取り、自分のホスト名を取得したら、それをデータに書き込み、再び送信する。送信が終了したら、プログラムを終了する。

そして再び親プログラムに戻り、

(Fig,(4.4))4 でデータを待っていた状態からデータを受け取る。そして、データを表示し終了する。

ある一つのプログラムを勝手に PVM が分散処理してくれる訳ではなく、プログラムにライブラリを指定して PVM のコマンドを使用し、自分で分散を指定しなければならない。が、PVM では使用するマシンの指定を必要としない為、分散さえ指定すればどの子にどのようなプログラムを実行させるかをプログラマは意識しなくてよい。

実際に使うことができるのは「C 言語」と「ForTran」が使用できる。

実際のインストール作業も使用している OS が対応していれば容易であるが、そうでない場合、使用するコンパイラや PVM のパスの指定が必要である。

5 保存則方程式の差分

保存則方程式の一つの非粘性バーガース方程式の形を実際にプログラムに変換しやすい形式にしていきたい。

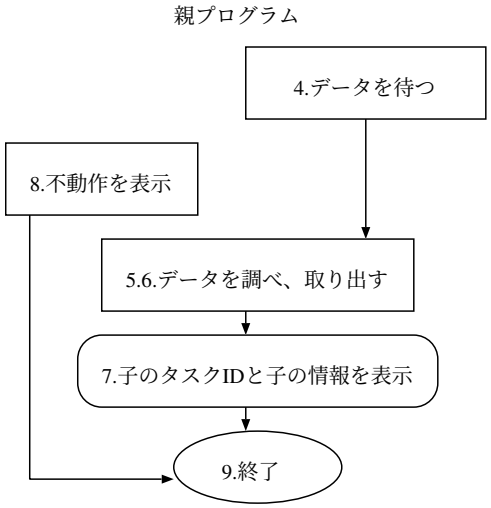

Fig. 4.4 返信データを受取り、子の情報を表示

5.1 Lax–Friedrichs 法

まず、保存則方程式を Lax–Friedrichs 法で解いていく。与えられた式

$$\begin{cases} u_t + f(u)_x = 0 \\ f(u) = \frac{u^2}{2} \end{cases} \tag{5.1}$$

これを

$$\begin{cases} i = t \text{ 軸の座標} \\ j = x \text{ 軸の座標} \end{cases} \tag{5.2}$$

として $A_{i+1,j}$ について差分化すると、

$$u_{i+1,j} = \frac{u_{i,j+1} + u_{i,j-1}}{2} - \frac{\Delta t}{2\Delta x}\{f(u_{i,j-1}) - (u_{i,j+1})\} \tag{5.3}$$

$$\tag{5.4}$$

となる。この (5.4) 式を実際にプログラムで使用するが、その時の初期条件を

$$\begin{cases} u(x,0) = \sin(\pi x) \qquad (0 \le x \le 1) \\ u(0,t) = u(1,t) \qquad (t \ge 0) \end{cases} \tag{5.5}$$

として考える。そして図示すると、

Fig.5.1 となる。ここでは一つ前と一つ後の値より答えを得る。(5.4) 式中の $A_{i,j+1}$ や $A_{i,j-1}$ がそれにあたるが、$x = 0$ と $x = 1$ の場合には、片隣しか値が存在しない為、Fig.5.1 中の $x_{-1} = x_3$、$x_5 = x_1$ と置き換える (文献番号 (1) 参照)。

また、同じく安定条件より (5.4) 式の $\frac{\Delta t}{4\Delta x}$ の部分を求めると、

$$\frac{\Delta t}{4\Delta x} = 0.25 \tag{5.6}$$

となり、それより、分割時間の Δt はそこより逆算できる。

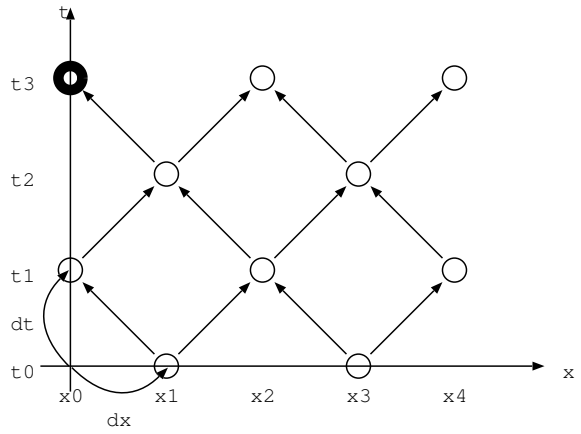

Fig. 5.1 Lax–Friedrics 法

5.2 分散プログラムの概要

この分散プログラム (フローチャート中の番号記載) と親、子の同時進行でのフローチャートを文末に示す。そして、その部分部分のフローチャートを追って見ていくと、(Fig,5.2 参照)

まず、子のプログラムを起動し、その後初期値を入力している。初期値はこの場合、$\sin(\pi x)$ でこの値より $t = 0$ の時の x 座標の場所を与えている。この時、同時に $\frac{\Delta t}{4\Delta x}$ の値も同時に送信する。こうする事でこの部分は子で計算させずに済む。

そしてノードごとのそれぞれの担当座標の部分の情報と、その隣り合う座標の交換条件も送信し、後は結果を待つ。

そして計算本体は子が行なう。まず、最初の (フローチャート番号 2) の通信で子が起動する。そして、(Fig,0022 参照)

子は起動したらまず、PVM のタスクになり、親より送られてきた諸データを受け取る (フローチャート番号 5 を 13 で)。そうして自らの配列にデータをコピーし、計算に移る。

式の計算は少し複雑になる。Lax–Friedrichs 法では次の時間の j_n を求めるには前の時間の j_{n-1} と j_{n+1} が必要であった。(Fig,5.1,5.4 式参照) 要するに、ある i の j の数は必ず分割数の半分しか入らない。よって、これをクリアするには 2 次元配列を偶数の部分のみを保持し、奇数ステップは保持しないようにする。そのようにして、

Fig,5.2 の変更前から変更後の様にデータを敷き詰める。そして、本来の $i+2$ を求める場合に必要な $i+1$ を中間ステップとして配列にして組み込んでしまい、直接 $i = n$ から $i = n+2$ を求められるようにする。

しかし、実際には順をおって計算しているので、座標的には Fig,/ref23 の変更前と同じ数の計算をしているが、そのまま座標に表してしまうと Fig,/ref23 の変更後の数字のままで結果が出てしまうので、Δt と Δx は半分の大きさしかない。よって、後に親でこれを正常な値に戻す計算をする。計算の部分の具体的なフローチャートは Fig,5.2 の様に

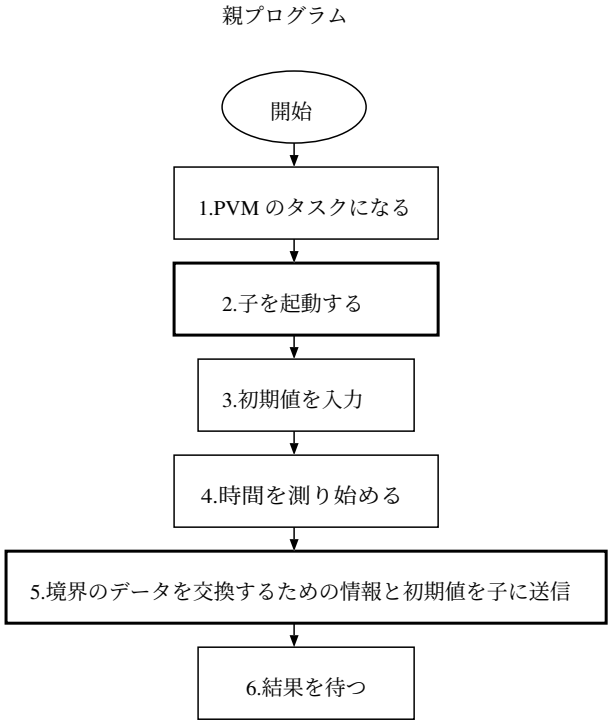

Fig. 5.2 子を起動し、必要なデータを送信するまで

なっている。

この 3 つの中間ステップの計算は

それぞれ Fig,5.2 の様になっている。$20,1$ は親で周期境界条件にしたので、2 段階踏んでも $x=0$ が求められるように本来の $x-2$ の部分 (勿論、詰めた座標上での x ー 1) を受け取っているので、そこより本来の $x-1$ を求めている。

$20,2$ では、元々の自分のデータより奇数ステップを求めている。

$20,3$ は $20,1$ と同じ理由で、x_{max+1} を求めている。($20,1$、$20,2$、$20,3$ の両端は $x=0$、$x=1$ ではないが、両端という事を分かり易く変更した)

そして、$20,1$ と $20,2$ と $20,3$ で求めた座標を元に $20,4$ で $i+1$(本来の $i+2$) を求める。(Fig,5.2 参照)

そしてようやく for の頭に戻り、境界の座標を交換し計算を繰り返す。

計算が終了した後、結果を親に送信し終了する。(Fig,5.2)

そして、親にプログラムが戻るが、(Fig,5.2)

親は子より送られてきたデータを受け取り、断片化しているデータを 1 つにまとめ、測定していた時間を止める。そうして使用時間を出す。そして、結果を出力するが、Δt は半分に詰めてあるので 2 倍して実際の時間にに戻す。

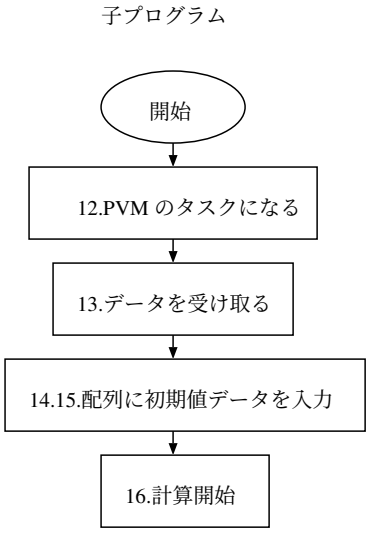

Fig. 5.3 子が起動し、親からのデータを入力するまで

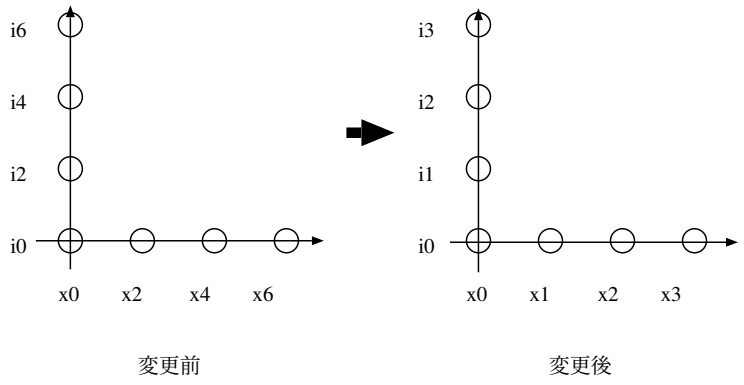

Fig. 5.4 座標を詰める

5.3 プログラムの検討

以上の方法で、保存則方程式の分散処理をしてみたが、このプログラムの問題点を検討したい。

まず、プログラムの実行結果を図示する。

Fig,5.3 より放物を描いた結果が出力された為、プログラム自体は正常に作動した。よってここには問題はないが、今回挙げた方法を実行すると、Δt を 1 回ずつ計算するごとに通信が発生する。どの場合でも、Δt を細かくすればする程、通信が遅くなり、処理速度はどんどん遅くなっていく。よってこの解決方法を検討しなくてはならない。

なるべく通信を少なくする方法として、考察中の方法を 1 つ挙げると、まず、今回は 5 つのノードを使用するとして、x 軸を 1000 分割するとする。1 つ目のノードに 0　200、

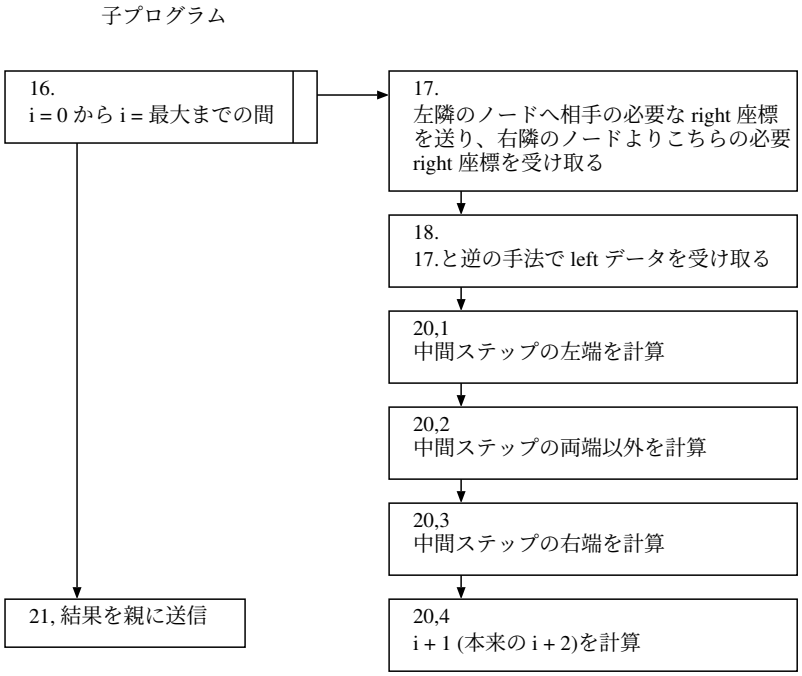

Fig. 5.5 計算の本体

2 つ目に 200　400、3 つ目に 400　600、4 つ目に 600　800、5 つ目に 800　0 番目のデータを送る (両端でデータは同じものを与えられている)。

この状態で通信をせずに、既知のデータのみで計算を実行していくとそれぞれが、三角形になると思われる。(Fig,5.3)

この三角形の辺 (底辺を除く) を形成している座標を今度は、三角形の時間軸方向に伸びている頂点がもし、$x = 100$、300、500、700、900、の部分だとしたら、その軸でノードを分割する。(もし、900 1100 $= -100$ 100 と置くことが出来ればノードの数はそのままだが、置けなければノードが $+1$ 台必要) そこに、各ノードが接している辺の座標を送ることで、今度は逆三角形を計算して、三角形の時間軸方向に伸びている頂点の i までは、合計 2 回の通信で済む。(Fig,5.3)

ここまでくると、この i の j 座標は全て求まっているので、再び今の手順を繰り返すことで、かなりの通信時間の削減ができると思われる。

しかし子の方法では今回使用した保存則方程式の左辺 $(A_{i+1,j})$ の部分にも何らかの三角形の出力を格納できる作業をしなくてはならないかと思う。さらにノードの分割をずらす事ができるのかが、どうなのかは考えがついてゆかなく、まだまだ問題点も考察していかなくてはならないが、案の一つとして挙げる。

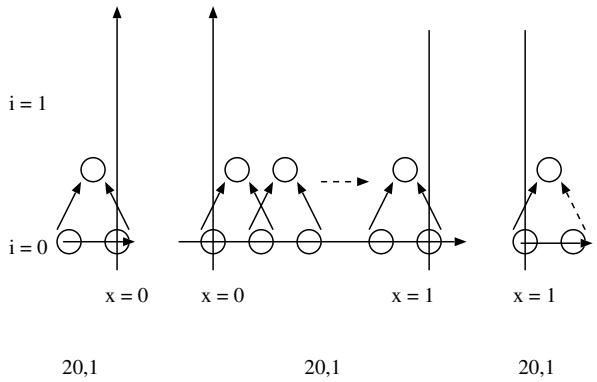

Fig. 5.6 計算の本体、第 1 ステップ目

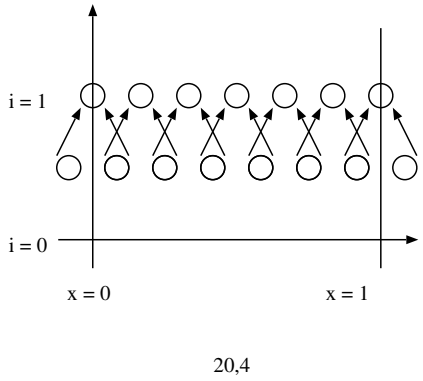

Fig. 5.7 計算の本体、第 2 ステップ目

6 マシン数、タスク数、x 軸の分割数の比較

実際にどのくらいのマシン数やタスク数、x 軸の分割が適切か、それぞれに結果を出し、考察する。

まず、コンピュータを 1 台使用し、その中でタスクを 1 つから 10 まで増やした場合の結果を示す。(Fig,6)

このグラフでは下から順番にタスクの少ない数から多くなっている。タスクが一つでは 10000 分割でも 75 秒しかかかっていないが、タスクの数が 10 では、225 秒になっている。タスクが多ければ多い程、処理が増えている為に大きな時間の差ができてしまったと考えられる。これは一つのマシンの中でも、実際には別タスク同士が PVM による通信をしている為である。その分、PVM のプログラムの割合が増え、時間がかかっている。よって、マシン一つでは一つのタスクが最適である。

次にマシンを複数にした場合の関係を考察する。まず、同じスペックを持つマシンを親を一台、子を四台でデータをとった。タスクはそれ以上の数で試したものの、前の結果よ

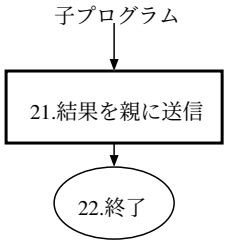

Fig. 5.8 子のプログラムの終了

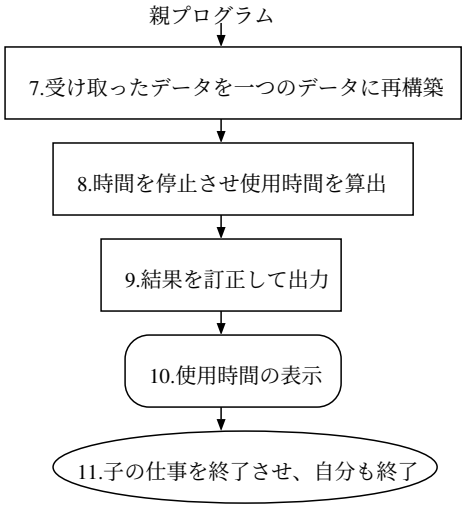

Fig. 5.9 子のプログラムの終了

り、マシン一つに対しタスク一つが最速となっている為、その結果のみ示す。(Fig,6)

この場合、データが均衡しており余り差は見えない。そこで最も速いものと、マシン一台でのものとを比較する。(Fig,6)

ここではマシン一台と二台に対してのグラフを示す。これは 7000 分割まではマシン一台の方が速いが、マシン二台では 8000 分割以降が速くなった。マシン三台と四台がそれよりも遅いのはバランスの問題であると考えられる。一つのマシンに対して計算させる量が減ることは良い事ではあるが、その分、通信が増えてはそれが時間をとってしまう。よって、その二つが適度につりあって処理は速くなるといえる。これは様々な環境で常に変化する。計算量が増えたとしても、マシン一台一台自体が処理速度が速ければ時間がかかるわけでもない。もしくは通信が効率良く行なわれれば、通信が増えても構わない。元々、スター型では通信効率が非常に悪い為、今回の結果で時間に余り変化がないのは、これから別な通信方式を使用する事で分散処理は速くなるといえる。

しかし、その分通信用のプログラムが減ると言う訳でもないので、マシンを二台使用すれば二倍の速さになる。という事はないと想われる。よって、今回の結果は他の状況にあ

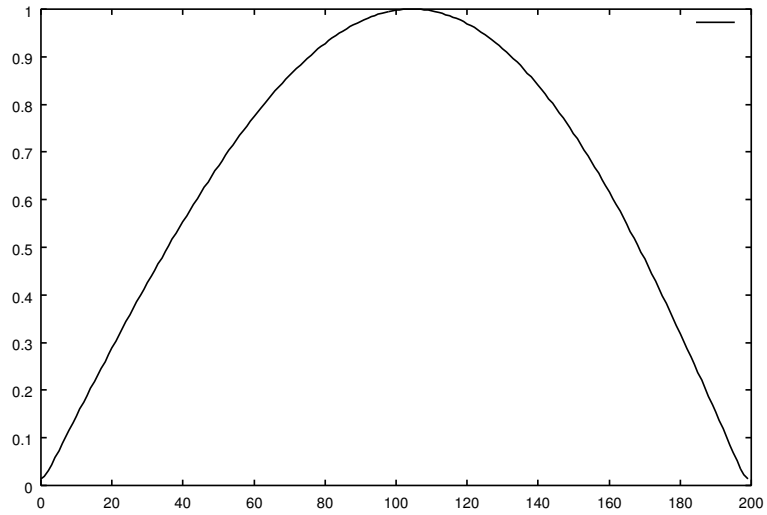

Fig. 5.10 出力結果

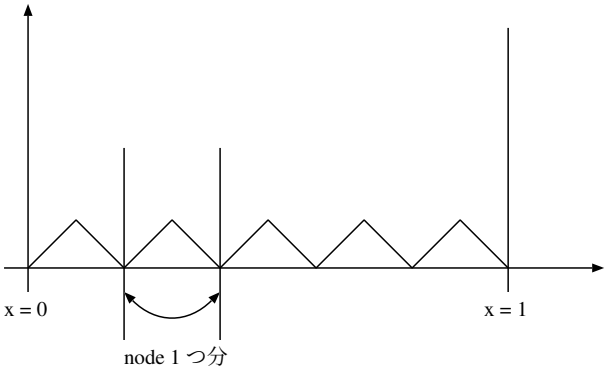

Fig. 5.11 三角形ができる予想図

てはめる事ができないが、このような改善点を見ていく事で分散処理は効果があると分かった。

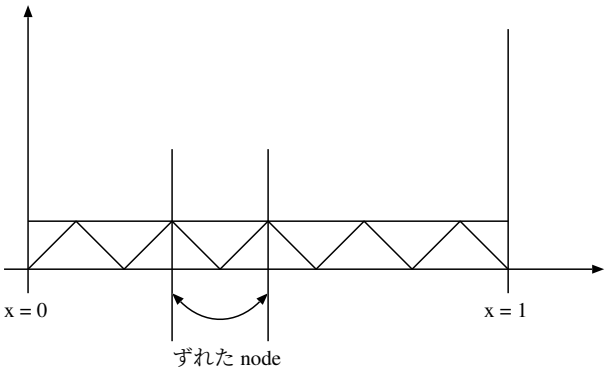

Fig. 5.12 逆三角形を計算した予想図

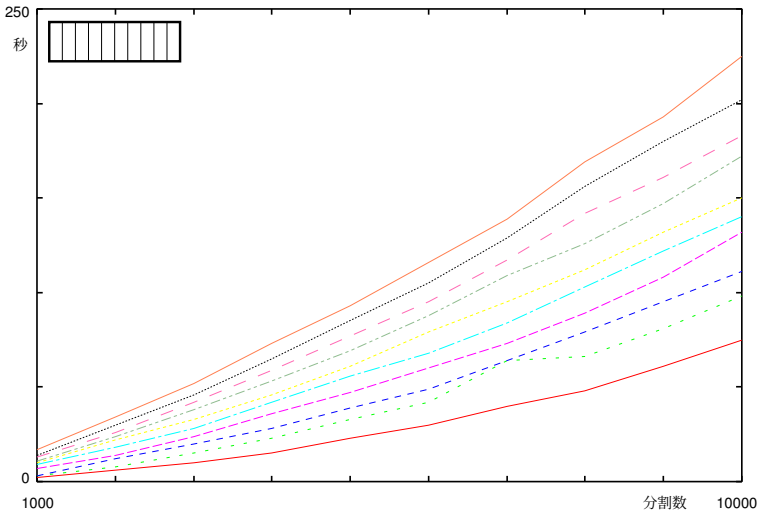

Fig. 6.1 一台の中でタスク 1 から 10

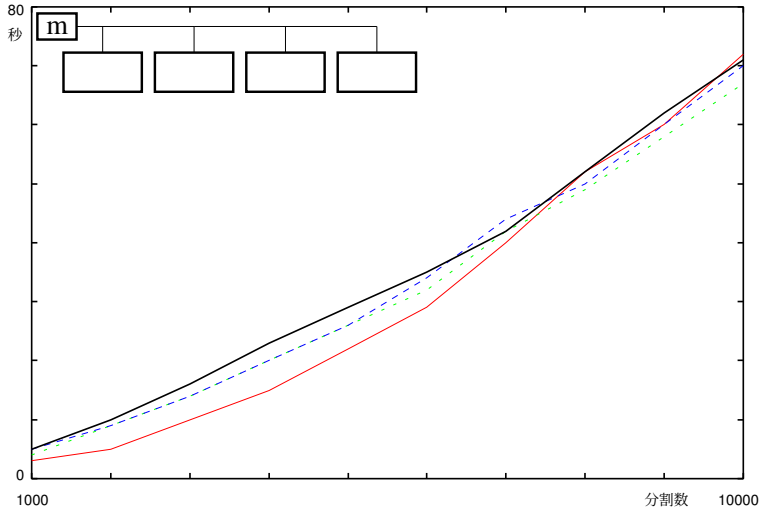

Fig. 6.2 マシン一台に対してタスク一つ

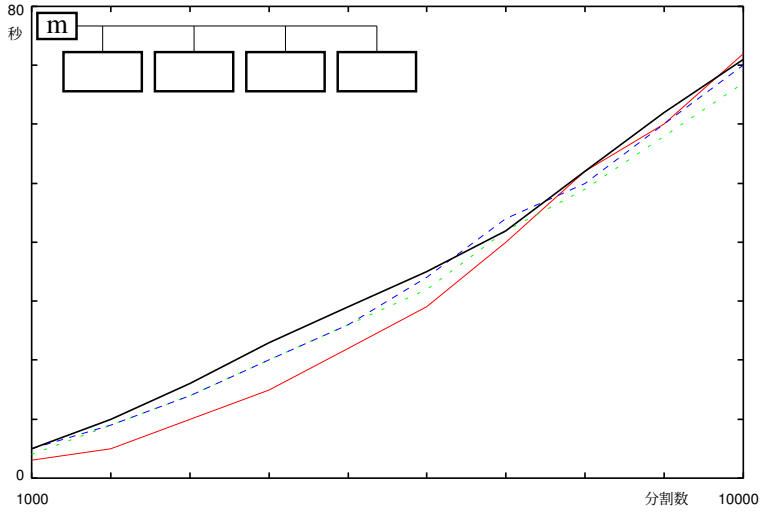

Fig. 6.3 マシン一台と二台

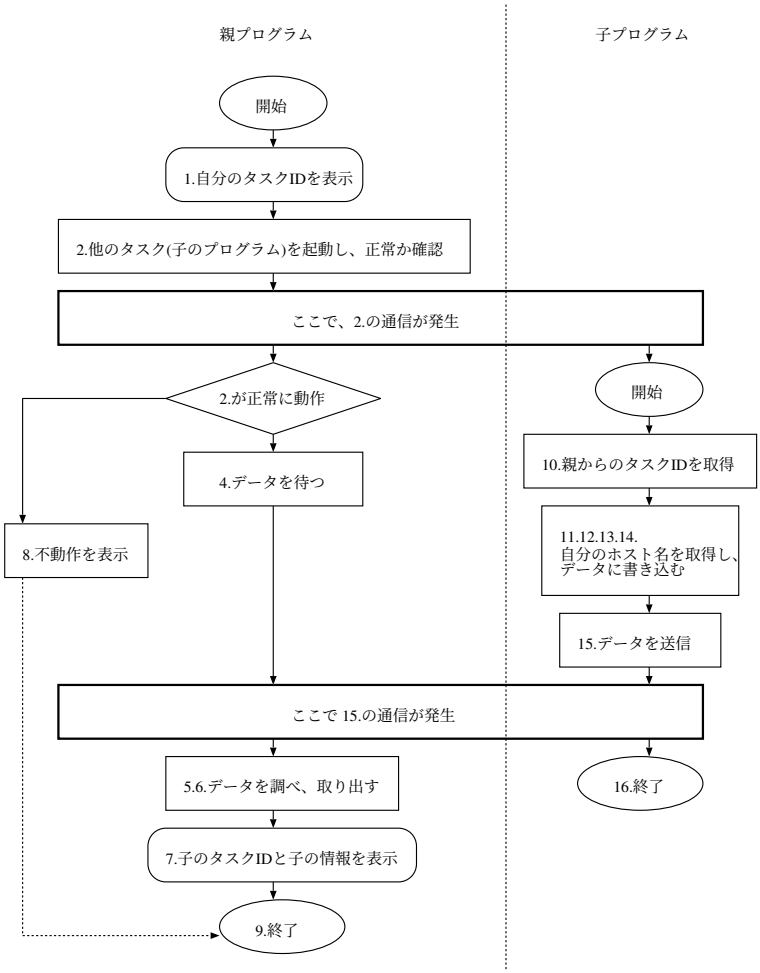

Fig. 6.4 簡単な PVM のプログラムの親と子を同時に追う場合のフローチャート

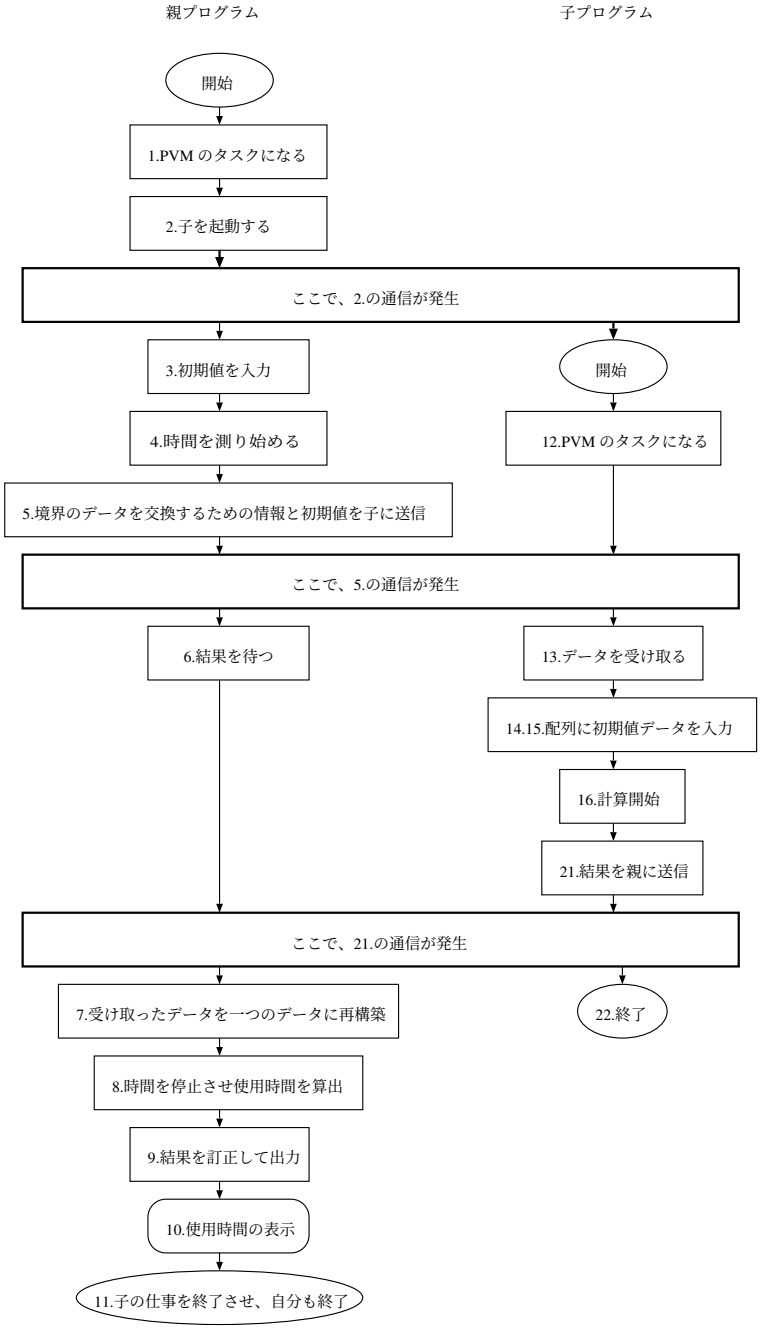

Fig. 6.5 保存則方程式のフローチャート

7　まとめ

今回の研究は接続手段とその考察、分散処理ソフトウェアとそれを使用した保存則方程式の分散化がテーマであった。接続手段に関しては既にある設備を使用する為、自由度は少なかったが、最も一般的な方式の接続手法を学んだ。他の手法では利点なども存在しただろうが、設備面を整えるというと、かなりのコストが必要で短い期間では設備を入れ換える事は到底できない。コンピュータの性能比較等でも必ずコスト比較が存在するのもそのためであると分かった。

従って、既存のシステムをアップグレードしていく事でネットワークシステムの技術が進んでいる。よって、どのような手法を取り入れようかというよりもどのように変化させられるかという考察が課題として残る。

分散処理ソフトウェアはそのインストールに過大な時間を要した。OS が対応していない機種へのインストールはそのシステムに非常に詳しくなければかなり難しい。このような問題点が存在した。

保存則方程式の分散化は通信回数、により場合によっては分散させた方が時間を要する事があると分かった。具体的にはマシン一台ごとにタスク一つずつが最速になり、マシンの台数の多すぎ少なすぎにはかえって時間がかかるという事である。よって通信時間と計算時間のバランスをとる事が重要である。その為にも適度な通信が起こるアルゴリズムの考慮を考えなくてはならない。勿論、通信回数を減らす事が一番の解決法だが、それ以外の方法で解決できるのかどうか。できるのならばどうなのか課題が残った。

参考文献

[1] 渡邉 伸征: ”衝撃波による非粘性バーガース方程式の差分法の比較”, 新潟工科大学卒業論文 (2000)

[2] 湯淺 太一 安村 道晃 中田 登志之: ”はじめての並列プログラミング”,(共立出版,1999)

[3] 飯塚 肇 緑川 博子: ”並列プログラミング入門”,(丸善,1999)

[4] 日本原子力研究所原子炉工学部炉特性研究室のホームページ
http://gaia.tokai.jaeri.go.jp/

[5] 日本 Linux 協会
http://www.linux.or.jp/

[6] 株式会社インセプト 情報通信辞典
http://www.e-words.ne.jp/

[7] 村田 英明: ”PVM 3 ユーザーズガイド リファレンスマニュアル 日本語版”,(1995)