東大センターにおける
スーパーコンピューター利用入門
— SR11000編 —
第4版

東京大学情報基盤センター

（スーパーコンピューティング部門）

2009 年 3 月

まえがき

本利用入門は東京大学情報基盤センターのスーパーコンピューターを利用する際に必要とする基本的なコマンド，機能などについて説明したものです。本センターではスーパーコンピューターシステムによるサービス運用を行っており，このシステムは経験豊富な利用者によって高度に活用されています。しかし，スーパーコンピューターで UNIX あるいは並列計算をはじめようという方々にとって，豊富な機能や複雑な構成はむしろ掴み所がなく，利用にあたっては戸惑いも多いかと思います。本利用入門は過去にスーパーコンピューティングニュースに掲載した記事をもとにシステムの概要からジョブの実行方法，センター固有の設定等について例を挙げながら解説した入門書です。これから本センターのシステムを御利用頂く方々にとって本利用入門がスーパーコンピューティングへの道標となれば幸いです。

（発行にあたって）

本利用入門は 2007 月 4 月発行の第 3 版をもとに，2009 年度用に改訂したものです。

今後，記載内容は変更となる場合がありますので最新の情報はスーパーコンピューティングニュース，Web ページ（http://www.cc.u-tokyo.ac.jp/）やシステム上の show-info コマンドにて御確認頂きますようお願い致します。

2009 年 3 月

東京大学情報基盤センター
システム運用係

目　次

付録

1. システム概要

1.1. システム構成

本センターでは,

- HITACHI　SR11000
- HA8000 クラスタシステム（T2K オープンスーパーコンピューター）

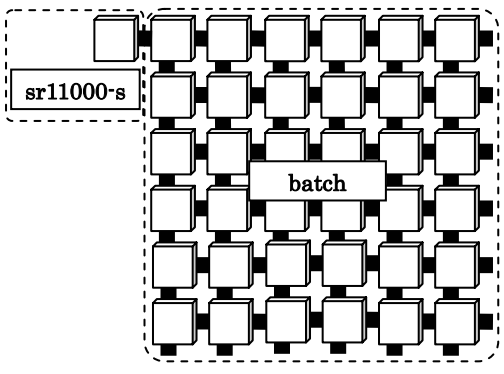
の２式のスーパーコンピューターシステムを運用しています。本利用入門では,「SR11000」について解説します。

SR11000 は 1 ノード当り 16 台の CPU（プロセッサー）を備えたノードを複数台搭載しており，ノードを並列に動作させるのと同時にノード内でも並列処理が可能です。システムの諸元は以下のとおりです。

スーパーコンピューターシステム名	HITACHI SR11000
ノード数	128 (16CPU/ノード)
1CPU 当りの理論性能	9.2 GFLOPS
1 ノード当りの理論演算性能	147.2 GFLOPS
総理論演算性能	18,841.6 GFLOPS
1 ノード当りの主記憶（メモリー）容量	128 GB
総主記憶容量	16,384 GB
OS	AIX 5L

☞ 2005 年 3 月 44 ノードでサービス開始。2007 年 4 月に 128 ノードに増強。
☞ 一部のノードについては, 物理的なノードを論理分割し 8CPU を 1 ノードとして運用。その場合, 1 ノード当りの理論演算性能は 73.6GFLOPS，主記憶容量は 64GB。

各ノードは多段クロスバーネットワークと呼ばれる内部ネットワークで結合されており，互いに通信することができます。本センターでは搭載された多くのノードと機能を生かし，様々なバッチ処理やインタラクティブ処理を行えるよう，通常はシステムを次のようにインタラクティブ処理用，バッチ処理用の２つに分割して運用しています。



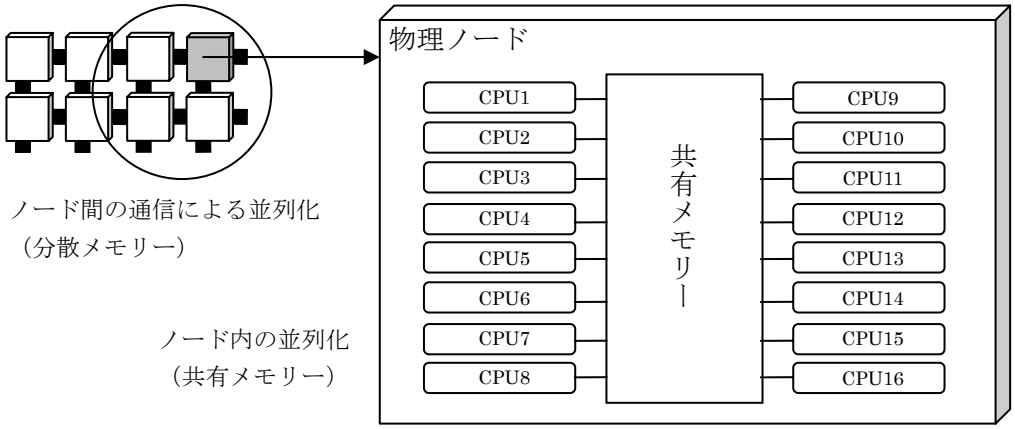
SR11000

注) 図は実際のノードの配置や数とは異なります。

☞ SR11000 は SMP（Symmetric Multiple Processor）クラスター型システムであり，複数台のノードで構成されます。
☞ 本センターでは SR11000 のバッチ用システムを総称して "batch" としています。

1.2. ノード構成

SR11000 は 1 ノード当たり 16 台の CPU を搭載しています。ノード内では 16 台の CPU がメモリーを共有する共有メモリー型，ノード間では分散メモリー型になります。ノード単体でも並列実行やスカラージョブの多重処理が可能です。



なお, 一部のノードについては物理的なノードを論理分割し 8CPU で構成する SMP を 1 ノードとして運用しています。

1.3. ソフトウェア

SR11000 システムは，1 つのシステムにインタラクティブ（TSS）環境とバッチ環境を備えています。

インタラクティブ処理用システム（ホスト名: **sr11000-s.cc.u-tokyo.ac.jp**）

ログインしてコマンド実行（対話的に実行）できます。主にプログラムの作成・編集，コンパイル，バッチジョブの投入に使用します。

バッチ処理用システム

バッチジョブの実行ができます。ジョブはインタラクティブ処理用システムから NQS（Network Queuing System）により投入します。バッチジョブ実行専用のシステムなのでログインはできません。

申し込みのコース（パーソナル，グループ，研究室の各コース）によっては，利用できるバッチ環境に制限があります。インタラクティブ環境については各コースとも同様に利用可能です。

なお，毎月最終金曜日（休日の場合は木曜日）午後から翌週月曜日（休日の場合は火曜日）朝までの期間は 64 ノード用ジョブキュー(P064 キュー)のサービスを行います。本サービスはパーソナルコース 4 またはグループコース 2 の利用者が利用できます。

☞ 詳細は Web ページ（http://www.cc.u-tokyo.ac.jp/）または「付録E 超並列ジョブ実行サービスについて」を御覧下さい。

インストールされている主なソフトウェアは以下のとおりです。

	ソフトウェア
言語プロセッサー （コンパイラー）	最適化 FORTRAN77 最適化 Fortran90 最適化 C 最適化標準 C++[*1]
並列化支援アプリケーション	MPI（MPI-2）
数値計算ライブラリー	MATRIX/MPP MATRIX/MPP/SSS MSL2 BLAS, LAPACK, ScaLAPACK
その他ライブラリー	Gaussian03[*2]

*1 ANSI/ISO C++ 1998 準拠 および ARM 仕様対応

*2 利用については「付録D Gaussian03 の使い方」を御覧下さい

☞ 最新のバージョン情報等は Web ページまたはスーパーコンピューティングニュース「システム変更等のお知らせ」にて御確認下さい。

1.4. フリーソフトウェア

SR11000 には以下の表にあるフリーソフトウェアがインストールされています。各自の責任の上で，必要に応じて PATH の追加等の設定を行い，ご利用下さい。

/usr/local/unsupported/bin
ソフトウェア名称
emacs
tcsh
bash
less
ATLAS

/opt/freeware/bin		
ソフトウェア名称		
autoconf	gmake	sed
automake	gzip	tar
bzip2	less	vim
cvs	libtool	wget
gcc	rsync	

注意事項

・ソフトウェアのバージョン，種類等は今後変更となる場合があります。

・ソフトウェアの追加，更新等のご要望はご容赦下さい。

・tcsh，bash のシェルプログラムは，ログインシェルとして使用する事はできません。

・バッチ環境（NQS）では利用できません（ATLAS 除く）。

・ATLAS パッケージ（BLAS，LAPACK）については，「12.3 BLAS・LAPACK の利用について」を参照して下さい。

なお，これらのソフトウェアには実行時にエラーなどの不具合が発生する場合があります。また，システム固有の構成に対応していないプログラムや本センターシステムで使用が制限されている機能を使用したプログラムは動作不良や，場合によってはシステム障害を引き起こす可能性があり，本センターでは保守・管理上の対応が困難なため，フリーソフトウェアに関しては，保証，サポートを行いません。フリーソフトウェアの使用により生じた問題には対処致しませんので各自の責任でご利用下さい。特に gcc が生成したコードが起こした不都合に対してもサポートを行いません。また，gcc は日立製コンパイラーに比べて，性能が出ませんのでご了承下さい。障害等が発生した場合には当該ソフトウェアのサービスを中止することがありますのでご了承下さい。使用方法，性能，障害等に関する質問等についても回答できませんので，予めご承知置き下さい。

また，本センタースーパーコンピューティング研究部門の教員が作成した，高速ファイル転送ソフト SRFT があります。詳細は「付録Ｂ 高速ファイル転送ソフト SRFT について」または Web ページを御覧下さい。

2. 利用者登録

SR11000 システムを利用するためには次の 3 つの登録形態（コース）があります。

・個人で利用する場合

『スーパーコンピューターシステム利用申込書（新規）(パーソナルコース)』
を提出し利用者番号を取得します。

・グループで利用する場合

『スーパーコンピューターシステム利用申込書（新規）(グループコース)』
をグループの代表者が提出しメンバーの利用者番号を取得します。

・研究室で利用する場合

『スーパーコンピューターシステム利用申込書（新規）(研究室コース)』
を研究室の代表者が提出しメンバーの利用者番号を取得します。メンバーは同一研究室で 3 名以上（教員，学生各 1 名以上）であることが申し込みの条件です。ただし，教員がパーソナルコース 2 以上に登録している場合は 2 名からでも申し込みが可能です。

☞ 各種申込書は Web ページに掲載しています。
☞ コースにより利用できるバッチキューが異なります。詳細は Web ページ, スーパーコンピューティングニュースまたは「付録H ジョブクラス制限値」を参照して下さい。

3. ログイン方法

3.1. 概要

2009 年 4 月以降，それ以前とは異なり，ログイン自体にはパスワードを使用せず「SSH 鍵による認証」を行います。また，SSH は Version2 のみを有効とします。

そのため，初回ログイン時にのみ鍵の生成とシステムへの登録作業が必要となります。おおまかな流れは次のようになります。

1. Windows の場合はターミナルソフトを入手する
2. 認証に用いる鍵を生成する
3. Web ブラウザを用いて SR11000 に公開鍵を登録する
4. ログインする

手順は手元のマシンが Windows か UNIX 系オペレーティングシステム（MAC OS X を含む）かで異なりますので，OS ごとに分けて説明します。すでに公開鍵をお持ちの方は改めて鍵を生成する必要はありません。SR11000 システムへの鍵の登録のみで使用できます。

3.2. 接続情報

ログインに必要な情報は以下のとおりです。

ホスト名	sr11000-s.cc.u-tokyo.ac.jp
接続方法	SSH Protocol Version 2
認証方法	鍵による認証（センター発行のパスワードはログインには使いません） 初回は Web による鍵登録が必要です。本章で説明します。

3.3. Windows システムからのログイン方法

Windows で使用できるターミナルソフトには PuTTY や Tera Term Pro などがあります。これらの中では PuTTY がもっとも鍵の扱いが容易なので， PuTTY を本センターの推奨ターミナルソフトとし接続方法を説明します。他のターミナルソフトについては解説しませんが，多くのソフトは鍵での認証に対応しているので SR11000 システムへのログインに使用することができます。また Cygwin を使用される方は UNIX 向けの解説をご覧ください。

① PuTTY の入手

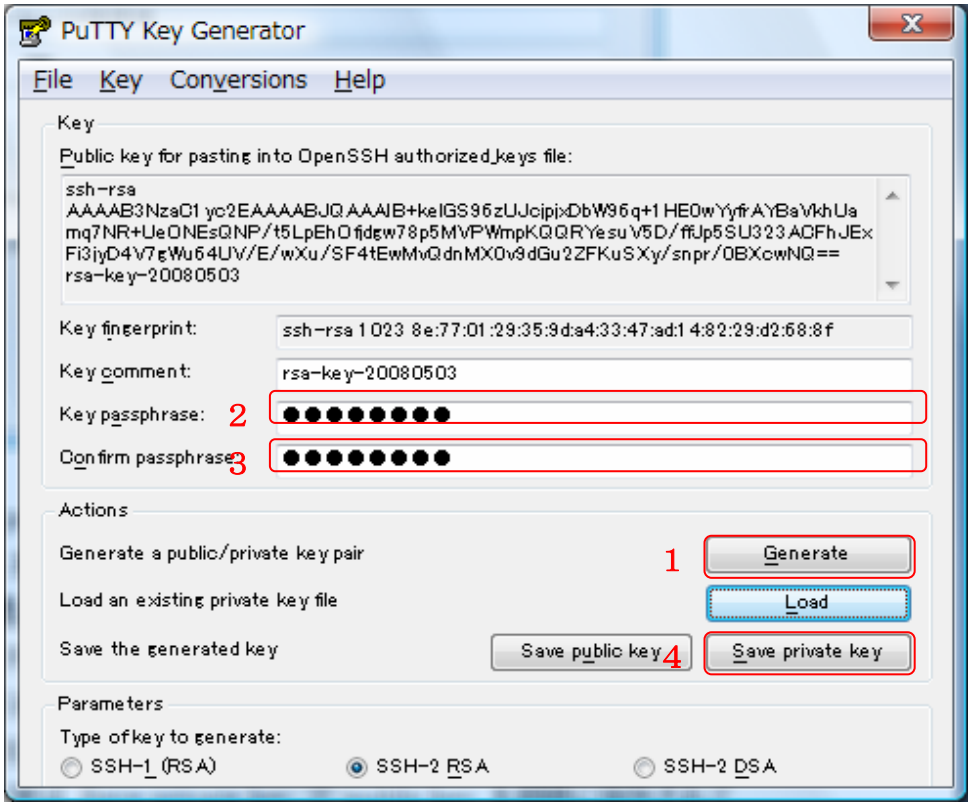
2009 年 3 月現在 PuTTY は日本語化されたものが次のページより入手できます。

http://hp.vector.co.jp/authors/VA024651/

このページより日本語化 PuTTY をダウンロードし，適切なフォルダに展開してください。

② 鍵の生成

PuTTY の配布ファイルを展開すると puttygen.exe という実行可能ファイルがあるのでそれを実行します。PuTTY Key Generator の Window が開くので Generate ボタンをクリックします。すると画面の上の方にバーが出現するので，バーが右端に到達するまでバーの下あたりでマウスを動かします。



Key passphrase と Confirm passphrase に適当な他人に知られない文字列を入力します。センターから通知されたパスワードとは無関係なので，センターから発行されたパスワードを入力しないように注意してください。また，入力しなくても鍵としては使えますが，セキュリティー強化のため SR11000 システムに登録する鍵はパスフレーズで保護されていることを必須とします。パスフレーズはメモしたりファイルに書いたりしなくて済むよう，他人には推測されず，自分には覚えやすいものを設定してください。パスフレーズの入力が終了したら Save private key ボタンをクリックして鍵を保存します。この秘密鍵ファイルは絶対に他人に読まれることがないよう，アクセス権には十分注意してください。この鍵が他人に読まれると SR11000 システムに不正侵入される危険性があります。Save public key は必要ありません（Save private key で public key も同時に保存されています）。以上で鍵の生成は完了ですが，次の登録作業のためにこの Window は開いたままにしておいてください。

③ 鍵の SR11000 システムへの登録

SR11000 システムには Window の上の方にある”Public key for pasting into OpenSSH authorized_keys file” と書かれた欄にある文字列を登録します。もし，前のステップの後 puttygen を終了してしまった場合は，再び puttygen を起動した後 ”Load”ボタンを押して前のステップで保存した鍵のファイルを読み込めば登録に必要な文字列が得られます。

鍵の登録のために，次の URL にアクセスしてください。

https://regist.cc.u-tokyo.ac.jp/sr11000-key/

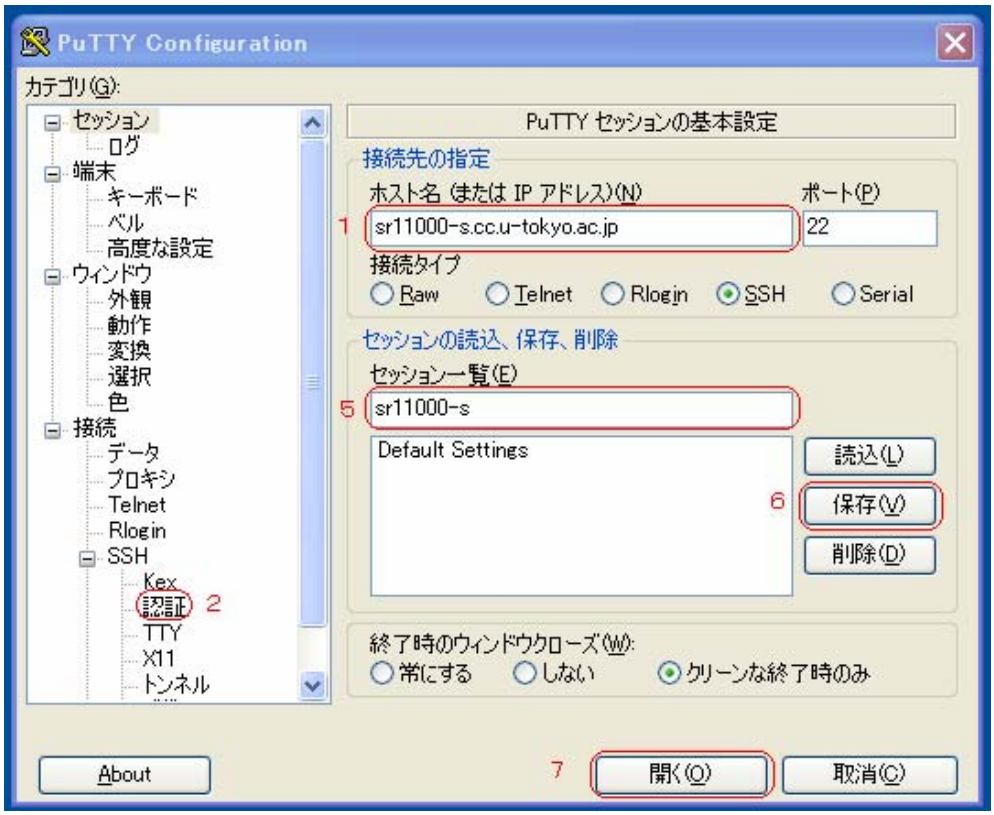
ユーザー名とパスワードを求められるので，センターから通知されたユーザー名とパスワードを入力してください。ここで入力するパスワードは鍵の保護のために入力したパスフレーズではなく，センター発行のパスワードです。認証に成功するとユーザーページが開きますので「公開鍵登録」のリンクをクリックしてください。開いたページの “Public Key”の欄に PuTTY Key Generator ウインドウの ”Public key for pasting into OpenSSH authorized_keys file” に表示されている文字列をコピー&ペーストし，”Password” の欄にセンター発行のパスワードを入力してください（すでに認証済みですが，セキュリティー上非常に重要な部分なので再度パスワード入力をお願いしております)。「登録」をクリックして完了画面が出れば登録完了です。登録まで少し時間がかかることがありますが，「登録」ボタンを何度もクリックしないようお願いします。

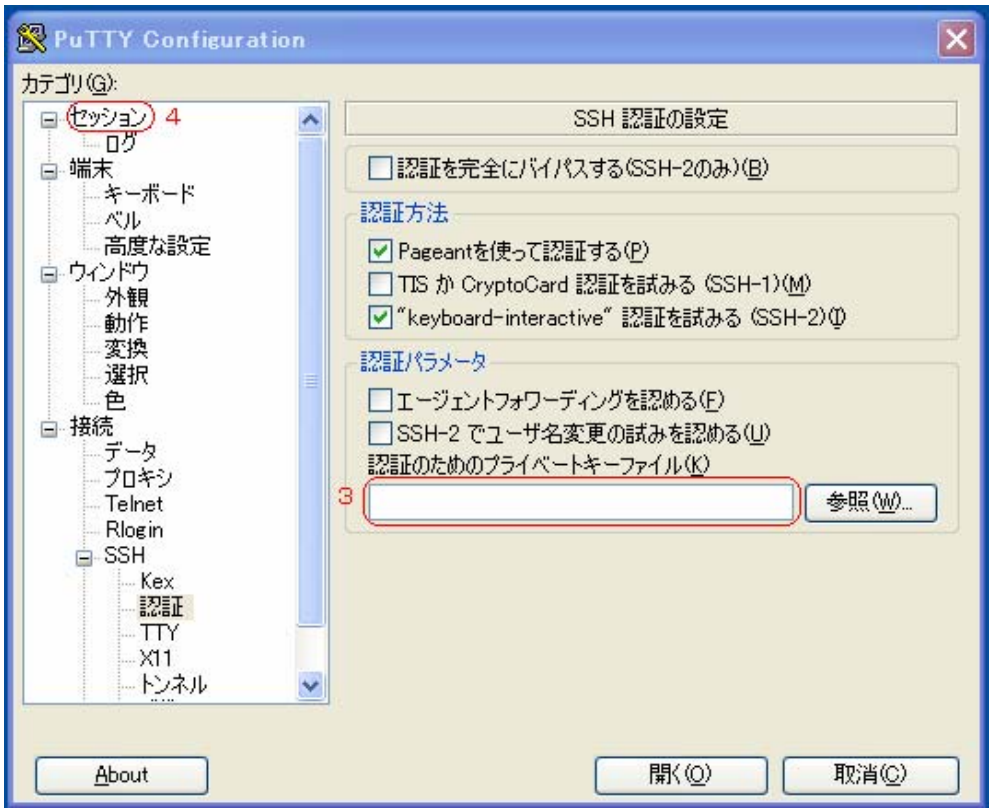
登録作業を 2 回以上行うと，最初に登録した鍵は削除されます。また，一度でもログインすると Web 経由での公開鍵登録はできなくなります。複数のコンピュータからログインするために公開鍵を 2 個以上登録する方法や，ログインした後の鍵の変更方法は後の「鍵について」の節で説明します。

④ ログイン

PuTTY を起動すると次ページ上のような画面が出ます。図に示した順番で以下のように情報を設定します

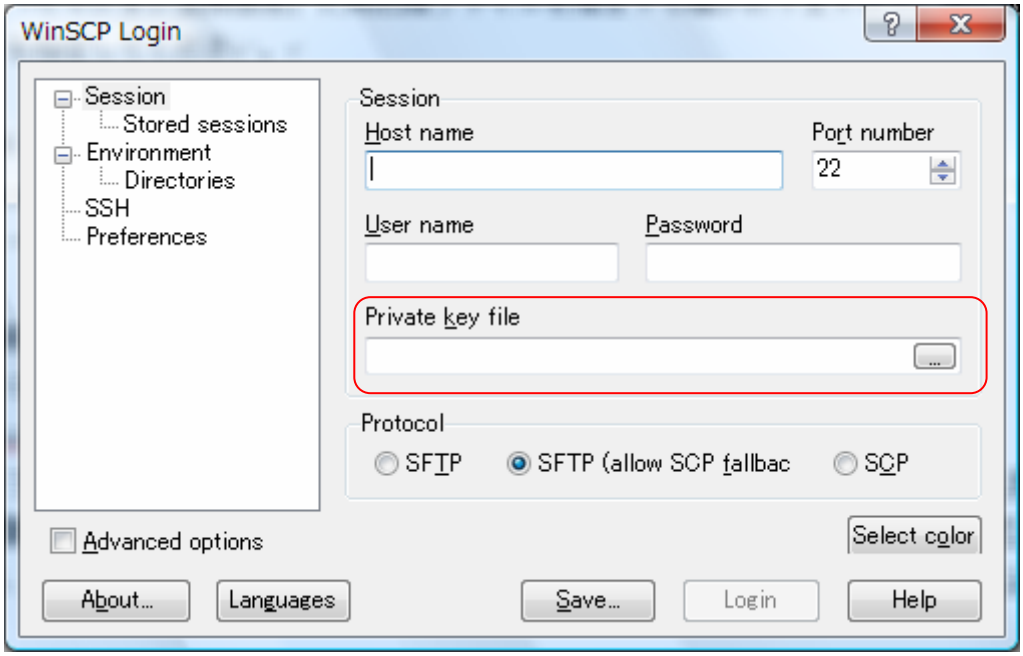
1. ホスト名を入力
2. 「SSH」の「認証」画面に移動して秘密鍵ファイルを設定する
3. 「セッション」画面に戻ってセッション名を入力し，保存する
4. 「開く」ボタンで接続する。ユーザー名を入力すると鍵のパスフレーズを聞かれるので，鍵作成の時に設定したパスフレーズを入力する。


PuTTY Configuration
カテゴリ(G):
セッション
ログ
端末
キーボード
ベル
高度な設定
ウィンドウ
外観
動作
変換
選択
色
接続
データ
プロキシ
Telnet
Rlogin
SSH
Kex
認証
TTY
X11
トンネル
PuTTY セッションの基本設定
接続先の指定
ホスト名 (または IP アドレス)(N)
ポート(P)
1
sr11000-s.cc.u-tokyo.ac.jp
22
接続タイプ
Raw
Telnet
Rlogin
SSH
Serial
セッションの読込、保存、削除
セッション一覧(E)
5
sr11000-s
Default Settings
読込(L)
6
保存(V)
削除(D)
終了時のウィンドウクローズ(W):
常にする
しない
クリーンな終了時のみ
2
About
7
開く(O)
取消(C)


PuTTY Configuration
カテゴリ(G):
セッション
4
ログ
端末
キーボード
ベル
高度な設定
ウィンドウ
外観
動作
変換
選択
色
接続
データ
プロキシ
Telnet
Rlogin
SSH
Kex
認証
TTY
X11
トンネル
SSH 認証の設定
認証を完全にバイパスする(SSH-2のみ)(B)
認証方法
Pageantを使って認証する(P)
TIS か CryptoCard 認証を試みる (SSH-1)(M)
"keyboard-interactive" 認証を試みる (SSH-2)(I)
認証パラメータ
エージェントフォワーディングを認める(F)
SSH-2 でユーザ名変更の試みを認める(U)
認証のためのプライベートキーファイル(K)
3
参照(W)...
About
開く(O)
取消(C)

⑤ ファイルコピー

ファイルコピーには WinSCP というソフトが使用できます。WinSCP は PuTTY Key Generator で生成して保存した秘密鍵がそのまま使えます。ホスト名やユーザー名を登録する「セッション登録画面」に秘密鍵ファイルを指定する欄がありますので，秘密鍵ファイルを指定してください。



3.4. UNIX システムからのログイン方法

Linux や MacOS X，Windows 上の Cygwin からのログインには OS にデフォルトでインストールされている OpenSSH が使用できます。

① 鍵の生成

次のコマンドを入力してください

```
$ ssh-keygen -t rsa
```

最初に鍵の保存場所を聞かれます。これはそのままリターンで大丈夫です。次にパスフレーズを求められるので入力して下さい。センターから通知されたパスワードとは無関係なので，センターから発行されたパスワードを入力しないように注意してください。また，入力しなくても鍵としては使えますが，セキュリティー強化のため SR11000 システムに登録する鍵はパスフレーズで保護されていることを必須とします。パスフレーズはメモしたりファイルに書いたりしなくて済むよう，他人には推測されず，自分には覚えやすいものを設定してください。確認のためもう一度パスフレーズを入力したら鍵生成は完了です。

② 鍵の SR11000 システムへの登録

鍵の登録のために，次の URL にアクセスしてください。

https://regist.cc.u-tokyo.ac.jp/sr11000-key/

ユーザー名とパスワードを求められるので，センターから通知されたユーザー名とパスワードを入力してください。ここで入力するパスワードは鍵の保護のために入力したパスフレーズではなく，センター発行のパスワードです。認証に成功するとユーザーページが開きますので「公開鍵登録」のリンクをクリックしてください。開いたページの “Public Key”の欄に ~/.ssh/id_rsa.pub の内容をコピー&ペーストし，”Password” の欄にセンター発行のパスワードを入力してください（すでに認証済みですが，セキュリティー上非常に重要な部分なので再度パスワード入力をお願いしております)。「登録」をクリックして完了画面が出れば登録完了です。登録まで少し時間がかかることがありますが，「登録」ボタンを何度もクリックしないようお願いします。

登録作業を 2 回以上行うと，最初に登録した鍵は削除されます。また，一度でもログインすると Web 経由での公開鍵登録はできなくなります。複数のコンピュータからログインするために公開鍵を 2 個以上登録する方法や，ログインした後の鍵の変更方法は次節「鍵について」で説明します。

③ ログイン

```
$ ssh sr11000-s.cc.u-tokyo.ac.jp -l xxxxxx
```

でパスフレーズを入力すればログインできます。自動的に公開鍵認証が行われます。

④ ファイルコピー

```
$ scp file xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:
```

でパスフレーズを入力すればコピーできます。自動的に公開鍵認証が行われます。

3.5. 鍵について（Windows，UNIX 共通）

① Web 登録システムの有効期間

Web による鍵登録が可能なのは最初にログインに成功するまでです。常に登録システムを開いておくと，悪意を持った第三者にパスワードが漏れた場合に登録システムを通じて攻撃者が生成した鍵が登録されてしまい，鍵認証にしている意味がなくなってしまうためです。鍵が未登録なはずなのに登録システムに登録を拒否された場合は，何者かによって鍵を登録されてしまったことになるので，至急センターまでご連絡下さい。

② Web 登録システムに入力できる鍵

Web 登録システムは OpenSSH 形式の鍵を受け付けます。Windows では推奨ソフトである PuTTY の他，Tera Term Pro もこの形式の鍵を生成します。

③ 鍵の追加と変更

ログインした後に，他のマシン用の鍵を追加したり，登録済みの鍵を変更する場合はSR11000 システムにログインし，~/.ssh/authorized_keys に書かれている公開鍵をエディタで直接編集してください。一行に一個の公開鍵を書きます。鍵の文字列は長いため，多くの場合は複数行にわたって表示されることになりますが，データ上は一個の鍵は一行でなくてはなりません。鍵の途中に改行を入れないようご注意下さい。このファイルを削除したり，書かれている公開鍵を壊してしまうとログインできなくなります。

④ 秘密鍵の管理

鍵の生成方法で説明したとおり，秘密鍵を守ることは SR11000 システムのセキュリティーを守る上で非常に重要なので，パスフレーズや秘密鍵が他人に漏れることのないよう十分ご注意下さい。万が一秘密鍵が他人に漏れた場合は必ずセンターに連絡し，適切な処置についてご相談下さい。複数のマシンから SR11000 システムにログインする場合は，それぞれのマシンで鍵を生成して，上で説明した方法で SR11000 システムの方に鍵を追加します。秘密鍵をコピーして他のマシンで使用してはいけません。

⑤ 別システムへのログインに使用している鍵の流用

一台のクライアントマシンから SR11000 システムと他の公開鍵認証を用いるシステム両方にログインする場合，他のシステムに登録した鍵を SR11000 システムにも登録することは差し支えありません。ただし，他のシステム用に登録していた鍵がパスフレーズで保護されていない場合は SR11000 システムには登録できません。一般的に，パスフレーズのない鍵の使用は非常に危険なので，その場合は，別のシステムに登録していた鍵の方をパスフレーズ付きに変更し，それを SR11000 システムにも登録することをお勧めします。

UNIX システムで複数の鍵を使い分けたいときは ~/.ssh/config に次のように書きます。

（例）sr11000-s.cc.u-tokyo.ac.jp には秘密鍵 ~/.ssh/id_rsa_sr11000 を使用するとき

```
Host sr11000-s.cc.u-tokyo.ac.jp
        User z00000
        IdentityFile ~/.ssh/id_rsa_sr11000
```

4. シェルとコマンド

4.1. ログインメッセージ

ログイン名，パスフレーズが正しいとログインメッセージが表示されます。

```
SR11000 Unix at Information Technology Center, University of Tokyo.
AIX 5L (Mon Apr 02, 2007) / sr11000-s.cc.u-tokyo.ac.jp.
（以下ログインメッセージ）
```

ログインメッセージはシステムに関する重要な内容が表示されますので必ず目を通すようにして下さい。以下のメッセージはシステム停止予定時刻と起動予定時刻です。定期保守や障害時などシステムを停止する必要がある場合に表示されます。

```
SR11000 system will SHUTDOWN at Mon Mar 30  9:00, 2009
        system will START     at Wed Apr  1  9:30, 2009
```

ログインメッセージ後，プロンプト % が出ればログイン完了です。

ログインが完了したら，センターで用意しているコマンド show-info を実行してみて下さい。このコマンドはセンターからのお知らせを表示するもので，サービス休止や障害に関する情報，FAQ など「センターからのお知らせ」を日本語で掲載しています。重要な情報が記述してありますので必ず目を通すようにして下さい。

```
% show-info
   0 Announce (2009.3.10)
   1 About SSH connection (2009.3.10)
   q (quit)
  number/character:
```

番号を選択すると，内容が表示されます。画面が停止したら，スペースキーで続きを読むか，「q」キーでメニューに戻ります。

4.2. ログアウト

ログアウトする場合には logout または exit コマンドで終了します。

```
% logout
```

☞ このとき不要なプロセス（実行中のプログラム）が残っていないか確認し，残っている場合は kill コマンドでプロセスを終了して下さい。

```
% ps -e
    PID    TTY  TIME CMD
 319536 pts/11  0:00 ./a.out
 393326 pts/11  0:00 ps
```

```
 598224 pts/11  0:00 -csh
% kill -9 319536   (./a.out [PID=319536] が不要なプロセスの場合)
```

4.3. シェルとコマンド

ログインするとログインメッセージの後にプロンプト % が現れます。これは C シェル（csh）と呼ばれるコマンドインタープリターが入力要求のメッセージとして出力しており，UNIX のコマンドを入力できる状態であることを意味します。コマンドの文字列を入力するとシェルがオプションや引数を展開してコマンドに実行を移します。

```
%   ←プロンプト
% ls  ←コマンド
a.f     a.out     ←実行結果
%
```

コマンドが実行を終えると，制御は C シェルに戻り，再びプロンプト % が表示されます。

☞ シェルには csh 以外に sh（Bourne シェル）等がありますが，ここでは本センター標準の csh を使用しているものとして説明します（tcsh，bash についてはサポート対象ではありませんが /usr/local/unsupported/bin/ にインストールされています)。

コマンドにはオプションを指定できるものがあります。例えば ls コマンドに -a オプションを指定すると .（ピリオド）で始まるファイルを表示できます。

```
% ls  -a  ←オプション
.     ..     .cshrc     .login     a.f     a.out
```

☞ 初期設定ファイル .cshrc および .login は利用登録時にセンターで作成しています。

さらに ls コマンドは引数としてファイル名やディレクトリー名を指定できます。例えば -l オプションとファイル名を指定するとファイルの属性（保護パーミッション，所有者，グループ，更新日付等）を見ることができます。

```
% ls  -l  a.f  ←引数
-rw-------    1 p08000   users              43 Apr  2 12:00 a.f
```

☞ 保護パーミッションは r（読み出し可)，w（書き込み可)，x（実行可)，-（許可なし）を所有者，グループおよび他人に対して設定できます。変更は chmod コマンドを使用します。

UNIX コマンドの基本はファイル，ディレクトリーの操作です。ファイルやディレクトリーの作成，削除，複写等ができます。主なコマンドの例を挙げます。

コマンド	機能	例
ls	ファイル情報の表示	% ls -l
cat	ファイル内容の表示，結合	% cat a.f b.f
more	ファイル内容の表示	% more a.f
less[*1]	ファイル内容の表示	% less a.f
vi	ファイルの作成，編集	% vi a.f
grep	パターン検索	% grep abc a.f

cp	ファイルのコピー	% cp a.f b.f
rm	ファイルの削除	% rm -i a.f
mv	ファイルの移動（ファイル名変更）	% mv -i a.f b.f
cd	ディレクトリーの移動	% cd work
pwd	カレントディレクトリーの表示	% pwd
mkdir	ディレクトリーの作成	% mkdir work
rmdir	ディレクトリーの削除	% rmdir work

*1 フリーソフト扱い(/usr/local/unsupported/bin/less)

ファイルの作成や編集を行う場合には UNIX システム標準のエディターである vi を使用します。作成，編集しようとするファイル名が a.f のとき以下のように起動します。

```
% vi   a.f
```

☞ vi エディターの使用方法は「付録 A vi エディターの使い方」を参照して下さい。

入力が完了したら，ファイルに保存して，エディターを終了します。

ファイルの内容を端末の画面に表示するには cat コマンドを使用します。

```
% cat   a.f
c  test
      write(*,*) 'abc'
      end
```

ただし，cat コマンドはファイルの行数が端末の画面の行数より多いと，流れてしまいます。このような場合には，ファイルを 1 画面ずつ表示するコマンドを使用します。

```
% more  a.f
```

ファイルの内容に漢字が含まれている場合には次のコマンドを使用して下さい。

```
% less  a.f
```

☞ more および less は途中でプロンプトに戻る（“q”），1 行前に戻す（“k”），1 行先に進める（“j”）という操作もできます。

これらのコマンドは標準的な UNIX のコマンドですのでコマンドの解説書など一般の書籍が十分参考になります。ただし，オプションや機能には OS によって若干の違いがあるため，詳細はオンラインマニュアル（man コマンド）で調べて下さい。

以下 UNIX コマンドや C シェルを使用する上で知っておきたい機能，利用において便利なコマンドについて説明します。

① 標準入出力

UNIX のコマンドやプログラムは通常，データーの入力（標準入力）がキーボード，実行結果の出力（標準出力）がディスプレイとなっていますが，コマンドやプログラムの実行結果をディスプレイではなく，ディスク上のファイルに出力することができます。

```
% cat   a.f  >  b.f
```

ディスプレイには表示されず，ファイル b.f が作成されて a.f の内容が書き込まれま

す。これを，標準出力を変更（リダイレクト）する，と言います。また，

```
% cat  a.f  >>  b.f
```

としてファイルに追加書きすることもできます。追加書きではなく，上書きするには

```
% cat  a.f  >! b.f
```

とします。

☞ 出力をリダイレクトしてもエラーメッセージだけはディスプレイに表示されます。エラーメッセージは特別な出力先（標準エラー出力）なので，必要な場合は以下のようにリダイレクトします。

```
% cat  c.f  >&  e.f
```

また，標準入力もリダイレクトできます。以下の grep コマンドはキーボードからではなく，ファイルからデーターを入力します。

```
% grep  abc  <  a.f
```

この例ではファイル a.f 中の文字列 abc を含む行を表示します。

☞ 入力が必要なコマンドはリダイレクトしないと入力待ちとなり，プロンプトに戻りません。入力の終わりを指示する場合には Ctrl-D（Ctrl キーを押しながら D キーを押す）を使用して下さい。

```
% grep  abc
```

なお，パイプという機能を使っても同様のことができます。

```
%  cat a.f | grep abc
```

パイプ（|）は標準出力の内容を次のコマンドに渡す役割をします。 a.f の内容は cat の標準出力からパイプを通して grep の標準入力に渡され，文字列を検索した結果は grep の標準出力（ディスプレイ）に表示します。

これらの入出力リダイレクション機能は以下のようにプログラムへのデーター入力や計算結果のファイルへの出力に大変便利ですので憶えておくと良いでしょう。

```
%  ./program < data > result
```

② 初期設定ファイル

C シェルはログイン時に自動的に各利用者のホームディレクトリーにある以下の初期設定ファイルを読み込み，環境設定を行います。初期設定ファイルは利用登録時に作成されます。

```
.cshrc      .login
```

これらのファイルはコマンド検索パス（コマンドの置いてあるディレクトリー名のリスト）等のシェルの環境設定に使用されます。これらのファイルは変更も可能ですが，トラブルの原因となりますので特に必要がない限り変更しないで下さい。なお，センター作成の初期ファイルに戻す場合には以下のファイルをホームディレクトリーにコピーして下さい。

```
/usr/local/skel/.cshrc
/usr/local/skel/.login
```

☞ ピリオドで始まるファイルを ls で見るには以下のようにします。

```
% ls -a
```

③ プロセスの強制終了

実行中のプログラムやコマンドのことをプロセスと言います。時間のかかるプログラムを実行したが一度止めたい，コマンドが入力待ち状態となって先に進まない場合などプロセスを途中で停止したい場合には，

Ctrl-C （Ctrl キーを押しながら C キーを押す）

を使用します。コマンド，プログラムが終了するとシェルのプロンプト % に戻ります。これでも停止しないときは

Ctrl-Z （Ctrl キーを押しながら Z キーを押す）

でプログラムを一時停止し，次のようにプロセスを強制終了します。

```
% ps -e
   PID    TTY  TIME CMD
 319536 pts/11  0:00 ./a.out
 393326 pts/11  0:00 ps
 598224 pts/11  0:00 -csh
% kill -9 319536   (a.out が停止したいプログラム PID=319536 の場合)
```

Ctrl-Z による停止も効かないときは，別の端末からログインして上記操作 kill を行います。

④ オンラインマニュアル

オンラインマニュアルは man コマンドを使用します。引数に調べたいコマンド名を入力すると説明が表示されます。画面が停止したら，スペースキーで続きを読むか，「q」キーでシェルに戻ります。

```
% man ls
```

☞ 端末の漢字コードに合わせて次のように設定することにより日本語で表示することもできます（f90, f77, cc, CC 等，一部のコマンドのみ）。

```
csh の場合
% setenv LANG Ja_JP            (元に戻す場合は % unsetenv LANG)
sh の場合
$ LANG=Ja_JP ; export LANG     (元に戻す場合は $ unset LANG)
```

※マニュアル Web 閲覧サービス

Web ブラウザにてマニュアルの閲覧が可能です（言語，ライブラリー，チューニングガイド等）。SSH による sr11000-s へのログインと，SSH のポート転送機能を用いて，ローカルホストのポート（空いている任意のポート，例えば 8080 番）宛の通信を "manual" マシンのポート 80 番に転送する設定を行い，次の URL を参照します。

```
http://localhost:8080/manual-j/index.html
```

☞ 詳細は Web ページまたは「付録K マニュアル Web 閲覧サービス」を御覧下さい。

⑤ 電子メール

電子メールの送信および受信を行うことができます。
メールアドレスは「ログイン名@sr11000-s.cc.u-tokyo.ac.jp」となります。

送信

```
% mail p08000@sr11000.cc.u-tokyo.ac.jp（送り先のメールアドレス）
Subject: test  ← 件名
test mail      ← 本文
.              ← .（ピリオド）のみの行
Cc:            ← カーボンコピー（不要なら空欄）
%
```

標準入力を利用してファイル testmail の内容を送ることもできます。

```
% mail p08000@sr11000-s.cc.u-tokyo.ac.jp < testmail
```

受信（参照）

```
% mail
Mail [5.2 UCB] [AIX 5.X]  Type ? for help.
"/var/spool/mail/p08000": 1 message 1 new
>N  1 p08000          Fri Mar 21 11:53  12/399  "test"
? 1   ←メール番号
 --ヘッダーと本文が表示されます--
? q  ←コマンドの終了
Saved 1 message in /batch/p08000/mbox
%
 --メールは mbox というファイルに保存され，メールスプールから削除されます--
```

なお，mbox に保存されたメールを読むときは以下のようにします。

```
% mail -f mbox
```

サブコマンド	機能	例
番号	メールの表示	? 2
h	メール一覧表示	? h
s	ファイルへ保存	? s file
d	メールの削除	? d 1
x	コマンドの終了（何もせず終了）	? x
q	コマンドの終了（mbox へ保存）	? q

☞ 本システムは POP3，IMAP4 などをサポートしていませんので，メールクライアントソフトウェア（Eudora，Outlook など）は使用できません。

⑥ ファイル転送

ファイル転送には scp コマンドを使用します。認証および通信を暗号化したファイル転送（コピー）が可能です。

```
scp  ログイン名@sr11000-s.cc.u-tokyo.ac.jp:リモートファイル名   ローカルファイル名
scp  ローカルファイル名   ログイン名@ sr11000-s.cc.u-tokyo.ac.jp:リモートファイル名
```

利用者 p08000 が sr11000-s のホームディレクトリー（/batch/p08000）にあるファイル file1 をローカルファイル file2 にコピーする例

```
% scp p08000@sr11000-s.cc.u-tokyo.ac.jp:file1  file2
```

ローカルファイル file1 を sr11000-s のホームディレクトリーにコピーする例

```
% scp  file1 p08000@sr11000-s.cc.u-tokyo.ac.jp:file2
```

☞ ftp コマンドによる接続はできません。

⑦ CPU リソース情報

CPU の使用に関する情報は la コマンドで出力できます。

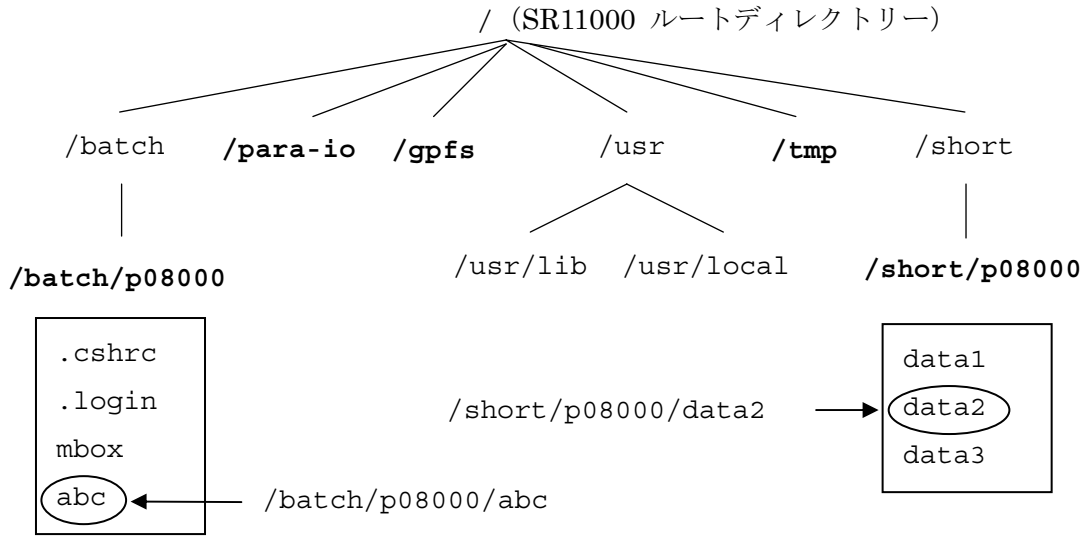
```
% la
cpu resource for p08000  (HH:MM:SS)
     system         limit       used( this month)         remain
 SR11000     596523:14:07  143:27:54(    0:04:21)   596379:46:13
               CPU 時間上限値   CPU 使用時間(年間)  CPU 使用時間(当月)      残り CPU 時間
```

5. ファイルとディレクトリー

UNIX システムにおいてはプログラムやデーター，テキストなどの情報は「ファイル」という単位でディスク上に格納します。格納する場所は「ディレクトリー」と呼ばれるツリー構造によってグループ分けされており，利用者ごとに区画が割り当てられています。以下の例のようにファイルの保存されている場所（パス）はディレクトリーによって指し示すことができます。

/	ルートディレクトリー
/batch	長期保存ファイルディレクトリー
/batch/p08000	利用者 p08000 のホームディレクトリー
/batch/p08000/work	利用者の作成した work ディレクトリー
/batch/p08000/work/program.f	利用者のファイル program.f

SR11000 は以下の図のようなディレクトリー構成となっています。利用者がファイルを作成できるのは利用者用に開放されたディレクトリー（例えば図中の太字部分）です。



利用者が使用できるディレクトリーとして以下を用意しています。

/batch/{ログイン名}	（ホームディレクトリー）
/short/{ログイン名}	
/para-io/{ログイン名}	
/tmp	
/gpfs/{ログイン名}	（※申込が必要）

/batch

/batch はホームディレクトリーであり，デフォルトのディレクトリーです。ここに作られたファイルは利用者自身で削除するまで保存されます。

/batch ディレクトリーのファイル使用量および上限値は以下のコマンド（la）で確認できます。

```
% la -d
disk resource    user(p08000)
 file system      limit(MB)  occupied(MB)
  /batch             10240           1
 ディレクトリー名     ファイル使用量    ファイル上限値
```

☞ グループおよび研究室コースの場合は以下の表示となります。

```
disk resource    user(g00000)              group(g00)
 file system      limit(MB)  occupied(MB)  limit(MB)  occupied(MB)
  /batch                --           1       204800           1
 ディレクトリー名     ユーザー上限値    ユーザー使用量  グループ上限値  グループ使用量
```

ディスク資源制限は利用者ごと（グループおよび研究室コースではグループごと）にファイル使用量上限値（quota）で制限しています。上限値を超えてファイルを作成しようとすると以下のエラーとなりますので，不要なファイルの削除や上限値の増量（申請が必要です）で対処して下さい。

```
Disc quota exceeded
```

/short

インタラクティブから読み込み書き込みともに可能です。ここに作られたファイルは作成（または最終更新）から 5 日後に削除されます。

/para-io

バッチジョブから読み書きが可能で，インタラクティブからは読み込みのみ可能です。ここに作られたファイルは作成（または最終更新）から 5 日後に削除されます。

/tmp

システムおよび利用者が短時間の作業用に使用するディレクトリーです。ここに作られたファイルは 1 日以内に削除されます。

/gpfs

AIX 標準のファイルシステムである GPFS (General Parallel File System) により構築されています。MPI-IO を利用する場合等， HSFS では動作しないプログラムがまれにありますので，その場合は，この /gpfs をご利用ください。ただし，システム登録時（利用者番号発行時）に ディレクトリは作成いたしませんので，ご利用を希望される場合は，センター 宛メール (uketsuke @ cc.u-tokyo.ac.jp) でその旨ご連絡をお願いいたし ます。

インタラクティブから読み込み書き込みともに可能です。ここに作られたファイルは作成（または最終更新）から 5 日後に削除されます。

以下は各ファイルシステムの比較です。

ディレクトリ名	ファイル上限値	保存期間	インタラクティブから	申込	利用負担金
/batch	あり	--	Read/Write	不要	利用負担金に含まれる※
/short	なし	5 日	Read/Write	不要	無料
/para-io	なし	5 日	Read only	不要	無料
/gpfs	なし	5 日	Read/Write	必要	無料

※パーソナルコースについては 10GB，グループコースについては 利用者番号 10 個までにつき 200GB は，利用負担金に含まれます。 それ以上必要な場合は，追加オプションにより，申込可能です。 利用負担金は 600 円/(GB*年) です。

/para-io と /short では，ファイル上限値(quota)制限がないこと， 保存期間が 5 日であることは同じですが，インタラクティブからの 読み書きにおいて異なります。

/batch，/short，/para-io の 3 つのファイルシステムは，Hitachi Striping File System（HSFS）で構築されています。この HSFS は，ストライピングファイル機構として，ファイルストライプ機能とブロックストライプ機能があります。/batch と /short は，ファイルストライピング機能，/para-io は， ブロックストライピング機能により構築されています。

/para-io は，単一ファイルに対してプログラムから一回に書き込むサイズを大きく(8MB 程度以上)すれば，性能が生かせます。一方で，書き込みサイズが小さい場合や，複数のファイルへの書き込みの場合は，性能が出ないため，/short をお使いください。（サイズが小さい場合は，処理オーバーヘッドが大きくなり，並列化による効果を期待できません。また，複数ファイルの場合は，ディスク装置への入出力要求がランダム になります。）

ディスク資源は全ての利用者で共有しているため，無駄なファイルの保存はディレクトリーを圧迫します。不要になったファイルは残しておかないようお願い致します。また，ファイル数にも限界があります。ファイルの容量が小さくても数が多い場合には一つにまとめる（tar コマンドでアーカイブする）などディスクを効率的に利用するよう心掛けて下さい。なお，本センターではファイルのバックアップを取っていません。ファイル保存にあたっては万一の事故に備えて大切なファイルは各自でバックアップを取っておかれますようお願い致します。

6. 要素並列とジョブの形態

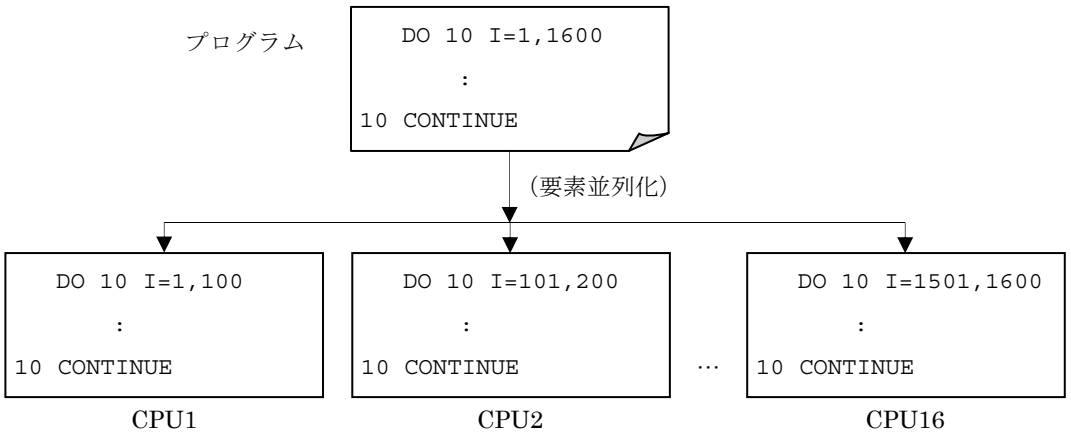
6.1. 要素並列処理

各ノードは，並列処理単位（スレッド）に分割したプログラムをノード内の複数の CPU で並列実行する「要素並列」処理機能を備えています。コンパイルオプションの指定やソースプログラム中に指示文を記述することにより，コンパイラーは要素並列化を施したオブジェクトを生成します。以下に要素並列化変換の例を示します。

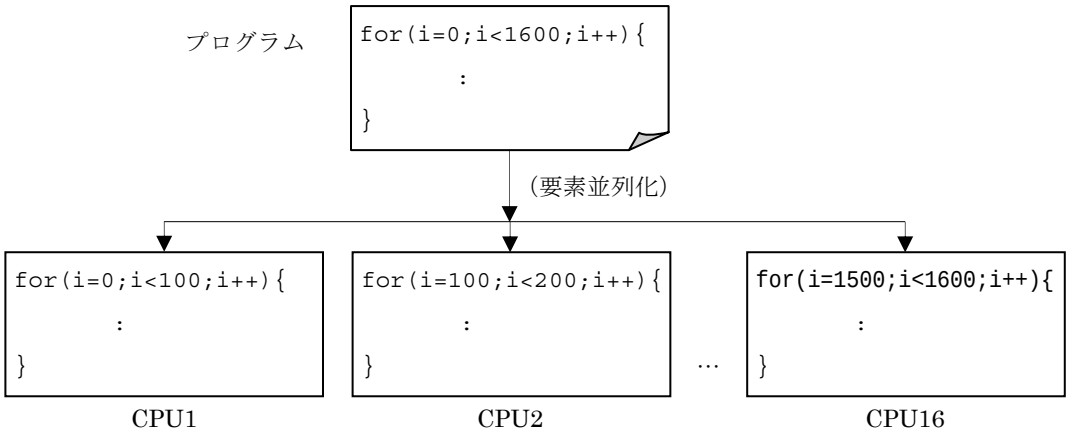
● ループ要素並列化（自動並列化）

DO ループをスレッドに分割し，複数 CPU で並列に実行します。コンパイルオプションの指定によりコンパイラーが判断して自動的に変換します。

Fortran の場合



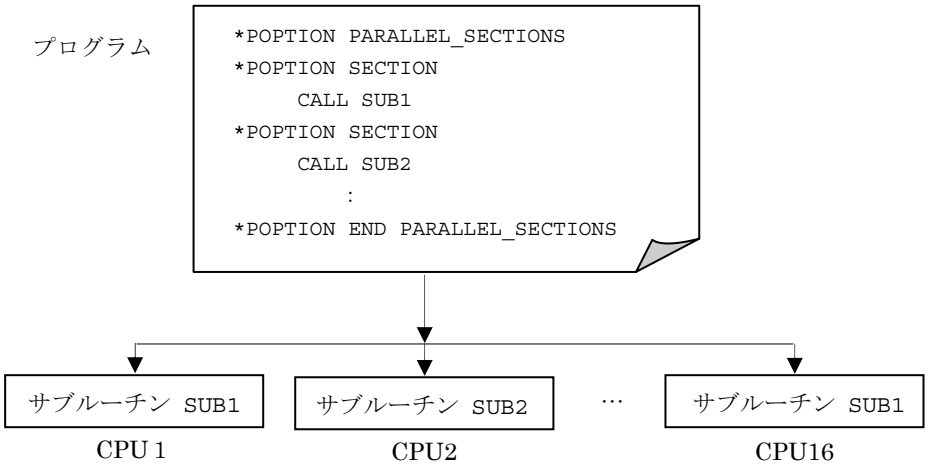
C の場合



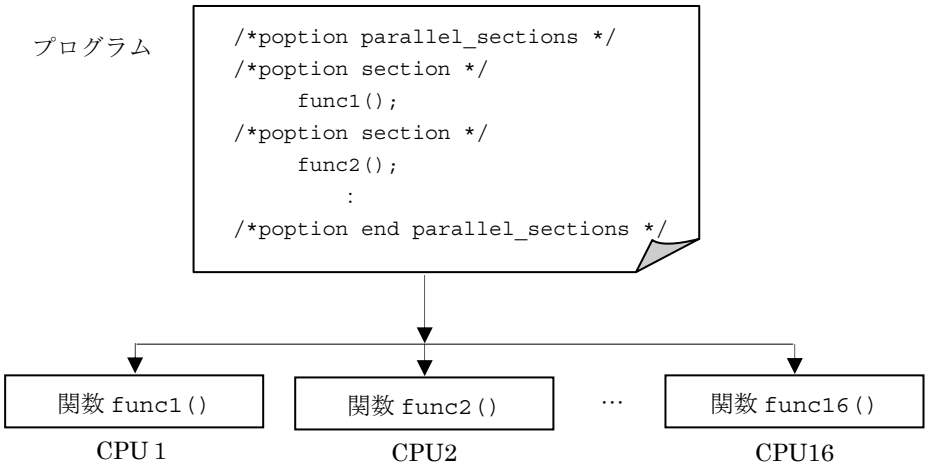
● SECTION 型要素並列化

関数，サブルーチンなどの文の集まりをスレッドに分割し，複数 CPU で並列に実行します。ソースプログラム中に指示文を記述することで並列化を指示します。

Fortran の場合



C の場合



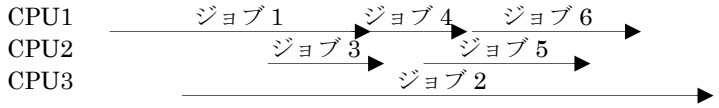
6.2. ジョブの形態

SR11000 の各ノードは複数の CPU で構成されているため，要素並列処理だけでなく，以下に示すジョブ処理が可能です。さらにメッセージ通信ライブラリー（MPI）を組み合わせて使用することで複数ノードによる大規模な並列プログラムを実行することができます。

☞ MPI については「11.2. MPI」を御覧下さい。

① スカラージョブ

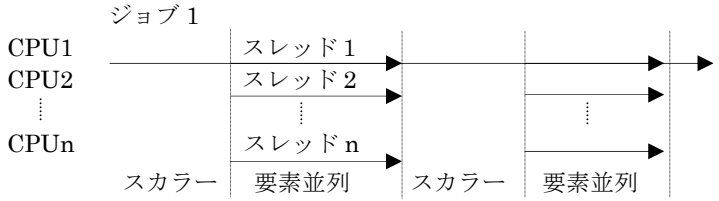
スカラープログラムを実行するジョブです。1CPU のみ使用するので他のジョブとノードを共有し，ノード内で複数実行されます。



☞ スカラージョブクラス（A～D）で実行します。

② 要素並列ジョブ

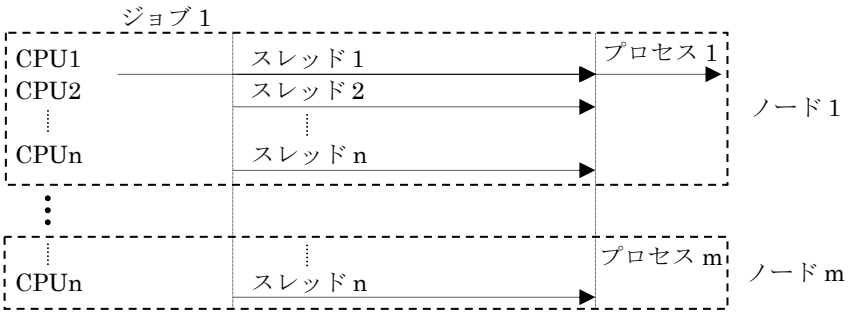
要素並列プログラムを実行するジョブです。要素並列化により分割されたスレッドが各 CPU に配置され並列に実行します。



☞ 並列ジョブクラス（P001, S1, H1,等）で実行します。S1, H1 は 8CPU/ノード用のジョブクラスです。これらのジョブクラスのジョブはノードを専有します。

③ ノード内要素並列＋ノード間 MPI ジョブ

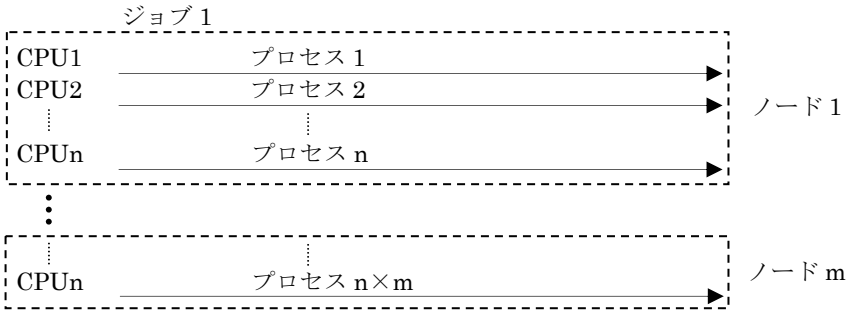
ノード内は要素並列，ノード間は MPI を使用して通信します。m 台のノードを使用すると m プロセス並列が実現できます。



☞ 並列ジョブクラス（P004～P064, debug 等）で実行します。これらのジョブクラスのジョブはノードを専有します。

④　ノード内およびノード間 MPI ジョブ

MPI のプロセスが CPU 数分生成されます。ノード内は最大 n（16 または 8）プロセスの並列実行ができます。m 台のノードを使用すると n×m プロセス並列が実現できます。



☞ 並列ジョブクラス（P001～P064, S1, H1, debug 等）で実行します。これらのジョブクラスのジョブはノードを専有します。実行にはジョブタイプ SS（#@$-J SS）の指定が必要です。または, 16CPU/ノードの場合 T16（#@$-J　T16）, 8CPU/ノードの場合 T8（#@$-J　T8）でも同じです。

7. コンパイルと実行

7.1. Fortran

Fortran プログラムのコンパイル，リンクは f90（または f77）コマンドを使用します。

```
f90（または f77）
    ファイル名..
    [-c]
    [-o ロードモジュールファイル名]
    [-コンパイルオプション..]
    [-リンケージオプション..]
    [-オプション..]
    [-I インクルードディレクトリー名]
    [-L ライブラリーディレクトリー名]
    [-l ライブラリー名]
    [-i,言語仕様拡張オプション]
```

☞ 詳細は「10. Fortran のコンパイルと実行の詳細」および以下のマニュアルを御覧下さい。
SR11000 マニュアル 「最適化 FORTRAN90 使用の手引（3000-3-C22)」
「最適化 FORTRAN77 使用の手引（3000-3-C24)」

使用例

```
% f90  -parallel  main.f  sub.f
```

Fortran プログラムのソースファイル名は通常 .f で終わる名前を使用します。(大文字で .F とした場合は C 言語プリプロセッサーを通した後，Fortran のコンパイルを行います。また，Fortran90 プログラムのソースファイル名には，.f90 の拡張子を使わないとコンパイル出来ない場合があります。）コンパイルが正常に終了するとリンクが行われ，ロードモジュール a.out が作成されます。-o オプションを使用すれば生成するロードモジュール名を指定することができます。

```
% f90 -parallel -o loadmodule main.f sub.f
```

なお，オプションは指定した順番（左から右）に処理されます。ファイル名の前後に置くことができますが，ライブラリー名を指定する-l オプションはライブラリーを呼び出すプログラムより後に指定して下さい。

MATRIX/MPP ライブラリーを呼び出す例

```
% f90 -parallel -o loadmodule main.f sub.f -lmatmpp
```

オブジェクトモジュール（*.o）を生成する場合は，-c オプションを指定します。

```
% f90 -parallel -c main.f sub.f
```

オブジェクトモジュールをファイル名として指定するとリンクを行い，ロードモジュールを生成します。

```
% f90 -parallel -o loadmodule main.o sub.o -lmatmpp
```

MPI 機能を使用したプログラムをコンパイルする場合には以下のコマンドを使用します。

```
mpif90（または mpif77）
    ファイル名..
    [-c]
    [-o ロードモジュールファイル名]
    [-オプション]
```

使用例

```
% mpif90 sample.f -o program
```

これらのコマンドは f90 または f77 コンパイラーで MPI 機能を使用する場合に必要なヘッダーファイルやリンクするライブラリー，コンパイルオプションを自動的に設定します。-オプションを指定してコンパイラーにオプションを渡すこともできます。

☞ 詳細は「11.2. MPI」を御覧下さい

7.2. C，C++

C プログラムのコンパイルは cc，C++プログラムは sCC コマンドを使用します。

```
cc または sCC
    [オプション]
    ファイル名..
```

☞ 詳細は SR11000 マニュアル 「最適化 C 使用の手引（3000-3-C32)」
「最適化標準 C++使用の手引（3000-3-C33)」

使用例

```
% cc -Os +Op -parallel main.c sub.c -lm
```

C プログラムのソースファイル名は通常 .c で終わる名前を使用します。C++プログラムの場合は .C です。コンパイルが正常に終了するとリンクが行われ，ロードモジュール a.out が作成されます。-o オプションを使用すれば生成するロードモジュール名を指定することができます。

```
% cc -Os +Op -parallel -o loadmodule main.c sub.c -lm
```

なお，ライブラリー名を指定する-l オプションはライブラリーを呼び出すプログラムより後に指定して下さい。-c オプションを指定した場合はオブジェクトモジュール.o を生成します。

```
% cc -c -Os +Op -parallel main.c sub.c
```

オブジェクトモジュールをファイル名として指定するとリンクを行い，ロードモジュールを生成します。

```
% cc -Os +Op -parallel -o loadmodule main.o sub.o -lm
```

MPI 機能を使用した C（または C++）プログラムをコンパイルする場合には以下のコマンドを使用します。

```
mpicc または mpiCC
    ファイル名..
    [-c]
    [-o ロードモジュールファイル名]
    [-オプション]
```

このコマンドは cc（または sCC）コンパイラーで MPI 機能を使用する場合に必要なヘッダーファイルやリンクするライブラリー，コンパイルオプションを自動的に設定します。-オプションを指定してコンパイラーにオプションを渡すこともできます。

☞ C++ で STL（標準テンプレートライブラリー）を使用する場合は，次のとおり指定してください。

```
% sCC  sample.cpp  -I/opt/STLport-4.5/stlport  -L/opt/STLport-4.5/lib
  -lstlport_sCC  -lm
```

7.3. 実行

ロードモジュールは実行ファイルとも呼ばれ，ロードモジュール名をコマンドのように入力（バッチジョブの場合はスクリプトファイルに記述）することでプログラムを実行することができます。

```
./a.out  またはロードモジュール名
```

なお，Fortran プログラムの場合は実行時オプションを与えることができます。

```
./a.out  またはロードモジュール名
     [-F,'実行時オプション,..']
```

☞ 詳細は「10.3. 実行時オプション」および以下のマニュアルを御覧下さい。

SR11000 マニュアル　「最適化 FORTRAN90 使用の手引（3000-3-C22)」
「最適化 FORTRAN77 使用の手引（3000-3-C24)」

使用例

```
% ./a.out  -F,'port(stduf)'
```

☞ 要素並列化したロードモジュールは，インタラクティブおよびスカラージョブクラスではエラーとなり実行できません。また，複数ノードを使用した並列実行もできません。バッチジョブで並列処理用のジョブクラスを使用して下さい。なお,ジョブクラスについては「9.2. ジョブクラスとキュー」を御覧下さい。

```
% ./a.out
KCHF023P The number of threads for parallel execution exceeds the limit. The
number of available threads is 0.
0x1000f530 h_comp_threaderr+0x7c
0x10010840 _hf_para_init+0x3b8
0x10016ed0 __hf_init+0x2ebc
0x100006e4 _hf_mplg+0x1e4
0x10000424 MAIN+0x24
0x100001fc __start+0x9c
```

・MPI 機能を使用せず，各ノードにプログラムを配置して実行する場合は prun コマンドを使用します。指定したノード数でプログラムを実行します。

```
prun
    [-n ノード数]
    ロードモジュール名
```

使用例（ジョブスクリプト例）

```
% cat  job.csh
#@$-q debug
#@$-N 2
cd test
prun  -n  2  ./a.out
```

または

```
prun
    [-f 定義ファイル名]
```

使用例（ジョブスクリプト例）

```
% cat  job.csh
#@$-q debug
#@$-N 2
cd test
prun  -f  sample.def
```

```
定義ファイル
# sample.def          [# コメント]
*2 ./a.out            [*ノード数] [プログラム名]
```

☞ 詳細は「11.1. 並列実行（prun コマンド）」を御覧下さい。

なお, 実行にはバッチジョブを使用して下さい。インタラクティブでは実行できません。

・MPI 機能を使用したプログラムを実行する場合には mpirun コマンドを使用します。

```
mpirun
    ロードモジュール名
```

使用例（ジョブスクリプト例）

```
% cat  job.csh
#@$-q debug
#@$-N 4
#@$-J SS    ← #@$-J  T16  でも良い
cd test
mpirun  ./a.out
```

☞ 詳細は「11.2. MPI」を御覧下さい。

プログラムの実行に必要なノードを確保して，各ノードにプロセスを生成します。上記の例では 4 ノードを確保し，64 プロセス（1 ノードに 16 プロセス）を生成します。

なお, 実行にはバッチジョブを使用して下さい。インタラクティブでは実行できません。

バッチジョブの使用方法については「9. NQS（バッチジョブ）」を御覧下さい。

8. システム資源

SR11000 は多くの利用者が同時に使用するため，資源専有や性能低下を防ぐ目的で各種システム資源に上限値を設けています。ここでは利用者が使用する上でのシステム資源設定について述べます。

8.1. CPU 資源

インタラクティブ環境（TSS）ではログイン時間を制限しています。ログインから 18 時間が経過，または無入力（キー入力がない）の状態で 2 時間が経過した場合は自動的にログアウトするように設定しています。さらにプロセスごとに CPU 資源制限をしています。この制限値は limit コマンドで確認することができます。

```
% limit
cputime         1:00:00             CPU 時間制限
```

プログラムの実行はインタラクティブ環境でも可能ですが，本格的なジョブの実行はバッチジョブを使用して下さい。バッチジョブ環境の CPU 資源制限はスクリプトファイル中のオプションで設定します。設定方法は「9. NQS（バッチジョブ）」を御覧下さい。

```
#@$-lT          リクエスト（ジョブ）の経過時間制限
#@$-lt          プロセスごとの CPU 時間制限
```

最大値はジョブクラス（キュー）ごとに異なります。Web ページ，スーパーコンピューティングニュース表紙裏または「付録H ジョブクラス制限値」を御覧下さい。

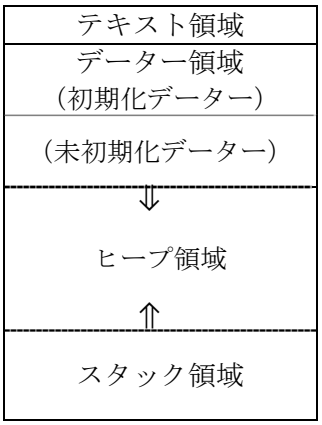
8.2. メモリー資源

メモリー資源はジョブリクエスト，あるいはプロセス（実行中のプログラム）ごとに制限します。このため，ジョブクラスおよびログインセッションに 1 ノード当たりのメモリー使用量の制限値（ジョブクラス制限値を参照）を設定して資源管理を行っています。

利用者が上限値を変更できるのはジョブまたはセッションが使用するメモリーの大きさ（仮想メモリーサイズ）とデーター領域，スタック領域です。例えば Fortran プログラムでは配列などの変数は通常データー領域に配置しますので不足する場合は仮想メモリーサイズとデーター領域を拡張します。参考までにプロセスごとのメモリー構成を以下に示します。ヒープ領域は仮想メモリーサイズからテキスト領域，データー領域，スタック領域を除いた残り分が使われます。

テキスト領域（.text）
　プログラム（実行命令等）を配置します。
データー領域
- 初期化データー（.data）
　初期値のある静的変数を配置します。
- 未初期化データー（.bss）
　初期値なしの静的変数を配置します。

ヒープ領域
　プログラム実行時に動的に配置します。
スタック領域
　ローカル変数，関数呼び出し時の引数，
　戻り値等を配置します。



☞ プログラムの大きさ（.text+.data+.bss 他）は size コマンドで確認できます。
```
% size -f  a.out
```

インタラクティブ環境（TSS）のメモリー資源制限は limit コマンドで確認することができます。システムで設定した上限値以内なら変更も可能です。

```
% limit
cputime         1:00:00             CPU 時間制限
filesize        unlimited           ファイルサイズ
datasize        524288 kbytes       データーサイズ
stacksize       32768 kbytes        スタックサイズ
coredumpsize    0 kbytes            コアダンプサイズ
```

データー領域が不足するときはデーターサイズを拡大または unlimited とします。ただし unlimited としてもシステムの設定値を超えることはできません（スタック領域も同様です）。

```
% limit  datasize  unlimited
% limit  stacksize 64M
```

バッチジョブ環境のメモリー資源制限はスクリプトファイル中のオプションで設定します。

```
#@$-lM          リクエスト（ジョブ）ごとのメモリーサイズ
#@$-lm          プロセスごとのメモリーサイズ
#@$-ld          プロセスごとのデーターサイズ
#@$-ls          プロセスごとのスタックサイズ
```

オプションの設定方法の詳細は「9. NQS（バッチジョブ）」を御覧下さい。

以下はメモリー使用に関する注意事項です。

(1) 複数ノードのジョブは確保したノード数分のメモリーを使用できます。ただし，プロセスが各ノードに分散して配置されるため，全メモリーを使用する大規模配列を単純に定義する

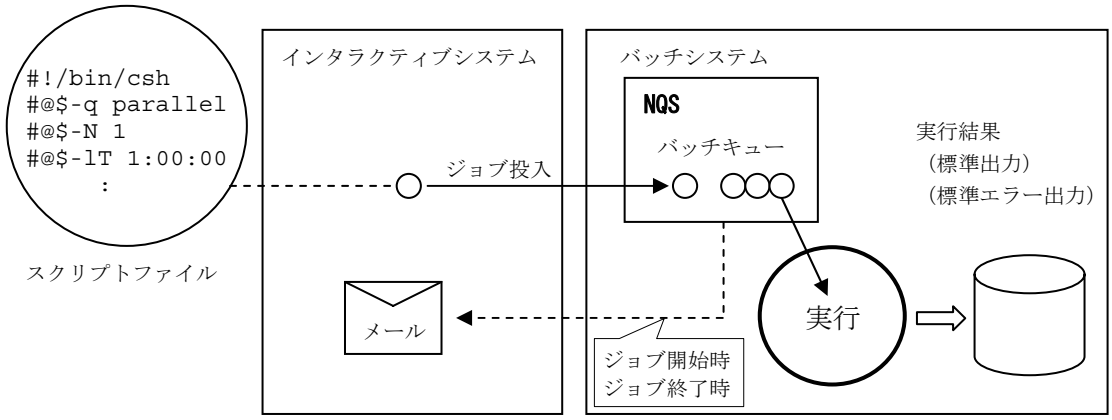
ことはできず，プログラム上で配列を各ノードに分散する必要があります。

(2) C プログラムでは自動変数（auto）がスタック領域を使用しますのでスタックが不足することがあります。この場合はスタック領域を拡張します。ただし，スタック領域は小さいので配列は静的変数（static）で宣言し，データー領域を使用するようにして下さい。

9. NQS（バッチジョブ）

9.1. NQS とは

NQS（Network Queuing System）はジョブをバッチシステムにキューイングする機能です。利用者が投入したジョブは，キューと呼ばれる待ち行列で先に実行しているジョブの終了を待ち，順番がきた時点で実行を開始します。対話形式の TSS とは異なり，実行結果はファイルに残りますのでジョブ投入後はログインしている必要がありません。



9.2. ジョブクラスとキュー

NQS ではジョブを効率よく実行するためにジョブの種類や大きさで分類したジョブクラスを設定しています。ジョブクラスごとに「バッチキュー」と呼ばれるキューが設けてあり，CPU 数や最大ノード数，制限時間，メモリー容量等が異なります。

SR11000 では以下のジョブクラスを用意しています。なお，バッチキューA～D および P001～P064 については，バッチキューへの橋渡しを行う「パイプキュー」で指定する必要があります。この場合，実行時間制限やノード数の指定によりバッチキューが自動的に決定されます。

スカラージョブクラス

スカラープログラムを実行するジョブクラスです。ジョブの実行時間（経過時間）でクラス分けしています。

パイプキュー名　single（バッチキュー名　A，B，C，D）

注）バッチキュー名の明示的な指定（`#@$-q` バッチキュー名）はできません。

並列ジョブクラス

・16CPU/ノード ジョブクラス

要素並列プログラムの実行および複数ノードを使用したジョブの実行を行うジョブクラスです。使用ノード数でクラス分けしています。

パイプキュー名　parallel（バッチキュー名　P001，P004，P008，P016，P064）

注）バッチキュー名の明示的な指定（#@$-q バッチキュー名）はできません。

・8CPU/ノード ジョブクラス

1 ノードが 8CPU で構成されています。バッチキューS4 は研究室コースのみ利用可能です。

バッチキュー名　S1, S4, H1

・リソース追加オプション専用ジョブクラス（16CPU/ノード）

利用するためには，オプションの申し込みが必要です（月単位)。それぞれのキューの各月の上限は 10 名です。詳細は Web ページまたは「付録G 計算リソース追加オプション」を御覧下さい

バッチキュー名　Q001, Q004, Q008

・デバッグ用ジョブクラス（16CPU/ノード）

4 ノードまで実行可能です。ジョブ実行時間は最大 5 分です。

バッチキュー名　debug

☞ 1 ユーザーの投入可能ジョブ数は 8，実行可能ジョブ数はキュー毎に 1,最大でも同時実行数は 2 となります（リソース追加オプション専用ジョブクラスを除く)。

☞ ジョブクラス制限値については Web ページ，スーパーコンピューティングニュース表紙裏または「付録H ジョブクラス制限値」を御覧下さい。申し込みのコースによりキューの使用に制限がありますので御注意下さい。

9.3. スクリプトファイルの作成

バッチジョブを実行するためには，まずジョブを作成する必要があります。ジョブの作成とは実行するプログラムの手順書を書くようなものであり，その手順書のことをスクリプトファイルと呼びます。スクリプトファイルは UNIX で一般的に使われるシェルスクリプトを使用して記述します。

```
#!/bin/csh      ← C シェルスクリプトで記述されていることを意味する
#@$-q parallel          ← オプション
#@$-N 1                   (#@$-で始まる行)
#@$-lT 1:00:00
      :
cd test                 ← コマンド
./a.out < data > result   (シェルスクリプト)
      :
```

スクリプトファイルの書き方の例です。

```
#!/bin/csh                                  #!/bin/csh
#@$-q パイプキュー名                        #@$-q parallel
#@$-N ノード数                              #@$-N 1
#@$-lT 実行時間制限値              ⇒        #@$-lT 1:00:00
#@$-lM メモリーサイズ制限値                 #@$-lM 112GB
cd ディレクトリー                           cd test
プログラム名 < 入力ファイル > 出力ファイル  ./a.out < data > result
```

オプション

オプション部分の 1 カラム目は全て#記号（シェルスクリプトではコメントを意味する）ではじまり，@$に続くオプションによってキューの選択や資源制限を行うことができます。なお，オプションは大文字，小文字を区別しますので意識して記述して下さい。

最初にジョブを投入するキュー名を指定します（パイプキュー名またはバッチキュー名を指定して下さい)。

```
#@$-q   キュー名
```

ここで指定するキュー名は以下のとおりです。

```
single, parallel, S1, S4, H1, debug, Q001, Q004, Q008
```

次に複数ノードを使用する並列ジョブクラス（parallel, S4, debug, Q004, Q008）の場合はノード数を指定します。スカラージョブクラス（single）の場合この指定は不要です。paralell の場合はこの指定によりバッチキュー（P001～P064）が決定されます。

```
#@$-N   ノード数
```

続いてジョブの実行時間（時間：分：秒 や 分：秒 での指定が可能）の見込みを設定します。この時間を過ぎると実行中のジョブでも強制的に終了します。スカラージョブクラス single の場合はこの指定によりバッチキュー（A～D）が決定されます。

```
#@$-lT   実行時間制限値（経過時間）
```

☞ 本センターの-lT オプションは経過時間制限値（-lE と同意）としています。メーカー標準の-lT（CPU 時間制限値）とは異なりますので御注意下さい。

☞ 「#@$-lT …」を省略した場合，実行時間制限値は single キューでは 10 分，それ以外のキューでは各キューの最大制限時間となります。

☞ 実行時間制限値の指定はジョブスケジューリングに影響します。詳細は「付録Ｆ　ジョブスケジューリングシステムについて」を御覧下さい

さらにジョブ全体で使用する 1 ノードあたりのメモリーサイズ制限値を設定します。

```
#@$-lM   メモリーサイズ制限値
```

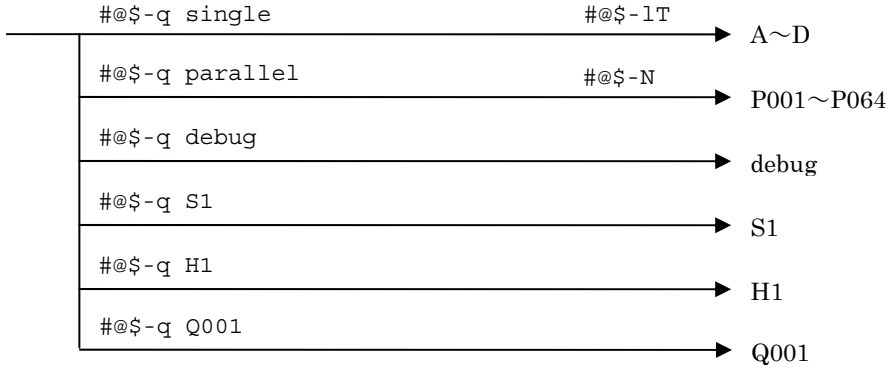
☞ 単位は MB（Megabyte）や GB（Gigabyte）等が使用できます。

☞ 「#@$-lM …」を省略した場合，メモリーサイズ制限値は single キューでは 2GB，それ以外のキューではノードを専有するため，各キューの最大メモリー容量となります。single キュー以外では指定する必要はありません。

主なオプションと指定例の一覧です。

オプション	内　　容	指　定　例
#@$-q	キュー名　（パイプキュー名）	#@$-q parallel
#@$-N	ノード数	#@$-N 4
#@$-lT	実行時間制限	#@$-lT 1:00:00
#@$-lM	メモリーサイズ	#@$-lM 4GB
#@$-J	ジョブタイプ	#@$-J SS　　または #@$-J T16
#@$-e	標準エラー出力ファイル名	#@$-e file.e
#@$-o	標準出力ファイル名	#@$-o file.o
#@$-eo	標準エラー出力の内容を標準出力ファイルに出力	#@$-eo
#@$-mu	メールの送信先	#@$-mu xxx@xxx.jp
#@$-lc	コアファイルサイズ	#@$-lc 0MB
#@$-ld	プロセス毎のデーターサイズ	#@$-ld 2GB
#@$-ls	プロセス毎のスタックサイズ	#@$-ls 192MB
#@$-lm	プロセス毎のメモリーサイズ	#@$-lm 2GB
#@$-lt	プロセス毎の CPU 時間制限	#@$-lt 30:00
#@$-nr	バッチシステム障害時にジョブを再投入しない	#@$-nr

バッチキュー（ジョブクラス）はこれらのオプションの組み合わせで決定します。図に示すようにパイプキューが single の場合はジョブの実行時間でバッチキューが決定されます。parallel の場合はノード数でバッチキューが決定されます。



以下にオプションの記述例を挙げます。

スカラージョブの例

```
#!/bin/csh
#@$-q  single
#@$-lM 4GB
#@$-lT 00:30:00
```

要素並列ジョブの例

```
#!/bin/csh
#@$-q  parallel
#@$-N  1
#@$-lT 01:00:00
```

並列ジョブ（複数ノード）の例

```
#!/bin/csh
#@$-q  parallel
#@$-N  4
#@$-lT 01:00:00
```

ノード内 MPI ジョブ（複数ノード）の例

```
#!/bin/csh
#@$-q  parallel
#@$-N  2
#@$-lT 01:00:00
#@$-J  SS    ← T16 でも良い
```

☞ MPI については「11.2. MPI」を御覧下さい。

1 ノードあたりのプロセス数の指定が可能で，ノード内で要素並列化とＭＰＩ並列化を混在させることができます。なお，その際，要素並列化で起動されるスレッド数には注意が必要です。指定方法は以下のとおりです。

・プロセス数の指定

バッチジョブのスクリプトファイルに以下を記述します。1 ノードの CPU 数（キューにより異なり，8 または 16）を設定した場合は ″#@$-J SS″ と同意になります。CPU 数を超える値は指定できません。より詳しい使い方は，「付録Ｃ　複数プログラムの実行方法と環境変数について」をご覧ください。

#@$-J Tn（n：1 ノードあたりのプロセス数）

例）

```
#@$-J T4  →  4 プロセス
```

・スレッド数の指定

環境変数 HF_PRUNST_THREADNUM にスレッド数（1～16）を設定します（最適化 FORTRAN90/77，最適化 C，最適化標準 C++ 共通）。

″1 ノードあたりのプロセス数×スレッド数″ が 1 ノードの CPU 数を超えると実行効率が低下する場合があります。1 ノードの CPU 数を超えないように設定してください。

setenv HF_PRUNST_THREADNUM n（n：スレッド数）

例）

```
setenv HF_PRUNST_THREADNUM 4  →  4 スレッド
mpirun ./a.out
```

※最適化 FORTRAN90/77 のみ，実行時オプション PRUNST(THREADNUM(n)) による指定も可能です。

例）

```
mpirun ./a.out -F’ PRUNST(THREADNUM(4))’  →  4 スレッド
```

・スクリプトファイル記述例

＊1 ノード，4 プロセス，4 スレッド #@$-q debug #@$-N 1 #@$-J T4 setenv HF_PRUNST_THREADNUM 4 mpirun ./a.out（要素並列化モジュール）	＊1 ノード，4 プロセス，要素並列化なし #@$-q debug #@$-N 1 #@$-J T4 mpirun ./a.out（非要素並列化モジュール）
＊4 ノード，8 プロセス，8 スレッド #@$-q debug #@$-N 4 #@$-J T2 setenv HF_PRUNST_THREADNUM 8 mpirun ./a.out（要素並列化モジュール）	＊2 ノード，32 プロセス，要素並列化なし #@$-q debug #@$-N 2 #@$-J SS　　（または ″#@$-J T16″ ） mpirun ./a.out（非要素並列化モジュール）
＊1 ノード，1 プロセス，16 スレッド #@$-q debug #@$-N 1 ./a.out（要素並列化モジュール）	

コマンド

コマンド部分にはバッチジョブで実行したいコマンドを並べて書きます。書き方はシェルスクリプトの記述です(シェルスクリプトには C シェル, Bourne シェル等があります)。

NQS はこのコマンド部分をホームディレクトリーで実行します。このため，プログラムやデーターがホームディレクトリー以外にあるときにはディレクトリーを移動するか，パスを指定しなければなりません。例えば work（/{ホームディレクトリー}/work）というディレクトリーにある実行ファイル a.out を実行するときには

```
cd work
./a.out
```

とスクリプトファイルに記述します。ここで，標準入力ファイル data が必要な場合は，

```
./a.out <  data
```

と書くことができます。さらに，ジョブの実行結果は所定のファイル（「9.5.ジョブの結果」参照）に出力されますが，例えばプログラムの標準出力を決まったファイルに書き出したい場合には以下のように記述します。

```
./a.out <  data  > result　（入力ファイルがなければ ./a.out > result ）
```

このとき a.out の結果はファイル result に書き出されます。

9.4.　ジョブの実行（投入および確認）

ジョブの実行はスクリプトファイルをバッチシステムに投入（サブミット）することから始めます。コマンドは qsub を使用します。(以下の例で job.csh は既存のスクリプト

ファイル名とします。）

```
% qsub job.csh
Request 1425.n121 submitted to queue: A.
```

NQS はジョブを識別するリクエスト ID（リクエスト番号＋ジョブ投入ホスト名）を付加し，バッチシステムにジョブを送り込みます。ジョブの状態（待機中や実行中）は qstat コマンドで知ることができます。

```
% qstat

2009/03/10 (Tue) 16:06:55:    REQUESTS on SR11000
NQS schedule stop time : 2009/03/27 (Fri) 9:00:00 (Remain:404h 53m 5s)
   REQUEST   NAME     OWNER    QUEUE  PRI NICE     CPU    MEM   STATE
1425.n121    job.csh  p08000   A      63  0    unlimit 2048MB  QUEUED
```

リクエスト ID，スクリプト名，ログイン名が表示されている行が，先程投入したジョブの状況です。ジョブはバッチキューA で待機（QUEUED）していることがわかります。
ジョブが実行を開始すると qstat の結果は実行中（RUNNING）になります。

```
   REQUEST   NAME     OWNER    QUEUE  PRI NICE     CPU    MEM   STATE
1425.n121    job.csh  p08000   A      63  0    unlimit 2048MB  RUNNING
```

待機中，または実行中のジョブがないときは No requests. となります。なお，キューごとの実行ジョブ，待ちジョブ数は以下のように確認できます。

```
% qstat -b
2009/03/10 (Tue) 16:06:55:    REQUESTS on SR11000
NQS schedule stop time : 2009/03/27 (Fri) 9:00:00 (Remain:404h 53m 5s)
 QUEUE NAME STATUS   TOTAL   RUNNING  RUNLIMIT  QUEUED  HELD   IN-TRANSIT
 A          AVAILBL    1       1        6         0       0      0
  :                                     :
 P001       AVAILBL   10       6        6         4       0      0
  :                                     :
```

・計画停止時刻について

qstat コマンドの出力には以下のような日付，時刻と残り時間が表示されます。これはセンターがシステム（または NQS）を停止する予定時刻であり，計画停止時刻と呼びます。

```
NQS schedule stop time : 2009/03/27 (Fri) 9:00:00 (Remain: 404h 53m 5s)
```

この計画停止時刻が設定されているとき，この時刻を超える実行時間制限値（#@$-lT）を記述しているジョブは実行されませんので御注意下さい（実行時間制限値の記述がない場合には，single キューは 10 分，その他のキューはキューの最大実行時間を仮定します）。

・ジョブの待ち状態について

ジョブが待ち状態（QUEUED）となるのは通常のジョブ待ち，つまり，目的のキューで実行中のジョブ数（RUNNING）が実行可能な最大ジョブ数（RUNLIMIT）に達しているときですが，キューに空きがあるのに実行されないときは次のような理由が考えられます。

① 自分のジョブがそのキューで実行中である
② 自分のジョブ（２本）が他のキューで実行中である
③ 計画停止時刻までに終了しないジョブである
④ バッチシステムのノードに空きがない（またはその他の資源不足）
⑤ 障害，保守等の理由でキューが停止状態となっている
※RUNLIMIT は利用状況によって適宜変更します。

また,投入したジョブが他のジョブに追い越される場合は,年度当初から現在までの CPU 使用量や，投入したジョブに設定したジョブ経過時間制限値等を考慮したジョブスケジューリングにより，ジョブの実行優先度が低く設定されていることが考えられます。

☞ ジョブスケジューリングについては Web ページまたは「付録F ジョブスケジューリングシステムについて」を御覧下さい。

・バッチシステムの障害について

バッチシステムに障害が発生した場合，実行中のバッチジョブが異常終了することがあります。この場合，ジョブは障害回復後，キューに自動的に再投入される場合があります。なお，ジョブを再実行したくない場合には予めオプション #@$-nr を設定して下さい。

☞ 障害の内容によってはジョブが再投入されない場合があります。

9.5. ジョブの結果

ジョブが終了した場合，オプション（#@$-e，-o，-eo 等）で特に指定していなければ以下の 2 つのファイルが作成されます。プログラムやスクリプトファイルに出力ファイルを作成するように記述している場合にはそのファイルも作成されます。

標準出力： スクリプトファイル名.oN
標準エラー出力： スクリプトファイル名.eN

ここで，N はリクエスト番号で 1～5 桁，スクリプトファイル名は最初の 7 文字を使用します。 実際には以下のようなファイル名で作成されます。

標準出力： job.csh.o6128
標準エラー出力： job.csh.e6128

通常，標準出力には実行結果が，標準エラー出力にはエラーメッセージが出力されます。ただし，標準出力にエラーが出力されることもありますので両方のファイルを確認する必要があります。

☞ 標準エラー出力が空の場合はファイルが作成されません。
☞ 標準出力，標準エラー出力はプログラムやコマンドを端末で実行したとき，通常なら画面に表示される出力です。また，スクリプトファイルを使用せずに端末からジョブを直接入力（標準入力）した場

合には以下のファイルが作成されます。(N はリクエスト番号)

標準出力： STDIN.oN
標準エラー出力： STDIN.eN

9.6. ジョブのキャンセル

実行中または待機中のジョブをキャンセルしたい場合には qstat コマンドでリクエスト ID を確認して qdel コマンドでキャンセルします。

```
% qstat
   REQUEST   NAME     OWNER    QUEUE  PRI NICE     CPU    MEM   STATE
1425.n121    job.csh  p08000   A       63  0    unlimit 2048MB  RUNNING
```

例えば，上記のジョブ（REQUEST:1425.n121）をキャンセルする場合は次のように入力します。

```
% qdel  1425
```

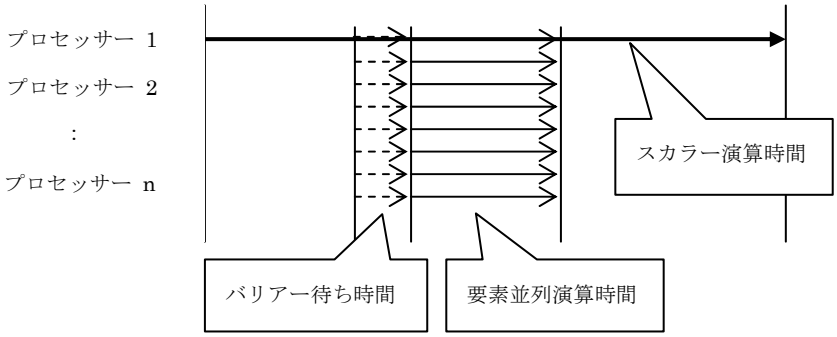
9.7. ジョブの開始，終了メール

ジョブが実行を開始,または終了したとき NQS はジョブの投入者(オプション#@$-mu でメールアドレスを指定しているときは送信先）にメールを送信します。ジョブが実行を開始すると開始時のメールにはバッチキュー名や開始時刻が出力されますが，ジョブが実行されなかった場合には実行に失敗した理由等のエラーメッセージが出力されます。また，終了時のメールには以下の情報が出力されます。

submit user	ジョブをサブミットしたユーザー名
request id	リクエスト ID（ジョブ ID)
request name	リクエスト名
submit host	ジョブをサブミットしたホスト名
execute host	サブミットしたジョブを実行したホスト名
submit queue	ジョブをサブミットしたキュー名
submit time	ジョブのサブミット時間
request start time	ジョブ開始時刻
request finish time	ジョブ終了時刻
request end status	ジョブの終了状態（返却コード+シグナル)
request cpu time (user + system)	ユーザー CPU 時間とシステム CPU 時間の合計
request exist time	ジョブ経過時間
number of nodes	ジョブが使用したノード数

要素並列プログラムの各時間には以下の関係があります。

要素並列プログラムの総演算時間 =（スカラー演算時間 －要素並列演算時間）+　要素並列演算時間×n



9.8. 実行例

Fortran プログラムをコンパイルした後，NQS を使用してバッチ処理したときの実行例は以下のようになります。

```
ディレクトリー "test" （/batch/p08000/test）での作業例です。
% cd test

ソースプログラムは以下の test.f を使用します。
% cat test.f
      parameter (n=10)
      dimension a(n,n),x(n),b(n)
      read(*,*)  ((a(j,i),j=1,n),i=1,n)
      read(*,*)  (x(j),j=1,n)
      do 10 i=1,n
       b(i)=0
 10   continue
      do 20 i=1,n
       do 30 j=1,n
        b(i)=b(i)+a(j,i)*x(j)
 30    continue
 20   continue
      do 40 i=1,n
       write(*,*) b(i)
 40   continue
      stop
      end

コンパイラーは f90 を使用，ロードモジュール名を program とします。
% f90 test.f -o program
f90: compile start : test.f

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*program units = 0001, no diagnostics generated.

エラーはなく，ロードモジュール program が作成されました。
% ls
data     go.csh   program  test.f

スクリプトファイルは go.csh を使用します。（go.csh と data は予め用意しています。）
% cat go.csh
#!/bin/csh
#@$-q single                  … スカラージョブなのでキューは single
#@$-lT 1:00:00                … 実行時間は 1 時間
```

```
#@$-lM 100MB                    … メモリー使用量は100MB
cd test                         … 作業ディレクトリーtestへ移動
./program < data                … プログラムの実行（ファイルdataは入力データー）
```

スクリプトファイル go.csh をサブミットします。

```
% qsub go.csh
Request 1434.n121 submitted to queue: A.
```

ジョブの実行状況は qstat コマンドで確認できます。

```
% qstat
2008/05/15 (Thu) 16:53:59:    REQUESTS on SR11000
NQS schedule stop time : 2008/05/30 (Fri)  9:00:00 (Remain: 352h  6m  1s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
105851.n121     go.csh    p08000     A         63  0   unlimit   100MB  RUNNING
```

実行が終了すると結果のファイル（この例では go.csh.o1434）が作成されます。標準エラー出力ファイルは空のため作成されませんでした。

```
% ls
data           go.csh         go.csh.o1434   program        test.f
% cat go.csh.o1434
  0.000000000E+00
   1.00000000
   2.00000000
   3.00000000
   4.00000000
   5.00000000
   6.00000000
   7.00000000
   8.00000000
   9.00000000
```

NQS からのメールも到着しています。（1 通目はジョブ開始時メール，2 通目がジョブ終了時メール）

```
% mail
Mail [5.2 UCB] [AIX 5.X]  Type ? for help.
"/var/spool/mail/p08000": 2 messages 2 new
>N  1 loadl@batch.cc.u  Thu May 15 16:54  26/1180 "NQS Initiator Report: 105851"
 N  2 loadl@batch.cc.u  Thu May 15 16:54  29/1375 "NQS Terminator Report: 10585"
? 2
Message  2:
From loadl@batch.cc.u-tokyo.ac.jp Thu May 15 16:54:45 2008
Date: Thu, 15 May 2008 16:54:45 +0900
From: loadl@batch.cc.u-tokyo.ac.jp
To: p08000@sr11000-s.cc.u-tokyo.ac.jp
Subject: NQS Terminator Report: 105851.n121

submit user                          = p08000
account number                       = (default)
request id                           = 105851.n121
request name                         = go.csh
submit host                          = n121
execute host                         = SR11000
submit queue                         = A
submit time                          = 2008/05/15 (Thu) 16:53:53
request start time                   = 2008/05/15 (Thu) 16:54:39
request finish time                  = 2008/05/15 (Thu) 16:54:44
request end status                   = Completed
request cpu time (user + system)     = 0.316616 (0.043884 + 0.272732) sec
request exist time                   = 5 (sec)
number of nodes                      = 1

? q
 (省略)
%
```

☞ 上記の例ではプログラム（test.f），スクリプトファイル（go.csh）のほか，入力データー(data)が必要です。エディターなどで予め作成して下さい。

```
% vi data
1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 1 2 3 4 5 6 7 8 9
```

10. Fortran のコンパイルと実行の詳細

10.1. コンパイルとリンク

Fortran プログラムのコンパイルには f90（最適化 FORTRAN90）または f77（最適化 FORTRAN77）コマンドを使用します。また，リンクにも f90, f77 コマンドを使用します。

```
f90（または f77）
    ファイル名..
    [-c]
    [-o ロードモジュールファイル名]
    [-コンパイルオプション..]
    [-リンケージオプション..]
    [-オプション..]
    [-I インクルードディレクトリー名]
    [-L ライブラリーディレクトリー名]
    [-l ライブラリー名]
    [-i,言語仕様拡張オプション]
```

① オプション

オプションは空白で区切り複数指定できます。左から右へ順に処理するので同一または背反するオプションは後ろに指定したものが有効になります。特にライブラリー名を指定する–l オプションはオプションの順序がコンパイル，リンクに影響するので注意が必要です。

☞ オプションの指定方法には –W0,'コンパイルオプション', –Wl,'リンケージオプション'と記述する方法がありますが，ここでは －コンパイルオプション，－リンケージオプションの形式を使用します。

なお，本センターでは 64 ビットアドレッシングモード（10.2.④を参照）がデフォルトです（2007 年 4 月から）。また，次のオプションをデフォルトとして設定しています。オプション無指定時にも以下のオプションが仮定されますので御注意下さい（デフォルトオプションは変更となる場合があります）。

f90 および f77: –e　　☞ 10.2.⑫を参照

上記に加えて利用者がデフォルトオプションを設定する場合は，環境変数 F90USEROPTS（または F77USEROPTS）に以下のように設定して下さい。

```
% setenv  F90USEROPTS  '-Oss  -parallel=4'    デフォルトオプションの設定例
% setenv  F90USEROPTS  '-Oss  -noparallel'
```

② ファイル名

以下のサフィックスを持つファイル名を指定するとファイルが Fortran ソースプログラムファイルであるとして Fortran コンパイラーを起動します。

.f （固定形式）　.f77　.f90　.f95　（各言語仕様の自由形式）

以下は C 言語プリプロセッサーを通した後，Fortran コンパイラーに入力します。

.F（固定形式） .F77 .F90 .F95 （各言語仕様の自由形式）

ただし，オプションにより固定形式，自由形式の指定（–fixed，–free），または言語仕様の切り替え（–hf77，–hf90，–hf95），C 言語プリプロセッサー（–cpp）を指定した場合にはオプションが優先されます。サフィックスが .c または .i の場合は C 言語ソースプログラムであるとして C コンパイラーを起動します。また上記以外，オブジェクトファイル .o 等のファイルが指定された場合にはリンケージエディターを起動します。

コンパイル，リンクの例

```
% f90  main.f  sub.f                          main.f, sub.f をコンパイルして実行ファイル
main.f:                                       を作成
f90: compile start : main.f

*OFORT90 V01-05-/C entered.
*program name = MAIN                          (main.f のコンパイルメッセージ)
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.
sub.f:
f90: compile start : sub.f

*OFORT90 V01-05-/C entered.
*program name = SUB                           (sub.f のコンパイルメッセージ)
*end of compilation : SUB
*program  units  =  0001,  no  diagnostics
generated.                                    (この後リンクが行われる)
% ls
a.out   main.f   sub.f                        実行ファイル a.out ができる
%
```

オブジェクトモジュール作成後，リンクする例

```
% f90  -c  main.f  sub.f                      main.f, sub.f をコンパイルしてオブジェク
 (省略)                                       トモジュールを作成
% ls                                          オブジェクトモジュール main.o, sub.o ができる
main.f  main.o  sub.f  sub.o
%
% f90  -o  program  main.o  sub.o             main.o, sub.o をリンクして実行ファイルを
% ls                                          作成
main.f  main.o  program  sub.f  sub.o         実行ファイル program ができる
%
```

10.2. オプションと機能

最適化 FORTRAN90（または最適化 FORTRAN77）で使用される主なオプションとその機能について説明します。代表的な使用例は次のとおりです。

```
% f90  -Oss  program.f                         自動並列化を行うコンパイル
% f90  -Oss  -omp  program.f                   OpenMP 使用時のコンパイル
% f90  -Oss  -noparallel  program.f            スカラープログラムのコンパイル
```

① 最適化オプション

最適化とは演算子の変更，ループ構造の変換，演算順序の変更などをコンパイラーが自動的に行い，プログラムの実行速度を向上させる機能です。各種オプションが最適化機能毎に用意されていますが，以下に示すレベルに従って最適化することができます。

オプション無指定時はレベル 3（-03）でコンパイルされます。

なお，最適化機能には副作用を含むものがあり，最適化によっては計算結果が異なったり，エラーが生じる場合がありますので十分理解した上でオプションを使用して下さい。

–O** 最適化レベル* ***最適化オプション

```
% f90  -O4   program.f      最適化オプション-O4 でコンパイル
% f90  -Oss  program.f      最適化オプション-Oss でコンパイル
% f90        program.f      最適化オプション-O3 でコンパイル
```

最適化オプション

-O0 (レベル 0)	原始プログラムどおりのコンパイル，レジスタの効果的な使用方法を中心に文の実行順序を変更しない一文単位の最適化（べき演算の乗算化，文関数のインライン化，局所的なレジスタの割り当て等）
-O3 (レベル 3) 【デフォルト】	プログラム全体での大域的な最適化（分岐命令の最適化，命令の並べ替え，演算子の変更，不変式のループ外への移動，外部手続きインライン展開等）
-O4 (レベル 4)	制御構造の変換，演算順序の変更を含むプログラム全体の最適化（演算順序の変更，除算の乗算化，ループ展開・交換・分配・融合等）
-Os	実行速度が速くなるようなコンパイルオプション※の自動設定（ソフトウェアパイプライン，要素並列化を含む各種最適化オプションの設定） ※ -nolcheck, -nodochk, -approx, -noconvcheck, -disbracket, -divmove, -expmove, -fma, -invariant_if, -ischedule=3, -loopdistribute, -loopexpand, -loopfuse, -loopinterchange, -loopreroll, -O4, -parallel=2, -prefetch, -prod, -rapidcall, -scope, -swpl, -noargchk, -nobreak, -noerstmt, -noagochk, -nosubchk, -workarray
-Oss	-Os に加えてさらに実行速度が速くなるようなコンパイルオプション※の自動設定（-Os に加えてライブラリーコード使用，ベクトル数学関数の引用等） ※ -approxlib, -ifswpl, -parallel=4, -pvfunc=2

高いレベルの最適化は，それより低いレベルの機能を含んでいます。また，一般にレベルが高い最適化ほどコンパイルに時間がかかります。

–loopdiag ***最適化ループ診断メッセージ***

```
% f90 -Os -loopdiag program.f          program.f をコンパイルしたときのループ
f90: compile start : program.f         構造診断メッセージを出力する

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
   KCHF1805K
     the do 10 loop is fused with the following    (DO10 ループと後続のループを融合)
one. line=11
   KCHF1809K
     the do 10 loop is unrolled 4 times.    (DO10 ループを 4 回展開)
line=11
*program units = 0001, no diagnostics
generated.
```

オプションの詳細および個別の最適化オプションにつきましてはマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22）」または「最適化 FORTRAN77 使用の手引（3000-3-C24）」を御覧下さい。

最適化オプションと仮定されるオプションの関係を次に示します。

最適化 オプション	仮定される オプション	内容
-O3 （レベル 3） 【デフォルト】	-convcheck	データー型変換関数（aint,anint,nint）の引数範囲をチェックする
	-expmove	条件式下にあるループ不変式をループ外に移動する
	-ischedule=1	命令の実行順序を変更する
	-nomathinline	数学関数のインライン化をしない
	-scope	ループの最適化でスコープ分割をする
-O4 （レベル 4）	-O3	最適化レベル 3
	-arraycomm=1	演算操作が増加しない範囲で配列要素を共通化する
	-fma	浮動小数点乗加減算命令を使用する最適化をする
	-swpl	ソフトウェアパイプラインによる最適化をする
-Os	-O4	最適化レベル 4
	-nolcheck	新 Fortran 規格（JIS X 3001-1:1994 または JIS X 3001-1:1998）から拡張した仕様に対して，エラーメッセージを出力しない
	-nodochk	DO 文の繰り返し回数 0 をチェックしない
	-approx	除算で，逆数による乗数化を許可する
	-noconvcheck	データー型変換関数（aint,anint,nint）の引数範囲をチェックしない（注：-03,-04 と異なる）
	-disbracket	括弧や文の順序によって明示された演算順序を変更する
	-divmove	条件式下で除算割り込みが起こる可能性があるループ不変式をループ外に移動する
	-invariant_if	ループ不変条件展開最適化をする
	-ischedule=3	-ischedule=1 に加え，より大域的に命令の実行順序を変更する。また，分岐予測を行い投機的に実行順序を変更する。（注：-03,-04 と異なる）
	-loopdistribute	ループ分配による最適化をする
	-loopexpand	ループ展開による最適化をする
	-loopfuse	ループ融合による最適化をする
	-loopinterchange	ループ交換による最適化をする
	-loopreroll	ループ巻き戻しによる最適化をする
	-parallel=2	※ 次頁を参照
	-prefetch	最内側ループ中の配列に対してプリフェッチ最適化をする
	-prod	乗算の代わりに組込み関数 DPROD，QPROD を引用する
	-rapidcall	組込み関数に対して呼び出し時の引数チェックを行わない，引数を値渡しとする
	-noargchk	サブルーチンおよび関数を引用する際，引数の個数と型，属性，名称の重なりをチェックしない
	-nobreak	プログラム実行中の端末の割り込みシグナル（SIGINT）およびタイマ割り込み（SIGALRM）を受け付けない
	-noerstmt	トレースバックマップ中にエラーが発生した文の行番号を出力しない
	-noagochk	プログラム実行時に，割当て形 GO TO 文の文番号並びの有無をチェックしない
	-nosubchk	配列参照の添字の値が，宣言の範囲内にあるかどうかをチェックしない
	-workarray	作業配列を利用した最適化をしない
-Oss	-Os	最適化レベル S
	-approxlib	除算，逆数演算，SQRT 組込み関数呼び出しを実行時ライブラリーの呼び出しで計算する
	-ifswpl	条件分岐を含むループのソフトウェアパイプラインによる最適化をする
	-parallel=4	※ 次頁を参照
	-pvfunc=2	ベクトル数学関数を引用する（一時配列の導入含む）

演算順序が変更となる最適化（–approx，–disbracket 等）は浮動小数点演算での精度誤差が発生する場合があります。また，–expmove，–divmove は例外の発生する可能性があります。これらを回避するためには –no*option* を指定して原因となる最適化機能を抑止して下さい（例：–Os –noapprox)。反対に –no*option* を有効にするためには –*option* を指定して下さい（例：–Os –argchk)。その他の最適化においても副作用を含む場合がありますので御注意下さい。

☞ 最適化の副作用について詳細はマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」または「最適化 FORTRAN77 使用の手引（3000-3-C24)」を参照して下さい。

② 要素並列化オプション

要素並列化とはプログラムを並列処理単位に分割して，ノード内の複数のCPUで並列実行するためのオブジェクトを生成する機能です。メッセージ通信を行う並列化とは異なり，コンパイルオプションや指示文により利用者が並列処理自体をコーディングすることなく，プログラムを並列化することができます（「6.1. 要素並列処理」参照）。

要素並列化を行うためには以下のオプションを指定してコンパイルします。また，オブジェクトモジュールをリンクする際にもリンケージオプション（–parallel）の指定が必要です。

–parallel=要素並列化レベル	*要素並列化オプション*
% f90 -parallel program.f	要素並列化（レベル２）する
% f90 -parallel=4 program.f	要素並列化（レベル４）する
% f90 -noparallel program.f	要素並列化を抑止
% f90 -parallel main.o sub.o	要素並列オブジェクトをリンクする場合

–parallel オプションにレベル指定がない場合は –parallel=2（レベル2）を仮定します。
–parallel リンケージオプションにはレベル指定は不要です。

要素並列化オプションの詳細は次のとおりです。

-noparallel -parallel=0	要素並列化をしない
-parallel=1	SECTION 型要素並列化，強制ループ要素並列化
-parallel -parallel=2	変数・配列のプライベート化，リダクション変数要素並列化
-parallel=3	ループ分配，ループ分割，ループの一重化，サイクリック分割
-parallel=4	パイプライン要素並列化，インダクション要素並列化

高いレベルの要素並列化は，それより低いレベルの機能を含んでいます。

要素並列化変換にはオプション指示でDOループを自動的に変換するループ要素並列化とパラメーター指示（POPTION）で文の集まりをCPUに分配するSECTION型要素並列化があります。ループ要素並列化が適用される条件は以下のとおりです。

- SECTION型要素並列化指示された範囲内にあるDOループでない
- DOループに以下の文を含まない
  - ループ脱出文（GOTO文，EXIT文など）
  - 割り当てGOTO文
  - 関数，手続き呼び出し
  - 入出力文
  - 終了・停止文（STOP文，PAUSE文，RETURN文など）
- DOループにNOPARALLEL指示がない
- DOループ内に同期制御または排他制御指示がない
- DOループの実行性能向上が見込める
- DOループ内に現れる変数または配列に，ループ繰り返しにわたる依存関係がない。

–pardiag ***要素並列化診断メッセージ***

```
% f90 -parallel -pardiag program.f          program.f　を要素並列化してコンパイルし
f90: compile start : program.f             たときの診断メッセージを出力する

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
(diagnosis for loop structure)
   KCHF2000K
     the do 10 loop is parallelized.line=7     (DO10 ループを要素並列化)
   KCHF2011K
     the variable(s) or array(s) in do 10 loop (DO10 ループ内の変数を TLOCAL 化)
is applied to tlocal transformation.name=I
line=7
   KCHF2013K
     the final value is guaranteed for the     (DO10 ループ内の TLOCAL 化する変数の終
variable(s) or array(s) in do 10 loop subject   値保証を行う)
to tlocal transformation.name=I line=7
*program units = 0001, no diagnostics
generated.
%
```

オプションの詳細および個別の要素並列化オプションにつきましてはマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」または「最適化 FORTRAN77 使用の手引（3000-3-C24)」を御覧下さい。

要素並列化されたプログラムは，ノード内の複数の CPU（16 または 8CPU）を使用して実行します。このため,本センターではノードを専有できる並列ジョブクラス(P001～P064, S1, S4, H1, debug 等）でプログラムを実行する必要があります。

③ OpenMP

最適化 FORTRAN90 では OpenMP（OpenMP Fortran Application Program Interface Version 2.0, November 2000 ）を用いて要素並列化を行うことができます。OpenMP とは共有メモリー型の並列計算機における並列化の指示文やライブラリー，環境変数等を規定した規格です。

–omp *OpenMP オプション*

```
% cat omp.f
      real a(1000)
!$omp parallel                          OMP 指示文の例
!$omp do
      do i=1,1000
       a(i)=i
      enddo
!$omp enddo
!$omp end parallel
      end
% f90 -parallel -omp omp.f              OpenMP を使用したプログラムのコンパイル
                                        -parallel オプションの指定が必要
```

OMP 指示文や OpenMP オプションにつきましてはマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」を御覧下さい。

OpenMP は最適化 FORTRAN77 ではコンパイルできません。–o mp オプションとして処理するため，エラーにはならず，mp という名の実行ファイルが生成されてしまいます。

④ 64 ビットアドレッシングモード

プログラムで使用するメモリーサイズが 2GB 以上の場合には，64 ビットアドレッシングモードのオブジェクトモジュールの作成が必要です。SR11000 では環境変数 OBJECT_MODE に 64 をデフォルトで設定しているため，64 ビットアドレッシングモードがデフォルトとなります（2007 年 4 月から)。

32 ビットアドレッシングモードで作成する場合は setenv OBJECT_MODE 32 または unsetenv OBJECT_MODE とするか –32 オプションを指定してコンパイルして下さい。なお，オブジェクトモジュールをリンクする際にも，この指定が必要です。

☞ Web ページまたはスーパーコンピューティングニュース Vol.9 No.2, 2007.3「64 ビットモード標準化における注意点」を併せて御覧下さい。

-32 / -64 *32 / 64 ビットアドレッシングモードオプション*

```
% f90  -32  program.f          32 ビットモードでコンパイルする
% f90  -32  main.o  sub.o      32 ビットモードのオブジェクトをリンクする
% f90  -64  program.f          64 ビットモードでコンパイルする
% f90        program.f         環境変数 OBJECT_MODE の値による
                                （デフォルトは 64 ビットモード）
```

以下は 64 ビットアドレッシングモードにおける注意事項です。

- 名前付き定数および初期化項目には，2,147,483,647 バイトを超えるデーターは使用できない。
- ASSIGN 文，割り当て型 GOTO 文，および FMT 指定子の変数は，整数型 8 バイトでなければならない。
  - 型変更は利用者自身で行うか，オプション–intexp=full で全ての整数型の変数，配列，定数を整数型 8 バイトへ拡張（引数の整合性には注意）する必要がある。
- サービスサブルーチンの引数には，整数型 8 バイトは使用できない。
  - サービスサブルーチンを使用している場合には（引数を）整数型 8 バイトを整数型 4 バイトに変換するオプション–exsrvc を指定する必要がある。
- オプション–hugeary が仮定され，以下の組込み関数は整数型 8 バイトを返す。(f90 のみ)
  - LBOUND，SHAPE，SIZE，UBOUND，COUNT，MAXLOC，MINLOC

⑤ ログメッセージ

ログメッセージ出力オプションを指定することで，コンパイル診断メッセージをファイルに出力することができます。ログメッセージファイルはソースプログラム毎に「ソースプログラム名.log 」というファイル名で出力されます。要素並列化等を適用した様子がソースプログラムに併記されるのでプログラムのチューニングが容易になります。

–loglist *ログメッセージ出力オプション*

```
% f90 -Oss -loglist program.f                    ログメッセージファイルを出力する
f90: compile start : program.f

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*program units = 0001, no diagnostics
generated.

% ls                                              program.log が作成される
a.out        program.f    program.log

% cat program.log                                ログメッセージファイル program.log の内
      program main                                容を表示
      parameter (n=10000)
      real a(n), b(n), c(n)

**
** Parallel processing starting at loop entry   (並列処理開始)
** Parallel function: _parallel_func_1_MAIN
** Parallel loop                                (並列ループ)
** --- 2 loops (DO10,DO20) fused (depth 1) ---  (DO10, DO20 ループ融合)
** Parallel processing finishing at loop exit
**
**    [DO 10]
**    Innermost loop unrolled (10 times).       (最内側ループ展開)
**    SWPL applied.                             (ソフトウェアパイプライン適用)
**
 (省略)
```

ログメッセージは日本語で出力することができます。このときコンパイルメッセージも日本語になります（デフォルトは環境変数 LANG に設定された文字コード種別であり，LANG=C [英語]です)。

–listlang=文字コード種別

```
% f90 -Oss -loglist program.f -listlang=c      英語（ASCII）で出力
% f90 -Oss -loglist program.f -listlang=sjis   日本語（SJIS）で出力
% f90 -Oss -loglist program.f -listlang=euc    日本語（EUC）で出力
```

```
% f90 -Os -loglist program.f -listlang=sjis    ログメッセージファイルを出力する
f90: compile start : program.f                （オプション –listlang=sjis　を指定）

*OFORT90 V01-05-/C 開始                       コンパイルメッセージが日本語で表示され
*プログラム名 = MAIN                          る（端末の文字コードを SJIS に設定するこ
*end of compilation : MAIN                    と）
*end of compilation : _parallel_func_1_MAIN
*プログラム数 = 0001 , エラーはありません。

% cat program.log                              ログメッセージファイル program.log が
      program main                            日本語で作成される
      parameter (n=10000)
```

```
      real a(n), b(n), c(n)

**
**  ループ入口で並列処理 開始
**  並列手続き名 : _parallel_func_1_MAIN
**  並列ループ
**  --- 2 個のループ(D010,D020)にループ融合(1
重)を行った ---
**  ループ出口で並列処理 終了
**
**    [DO 10]
**    最内側ループ展開(10 倍)を行った。
**    SWPL を適用した。
**
 (省略)
```

⑥ 性能モニター

性能モニター情報出力オプションを指定することで実行時に性能，負荷等の情報を得ることができます。本オプションはリンク時にも指定が必要です。本オプションを指定してコンパイルしたプログラムを実行すると性能モニター情報ファイルが生成されます（”pm_実行ファイル名_日付_時間_ノード番号_プロセス番号”というファイル名で出力されます)。このファイルはバイナリー形式のため，**pmpr** コマンドを使用して参照します。日本語による出力も可能です（デフォルトは環境変数 LANG に設定された文字コード種別であり，LANG=C [英語]です)。

なお，性能モニターはサブルーチンまたは関数の呼び出し回数が多い場合，実行時間に影響しますので御注意下さい。

–pmfunc　性能モニター情報（プロセス単位，関数／手続き単位）出力オプション

```
% f90 -parallel program.f -pmfunc          性能モニター情報を出力する実行ファイル
f90: compile start : program.f            を作成

*OFORT90 V01-05-/C entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*end of compilation : SUB
*end of compilation : _parallel_func_2_SUB      このファイルはバイナリー形式のため，
*program units = 0002, no diagnostics          性能モニター情報出力コマンド pmpr
generated.                                     を使用してテキスト形式で表示する。
%
 (バッチジョブで a.out を実行)                 実行ファイル a.out を実行する
% ls
…
a.out
pm_a.out_Oct24_1139_n75_696542                 性能モニター情報ファイルが作成されてい
program.f                                      ることを確認

% pmpr pm_a.out_Nov02_1154_n12_659518          pmpr コマンドで情報をテキスト形式で表示
pm_a.out_Nov02_1154_n12_659518:                （pmpr –j ファイル名とすると日本語 SJIS
 (省略)                                        で表示される)
```

```
############################################################
## Function/Procedure                                     ##
############################################################
========================================
== Function/Procedure Ranking         ==
========================================
       CPU time[%]        Times  Func(File+Line)
------------------------------------------------
[  1]  0.000065[ 88.24]       1  MAIN(program.f+5)
[  2]  0.000009[ 11.76]       1  SUB(program.f+20)
----------------------
TOTAL  0.000074[100.00]
```

性能モニターの使用法についてはスーパーコンピューティングニュース Vol.7 No.3, 2005.5「ベクトル並列型スーパーコンピューターSR11000 チューニングガイド」を御覧下さい。

要素並列化の単位で性能モニター情報を出力することもできます。同様にコンパイル，実行して以下のように参照します。

–pmpar　　性能モニター情報（プロセス単位，要素並列化単位）出力オプション

```
% f90 -parallel program.f -pmpar                性能モニター情報を出力する実行ファイル
f90: compile start : program.f                  を作成

*OFORT90 V01-05-/C entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*end of compilation : SUB
*end of compilation : _parallel_func_2_SUB      このファイルはバイナリー形式のため，
*program  units  =  0002,  no  diagnostics     性能モニター情報出力コマンド pmpr
generated.                                      を使用してテキスト形式で表示する。
%
（バッチジョブで a.out を実行）                 実行ファイル a.out を実行する
% ls
…
a.out
pm_a.out_Nov02_1159_n12_643084                  性能モニター情報ファイルが作成されてい
program.f                                       ることを確認

% pmpr pm_a.out_Nov02_1159_n12_643084           pmpr コマンドで情報をテキスト形式で表示
pm_a.out_Nov02_1159_n12_643084:                 （pmpr –j ファイル名とすると日本語 SJIS
（省略）                                        で表示される）
############################################################
## Element Parallel Region                                ##
############################################################
========================================
== Element Parallel Region Ranking    ==
========================================
       CPU time[%]         MFLOPS      MIPS    Times  Func[Rank](File+Line)
-------------------------------------------------------------------------------
[  1]  0.000005[ 52.76]  2097.188  10865.438        1  SUB[-](program.f+20)
[  2]  0.000004[ 47.24]  2342.337  12134.609        1  MAIN[-](program.f+8)
----------------------
TOTAL  0.000009[100.00]
（省略）
```

性能モニターの使用法についてはスーパーコンピューティングニュース Vol.7 No.3, 2005.5 または別冊「ベクトル並列型スーパーコンピューターSR11000 チューニングガイド」を御覧下さい。

pmpr コマンドによる文字コード種別の指定例

```
% pmpr  -e    性能モニター情報ファイル名      英語（ASCII）で出力
% pmpr  -j    性能モニター情報ファイル名      日本語（SJIS）で出力
% pmpr  -euc  性能モニター情報ファイル名      日本語（EUC）で出力
```

⑦ ソースプログラム差分コンパイル

ソースプログラム差分コンパイル機能はオブジェクトファイル中にソースプログラム情報を保持し，再度そのオブジェクトを出力先としてコンパイルしたときには，ソースプログラムの変更があったプログラム単位のみコンパイルする機能です。これによりコンパイル時間を短縮することができます。

–diffcomp ***ソースプログラム差分コンパイルオプション***

```
% f90  -diffcomp  program.f                  差分コンパイルオプションを指定する
f90: compile start : program.f              通常と同様にコンパイルを開始

*OFORT90 V01-05-/C entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN                  プログラム MAIN コンパイル終了
*end of compilation : SUB                   プログラム SUB コンパイル終了
*program  units  =  0002,  no  diagnostics
generated.
% ls
a.out      program.f    program.o           オブジェクトファイル program.o および実
                                            行ファイル  a.out が作成される
～ プログラム SUB を編集 ～

% f90  -diffcomp  program.f                  プログラム SUB の内容を変更後，改めて差
f90: compile start : program.f              分コンパイルする

*OFORT90 V01-05-/C entered.                  プログラム MAIN に変更がないので構文
*program name = MAIN                        チェックのみ行う
*program name = SUB                         プログラム SUB は変更があったのでコンパ
*end of compilation : SUB                   イルを行う
*program  units  =  0002,  no  diagnostics  メッセージ「*end of compilation : MAIN」が
generated.                                  ないことに注意
%
```

⑧ オブジェクト再コンパイル

オブジェクト再コンパイル機能はオブジェクトファイル中に保持しているソースプログラム情報を使用し，オブジェクトを入力として再コンパイルする機能です。

–recomp ***オブジェクト再コンパイルオプション***

```
% f90  -diffcomp  program.f                  差分コンパイルオプションを指定する
f90: compile start : program.f              通常と同様にコンパイルを開始

*OFORT90 V01-05-/C entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN                  プログラム MAIN コンパイル終了
*end of compilation : SUB                   プログラム SUB コンパイル終了
*program  units  =  0002,  no  diagnostics
generated.                                  オブジェクトファイル program.o および実
                                            行ファイル a.out が作成される
% mv  program.f  program.f.bak              試しにソースプログラム program.f の名前
% ls                                        を変更してみる
a.out          program.f.bak  program.o

% f90  -recomp  -parallel  program.f        再コンパイルオプションを指定してコンパ
f90: compile start : program.f              イル（参考のため–parallel オプション追加）

*OFORT90 V01-05-/C entered.                  オブジェクトモジュール内のソースプログ
*program name = MAIN                        ラム情報を利用して再コンパイルを行う
```

```
*program name = SUB
*end of compilation : MAIN                    プログラム MAIN コンパイル終了
*end of compilation : SUB                     プログラム SUB コンパイル終了
*program  units  =  0002,  no  diagnostics
generated.
%
```

⑨ 制限コンパイル

通常コンパイラーは最大実行性能の実行ファイルを作成しようとするために，コンパイル時間やコンパイル時使用メモリー量は無制限に使用して最適化を行いますが，時間がかかり過ぎる場合やコンパイル時のメモリー不足の場合に，これらに制限を設けてコンパイルすることができます。

–limit ***制限コンパイルオプション***

```
% f90  -limit    program.f                    制限コンパイルを行う
% f90  -nolimit  program.f                    制限コンパイルを行わない
```

オプション無指定の場合 –limit がデフォルト。

⑩ デバッグオプション

デバッグ情報を表示します。

–debug ***デバッグオプション***

```
% f90  -debug    program.f                    デバッグ情報を表示する
% f90  -argchk=all  program.f                 引数の個数，型をチェックする
```

–debug

以下に示すオプションを設定し，デバッグ情報を出力します。

–lcheck –dochk –argchk=all –dline –erstmt –agochk –subchk

–lcheck

新 Fortran 規格（JIS X 3001-1:1994，または JIS X 3001-1:1998）から拡張した仕様に対して，エラーメッセージを出力します。

–dochk

DO 文の繰り返し回数が 0 の場合，エラーメッセージを出力します。

–argchk=all

関数，サブルーチンの引数の個数，型，属性の不一致をチェックし，実行時にエラーメッセージを出力します。また，引数並びに同一名称が指定されている場合にコンパイル時にメッセージを出力します。

–subchk

配列参照の添字の値が宣言の範囲内にない場合，実行時にエラーメッセージを出力します。

☞ 関数の引数や配列参照に問題があるとプログラムは正常に動作しないため，デバッグオプションを指定していてもエラーにならないことがあります。（結果不正の可能性があります。）また，以下のエラーメッセージを出力し，プログラムが異常終了することがあります。

```
KCHF446R  segmentation violation occurred.
```

⑪ トレースバックマップ

エラーを検出したプログラムのトレース情報を実行時にエラー発生場所［ファイル名：行番号］の形式で出力します。これにより，エラー発生箇所を特定できる場合があります。

–s,TRACE ***トレース情報オプション***

% f90 -s,TRACE program.f	トレースバックマップ出力を行う

実行時にエラーメッセージと共に出力されます。（例 0x100003b4　MAIN+0x34　[a.f:4]）

⑫ 言語仕様の拡張

他の Fortran 処理系でコンパイルできるプログラムが本センターの SR11000 でコンパイルエラーとなるとき，以下のオプション指定により対応できる場合があります。なお，オプションの詳細についてはマニュアルを参照して下さい。

–e

以下に示すオプションを設定し，コンパイルの制限を緩和，および実行モードの切り替えによる言語仕様の拡張を行います。SR11000 ではこのオプションをデフォルトで設定しています。

コンパイルオプション（f90）

–i,P –i,PL –conti199 –h8000 –intptr –commentinclude=asterisk –excmplx

コンパイルオプション（f77）

–i,U –i,P –i,PL –conti199 –h8000 –intptr –commentinclude=asterisk –typeparam

実行時オプション（f90 および f77 共通。実行時に自動的に設定される。）

–F'PORT(ECONV,EOFBACK(0),EOFRD,EOFRDT,GETARG,GETENV,IARGC,
MSGOUT(STDERR),NMLIST,NSCRACH,PRCNTL,REALEDT,REWNOCL,
STDUNIT,TABSP),RUNST(DAMNONL,UMASK)'

–i,U

外部手続き名称中の_（アンダースコア）および 31 文字までの外部手続き名称を使用できます（f77 のみ。f90 ではこの制限はありません）。

–i,P

以下の項目について言語仕様を拡張します。

- デバッグ行（D，？），タブコードの扱い
- $ 型，NL 型，O 型，Q 型編集
- 組込み関数（%VAL，%REF）
- 実定数が上限，下限値を超える場合の仮定処理
- DO WHILE 文
- DOUBLE COMPLEX 型　等

–i,PL

16 進定数（X'xx'または'xx'X）が使用できます。

–i,EU

外部手続き名（モジュール名を除く）の末尾にアンダースコアを付加します。

–i,LT

"（ダブルクォーテーション）で囲まれた文字列を定数とみなします。指定がないとき" 以降，行の最後までを注釈とみなします（f77 のみ。f90 では定数とみなします）。

–h8000

OS7 Fortran 言語仕様に合わせてコンパイルします。マスキング式，マスキング代入文，BOOL 組込み関数，論理定数を使用できます。

–cpp

C 言語プリプロセッサー呼び出しを行います。

10.3. 実行時オプション

実行時オプションは実行時環境の変更や他社 Fortran との互換性に対応するためのオプションです。ロードモジュール（実行ファイル）を実行するときに，以下のように「-F」に続けてアポストロフィーで囲んで指定します。カンマで区切ることで複数のオプションを指定することもできます。

ロードモジュール名　-F'実行時オプション,実行時オプション,…'

実行時オプションはサブオプションを持っており，括弧で囲んで指定します。カンマで区切ることで複数のサブオプションを指定することができます。例えばロードモジュール名が a.out の場合，以下のように記述します。（実行時オプションは大文字でも，小文字でも構いません）

```
%  ./a.out -F'PORT(STDUF),RUNST(UFLOW)'
```

主な実行時オプションを紹介します。

RUNST(UFLOW)

標準では浮動小数点演算中にアンダーフローが発生した場合でも処理を継続しますが，本オプション指定時にはエラーメッセージを出力してプログラムを終了します。

RUNST(OFLOW)

標準では浮動小数点演算中にオーバーフローが発生した場合でも処理を継続しますが，本オプション指定時にはエラーメッセージを出力してプログラムを終了します。

☞ RUNST(fpecntl(0)) を指定すると，0 を 0 で割る除算例外の場合でもプログラムを終了します。

PORT(ECONV)

E 形，ES 形，EN 形，または G 形編集記述子で指数部を表示する場合，倍精度の実数，拡張精度の実数，または複素数の指数表示文字を「E」とします。このオプションがない場合は倍精度が「D」，拡張精度が「Q」の指数表示文字となります。（デフォルト）

PORT(REALEDT)

実数型データー0.0 を E 形，D 形，Q 形，または G 形編集記述子で出力するときに指数部を付加します（例 0.00E+00）。また，小数点以下の数値列が書式の長さに満たない場合，書式の長さに合わせて 0 を挿入します（例 0.200000000　E+01 は 0.200000000000E+00 となる）。（デフォルト）

PORT(TABSP)

入力データーに TAB コードが含まれる場合，空白として扱います。(デフォルト)

PORT(PRCNTL)

画面，またはプリンタに書式付き記録を出力する場合，記録の先頭の一文字目を印刷制御文字として扱いません。(デフォルト)

PORT(DSTDUF)

書式なし入出力に対応するファイル形式を業界標準書式なしファイルとします。このファイル形式は他社 Fortran との互換形式として使用します。(デフォルト)

PORT(STDUF)

書式なし入出力文に対応するファイル形式を標準書式なしファイルとします。このオプションは BACKSPACE 文の動作を除けば PORT(DSTDUF)オプションと同じです。

PORT({ENDIANIN | ENDIANOUT | ENDIANINOUT}({ALL|NONE|n[,n]…})

リトルエンディアン形式で入力（ENDIANIN)，出力（ENDIANOUT）または入出力（ENDIANINOUT）するかどうかを指定します。すべての装置番号（ALL）または個別の装置番号（n）を指定できます。

10.4. サービスサブルーチン

JIS Fortran の組込み関数以外にサービスサブルーチンと呼ばれる最適化 FORTRAN90 または最適化 FORTRAN77 が用意しているサブルーチンがあります。そのなかでよく使われる CPU 時間や経過時間を計測するサービスサブルーチンと他社 Fortran で標準的にサポートされているサービスサブルーチンを使用する方法を以下に紹介します。

① プログラムの時間計測

プログラムのある部分の実行に要した時間を計測する場合には以下の clock または xclock サブルーチンを使用します。なお，本サブルーチンの使用についてはコンパイル，リンク時の特別な指定は必要ありません。

```
CPU 時間計測の例
    real t                                  変数 t は実数型 4 バイト（単精度）
    call clock                              CPU 時間測定タイマー起動
   -- プログラム --
    call clock(t)                           タイマー起動後の CPU 時間（秒）を変数 t
    write(*,*) t                            に代入
    stop
    end

 (高精度タイマー使用の場合)
    real*8 t                                変数 t は実数型 8 バイト（倍精度）
    call xclock                             CPU 時間測定タイマー起動
   -- プログラム --
    call xclock(t)                          タイマー起動後の CPU 時間（秒）を変数 t
```

```
    write(*,*) t                          に代入
    stop
    end

経過時間計測の例
    real t                                変数ｔは実数型４バイト（単精度）
    call clock(t,7)                       経過時間測定タイマー起動
   -- プログラム --
    call clock(t,8)                       タイマー起動後の経過時間（秒）を変数ｔに
    write(*,*) t                          代入
    stop
    end

（高精度タイマー使用の場合）
    real*8 t                              変数 ｔ は実数型８バイト（倍精度）
    call xclock(t,7)                      経過時間測定タイマー起動
   -- プログラム --
    call xclock(t,8)                      タイマー起動後の経過時間（秒）を変数ｔに
    write(*,*) t                          代入
    stop
    end
```

オプションの詳細および個別のサービスサブルーチンにつきましてはマニュアル「最適化 FORTRAN90 言語（3000-3-C21-30)」または「最適化 FORTRAN77 言語（3000-3-C23-20)」を御覧下さい。

② 他社 Fortran で標準的にサポートされているサービスサブルーチン

コンパイル，リンク時のオプションに -lf90c（FORTRAN77 では -lf77c）を指定することで以下のサービスサブルーチンを使用することができます。

ABORT, ACCESS, ALARM, BIC, BIS, BIT, CHDIR, CHMOD, CLOCKM, CTIME, DTIME, ETIME, FDATE, FGETC, FORK, FPUTC, FREE, FSEEK, FSEEKO64, FSTAT, FSTAT64, FSYNC, FTELL, FTELLO64, GETC, GETCWD, GETFD, GETGID, GETLOG, GETPID, GETUID, GMTIME, HOSTNM, IDATE, IERRNO, INMAX, ISATTY, ITIME, KILL, LINK, LSTAT, LSTAT64, LTIME, MALLOC, PUTC, QSORT, RENAME, SECOND, SETBIT, SIGNAL, SLEEP, STAT, STAT64, SYMLNK, SYSTEM, TIME, TTYNAM, UNLINK, WAIT

```
% f90  program.f  -lf90c                 サービスサブルーチンを使用（f90）
% f77  program.f  -lf77c                 サービスサブルーチンを使用（f77）
```

オプションの詳細および個別のサービスサブルーチンにつきましてはマニュアル「最適化 FORTRAN90 言語（3000-3-C21)」または「最適化 FORTRAN77 言語（3000-3-C23)」を御覧下さい。

11. 並列アプリケーション

並列アプリケーションには，複数ノードで同一のプログラムをパラメーターを変えて実行する並列実行コマンドと，各 CPU で実行されるプログラムが通信を行いながら一つの計算を行う MPI プログラムの 2 種類があります。

11.1. 並列実行（prun コマンド）

プログラムを複数のノードで並列に実行するコマンドです。定義ファイルを用いるとデーターのファイル名にプロセス番号等を付加することができるので複数の異なる入出力データーに対してプログラムを並列に実行することができます。

使用例

```
% cat a.f                                          … サンプルプログラム
        read(5,*) a,b
        write(6,*) a+b,a-b
        stop
        end

% f90 a.f                                          … コンパイル
f90: compile start : a.f

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat sample.def                                   … 定義ファイル
*2 ./a.out < data.%n > result.%n                    2 ノードを使用して 2 種類のデーター
                                                    data.1, data.2 をノード毎に入力し，結
% ls                                                果をそれぞれ result.1, result.2 に出力
a.f      data.1     job.csh                         する。
a.out    data.2  sample.def

% cat job.csh                                      … ジョブスクリプト例
#!/bin/csh
#@$-q debug                                         debug キュー使用
#@$-N 2                                             2 ノード使用
cd  {sample.def のあるディレクトリーを指定}
prun -f sample.def                                  prun コマンドを使用
                                                    （定義ファイル指定）
% qsub job.csh
Request xxxxx.n121 submitted to queue: debug.     … ジョブを投入

～ バッチジョブ終了後 ～

% ls
a.f      data.1     job.csh      result.2
a.out    data.2    result.1    sample.def         … result.1, result.2 が作成される
```

定義ファイルは "*ノード数　プログラム < 入力 > 出力" と記述します。
また，以下の記号を使用できます。

定義ファイル記述用の記号

リダイレクト入出力指定		ファイル名修飾	
<	標準入力	%n	1～並列プロセス数の数値
>	上書き 標準出力	%r	各プロセスが動作している相対ノード番号
>&	〃 標準エラー出力	%a	各プロセスが動作している絶対ノード座標
>>	追記 標準出力	%t	prun が起動した時刻
>>&	〃 標準エラー出力	%d	prun が起動した日付
		%p	起動した並列プロセスのプロセス ID

11.2. MPI

MPI（Message Passing Interface）は MPI Forum によって標準化されたメッセージ通信ライブラリーのインターフェース規約です。ノード間およびノード内のプロセス間通信に MPI 通信ライブラリーを用いたメッセージ通信ができます。多くの計算機に実装されており移植性の高いインターフェースです。なお,本センターの SR11000 では MPI-2 をサポートしています。

使用例

```
% cat mpi_sample.f                              … サンプルプログラム
      program sample
      include 'mpif.h'

      integer size, rank, ierr

      call MPI_INIT(ierr)
      call MPI_COMM_SIZE(MPI_COMM_
     &         WORLD, size, ierr)
      call MPI_COMM_RANK(MPI_COMM_
     &         WORLD, rank, ierr)

      write(*,*) 'proc=',rank, 'size=',size

      call MPI_FINALIZE(ierr)
      stop
      end

% mpif90 mpi_sample.f                           … コンパイル
f90: compile start : mpi_sample.f                 MPI コンパイルコマンドを使用

*OFORT90 V01-05-/C entered.
*program name = SAMPLE
*end of compilation : SAMPLE
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                   … ジョブスクリプト例
#!/bin/csh
#@$-q debug                                       ジョブクラス debug を使用
#@$-N 2                                           2 ノード使用
#@$-J SS   (#@$-J T16 でも良い)                   ジョブタイプ SS を設定
cd  {a.out のあるディレクトリーを指定}
mpirun ./a.out                                    MPI 実行コマンドを使用
                                                  (2 ノード 32 プロセスで実行)
% qsub job.csh                                 … ジョブを投入
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～
```

```
% cat job.csh.oxxxxx                              … 結果を表示
 proc=           0 size=          16
 proc=           1 size=          16
                 :
 proc=          31 size=          16
```

MPI 使用における注意点

(1) MPI を使用したプログラムのコンパイルには **mpif90** または **mpif77**（C 言語は **mpicc**, C++ は **mpiCC**）という専用のコマンドが用意されています。これらのコマンドは MPI を使用する場合に必要なヘッダーファイルやリンクするライブラリー，コンパイルオプションを自動的に設定し，f90 または f77 コンパイラーを起動します。

MPI プログラムのコンパイルオプション指定例を以下に示します。

```
% mpif90  -o  program  program.f          実行ファイル program を作成する
% mpif77  program.f                       f77 でコンパイルする
% mpif90  -Oss  -noparallel  program.f    スカラープログラムとしてコンパイルする
```

(2) MPI は同一ノードで複数のプロセスを同時に実行することもできます。例えば, P008 キュー（8 ノード）を使用すると 128 並列（8 ノード×16CPU）の計算が可能となります。使用例にジョブタイプ SS（#@$-J SS）という記述がありますが，プロセスを CPU 数分生成するためにはジョブタイプ SS（スカラープログラムによるノード共有属性）の指定が必要です。（またはジョブタイプ T16 でも同じです。）なお，この設定においては要素並列化されたプログラムの実行はできません。

(3) MPI プログラムの実行には **mpirun** コマンドを使用します。mpirun コマンドが生成するプロセス数は，ジョブスクリプトで指定したノード数（#@$-N）とジョブタイプ（#@$-J SS または Tn）の有無で決定されます。

ジョブタイプ SS または Tn を指定しない場合は，各ノードに 1 つずつプロセスを生成します。

ジョブタイプ SS を指定した場合は，各ノードに CPU 数分のプロセスを生成し，全体のプロセス数はノード数×CPU 数（16 または 8）となります。CPU 数はバッチキューにより異なります。

ジョブスクリプト例 1

```
#@$-q debug
#@$-N 2
cd test
mpirun ./a.out  → 2 ノード 2 プロセスで実行します。
```

ジョブスクリプト例 2

```
#@$-q debug
#@$-N 2
#@$-J SS
cd test
mpirun ./a.out  → 2 ノード 32 プロセスで実行します。
```

なお mpirun コマンドの –n および –np オプションは無効になりますので御注意下さい。

(4) 同一ノードで複数のプロセスを実行するとメモリー不足で実行時エラー（System error: Not enough space）となることがあります。メモリー制限値，プログラムサイズを確認して下さい。

(5) MPI プログラムでは送信と受信が一対一に対応していないと受信または送信側のプロセスが待ち続け，処理が進まない「デッドロック」の状態に陥ることがあります。プログラム作成の際には御注意下さい。

12. 数値計算ライブラリー

SR11000 システムには数値計算ライブラリーとして MATRIX/MPP, MATRIX/MPP/SSS, MSL2, BLAS, LAPACK, ScaLAPACK があります。これらのライブラリーを利用するには f90, f77 コマンドのオプションとして

–L ライブラリー検索パス名
–l ライブラリー名

を指定します。–l オプションはプログラムファイル名の後ろに指定します。このオプションは左から順に処理されるのでライブラリー名を指定する順序には注意して下さい。なお, センター提供の数値計算ライブラリーの検索パスは標準で設定されていますので, –L オプションは省略できます。

12.1. MATRIX/MPP

基本配列演算, 連立 1 次方程式, 逆行列, 固有値・固有ベクトル, 高速フーリエ変換, 擬似乱数等に関する副プログラムライブラリーです。並列処理用インターフェースを用いることにより, データーを各ノードに分散して配置, 並列に実行することができます。MATRIX/MPP を使用する場合にはコンパイル時にオプションとして以下のライブラリーを指定します。(要素並列版は–parallel オプションも同時に指定して下さい。)

MATRIX/MPP	(要素並列版)	–lmatmpp
	(スカラー版)	–lmatmpp_sc
MATRIX/MPP/SSS (スカイライン法)	(要素並列版)	–lmatmpps
	(スカラー版)	–lmatmpps_sc

参考マニュアル 「行列計算副プログラムライブラリ MATRIX/MPP」(3000-3-C87)
「行列計算副プログラムライブラリ -スカイライン法- MATRIX/MPP/SSS」(3000-3-C88)

使用例 (単一ノードの例)

```
% cat hsru1m.f                                  … サンプルプログラム
      parameter ( n=10 )                          逐次処理用インターフェースの例
      implicit real*4(a-h,o-z)
      dimension x(n)
      ix=0
      call hsru1m(n,ix,x,ier)
      do 10 i=1,n
        write(6,*) i,x(i)
  10  continue
      end

% f90 hsru1m.f -lmatmpp_sc                     … コンパイル
f90: compile start : hsru1m.f                    スカラー版ライブラリーを使用

*OFORT90 V01-05-/C entered.
```

```
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                    … ジョブスクリプト例
#!/bin/csh
#@$-q single
cd  {a.outのあるディレクトリーを指定}
./a.out

% qsub job.csh                                   … ジョブを投入
Request xxxxx.n121 submitted to queue: A

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                            … 結果を表示
           1  0.148270369
           2  0.158839539
           3  0.645628750
                 :
          10  0.629399896

% f90 -parallel hsru1m.f -lmatmpp               … コンパイル
f90: compile start : hsru1m.f                    要素並列版ライブラリーを使用

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                    … ジョブスクリプト例
#!/bin/csh
#@$-q debug
#@$-N 1
cd  {a.outのあるディレクトリーを指定}
./a.out

% qsub job.csh                                   … ジョブを投入
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                            … 結果を表示
           1  0.148270369
           2  0.158839539
           3  0.645628750
                 :
          10  0.629399896
```

使用例（複数ノード実行の例）

```
% cat hdru3mdp.f                                … サンプルプログラム
      parameter ( n=20,npu=2 )
      implicit real*8(a-h,o-z)
      dimension x(n),lstpu(0:npu-1),
     & iopt2(2),ienv(4)
      do 10 k=0,npu-1
         lstpu(k)=k
  10  continue
      iwksize=max(128,(npu+1)*8)
      call hmatinit(iwksize,lstpu,npu,ier)
      call hkxpara(ienv)
      me=ienv(1)
      ix=0
```

```
      iopt1=1
      iopt2(1)=1
      iopt2(2)=1
      call  hdru3mdp(n,ix,lstpu,npu,iopt1,        並列処理用インターフェース
     & iopt2,x,ier)                               (2 ノードを使用する例)
      write(6,*) me, n, ier
      do 20 i=1,n
        write(6,*) i,x(i)
  20  continue
      end

% mpif90 -parallel hdru3mdp.f -lmatmpp          … コンパイル
f90: compile start : hdru3mdp.f                   mpif90 コマンドおよび要素並列版ライブ
                                                  ラリーを使用
*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                   … ジョブスクリプト例
#!/bin/csh
#@$-q debug
#@$-N 2
cd  {a.out のあるディレクトリーを指定}
mpirun ./a.out

% qsub job.csh                                  … ジョブを投入
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                           … 結果を表示
          0          20           0
          1  0.108420217337368285E-005
                     :
```

12.2. MSL2

行列計算（連立 1 次方程式，逆行列，固有値・固有ベクトル等），関数計算（非線形方程式，常微分方程式，数値積分等），統計計算（分布関数，回帰分析，多変量解析等）に関する副プログラムライブラリーです。MSL2 を使用するためにはコンパイル時にオプションとして以下のライブラリーを指定します。（要素並列版は–parallel オプションも同時に指定して下さい。）

MSL2	（要素並列版）	–lMSL2P
	（スカラー版）	–lMSL2

参考マニュアル　「数値計算副プログラムライブラリ MSL2 操作」(3000-3-C86)
「数値計算副プログラムライブラリ MSL2 行列計算」(3000-3-C83)
「数値計算副プログラムライブラリ MSL2 関数計算」(3000-3-C84)
「数値計算副プログラムライブラリ MSL2 統計計算」(3000-3-C85)

使用例（スカラー処理の例）

```
% cat msgu1m.f                                    … サンプルプログラム
      parameter ( n=10 )
      implicit real*4(a-h,o-z)
      dimension x(n)
      ix=0
      call msgu1m(n,ix,x,ier)
      do 10 i=1,n
        write(6,*) i,x(i)
   10 continue
      end

% f90 msgu1m.f -lMSL2                             … コンパイル
f90: compile start : msgu1m.f                       スカラー版ライブラリーを使用

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                     … ジョブスクリプト例
#!/bin/csh
#@$-q single
cd  {a.outのあるディレクトリーを指定}
./a.out

% qsub job.csh                                   … ジョブを投入
Request xxxxx.n121 submitted to queue: A

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                             … 結果を表示
           1  0.148270369
           2  0.158839539
           3  0.645628750
              :
          10  0.629399896
```

使用例（要素並列処理の例）

```
% cat msvafm.f                                   … サンプルプログラム
       parameter (n=2,m=2)
       real a(n,m),b(n,m),r(n,m)
       iopt=2
       data ((a(i,j),i=1,n),j=1,m)/1,2,3,4/
       data ((b(i,j),i=1,n),j=1,m)/1,2,3,4/
       call msvafm(a,n,m,n,b,n,iopt,r,n,ier)
       write(*,*) ((r(i,j),i=1,n),j=1,m)
       stop
       end

% f90 -parallel msvafm.f -lMSL2P                  … コンパイル
f90: compile start : msvafm.f                       要素並列版ライブラリーを使用

*OFORT90 V01-05-/C entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                     … ジョブスクリプト例
#!/bin/csh
#@$-q debug
```

```
#@$-N 1
cd  {a.out のあるディレクトリーを指定}
./a.out

% qsub job.csh                                        … ジョブを投入
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                                  … 結果を表示
   2.00000000        4.00000000        6.00000
000        8.00000000
```

12.3. BLAS・LAPACK・ScaLAPACK

BLAS（Basic Linear Algebra Subprogram）はベクトル，行列に関する基本演算ライブラリー，LAPACK（Linear Algebra PACKage）は連立 1 次方程式，固有値，固有ベクトルなどの線形計算ライブラリー，ScaLAPACK（Scalable Linear Algebra PACKage）は並列版の行列計算ライブラリーです。これらのライブラリーの機能の詳細は以下の Web ページを御覧下さい。

http://www.netlib.org/blas/
http://www.netlib.org/lapack/
http://www.netlib.org/scalapack/

これらのライブラリーを使用する場合にはコンパイル時にオプションとして以下のライブラリーを指定します。（要素並列版は –parallel オプションも同時に指定して下さい。）

	（要素並列版）	（スカラー版）	
BLAS	–lblas	–lblas_sc	
LAPACK	–llapack –lblas	–llapack_sc –lblas_sc	
ScaLAPACK	–lscalapack –lblacsBASE –lblacsF77 –lblas	–lscalapack_sc –lblacsBASE_sc –lblacsF77_sc –lblas_sc	(ScaLAPACK) (BLACS Base) (BLACS Fortran) (BLAS)

ライブラリーオプションの指定順序は使用例を参照して下さい。

なお，ScaLAPACK の通信関数には MPI を使用していますのでコンパイルには MPI ライブラリーの指定も必要です。（mpif90，mpif77 コマンド使用の場合は MPI ライブラリーの指定は省略できます。）

使用例（BLAS）

```
% cat blas.f                                          … サンプルプログラム
      program blas
      parameter ( n=4 )
      dimension x(n)
      do 10 i=1,n
        x(i)=1d0*i
```

```
   10 continue
      alpha=2d0
      incx=1
      call dscal(n,alpha,x,incx)
      do 20 i=1,n
        write(*,*) x(i)
   20 continue
      stop
      end

% f90 -parallel blas.f -L/usr/local/lib -lblas     … コンパイル
f90: compile start : blas.f                          要素並列版ライブラリーを使用

*OFORT90 V01-05-/C entered.
*program name = BLAS
*end of compilation : BLAS
*program units = 0001, no diagnostics generated.

% cat job.csh                                       … ジョブスクリプト例
#!/bin/csh
#@$-q debug
#@$-N 1
cd  {a.out のあるディレクトリーを指定}
./a.out

% qsub job.csh                                      … ジョブを投入
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                               … 結果を表示
   1.12500000
   2.00000000
   3.50000000
   4.00000000
```

使用例（LAPACK）

```
% cat lapack.f                                     … サンプルプログラム
       program lapack
       parameter (n=2,lda=n+1,ldb=n+1,nrhs = 1)
       double precision a(lda,n),b(ldb,nrhs)
       integer ipiv(n),info
       data ((a(i,j),j=1,n),i=1,n)/2d0,4d0,1d0,
     & 1d0/
       data (b(i,1),i=1,n)/14d0,5d0/
       call dgesv(n,nrhs,a,lda,ipiv,b,ldb,info)
       do 10 i=1,n
       write(*,*) b(i,nrhs)
 10    continue
       stop
       end

% f90 -parallel lapack.f -L/usr/local/lib          … コンパイル
 -llapack -lblas                                     要素並列版ライブラリーを使用
f90: compile start : lapack.f

*OFORT90 V01-05-/C entered.
*program name = LAPACK
*end of compilation : LAPACK
*program units = 0001, no diagnostics generated.

% cat job.csh                                       … ジョブスクリプト例
#!/bin/csh
```

```
#@$-q debug
#@$-N 1
cd  {a.out のあるディレクトリーを指定}
./a.out

% qsub job.csh                                    … ジョブを投入
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                              … 結果を表示
   3.00000000000000000
   2.00000000000000000
```

使用例（ScaLAPACK）

```
% mpif90 -Oss -noparallel example1.f -i,L          … コンパイル（mpif90 コマンド）
  -L/usr/local/lib -lscalapack_sc -lblacsF77_sc
  -lblacsBASE_sc -lblas_sc                          サンプルプログラムは
f90: compile start : example1.f                     http://www.netlib.org/scalapack/
                                                    examples/ にある example1.f を使用
*OFORT90 V01-05-/C entered.
*program name = example1
   KCHF476K  00               DESCB
             the variable is declared, but never
appears in an any executable statement.
*program name = matinit
*program name = sl_init
*end of compilation : example1
*end of compilation : matinit
*end of compilation : sl_init
*program  units  =  0003,  0001    diagnostics
generated, highest severity code is  00
                                                  … ジョブスクリプト例
% cat job.csh
#!/bin/csh
#@$-q debug
#@$-N 1
#@$-J SS       （または #@$-J T16）
cd  {a.out のあるディレクトリーを指定}               実行は mpirun コマンドを使用
mpirun ./a.out
                                                  … ジョブを投入
% qsub job.csh
Request xxxxx.n121 submitted to queue: debug.

～ バッチジョブ終了後 ～
                                                  … 結果を表示
% cat job.csh.oxxxxx

ScaLAPACK Example Program #1 -- May 1, 1997

Solving Ax=b where A is a   9 by   9 matrix with
a block size of   2
Running on   6 processes, where the process grid
is   2 by   3

INFO code returned by PDGESV =   0

According to the normalized residual the solution
is correct.

||A*x  -  b||  /  (  ||x||*||A||*eps*N  )  =
0.00000000E+00
```

・フリーソフトウェア ATLAS について

SR11000 上にフリーソフトウェアの ATLAS（Automatically Tuned Linear Algebra Software）を公開しています。これは，BLAS ライブラリーおよび LAPACK ライブラリーとして利用できますが，キャッシュを効率的に利用するなど高速化が図られています。詳細につきましては，下記の Web ページを御覧下さい。なお，インストールされているバージョンは 3.6.0 です。

http://math-atlas.sourceforge.net/

- ATLAS のインストール先

/usr/local/unsupported/atlas/include/	C 言語用ヘッダファイル
/usr/local/unsupported/atlas/lib/	ライブラリー本体

32bit スカラー版のみの提供となっており，64bit 版および要素並列版はありません。また，C 言語用の BLAS インターフェースである CBLAS を利用する場合には，プログラム中で下記のように cblas.h ファイルをインクルードして下さい。

```
#include "/usr/local/unsupported/atlas/include/cblas.h"
```

本ライブラリーを利用する場合には，コンパイル時にオプションとして以下のように指定して下さい。

- Fortran での利用の仕方

（BLAS のみを利用）

```
-L/usr/local/unsupported/atlas/lib -lf77blas -latlas
```

（LAPACK を利用）

```
-L/usr/local/unsupported/atlas/lib -llapack -lcblas -lf77blas -latlas
```

- C 言語での利用の仕方

（CBLAS のみを利用）

```
-L/usr/local/unsupported/atlas/lib -lcblas -latlas
```

（LAPACK を利用）

```
-L/usr/local/unsupported/atlas/lib -llapack -lcblas -lf77blas -latlas
```

本ライブラリーの利用に関しての保証・サポートは行っておりません。使用方法，性能，障害等に関する質問等については一切回答できませんので，予めご了承下さい。また，予告なくバージョンアップや不具合修正などの更新を行うことがあります。最新の情報に関しては，必ずセンターの Web ページにてご確認下さい。

13. ファイル入出力

13.1. データー形式

SR11000 の浮動小数点形式は以下の表現範囲になっています。単精度および倍精度浮動小数点形式は IEEE 形式に準拠しています。

精度	PowerPC 形式
単精度	$\pm 1.175495 \times 10^{-38}$～ $\pm 3.402823 \times 10^{38}$
倍精度	$\pm 2.225074 \times 10^{-308}$～ $\pm 1.797693 \times 10^{308}$
拡張精度	$\pm 2.225074 \times 10^{-308}$～ $\pm 1.797693 \times 10^{308}$

13.2. ファイル形式

UNIX システムの書式なし入出力文で入出力する場合のファイル形式には Fortran 固有ファイルと標準書式なしファイルとがあり，標準書式なしファイルはさらにファイル位置付け動作の異なる 2 種類（ファイル形式は同じ）に分けられます。

書式なし	標準書式なしファイル	業界標準書式なし（デフォルト）
		標準書式なし
	Fortran 固有ファイル	
書式付き	標準テキストファイル	

標準書式なし（stduf）と業界標準書式なし（dstduf）は同じファイル形式ですが，ファイル終了記録検出後の BACKSPACE 文の動作が以下のように異なります。

`stduf`　ファイル終了記録検出後の BACKSPACE 文の実行によって，ファイルポインターがファイル終了記録の直前のデーターの先頭に位置付けられます。

`dstduf`　ファイル終了記録検出後の BACKSPACE 文の実行によって，ファイルポインターがファイル終了記録の直前に位置付けられます。

SR11000 のデフォルトの形式は業界標準書式なしファイルです。

標準書式なしファイルとして入出力を行う場合は

```
% ./a.out -F'PORT(STDUF)'
```

とします。

13.3. ファイル接続

装置番号を使用して，入出力文とファイルを接続するには以下の方法があります。

① 標準入出力

装置番号 5 が標準入力（キーボード），装置番号 6 が標準出力（画面）に接続されていますので，例のような端末に対する入出力を行うことができます。

```
端末から入力，端末へ出力する例
% cat program.f
       read(5,*) a,b
       write(6,*) a*b,a/b
       stop
       end
% f90 program.f
（メッセージは省略）
% ./a.out
2 3                                          …  端末（キーボード）から入力。
   6.00000000       0.666666687              …  端末の画面に表示される。
```

標準入出力はシェルの機能を利用してデーターをファイルから入力したり，結果をファイルに出力したりすることができます。

```
ファイル a.data から入力し，標準出力へ出力する例
% cat a.data
2 3
% ./a.out < a.data                           …  a.data から入力。
   6.00000000       0.666666687

ファイル a.data から入力し，ファイル a.result に出力する例
% cat a.data
2 3
% ./a.out < a.data > a.result                …  a.data から入力，  a.result に出力。
% cat a.result
   6.00000000       0.666666687              …  結果
```

② OPEN 文

装置番号 n と指定した既存のファイルを接続,または指定したファイル名で新規にファイルを生成して接続することができます。

```
装置番号 10 を既存のファイル data に書式なし入力文として接続する例
      open(10,file='data',stauts='old',form='unformatted')
      read(10) a,b
      write(6,*) a*b,a/b
      close(10)
      stop
      end
装置番号 11 を新規に生成したファイル result に書式付き出力文として接続する例
      open(11,file='result',stauts='new',form='formatted')
      read(5,*) a,b
      write(11,100) a*b,a/b
  100 format(1h ,2d12.4)
      close(11)
      stop
      end
```

③ 環境変数 FT*nn*F*xxx*

OPEN 文を使用すると上記の例のようにプログラム中で指定したファイルと接続できますが，OPEN 文を使用しない場合は環境変数 FT*nn*F*xxx* を使用してファイル名を指定できます。ここで，*nn* は装置番号，*xxx* は Fortran 順序番号です。（Fortran 順序番号は，

同一の装置番号を使用して，異なるファイルを順次処理する場合などに利用するもので，プロブラムの実行開始時点では 001 です。）環境変数の英字は大文字でなければなりません。設定方法は以下のとおりです。

使用例

```
% cat program.f
      read(10,*) a,b
      write(11,100) a*b,a/b
  100 format(1h ,2d12.4)
      stop
      end
% f90 program.f
 (メッセージは省略)
% cat data
2 3
% setenv FT10F001 data                    …  環境変数の設定（入力ファイル data)
% setenv FT11F001 result                  …  環境変数の設定（出力ファイル result)
% ./a.out
% cat result
   0.6000E+01  0.6667E+00

 (NQS スクリプトの場合)
#!/bin/csh
#@$-q single
#@$-lT 00:30:00
#@$-lM 1GB
cd work
setenv FT10F001 data
setenv FT11F001 result
./a.out
```

④ 特定ファイル名 ft.*nn*

OPEN 文も環境変数も設定しない場合には，READ，WRITE 文は特定ファイル名 ft.*nn*（*nn* は装置番号）に接続します。READ 文の場合，ファイルが存在しないとエラーになります。WRITE 文の場合，ファイルが存在しないと新しく作成します。既に存在する場合は，先頭から書き込まれます。

使用例

```
標準入力（端末）から入力，ファイル ft.11 へ出力する例
% cat a.f
       read(5,*) a,b
       write(11,*) a*b,a/b
       stop
       end
% f90 a.f
 (メッセージは省略)
% ./a.out
2 3                                          …  端末（キーボード）から入力。
% cat ft.11
   6.00000000       0.666666687              …  ファイルがなければ作成される。

ファイル ft.10 から入力，ファイル ft.11 へ出力する例
 (ファイル ft.10 が存在しないとエラーになる)
% cat a.f
      read(10,*) a,b
      write(11,*) a*b,a/b
      stop
      end
```

```
% f90 a.f
 （メッセージは省略）
% ./a.out
KCHF348R the read statement for unit id 10 is invalid. the file does not exist.
file name is ft.10.

 （ファイル ft.10 を作成し，ft.10 から入力，ファイル ft.11 へ出力する）
% vi ft.10                                                …  入力ファイル
2 3
% ./a.out
% cat ft.11
   6.00000000       0.666666687
```