研究速報

特集 16
研究速報

流れ場シミュレーションにおけるPVMの使用法

PVM Implementation of Parallel Computation in NST

松田進也*・大島まり*・谷口伸行*
Shinya MATSUDA, Marie OSHIMA and Nobuyuki TANIGUCHI

1. PVMについて

PVM [1]（Parallel Virtual Machine）とは，ネットワークによって接続されたコンピュータ間で，メッセージパッシングを用いて並列・分散処理環境を構築するツールです．超並列マシンのようにコンパイラが勝手に並列化してくれたり，プログラム中に並列化指示を書いておけばすむようなものとは違い，設計の段階から並列化を意識して行わなければならず，後述するようにデバッグも難しくなりますが，ネットワークでつながってさえいればUNIXからWindowsまでどんな機種でも使えること，並列化の自由度が高いことなどがメリットとして挙げられます．このPVMを用いて行ったトルクコンバータ内流れの計算を例に，PVMの具体的な使用法を説明しようと思います．

並列化には大きくわけて各コンピュータに得意な仕事（ベクトルコンピュータには重い計算，グラフィック・ワークステーションには途中経過の表示など）を割り当てる機能中心の並列化と，一台のマシンでは扱えないような大きなデータを分割するデータ中心の並列化の二種類があります．今回行ったトルクコンバータの計算では，各部品ごとに一台のワークステーションを割り当て，プログラム自体は同じというデータ並列型を用いています．またプロセス管理についても，ひとつのプロセスが他のすべてを管理するマスター・スレーブ型と，複数の対等なプロセスが互いに協力して処理を行うSPMD（Single Program Multiple Data）があります．今回はC++によるマスタープロセスを一つ起動し，計算はFORTRANによるスレーブプログラムが行う形式をとっています．

2. PVMの入手とインストール

PVMの入手方法は，anonymous ftpで持ってくるのが一番簡単でしょう．オリジナルはftp://cs.utk.edu/pub/-xnetlib/pvm3にありますが，国内ではftp.kyoto-u.ac.jpなどのミラーサイトがありますので，できるだけこちらを利用するようにしましょう．それ以外にもnetlibにメールを出してshar（シェルアーカイブ）形式やエンコードされたファイルを送ってもらう方法もあります．

ftpで持ってきたファイルはgzipで圧縮してあると思いますので，

```
% gunzip pvm.3.3.8.tar.gz
% tar -xvf pvm.3.3.8.tar
```

とすればpvm3というディレクトリができます．この文章ではpvm3がホームディレクトリにあるものとして話をすすめていきます．まず環境変数の設定が必要になりますが，Cシェルを使っていればpvm3/lib/cshrc.stubを自分の.cshrcに追加することによって簡単に設定できます．

```
% cat pvm3/lib/cshrc.stub >> ~/.cshrc
% source ~/.cshrc
```

あとはpvm3の下でmakeするだけです．

```
% cd ~/pvm3
% make
```

これでlibディレクトリに実行ファイルができるのですが，注意する点として，PVMを走らせるすべてのマシン上でこのmakeを行わなければなりません．libディレクトリの下には各機種ごとにサブディレクトリができますが，これらはlib/pvmなどのシェルスクリプトが自動的に機種を判断して呼び出してくれますので，パスを通しておく必要はありません．

これでインストールは完了です．PVMはそのサイトのどのユーザーでもインストールすることができますが，たくさんインストールしてもディスク容量が無駄ですので，管理者に頼んで/usr/local/pvm3などにインストールしてもらうといいでしょう．

*東京大学生産技術研究所　第2部

3. PVMによる並列化の実装

次に，プログラム中でPVMをどう使用していくかということですが，これは最初に述べた機能並列・データ並列のどちらを用いるのか，プロセスをマスター・スレーブ型にするのかSPMDにするのかによって異なってきます．ただPVMはメッセージパッシングによって並列化を行いますから，データ並列を用いるほうが実装しやすいでしょう．数値計算では，計算領域や巨大行列の分割などに当たります．

まず最初にユーザーによって起動されるプログラム（マスタープログラムもしくはSPMDの最初のひとつ）が子プロセスを生成するところから始まります．デフォルトではプロセスの生成場所はPVMによって自動的に決められるのですが，現在の負荷状況を判断するといった高度なことまではしてくれませんので（初期設定ファイルに各マシンの相対的な速さを書いておくことによって少しは改善しますが），ホスト名を指定して生成させる方が効率的です．ですから動的にプロセス数を増減させるようなもの（領域分割で分割数が多いものなど）では，各ホストの負荷状況を調べるなどの高度なプログラミングが必要となります．また生成される実行ファイルは特定の場所においておく必要があります（文献1）参照）．

データのやり取りを行う手順ですが，バッファにパックして相手プロセスに送り，受け取った側はそれをアンパックして使うということになります．

＜C/C++＞

```
/* バッファの初期化 */
pvm_initsend（PvmDataDefault）；
/* 整数をパック */
pvm_pkint（&n, 1, 1）；
/* 要素数10の実数配列をパック */
pvm_pkdouble（vec, 10, 1）；
/* タグを付けて送信 */
pvm_send（tid, msgtag）；
```

＜FORTRAN＞

```
CALL PVMFINITSEND（PVMDEFAULT, INFO）
*FORTRANではパックする関数はひとつだけ
CALL PVMFPACK（INTEGER4, N, 1, 1, INFO）
CALL PVMFPACK（REAL4, VEC, 10, 1, INFO）
CALL PVMFSEND（TID, MSGTAG, INFO）
```

ここで，ヴァーチャルマシンを構成するコンピュータがヘテロジニアスな場合，データをパックする際にXDRという共通のデータエンコーディングを行いますので，多少オーバーヘッドが生じてしまいます．常に同じ機種間でしかPVMを走らせないのならば，上記のPvmDataDefaultの部分をPvmDataRawにすることによってそれを防ぐことができます．また，CとFORTRANを同時に用いる場合に注意しなければならないのは，文字列の送受信です．これはCとFORTRANで文字列の扱いが違うためにpvm_pkstr（...）とPVMFPACK（STRING,...）の内部仕様が異なる（実際PVMのFORTRANサブルーチンはCの関数を呼び出している）ことによるもので，数値計算ではファイル名くらいしか使わないと思いますが注意してください．

データを受け取った側では，送ったときと全く同じ順序でアンパックし，自分の計算に使用します．

＜C/C++＞

```
/* データがくるまで待つ */
pvm_recv（tid, msgtag）；
/* データをアンパック */
pvm_upkint（&n, 1, 1）；
pvm_upkdouble（vec, 10, 1）；
```

＜FORTRAN＞

```
CALL PVMFRECV（TID, MSGTAG, INFO）
CALL PVMFUNPACK（INTEGER4, N, 1, 1, INFO）
CALL PVMFUNPACK（REAL4, VEC, 10, 1, INFO）
```

また，PVM使用時にはユーザーがコンソール上で起動したプロセス以外の全プロセスの出力は，そのコンソールのあるホストのファイルにリダイレクトされます．これはこちらが意図しなくてもデータ送信が行われてしまうことになりますので，チェックライトはデバッグ後なるべくコメントアウトする方がいいでしょう．

4. PVMをサポートするツール類

PVMは情報分野で多く用いられているため，他の並列化ツールに比べてプログラミングを支援するツール類が充実しているのも大きなメリットです．プログラマが直接行う並列化は自由度は高いのですが，デバッグの際にどのホストのどのプロセスがエラーを出しているのかわかりにくくなってしまう場合があります．またPVMのようにメッ

研　究　速　報

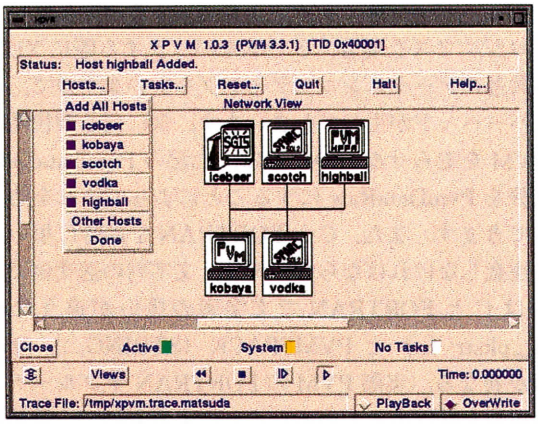

図1　XPVMによるホストの追加

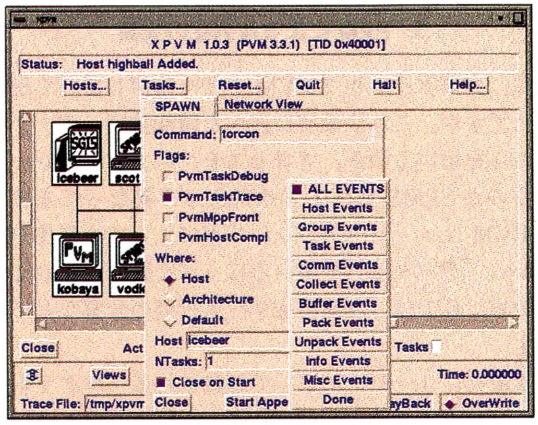

図2　XPVMによるプロセスの始動

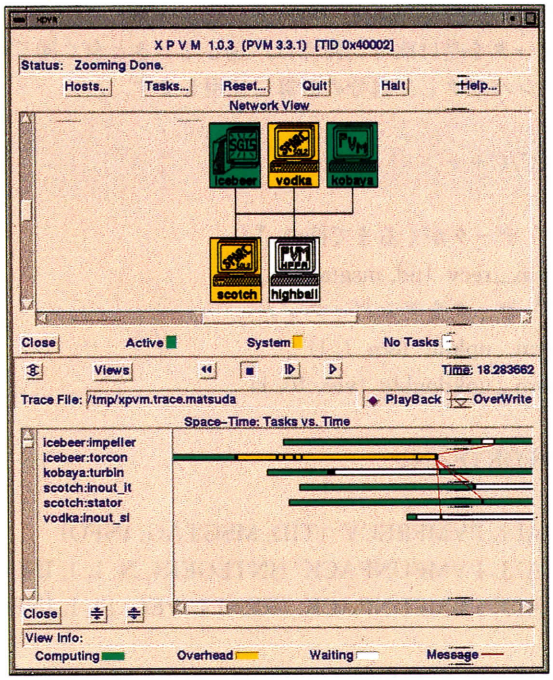

図3　ネットワークの構成と実行の様子

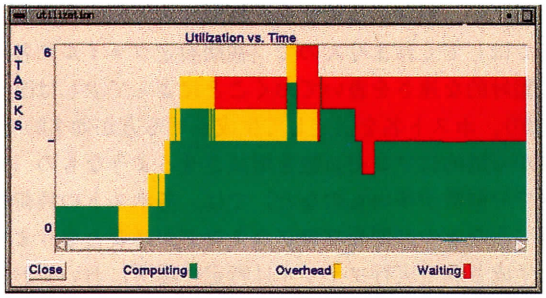

図4　プロセスの状況の表示

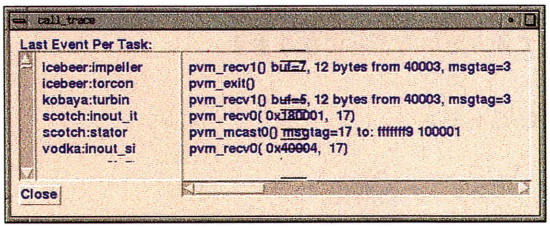

図5　コールトレース

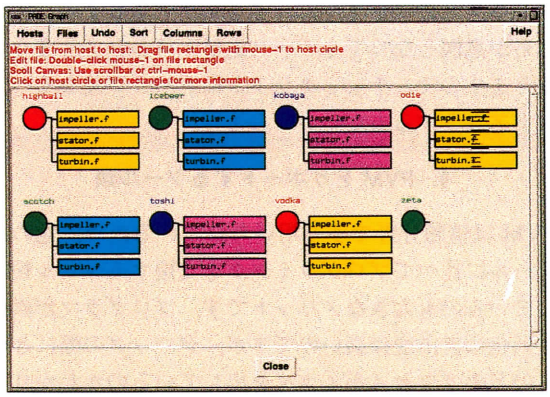

図6　PADEによるプログラム管理

セージパッシングを用いて並列化を実装しているツールでは，あるメッセージがどのプロセスからどのプロセスへ何番のタグをつけて送信されたのか，このプロセスはどのプロセスからくるどのタグのメッセージを待っているのかなどをトレースできれば，デッドロックや無駄な処理待ちを防ぐことができます．またプログラミングの段階でも，多くのソースファイルを異なる各機種ごとにコピーし，デバッグが行われるごとに確実に最新のファイルを全機種でコンパイルしなおさなければならず，その手助けとなるツールもぜひ必要です．

PVMにはこのどちらもGUI（Graphical User Interface）を用いた便利なツールがありますので，デバッ

グが容易になっています．まず実行時のプロセスやメッセージのトレースには，XPVMというツールを使用しました．これはスクリプト言語Tcl/Tkがベースとなっており，画面上でホストの追加や削除，プロセスの実行，メッセージや各プロセスの出力のトレースなどを行うことができます．PVM関数のコールトレースなどまでできるのですが，かなり重い処理になってしまうのと，並列処理という性質上各プロセスからのメッセージが非同期であることなどから，リアルタイムに処理を確認するのは困難です．そこでXPVMではまずプロセス全体の挙動をログファイルに記録しておき，あとからコマ送りで再生することができるようになっています．もちろん各トレースもみることができますので，デバッグに非常に便利です．

またプログラミング段階でのツールとしては，NIST_PADE[2]（National Institute of Standard and Technology Parallel Applications Development Environment）というパラレルメイクツールがあります．通常ヘテロジニアスな環境でのネットワークといっても，プログラミングは一台のマシンで行い，それを各機種にコピーしてメイクしなおすのが一般的です．PADEはGUI上でそれら一連の作業を行うことができます．コピーするファイル名や実行するコマンド（makeなど）を登録しておけば，メニューから自動的にPVMを起動して各機種ごとにメイクを行ってくれます．PVM自身にもaimkというメイクツールが付属していますが，PADEを用いればより簡単にプログラムを構築することができます．

コンパイラまかせの並列化とは違い，人間が直接プログラミングからデバッグまで行う必要があるわけですから，こういった開発支援ツールのよしあしが並列化ツールの普及の決め手になるといってもいいでしょう．

5. まとめ

PVMによる並列化の実際について紹介してきましたが，ネットワークでつながってさえいれば大規模な計算が取り扱えるというメリットはかなり大きいと思います．プログラミングやデバッグも前出のようなツールがあることによって，まったく手の出ないものではなくなりつつあります．数値計算においては，多くのプロセッサとメモリをもつ超大型汎用並列計算機とは違う方向の，研究室規模で行える並列処理としてこれから普及していくと思われます．

（1995年11月28日受理）

参考文献

1) Beguelin, A.L., Dongarra, J.J., Geist, G.A., Jiang, W.C., Manchek, R.J., Moore, B.K., Sunderam, V.S., Parallel Virtual Machine System, Univ. of Tennessee, Oak Ridge National Lab., Emony Univ. (1992).
2) Devaney, J.E., Lipman, R., Lo, M., Mitchell, W.F., Edwards, M., Clark, C.W., The Parallel Applications Development Environment (PADE) User's Manual, Computing and Applied Mathematics Lab., Physics Lab., National Institute of Standards and Technology (1995).