可視化サーバ(ismprsm)利用の手引き

Version 1.4

2007 年 11 月

1 システム構成 1

2 可視化サーバへのログイン 2

3 SGI OpenGL Vizserver の利用方法 3

3.1 OpenGL Vizserver クライアント・ソフトウェアのインストール 3
3.2 OpenGL Vizserver サーバへの接続開始・終了方法 3
3.2.1 接続開始 3
3.2.2 接続終了 5
3.3 利用時の注意点 5

4 AVS Express PST の利用方法 6

4.1 AVS/Express について 6
4.1.1 1つのモジュール内での並列処理 6
4.1.2 モジュール毎の並列処理 7
4.1.3 ファイル読み込みの並列化 7
4.1.4 レンダリング（描画処理）の並列化 8
4.2 AVS のインストール場所と起動 9
4.2.1 インストール場所 9
4.2.2 起動 9
4.2.3 並列化 9
4.2.4 起動シェル 10
4.2.5 AVS の各種マニュアルについて 11

5 IDL の利用方法の利用方法 12

5.1 IDL について 12
5.1.1 iTools 12
5.1.2 アプリケーション開発 12
5.1.3 対話的処理 12
5.1.4 データ I/O 12
5.2 IDL のインストール場所と起動 13
5.2.1 インストール場所 13
5.2.2 起動 13
5.3 IDL の各種マニュアルについて 13

6 ISL の利用方法の利用方法 14

7 コンパイラ 15

7.1.1 Intel Compiler 15
7.1.2 GNU コンパイラ 15
8 Intel コンパイラにおけるプログラムの最適化方法 16
8.1 シングルプロセッサプログラムの最適化方法 16
8.1.1 データ型 16
8.1.2 オプション 17
8.1.3 makefile について 20
8.2 並列プログラムの最適化方法 21
8.2.1 自動並列化 21
8.2.2 OpenMP プログラムのコンパイル 23
8.2.3 MPI プログラムのコンパイル 31
8.3 並列ループのデータ分散 33
9 並列プログラムの実行：dplace コマンド 34
9.1 自動並列化／OpenMP プログラムでの dplace コマンド 34
9.2 MPI プログラムで dplace コマンド 34
9.3 dplace のオプション 34
10 デバッグ 35
10.1 デバッグオプション 35
10.2 デバッグツール 35
10.2.1 gdb 35
10.2.2 idb 35
10.2.3 ddd 35
11 科学技術計算ライブラリ 36
11.1 SCSL ライブラリ 36
11.1.1 SCSL 利用方法 36
12 性能解析ツール 37
12.1 lipfpm （Linux IPF Performance Monitor）コマンド 38
12.1.1 MFLOPS 値計測 38
12.1.2 自動並列化／OpenMP プログラムの解析 38
12.1.3 MPI プログラムの解析 40
12.2 histx （HISTogram eXecution)コマンド 42
12.2.1 histx を用いた自動並列化／OpenMP プログラムの解析 42
12.2.2 histx を用いた MPI プログラムの解析 43

12.2.3 histx を用いたコールスタックの解析 43
13 バイナリデータの取り扱い 46
14 各種マニュアルについて 47

1　システム構成

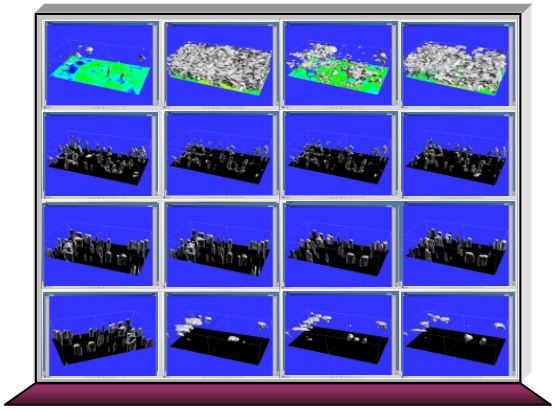
可視化サーバ（ismprsm）のシステム構成を以下に示します。可視化サーバはサーバ本体（SGI Prism）と表示装置で構成されております。表示装置としては、玄関前に設置している 16 面大型表示システムと IBM 社製　T221 モニタがあります。

SGI　Prism

SGI Prism

プロセッサ	: Intel　Itanium2　1.5GHz x16
メモリ	: 32GB　共有メモリ
グラフィックスボード	: ATI　FireGL X2 x8
コンポジタ	: 2 x Scalable Graphics Composito

表示装置



16 面大型表示装置

玄関に設置。最大解像度は 5120x5096 ドット。Prism 上の 8 つのグラフィックスボード 16 チャネルからの出力を描画することが可能。

IBM　T221 グラフィックスモニター

電算室に設置。最大解像度は解像度 3840 x 2400 ドット。コンポジタを利用することにより Prism 上の 4 つのグラフィックスボード 8 チャネルからの出力を描画することも可能。

2　可視化サーバへのログイン

可視化サーバは、IBM　T221、大型表示装置を利用してローカルログインする他、TELNET、SSH を利用したリモートログイン、gdm を利用したグラフィカルリモートログインに対応しています。
可視化サーバのホスト名は以下のとおりです。

ホスト名	IP アドレス
ismprsm.ism.ac.jp	133.58.105.30

3 SGI OpenGL Vizserver の利用方法

OpenGL Vizserver は、可視化サーバのグラフィックス処理性能を手元の PC などで活用できる遠隔可視化用ソフトウェアです。X Window System のリモートウィンドウと違い、描画処理はすべてサーバ側で行われ、結果の画像のみが配信されるため、ノート PC 上で可視化サーバの処理性能を利用することが可能になります。

3.1 OpenGL Vizserver クライアント・ソフトウェアのインストール

OpenGL Vizserver を利用するには、クライアント・ソフトウェアをインストールする必要があります。クライアントは、IRIX、Linux、Windows、Solaris、MacOS X に対応していますが、ここでは、Windows 版のインストール方法について説明します。他のバージョンのインストール方法については、「OpenGL Vizserver User's Guide」をご覧ください。

1. クライアント・ソフトウェアのインストーラを以下の場所から入手します。
   ismprsm: /home/doc/prism/viz/clientsoft/
   利用する OS にあったクライアント・ソフトウエアをダウンロードしてください。

Windows クライアント	SGIviz351.exe
Linuxクライアント	linux_vizserver_client_3.5.1.tar
MacOS Xクライアント	mac_vizserver_client_3.5.1.tar
Solarisクライアント	sollaris_2.x_vizserver_client_3.5.1.tar
IRIXクライアント	6.5.11_vizserver_client_3.5.1.tar

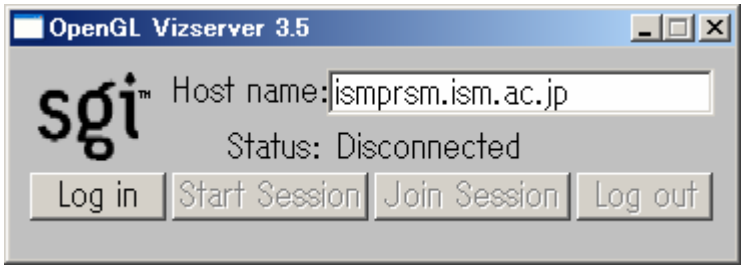
2. インストーラをダブルクリック（Windows の場合）し、インストールを開始します。
3. 途中、SDK の選択画面がありますが、これは独自の圧縮手法を実装する場合にのみ使われるもので、通常は必要ありません。

3.2 OpenGL Vizserver サーバへの接続開始・終了方法

3.2.1 接続開始

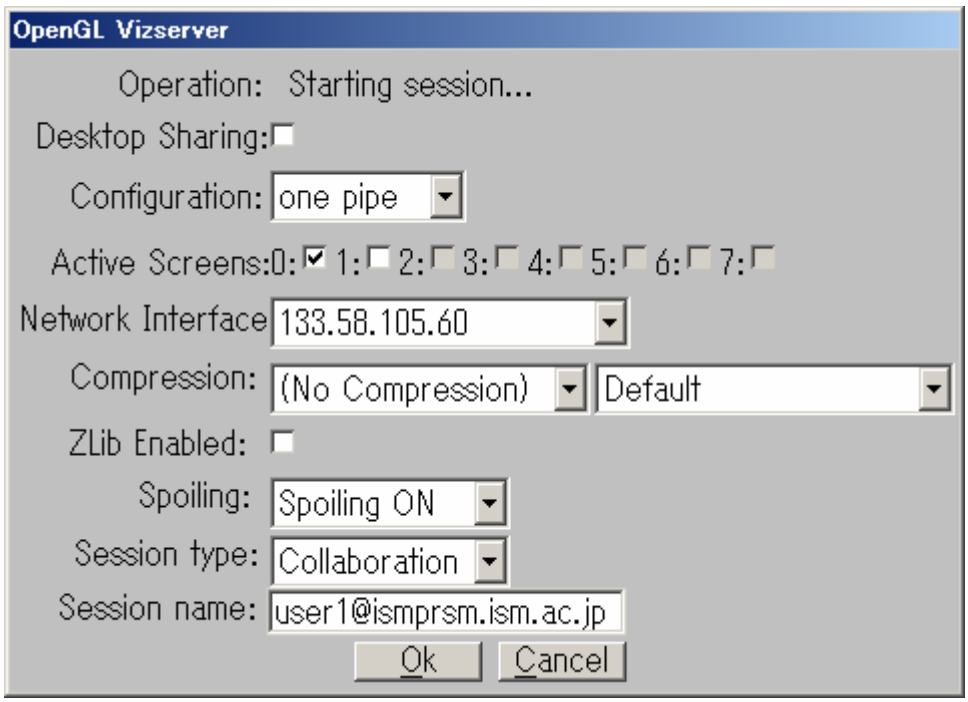
OpenGL Vizserver クライアントを起動するには、スタートメニューから SGI->vizserver を選びます。

最初に以下のようなログイン画面が表示されます。ここに、可視化サーバのホスト名「**ismprsm.ism.ac.jp**」を入力します。



次に、ログイン名とパスワードを入力します。

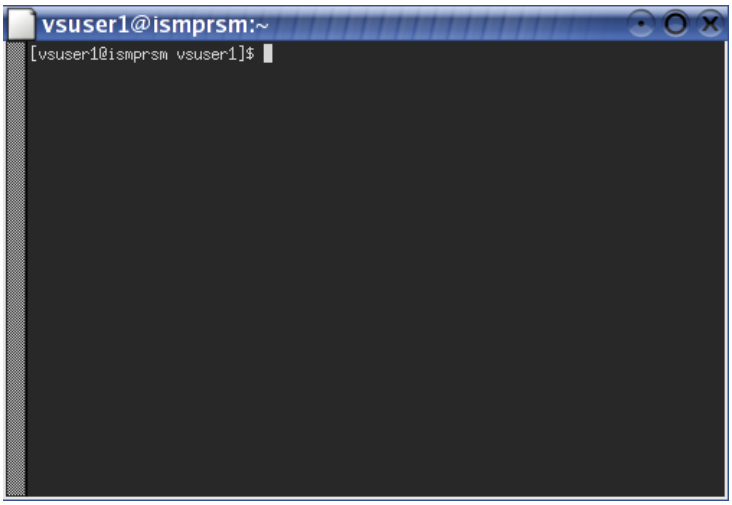
ログイン認証に成功すると、Status が「Connected」になります。
ここで「Start Session」ボタンを押すと、以下のウィンドウが表示されます。



ここでは、以下の項目を選び、最後に OK を押します。

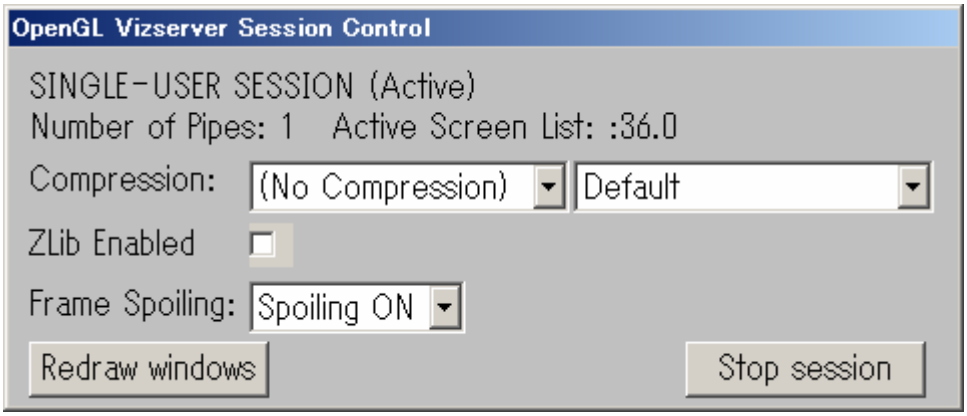
- Configuration: 使用するグラフィックス・パイプの数です。one pipe を選びます。
- Compression: 圧縮率です。No Compression は、ネットワーク転送量が多くなるため、Delta Color Cell の 4:1 または 8:1 を推奨します。
- Collaboration: 他のユーザとコラボレーションをしない場合は、Single を選びます。
  - ➢ コラボレーションの方法については、”OpenGL Vizserver User’s Guide”をご覧ください。

セッションが開始されると、ターミナルウィンドウが表示されます。ここから、アプリケーションを実行します。



3.2.2 接続終了

セッションコントロールウインドウにおいて、「Stop session」ボタンをクリックし接続を終了します。



3.3 利用時の注意点

- 可視化サーバには、4 つまでの同時接続（セッション）が可能なライセンスが設定されています。
- どのセッションでも複数箇所から接続をする「コラボレーション」が可能ですが、システム全体で許されるコラボレーション実行者は 5 人までです。具体的には、以下のような場合が可能です。
  - セッション A は 5 人でコラボレーション、セッション B～D はシングルセッション
  - セッション A は 3 人でコラボレーション、セッション B: 2 人でコラボレーション、セッション C と D はシングルセッション

4 AVS Express PST の利用方法

4.1 AVS/Express について

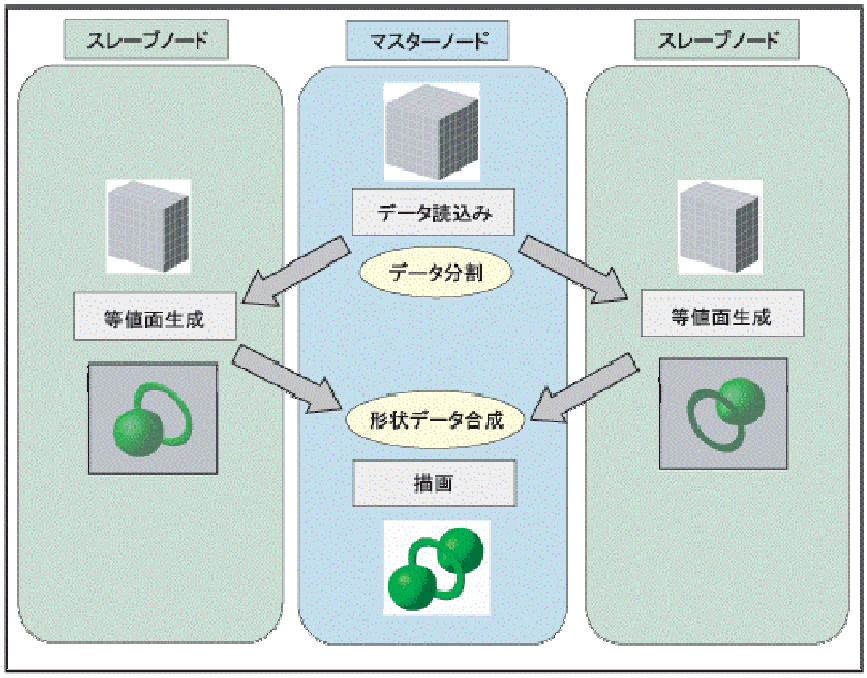
AVS/Express（以下単に AVS と表記します）はシミュレーションや測定データを 2 次元／3 次元可視化するアプリケーションです。

ユーザは AVS のフォーマットに合わせてデータを作成するか、AVS にデータの読み方を指示するファイル（フィールド・ファイル）を作成することで AVS にデータを取り込むことが可能です。可視化機能はマウスで可視化部品（モジュール）を接続するビジュアルプログラミングで行います。一度ビジュアルプログラミングした可視化スクリプト（ネットワークファイル）は保存可能です。

システムには可視化処理の並列化が可能な AVS/Express PST(Parallel Support Toolkit)を最大 5 ユーザが同時利用できる「マルチユーザライセンス」がインストールされています。

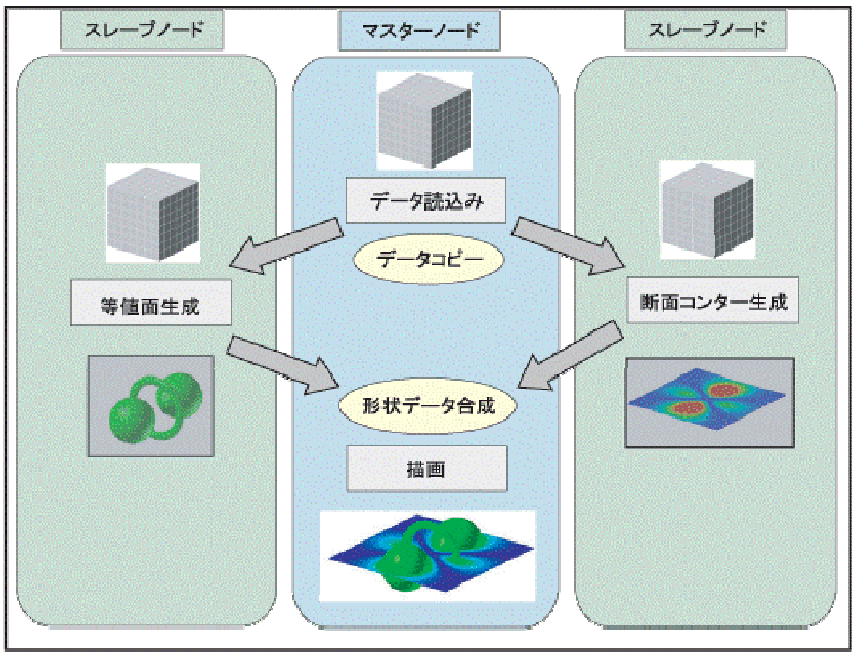
AVS/Express PST の可視化処理の並列化は以下の 4 方法が考えられます。

4.1.1 1つのモジュール内での並列処理



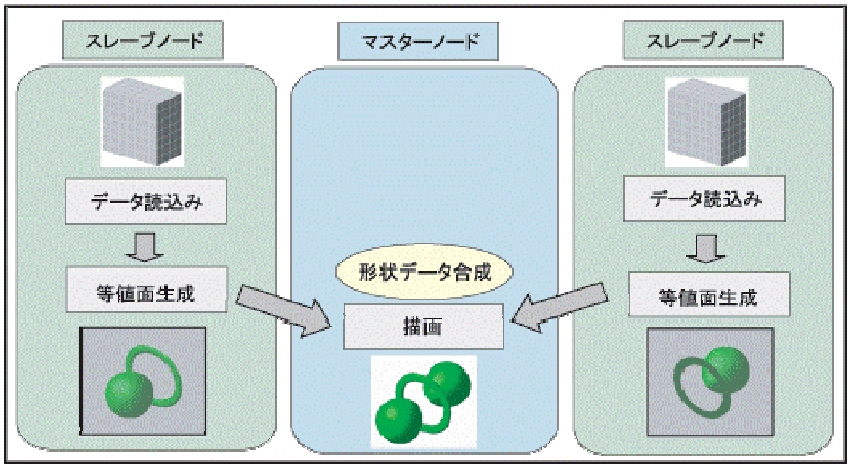
等値面生成等の負荷の高い処理は、データを分割し、複数のノード（共有メモリ型の場合は CPU）で処理をさせることによって、 負荷を軽減します。（注：この例では、各スレーブノードで生成された形状データが、マスターノードで合成され、マスターノードで描画されています。）

4.1.2 モジュール毎の並列処理



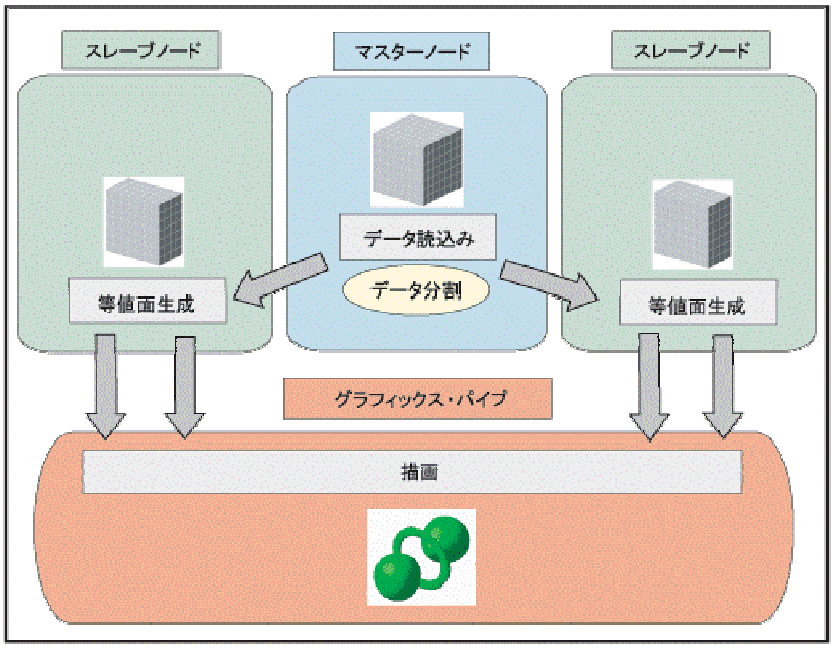
同じデータを複数のモジュールが処理をする場合は、モジュール毎に別のノードで並列処理することによって、 処理を高速化します。また、任意のモジュールを別のノード（またはマシン）で実行されることが可能です。 他のマシンにデータファイルがあるときは、読み込みモジュールをそのマシンで実行させることにより、 ファイル転送が不要になります。

4.1.3 ファイル読み込みの並列化



複数のノード間で分散化されたデータファイルを並列に読み込むことが可能です。 また、ファイル共有の環境では、複数の CPU による同時アクセスにより、 ファイルの読み込み速度を向上させることができます。

4.1.4　レンダリング（描画処理）の並列化



SGI Prism のマルチパイプ・システムでは、複数のグラフィックス・パイプを 使って並列にレンダリングし、その結果を1つのグラフィックス・パイプに合成することにより、 レンダリングが高速化されます。

4.2 AVS のインストール場所と起動

4.2.1 インストール場所

/opt/ExpressPST/
にAVS/Express PST一式がインストールされています。

4.2.2 起動

/opt/ExpressPST/bin/linux_ia64
に起動シェル「xp」があります。起動時の第一引数に数字を指定することで並列実行で AVS/Express PST が起動します。数字は割り振るノードの数になります。
たとえば以下の例では AVS/Express PST は 4 ノードで動作します。

日本語メニューを利用したい場合は起動シェル「xp_jp」をご利用ください。利用方法は xp コマンドと同様です。なお日本語環境は、ご利用の X サーバの環境により表示が乱れることがございます。

```
$ xp 4

AVS/Express Developer Edition
      Version: 7.0 fcs linux_ia64
      Project: /opt/ExpressPST (read-only)
```

4.2.3 並列化

並列化された可視化モジュールのパラメータを変更することで並列実行の動作を設定可能です。
詳細は「AVS/Express PST インストレーションガイド／リリースノート」参照ください。

4.2.4 起動シェル

起動シェル xp の内容は以下のとおりです。AVS/Express PST は mpirun コマンドにより起動されています。

```
#!/bin/sh
# runpst
# James S Perrin (MVC)
# MPICH 2 mpd
# This example script will run the pstmaster (express) and N (default 4) # pstnodes. The mpd must have been started on at least the local machine to run # all the processes locally. A mpd ring should be launched across the cluster # for pstnodes to run remotely. Please see the MPICH2 mpd documentation for # further details.

export MACHINE=linux_ia64
export XP_LANG=C
export LANG=C
export XP_FEATURE=EXPRESS_MPE
export XP_LICENSE_SERVER=ismprsm:33333
export XP_INCLUDE_MAVS=1

NUMBER=$1

expr "$NUMBER" + 1 > /dev/null 2>&1
if [ $? -lt 2 ]; then

export XP_ROOT=/opt/ExpressPST
export LD_LIBRARY_PATH=$XP_ROOT/lib/$MACHINE

#cd $XP_ROOT

mpirun -np 1 $XP_ROOT/bin/$MACHINE/express $2 $3 $4 $5: -np $NUMBER
$XP_ROOT/bin/$MACHINE/pstnode

else

export XP_ROOT=/opt/Express
export LD_LIBRARY_PATH=$XP_ROOT/lib/$MACHINE
echo pst
#cd $XP_ROOT
$XP_ROOT/bin/$MACHINE/express $1 $2 $3 $4 $5

fi
```

4.2.5 AVS の各種マニュアルについて

AVS の一般的な利用方法については以下のマニュアルを参照ください。またテキスト内に記載されている利用方法のサンプルファイルが Zip ファイル形式で圧縮されています。

下記マニュアルは ismprsm:///home/doc/prism/avs 以下に保存されております。

内容	冊子名	PDF ファイル／サンプルファイル
利用の概論（まず初めに）	AVS/Express チュートリアル・ガイド	TutrialGuide70.pdf TutrialGuide_sample70.zip
可視化機能の概論	AVS/Express ユーザーズ・ガイド	UsersGuide70.pdf
可視化モジュール機能の説明	AVS/Express モジュール・リファレンス（第 1 部、第 2 部）	ModuleReference(1)_70.pdf ModuleReference(2)_70.pdf ModuleReference_sample70.zip
モジュール開発	AVS/Express デベロッパーズ・ガイド（第 1 部、第 2 部）	DevelopersGuide(1)_70.pdf DevelopersGuide(2)_70.pdf DevelopersGuide_sample70.zip

5 IDL の利用方法の利用方法

5.1 IDL について

IDL＜Interactive Data Language＞は地球科学、宇宙科学、医用画像、数値解析、実験・計測データ解析シミュレーションなど様々な分野で使用されているソフトウェアです。 IDL はデータ解析、可視化、アプリケーション開発などを含んだ統合環境を提供します。また IDL が持つ IDL 言語は対話形式（インタプリタ）、実行形式（テキスト、バイナリ）双方が可能です。 また IDL で作成されたアプリケーションは IDL がサポートする OS（Windows、Linux、UNIX、MacOS X）で同じように動作する上、無償の実行環境（IDL Virtual Machine）も提供されるので、 マルチ OS 用アプリケーション開発から配布まで IDL だけで一貫して行えます。

5.1.1 iTools

iTools は IDL が持つオブジェクト指向のツール群です。ユーザは iTools を利用することによりほとんどの操作（データ入力、表示、画像処理、回転、拡大、属性の変更など）をマウスだけで行え、IDL が持つオブジェクト機能をすぐに使用できます。また iTools はオープンソースコードとなっており、ユーザがこれら機能をカスタマイズして利用することもできます。

5.1.2 アプリケーション開発

IDL は 600 以上の関数、プロシージャを持ち、アプリケーション開発に必要なツール群（GUI 作成ツール、コンパイラなど）も全て１つにパッケージ化されています。また、他プログラミング言語とリンクすることが可能であり、IDL プログラムから FORTRAN、C、JAVA で作成されたルーチンを呼び出すことや、FORTRAN、C、Visual Basic、VC++、Delphi で作成されたプログラムから IDL で作成されたルーチンを呼び出すこともできます。これにより、過去のプログラム資産を生かしながら、統合的なアプリケーション開発を行うことができます。

5.1.3 対話的処理

コマンドインタープリタ形式で、演算子と関数を配列全体に対して適用でき、即時コンパイルと IDL コマンドの実行によって、ただちにフィードバックが得られ、対話形式の解析が容易になり、プログラミングの時間を短縮することができます。

5.1.4 データ I/O

各種画像フォーマット、ASCII/バイナリ、データ構造(scalar, vector, array)、科学技術フォーマット（HDF、HDF-EOS、CDF、NetCDF）等を標準でインプット・アウトプットが可能です。

5.2 IDL のインストール場所と起動

5.2.1 インストール場所

/opt/rsi/

にIDL一式がインストールされています。

5.2.2 起動

idl コマンドにて起動します。

```
$ idl
```

5.3 IDL の各種マニュアルについて

IDL の一般的な利用方法については以下のマニュアルを参照ください。下記マニュアルに関しては

ismprsm://doc/prism/idl

以下に保存しております。

内容	冊子名	PDF ファイル
Version6.2 新機能	What's New in IDL6.2（英語） IDL6.2 Release Notes(英語) IDS6.2 リリースノート（日本語）	IDL 6.2_relnotes.pdf（日本語）
IDL の利用概論（チュートリアル）	Getting Started with IDL6.0 Getting Started with IDL（日本語）	IDL61_getstart.pdf（日本語）
IDL の利用方法	Using IDL v6.0	
IDL クイックリファレンス	IDL Quick Reference Guide	
IDLV6.2 リファレンスガイド	IDL Reference Guides V6.2 （英語）	
IDL アプリケーション開発	Building IDL Applications（英語）	
iTools の利用	iTools User Guide v6.2（英語）	iTool_Users_Guide_jp.pdf （日本語）
iTools 開発	iTools Developer's Guide（英語）	

6 ISL の利用方法の利用方法

ISL（Interactive Stereo Library）は、OpenGL アプリケーションを裸眼立体視ディスプレイ対応に改造するためのライブラリです。
ISL を使用することによって、利用者は、OpenGL アプリケーションを複数ベンダの裸眼立体視ディスプレイに対応させることができます。ISL を使用した OpenGL アプリケーションは、エンドユーザが環境変数を設定することにより、裸眼立体視ディスプレイで立体表示することが可能です。
裸眼立体視環境と致しましては裸眼表示用ノートパソコン（Sharp 製 Mebius）があります。こちらの利用に関しては統計数理研究所統計科学技術センター（mail to: cdsc@ism.ac.jp）までご連絡願います。

ISL の基本機能は、アプリケーションの描画関数を複数回呼んで多視点の裸眼立体視ディスプレイ用の表示画像を合成するというものです。ISL は、各視点の描画を行う前に、裸眼立体視フィルタのマスクパターンに合わせて描画マスクを設定します。
アプリケーション側から見た場合、ISL を使用方法としてコールバック方式、アプリケーション制御ループ方式の二つの異なった方法があります。図 1、図 2 にそれぞれの方式の呼び出しの関係を図示します。

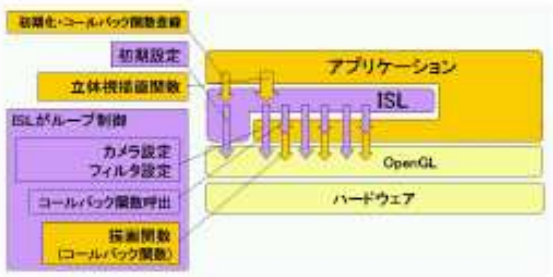


図 1. コールバック方式のソフトウェア呼び出し関係

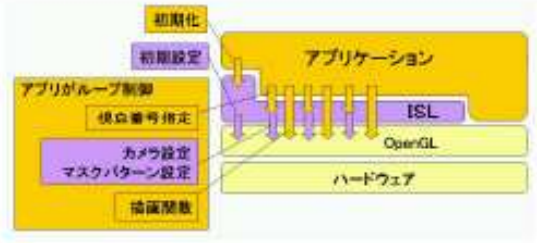


図 2. アプリケーション制御ループ方式のソフトウェア呼び出し関係

なお、ISL の具体的な利用方法に関しましては ISL 利用マニュアルをご参照ください。マニュアルは電子ファイル（PDF）で提供します。参照方法に関しましては本資料の「各種マニュアルについて」をご参照願います。

7 コンパイラ

可視化サーバでは以下のコンパイラが利用可能です。

7.1.1 Intel Compiler

Intel Fortran Compiler は Fortran 77 90 95 をサポートしますので、どの規格に沿って書かれたソースプログラムでもコンパイル可能です。Fortran95 は Fortran90 に対して、Fortran90 は Fortran77 に対して上位互換であるため、それぞれの規格が混在しているようなソースプログラムもコンパイルできます。

コマンド	内容
ifort	Fortran 77 90 95 をサポート
icc	C C++ をサポート

7.1.2 GNU コンパイラ

GNU コンパイラは、GNU プロジェクトで開発されたコンパイラです。GNU コンパイラでは以下の言語が利用可能です。

コマンド	内容
gcc	C をサポート
g++	C++ をサポート
g77	Fortran77 をサポート

最良のパフォーマンスを得るために Intel Compiler を推奨致しますが、GNU ツールである g77、gcc、g++もお使い頂けます。このガイドでは、Intel Compiler ifort、icc のオプションをご紹介します。尚、--help オプションを指定して頂くとコンパイラオプションの一覧が表示されます。

8 Intel コンパイラにおけるプログラムの最適化方法

8.1 シングルプロセッサプログラムの最適化方法

本章では、Intel コンパイラでのシングルプロセッサプログラムの最適化についてご説明します。まず正しい結果が得られていることを確認してください。他のシステムからプログラムを移植する場合には、結果の妥当性に対して検討が必要です。一般には、プログラムが高級言語で書かれていて、それらが言語規格に正しく準拠している場合には、その移植には何ら問題はありません。デバッグの方法については「デバッグ」を参照してください。

8.1.1 データ型

Intel コンパイラのデータ型の一覧を以下に示します。

C	bit	Fortran
char	8	CHARACTER
short int	16	INTEGER(KIND=2)
int	32	INTEGER(KIND=4)
long int	64	-
long long int	64	INTEGER(KIND=8)
pointer	64	POINTER
float	32	REAL(KIND=4)
double	64	REAL(KIND=8)/double precision
long double	128	REAL(KIND=16)

8.1.2 オプション

以下に Intel コンパイラの各種オプションについて説明します。

8.1.2.1 最適化、プロセッサ特化、浮動小数点演算関連のオプション

オプション	内容
最適化レベルオプション	
-O0	すべての最適化を無効にします。デバッグを行うときに指定してください。
-O1	保守的な最適化を行います。
-O2	ソフトウェアパイプラインによる最適化を行います。多くの場合有効です。デフォルトの最適化レベルです。最適化のオプションを何も指定しないとこのオプションで最適化されます。
-O3	積極的な最適化を行います。-O2 の最適化に加えて、ループの交換やデータプリフェッチを行います。
浮動小数点演算に関するオプション	
-ftz	アンダーフローが発生したときに、値をゼロに置き換えます。このオプションは指定しなくともデフォルトで付加されます。

8.1.2.2 プロシージャ間解析オプション(IPO)

インライン展開を有効にします。プログラムのなかで小さいサイズの関数を何回も呼び出しているような場合に性能向上の効果があります。関数コールを含むループの最適化にも効果的です。

オプション	内容
-ip	1つのソースファイルにあるプロシージャ間の解析、最適化を行います。
-ipo	複数のソースファイルにあるプロシージャ間の解析、最適化を行います。リンク時にもオプションとして指定してください。

コンパイル例

```
% ifort  -c  -ipo  exm1.f
```

リンク例

```
% ifort  -o  exm1  -ipo  exm1.o
```

8.1.2.3 エイリアス解析オプション

プログラム中の異なるポインタ変数が同じメモリを指している可能性があるとき、コンパイラは積極的な最適化を行うことができません。エイリアスがないとわかっている場合には次のオプションを指定してください。

-fno-alias

どの 2 つのポインタも決して同じメモリ領域を指すことはない（エイリアスがない）とコンパイラは仮定します。積極的な最適化の可能性が増えます。

例）

```
int x
foo(int *p, int *q)
{
   while (q!= NULL)
   {
      x+= *p * q->cost;
      q = q->next;
   }
}
```

変数 x を更新することによって、*p も更新される可能性があります（コンパイラには判断できない）。しかし、プログラマがエイリアスはないと判断できれば-fno_alias を指定します。コンパイラは積極的な最適化を行うことができます。

8.1.2.4 最適化レポートオプション

オプション	内容
-opt_report	最適化レポートを標準エラー出力に表示
-opt_report_file ファイル名	最適化レポートを指定したファイルに出力

次ページに最適化レポートを使用した場合の例を示します。

例）

```
% cat ex1.f
       1            program ex1
       2            parameter(n=500)
       3            real a(n,n),b(n,n),c(n,n)
       4            common /data/a,b,c
       5
       6            do i=1,n
       7            do j=1,n
       8               a(i,j)=i+k
       9               b(i,j)=i+j+k
      10               c(i,j)=0.0
      11            enddo
      12            enddo
      13            time1=dclock()
      14            do i=1,n
      15            do j=1,n
      16            do k=1,n
      17               c(i,j)=c(i,j)+a(i,k)*b(k,j)
      18            enddo
      19            enddo
      20            enddo
      21
      22            print *,c(n,n)
      23            stop
      24            end

% ifort -opt_report ex1.f
================================================================================
SWP REPORT LOG OPENED ON Fri Feb 14 04:22:15 2003
================================================================================
Optimization Report for: main()
Phase : Extend Insertion

% ifort -O3 -opt_report ex1.f

================================================================================
SWP REPORT LOG OPENED ON Fri Feb 14 04:22:29 2003
================================================================================
HLO REPORT LOG OPENED ON Fri Feb 14 04:22:29 2003
================================================
Total #of lines prefetched in main for loop in line 7=3
#of Array Refs Scalar Replaced in main at line 16=28
Total #of lines prefetched in main for loop in line 16=8
#of Array Refs Scalar Replaced in main at line 16=3
Total #of lines prefetched in main for loop in line 16=5
Total #of lines prefetched in main for loop in line 16=2

LOOP INTERCHANGE in main at line 6
LOOP INTERCHANGE in main at line 7
LOOP INTERCHANGE in main at line 14
LOOP INTERCHANGE in main at line 15
LOOP INTERCHANGE in main at line 16
          /* －O3 のとき、ループの最適化を行います。 */
Block, Unroll, Jam Report:
(loop line numbers, unroll factors and type of transformation)
Loop at line 7 unrolled without remainder by 4
Loop at line 14 unrolled and jammed by 4
Loop at line 15 unrolled and jammed by 4
```

8.1.3 makefile について

以下に makefile の例を示します。

```
SHELL= /bin/csh
BIN        = a.out
FFLAGS   = -c
CFLAGS   = -c
LFLAGS   =
LIB        = -lscs   ー lmpi
F90        = ifort
CC          = icc

OBJ= ¥
main.o      routine1.o      routine2.o      prog.o

$(BIN): $(OBJ)
            $(F90) $(LFLAGS) -o $(BIN) $(OBJ) $(LIB)

.f.o:
            $(F90) $(FFLAGS) $*.f

.c.o:
            $(CC) $(CFLAGS) $*.c

clean:
            -rm -f core $(BIN) $(OBJ)
```

Intel Compiler のコンパイラオプションはデフォルトで最適化を進めるようなレベルが設定されています。そのため、特にオプションを指定しなくてもコンパイラが最適化した実行モジュールを生成します。デフォルトのコンパイラオプションの内容は「最適化オプション」をご参照ください。

マクロ名 LIB で指定しているのは、科学技術計算ライブラリ（SCSL）と MPI のリンクです。SCSL ライブラリについては「科学技術計算ライブラリ」を、MPI については「MPI」をご参照ください。

8.2 並列プログラムの最適化方法

本章では、Intel コンパイラでのシングルプロセッサプログラムの最適化についてご説明します。

8.2.1 自動並列化

Intel コンパイラでは「-parallel」オプションの指定により、コンパイラがソースコード内のループに対して、自動並列化の解析を行います。

オプション	内容
-parallel	自動並列化の指定です。
-par_report{0|1|2|3}	並列化されたループを表示し、並列化されなかったループについてはなぜ並列化されなかったかを説明します。末尾の数字が大きいほど詳細な情報を出力します。
-override_limits	1300 行を越える大きなソースコードで自動並列化オプション -parallel を指定するときには、このオプションも一緒に指定してください。指定しない場合、コンパイルが止まってしまうこともあります。

次ページの自動並列化、並列化解析オプションの例を参照してください。

例）

```
% cat ex8.f
1.            program ex8
2.            parameter(n=1000)
3.            real a(n,n),b(n,n)
4.            do j=1,n
5.            do i=1,n
6.               a(i,j)=1.
7.               b(i,j)=1.
8.            enddo
9.            enddo
10.           do j=1,n
11.           do i=1,n
12.              a(i,j)=a(i,j)*b(i,j)
13             enddo
14.           enddo
15.           do j=1,n
16.           do i=1,n
17.              print *,a(i,j)
18.           enddo
19.           enddo
20.           stop
21.           end

% ifort -parallel -par_report3 ex8.f

    serial loop: line 5: not a parallel candidate due to insufficent work
    serial loop: line 16: not a parallel candidate due to statement at line 17
    serial loop: line 15: not a parallel candidate due to statement at line 17
       /*　並列化できなかったループについて、その理由が出力されます。*/
ex8.f(4) : (col. 0) remark: LOOP WAS AUTO-PARALLELIZED.
    parallel loop: line 4
         shared: {"a", "b"}
         private: {"j", "i"}
         first private: { }
         reductions: { }
ex8.f(10) : (col. 0) remark: LOOP WAS AUTO-PARALLELIZED.
    parallel loop: line 10
         shared: {"a", "b"}
         private: {"j", "i"}
         first private: { }
         reductions: { }
21 Lines Compiled
```

8.2.2 OpenMP プログラムのコンパイル

Intel コンパイラでは「-openmp」オプションの指定により、ソースコード内の OpenMP 指示行を有効にします。

サンプルプログラム 1（Fortran）

```
% cat ex9.f
1.            program ex9
2.            parameter(n=1000)
3.            real a(n,n),b(n,n)
4.            do j=1,n
5.            do i=1,n
6.               a(i,j)=1.
7.               b(i,j)=1.
8.            enddo
9.            enddo
10.       !$OMP PARALLEL DO
11.           do j=1,n
12.           do i=1,n
13.              a(i,j)=a(i,j)*b(i,j)
14.           enddo
15.           enddo
16.           do j=1,n
17.           do i=1,n
18.              print *,a(i,j)
19.           enddo
20.           enddo
21.           stop
22.           end
% ifort -openmp ex9.f
ex9.f(10) : (col. 0) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
23 Lines Compiled
    %
```

サンプルプログラム 2(C)

```
1.#include<stdio.h>
2.
3.#define n 1000
4.
5.int main(void)
6.{
7.          float a[n][n],b[n][n];
8.          int i,j;
9.          for (i=0;i<n;i++){
10.                   for (j=0;j<n;j++){
11.                             a[i][j]=1;
12.                             b[i][j]=1;
13.                   }
14.         }
15.
16.#pragma omp parallel for
17.           for (i=0;i<n;i++){
18.                   for (j=0;j<n;j++){
19.                             a[i][j]=a[i][j]*b[i][j];
20.                   }
21.         }
22.
23.           for (j=0;j<n;j++){
24.                   for (i=0;i<n;i++){
25.                             b[i][j]=a[i][j];
26.                   }
27.           }
28.}

% icc -openmp ex9.c
ex9.c(16) : (col. 1) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
```

-openmp と-parallel を同時に指定した場合

-openmp と-parallel を同時に（同じソースファイルに）指定した場合、-parallel オプションは OpenMP 指示行を含まないルーチンにだけ有効になります。OpenMP 指示行を含むルーチンでは-openmp だけが有効です。

指定オプション	OpenMP 指示行のあるルーチン	OpenMP 指示行のないルーチン
-openmp	指示行のあるループのみ並列化。その他のループは並列化しない（自動並列化なし）	並列化しない
-parallel	指示行は無効。全てのループに対して自動並列化を試みる	全てのループに対して自動並列化を試みる
-openmp と-parallel	指示行のあるループのみ並列化。その他のループは並列化しない（自動並列化なし）	全てのループに対して自動並列化を試みる

例） -openmp と-parallel を指定した場合(Fortran)(% ifort -openmp -parallel ex10.f)

```
1.          program ex10
2.          parameter(n=1000)
3.          real a(n,n),b(n,n)
4.          do j=1,n
5.          do i=1,n
6.             a(i,j)=1.
7.             b(i,j)=1.
8.          enddo
9.          enddo
10.           call comp(a,b,n)
    … … … …
20.         subroutine comp(a,b,n)
21.         integer n
22.         real a(n,n),b(n,n)
23.      !$OMP PARALLEL DO
24.             do j=1,n
25.             do i=1,n
26.                 a(i,j)=a(i,j)*b(i,j)
27.             enddo
28.             enddo
29.             do j=1,n
30.             do i=1,n
31.                 b(i,j)=a(i,j)
32.             enddo
33.             enddo
34.             return
35.             end

ex10 メインルーチン          :4 行目のループは自動並列化
comp ルーチン                :24 行目のループは-openmp による並列化
                             :29 行目のループは並列化されない
```

例） -openmp と-parallel を指定した場合(C)(% ifort -openmp -parallel ex10.c)

```
1. #include<stdio.h>
2.
3. #define N 1000
4.
5. float comp(float a[N][N],float b[N][N]);
6.
7. int main(void)
8. {
9.              int i,j;
10.             float a[N][N],b[N][N];
11.             for (i=0;i<N;i++){
12.                             for (j=0;j<N;j++){
13.                                             a[i][j]=1+i;
14.                                             b[i][j]=1+j;
15.                             }
16.             }
17.             comp(a,b);
18.     … … … …
30. }
31. float comp(float a[N][N],float b[N][N])
32. {
33.             int i,j;
34. #pragma omp parallel for
35.              for (i=0;i<N;i++){
36.                             for (j=0;j<N;j++){
37.                                             a[i][j]=a[i][j]*b[i][j];
38.                             }
39.             }
40.             for (i=0;i<N;i++){
41.                             for (j=0;j<N;j++){
42.                                             b[i][j]=a[i][j];
43.                             }
44.             }
45. }
```

ex10 メインルーチン　　:11 行目のループは自動並列化
comp ルーチン　　:35 行目のループは-openmp による並列化
　　:40 行目のループは並列化されない

例） -openmp だけを指定した場合(Fortran)(% ifort -openmp ex10.f)

```
1.          program ex10
2.          parameter(n=1000)
3.          real a(n,n),b(n,n)
4.          do j=1,n
5.          do i=1,n
6.             a(i,j)=1.
7.             b(i,j)=1.
8.          enddo
9.          enddo
10.           call comp(a,b,n)
    … … … …
20.         subroutine comp(a,b,n)
21.         integer n
22.         real a(n,n),b(n,n)
23.       !$OMP PARALLEL DO
24.               do j=1,n
25.               do i=1,n
26.                   a(i,j)=a(i,j)*b(i,j)
27.               enddo
28.               enddo
29.               do j=1,n
30.               do i=1,n
31.                   b(i,j)=a(i,j)
32.               enddo
33.               enddo
34.               return
35.               end

ex10 メインルーチン : 4 行目のループは並列化されない
comp ルーチン                    : 24 行目のループは-openmp による並列化
                                 : 29 行目のループは並列化されない
```

例） -openmp だけを指定した場合(C)(% ifort -openmp ex10.c)

```
 1. #include<stdio.h>
 2.
 3. #define N 1000
 4.
 5. float comp(float a[N][N],float b[N][N]);
 6.
 7. int main(void)
 8. {
 9.             int i,j;
10.             float a[N][N],b[N][N];
11.             for (i=0;i<N;i++){
12.                         for (j=0;j<N;j++){
13.                                     a[i][j]=1+i;
14.                                     b[i][j]=1+j;
15.                         }
16.             }
17.             comp(a,b);
18.     … … … …
30. }
31. float comp(float a[N][N],float b[N][N])
32. {
33.             int i,j;
34. #pragma omp parallel for
35.              for (i=0;i<N;i++){
36.                         for (j=0;j<N;j++){
37.                                     a[i][j]=a[i][j]*b[i][j];
38.                         }
39.             }
40.             for (i=0;i<N;i++){
41.                         for (j=0;j<N;j++){
42.                                     b[i][j]=a[i][j];
43.                         }
44.             }
45. }
```

ex10 メインルーチン　　:11 行目のループは並列化されない
comp ルーチン　　:35 行目のループは-openmp による並列化
　　:40 行目のループは並列化されない

例） -parallel だけを指定したとき(Fortran)(% ifort -parallel ex10.f)

```
1.          program ex10
2.          parameter(n=1000)
3.          real a(n,n),b(n,n)
4.          do j=1,n
5.          do i=1,n
6.             a(i,j)=1.
7.             b(i,j)=1.
8.          enddo
9.          enddo
10.           call comp(a,b,n)
    … … … …
20.         subroutine comp(a,b,n)
21.         integer n
22.         real a(n,n),b(n,n)
23.       !$OMP PARALLEL DO
24.             do j=1,n
25.             do i=1,n
26.                 a(i,j)=a(i,j)*b(i,j)
27.             enddo
28.             enddo
29.             do j=1,n
30.             do i=1,n
31.                 b(i,j)=a(i,j)
32.             enddo
33.             enddo
34.             return
35.             end

ex10 メインルーチン :4 行目のループは自動並列化
comp ルーチン                    :24 行目のループは自動並列化
                                 :29 行目のループは自動並列化
```

例） -parallel だけを指定したとき(C)(% icc -parallel ex10.c)

```
 1. #include<stdio.h>
 2.
 3. #define N 1000
 4.
 5. float comp(float a[N][N],float b[N][N]);
 6.
 7. int main(void)
 8. {
 9.             int i,j;
10.             float a[N][N],b[N][N];
11.             for (i=0;i<N;i++){
12.                         for (j=0;j<N;j++){
13.                                     a[i][j]=1+i;
14.                                     b[i][j]=1+j;
15.                         }
16.             }
17.             comp(a,b);
18.     … … … …
30. }
31. float comp(float a[N][N],float b[N][N])
32. {
33.             int i,j;
34. #pragma omp parallel for
35.              for (i=0;i<N;i++){
36.                         for (j=0;j<N;j++){
37.                                     a[i][j]=a[i][j]*b[i][j];
38.                         }
39.             }
40.             for (i=0;i<N;i++){
41.                         for (j=0;j<N;j++){
42.                                     b[i][j]=a[i][j];
43.                         }
44.             }
45. }
```

ex10 メインルーチン　　:11 行目のループは自動並列化
comp ルーチン　　:35 行目のループは自動並列化
　　:40 行目のループは自動並列化

8.2.3 MPI プログラムのコンパイル

MPI ライブラリのリンクを指示してください。

```
% ifort   ex_mpi.f   -lmpi
```

（実行時には mpirun コマンドを使ってジョブを起動してください。）

8.2.3.1 MPI のデータ型

Fortran プログラムでの MPI のデータ型とそれに対応する Fortran でのデータの型は下記の表の通りです。

MPI Data type	Fortran Data type	データ長（バイト）
MPI_DATATYPE_NULL	対応する型はありません	-
MPI_INTEGER	integer	4
MPI_REAL	real	4
MPI_DOUBLE_PRECISION	double precision	8
MPI_COMPLEX	complex	8
MPI_DOUBLE_COMPLEX	double complex	16
MPI_LOGICAL	logical	4
MPI_CHARACTER	character	1
MPI_INTEGER1	integer*1	1
MPI_INTEGER2	integer*2	2
MPI_INTEGER4	integer*4	4
MPI_INTEGER8	integer*8	8
MPI_REAL4	real*4	4
MPI_REAL8	real*8	8
MPI_REAL16	real*16	16

C プログラムでの MPI のデータ型とそれに対応する C でのデータの型は下記の表の通りです。

MPI Data type	Fortran Data type	データ長（バイト）
MPI_CHAR	char	1
MPI_SHORT	short	2
MPI_INT	int	4
MPI_LONG	long	8
MPI_UNSIGNED_CHAR	unsigned char	1
MPI_UNSIGNED_SHORT	unsigned short	2
MPI_UNSIGNED	unsigned int	4
MPI_UNSIGNED_LONG	unsigned long	8
MPI_FLOAT	float	4
MPI_DOUBLE	double	8
MPI_LONG_DOUBLE	long double	16

サンプルプログラム

例）Fortran プログラム

```
% cat simple1.f
            program simple1
            include 'mpif.h'
            call mpi_init(istat)
            call mpi_comm_size(mpi_comm_world, num_procs, ierr)
            call mpi_comm_rank(mpi_comm_world, my_proc, jerr)
            if (my_proc .eq. 0)
        &         write(6,1) 'I am process ',myproc,'.   Total number of proce
        &sses: ',num_procs
            format(a,i1,a,i1)
            call mpi_finalize(ierr)
            stop
            end
% ifort simple1.f -lmpi

% mpirun -np 4 dplace -s1 ./a.out
I am process 0.   Total number of processes: 4
```

例）C プログラム

```
% cat simple1.c
#include <mpi.h>
#include <stdio.h>
main(argc, argv)
int         argc;
char       *argv[];
{
            int         num_procs;
            int         my_proc;

            MPI_Init(&argc, &argv);
            MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
            MPI_Comm_rank(MPI_COMM_WORLD, &my_proc);

            if (my_proc == 0)
                        printf("I am process %d.   Total number of processes: %d¥n",
                                    my_proc,num_procs);
            MPI_Finalize();
}
% icc -w simple1.c -lmpi
% mpirun -np 4 dplace -s1./a.out
I am process 0.   Total number of processes: 4
```

8.3　並列ループのデータ分散

共用計算サーバでは、データ配置に「ファーストタッチ」を採用しています。ファーストタッチとは、データは、最初にそのデータにタッチ（書き込みまたは読み込み）したプロセスのローカルメモリに配置されるというものです。もし、配列（配列 A とする）を初期化しているループが並列化されていないと 1CPU 実行であるために、配列 A はある1つのノードに配置されます。配列 A を処理する並列実行領域では、複数のプロセッサが1つのノードへアクセスすることになります。このような状況ではプログラムの性能向上は望めません。可能な限り初期化ループも並列化してください。

例）

```
real*8    A(n), B(n), C(n), D(n)
!$omp parallel do private(i)
      do    i=1, n
         A(i) = 0.
         B(i) = i/2
         C(i) = i/3
         D(i) = i/7
      enddo
!$omp parallel do private(i)
      do    i=1, n
      A(i) = B(i) + C(i) + D(i)
      enddo
```

計算をしている 2 番目のループを並列化するときは、配列を初期化している1番目のループも並列化すれば、より性能が向上します。

9　並列プログラムの実行：dplace コマンド

dplace コマンドは、複数のプロセスをアイドル状態の CPU にラウンドロビン方式で割り当てます。並列プログラムの実行時には、dplace コマンドを指定されることを推奨します。自動並列化／OpenMP プログラムと MPI プログラムでオプションが異なりますのでご注意ください。数字のあるオプションですが、実行 CPU 数がいくつであっても変わりません。

9.1　自動並列化／OpenMP プログラムでの dplace コマンド

環境変数 OMP_NUM_THREADS に実行プロセス数を指定してください。デフォルトでは dplace コマンドの-x2 オプションを使用してください。

例）

```
% setenv OMP_NUM_THREADS 4
% dplace   -x2 ./a.out
```

9.2　MPI プログラムで dplace コマンド

dplace コマンドの-s1 オプションを使用してください。

例）

```
% mpirun -np 4 dplace   -s1 ./a.out
```

9.3　dplace のオプション

dplace のオプション、-x -s ともに CPU に配置しないプロセスを指定するものです。

オプション	内容
-x	CPU に配置しないプロセスを指定します。自動並列化／OpenMP プログラムを実行すると、OMP_NUM_THREADS で指定した数のプロセスの他に 2 つの管理プロセスが起動されます。-x2 オプションはこの 2 つの管理プロセス以外のプロセスを CPU に割り当てることを指示するものです。
-s	-s オプションを指定すると最初の n 個のプロセスを CPU に配置しません。-s1 オプションで MPI プログラムは mpirun で N+1 個のプロセスが起動されます。最初のプロセスは実際にはインアクティブな状態です。-s1 を指定してインアクティブなプロセスを CPU に割り当てません。

10 デバッグ

10.1 デバッグオプション

オプション	内容
-mp, -mp1	IEEE754 規格に則った浮動小数点演算コードを生成します。例えば、-mp オプションは乗加算命令を採用しません。-mp1 オプションは-mp オプションよりも最適化を進めます。
-IPF_fltacc	浮動小数点の精度を保つような最適化を行います。つまり、精度を落とす可能性がある最適化は無効にします。
-ftz	ゼロ割などの例外処理で不正な値になったとき、値をゼロにして計算を続行します。注意：最適化オプション-O2 のときは-ftz は無効ですが、－O3 で有効になります。例外処理の発生するようなプログラムでは、-O2 ではエラーになるが、-O3 ではエラーにならないという場合があります。
-r8	real/compelx 型で宣言された変数を real*8/complex*16 型の変数として取り扱います。
-i8	integer 型で宣言された変数を integer*8 型の変数として取り扱います。

10.2 デバッグツール

デバッグツールをお使いになる場合には、コンパイル時に「-g」オプションをご指定ください。-g を指定するとデバッグシンボルを有効にします。

10.2.1 gdb

GNU プロジェクトのデバッガです。C、C++、Fortran95 で書かれたプログラムのデバッグに使用できます。

10.2.2 idb

Intel 製のデバッガです。C、C++、Fortran77、Fortran90 で書かれたプログラムのデバッグに使用できます。現時点では並列プログラムのデバッグには使用できません。詳細はマニュアルをご覧ください。

10.2.3 ddd

コマンドラインデバッガに接続する GUI です。起動後、画面の各ペインに次の情報が表示されます。

- 配列表示
- ソースコード
- 逆アセンブルコード
- コマンドを入力するウィンドウ

11 科学技術計算ライブラリ

11.1 SCSL ライブラリ

科学技術計算ライブラリとして SCSL ライブラリをご利用頂けます。

以下に SCSL に含まれる内容を示します。

ライブラリ名	内容
BLAS	基本的な線形計算
LAPACK	密行列計算パッケージ
FFT	1 次元、2 次元、3 次元の FFT
psldlt, psldu	直接法スパースソルバー

11.1.1 SCSL 利用方法

コンパイル時に以下のように SCSL のオプションを付加し、SCSL ライブラリをリンクさせてください。

オプション	内容
-lscs	SCSL 逐次版をリンクします。
-lscs_mp	SCSL 並列版をリンクします。

例）

```
% cat ex7.f
      program ex7
    integer          M, N, LDA, NRHS, LDB, INFO
    character      TRANS
      parameter     (M=3, N=3, LDA=3, NRHS=1, LDB=3, TRANS='N')
      integer        IPVT(n)
      real             A (LDA,LDA), B(LDB), X(n)
C       A=( 1.0       3.0       3.0       )
C             1.0       3.0       4.0       )
C             1.0       4.0       3.0       )
C
C             B=( 1.0
C                 4.0
C                -1.0     )
data     A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
data     B/1.0, 4.0, -1.0/
C     compute an LU factorization of a general M-by-N matrix A
        call    SGETRF(M, N, A, LDA, IPVT, INFO)
C     solve a system of linear equations A*X=B
        call    SGETRS(TRANS, N, NRHS, A, LDA, IPVT, B, LDB, INFO)
        do    i=1,n
            print *, B(I)
        enddo
        stop
        end
% ifort ex7.f -lscs
% ./a.out
-2.000000
-2.000000
 3.000000
```

なお、SCSL ライブラリの詳細に関しては、SCSL ライブラリのマニュアルをご参照ください

12 性能解析ツール

プログラムの性能向上を妨げている要因を知りたいとき、プログラムのどこで実行時間がかかっているのかを知りたいときなどに有用な2つの性能解析ツールをご紹介します。どちらも実行時に指定しますと、実行終了後に解析結果を出力します。

lipfpm コマンドは、プログラム全体のふるまいを調べるときにお使い頂けます。命令の実行、変数のロード／ストア、キャッシュミスなどの現象をイベントと呼んでいますが、lipfpm はプログラム実行全体のイベントカウントを知るためのツールです。

histx コマンドはプログラムのどのルーチンで実行時間がかかっているのか、あるいはソースコードのどの行の実行に時間がかかっているのかを調べるときにお使い頂けます。

12.1 lipfpm （Linux IPF Performance Monitor）コマンド

Itanium2 には 4 つのイベントカウンタがあるので同時に 4 つのイベントをカウントすることができます。 デフォルトでは CPU サイクルをカウントします。再コンパイルは必要ありません。

例）

```
% lipfpm ./a.out

lipfpm summary
====== ========
CPU Cycles..... 464827
```

次は 3 次キャッシュにおけるキャッシュミスのカウントの例です。イベントの名前を-e オプションと共に指定してください。

例）

```
%   lipfpm -e L3_MISSES ./a.out

lipfpm summary
====== ========
L3 Misses..... 1642
CPU Cycles..... 458637
Average MB/s requested by L3..... 595.074086
```

オプションを付けずに lipfpm を実行するとオプションの内容などが表示されます。 -i オプションで全イベントの名前がインタラクティブにアルファベット順に表示されます。

space で次のイベント、backspace で前のイベントが表示されます。

enter でイベントの選択ができます。Esc で抜けます。

12.1.1 MFLOPS 値計測

MFLOPS 値を計測するには、カウントするイベントとして FP_OPS_RETIRED を指定してください。

例）

```
% lipfpm -e FP_OPS_RETIRED ./a.out

lipfpm summary
====== ========
Retired FP Operations..... 6868
CPU Cycles..... 459158
Average MFLOP/s..... 19.044516
```

12.1.2 自動並列化／OpenMP プログラムの解析

並列プログラムを解析するときは、-f オプションを追加してください。また、-o オプションで結果を格納するファイルを指定してください。結果ファイルが指定されないと、各スレッドがばらばらに標準エラー出力に結果を書き出してしまいます。自動並列化／OpenMP プログラムを実行すると、OMP_NUM_THREADS で指定した数のプロセスの他に 2 つの管理プロセスが起動されます。そのため解析結果ファイルも実行スレッド数＋2 個作られます。

例）

```
% setenv OMP_NUM_THREADS 4
% lipfpm -f -o cnt ./a.out
% ls cnt*
cnt.a.out.17986   cnt.a.out.17987   cnt.a.out.17988   cnt.a.out.17989   cnt.a.out.17990
% cat cnt*

lipfpm summary
====== =======
CPU Cycles..... 382948138

lipfpm summary
====== =======
CPU Cycles..... 222151

lipfpm summary
====== =======
CPU Cycles..... 291807031

lipfpm summary
====== =======
CPU Cycles..... 321855693

lipfpm summary
====== =======
CPU Cycles..... 322188429
```

dplace と共に実行するとき次のように histx を指定します。

```
% dplace -x13 lipfpm -f -o 解析結果ファイル名   ./a.out
/* dplace のオプション  -x  の値は 13 です  */
```

12.1.3 MPI プログラムの解析

MPI プログラムも-f オプションを指定してください。

例）

```
% mpirun -np N dplace -s3 lipfpm -f -o out a.out
```

サンプルプログラム （次ページも含む）

```
% cat test.f
              program main
              parameter(n=800)
              real a(n,n),b(n,n),c(n,n)
              call init(n,a,b,c)
              call matmul(n,a,b,c)
              print *,c(1,1)
              stop
              end
c
c******* initialization **************
c
              subroutine init(n,a,b,c)
              integer n
              real a(n,n),b(n,n),c(n,n)
              do j=1,n
              do i=1,n
                   a(i,j)=1.
                   b(i,j)=1.
                   c(i,j)=0.
              enddo
              enddo
              return
              end
c
c******* calculation *****************
c
              subroutine matmul(n,a,b,c)
              integer n
              real a(n,n),b(n,n),c(n,n)
              do k=1,n
              do j=1,n
              do i=1,n
                   c(j,k)=c(j,k)+a(k,i)*b(i,j)
              enddo
              enddo
              enddo
              return
              end
```

最適化オプション -O2 でコンパイルし、MFLOPS 値を計測。

```
% ifort test.f
% lipfpm -e FP_OPS_RETIRED ./a.out
     800.0000

lipfpm summary
====== =======
Retired FP Operations..... 1025922484
CPU Cycles..... 3707305492
Average MFLOP/s..... 359.074894
```

※ 359MFLOPS

最適化オプション -O3 でコンパイルし、MFLOPS 値を計測。

```
% ifort -O3 test.f
% lipfpm -e FP_OPS_RETIRED ./a.out
     800.0000

lipfpm summary
====== =======
Retired FP Operations..... 1024002484
CPU Cycles..... 329511032
Average MFLOP/s..... 4039.093524
```

※ 4039MFLOPS

12.2 histx (HISTogram eXecution)コマンド

histx コマンドは、プログラムのどこで実行時間が消費されているか、どこでイベントが発生しているかを知るためのツールです。プログラムの実行が終わると解析結果ファイルが作られるので、iprep コマンドを使って整形してください。

例）

```
% histx -o out ./a.out
800.0000
/*  ーo オプションは必須です。解析結果を収めるファイル名の先頭にこの名前が付きます。*/
% ls out*
out.a.out.18844
% iprep out.a.out.18844
Functions sorted by count
========== ======= == ====

          Count              Self   %     Cum.   %     Name
---------------------     --------     --------     ---------------------
                    260      99.237       99.237     a.out:matmul_
                      2       0.763      100.000     a.out:init_
```

ソースコードのライン毎の解析を行いたいときは、-g オプションを付けてコンパイルしてください。histx には、-l オプションを指定してください。

```
% ifort -g -o a.out test.f
% histx -l -o out ./a.out
% ls out*
out.a.out.19087
% iprep out.a.out.19087
Functions sorted by count
========== ======= == ====

          Count              Self   %     Cum.   %     Name
---------------------     --------     --------     ---------------------
                  53119      99.790       99.790     a.out:matmul_ [test.f:27]

Source lines sorted by count
======= ====== ======= == ====

          Count              Self   %     Cum.   %     Name
---------------------     --------     --------     ---------------------
                  51242      95.652       95.652     [test.f:33] (a.out)
                   2200       4.107       99.759     [test.f:34] (a.out)
            . . .    . . .    . . .
```

オプションを指定せずに histx コマンドを実行すると、使い方の概要が表示されます。
histx はコールスタックのサンプリングを採ることもできます。「12.2.3 histx を用いたコールスタックの解析」を参照ください。

12.2.1 histx を用いた自動並列化／OpenMP プログラムの解析

自動並列化/OpenMP コードを histx コマンドを使って解析するときは -f オプションを指定してください。lipfpm コマンドと同様に実際の実行スレッド数＋2 の解析結果ファイルが作成されます。

```
% histx -f -o out ./a.out
```

dplace と共に実行するとき次のように histx を指定します。

```
% dplace -x13 histx -f -o  解析結果ファイル名    ./a.out
/* dplace のオプション  -x  の値は 13 です  */
```

12.2.2 histx を用いた MPI プログラムの解析

MPI プログラムを解析するときは -f オプションを指定してください。

```
% mpirun -np N dplace -s3 histx -f -o  解析結果ファイル名  ./a.out
```

12.2.3 histx を用いたコールスタックの解析

```
% histx -o cs -s callstack10 ./prog
```

csrep コマンドを使って整形します。標準出力に表示されます。

```
% csrep < cs.mpirun.20138 > out
```

コールスタックに多く現われたルーチンから表示されます。

```
% cat out
Appearance Frequency
========== =========
  80.00% libxmpi.so:mpirun
  40.00% libarray.so:asopenserver
  40.00% libarray.so:newarraysess
  40.00% libxmpi.so:newash
  40.00% a.out:main
  40.00% ld-linux-ia64.so.2:_dl_catch_error_internal
  40.00% libc.so.6.1:getservbyname
  40.00% libarray.so:ASGetServiceByName
  40.00% libc.so.6.1:getservbyname_r@@GLIBC_2.2
  40.00% libarray.so:ASGetPortNum
   .
   .
   .
```

各ルーチン毎に、コール元のルーチン、コール先のルーチンの情報が表示されます。

```
Butterfly Report
========= ======
-------------------------------------------------------------------------------
100.00% ( 40.00%) a.out:main+0x60

..... ( 80.00%) libxmpi.so:mpirun
```

上記の例で注目しているのは「main」です。上下が空行になっていることでわかります。main よりも上に表示されているのは、main をコールしたルーチン、下に表示されているのは、main がコールしたルーチンです。

サンプルプログラム（3 ページ）

```
% cat test.f
            program main
            parameter(n=800)
            real a(n,n),b(n,n),c(n,n)
            call init(n,a,b,c)
            call matmul(n,a,b,c)
            print *,c(1,1)
            stop
            end
c******* initialization **************
            subroutine init(n,a,b,c)
            integer n
            real a(n,n),b(n,n),c(n,n)
            do j=1,n
            do i=1,n
                a(i,j)=1.
                b(i,j)=1.
                c(i,j)=0.
            enddo
            enddo
            return
            end
c******* calculation *****************
            subroutine matmul(n,a,b,c)
            integer n
            real a(n,n),b(n,n),c(n,n)
            do k=1,n
            do j=1,n
            do i=1,n
                c(j,k)=c(j,k)+a(k,i)*b(i,j)
            enddo
            enddo
            enddo
            return
            end
```

行毎の解析を行なうために、-g オプションでコンパイル。histx には、-l オプションをつける。

```
% ifort -O3 -g test.f
%   histx -l -o OUT ./a.out
    800.0000
% iprep OUT.a.out.20546
Functions sorted by count
========= ====== == ====

         Count             Self   %    Cum.   %    Name
----------------------    -------    --------    ----------------------
                   258    100.000    100.000    a.out:matmul_ [test.f:27]

Source lines sorted by count
====== ===== ====== == ====

         Count             Self   %    Cum.   %    Name
----------------------    -------    --------    ----------------------
                   230     54.502     54.502    [test.f:33] (a.out)
                   191     45.261     99.763    [test.f:32] (a.out)
```

自動並列化オプションを指定してコンパイル。

```
% ifort -parallel -O3 -g test.f
test.f(30) : (col. 0) remark: LOOP WAS AUTO-PARALLELIZED.
% setenv OMP_NUM_THREADS 2
% histx -l -o OUT ./a.out
    800.0000
```

```
% ls OUT*
OUT.a.out.20546    OUT.a.out.20737    OUT.a.out.20738    OUT.a.out.20739
```
環境変数 OMP_NUM_THREADS で指定した数＋2 の解析結果ファイルが生成されますが、そのうち解析結果があるのは OMP_NUM_THREADS で指定した数のファイルです。
```
% iprep OUT.a.out.20546
Functions sorted by count
========= ====== == ====

        Count             Self   %    Cum.   %    Name
--------------------    -------    -------    --------------------
                  258    100.000    100.000    a.out:matmul_ [test.f:27]

Source lines sorted by count
====== ===== ====== == ====

        Count             Self   %    Cum.   %    Name
--------------------    -------    -------    --------------------
                  230     54.502     54.502    [test.f:33] (a.out)
                  191     45.261     99.763    [test.f:32] (a.out)
% iprep OUT.a.out.20737
Functions sorted by count
========= ====== == ====

        Count             Self   %    Cum.   %    Name
--------------------    -------    -------    --------------------
                  119     90.840     90.840    a.out:_matmul_30_par_loop0 [test.f:38]
                    4      3.053     93.893    libguide.so:__kmp_yield
                    4      3.053     96.947    libguide.so:__kmp_wait_sleep
                    1      0.763     97.710    libc.so.6.1:sched_yield
                    1      0.763     98.473    libguide.so:__kmp_ia64_pause
                    1      0.763     99.237    libc.so.6.1:getenv [getenv.c:39]
                    1      0.763    100.000    a.out:init_ [test.f:12]

Source lines sorted by count
====== ===== ====== == ====

        Count             Self   %    Cum.   %    Name
--------------------    -------    -------    --------------------
                  103     49.519     49.519    [test.f:32] (a.out)
                  102     49.038     98.558    [test.f:33] (a.out)
% iprep OUT.a.out.20738
% iprep OUT.a.out.20739
Functions sorted by count
========= ====== == ====

        Count             Self   %    Cum.   %    Name
--------------------    -------    -------    --------------------
                  119     99.167     99.167    a.out:_matmul_30_par_loop0 [test.f:38]
                    1      0.833    100.000    libguide.so:__kmp_ia64_pause

Source lines sorted by count
====== ===== ====== == ====

        Count             Self   %    Cum.   %    Name
--------------------    -------    -------    --------------------
                  117     51.092     51.092    [test.f:32] (a.out)
                  112     48.908    100.000    [test.f:33] (a.out)
```

13 バイナリデータの取り扱い

Fortran プログラムにおけるバイナリデータファイルの入出力を対象とした、エンディアンの変換方法を説明します。 共用計算サーバではバイナリデータの形式としてリトルエンディアンを採用しています。エンディアン形式変換機能を使って、ビッグエンディアン形式のファイルを読み込む、あるいはビッグエンディアン形式のファイルを書き出すといった処理が可能になります。

環境変数 F_UFMTENDIAN にファイルのユニット番号を指定しますと、該当のユニット番号のデータに対してプログラム実行時に次のような変換を行います。

- READ を実行するとき:ビッグ→リトル
- WRITE を実行するとき:リトル→ビッグ

環境変数のパラメータ(空白は不可)

```
% setenv F_UFMTENDIAN MODE | [MODE; ]  EXCEPTION
MODE = big | little
EXCEPTION = big:ユニット番号 | little:ユニット番号 | ユニット番号
```

MODE はファイルの形式を指定します。little を指定すると変換を行いません。big を指定すると変換を行います。省略すると little が設定されます。EXCEPTION には MODE 以外の形式を持つファイルを指定します。

- ユニット番号 10 と 20 のファイルだけを変換したいとき

```
% setenv F_UFMTENDIAN 10,20
```

全てのファイルはリトルエンディアン形式であることが前提で、10 と 20 だけがビッグエンディアン形式を採用するということを設定しています。ユニット番号 10 と 20 のファイルは READ/WRITE 文によって変換されます。

- 全てのファイルを変換したいとき

```
% setenv F_UFMTENDIAN big
```

READ 文によって入力ファイルはビッグ→リトル変換が行われ、WRITE 文によってリトル→ビッグ変換が行われます。

- 上記の条件に加えてユニット番号 8 だけ変換したくないとき

```
% setenv F_UFMTENDIAN "big;little:8"
```

MODE で big が指定されているので、全てのファイルについて変換されます。しかし、ユニット番号 8 のファイルだけは変換が行われません。

- ユニット番号 10 から 19 までのファイルを変換したいとき

```
% setenv F_UFMTENDIAN 10-19
```

注意:ここで紹介した操作は C/C++プログラムには適用できません。C/C++プログラムで作られたデータファイルを変換する場合には、それぞれ変換プログラムが必要です。

14 各種マニュアルについて

主な各種マニュアルは以下のディレクトリに格納されております。

内容	保存先ディレクトリ
AVS Express PST 関連マニュアル	/home/doc/prism/avs
IDL 関連マニュアル	/home/doc/prism/idl
ISL 関連マニュアル	/home/doc/prism/isl
Vizserver 関連マニュアル	/home/dic/prism/viz
Intel コンパイラ関連マニュアル	/home/doc/prism/compiler
OpenMP 関連マニュアル	/home/doc/prism/openmp
MPI ライブラリ（MPT）関連マニュアル	/home/doc/prism/mpi
科学技術計算ライブラリ（SCSL）関連マニュアル	/home/doc/prism/scsl