◇◇◇◇◇◇
特　集
◇◇◇◇◇◇

研究用計算機の使い方

山之上　卓 [1]
望月　雅光 [2]

1　はじめに

「研究用計算機の構成と特徴」で述べたように，研究システムでは，プログラムの作成やコンパイル，小さなプログラムの実行はフロントエンドプロセッサ monet で行ない，大きなプログラムの実行は，バッチを使ってバックエンドで実行するように構成されています．

本稿では，研究システムの具体的な使い方について述べます．

2　はじめて利用される方へ

研究システムは，九州工業大学の学生，教職員であれば利用できますが，課金を行なうため，指導教官などの支払責任者登録と利用者の利用登録が必要です．教官が利用者である場合は，本人が支払責任者になることができます．また，前もって教育システムの ID を持っておく必要があります．研究システムのユーザ ID は，教育サブシステムと同じです．

研究システムの利用申請と課金については,「研究用計算機の構成と特徴」の「2. 利用申請と課金について」をご覧ください．

研究システムを利用するためには，学内 LAN に接続したパソコンや UNIX ワークステーションから，研究システムにログインする必要があります．情報科学センター教育用サブシステムから研究システムを利用することも可能です．研究システムにログインするためには，telnet コマンドなどでフロントエンドプロセッサ「 monet.isc.kyutech.ac.jp 」に接続した後，ユーザ ID とパスワードを入力します．OS は Soralis ですので，通常の UNIX のコマンドが使えます．終了時には，exit コマンドなどでログアウトします．図 1 に telnet による研究システムのログイン例を示します．

telnet でログインすると，画面の最下行に「プロンプト (この例では `yamanoue@monet%`)」と共に「コマンドライン」が表示されます．プロンプトの右に，コマンドを入力し「Return」または「Enter」キーを押すとそのコマンドが実行されます．UNIX の基本的な使い方については,「インターネット時代のフリー UNIX 入門」や教育システムのオンラインガイドなどをご覧ください．

[1]情報科学センター，yamanoue@isc.kyutech.ac.jp
[2]情報科学センター，mochi@isc.kyutech.ac.jp (2002 年 4 月より創価大学へ転出)

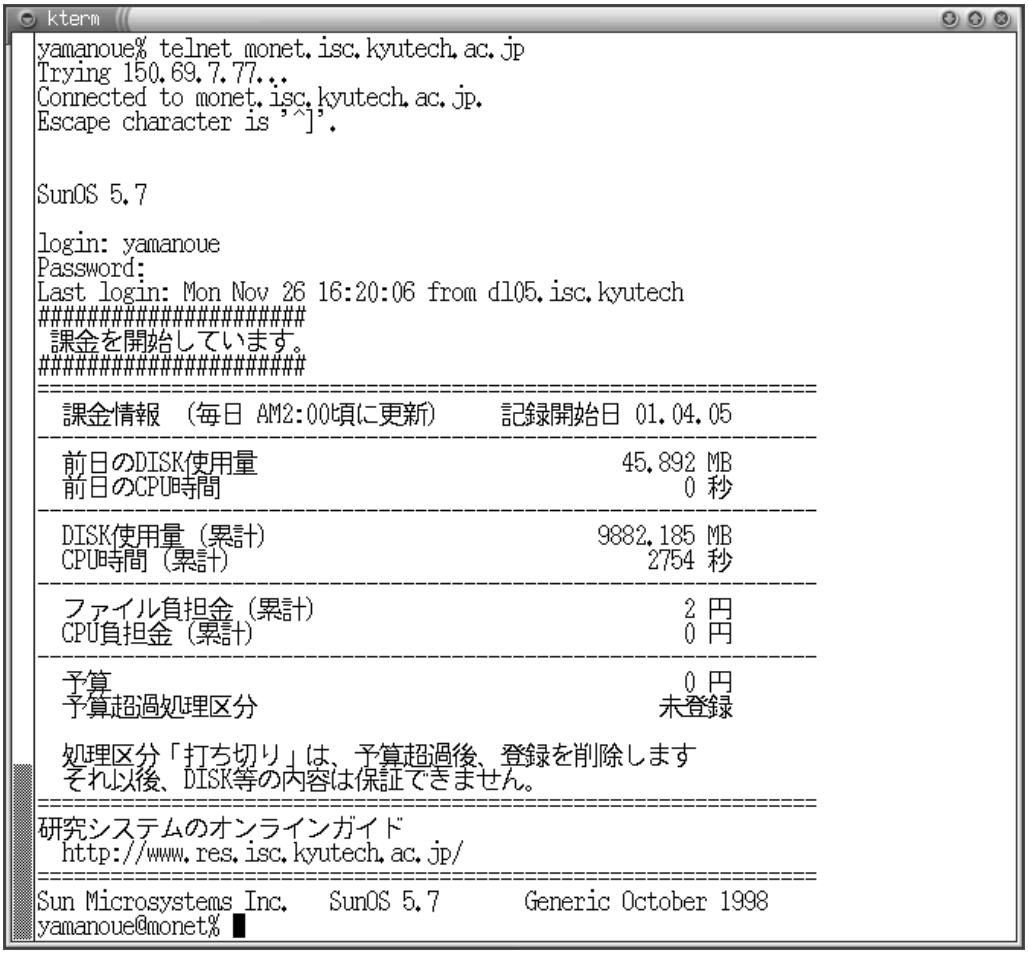

図 1: telnet による研究システムのログイン

フロントエンドプロセッサでは，UNIX のコマンドの他,「研究用計算機の構成と特徴」の「4. 開発環境」で述べている FORTRAN や C などのコンパイラ，xanalyzer や pwb などのプログラム開発ツールなどが使えます.

telnet を終了 (接続の終了) するためには,「exit」コマンドを実行します.

また,「研究用計算機の構成と特徴」の「4.2 パラレルワークスについて」で述べているように，Web ページの上で研究システムにログインし，コマンドを実行したり，バッチ投入を行なったりすることもできます (Web ブラウザによっては利用できない場合があります).

研究システムの利用の手引は，Web ページ

http://www.res.isc.kyutech.ac.jp/

に載せています.

3 コマンド ラインにおける利用方法

3.1 コンパイルと実行の例

図 2 のような台形法によって，円周率の近似値を求める FORTRAN プログラムの例を使ってコンパイルから実行までを説明します．ソースプログラムは，研究システム上の emacs などのエディタを使って作成することもできますが，手元のパソコンにソースプログラムがある場合，そのパソコンを学内 LAN に接続して ftp などでプログラムを研究システムに転送することもできます.

ソースプログラムが，ファイル dakei.f に格納されていた場合，コンパイルは以下のように frt コマンドを実行して行ないます.

```
user_id@monet% frt daikei.f
```

このとき，実行プログラムは，ファイル a.out に格納されます．実行プログラムファイル名 (実行プログラム名) を指定してコンパイルするには,

```
user_id@monet% frt -o daikei daikei.f
```

のように，「-o 実行プログラム名」を付けて frt コマンドを実行します.

コンパイルされた実行プログラムを実行するには，その実行プログラム名をコマンドとして入力します．daikei が実行プログラム名の場合は,

```
user_id@monet% daikei
 n?
1000000
 n=1000000
 pai=3.141592653589793
user_id@monet%
```

のようになります．ここで，このプログラムは実行時に，変数 n に入力する値を求めますので，n? の表示の下で，コンソールから，1000000 を手で入力しています．

入力する値を，あらかじめ作っておいたデータファイルから入力させたり，出力をファイルに格納したい場合は,「リダイレクト」を使うことができます．リダイレクトとは,「入出力先を変更する」，という意味で，入力については,「<」，出力については,「>」を使います．

このプログラムで，入力はファイル input に格納されているデータを使い，出力結果をファイル output に格納する場合は，

```
user_id@monet% daikei < input > output
```

のように実行します．

3.2 バッチ投入

「バッチ」とはコンピュータを操作するための一連のコマンドを記述して，まとめて実行できるようにしたものです．直接人間とやりとりを行なう必要がない，定型的な処理を行なうプログラムの場合は，バッチを利用した方が便利で，実行効率があがります．

ユーザは，バッチを「バッチキュー」に投入することによって，バッチの実行を待ちます.「バッチキュー」とは，バッチの実行待ち行列のことです．バッチキューに投入されたバッチはキューの最後に並びます．基本的には，キューの先頭にあるバッチから実行が開始されます．

研究システムのバッチキューには，バッチの実行環境 (利用できる CPU の数やメモリの大きさ，実行時間の制限など) が異なる 3 種類 (「クラス」) のキューがあります．クラス A は比較的小さな処理，クラス B は中規模の処理に適しています．クラス C はバックエンドプロセッサを用いた，大量のメモリを使う計算や，並列計算を行なうための処理に適しています．研究システムのバッチキューの詳細については「研究用計算機の構成と特徴」の「4.3 キューの構成」をご覧ください．

研究システムは，バッチシステムとして，NQS を使っています．NQS の代表的なコマンドとそのオプションを表 1 に示します．なお，NQS の詳細な利用方法は，以下に示す URL のオンライマニュアルの第 5 章 NQS の使用方法を参照してください．

NQS のオンラインマニュアル (日本語)

http://www.res.isc.kyutech.ac.jp/manual3/japanese/nqs/index.htm

NQS のオンラインマニュアル (英語)

http://www.res.isc.kyutech.ac.jp/manual3/english/nqs/index.htm

ここで，簡単な例を用いて，ジョブ投入の方法を説明します．

(1) バッチジョブの実行依頼 qsub コマンドを用いてジョブの実行依頼をします．ここでは，キュー C を利用し，a.sh を実行します．実行すると，次のようなメッセージが表示されます．

```
      program exdaikei
      EXTERNAL F1
      double precision a,b,f1,s2,daikei
      A=0.0
      B=1.0
      write(*,*) 'n?'
      read(*,*) n
      write(*,*) 'n=',n
      S2=daikei(F1,N,A,B)
      WRITE(*,*) 'pai=',S2
      STOP
      END
*
      double precision FUNCTION F1(X)
      double precision x
      F1=4.0D0/(1.0D0+X*X)
      RETURN
      END
*
      double precision FUNCTION daikei(F,n,A,B)
      double precision f,a,b,s,x,h
      H=(B-A)/n
      S=0.0
      DO 10 I=1,n-1
         X=I*H+A
         S=S+F(X)
 10   CONTINUE
      daikei=H*(F(A)+F(B))*0.5+h*s
      RETURN
      END
```

図 2: FORTRANプログラム (daikei.f) の例

表 1: 主要なコマンド

コマンド名とその形式	機能	主要なオプション
qsub [option] [shell-script-file]	ジョブの投入	-a 指定時間後に実行 -o　標準出力の出力先 -q キューの指定
qstat [option]	ジョブの状態を調査	-a すべてのジョブを表示 -l　長い形式での表示 -u ユーザ名の指定
qdel [option] requestID...	ジョブの取り消し	-k 実行中のジョブを中止

```
user_id@monet% qsub -q C a.sh
Request 2832.monet submitted to queue: C.
user_id@monet%
```

ここで，2832.monet が requestID になります．なお，a.sh の内容は次のとおりです．実行結果を出力するためのディレクトリに移動し，絶対パスで実行するプログラムを指定します．

```
#!/usr/bin/sh

cd /home/RES/a00001ty/output
/home/RES/a00001ty/a.out
```

(2) バッジジョブの状態を調査 qstat コマンドを用いてジョブの状態を調査します．ここでは，すべてのジョブを表示してみます．すぐに実行が完了するジョブの場合には，ジョブが表示されないまま，実行が完了してしまうことがあります．そのときは，次の (4) のようにして確認してください．

```
user_id@monet% qstat -a
A@monet;  type=BATCH;  [ENABLED, INACTIVE];  pri=16
  0 exit;   0 run;   0 queued;   0 wait;   0 hold;   0 arrive;

B@monet;  type=BATCH;  [ENABLED, INACTIVE];  pri=16
  0 exit;   0 run;   0 queued;   0 wait;   0 hold;   0 arrive;

C@monet;  type=BATCH;  [ENABLED, RUNNING];  pri=16
  0 exit;   6 run;   0 queued;   0 wait;   0 hold;   0 arrive;

        REQUEST NAME        REQUEST ID            USER  PRI    STATE      PGRP
   1:           a.sh      2832.monet             mochi  31  RUNNING     28975
```

(3) バッチジョブの出力を確認 プログラム名と requestID の番号がついたファイルが作成され，結果が出力されます．次のようにして，ファイルを確認します．

```
user_id@monet% ls a.sh*2832
a.sh.e2832  a.sh.o2832
```

ここで，a.sh.e2832 は標準エラー出力，a.sh.o2832 は標準出力がファイルに書き出されています．それぞれテキストファイルですので，エディタ等で内容を確認してください．

(4) バッチジョブの削除 qdel コマンドを用いて実行中のバッジジョブを削除します．ここでは，(2) において実行したジョブを次のようにして削除します．しばらくして，削除が行なわれます．

```
user_id@monet% qdel -k 2832.monet
Request 2832.monet is running, and has been signalled.
```

このあと念のため，本当に削除できたかどうかを，qstat コマンドを用いて確認してください．

4 Web ページで研究システムを利用する方法 (ParallelWORKS)

手元のコンピュータの Web ブラウザで研究システムの ParallelWORKS のページ

ParallelWORKS の web ページ

```
http://www.res.isc.kyutech.ac.jp/pworks/
```

を表示することによって，このページの上で研究システムを利用することができます．このページの表示は，図 3 のようになっています．

図 3: ParallelWORKS の画面

なお，Web ブラウザによっては ParallelWORKS が動作しない場合がありますのでご注意ください．教育システムの netscape では現在動作しません．

4.1 ログイン

ParallelWORKS で研究システムへのログインは，以下のように行ないます.

1. 図 3 の画面の上で「開発者向け GUI (一般ユーザ用)」の部分をクリックしてしばらく待つと，図 4 のようなウィンドウが表れます.

   最初に使用する場合は，Java のプラグインのインストール作業が始まる場合があります．このときは，表示される指示に従って，Java プラグインのインストールを行なってください.

2. 図 4 のウィンドウの 「Group Name」の下の,「res」の左側の四角形をクリックしてこの四角形にチェックを入れます.

3. 「User ID」の下にある長方形の中にユーザ ID を入力し,「Password」の下にある長方形の中にパスワードを入力します.

4. パスワードを入力した長方形の右にある「OK」ボタンをクリックします.

5. ログインに成功すると，確認のウィンドウが表示されます．ここで「了解」ボタンをクリックすると，図 5 のような画面が表れます.

   この画面の「Help」の部分をクリックすることによって，ParallelWORKS の利用方法を調べることができます.

6. この画面の上部中央付近にある「res」ボタンをクリックし，チェックを入れると，その下にある,「Job Entry」と「System Monitor」の文字が濃い色に変わり，これらの機能が使えるようになったことを表します．バッチ投入やコマンドの実行を行なう時は Job Entry の部分をクリックします．研究システムの負荷の状態などを調べる時は，System Monitor の部分をクリックします.

7. Job Entry の部分をクリックすると，図 6 のように「job Submission」と書かれた四角形が現われます．この四角形をクリックすると，図 7 のような画面が表示されます．この画面でファイルを選択し，ボタンを押してバッチを投入したり，ファイルの内容を表示したり，編集したり，コマンドを実行したりすることができます.

4.2 ログアウト

ParallelWORKS で研究システムからログアウトを行なうには，図 5 または 図 7 の画面の右上のにある,「Logout」ボタンをクリックします．ここで，図 8 のようなウィンドウが現れますので,「Group Name」の下の,「res」の左側の四角形をクリックしてこの四角形にチェックを入れ，このウィンドウの右にある「Logout」ボタンをクリックします．ログアウトに成功すると，確認のウィンドウが表示されますので,「了解」ボタンをクリックします.

図 4: ParallelWORKS のログイン画面

図 5: ログイン後の画面

図 6: Job Entry の選択

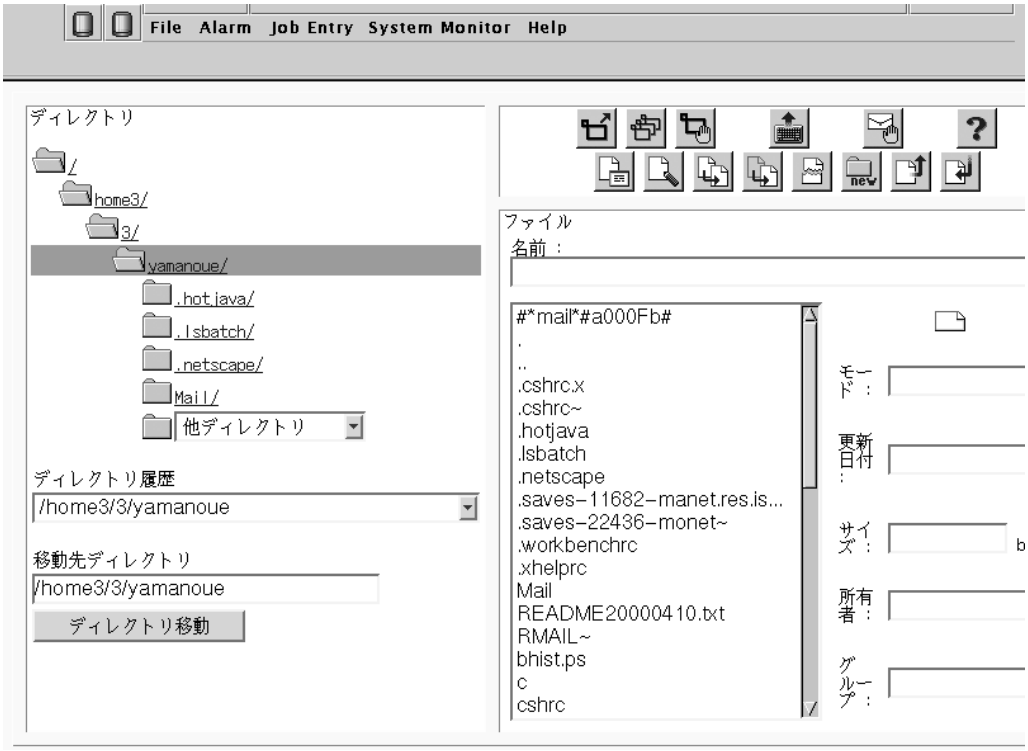

図 7: バッチ投入操作画面

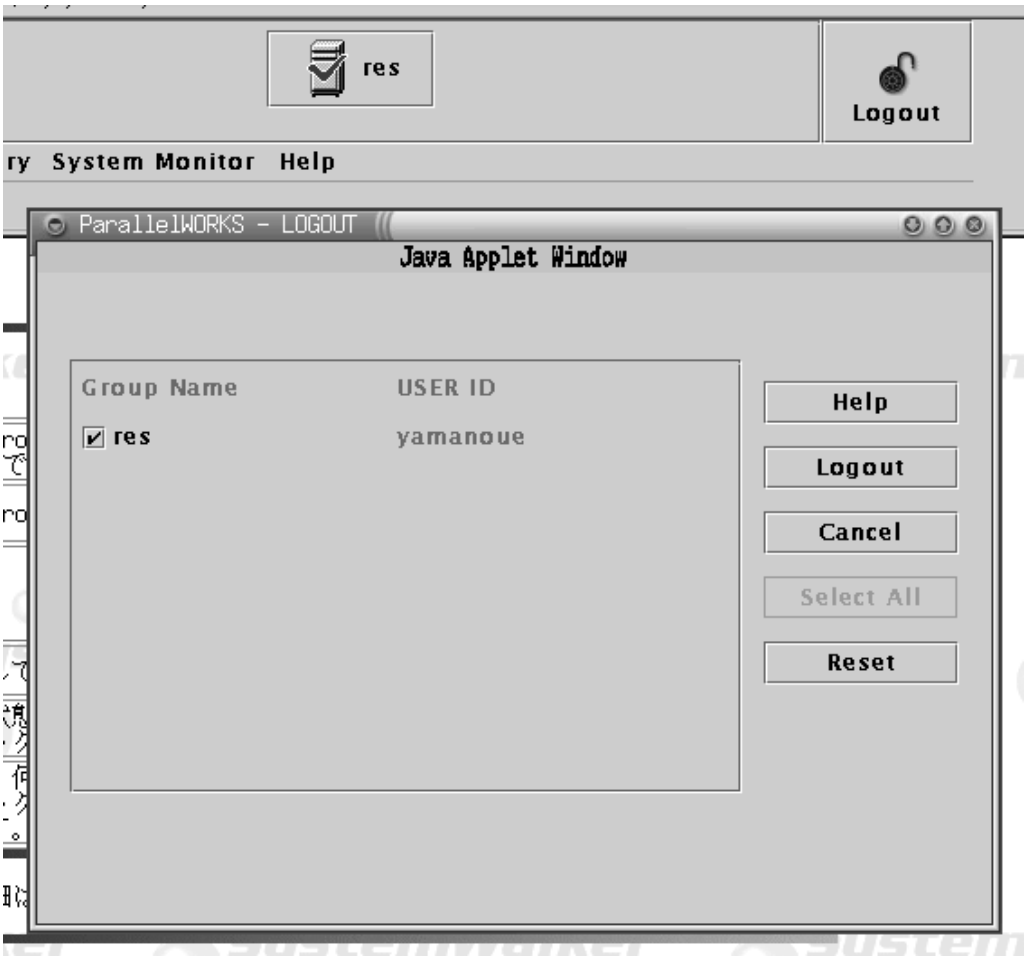
res
Logout
ry System Monitor Help
ParallelWORKS - LOGOUT
Java Applet Window
Group Name
USER ID
res
yamanoue
Help
Logout
Cancel
Select All
Reset

図 8: ログアウト

4.3 ディレクトリとファイルの選択

ファイルの内容を表示したり，編集したり，バッチ投入したりする場合，ParallelWORKS は，まず，対象のファイルを選択する必要があります.

これを行なうためには以下を行ないます.

1. 図 7 の画面の左側のディレクトリ (書類入れの形をしたアイコン) の一覧表のなかから，そのファイルを格納しているディレクトリをクリックします．ここで，もしあるはずのディレクトリが見当たらなかったら,「他のディレクトリ」の表示の右側にある三角形が書かれたボタンをクリックすることにより，ここに表示されていなかったディレクトリの一覧表を表示することができます.
2. ディレクトリ一覧表の右側にある，ファイルの一覧表の中から該当するファイル名をクリックして選択します.

4.4 テキストファイルの表示と編集

テキストファイルを表示するには，目的のファイルを選択した後，図 9 のファイル表示ボタンをク

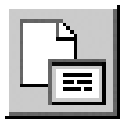

図 9: ファイル表示ボタン

リックします．同様に 図 10 のファイル編集ボタンをクリックすることによって，選択したファイルの

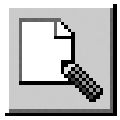

図 10: ファイル編集ボタン

編集を行なうことができます.

4.5 コマンド実行

コマンドを表示するには，図 11 のボタンをクリックし，図 12 の画面の「コマンド入力」の右の長方形の中にコマンドを入力します．この図は，daikei.f を並列化オプションを漬けて frt コマンドでコンパイルするときの入力例を表しています.「コマンド投入」ボタンをクリックすることによって，このコマンドが実行されます.

図 11: コマンド実行ボタン

このコマンドがコンソールから何か入力を求めるものであれば，あらかじめその内容をファイルに作成しておき，リダイレクト「<」の右にそのファイル名を記述します．コマンドの出力をファイルに格納するには，リダイレクト「>」の右に格納するファイル名を記述します．

コマンドの実行結果などは,「コマンド投入結果表示欄」に表示されます．

図 12: コマンド入力画面

4.6 ファイル転送

手元のコンピュータにあるファイルを研究システムに転送するには，図 13 のアップロードボタンを

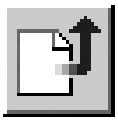

図 13: ファイルアップロードボタン

クリックします．転送するファイルを選択する画面が表示されたら,「転送するファイル:」の右側に直接

ファイル名を入力するか,「参照ボタン」をクリックして，ファイル選択ウィンドウを表示し，ここで目的のファイルを選択して,「開く」ボタンをクリックします (図 14)．転送するファイルが入力されたら，

図 14: ファイルアップロード画面

この画面に表示されているアップロードボタンをクリックすることによって，そのファイルが研究システムに転送されます．

4.7 バッチ投入

ParallelWORKS は，バッチを実行するための様々な機能を備えています．また，バッチを使うことによって，バックエンドコンピュータを利用することができます．

ここでは，以下のバッチを例に説明します．

ParallelWORKS で投入するバッチの例

```
#!/usr/bin/sh
#  @$-q A
cd $QSUB_WORKDIR
daikei <input >output
```

この例で，

```
#!/usr/bin/sh
```

は，B-sh のスクリプトでバッチを実行することを表します．

```
#  @$-q A
```

は，クラス A のバッチキューにこのバッチを投入することを表します．

```
cd $QSUB_WORKDIR
```

は，バッチを実行するためのディレクトリに移動することを表します．これは，投入したバッチが存在するディレクトリになります．

```
daikei <input >output
```

は，実行プログラム daikei を，input ファイルからデータを入力して実行し，結果を，output ファイルに格納することを表しています．

ParallelWORKS は，パラメータを選択することによってバッチファイルを自動的に生成する機能も持っています．

バッチを投入するには，図 7 の画面でバッチファイルを選び 図 15 のサブミットボタンをクリックし

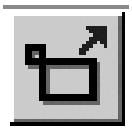

図 15: サブミットボタン

ます．選択したバッチの名前の拡張子が,「バッチ名.sh」のように，sh である場合，もしこのバッチにクラスの指定があれば，そのクラスのキューに投入されます．クラスの指定がなければ，クラス A のキューに投入されます．拡張子が sh でない場合は，クラスの指定や，その他のパラメータを指定するための画面が表れ，そこで各種の指定を行なってバッチを投入します．

投入したバッチの状態を表示するには，図 16 のバッチ状態表示ボタンをクリックします．

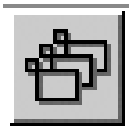

図 16: バッチ状態表示ボタン

投入したバッチジョブを削除するには，図 17 のバッチ操作ボタンをクリックします．ここで，図 20 のようなバッチ操作画面が表示され，対象のバッチの左の四角形をクリックしてチェックし，この画面に表示されている操作を選び，実行します．

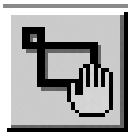

図 17: バッチ操作ボタン

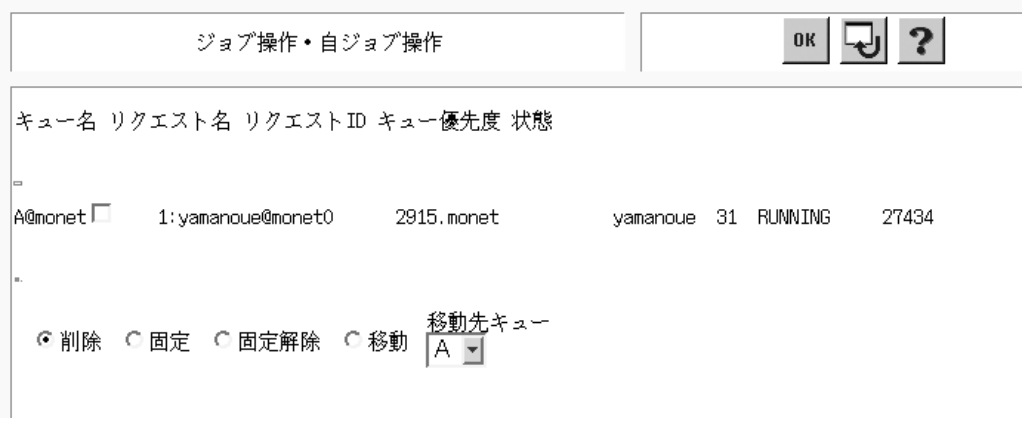

図 18: バッチ操作画面

5 プログラムの高速化

研究システムでプログラムを高速化するには以下のような方法が有効です.

5.1 最適化オプションをつけてコンパイルする

最も簡単で有効な高速化の手法です. コンパイル時に最適化オプションを指定すると, 最適化を行なわない場合の 2〜3 倍のスピードが出る場合があります. 長時間動作させるプログラムを最適化することによって, 計算が早く終了し, CPU 課金が安くなり, 多くの人が快適にシステムを 利用できるようになります.

以下は FORTRAN のコンパイル時に, 最適化を行なう例です.

```
frt -Kfast,ULTRA2,V8PLUS,gs,eval,prefetch -o daikei daikei.f
```

ここで -o daikei は, 実行プログラム名を daikei とすることを表します. daikei.f がソースプログラムです. c の場合は, frt の代わりに fcc を使います.

-Kfast,ULTRA2,V8PLUS,gs,eval,prefetch が 最適化を行なうことを表します. 最適化オプションの各部分の意味は以下の通りです.

研究システム FORTRAN の最適化オプション

fast	最適化オプションの自動選択を行ないます.
URTRA2,V8PLUS	UltraSPARC-II 向け (32bit) の最適化を行ないます.
gs	広域命令スケジューリングの最適化を行ないます.
eval	式の評価順序変更を行ないます.
prefetch	プリフェッチ命令 (メモリ先読み) の使用を行ないます.

この他の最適化オプションとして，-Kparallel, -Kreduction などの自動並列化のオプションがあります．また，自動並列化されたループを表示するオプションとして，-Et があります．自動並列化は，ループ内の計算において，計算データ間に依存関係がないなどの限られた場合に最適化を行ないます．それ以上の並列化を行なうためには，OpenMP や PVM を使ったりする必要があります.

最適化には副作用が伴う場合があります．計算の種類によっては，これらのオプションの一部を外さないと計算結果がおかしなものになる場合があります．また，実数計算を行なう場合は，計算精度を落さないために，できる限り倍精度実数 (double precision) を使うことを勧めます．倍精度実数を使っても計算速度が遅くなることはあまりありません.

並列化オプションの詳しい説明は

```
http://www.res.isc.kyutech.ac.jp/manual2/japanese/index_J.html
```

の富士通オンラインマニュアル「並列言語処理パッケージ」をご覧ください (九工大学内のみ閲覧可能).

5.2 高速なライブラリを利用する

連立一次方程式の計算など一般的に良く使われる プログラムは，既に存在している高速ライブラリを利用することによって， プログラムを新たに作成することなく，簡単に高速化できる場合があります.

研究システムでは，数値計算ライブラリ SSL II を利用することができます．このライブラリは，並列計算を使った高速計算を行なうサブルーチンなども含んでいます． SSL II の利用法についても，富士通オンラインマニュアル「並列言語処理パッケージ」をご覧ください.

インターネット上で入手可能な高速ライブラリとして, netlib (http://www.netlib.org/) などもあります.

5.3 プログラムを見直す

ちょっとした工夫をすることによって，速く計算できる場合があります．プログラムを解析するためのツールとして，coll と samp コマンドなどを利用することができます．coll コマンドは，プログラムの実行時に，プログラムのどの部分がどのくらい時間がかかっているかの情報を収集します． samp コマンドは， coll コマンドで収集された情報を表示します．以下は，frt コマンドでプログラムのコンパイルを行ない coll と samp コマンドでプログラムのどの部分がどれくらい時間がかかっているかを調べた例です.

- コンパイル

  以下はコンパイルの例です．特に特別なオプションを付ける必要はありません．

```
user_id@monet% frt -Kfast,ULTRA2,V8PLUS,gs,eval,prefetch -o daikei daikei.f
```

- coll コマンドで実行時の情報収集を行なう

  以下のように coll コマンドで実行時の情報収集を行ないます．ここで-d samp.dat は，収集した情報を， samp.dat ファイルに格納することを表します．

```
user_id@monet% coll -d samp.dat daikei
 n?
10000000
 n=10000000
 pai=3.141592653589987
```

- samp コマンドで結果を表示する

  samp コマンドを -f オプションを付けて実行することによりサブルーチン単位での実行時間分布を表示することができます．

```
user_id@monet% samp -d samp.dat -f daikei
************************************************
*             running cost by function         *
************************************************

 sampling    elapsed         %        name
 count       time(sec)
-------------------------------------------------
      122         1.22     72.19      f1  (F)
       45         0.45     26.63      daikei  (F)
        1         0.01      0.59      _start
        1         0.01      0.59      other
        0         0.00      0.00      main
        0         0.00      0.00      MAIN  (F)
        0         0.00      0.00      _init
        0         0.00      0.00      _fini
-------------------------------------------------
      169         1.69                total
```

図 19: samp コマンドによるプログラムの解析 (サブルーチン単位の集計)

  samp コマンドを -l オプションを付けて実行することにより行単位での実行時間分布を表示することができます．

```
user_id@monet% samp -d samp.dat -l daikei
************************************************
*             cost by line                     *
************************************************


---------->  daikei.f
line  count
  1    0           program exdaikei
  2    0           EXTERNAL F1
  3    0           double precision a,b,f1,s2,daikei
  4    0           A=0.0
  5    0           B=1.0
  6    0           write(*,*) 'n?'
  7    0           read(*,*) n
  8    0           write(*,*) 'n=',n
  9    0           S2=daikei(F1,N,A,B)
 10    0           WRITE(*,*) 'pai=',S2
 11    0           STOP
 12    0           END

 13    0     *
 14    0           double precision FUNCTION F1(X)
 15    0           double precision x
 16  118           F1=4.0D0/(1.0D0+X*X)
 17    0           RETURN
 18    4           END
 19    0     *
 20    0           double precision FUNCTION daikei(F,n,A,B)
 21    0           double precision f,a,b,s,x,h
 22    0           H=(B-A)/n
 23    0           S=0.0
 24    0           DO 10 I=1,n-1
 25    8              X=I*H+A
 26   20              S=S+F(X)
 27   17       10   CONTINUE
 28    0           daikei=H*(F(A)+F(B))*0.5+h*s
 29    0           RETURN
 30    0           END
 31    0
```

図 20: samp コマンドによるプログラムの解析 (行単位の集計)

この例では，関数 F1 の計算に全体の計算時間の 72.19 その中でも 16 行目の F1=4.0D0/(1.0D0+X*X) に最も時間がかかっていることがわかります.

時間がかかっている部分がわかれば，その部分の計算方法を工夫することによって大幅に計算時間を短縮できることがあります.

5.4 プログラムを PVM などを使って並列化する

プログラムの並列化 (同時に複数のコンピュータを使って計算をさせる) によって，速くなる場合があります．最近，大型計算機センターに 導入されているコンピュータは，すべて，並列プログラムを主に走らせるための並列コンピュータです (従来のシーケンシャルなプログラムも利用できます)． 情報科学センターでは，並列プログラムを作成するためのパッケージとして， PVM や OpenMP を利用することができます.

6 PVMを使った並列動作プログラムの作成

PVM は並列コンピュータやコンピュータ同士をネットワークで接続したコンピュータクラスタなどで並列プログラムを動作させるためのソフトウェアパッケージです．並列計算システムの業界標準になっています．PVM は九州大学大型計算機センターでも利用できます.

6.1 PVM のプログラム例

PVM でプログラムは以下のようにして並列計算を行ないます.

1. マスタータスク (プログラム) をユーザが起動する.
2. マスタータスクが，異なる複数の CPU 上でそれぞれワーカータスクを起動する．これらのタスクはそれぞれの CPU で同時に並行動作する.
3. マスタータスクやワーカータスク間でメッセージパッシングを行なうことによって，同期を取ったり，データ交換を行なう.

以下は，数値積分によって，演習率を計算する PVM プログラムの例です．このプログラムの詳細については広報第 9 号の「PVM を使った並列プログラミング」(http://www.isc.kyutech.ac.jp/ kouhou/ kouho9/ pvm.html) をご覧ください．また，使用している PVM のサブルーチンなどについては，PVM の Web ページ (http://www.epm.ornl.gov/ pvm/) からリンクされている，Man pages (http://www.epm.ornl.gov/ pvm/man/ manpages.html) や，PVM に関する書籍などをご覧ください.

マスタータスクとワーカータスクの記述例を図 21 と図 22 に示します.

```
*     pai_master
*        <No.of processor> <No.of rectangle...precision>
      program paimas
      include '/usr/local/pvm3/include/fpvm3.h'
      integer mytid,me,i,nproc,n,tids(0:32),status,numt,msgtype,info
      character*8 arch
      double precision d,s,x
      write(*,*) 'nproc, n?'
      read(*,*) nproc,n
      call pvmfmytid(mytid)
      arch = '*'
      call pvmfspawn('pai_workerf', PVMDEFAULT, arch, nproc, tids,numt);
      if( numt .lt. nproc) then
         print *, 'trouble spawning '
         call pvmfexit(info)
         stop
      endif
      d=dfloat(1.0)/n
*
      do 10 i=0, nproc-1
         call pvmfinitsend(PVMDEFAULT,info)
         call pvmfpack(INTEGER4, nproc, 1, 1, info)
         call pvmfpack(INTEGER4, i, 1, 1, info)
         call pvmfpack(INTEGER4, n, 1, 1, info)
         call pvmfpack(REAL8,    d, 1, 1, info)
         call pvmfsend(tids(i),  0, info)
 10   continue
      s=0.0
      msgtype = 5;
      do 20 i=0, nproc-1
         call pvmfrecv(-1, msgtype, info )
         call pvmfunpack( REAL8, x, 1, 1, info)
         s=s+x
 20   continue
      write(*,100) s*d
 100  format(' pai=',f18.13)
      call pvmfexit(info)
      end
```

図 21: 円周率を計算する PVM プログラムのマスタータスク (pai_master.f)

```
*    calculate pai (3.191592) ... worker
*
      program paiwrk
      include '/usr/local/pvm3/include/fpvm3.h'
      integer mytid, me, i, nproc, n, msgtype,master,info
*
      double precision s,x,d

* enroll in pvm
      call pvmfmytid(mytid)

      msgtype=0;
      call pvmfrecv(-1,msgtype,info)
      call pvmfunpack(INTEGER4,nproc,1,1,info)
      call pvmfunpack(INTEGER4,me,1,1,info)
      call pvmfunpack(INTEGER4,n,1,1,info)
      call pvmfunpack(REAL8,   d,1,1,info)

      s=0.0

      do 10 i=me, n-1,nproc
        x=(i+0.5)*d
        s=s+4.0/(1.0+x*x)
 10   continue

      call pvmfinitsend(PVMDEFAULT,info)
      call pvmfpack(REAL8, s,1,1, info)
      msgtype=5
      call pvmfparent(master)
      call pvmfsend(master, msgtype,info)
      call pvmfexit(info)
      end
```

図 22: 円周率を計算する PVM プログラムのワーカータスク (pai_worker.f)

6.2 PVM プログラムのコンパイル

PVM プログラムをコンパイルするためには，Makefile.aimk という名のファイルにコンパイル方法などを記述して， aimk コマンドでこれをコンパイルする必要があります．Makefile.aimk の記述は，UNIX の Makefile と同様に行ないます．左にある空白は TAB で空けるなどの注意が必要です．

図 23 は，ファイル pai_master.f に格納された図 21 のマスタータスクと，ファイル pai_worker.f に格納された図 22 のワーカータスクのプログラムをコンパイルする，Makefile.aimk ファイルです．この例では，

PVM プログラムのコンパイル例

```
user_id@monet% aimk pai
```

を実行することによって，PVM プログラムを最適化してコンパイルし，PVM のライブラリとリンクして，ディレクトリ~/pvm3/bin/SUN4SOL2 の下に，pai_master と pai_worker という名前でマスタータスクとワーカータスクの実行プログラムが格納されます．

6.3 PVM プログラムの実行

コンパイルされた PVM プログラムは，そのまま実行プログラム名をコマンドとして実行しても動きますが，フロントエンドプロセッサでは複数の CPU を利用することができません．PVM プログラムを高速に実行するためには，バッチをクラス C のバッチキューに投入する必要があります．図 24 にバッチプログラムの例を示します．

この例で，

```
# @$-q C
```

はクラス C のバッチキューにこのバッチを投入することを表します．

```
echo |pvm
```

は，PVM デーモンを起動することを表します．バッチの投入は，コマンドラインで qsub コマンドを実行したり，ParallelWORKS の上でバッチ投入ボタンをクリックしたりすることで行ないます．

7 おわりに

本稿では新研究システムの利用法について述べました．既に多くのユーザが研究システムを利用しています．研究システムには本稿では述べていない機能がまだありますが，これらについては，オンラインガイドなどで参照していただけると幸いです．

本稿ではプログラムの並列化として PVM を使った例しか挙げていませんが，研究システムでは，共有メモリ型並列コンピュータを使う場合の並列プログラミングに有効な OpenMP を使うこともできます．また，近年，分散メモリ型並列コンピュータで利用するための並列化ライブラリとして，MPI が多

```
#
# Makefile.aimk for PVM example programs.
#
# Set PVM_ROOT to the path where PVM includes and libraries are installed.
# Set PVM_ARCH to your architecture type (SUN4, HP9K, RS6K, SGI, etc.)
# Set ARCHLIB to any special libs needed on PVM_ARCH (-lrpc, -lsocket, etc.)
# otherwise leave ARCHLIB blank
#
# PVM_ARCH and ARCHLIB are set for you if you use "$PVM_ROOT/lib/aimk"
# instead of "make".
#
# aimk also creates a $PVM_ARCH directory below this one and will cd to it
# before invoking make - this allows building in parallel on different arches.
#

SDIR       =       ..
BDIR       =       $(HOME)/pvm3/bin
XDIR       =       $(BDIR)/$(PVM_ARCH)

FRT        =       frt
OPTIONS    =       -Kfast,ULTRA2,V8PLUS,gs,eval,prefetch
CFLAGS     =       $(OPTIONS) -I$(PVM_ROOT)/include $(ARCHCFLAGS)
LIBS       =       -lpvm3 $(ARCHLIB)
GLIBS      =       -lgpvm3

LFLAGS     =       -L$(PVM_ROOT)/lib/$(PVM_ARCH)

$(XDIR):
        - mkdir $(BDIR)
        - mkdir $(XDIR)

pai:   pai_master pai_worker
pai_master:        $(SDIR)/pai_master.f $(XDIR)
        $(FRT) $(CFLAGS) -o pai_master $(SDIR)/pai_master.f \
        $(LFLAGS) $(GLIBS) $(LIBS)
        mv pai_master $(XDIR)
pai_worker:        $(SDIR)/pai_worker.f $(XDIR)
        $(FRT) $(CFLAGS) -o pai_worker $(SDIR)/pai_worker.f \
        $(LFLAGS) $(GLIBS) $(LIBS)
        mv pai_worker $(XDIR)
```

図 23: プログラムのをコンパイルする Makefile.aimk ファイルの例

```
# @$-q C
#!/bin/sh
cd $QSUB_WORKDIR
rm output
echo |pvm
pai_master < input > output
```

図 24: PVM プログラムのバッチの例

く使われるようになってきました．MPI については，九州大学情報基盤センターで利用できます．また，PVM から MPI へのプログラムの書き換えは比較的容易に行なうことができます．

参考文献

[1] 山之上卓「PVM を使った並列プログラミング」広報第９号, 九州工業大学・情報科学センター, pp.74-92, 1997

[2] 南里豪志, 天野浩文「OpenMP 入門」九州大学情報基盤センター広報, Vol. 1, No.3, pp.186-215, 2001