

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えます。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 年 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

1. システム概要	1
1.1. システム構成.....	1
1.2. ノード構成.....	2
1.3. ソフトウェア.....	2
1.4. フリーソフトウェア.....	3
2. 利用者登録	5
3. ログイン方法	6
3.1. 概要.....	6
3.2. 接続情報.....	6
3.3. WINDOWS システムからのログイン方法.....	6
① PuTTY の入手.....	6
② 鍵の生成.....	7
③ 鍵の SR11000 システムへの登録.....	8
④ ログイン.....	8
⑤ ファイルコピー.....	10
3.4. UNIX システムからのログイン方法.....	10
① 鍵の生成.....	10
② 鍵の SR11000 システムへの登録.....	11
③ ログイン.....	11
④ ファイルコピー.....	11
3.5. 鍵について (WINDOWS, UNIX 共通).....	11
① Web 登録システムの有効期間.....	11
② Web 登録システムに入力できる鍵.....	11
③ 鍵の追加と変更.....	12
④ 秘密鍵の管理.....	12
⑤ 別システムへのログインに使用している鍵の流用.....	12
4. シェルとコマンド	13
4.1. ログインメッセージ.....	13
4.2. ログアウト.....	13
4.3. シェルとコマンド.....	14
① 標準入出力.....	15

② 初期設定ファイル.....	16
③ プロセスの強制終了.....	17
④ オンラインマニュアル.....	17
⑤ 電子メール.....	18
⑥ ファイル転送.....	18
⑦ CPU リソース情報.....	19
5. ファイルとディレクトリー.....	20
6. 要素並列とジョブの形態.....	23
6.1. 要素並列処理.....	23
6.2. ジョブの形態.....	25
① スカラージョブ.....	25
② 要素並列ジョブ.....	25
③ ノード内要素並列+ノード間 MPI ジョブ.....	25
④ ノード内およびノード間 MPI ジョブ.....	26
7. コンパイルと実行.....	27
7.1. FORTRAN.....	27
7.2. C, C++.....	28
7.3. 実行.....	29
8. システム資源.....	30
8.1. CPU 資源.....	31
8.2. メモリー資源.....	31
9. NQS (バッチジョブ).....	34
9.1. NQS とは.....	34
9.2. ジョブクラスとキュー.....	34
9.3. スクリプトファイルの作成.....	35
9.4. ジョブの実行 (投入および確認).....	39
9.5. ジョブの結果.....	41
9.6. ジョブのキャンセル.....	42
9.7. ジョブの開始, 終了メール.....	42
9.8. 実行例.....	43
10. FORTRAN のコンパイルと実行の詳細.....	46
10.1. コンパイルとリンク.....	46

① オプション.....	46
② ファイル名.....	46
10.2. オプションと機能.....	47
① 最適化オプション	48
② 要素並列化オプション	50
③ OpenMP.....	51
④ 64 ビットアドレッシングモード	52
⑤ ログメッセージ	53
⑥ 性能モニター.....	54
⑦ ソースプログラム差分コンパイル	56
⑧ オブジェクト再コンパイル.....	56
⑨ 制限コンパイル	57
⑩ デバッグオプション	57
⑪ トレースバックマップ	58
⑫ 言語仕様の拡張	58
10.3. 実行時オプション.....	59
10.4. サービスサブルーチン	60
① プログラムの時間計測	60
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.....	61
11. 並列アプリケーション.....	62
11.1. 並列実行 (PRUN コマンド)	62
11.2. MPI	63
12. 数値計算ライブラリー.....	66
12.1. MATRIX/MPP.....	66
12.2. MSL2.....	68
12.3. BLAS・LAPACK・SCALAPACK.....	70
13. ファイル入出力.....	74
13.1. データ形式.....	74
13.2. ファイル形式.....	74
13.3. ファイル接続.....	74
① 標準入出力.....	75
② OPEN 文	75
③ 環境変数 FTnnFxxx.....	75
④ 特定ファイル名 ft.nn.....	76

付録

- A vi エディターの使い方
- B 高速ファイル転送ソフト **SRFT** について
- C 複数プログラムの実行方法と環境変数について
- D **Gaussian03** の使い方
- E 超並列ジョブ (64 ノード) 実行サービスについて
- F ジョブスケジューリングシステムについて
- G 計算リソース追加オプション
- H ジョブクラス制限値
- I 利用負担金
- J マニュアル一覧
- K マニュアル Web 閲覧サービス
- L センターによく寄せられる質問とその回答集 (FAQ)

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。

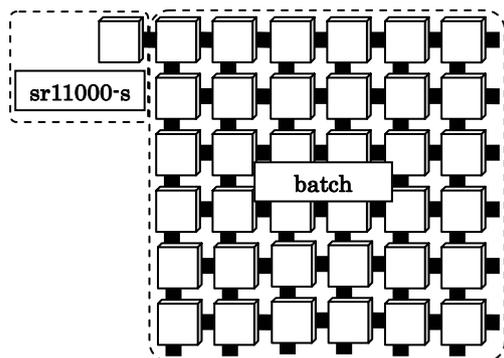
SR11000 は1ノード当たり16台のCPU (プロセッサ) を備えたノードを複数台搭載しており、ノードを並列に動作させるのと同時にノード内でも並列処理が可能です。システムの諸元は以下のとおりです。

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。

インタラクティブ
処理用 バッチ処理用

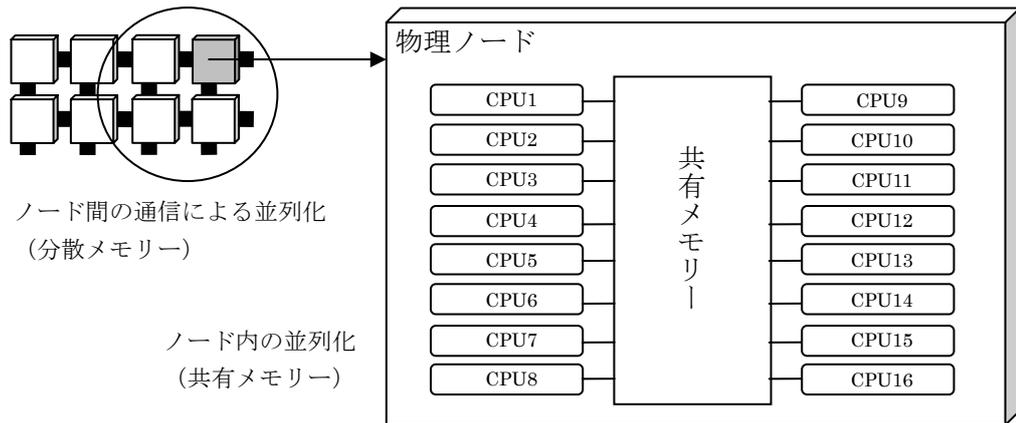


注) 図は実際のノードの配置や数とは異なります。

- ☞ 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」を御覧ください。

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 (コンパイラ)	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」を御覧ください

☞ 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	/opt/freeware/bin
ソフトウェア名称	ソフトウェア名称
emacs	autoconf gmake sed
tcsh	automake gzip tar
bash	bzip2 less vim
less	cvs libtool wget
ATLAS	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ます。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

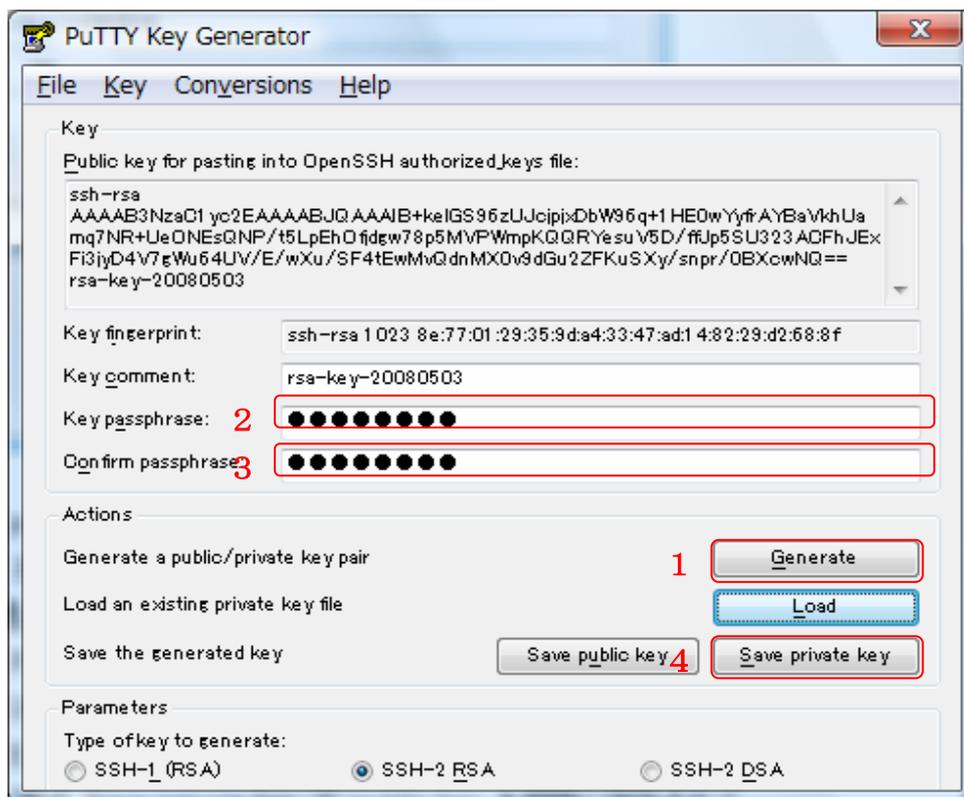
2009年3月現在 PuTTY は日本語化されたものが次のページより入手できます。

<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に、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/>

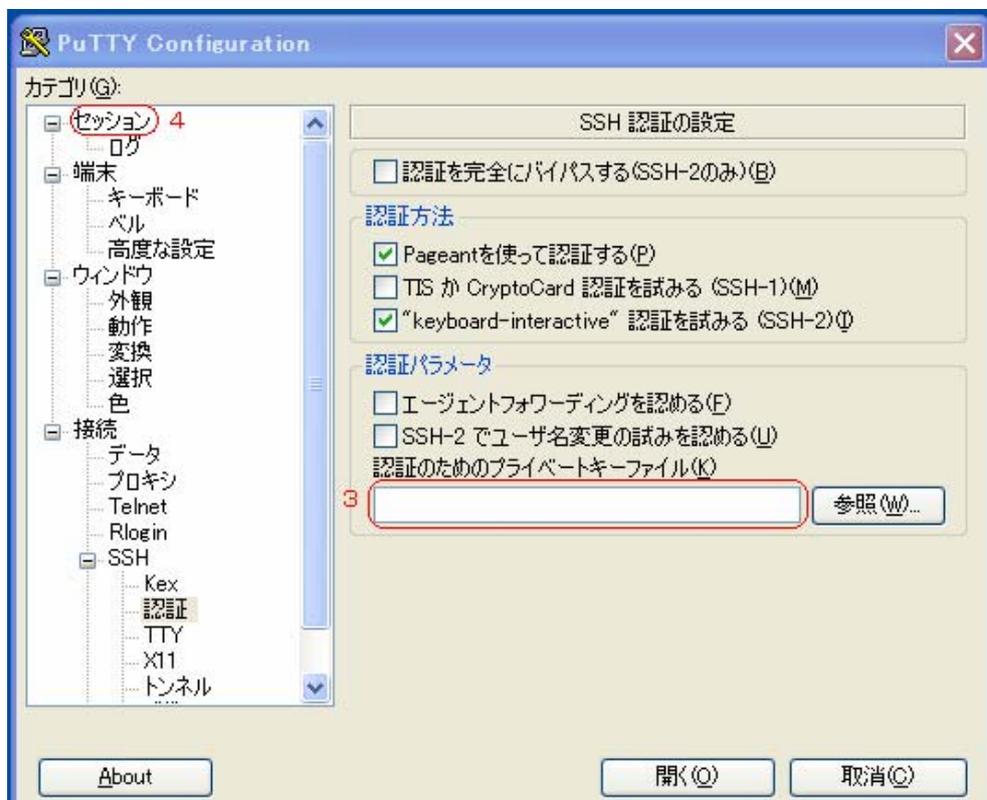
ユーザー名とパスワードを求められるので、センターから通知されたユーザー名とパスワードを入力してください。ここで入力するパスワードは鍵の保護のために入力したパスワードではなく、センター発行のパスワードです。認証に成功するとユーザーページが開きますので「公開鍵登録」のリンクをクリックしてください。開いたページの“Public Key”の欄に PuTTY Key Generator ウィンドウの ”Public key for pasting into OpenSSH authorized_keys file” に表示されている文字列をコピー&ペーストし、”Password” の欄に センター発行の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 回以上行くと、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

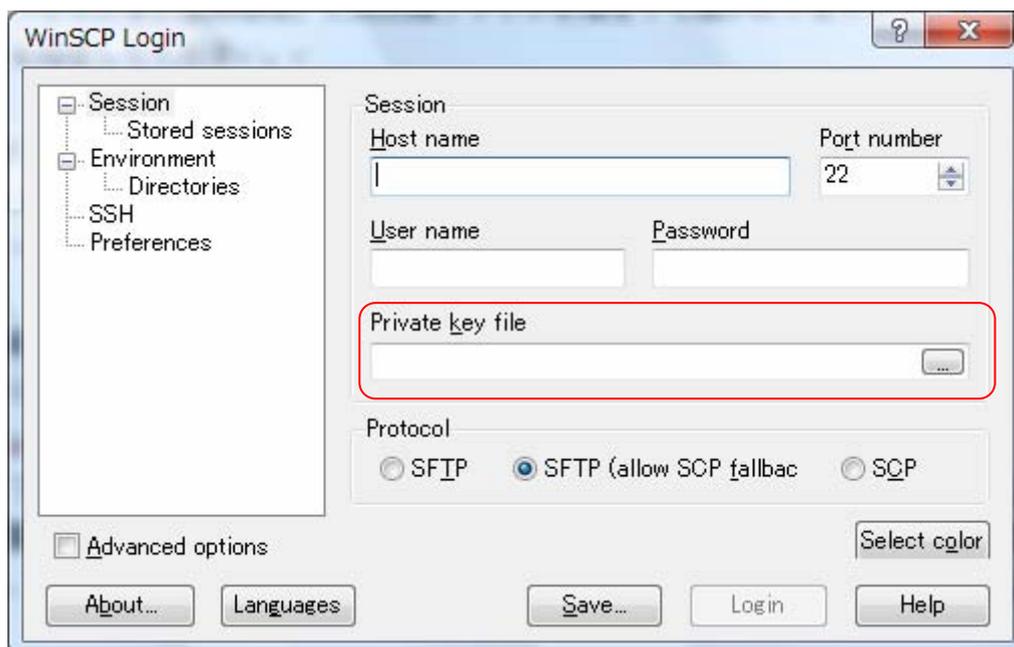
PuTTY を起動すると次ページ上のような画面が出ます。図に示した順番で以下のように情報を設定します

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ワードを聞かれるので、鍵作成の時に設定したパスワードを入力する。



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。次にパスワード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、セキュリティ強化のため SR11000 システムに登録する鍵はパスワードで保護されていることを必須とします。パスワード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ワード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ワードではなく、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いたします。

登録作業を 2 回以上行くと、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
```

でパスワード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:
```

でパスワード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となりますが、データ上は一個の鍵は一行でなくてはなりません。鍵の途中で改行を入れないようご注意ください。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を守る上で非常に重要なので、パスワードや秘密鍵が他人に漏れることのないよう十分ご注意ください。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ワード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、パスワード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ワード付きに変更し、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入力や計算結果のファイルへの出力に大変便利です。覚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
```

☞ ペ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」を御覧ください。

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。

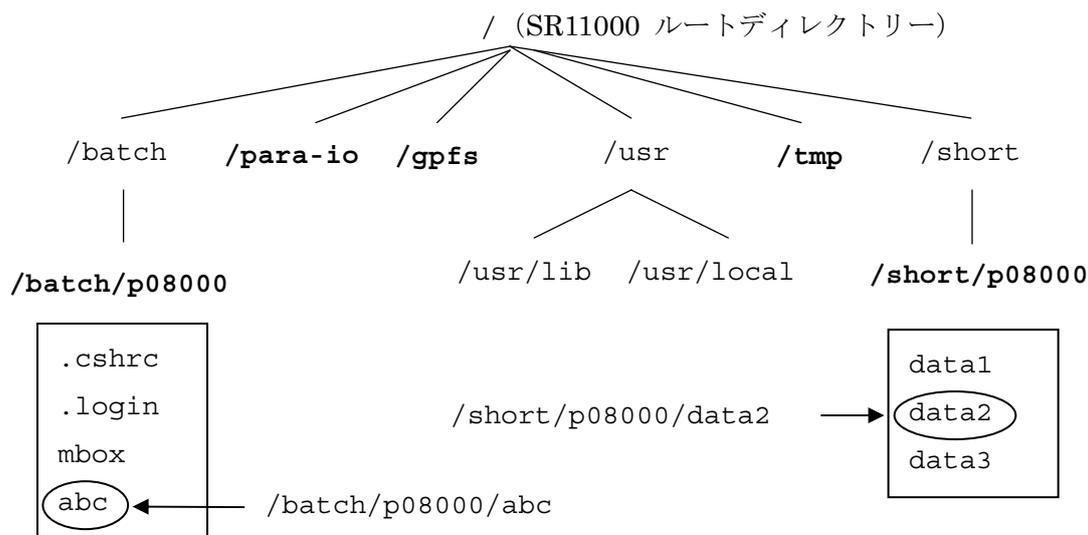
```
% la
cpu resource for p08000 (HH:MM:SS)
  system      limit      used(this month)      remain
SR11000      596523:14:07  143:27:54(   0:04:21)  596379:46:13
              CPU 時間上限値  CPU 使用時間(年間)  CPU 使用時間(当月)      残り CPU 時間
```

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、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）にディレクトリー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、性能が生かれます。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

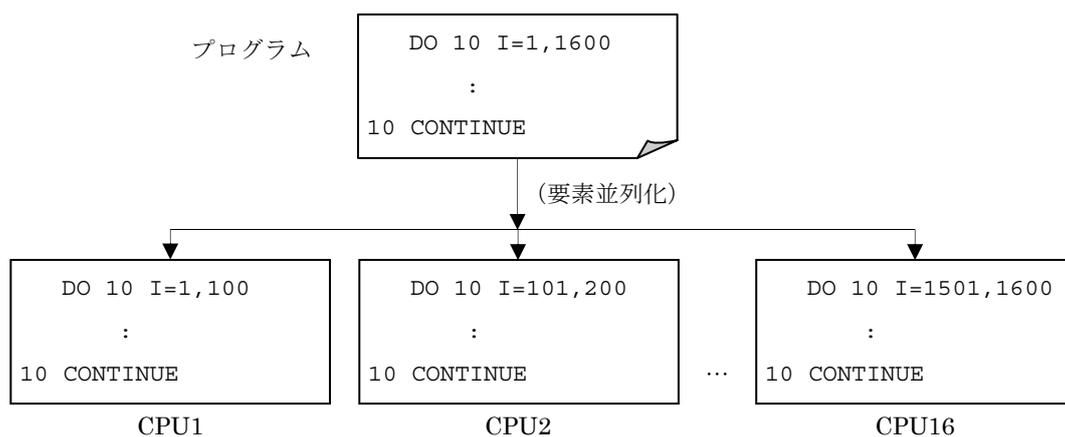
6.1. 要素並列処理

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。

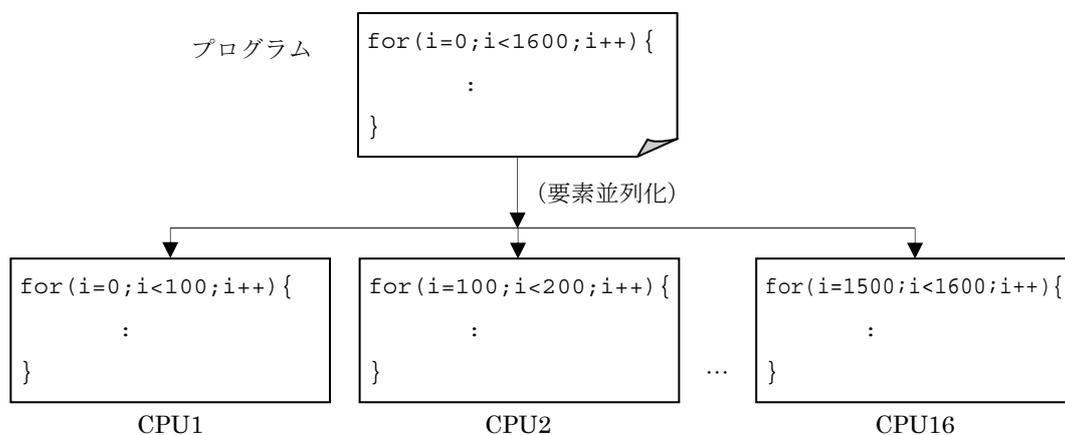
- ループ要素並列化（自動並列化）

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



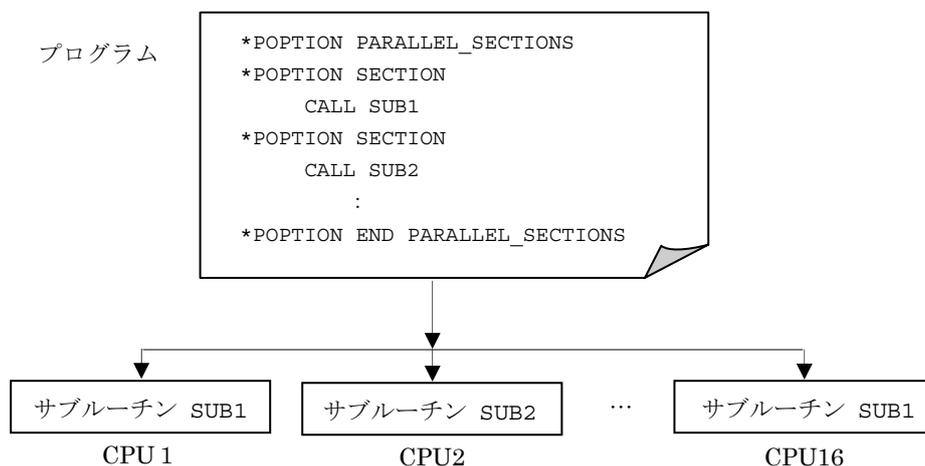
C の場合



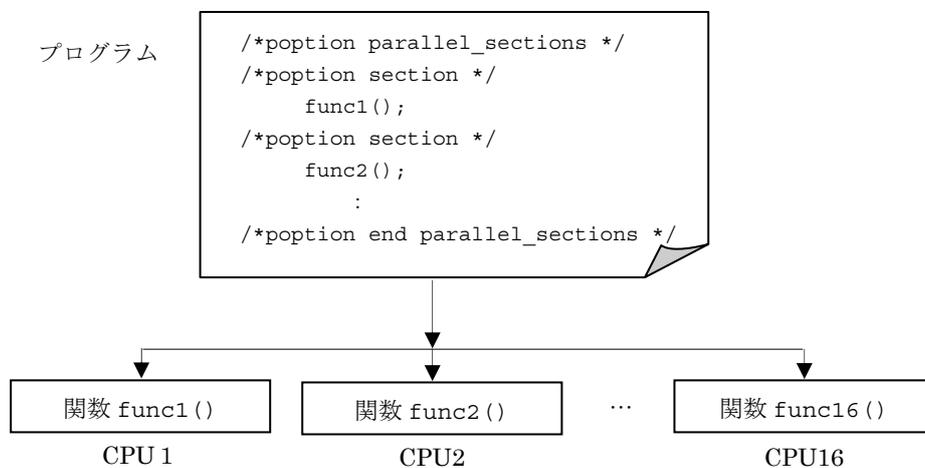
● SECTION 型要素並列化

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



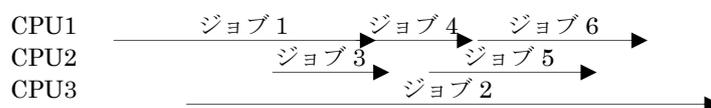
6.2. ジョブの形態

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」を御覧ください。

① スカラージョブ

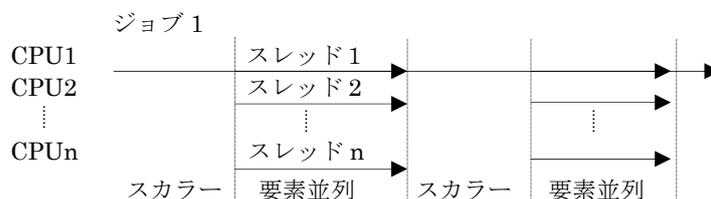
スカラープログラムを実行するジョブです。1CPUのみ使用するので他のジョブとノードを共有し、ノード内で複数実行されます。



☞ 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

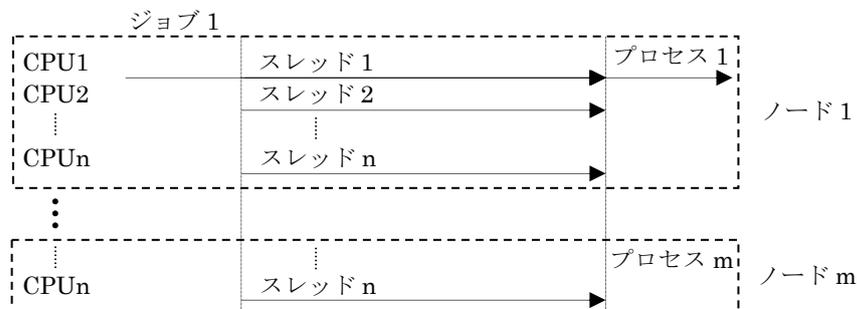
要素並列プログラムを実行するジョブです。要素並列化により分割されたスレッドが各 CPU に配置され並列に実行します。



☞ 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

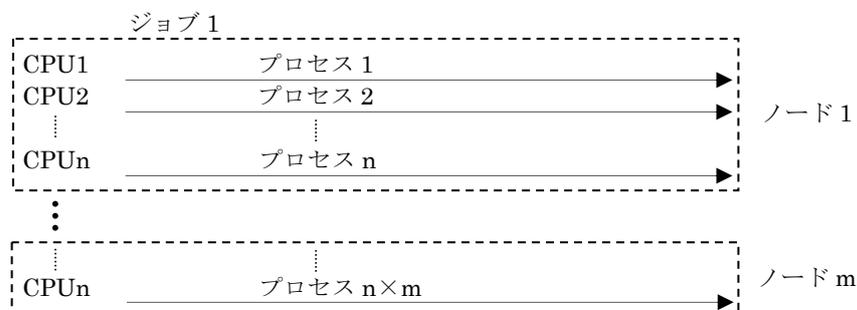
ノード内は要素並列，ノード間は MPI を使用して通信します。m 台のノードを使用すると m プロセス並列が実現できます。



☞ 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 $n \times m$ 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ください。
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。

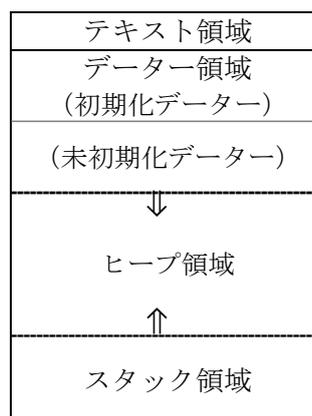
利用者が上限値を変更できるのはジョブまたはセッションが使用するメモリーの大きさ (仮想メモリーサイズ) とデータ領域、スタック領域です。例えば Fortran プログラムでは配列などの変数は通常データ領域に配置しますので不足する場合は仮想メモリーサイズとデータ領域を拡張します。参考までにプロセスごとのメモリー構成を以下に示します。ヒープ領域は仮想メモリーサイズからテキスト領域、データ領域、スタック領域を除いた残り分が使われます。

テキスト領域 (.text)
 プログラム (実行命令等) を配置します。

データ領域
 - 初期化データ (.data)
 初期値のある静的変数を配置します。
 - 未初期化データ (.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
```

データ領域が不足するときはデータ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

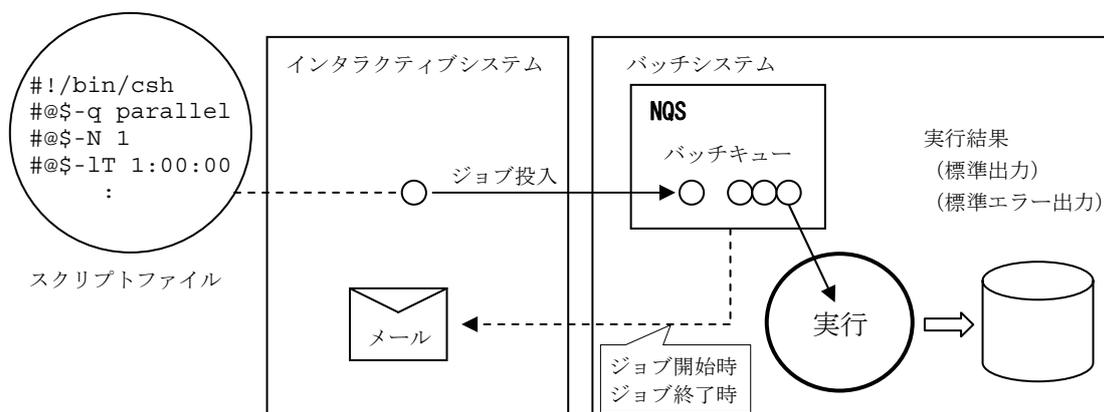
ことはできず、プログラム上で配列を各ノードに分散する必要があります。

- (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」を御覧ください

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」を御覧ください。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。

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

スクリプトファイルの書き方の例です。

```

#!/bin/csh
#@$-q パイプキュー名
#@$-N ノード数
#@$-lT 実行時間制限値
#@$-lM メモリーサイズ制限値
cd ディレクトリー
プログラム名 < 入力ファイル > 出力ファイル

#!/bin/csh
#@$-q parallel
#@$-N 1
#@$-lT 1:00:00
#@$-lM 112GB
cd test
./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。parallel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。

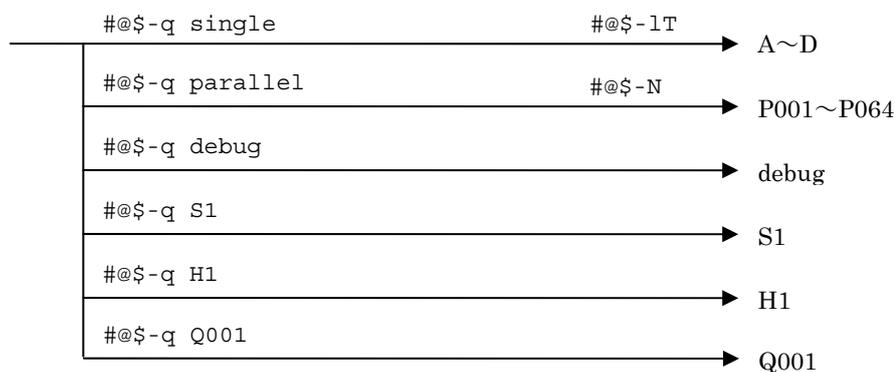
```
#@$-lM メモリーサイズ制限値
```

- ☞ 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」を御覧ください。

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

<pre>*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)</pre>	<pre>*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)</pre>
<pre>*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)</pre>	<pre>*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)</pre>
<pre>*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)</pre>	

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 (#@\$-1T) 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」を御覧ください。

・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
```

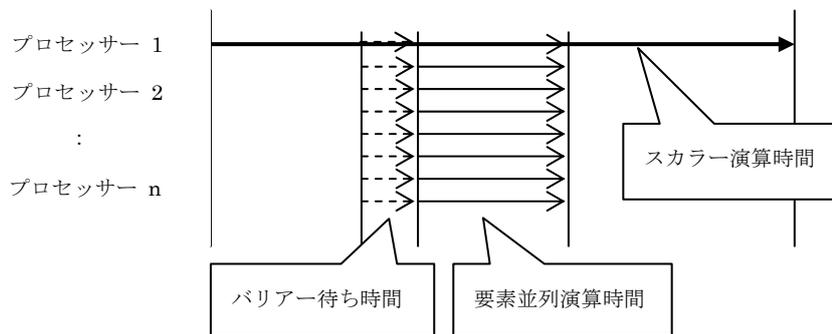
9.7. ジョブの開始, 終了メール

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
#@$-q single          ... スカラージョブなのでキューは single
#@$-IT 1:00:00        ... 実行時間は 1 時間
    
```

```

#@$-IM 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)

スクリプトファイル go.csh をサブミットします。
% gsub 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
0.00000000E+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 (`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', `-W1`, '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` ☞ 10.2.⑩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。

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

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 (**-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カージェネレーターを起動します。

コンパイル, リンクの例

<pre>% f90 main.f sub.f main.f: f90: compile start : main.f *OFORT90 V01-05-/C entered. *program name = MAIN *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 *end of compilation : SUB *program units = 0001, no diagnostics generated. % ls a.out main.f sub.f %</pre>	main.f, sub.f をコンパイルして実行ファイルを作成 (main.f のコンパイルメッセージ) (sub.f のコンパイルメッセージ) (この後リンクが行われる) 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

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

<pre>% f90 -Oss program.f % f90 -Oss -omp program.f % f90 -Oss -nparallel program.f</pre>	自動並列化を行うコンパイル OpenMP 使用時のコンパイル 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を行い，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 (-O3) 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 f90: compile start : program.f	program.f をコンパイルしたときのループ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	D0 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: -O3, -O4 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: -O3, -O4 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 DPRD, 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 `-nooption` 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 `-nooption` 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。

<code>-parallel=</code> 要素並列化レベル	要素並列化オプション
<code>% f90 -parallel program.f</code>	要素並列化 (レベル2) する
<code>% f90 -parallel=4 program.f</code>	要素並列化 (レベル4) する
<code>% f90 -nparallel program.f</code>	要素並列化を抑止
<code>% f90 -parallel main.o sub.o</code>	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。

<code>-nparallel</code> <code>-parallel=0</code>	要素並列化をしない
<code>-parallel=1</code>	SECTION 型要素並列化, 強制ループ要素並列化
<code>-parallel</code> <code>-parallel=2</code>	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
<code>-parallel=3</code>	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
<code>-parallel=4</code>	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 f90: compile start : program.f	program.f を要素並列化してコンパイルし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 is applied to tlocal transformation. name=l line=7	(DO10 ループ内の変数を TLOCAL 化)
KCHF2013K the final value is guaranteed for the variable(s) or array(s) in do 10 loop subject to tlocal transformation. name=l line=7	(DO10 ループ内の TLOCAL 化する変数の終 値保証を行う)
*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)」を御覧ください。

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

<pre>% cat omp.f real a(1000) !\$omp parallel !\$omp do do i=1,1000 a(i)=i enddo !\$omp enddo !\$omp end parallel end % f90 -parallel -omp omp.f</pre>	<p>OMP 指示文の例</p> <p>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</p>
--	---

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

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

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

<pre>% f90 -Oss -loglist program.f 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 a.out program.f program.log % cat program.log program main parameter (n=10000) real a(n), b(n), c(n) ** ** Parallel processing starting at loop entry ** Parallel function: _parallel_func_1_MAIN ** Parallel loop ** --- 2 loops (D010,D020) fused (depth 1) --- ** Parallel processing finishing at loop exit ** ** [DO 10] ** Innermost loop unrolled (10 times). ** SWPL applied. ** (省略)</pre>	ログメッセージファイルを出力する program.log が作成される ログメッセージファイル program.log の内容を表示 (並列処理開始) (並列ループ) (DO10, DO20 ループ融合) (最内側ループ展開) (ソフトウェアパイプライン適用)
---	---

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

<pre>% f90 -Oss -loglist program.f -listlang=c % f90 -Oss -loglist program.f -listlang=sjis % f90 -Oss -loglist program.f -listlang=euc</pre>	英語 (ASCII) で出力 日本語 (SJIS) で出力 日本語 (EUC) で出力
<pre>% f90 -Oss -loglist program.f -listlang=sjis f90: compile start : program.f *OFORT90 V01-05-/C 開始 *プログラム名 = MAIN *end of compilation : MAIN *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 program main parameter (n=10000)</pre>	ログメッセージファイルを出力する (オプション -listlang=sjis を指定) コンパイルメッセージが日本語で表示される (端末の文字コードを SJIS に設定すること) ログメッセージファイル program.log が日本語で作成される

```

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

<pre> % f90 -parallel program.f -pmfunc f90: compile start : program.f *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 *program units = 0002, no diagnostics generated. % (バッチジョブで a.out を実行) % ls ... a.out pm_a.out_Oct24_1139_n75_696542 program.f % pmpr pm_a.out_Nov02_1154_n12_659518 pm_a.out_Nov02_1154_n12_659518: (省略) </pre>	<p>性能モニター情報を出力する実行ファイルを作成</p> <p>このファイルはバイナリー形式のため、性能モニター情報出力コマンド pmpr を使用してテキスト形式で表示する。</p> <p>実行ファイル a.out を実行する</p> <p>性能モニター情報ファイルが作成されていることを確認</p> <p>pmpr コマンドで情報をテキスト形式で表示 (pmpr -j ファイル名とすると日本語 SJIS で表示される)</p>
--	---

```
#####
## 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。

-mpar **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 -mpar      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
*program units = 0002, no diagnostics
generated.
%
(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

% pmptr pm_a.out_Nov02_1159_n12_643084      pmptr コマンドで情報をテキスト形式で表示
pm_a.out_Nov02_1159_n12_643084:      (pmptr -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。

pmptr コマンドによる文字コード種別の指定例

```
% pmptr -e      性能モニター情報ファイル名      英語 (ASCII) で出力
% pmptr -j      性能モニター情報ファイル名      日本語 (SJIS) で出力
% pmptr -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
<pre>% f90 -diffcomp program.f f90: compile start : program.f *OFORT90 V01-05-/C entered. *program name = MAIN *program name = SUB *end of compilation : MAIN *end of compilation : SUB *program units = 0002, no diagnostics generated. % ls a.out program.f program.o ~ プログラム SUB を編集 ~ % f90 -diffcomp program.f f90: compile start : program.f *OFORT90 V01-05-/C entered. *program name = MAIN *program name = SUB *end of compilation : SUB *program units = 0002, no diagnostics generated. %</pre>	<p>差分コンパイルオプションを指定する 通常と同様にコンパイルを開始</p> <p>プログラム MAIN コンパイル終了 プログラム SUB コンパイル終了</p> <p>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</p> <p>プログラム SUB の内容を変更後、改めて差分コンパイルする</p> <p>プログラム MAIN に変更がないので構文チェックのみ行う プログラム SUB は変更があったのでコンパイルを行う メッセージ「*end of compilation : MAIN」がないことに注意</p>

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
<pre>% f90 -diffcomp program.f f90: compile start : program.f *OFORT90 V01-05-/C entered. *program name = MAIN *program name = SUB *end of compilation : MAIN *end of compilation : SUB *program units = 0002, no diagnostics generated. % mv program.f program.f.bak % ls a.out program.f.bak program.o % f90 -recomp -parallel program.f f90: compile start : program.f *OFORT90 V01-05-/C entered. *program name = MAIN</pre>	<p>差分コンパイルオプションを指定する 通常と同様にコンパイルを開始</p> <p>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 試しにソースプログラム program.f の名前を変更してみる</p> <p>再コンパイルオプションを指定してコンパイル (参考のため-parallel オプション追加)</p> <p>オブジェクトモジュール内のソースプログラム情報を利用して再コンパイルを行う</p>

```

*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。時間が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でない場合、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、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),EOFRTD,EOFRTDT,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(fpecnt1(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 (REALDT)

実数型データ 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 0.2000000000000000E+00 となる)。(デフォルト)

PORT (TABSP)

入力データに 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。

<pre> CPU 時間計測の例 real t call clock -- プログラム -- call clock(t) write(*,*) t stop end </pre>	<p>変数 t は実数型 4 バイト (単精度) CPU 時間測定タイマー起動</p> <p>タイマー起動後の CPU 時間 (秒) を変数 t に代入</p>
<pre> (高精度タイマー使用の場合) real*8 t call xclock -- プログラム -- call xclock(t) </pre>	<p>変数 t は実数型 8 バイト (倍精度) CPU 時間測定タイマー起動</p> <p>タイマー起動後の CPU 時間 (秒) を変数 t</p>

<code>write(*,*) t</code>	に代入
<code>stop</code>	
<code>end</code>	
経過時間計測の例	
<code>real t</code>	変数 t は実数型 4 バイト (単精度)
<code>call clock(t, 7)</code>	経過時間測定タイマー起動
-- プログラム --	
<code>call clock(t, 8)</code>	タイマー起動後の経過時間 (秒) を変数 t に
<code>write(*,*) t</code>	代入
<code>stop</code>	
<code>end</code>	
(高精度タイマー使用の場合)	
<code>real*8 t</code>	変数 t は実数型 8 バイト (倍精度)
<code>call xclock(t, 7)</code>	経過時間測定タイマー起動
-- プログラム --	
<code>call xclock(t, 8)</code>	タイマー起動後の経過時間 (秒) を変数 t に
<code>write(*,*) t</code>	代入
<code>stop</code>	
<code>end</code>	

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)」を御覧ください。

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, FSEEK064, 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

<code>% f90 program.f -lf90c</code>	サービスサブルーチンを使用 (f90)
<code>% f77 program.f -lf77c</code>	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)」を御覧ください。

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
                                     data.1, data.2 をノード毎に入力し、結
                                     果をそれぞれ result.1, result.2 に出力
                                     する。

% ls
a.f      data.1      job.csh
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對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)	-lmatmps
	(スカラー版)	-lmatmps_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 hsrulm.f -lmatmpp                 ... コンパイル
f90: compile start : hsrulm.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,
& iopt2, x, ier)
    write(6,*) me, n, ier
    do 20 i=1,n
        write(6,*) i,x(i)
20 continue
end

% mpif90 -parallel hdru3mdp.f -lmatmpp
f90: compile start : hdru3mdp.f
... コンパイル
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
#!/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
... ジョブスクリプト例

% 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 ~

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

```

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)	-IMSL2P
	(スカラー版)	-IMSL2

- 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 -IMSL2                               ... 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 -IMSL2P                   ... 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	<code>-lblas</code>	<code>-lblas_sc</code>	
LAPACK	<code>-llapack</code> <code>-lblas</code>	<code>-llapack_sc</code> <code>-lblas_sc</code>	
ScaLAPACK	<code>-lscalapack</code> <code>-lblacsBASE</code> <code>-lblacsF77</code> <code>-lblas</code>	<code>-lscalapack_sc</code> <code>-lblacsBASE_sc</code> <code>-lblacsF77_sc</code> <code>-lblas_sc</code>	(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.oxxxx                               ... 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.oxxxx                                 ... 結果を表示
3.000000000000000000
2.000000000000000000

```

使用例 (ScaLAPACK)

```

% mpif90 -Oss -noperallel 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
f90: compile start : example1.f
                                     サンプルプログラムは
                                     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}
mpirun ./a.out
                                     実行は mpirun コマンドを使用

% 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.oxxxx                                 ... 結果を表示

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

<code>/usr/local/unsupported/atlas/include/</code>	C 言語用ヘッダファイル
<code>/usr/local/unsupported/atlas/lib/</code>	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、必ずセンサーの 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	$\pm 1.175495 \times 10^{-38} \sim$ $\pm 3.402823 \times 10^{38}$
倍精度	$\pm 2.225074 \times 10^{-308} \sim$ $\pm 1.797693 \times 10^{308}$
拡張精度	$\pm 2.225074 \times 10^{-308} \sim$ $\pm 1.797693 \times 10^{308}$

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)
	Fortran 固有ファイル	標準書式なし
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、ファイルポインタがファイル終了記録の直前のデータ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
6.00000000 0.666666687
```

… 端末（キーボード）から入力。
… 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
6.00000000 0.666666687
```

… a.data から入力。

```
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
% cat a.result
6.00000000 0.666666687
```

… a.data から入力， a.result に出力。
… 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作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 FTnnFxxx

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 FTnnFxxx 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、プログ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
#@$-q single
#@$-IT 00:30:00
#@$-IM 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
```

付録 A vi エディターの使い方

1. vi エディターの起動

ファイル名（ここでは" abc "）を指定して vi コマンドを起動します。

```
% vi abc
```

```
█ ← カーソル  
~  
~  
~
```

```
"abc" [New file]
```

vi エディター起動直後は” コマンドモード ” になっています。この状態では文字は入力できません。

2. 文字の入力（カーソルの前に入力）

"i" と打って下さい。何も変化はありませんが、” 入力モード ” になっています。以後キーボードから入力した文字が画面に表示されます。以下のように入力して下さい。

```
vi editor de abc wo input█  
~  
~  
~  
"abc" [New file]
```

3. 入力モードの解除

” コマンドモード ” へ戻るためには ESC キー（Escape キー）を押します。カーソルは行末の文字 t の上に移動し、” コマンドモード ” に戻りました。以下で説明する各コマンドが使えます。

```
vi editor de abc wo input█  
~  
~  
~  
"abc" [New file]
```

4. カーソルの移動

キーの対応	キー	h	j	k	l
	移動	←	↓	↑	→

"h"と打つとカーソルが左へと動きます。abc の a の文字上にカーソルを移動して下さい。

```
vi editor de abc wo input
~
~
~
"abc" [New file]
```

5. 文字の削除, 修正

"x"と打つとカーソル位置の1文字が消えます。3回押してabcを消して下さい。

```
vi editor de wo input
~
~
~
"abc" [New file]
```

再び"i"と打って”入力モード”にします。textと打ってESCキーを押して下さい。

```
vi editor de text wo input
~
~
~
"abc" [New file]
```

6. ファイルへの保存

":"を押すとカーソルが画面左下に移動します。

```
vi editor de text wo input
~
~
~
:
```

"w"と打ちEnterキー (Returnキー)を押します。するとテキスト文字はファイル"abc"に保存されます。

```
vi editor de text wo input
~
~
~
:w
```

7. vi エディターの終了

vi エディターを終了するには": "を押してから" q "Enter です。

```
vi editor de text wo input
~
~
~
:q
```

% █

No write と出て終了できない時は":w " (ファイルへの保存) を忘れています。":wq " Enter と打てばファイルへの保存後, vi エディターを終了します。

```
:w   ファイルへの保存
:q   エディターの終了
:q!  ファイルへの保存をしないでエディターを終了する
:wq  ファイルへの保存, エディターの終了
```

以上でファイル" abc "が作成されました。cat コマンドで確認してみましょう。

8. ファイルの編集

先ほど作成したファイル" abc "を書き換えてみましょう。再びvi エディターを起動します。

```
█vi editor de text wo input
~
~
~
"abc" 1/31
```

9. 文字の入力 (行の追加)

2 行目に行を追加してみましょう。" o "を押して下さい。カーソルが下へ移動し," 入力モード " になります。

```
vi editor de text wo input
█
~
~
"abc" 1/31
```

そのまま文字を入力していきます。2行目を書くことができます。

```
vi editor de text wo input
 2 gyoume mo kaketa█
~
~
"abc" 1/31
```

ここで **Enter** キーを押すと3行目へと移ります。まだ”入力モード”は続いています。

```
vi editor de text wo input
 2 gyoume mo kaketa
 3 gyoume mo kakeru█
~
"abc" 1/31
```

ここで **ESC** キーを押すと”コマンドモード”になります。カーソルを動かすコマンド”h”, ”j”, ”k”, ”l”を使って移動してみましょう。

```
vi editor de text wo input
 2 gyoume mo kaketa█a
 3 gyoume mo kakeru
~
"abc" 1/31
```

10. 文字の入力（カーソルの後に入力）

2行目の末尾に文字を追加してみましょう。行末の文字 **a** の上にカーソルを移動し”**a**”と打つと”入力モード”になるので **yo** とでも打ってみてください。

```
vi editor de text wo input
 2 gyoume mo kaketayo█
 3 gyoume mo kakeru
~
"abc" 1/31
```

なぜ”**a**”を使うのでしょうか。カーソルの前に文字を入力する”**i**”では **a** の文字の前から書き込まれるので次のようになってしまうのです。

```
vi editor de text wo input
 2 gyoume mo kaketyoa█
 3 gyoume mo kakeru
~

"abc" 1/31
```

主な入力コマンドには次のようなものがあります。

- a カーソルの右側に文字を入力する
- i カーソルの左側に文字を入力する
- o カーソルのある行の下に文字を入力する。
- O カーソルのある行の上に文字を入力する

11. 行の削除

3行目の適当なところで" dd "と打つとその行が消えます。

```
vi editor de text wo input
 2 gyoume mo kaketayo
 3 gyoume█mo kakeru
~

"abc" 1/31
```

" dd "と打つと

```
vi editor de text wo input
 2 gyoume mo kaketayo█
~
~

"abc" 1/31
```

12. 変更の取消

誤って行を削除するなどした時は" u "コマンドで最後に行われた変更を元に戻します。

```
vi editor de text wo input
 2 gyoume mo kaketayo
 █3 gyoume mo kakeru
~

"abc" 1/31
```

ファイルに保存して vi エディターを終了しましょう。(":wq " Enter)

主な vi エディターのコマンド

画面制御 コマンド	Ctrl-l	現在の画面を再表示する。
	Ctrl-y	画面を 1 行上にスクロールする。
	Ctrl-e	画面を 1 行下にスクロールする。
ページ コマンド	Ctrl-f	次の 1 画面を表示する。
	Ctrl-b	前の 1 画面を表示する。
	Ctrl-d	半画面下にスクロールする。
	Ctrl-u	半画面上にスクロールする。
カーソル移動 コマンド	j	カーソルを同じ桁で 1 行下に移動する。
	k	カーソルを同じ桁で 1 行上に移動する。
	h	カーソルを 1 桁左に移動する。
	l	カーソルを 1 桁右に移動する。
	Return	カーソルを次の行の先頭に移動する。
	0 (ゼロ)	カーソルを現在行の先頭に移動する。
	\$	カーソルを現在行の最後に移動する。
	スペース	カーソルを 1 文字進める。
	nG	カーソルを n 行目の先頭に移動する。n 省略の時、ファイルの最後の行。
	/pattern	次の pattern が現れるところにカーソルを進める。.
	?pattern	前の pattern が現れたところまでカーソルを戻す。
	n	最後に行なったパターン検索 / または ? を繰り返す。
テキスト入力 コマンド	a	カーソルの右側にテキストを入力する。ESC で入力を終了する。
	A	カーソルのある行の行末にテキストを入力する。ESC で入力を終了する。
	i	カーソルの左側にテキストを入力する。ESC で入力を終了する。
	I	カーソルのある行の行頭にテキストを入力する。ESC で入力を終了する。
	o	カーソルのある行の下にテキストを入力する。ESC で入力を終了する。
	O	カーソルのある行の上にテキストを入力する。ESC で入力を終了する。
	Backspace	テキスト入力中、最後に入力した文字を削除する。
	ESC	テキスト入力を終了する。
テキスト削除 コマンド	dw	カーソルのある文字からその語 (ワード) の終わりまでを削除する。
	x	カーソルのある文字を削除する。
	dd	カーソルのある行を削除する。
	D, d\$	カーソルのある文字から行の終わりまでを削除する。
テキスト変更 コマンド	cw	カーソルのある文字からその語の終わりまでを削除する。
	C, c\$	カーソルのある文字から行の終わりまでを、文字列で置き換える。
	J	現在行と次の行をつなげる。
	r x	カーソルのある文字を別の文字 x に置き換える。
バッファ編集 コマンド	nY	現在行から n 行をヤंकする。n が省略されると、現在行をヤंकする。
	p	ヤंकしたテキストをカーソルの後に挿入する。
	P	ヤंकしたテキストをカーソルの前に挿入する。
ファイル編集 コマンド	:w name	ファイルをファイル名 name で保存する。
	:r name	ファイル名 name のファイルを読み込む。
vi の終了	:wq, ZZ	ファイルを保存して、エディターを終了する。
	:q, :q!	保存せずに、エディターを終了する。

付録B 高速ファイル転送ソフト SRFT について

1 SRFT とは

本センターに導入されている SR11000 では、ファイルシステムとして主に HSFS が使用されています。このファイルシステムは最適化されたプログラムが大量のデータを扱うときには高い性能を発揮しますが、HSFS の特性を考慮していない一般のプログラムでは本来の性能は得られません。ファイル転送に使用する SCP は HSFS の特性を考慮していないプログラムであるため、ファイルアクセスの性能が悪く、大容量ファイルの転送に長い時間を要します。この問題を解決するために情報基盤センタースーパーコンピューティング部門の松葉が HSFS に最適化したファイル転送ソフト SRFT を作成しました。実験環境において SRFT は SCP の 26 倍である 70MB/s を達成しています。

2 インストール

最初にお使いになる前に、本付録の最後にある注意事項をよくお読みください。SRFT は SR11000 と手元のクライアント双方にインストール作業が必要です。

SR11000

SR11000 には専用のバイナリを準備しました。コンパイル作業は不要です。

1. <http://www.il.cc.u-tokyo.ac.jp/~matsuba/srft/srft-1.1.0-sr11000-bin.tar.gz> をダウンロードする
2. 上記ファイルを scp など通常使用している方法で SR11000 にコピーする
3. `$ gzip -dc srft-1.1.0-sr11000-bin.tar.gz | tar xvf -` で解凍する。./local/bin の中に必要なファイルが展開される。
4. 展開されたファイルが存在するディレクトリに PATH を通す（必須）。手順 3 の作業をホームディレクトリで行った場合は、~/.cshrc に以下のように書く。
`set path=(/batch/xxxxx/local/bin $path)`

ソースからのコンパイルも可能ですが、そのためには OpenSSL もコンパイルしてインストールする必要があります。

手元のマシン

対応 OS は Linux, FreeBSD, MacOS X です。Windows は Cygwin も含め非対応です。また、OpenSSL の開発環境が必要です。通常 openssl-devel のようなパッケージ名で提供されているはずです。

1. `$ wget http://www.il.cc.u-tokyo.ac.jp/~matsuba/srft/srft-1.1.0.tar.gz`
2. `$ tar zxvf srft-1.1.0.tar.gz`
3. `$./configure --prefix=$HOME/local`
OpenSSL を特殊なディレクトリにインストールした場合は、`./configure` のオプションに `--with-openssl-prefix=<openssl の path>` を加えてください。
4. `$ make`
5. `$ make install`
6. インストール先に PATH を通す。tsch のときは上記 SR11000 と同じ方法、bash の時は `~/.bashrc` に以下のように書く。
`export PATH= /local/bin:$PATH`

3 使いかた

なるべく scp に近い方法で使えるように作りました。

- ・ 単一ファイルの転送
`$ srft ~/file.dat xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:`
- ・ 複数ファイルの転送
`$ srft ~/file.dat ~/somefile*.dat xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:`
- ・ SR11000 からのダウンロード(1)
`$ srft xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:file.dat .`
- ・ SR11000 からのダウンロード(2)
`$ srft xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:file*.dat .`
- ・ ディレクトリごと転送(1)
`$ srft -r ~/dir1 xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:somedir/`
- ・ ディレクトリごと転送(2)
`$ srft -r xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:somedir/ .`

4 起こりがちなエラー

- ・ `srft server: Command not found.`
リモートホストで PATH を通す作業ができていません。インストール方法に書いてある方法で PATH を通してください。
- ・ `Error: reading public key: xxxxx`
認証に使っている鍵が見つかりません。`rm -r ~/.srft` を手元のクライアントと SR11000 の両方で実行してから再度試してみてください。

5 性能

以下に SRFT の実行結果と、SCP の実行結果を載せます。x1 x2 は 500MB のファイルです。実験に使用したクライアントは Intel Xeon Processor 5140 x2 のマシンに Hitachi SANRISE AMS200 を接続したコンピュータです。また、ネットワーク的には SR11000 と近く Gigabit Ethernet の帯域が十分に発揮できる環境です。この実験環境では SCP の 2.7MB/s に対し、SRFT は 26 倍の 70.4MB/s を達成しています。ディスク、ネットワーク共に理想的な環境での測定なので、他の環境ではここに示すほどの性能は出ないことが予想されます。

SRFT

```
[matsuba@fs ~]$ srft x1 x2 xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:
xxxxxx@sr11000-s.cc.u-tokyo.ac.jp' s password:
x1          500.0MB/500.0MB (100.0%) 72.4MB/s 0h00m07s
x2          500.0MB/500.0MB (100.0%) 73.3MB/s 0h00m07s
Total 1.0GB/ 1.0GB (100.0%) 70.4MB/s 0h00m14s
```

Total 性能が各ファイルの転送性能の平均値を下回るのは、Total にはどのファイルの転送にも属さない制御情報の交換時間が含まれるためです。

SCP

```
[matsuba@fs ~]$ scp x1 x2 xxxxxx@sr11000-s.cc.u-tokyo.ac.jp:
xxxxxx@sr11000-s.cc.u-tokyo.ac.jp' s password:
x1          100% 477MB 2.7MB/s 02:59
x2          100% 477MB 2.7MB/s 02:59
```

6 技術的情報

6.1 安全性

転送経路中、データは RC4 と呼ばれる簡単な方式で暗号化されています。途中でパケットを盗聴されてもすぐに内容がわかることはありません。ただし、暗号強度はそれほど強くないので、何者かが悪意を持って解析すれば内容が判明してしまうかもしれません。一方、通信相手の正しさ（本当に SR11000 と通信しているかどうか）に関しては、最高で SSH と同等の安全性ということになります。これは SSH を用いて証明書の交換を行っているためです。ネットワーク上での改竄は検出できるはずですが。

6.2 高速な理由

HSFS で高速にファイルアクセスを行うためには、read または write システムコールに渡すバッファのサイズを 16MB 程度にする必要があります[1]。一方 Linux などで使用されている PC 向けファイルシステムでは 4KB 程度のバッファで十分な性能が出るので、SCP を含めほとんどのアプリケーションは 4KB 程度の単位でシステムコールを行います。そのため、これらの一般的なアプリケーションを HSFS で使用しても十分な性能が得られないこととなります。SRFT は 16MB 単位でファイル入出力を行うよう、送受信データをバッファリングしています。このバッファサイズの最適化の他、ファイルアクセス待ちの時間を暗号化処理に活用するなどの工夫を行い、上記の性能を実現しています。

7 Q & A

- ・ 80MB/s に程遠い性能しか出ません
途中で 100Mbps のネットワークが挟まっていないことを確認して下さい。また、上述のように SRFT では 16MB 単位でアクセスを行うことで高速化しているので、16MB に満たない小さなファイルが大量にある場合は SCP と同等の性能しか出ません。
- ・ うまく動かない／ファイルが壊れる
まずは表示をよく見て” Permission denied” や” No space left on device” などの分かりやすいエラーが出ていないことを確認してください。バグが疑われる時は状況を作者までご連絡ください。

8 注意事項

以下の事項を了解の上ご使用下さい。

- ・ 本ソフトを使用したことによるいかなる結果に対しても、作者の松葉および情報基盤センターは一切の責任を負いません。
- ・ 本ソフトは情報基盤センターによるスーパーコンピュータサービスの一部ではありません。情報基盤センターとしては本ソフトに関する質問はお受けできません。質問、要望などはシステムの相談窓口でなく、直接松葉までお願いいたします。ただし、作者もサポート義務は負いません。
- ・ 本ソフトのライセンスは GPLv2 で OpenSSL とのリンクを認めます。

9 最新情報および連絡先

本ソフトの URL: <http://www.il.cc.u-tokyo.ac.jp/~matsuba/srft/srft.html>

参考文献

- [1] 黒田久泰” C 言語におけるファイル入出力の高速化” , スーパーコンピューティング
ニュース Vol.8 No.5 pp.28-33, 2006 年 9 月

付録C 複数プログラムの実行方法と環境変数について

2007年4月より SR11000/J2 の並列用ジョブキューの大部分が1ノード16CPUとして運用されています。現在、1ノード16CPUのキューと1ノード8CPUのキューは下記のようになっています。

1ノード 16 CPU のキュー : parallel(P001~P064), debug, Q001, Q004, Q008

1ノード 8 CPU のキュー : S1, S4, H1

1ノード16CPUのキューでは1ノード8CPUのキューに比べて2倍のCPUが利用できますが、16CPUが有効に利用されるような並列化が行われていなければ、CPUリソースの無駄使いとなってしまいます。このような場合、無理に16スレッド並列で実行させるよりも、低めのスレッド数に設定して、入力データが異なるような複数のプログラムを同時に走らせる方が効率が良い場合もあります。ここではこのような場合の使い方を紹介します。

1. 16個の逐次プログラムを実行する場合

並列化をしない逐次プログラムの場合、1ノード16CPUのキューでは同時に16個までほとんど速度低下なく実行することができます。prog1からprog16までの16個のプログラムを走らせる場合には、次のようなNQSスクリプトになります。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
./prog1 &
./prog2 &
. . . . .
./prog15 &
./prog16
wait
```

最後の wait コマンドはバックグラウンドで実行されている全てのプロセスが終了するまで待機するコマンドです。この wait が無ければ、prog1 から prog15 が実行中であっても prog16 が終了したらジョブの実行は終了してしまいます。wait の代わりに sleep コマンドを用いて一定時間待つという方法もありますが、wait コマンドを使えば、全てのプログラムが終了した時点でジョブを終了できるので無駄がありません。

なお、1ノードのCPUの数よりも多くのプログラムを同時に実行させると、効率が悪くなります。そのため、1ノード8CPUのキューを利用する場合、同時に走らせるプログラムは最大8個にしましょう。

この方法は並列実行コマンド(prun コマンド)を利用して実現できます。prun コマンドの定義ファイルを用いることで、さまざまな方法でプログラムを並列実行させることが可能となります。ただし、1 ノード上で複数のプログラムを実行する場合には、NQS スクリプト中に「#@\$-J T16」あるいは「#@\$-J SS」の指定が必要となりますので注意してください。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
#@$-J T16
prun -f sample.def
```

prun 用定義ファイル(sample.def)

```
*16 ./prog%n
```

prun コマンドは本センターSR11000/J2 上において独自に提供されているコマンドです。利用方法の詳細については、オンラインマニュアル(「% man prun」を実行)、あるいは、「11.1 並列実行(prun コマンド)」をご覧ください。

2. 4 個の要素並列化プログラムを実行する場合

要素並列化プログラムの並列実行スレッド数は、特に指定しなければ実行するキューにおける 1 ノードの CPU の数と同じになります。しかし、高い並列性がないプログラムでは、並列実行スレッド数を上げててもそれに比例した速度向上は期待できません。この場合、並列スレッド数を低く設定して、複数のプログラムを同時に実行するといった方法も考えられます。4 個の要素並列化プログラムをそれぞれ並列実行スレッド数 4 で実行する場合の NQS スクリプトは下記のようになります。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
setenv HF_PRUNST_THREADNUM 4
./prog1 &
./prog2 &
./prog3 &
./prog4
wait
```

環境変数 HF_PRUNST_THREADNUM に各プログラムで利用する並列実行スレッド数(1~16)を指定します。実行するプログラム毎にこの環境変数の値を変更することもできます。また、この環境変数は最適化 FORTRAN77/90、最適化 C、最適化標準 C++ で共通です。

なお、並列実行コマンド(prun コマンド)を利用して実現できます。この場合、NQS スクリプトと prun 用定義ファイルは下記のようになります。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
#@$-J T4
setenv HF_PRUNST_THREADNUM 4
prun -f sample.def
```

prun 用定義ファイル(sample.def)

```
*4 ./prog%n
```

3. 4 個の OpenMP 並列プログラムを実行する場合

4 個の OpenMP 並列プログラムをそれぞれ並列実行スレッド数 4 で実行する場合の NQS スクリプトは下記のようになります。

NQS スクリプト

```
##$-q parallel
##$-N 1
setenv OMP_NUM_THREADS 4
./prog1 &
./prog2 &
./prog3 &
./prog4
wait
```

環境変数 OMP_NUM_THREADS には OpenMP で利用する並列実行スレッド数を設定します。もし、環境変数 HF_PRUNST_THREADNUM が設定されている場合、環境変数 OMP_NUM_THREADS の値は、環境変数 HF_PRUNST_THREADNUM の値以下でなくてはなりません。環境変数 OMP_NUM_THREADS が設定されていない場合には、環境変数 HF_PRUNST_THREADNUM の値とみなされます。

なお、並列実行コマンド(prun コマンド)を利用しても実現できます。この場合、NQS スクリプトと prun 用定義ファイルは下記のようになります。

NQS スクリプト

```
##$-q parallel
##$-N 1
##$-J T4
setenv OMP_NUM_THREADNUM 4
prun -f sample.def
```

prun 用定義ファイル(sample.def)

```
*4 ./prog%n
```

4. 複数ノードで複数の並列化されたプログラムを実行する場合

例えば、4 ノードを用いて 8 個の並列化されたプログラムを実行する場合、1 ノード 16CPU のキューであれば 1 つの並列プログラムに 8CPU を割り当てることができます。このような場合には、並列実行コマンド(prun コマンド)を利用します。4 ノードで 8 つのプログラム prog1、prog2、・・・、prog8 を実行する場合の NQS スクリプトと prun 用定義ファイルは下記のようになります。

NQS スクリプト

```
##$-q parallel
##$-N 4
##$-J T2
setenv HF_PRUNST_THREADNUM 8
prun -f sample.def
```

prun 用定義ファイル(sample.def)

```
*4 ./prog%n
```

NQS スクリプト中「##\$-J T2」の部分は、1 ノードあたり 2 つのプログラムを実行することを指定しています。「##\$-J T4」のように間違えて設定してしまうと、1 ノードに 4 つのプログラムが実行され、2 ノードだけで 8 つのプログラムが実行されてしまうことになるので、注意が必要です。

5. MPI プログラムで異なる実行モジュールを起動する方法

MPI プログラムでは、各プロセスのランク番号毎に異なる実行モジュールを起動させることもできます。各システム固有の方法がありますが、ここでは少し汎用的な方法を紹介します。

1 ノード 4MPI プロセスを利用する場合には、下記のような NQS スクリプトを記述します。「#@\$-JT4」のところで mpirun コマンドで起動するプロセス数を指定します。1 ノード 16CPU のキューで 4MPI プロセスを実行する場合、各プロセスは 4CPU まで占有できます。そこで、環境変数 HF_PRUNST_THREADNUM には 4 を設定します。この環境変数の設定を行わないと 16 が仮定されますので、注意が必要です。また、OpenMP 並列プログラムでは、環境変数 OMP_NUM_THREADS を 4 に設定しても構いません。ここでは、mpirun の引数に ./exec.sh を与えて、exec.sh を 4 つ実行するようにします。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
#@$-J T4
setenv HF_PRUNST_THREADNUM 4
mpirun ./exec.sh
```

exec.sh の中身

```
#!/bin/sh
if [ "$MP_CHILD" == "0" ]; then
    exec ./mpi_prog1
fi
if [ "$MP_CHILD" == "1" ]; then
    exec ./mpi_prog2
fi
if [ "$MP_CHILD" == "2" ]; then
    exec ./mpi_prog3
fi
if [ "$MP_CHILD" == "3" ]; then
    exec ./mpi_prog4
fi
```

SR11000/J2 では MPI プロセスを起動する際、環境変数 MP_CHILD には各プロセスのランク番号が設定されます。そこで、環境変数 MP_CHILD の値を見て、どのプログラムを実行するかを決定しています。この例では、ランク番号の小さい順に、mpi_prog1、mpi_prog2、mpi_prog3、mpi_prog4 を実行しています。この環境変数 MP_CHILD の名前は MPI の実行環境によって異なりますので、他のシステムで利用する際には変更が必要になります。

なお、並列実行コマンド(prun コマンド)を利用すると、もっと簡単になります。上記と同じように 1 ノード 4MPI プロセスで、ランク番号毎に異なる実行モジュールを起動させたい場合には、NQS スクリプトと prun 用定義ファイルは下記のようになります。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
#@$-J T4
setenv HF_PRUNST_THREADNUM 4
prun -f sample.def
```

prun 用定義ファイル(sample.def)

```
*4 ./mpi_prog%n
```

4 ノードで 4MPI プロセスを実行したい場合には、NQS スクリプト中の「#@\$-N 1」を「#@\$-N 4」に変更し、「#@\$-J T4」と「setenv HF_PRUNST_THREADNUM 4」は削除してください。

6. スレッドを CPU に固定的に割り付ける方法

環境変数 HF_PRUNST_BIND の設定で、スレッドを CPU に固定的に割り付けるかどうかを指定します。固定的に割り付けた場合、各スレッドは常に割り付けられた番号の CPU 上で実行されます。

```
setenv HF_PRUNST_BIND 0   . . .   スレッドを CPU に固定的に割り付けない
setenv HF_PRUNST_BIND 1   . . .   スレッドを CPU に固定的に割り付ける
```

この環境変数を設定しない場合には、0 を設定したとみなされます。この環境変数が有効となるのは、コンパイル時に `-parallel` オプションで要素並列化されたプログラム、および、`-parallel -omp` オプションで OpenMP 並列化が行われたものに限りです。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
setenv HF_PRUNST_BIND 1
./prog1
```

この例では、要素並列化あるいは OpenMP 並列プログラムである prog1 の各並列実行スレッドをそれぞれ特定の CPU で実行されるよう固定的に割り付けます。

ただし、この環境変数の設定による速度向上は実行するプログラムによって大きく異なり、速度向上がほとんどない場合もあります。また、独自にマルチスレッドプログラミングを行ったプログラムの場合は、環境変数 HF_PRUNST_BIND の設定では固定的に割り付けることはできません。プログラム中で `bindprocessor` システムコールを用いてください。また、一般のプロセスを固定的に割り付ける場合には `bindprocessor` コマンドを用いてください。これらの詳細はオンラインマニュアル(「% man bindprocessor」や「% man 2 bindprocessor」を実行)などをご確認ください。

7. スレッドの待機方法の指定

環境変数 HF_PRUNST_WAIT の設定で、スレッドの待機方法を指定することができます。

```
setenv HF_PRUNST_WAIT BUSY   . . .   待機スレッドはスピンドルを実行して待機する
setenv HF_PRUNST_WAIT SLEEP . . .   待機スレッドはスリープして待機する
```

この環境変数を指定しない場合には、BUSY を設定したとみなされます。スピンドルで待機している場合、次の仕事が来た場合に即座に開始できます。しかし、スピンドルをしている間は余計な CPU 時間を使っていることとなります。1 ノード 16CPU のキューで、各プロセスの並列実行スレッドの合計数が 16 以下であれば問題ありませんが、16 を超える可能性がある場合には、必ず「setenv HF_PRUNST_WAIT SLEEP」の設定をしておきましょう。

NQS スクリプト

```
#@$-q parallel
#@$-N 1
setenv HF_PRUNST_WAIT SLEEP
setenv OMP_NUM_THREADS 16
./prog1 &
./prog2
wait
```

OpenMP 並列プログラム prog1 と prog2 を同時に実行する場合の例です。要素並列化プログラムの場合には、「setenv OMP_NUM_THREADS 16」の代わりに「setenv HF_PRUNST_THREADNUM 16」とします。

付録D Gaussian03 の使い方

一般的非経験分子軌道計算プログラム Gaussian03 をスーパーコンピューター SR11000 上のプログラムライブラリーとして 2005 年 3 月 25 日に公開しました。Gaussian03 を実行するためのジョブの作成と使用方法について簡単に説明します。但し、Gaussian03 自体の内容や計算方法、入力ファイルの書き方に関しては触れませんので詳細は以下に示す文献を参照して下さい。

本センターにおける Gaussian03 のバージョンは次の通りです。(2009 年 3 月 1 日現在)

Gaussian03 Revision D.02

本ライブラリーは 1 ノードによる逐次実行版（並列版ではありません）で、プログラムは要素並列化されており、32bit モードで動作します。また、ソースプログラムは非公開です。

参考文献

- [1] Gaussian03 User's Reference (Manual Version: 7.0 Mar.2003)
- [2] Gaussian03 Programmer's Reference (Manual Version: 7.0 Mar.2003)
- [3] Gaussian03 IOps Reference (Manual Version: 7.0 Feb.2003)
- [4] 電子構造論による化学の探求 第二版
James B.Foressman, AEleen Frisch 共著 田崎健三 訳

Gaussian Inc. (ガウシアン社) から上記のマニュアルと解説書が出版されています。[1], [2], [3]は英文です。[2]はソースプログラムの解説書なので一般利用者には必要ないでしょう。[4]は入門用解説書です。これらはシステム運用係で管理しており、本センター内にて閲覧可能です。閲覧希望の方は本センター1F 受付までお申し出下さい。

また、以下の Web ページから Gaussian03 に関する様々な情報を得ることができます。

<http://www.gaussian.com/>

なお、Gaussian03 を使用して得られた計算結果や成果を公表する場合には、以下の Web ページに従って文献の引用を行うようにして下さい。

<http://www.gaussian.com/citation.htm>

1. スクリプトファイルの作成

Gaussian03 はバッチジョブ (NQS) で実行する必要があります。バッチジョブで実行するには環境変数の設定や入出力ファイルを記述するスクリプトファイルを作成する必要があります。以下にスクリプトファイルの例を挙げ説明します。

```
#!/bin/csh          ... (1) シェルの宣言
#@$-q parallel     ... (2) キューの選択
#@$-N 1           ... (3) ノード数
#@$-IT 00:30:00    ... (4) 経過制限時間 (ここでは 30 分)
#@$-ls 1536MB      ... (5) スタックサイズ
source /usr/local/g03/bsd/g03.login ... (6) 環境設定
cd gaussian/test   ... (7) 入力ファイルの場所
g03 < test178.com > test178.log    ... (8) プログラムの起動
```

- (1) スクリプトファイルを C シェル「/bin/csh」で記述することを宣言します。(本スクリプトでは source 等 C シェルのコマンドを使用しています。)
- (2) プログラムが要素並列化されているため、ノード専有可能なキューを指定します。P001 の場合は「parallel」を、S1・H1 の場合はそのまま「S1」または「H1」を指定します。(参考 表 1. ジョブクラス制限値 (抜粋))
- (3) 並列版ではないためノード数は「1」ノード固定です。
- (4) ジョブの実行時間を制限します。通常は省略可能です。(計画停止時刻 (qstat コマンドで確認) が迫っているときはこれを残り時間以内に設定しないとジョブが実行しません。)
- (5) スタックサイズの標準は 1GB, 最大は 2GB です。標準で足りない場合は 2GB 以内の値を指定してください。
- (6) コマンドパスなど各種環境変数を設定します。
- (7) 入力ファイルのある場所までディレクトリを移動します。
- (8) プログラム g03 を起動します。入力ファイル及び出力ファイルを以下のようにリダイレクションで指定することができます。

g03 < 入力ファイル > 出力ファイル

「> 出力ファイル」を省略すると実行結果はジョブの標準出力ファイルに出力します。

表 1. ジョブクラス制限値 (抜粋)

キュー名	C P U 数	ノ ー ド 数	制 限 時 間 (経 過 時 間)	メモリー 容量 (GB)	パーソナル				グループ		研究室
					1	2	3	4	1	2	
S1	8	1	2 時間	56(56)	○	○	○	○	○	○	○
H1	〃	1	24 時間	〃	×	○	○	○	○	○	×
(parallel)											
P001	16	1	24 時間	112(112)	×	○	○	○	○	○	×

2. プログラムの実行

スクリプトファイルが完成したらファイル（以下の例では `job.csh`）に保存し、バッチシステムにジョブを投入します。ジョブの投入は以下のように `qsub` コマンドで行います。実行状況は `qstat` コマンドで確認して下さい。

```
% qsub job.csh
Request 427.n121 submitted to queue: P001.
% qstat
2007/04/03 (Tue) 10:34:31: REQUESTS on SR11000
NQS schedule stop time : 2007/04/29 (Sun) 8:30:00 (Remain: 621h 55m 29s)
  REQUEST      NAME      OWNER   QUEUE  PRI NICE   CPU  MEM  STATE
  427.n121     job.csh  p07000  P001   63  0   unlimit 50GB  RUNNING
```

1. のスクリプトファイルの場合、ジョブが終了すると以下のファイルが作成されます。エラーメッセージが出ていないか各ファイルを確認して下さい。

```
job.csh.o427   ... ジョブの標準出力
job.csh.e427   ... ジョブの標準エラー出力（空の場合は作成されません）
test178.log    ... Gaussian03 の出力ファイル
```

エラーがなければ出力ファイルに計算結果が出力され、標準出力のファイルは空になります。

ジョブをキャンセルする場合には以下のようにします。

```
% qdel 427
deleting request 427.n121.
```

3. 実行時のエラー

Gaussian03 実行後は、出力ファイル（Gaussian03 の出力ファイル、ジョブの標準・エラー出力ファイル）のメッセージを確認して下さい。なお、問題解決には、入力ファイル中のルートセクションの記述を `#`（標準）や `#T`（簡略化した出力）ではなく `#P`（詳細の出力）としてみることも必要です。

4. ファイル属性

Gaussian03 で作成されるファイルは、センターでの運用上の都合上、グループ属性が "gaussian" となりますので、御了承下さい。

5. テスト用入力ファイル

Gaussian03 が用意しているテスト用の入力ファイルが以下にあります。参考文献[1]の「Chapter 6 Running Gaussian Test Jobs」にこれに関する記述があります。

`/usr/local/g03/tests/com/testxxx.com` (xxx は 000~718)

付録E 超並列ジョブ (64 ノード) 実行サービスについて

SR11000 における超並列ジョブ実行サービスを実施しています。通常運用構成における並列ジョブは最大 16 ノード (16×16=256 プロセッサ) ですが、本サービスでは 64 ノード (16×64=1024 プロセッサ) 使用可能なバッチジョブ環境を提供します。超並列ジョブクラス P064 (キューP064) には、以下のシステム資源を割り当てます。

	超並列ジョブクラス	通常のジョブクラス
バッチキュー名	P064	P001 ~ P016
ノード数 (プロセッサ数)	64 (1024)	1 ~ 16 (16 ~ 256)
理論演算性能	約 9420 GFLOPS	約 147 ~ 2355 GFLOPS
主記憶容量*	8192 GB	128 ~ 2048 GB
実行時間制限 (ETIME)	2 時間	24 ~ 6 時間
ジョブの投入方法	#@\$-q parallel #@\$-N 64	#@\$-q parallel #@\$-N 16 (P016 の場合)

*ユーザープログラムで使用できるメモリーの大きさはノード当たり 112GB です。

P064 キューの利用は、パーソナルコース 4 とグループコース 2 の利用者とします。

本サービスは原則として

毎月 1 回、最終金曜日午後 (休日の場合は前日) ~ 翌月曜日朝 (同翌日)

に実施します。但し、メンテナンス作業等により日程やサービス時間が変更となる場合がありますので詳細はログインメッセージにて御確認下さい。なお、実施期間中はシステム構成が変更となるため、一部のサービスが休止となります。

利用可能なサービス

インタラクティブ処理用システム	sr11000-s.cc.u-tokyo.ac.jp
スカラージョブクラス	A~D
並列ジョブクラス (8 CPU/ノード)	S1, S4, H1
並列ジョブクラス (16 CPU/ノード)	P001, P004, P064, debug

休止するサービス

並列ジョブクラス (16 CPU/ノード)	P008, P016
計算リソース追加オプション用	Q001, Q004, Q008

なお、サービス内容は今後変更となる場合がありますので、御利用の際は Web ページ <http://www.cc.u-tokyo.ac.jp/> , スーパーコンピューティングニュースまたは show-info コマンドにて御確認下さい。

付録F ジョブスケジューリングシステムについて

本センターでは利用者の皆様に有効かつ公平な計算機資源の提供を行うため、スーパーコンピュータSR11000のバッチジョブの実行において、ジョブスケジューリングシステムを導入しています。詳細は以下のとおりです。

本ジョブスケジューリングシステムでは、4月から現在までに実行済みのバッチジョブにて使用したCPUの使用量（以下、CPU使用量とする）と、投入したジョブに設定した実行予定時間より算出されるCPUの予定使用量（以下CPU予定使用量とする）から、各ジョブの実行優先度を決定します。

- 各キュー毎に、実行待ちのジョブにそれぞれ優先順位（63が最優先で0まで）を付け、順位の高いジョブから順に実行します。
- 順位付けの要素は、4月から現在までに実行したバッチジョブのCPU使用量を利用者毎に積算した値と、投入したバッチジョブで要求しているCPU予定使用量を合計したものであり、その値が少ないジョブから順位付けを行います。
- CPU（予定）使用量の算出は以下のとおり行います。CPU使用量については利用者毎に積算します。

バッチジョブを実行するキュー	CPU（予定）使用量の算出方法
スカラーキュー（A～D）	ジョブのCPU時間*1
パラレルキュー（上記以外）	ジョブの実行時間*1 × CPU数*2 × ノード数

*1 CPU 予定使用量を算出する場合は“実行予定時間”に置き換えてください。実行予定時間はジョブスクリプトに記述した値（#@\$\$-IT）を使用します。

なお、省略した場合はパラレルキューではバッチキューの制限時間、スカラーキューでは10分が実行予定時間となります。

*2 CPU 数は16または8です。キューにより異なります。

例) P004 キューで1時間（3,600秒）ジョブを実行した場合のCPU使用量の算出
 $3,600 \text{ (秒)} \times 16 \text{ (CPU)} \times 4 \text{ (ノード)} = 230,400$

- いずれかのジョブが終了する毎に当該利用者のCPU使用量を積算し、順位の付け替えを行います。大規模計算を行う利用者が不利とならないよう、一定時間毎にCPU使用量に一定の逓減率を掛け、逓減する仕組みとしています。
- キューに並んでいる際の待ち時間は、順位付けの要素に含みません。
- ジョブの実行は2つまで同時に可能です。ただし同一のキューでは1つとなります。
- ジョブの優先順位はqstatコマンドの“PRI”の値で確認が可能です。

```
% qstat
REQUEST      NAME      OWNER      QUEUE  PRI  NICE  CPU    MEM    STATE
99999.n121   job.csh   p07000     P001   62   0    unlimit 112GB  QUEUED
```

→ PRI=62なので順位は2番目

付録G 計算リソース追加オプション

追加オプションにより Q001(48 時間), Q004(24 時間), Q008(24 時間) キューが利用できるサービスを実施しています。制限時間が一般のキューP001(24 時間), P004(12 時間), P008(8 時間) より長く設定されています。また, 人数制限されているためジョブが実行しやすくなっています。

① 利用見込み額

- ・利用申し込みのためには, 利用見込み額の設定が必要です。
- ・新規および継続利用申込書には, 利用見込み額の記入欄があります。
- ・利用見込み額の追加は, Web ページまたは専用申込書 (書面) により行えます。
- ・利用見込み額として記入された金額が課金されるのではなく, ②に示す Web による利用申し込みをした時点で課金対象となります。
- ・利用見込み額を超えた利用申し込みはできませんが, 利用見込み額を追加することで利用申し込みが可能になります。

② 利用申し込み

- ・利用申し込みは, 本センタースーパーコンピューティング部門の Web ページ (<http://www.cc.u-tokyo.ac.jp/>) から行います。
- ・利用申し込みは, 月単位とします。3 ヶ月先まで予約可能とします。
- ・それぞれのキューについて各月の上限を10名とします。(複数のキューを利用したい場合はそれぞれ申し込みが必要です。Q008を申し込みればQ001, Q004も利用できるわけではありません。)

③ ジョブの実行

- ・NQS でのキューの指定は “#@\$-q Q001” のようにキュー名を指定してください。(キュー名は, 大文字・小文字を区別しますので, ご注意ください。)
- ・1 利用者番号あたり投入可能なジョブは 1 とします。
- ・ジョブの実行はFIFO(First In First Out)とします。つまり, 待ちキューに並んでいる順番に実行されます。
- ・ジョブスケジューリングシステムにおけるCPU使用量の積算は行いません。(詳しくは「付録F ジョブスケジューリングシステムについて」をご覧ください。)
- ・超並列ジョブP064キューサービス時 (原則として最終金曜日午後～翌月曜日朝) はご利用いただけません。

注: 「月」の扱いについて

月末処理日 (原則として毎月最終金曜日) から翌月の月末処理日までを 1 ヶ月として扱います。例えば, 2008 年は 5 月 30 日, 6 月 27 日が月末処理日ですので, 「6 月」の申し込みをした場合, 利用は 6 月 2 日 9:30 から 6 月 27 日 8:30 までになります。

利用負担金表（抜粋）は以下のとおりです。（表1）

表1. 利用負担金表（抜粋）

キュー	負担金額（税込）	備考
1ノード専用キュー	20,000 円/月	パーソナル2・3・4, グループ1・2 利用者が申込可
4ノード専用キュー	50,000 円/月	パーソナル3・4, グループ1・2 利用者が申込可
8ノード専用キュー	80,000 円/月	パーソナル4, グループ2 利用者が申込可

ジョブクラス制限値（抜粋）は以下のとおりです。（表2）

表2. ジョブクラス制限値（抜粋）

キュー名	CPU数	最大ノード数	制限時間 (経過時間)	メモリー容量 (GB) ※1	パーソナル				グループ		研究室コース
					コース1	コース2	コース3	コース4	コース1	コース2	
Q001	16	1	48 時間	112(112)	×	△	△	△	△	△	×
Q004	〃	4	24 時間	〃	×	×	△	△	△	△	×
Q008	〃	8	24 時間	〃	×	×	×	△	×	△	×

△ 追加オプションにより申込可

※1 1ノードあたりの標準値（括弧内は最大値）

付録H ジョブクラス制限値

以下のジョブクラス制限値は 2009 年 3 月 1 日現在のものを掲載しています。今後、運用状況に応じて変更を行なう場合がありますので、Web ページ、スーパーコンピューティングニュース表紙裏等で最新の情報を御確認下さい。

ジョブクラス制限値(2007 年 4 月 1 日変更)

キュー名	CPU 数	最大ノード数	制限時間 (経過時間)	メモリー容量 (GB) ※1	パーソナル				グループ		研究室コース
					コース1	コース2	コース3	コース4	コース1	コース2	
(single)											
A	1	-	1 時間	2(56)	○	○	○	○	○	○	○
B	"	-	10 時間	"	○	○	○	○	○	○	○
C	"	-	50 時間	"	○	○	○	○	○	○	○
D	"	-	225 時間	"	○	○	○	○	○	○	○
S1	8	1	2 時間	56(56)	○	○	○	○	○	○	○
S4	"	4	2 時間	"	×	×	×	×	×	×	○
H1	"	1	24 時間	"	×	○	○	○	○	○	×
(parallel)											
P001	16	1	24 時間	112(112)	×	○	○	○	○	○	×
P004	"	4	12 時間	"	×	×	○	○	○	○	×
P008	"	8	8 時間	"	×	×	×	○	×	○	×
P016	"	16	6 時間	"	×	×	×	○	×	○	×
P064 ※2	"	64	2 時間	"	×	×	×	○	×	○	×
debug	"	4	5 分	"	○	○	○	○	○	○	×
Q001	"	1	48 時間	"	×	△	△	△	△	△	×
Q004	"	4	24 時間	"	×	×	△	△	△	△	×
Q008	"	8	24 時間	"	×	×	×	△	×	△	×
インタラクティブ	1	-	18 時間	512MB (1GB)	○	○	○	○	○	○	○

△ 追加オプションにより申込可

※1 1ノードあたりの標準値(括弧内は最大値)

※2 P064 キューは月1回のみサービス(原則として最終金曜日午後～翌月曜日朝)

NQS でのキューの指定方法について

- single(A～D), parallel(P001～P064) 利用については, “#@\$-q single”, “#@\$-q parallel” と指定する。
- それ以外のキュー(S1, S4, H1, debug, Q001, Q004, Q008) については, キュー名をそのまま指定する。(例: “#@\$-q S1”, “#@\$-q debug”)
- キュー名は, 大文字・小文字を区別しますので, 注意が必要です。

付録 I 利用負担金

2009年3月1日現在の利用負担金は以下のとおりです。(2007年4月1日に利用負担金が改正されました。基本的には、それ以前までの定額制を継続しています。) 表に示している金額は利用開始月を4月とした場合の金額です。最新の情報はスーパーコンピューティング部門のWeb ページ等でご確認下さい。

利用負担金表(2007年4月1日変更)

コース		負担金額(税込)	/batch	備考
パーソナル コース	コース1	20,000 円/年	10GB	計算リソース追加オプションは 不可
	コース2	80,000 円/年		
	コース3	200,000 円/年		
	コース4	400,000 円/年		
グループ コース	コース1	1,500,000 円/年	200GB	利用者番号 10 個まで
	コース2	3,000,000 円/年		利用者番号 10 個まで
	追加オプション (1利用者番号 につき)	150,000 円/年	+20GB	コース1用利用者番号追加
		300,000 円/年		コース2用利用者番号追加
研究室コース (1利用者番号につき)		50,000 円/年	10GB	計算リソース追加オプションは 不可
追加 オプション	計算 リ ソ ー ス	1ノード専用 キュー	20,000 円/月	パーソナル2・3・4, グループ1・2 利用者が申込可
		4ノード専用 キュー	50,000 円/月	
		8ノード専用 キュー	80,000 円/月	
	ディスク	600 円/(GB*年)		GB 単位で申込可

- 新規登録時の利用期間は当該年度末サービス終了日までを基本とする。ただし、4月登録者については、申し込み時に年度途中で計算機利用を中止する月を指定できる。この場合を除き、年度途中で計算機利用を中止した場合であっても利用負担金額の変更は行わない。

- 利用負担金は、原則として以下のとおりそれぞれ初旬に一括して請求する。
 - パーソナルコースは、利用開始月が4月から5月までは7月、6月から8月までは10月、9月から11月までは1月、12月から3月までは3月。
 - グループコース・研究室コースは、利用開始月が4月から6月までは7月、7月から9月までは10月、10月から12月までは1月、1月から3月までは3月。
 - 計算リソース追加オプションについては、申し込み月直後の7月、10月、1月、3月。
 - 上記以外の月に請求を希望する場合は、別途相談に応じる。
- パーソナルコース(ただし、本センターのスーパーコンピューターシステムに初めて登録された利用者)においては、利用開始月の翌月末日までに利用を中止することができる。利用負担金はパーソナルコースの利用開始月3月の金額を適用し、請求する。
- 利用負担金が減額となる変更はできない。
- コース間の変更については、利用負担金が増額になる場合のみ別途相談に応じる。(ただし、利用者番号変更の場合がある。)
- グループコースのディスク量は、グループ全体の上限値である。
- 研究室コースの申し込みは、以下の要件を満たす必要がある。
 - 同一研究室内にてグループを構成する。
 - グループは3名以上(教員・学生各1名以上)で構成される。(教員がパーソナルコース2以上に登録している場合は、学生2名以上のみの登録を可能とする。)
 - 登録人数の上限はない。
 - 教員がパーソナルコース2以上に登録している場合は、当該教員に対してS4キューの実行を可能とする。
- 計算リソース追加オプションの申し込みは、月単位とする。それぞれのキューについて、各月の上限を10名とする。申し込み者の人数により、お断りする場合がある。利用申し込みは、スーパーコンピューティング部門のWebページから行う。

2007年度までは、年度途中での利用中止ができませんでしたが、2008年度より、前年度からの継続利用者および4月新規登録者についてのみ、当該年度途中での利用中止月を指定することができることになりました。継続申し込み時または新規登録申し込み時に、利用終了希望月を指定してください。

5月以降の新規登録者については、これまで通り中止はできません。ただし、上記注釈にあるとおり、パーソナルコース(ただし、本センターのスーパーコンピューターシステムに初めて登録された利用者)においては、利用開始月の翌月末日までに利用を中止することができます。

なお、上記の場合を除き、年度途中で計算機利用を中止した場合であっても利用負担金額の変更は行いませんのでご了承ください。

付録 J マニュアル一覧

以下のマニュアルはメーカーが発行しており、購入することができます。マニュアル購入については(株)日立インターメディックス(03-5281-5054)までお問い合わせ下さい。なお、これらのマニュアルはセンター1階プログラム相談室で閲覧できます。

マニュアル名称	マニュアル番号
最適化 FORTRAN90 言語	3000-3-C21
最適化 FORTRAN90 使用の手引	3000-3-C22
最適化 FORTRAN77 言語	3000-3-C23
最適化 FORTRAN77 使用の手引	3000-3-C24
最適化 C 言語	3000-3-C31
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
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
数値計算副プログラムライブラリ MSL2 操作	3000-3-C86
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一疎行列解法 MATRIX/MPP/SSS	3000-3-C88

(英文)

マニュアル名称	マニュアル番号
Optimizing FORTRAN90 Reference	3000-3-C21(E)
Optimizing FORTRAN90 User's Guide	3000-3-C22(E)
Optimizing FORTRAN77 Reference	3000-3-C23(E)
Optimizing FORTRAN77 User's Guide	3000-3-C24(E)
Optimizing C Reference	3000-3-C31(E)
Optimizing C User's Guide	3000-3-C32(E)
Optimizing Standard C++ User's Guide	3000-3-C33(E)
MSL2 Matrix Calculation User's Guide and Reference	3000-3-C83(E)
MSL2 Function Calculation User's Guide and Reference	3000-3-C84(E)
MSL2 Statistical Calculation User's Guide and Reference	3000-3-C85(E)
MSL2 User's Guide	3000-3-C86(E)
MATRIX/MPP User's Guide and Reference	3000-3-C87(E)
MATRIX/MPP/SSS User's Guide and Reference	3000-3-C88(E)

参考資料

本センターで発行されたスーパーコンピューティングニュースの記事のうち、システムの利用に関連する記事の一覧です。現在のシステムやサービス等と異なっている場合がありますので、閲覧される際は御注意下さい。

(スーパーコンピューティングニュース)

SR11000 のファイルシステムについて	Vol.10 No.5 (2008.9)
フリーソフトウェアについて	Vol.10 No.1 (2008.1)
高速ファイル転送ソフト SRFT	Vol.9 No.5 (2007.9)
SR11000 を使い始めた人のためのコンパイルのちょっとしたコツ	Vol.9 No.4 (2007.7)
複数プログラムの実行方法と環境変数について	Vol.9 No.4 (2007.7)
NQS におけるプロセス数の指定について	Vol.9 No.4 (2007.7)
センターによく寄せられる質問とその回答集 (FAQ) 2007 年度版	Vol.9 No.3 (2007.5)
超並列ジョブ (64 ノード) 実行サービスについて	Vol.9 No.3 (2007.5)
計算リソース追加オプションについて	Vol.9 No.3 (2007.5)
システム入れ替え及び 2007 年度のサービスについて	Vol.9 No.2 (2007.3)
64 ビットモード標準化における注意点	Vol.9 No.2 (2007.3)
共有ライブラリの作成方法と注意点	Vol.9 No.2 (2007.3)
環境変数の設定及びコンパイラオプションについて	Vol.9 No.2 (2007.3)
時間計測の方法について	Vol.8 No.5 (2006.9)
C 言語におけるファイル入出力の高速化	Vol.8 No.5 (2006.9)
プログラム実行時の入力ファイルの与え方について	Vol.8 No.4 (2006.7)
Fotran におけるファイル入出力の高速化	Vol.8 No.4 (2006.7)
ファイルシステムについて	Vol.8 No.2 (2006.3)
SR11000/J1 prun コマンド(-n, -f オプション)の利用について	Vol.8 No.1 (2006.1)
SR11000/J1 ScaLAPACK の利用について	Vol.8 No.1 (2006.1)
SR11000/J1 フリーソフトウェアについて	Vol.7 No.4 (2005.7)
SR11000/J1 BLAS・LAPACK の利用について	Vol.7 No.4 (2005.7)
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	Vol.7 No.3 (2005.5)
次期ベクトル並列型スーパーコンピューターシステム SR11000 のご紹介	Vol.6 No.6 (2004.11)
システム変更等のお知らせ	各号

付録K マニュアル Web 閲覧サービス

2005年4月6日よりSR11000 マニュアルの Web 閲覧サービスを開始しました。当サービスでは以下のマニュアル（日本語版 及び 英語版）を公開しています。

- ・ 最適化 FORTRAN90 言語
- ・ 最適化 FORTRAN90 使用の手引
- ・ 最適化 FORTRAN77 言語
- ・ 最適化 FORTRAN77 使用の手引
- ・ 最適化 C 言語
- ・ 最適化 C 使用の手引
- ・ 最適化標準 C++ 使用の手引
- ・ 数値計算副プログラムライブラリ MSL2 行列計算
- ・ 数値計算副プログラムライブラリ MSL2 関数計算
- ・ 数値計算副プログラムライブラリ MSL2 統計計算
- ・ 数値計算副プログラムライブラリ MSL2 操作
- ・ 行列計算副プログラムライブラリ MATRIX/MPP
- ・ 行列計算副プログラムライブラリ 疎行列解法 MATRIX/MPP/SSS
- ・ ベクトル並列型スーパーコンピュータ SR11000 チューニングガイド
（日本語版のみ、2005年5月30日公開）

利用にあたっては次の点にご注意ください。

- ・ マニュアル Web 閲覧サービスの利用は、スーパーコンピュータの利用が認められた利用者本人のみに限定しています。
- ・ マニュアルの印刷・コピーは、その利用者個人がスーパーコンピュータ利用に使用する限り認めます。

なお、マニュアルの Web ブラウザによる閲覧は sr11000-s へのログイン（利用者番号・パスワードが必要）とポートフォワーディングの設定が必要となります。以下に閲覧方法を説明します。

Web 閲覧方法

SSH による sr11000-s への通常のログインと、SSH のポートフォワーディング機能を用いて、ローカルホストの特定のポート*宛の通信を "manual" マシンのポート 80 番に転送します。以下の例ではローカルホストのポートに 8080 番を使用しています。

※ローカルホストのポートは 1024 番以降の空いているポートであれば、他のポートでも構いません。

UNIX, MacOSX の設定例

ターミナルに次のコマンドを入力します。（下線部は適宜変更してください。）

```
ssh -L 8080:manual:80 sr11000-s.cc.u-tokyo.ac.jp -l p07xxx
```

8080 : ローカルホストのポート番号
p07xxx : 利用者番号

次にパスワードを入力し sr11000-s にログインします。sr11000-s にログインしたままウェブブラウザで次の URL を参照してください。

- 日本語版マニュアル
<http://localhost:8080/manual-j/index.html>
<http://127.0.0.1:8080/manual-j/index.html> (MacOSX はこちら)
- 英語版マニュアル
<http://localhost:8080/manual-e/index.html>
<http://127.0.0.1:8080/manual-e/index.html> (MacOSX はこちら)

※ローカルホストのポート番号を変更した場合は、URL 中の"8080"を変更したポート番号に修正してください。

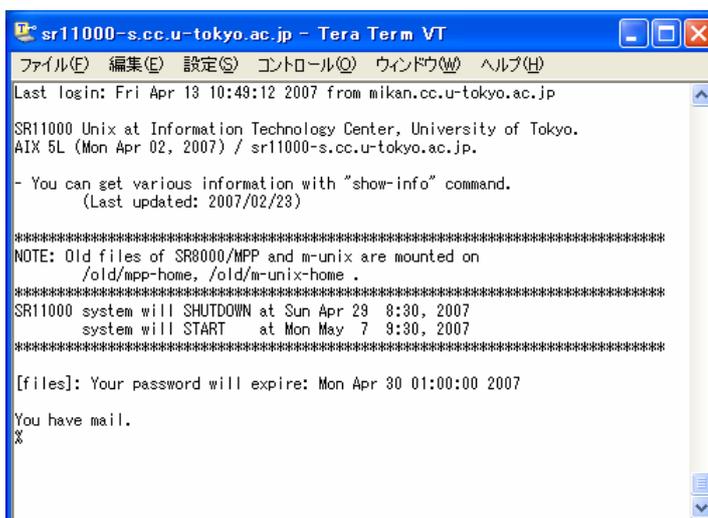
Windows の設定例

UTF-8 TeraTerm Pro with TTSSH2 を使用した例

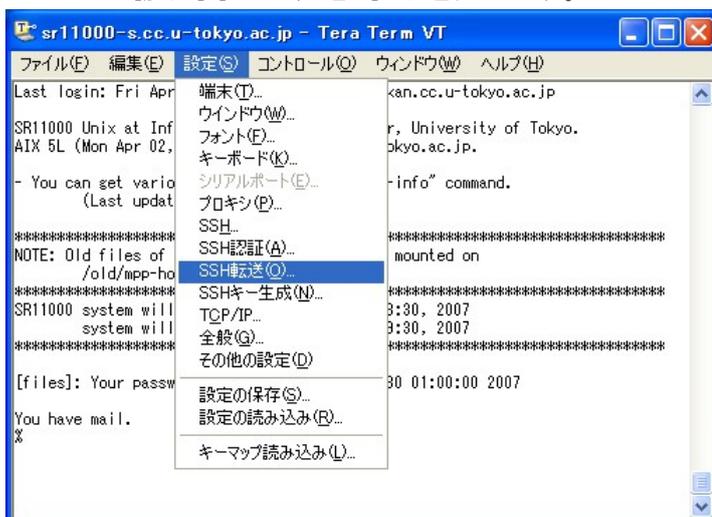
UTF-8 TeraTerm Pro with TTSSH2 を起動し、sr11000-s にログインします。



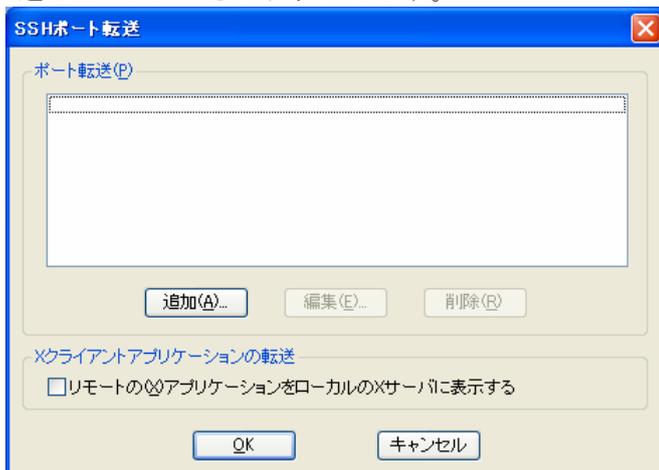
(ログイン後の画面)



メニュー [設定]-[SSH 転送...] を選択します。

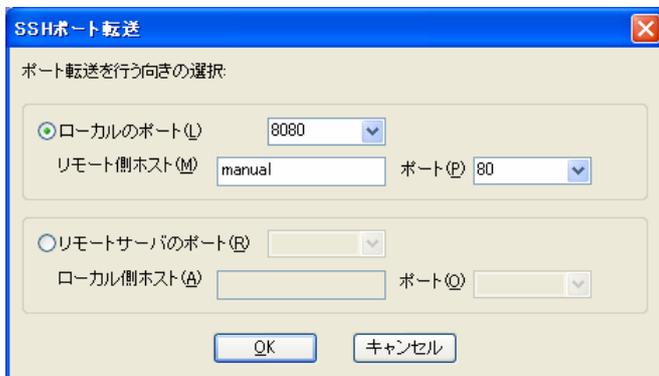


"追加..." ボタンをクリックします。

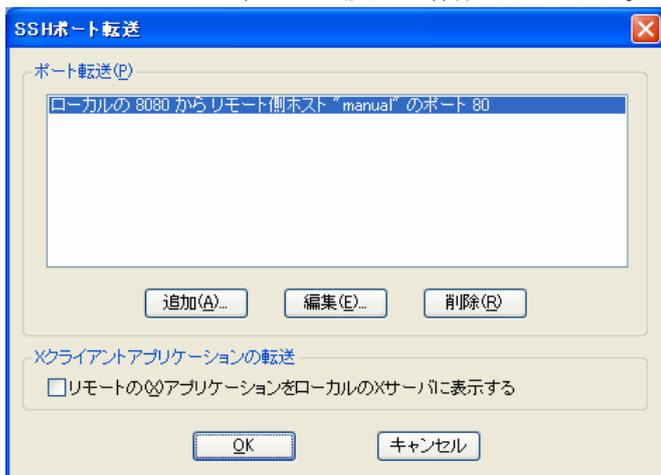


以下のとおり設定し, "OK" ボタンをクリックします。

ローカルのポート "8080" (→ 1024 番以降の空いているポートを指定する)
リモート側ホスト "manual", ポート "80"



ポートフォワーディングの設定が作成されました。"OK" ボタンをクリックします。



sr11000-s にログインしたまま、Web ブラウザで次の URL を参照してください。

- 日本語版マニュアル
<http://localhost:8080/manual-j/index.html>
- 英語版マニュアル
<http://localhost:8080/manual-e/index.html>

※ローカルホストのポート番号を変更した場合は、URL 中の"8080"を変更したポート番号に修正してください。

付録L センターによく寄せられる質問とその回答集 (FAQ)

プログラム相談室に寄せられた質問を中心に、Email などによる質問、スーパーコンピューティングニュース付録の質問書による質問の中から、センター固有の事柄で、他の文献を見てもわからないものを中心に、2009年3月時点での状況をまとめ、スーパーコンピューターに関する事項を抜粋したものです。(したがって、UNIXの使い方とか、一般的なFortranの文法、特定のアプリケーションの使い方に関する質問などは記述していません。) 上記のような質問とその回答を

1. 概要
2. 登録・利用負担金
3. システムへの接続
4. システム全般
5. Fortran
6. C・C++
7. ライブラリ
8. NQS
9. 制限値
10. 並列処理
 11. MPI
 12. ファイル転送
 13. 他システムからの移植
 14. その他

に分けて記述してあります。

1. 概要

(Q) センターでは、どんなコンピューターが使えますか？

(A) センターでは、以下のスーパーコンピューターシステムによる大規模計算サービスを行っています。

- ・ スーパーコンピューター HITACHI SR11000 (OS: AIX 5L)
- ・ スーパーコンピューター HITACHI HA8000 クラスタシステム (OS: RedHat Enterprise Linux 5)

2. 登録・利用負担金

(Q) パーソナル・グループ・研究室コースとは何でしょうか。

(A) パーソナルコースは個人利用者を対象に4コース設定しており、利用目的によってコースを選択できます。グループコースはプロジェクトグループを対象にしたコースで、複数人で利用する場合に便利です。グループコースは2コース設定してあります。研究室コースは中小規模研究室を対象としたコースです。それぞれ利用できる資源については、Web ページまたはスーパーコンピューティングニュース表紙裏の「利用負担金表」を参照して下さい。

(Q) 利用者番号を異動後も継続できますか？

(A) 4月上旬までに異動先の統括支払責任者の承諾が得られれば、同一の利用者番号によって継続することが可能です。

(Q) 現時点での課金額を調べるには？

(A) 定額制のため、計算リソース追加オプションを除き、年度途中で利用者自身でコース変更やディスク使用量等の変更手続きを行わない限り、利用負担金額が変わることはありません。

3. システムへの接続

(Q) システムに telnet で接続できない。

ファイル転送をしたいが、ftp で接続できない。

(A) セキュリティ向上のため、telnet, ftp での接続を禁止しています。システムへの接続には、ssh や scp を利用して下さい。利用方法の詳細は「3. ログイン方法」を参照して下さい。

(Q) センターのサービス機の IP address を教えて下さい。

(A) 現在、サービスマシンの IP address は公開しておりませんので、sr11000-s.cc.u-tokyo.ac.jp といった DNS (Domain Name System) によるホスト名を用いてアクセスするようにして下さい。現在、DNS によるアクセスができずに直接 IP address を指定している方は、担当のネットワーク管理者と御相談の上、DNS を参照してホスト名でアクセスできるよう環境を整えるようにして下さい。

4. システム全般

(Q) パスワードを忘れました。

(A) 利用登録のお知らせと身分証明書を持って、本センター受付までお越しいただくか、共同利用係 (03-5841-2717) まで御連絡下さい。なお、パスワードは電話やメールでの回答は行っておりませんので、御承知おき下さい。

(Q) 現時点での CPU 時間を調べるには？

(A) CPU 時間は下記の la コマンド (list account の略) で制限値 (無制限)、使用量 (合計と今月分)、残り時間が確認できます。また -a オプションを付けることで使用ディスク量 (制限値、使用量)、設定ノード数も調べられます。(以下は2008年4月現在の表示例です。今後変更になる場合があります。)

```
% la -a
cpu resource for p08000 (HH:MM:SS)
      system      limit      used(this month)      remain
SR11000  596523:14:07    40:10:20(   19:41:38)  596483:03:47
Course name for p08000
personal 4
disk resource  user(p08000)
      file system  limit(MB)  occupied(MB)
      /batch      10240     1313
```

なお、表示される値は 1 日に 2 回反映しているもので、待ち行列にあるジョブや実行中のジョブについては含まれていません。

(Q) ".... command not found" というメッセージが出て、コマンドを受け付けない。

(A) センターでは、システムへの登録時に、コマンドパス等を設定する .cshrc や .login ファイルをホームディレクトリに作成しています。このファイルを削除したり、中の記述がおかしくなってしまうと、正しい設定が行われなくなり、標準的なコマンドも使えなくなってしまうことがあります。この .cshrc や .login といったファイルの標準的なものは、/usr/local/skel にあり、各自で適宜コピーし回復することができます。

- (Q) nslookup や traceroute を実行すると " Permission denied " となる。
w や ps ax と入力したときに、自分以外のプロセスが見あたらないのですが。
- (A) 管理者用のコマンドについては、一般利用者は実行できないように設定してあります。また ps, w 等のシステムの状態を表示するコマンドについては、システム管理者以外の一般ユーザーが使った場合、本人に関する情報のみを表示するように設定してあります。
- (Q) 利用者番号間でファイルを譲渡したい。
- (A) UNIX 系はサブディレクトリを含む場合は、個別共有関係の設定ができないため、tar のようなアーカイブと、外部媒体を利用するのが容易です。グループコースの場合は、標準ではグループ内の利用者番号間では参照可能となっています。

5. Fortran

- (Q) % a.out と入力したが、実行できない。
- (A) a.out の前に ./ (ピリオド, スラッシュ) を入れてください。以下は例です。
- ```
% f77 test.f (コンパイル)
% ./a.out (実行)
```
- (Q) Fortran プログラムで長い名前の変数を使いたい。
- (A) FORTRAN77 プログラムでの変数名の長さはセンターの標準では 8 文字以内に制限されています。9 文字以上の変数名及び外部手続き名を使用する場合は、コンパイルオプション `-i,U` を指定して下さい。( `-i,U` は、外部手続きの名称中の `_` (アンダースコア) 及び 31 文字までの外部手続き名称を許すオプションです。) Fortran90 (f90) ではこの制限はありません。
- ```
% f77 -i,U a.f
```
- (Q) Fortran プログラムが "segmentation fault" で終わる。
コンパイルオプションによって結果が異なる。
- (A) 配列の宣言範囲を超えて使用している関数・サブルーチンの引数の個数・型が不一致であることが考えられます。プログラムを確認して下さい。 `-debug` オプションで原因がわかる場合があります。
- (Q) Fortran プログラムのチューニングをしたいのですが。
- (A) 性能モニター情報や、コンパイルログメッセージを出力することができます。コンパイルオプション `-pmpar, -pmfunc, -loglist` を使用します。チューニングの詳細については「ベクトル並列型スーパーコンピュータ SR11000 チューニングガイド」(冊子版および認証付きマニュアル Web 閲覧による参照) を御覧下さい。
- (Q) プログラムの実行が遅いのですが。
- (A) 最適化、要素並列化が効いてないかもしれません。 `-loglist` オプション等を使用してコンパイル診断メッセージを確認して下さい。またプログラムによっては、最適化が性能を劣化させる場合もあります。この場合、一部の最適化オプションを抑止してコンパイルして下さい。詳細は以下の日立発行のマニュアルの「最適化による副作用」を参照して下さい。
- ```
SR11000 最適化 FORTRAN90 使用の手引 (3000-3-C22)
SR11000 最適化 FORTRAN77 使用の手引 (3000-3-C24)
```

## 6. C・C++

(Q) C++ の STL (標準テンプレートライブラリー) は使用できますか。

(A) 使用できます。以下のように指定してください。

64 ビットモードでコンパイルする場合

```
% sCC sample.cpp -I/opt/STLport-4.6.2/stlport -L/opt/STLport-4.6.2/lib64 -lstlport_sCC -lm
```

32 ビットモードでコンパイルする場合

```
% sCC -32 sample.cpp -I/opt/STLport-4.6.2/stlport -L/opt/STLport-4.6.2/lib -lstlport_sCC -lm
```

## 7. ライブラリ

(Q) 使用できる数値計算ライブラリーにはどのようなものがありますか？ また、それぞれどのようにすれば使えるのですか？

(A) 以下のライブラリーが用意されています。使用方法に関しては以下の日立発行のマニュアル、または「13. 数値計算ライブラリー」を参照して下さい。

- ・ 行列計算副プログラムライブラリ MATRIX/MPP
  - SR11000 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
  - SR11000 行列計算副プログラムライブラリ 疎行列解法 MATRIX/MPP/SSS (3000-3-C88)
- ・ 数値計算副プログラムライブラリ MSL2
  - SR11000 数値計算副プログラムライブラリ MSL2 行列計算 (3000-3-C83)
  - SR11000 数値計算副プログラムライブラリ MSL2 関数計算 (3000-3-C84)
  - SR11000 数値計算副プログラムライブラリ MSL2 統計計算 (3000-3-C85)
  - SR11000 数値計算副プログラムライブラリ MSL2 操作 (3000-3-C86)
- ・ BLAS (Basic Linear Algebra Subprogram)
- ・ LAPACK (Linear Algebra PACKage)
- ・ ScaLAPACK (Scalable Linear Algebra PACKage)

(Q) MATRIX/MPP を使用したプログラムがコンパイルできない。

(A) 以下の 2 点にご注意下さい。

① コンパイルオプションの位置

```
ld: 1012 error : Undefined symbol: .HDFT2M
-lmatmpp はオプションの一番最後に指定して下さい。
```

```
% f90 -parallel program.f -lmatmpp
```

② 要素並列オブジェクトの作成

```
ld: 1012 error : Undefined symbol: _h_comp_link_var
ld: 1012 error : Undefined symbol: _h_total_ip_num
-parallel を指定して下さい。(要素並列オブジェクトのリンク)
```

```
% f90 -parallel program.f -lmatmpp
```

スカラー版を利用する場合は以下の指定をして下さい。

```
% f90 program.f -lmatmpp_sc
```

(Q) ライブラリー MSL2 を使うには？

(A) 日立提供のライブラリーとしては、MSL2 及び MATRIX が、マニュアルに記述された標準的な形態で登録してありますから、マニュアルの記述どおりに利用することができます。但し、センターでは f90, f77 コマンドを用いてリンクする場合、-L オプションの指定は必要ありません。

```
% f90 ... -lMSL2 ...
```

詳細は「12. 数値計算ライブラリー」を御覧下さい。

- (Q) スーパーコンピュータ上の Gaussian を利用しています。何か注意すべき事柄はありますか？
- (A) Gaussian については「付録D Gaussian03 の使い方」を参照して下さい。
- (Q) Gaussian を使っているが、PC と同じくらいの速さでしかない。なぜでしょうか？
- (A) 本センターの Gaussian は要素並列版ではありますが、スカラー処理が大部分を占めるような計算の場合、1 プロセッサ (1 ノードの 1/8 または 1/16) 程度の性能で処理するため、PC と同程度であることも考えられます。

## 8. NQS

- (Q) ジョブクラスにはどのような種類がありますか。
- (A) 以下に説明します。
- ・スカラージョブクラス single(A, B, C, D)  
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) でクラス分けをしています (要素並列ジョブ、複数ノードを使用するジョブの実行はできません)。
  - ・並列ジョブクラス  
8CPU/ノード用 S1, S4, H1  
16CPU/ノード用 parallel (P001, P004, P008, P016, P064) , debug, Q001, Q004, Q008  
要素並列プログラムの実行及び複数ノードを使用するジョブの実行を行うジョブクラスです。1 ノードあたりの CPU 数、使用ノード数と実行時間 (経過時間) でクラス分けをしています。Q001~Q008 は計算リソース追加オプションの申し込みが必要です。P064 は超並列ジョブ実行サービスとして、月に1回 (原則として最終金曜日午後~翌月曜日朝) パーソナルコース4 またはグループコース2の利用者が実行可能です。

(Q) 投入した NQS ジョブを kill する方法は？

(A) qdel コマンドで削除します。例えば、リクエスト ID 1425.n121 を削除する場合は次のように指定します。

```
% qstat
2007/04/09 (Mon) 13:47:21: BATCH QUEUES on SR11000
NQS schedule stop time : 2007/04/29 (Sun) 8:30:00 (Remain: 474h 42m 39s)
 REQUEST NAME OWNER QUEUE PRI NICE CPU MEM STATE
1425.n121 job.csh p08000 A 63 0 unlimit 2048MB RUNNING
% qdel 1425
```

- (Q) バッチジョブが実行されません (NQS の計画停止に引っかかる)。
- (A) 投入されたジョブのうち、システム停止予定時刻 (計画停止時刻) までに終了しないジョブについては実行しないよう設定しています。停止時刻直前まで実行するためには、実行時間制限値 (#@\$-1T) で JOB の終了が計画停止時刻を超えないよう指定して下さい。計画停止時刻と残り時間は qstat コマンドで確認することができます。(前項の例参照)

## 9. 制限値

(Q) ジョブが途中で終了してしまう。

配列を大きくしたら動かなくなった。

- (A) インタラクティブ、NQS の各キューに応じて、E-TIME、メモリーなどのリソースを制限していますので、その制限を越えるジョブは流せません。特に、インタラクティブ 環境では、メモリーの制限が厳しくなっていますので、大きなメモリーを必要とする場合には NQS を利用して下さい。各キューにおける制限値は、Web ページまたはスーパーコンピューティングニュース「ジョブクラス制限値」を参照して下さい。

- (Q) インタラクティブ処理の場合の CPU 時間の制限値は？

- (A) インタラクティブ実行の制限値は（経過時間で）18 時間です。また、2 時間以上無入力状態が継続する場合は、システムが自動的にキャンセルします。

- (Q) メモリーをどれだけ使えますか？

- (A) インタラクティブ処理では標準で 512MB、最大で 1GB です。NQS でバッチ処理をする時は、single キュー(A～D)は標準で 2GB 最大で 56GB、ノード専有キューでは 1 ノード当たりで使用可能なメモリーの最大値は、S1, S4, H1 キュー（8CPU/ノードのキュー）では 56GB、P001～P064, debug, Q001～Q008 キュー（16CPU/ノードのキュー）では 112GB です。

NQS の場合は、#@\$\_1M 値 MB または GB で指定します。single キュー以外では通常指定する必要はありません。single キューでは最大値以下を指定して下さい（この値を越えていると、目的のキューに投入されません）。

```
#@$-1M 8GB
```

また、インタラクティブ処理の場合、datasize が 512MB に標準設定されていますので、必要に応じて limit コマンドで上限値を変更して下さい。

```
% limit datasize 768M
```

- (Q) NQS ジョブを投入したところ、以下のようなメッセージが返ってきて、ジョブが投入できない。

```
No suitable queue found.
reason:
```

- (A) 制限時間、メモリー容量等の指定がキューの上限値を越えていると思われます。この場合、上記メッセージを出力し、ジョブは投入されません。ジョブクラス毎の制限値を越えないよう指定して下さい。

- (Q) NQS ジョブが “killed” または “no more process” で終わってしまう。

- (A) メモリー不足が考えられます。single キュー以外では#@\$\_1M を削除してください。single キューでは#@\$\_1M でメモリーを拡張して下さい。

## 10. 並列処理

- (Q) 要素並列実行とは何ですか。

- (A) 要素並列化機能とは 1 ノード内（16 または 8 プロセッサ）の複数プロセッサを並列に利用してプログラムを高速実行させる機能です。コンパイルオプションやソースプログラム中の指定によりコンパイラーが要素並列化オブジェクトを作成します。実行はバッチジョブ（single キュー以外）で行ないます。

コンパイル例

```
% f90 -parallel program.f
```

※リンク時にも -parallel の指定が必要です。

```
% f90 -parallel program.o object.o
```

詳細は日立発行のマニュアルを参照して下さい。

SR11000 最適化 FORTRAN90 使用の手引

(3000-3-C22)

|         |                     |              |
|---------|---------------------|--------------|
| SR11000 | 最適化 FORTRAN77 使用の手引 | (3000-3-C24) |
| SR11000 | 最適化 C 使用の手引         | (3000-3-C32) |
| SR11000 | 最適化標準 C++ 使用の手引     | (3000-3-C33) |

(Q) ノードの専有, 共有とは何ですか?

(A) スーパーコンシステムでは複数の利用者で1ノードを共有する「ノード共有」と、1利用者が1ノードを専有する「ノード専有」の2通りの利用形態があります。ノード専有の場合は要素並列ジョブ、ノード内並列ジョブの実行が可能です。

・ノード共有となる場合

```
ログインセッション sr11000-s
バッチジョブ single (キュー A ~ D)
```

・ノード専有となる場合

```
バッチジョブ parallel (キュー P001 ~ P064)
 S1, S4, H1, debug, Q001, Q004, Q008
```

(Q) sr11000-s での並列ジョブの実行ができますか?

(A) インタラクティブでは並列ジョブの実行はできません。バッチジョブ(NQS)をご利用ください。

## 11. MPI

(Q) MPI やPVMは使えますか? 使い方は?

(A) 並列プログラミングのためのMPIは、メーカーから提供されており、利用可能です。

詳細については、「11. 並列アプリケーション」を参照して下さい。特に、コンパイル・リンク時に必要な指定が省略できるように、mpif77, mpif90, mpicc, mpiCCの各コマンドが用意されていますので、こちらを御利用下さい。PVMは利用できません。

(Q) MPI のプログラムを動かしたい。

(A) コンパイルコマンドはmpif77, mpif90, mpicc, mpiCC, 実行コマンドはmpirunを使用して下さい。

mpirunの実行ノード数は、ジョブスクリプトで指定したノード数が使用され、-n オプションは無効となります。また、実行プロセス数はジョブスクリプトのノード数とジョブタイプで決定され、-np オプションは無効となります。ジョブタイプにSSを指定すると、プロセス数はノード数のCPU数倍(8または16)となります。ジョブタイプにTnを指定すると、1ノードあたりのプロセス数がn個となります。ジョブタイプを指定しない場合はプロセス数はノード数と同じ値となります。

例 1)

```
##$-q parallel
##$-N 4
mpirun ./a.out → 4 ノード 4 プロセスで実行します。
```

例 2)

```
##$-q parallel
##$-N 4
##$-J SS
mpirun ./a.out → 4 ノード 64 プロセスで実行します。
```

例 3)

```
##$-q parallel
##$-N 4
##$-J T5
```

mpirun ./a.out → 4 ノード 20 プロセスで実行します。

## 1.2. ファイル転送

- (Q) パソコンで作成したファイルをセンターのシステムに scp すると、改行が `M` になっていたり、ファイルの最後に `Z` が入っていたりする (scp したプログラムをコンパイルできない)。
- (A) パソコン側の scp の設定をバイナリーモードではなくテキストモードにしてください。これにより、MS-DOS(Windows) の改行コード (CR+LF) は UNIX の改行コード (LF) になります。

## 1.3. 他システムからの移植

- (Q) 他サイトでコンパイルしたプログラムがエラーで終了する。
- (A) 本センター以外でコンパイルしたプログラムの実行については、ソフトウェアのバージョンや環境等の違いにより正常に動作しない可能性があります。本センターにてコンパイルを行い、実行するようにしてください。

- (Q) WS で動いていた Fortran プログラムを動かしたい。

- (A) 取りあえずコンパイルしてみると、膨大な量のメッセージが出てきて驚かされることがしばしばあります。その代表的な例及び原因と対処法は次のとおりです。
  - ① 8 文字より長い変数名を使っている。  
前項のとおりコンパイルオプションで対処して下さい。
  - ② 「原始プログラム中に許されない文字が現れました」  
?, !, \$, " などの特殊文字が使われている場合は言語仕様の拡張が必要です。コンパイルオプション `-i,P` や `-i,{E,N},LT` を指定して下さい。また、注釈以外での漢字コードはシフト JIS コードを御使用下さい。
  - ③ 「分類不可能な文があります」  
プログラムが自由形式で書かれている場合はコンパイルオプション `-free` を指定して下さい。これだけの対処をしてやれば、メッセージの量がかなり減るはずですが、後は入出力関連や (時間の計測等の) 処理系固有のルーチンを使っていないかチェックする必要があります。センターで利用する Fortran 処理系の詳細については、日立発行のマニュアルを参照して下さい。
    - SR11000 最適化 FORTRAN90 使用の手引 (3000-3-C22)
    - SR11000 最適化 FORTRAN77 使用の手引 (3000-3-C24)

- (Q) SR11000 で作成した業界標準形式のバイナリーデータが PC (LINUX) で読めない。

- (A) 業界標準形式のデータでも CPU のアーキテクチャーにより、互換性がなくなります。SR11000 や多くの WS のバイナリーデータはビッグエンディアンと呼ばれる順序で並びますが、Intel の CPU はリトルエンディアンと呼ばれる逆の順序で並ぶため互換性がありません。「10.3 実行時オプション」にある実行時オプションを指定してください。

- (Q) パソコンで作ったデータが読めない。

- (A) この質問は、まずパソコン上のワープロを使いはじめ、それからセンターのサービス機を利用するユーザーによく見られます。こういう事例の原因は大きく二つに分類できます。一つは、「行」の概念がないことです。大型汎用機用のアプリケーション、及びそれをベースに移植したものは、必ず入出力データに「カード」、「レコード」或は「行」といった単位を想定しています。Fortran, C といったプログラミング言語はもちろん、ライブラリ・パッケージも、ほとんど全てこの「行」という概念を

必要としますので、入力データの各行の末尾には行末コード（いわゆるリターンコード）が必要です。一方、ワープロでは、文章が行の右端に達した場合には（行末コードを入れないままに）自動的に次の行の左端にデータが続きます。行末コードを使うのは、段落を替える時です。このため、例えば 3 行必要なデータを、3 行分の長さのある長い 1 行のデータとして作ったりすると、「データが読めない」ということになるわけです。

次によくある事例が、いわゆる「全角スペース」（2 バイトの漢字 1 文字として表される空白（スペース）のこと）の問題です。アプリケーションで入力データを扱う際に、データとデータの区切りとして「空白（スペース）」が使われるのがほとんどですが、この場合の「スペース」はいわゆる「半角のスペース」（1 バイトで表される空白（スペース）のこと）である必要があります。これは、「全角のスペース」は漢字の一つと見なされるため、空白と判断されないためです。また同様に、数字についても半角の数字を使う必要があります。半角の数字(1, 2, 3, ... など)は正しく数字として解釈されますが、全角の数字(1, 2, 3, ... など)は全角のスペース同様、漢字の一種と見なされ、数字としては読み込まれません。

また、パソコンの（表計算等の）ソフトでデータを作成した場合には、ファイルにセーブする際に、区切り文字を何にするかとか、テキスト形式のファイルとしてセーブする等の注意も必要です。

#### 14. その他

(Q) マニュアルが欲しいのですが。

(A) 日立発行のマニュアルは、(株)日立インターメディアックス (03-5281-5054) で購入できます。センター発行の手引は本センター 1 階の受付で入手できます。いずれも本センターにて閲覧が可能です。また、SR11000 のマニュアルについては Web での閲覧が可能です。詳しくは「付録K マニュアル Web 閲覧サービス」を御覧下さい。

(Q) 過去のスーパーコンピューティングニュースやセンターニュースを見たい。

(A) スーパーコンピューティングニュースについては本センタースーパーコンピューティング部門 Web ページに掲載されています。冊子の参照及び入手については本センター1 階受付 (03-5841-2716) までお問い合わせ下さい。

(Q) データを、テープに落したい。  
プリンターに出力したい。

(A) 2007 年 3 月をもって利用者フロアサービスは終了しました。利用可能なテープ装置、プリンタはありません。ご了承ください。

(Q) メールを Eudora で操作したいのですが、POP の設定はどうすればいいのですか？

(A) センターサービス機では POP, IMAP によるメールのサービスはしておりません。従って Eudora 等のメールソフトは御利用になれません。

(Q) ～～というアプリケーションを使いたいのですが、センターで導入してもらえないでしょうか？

(A) センターへの要望として、文書にて御連絡下さい。

(Q) ... ecc.u-tokyo.ac.jp に関して、...

(A) 情報メディア教育部門（教育用計算機システム）にて提供しているサービスに関するご質問は情報メディア教育部門の Web ページ (<http://www.ecc.u-tokyo.ac.jp/>) を御覧下さい。

(Q) 他の大型センター（国立情報学研究所を含む）の計算機を利用したい。

(A) それぞれのセンターにて利用登録申請の手続きを行って下さい。

|                   |                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------|
| 北海道大学情報基盤センター     | <a href="http://www.hucc.hokudai.ac.jp/">http://www.hucc.hokudai.ac.jp/</a>                                     |
| 東北大学サイバーサイエンスセンター | <a href="http://www.cc.tohoku.ac.jp/">http://www.cc.tohoku.ac.jp/</a>                                           |
| 名古屋大学情報連携基盤センター   | <a href="http://www2.itc.nagoya-u.ac.jp/center/index.html">http://www2.itc.nagoya-u.ac.jp/center/index.html</a> |
| 京都大学学術情報メディアセンター  | <a href="http://www.kudpc.kyoto-u.ac.jp/">http://www.kudpc.kyoto-u.ac.jp/</a>                                   |
| 大阪大学サイバーメディアセンター  | <a href="http://www.hpc.cmc.osaka-u.ac.jp/">http://www.hpc.cmc.osaka-u.ac.jp/</a>                               |
| 九州大学情報基盤研究開発センター  | <a href="http://www.cc.kyushu-u.ac.jp/scp/">http://www.cc.kyushu-u.ac.jp/scp/</a>                               |
| 国立情報学研究所          | <a href="http://www.nii.ac.jp/index-j.html">http://www.nii.ac.jp/index-j.html</a>                               |

## 15. おわりに

ここにあげた質問とその回答集は、センターに寄せられる質問の一部に過ぎませんし、必ずしも現状に即しているとは限らないこともお断りしておきます。センターからは、スーパーコンピューティングニュース、Web ページ、login message、show-info コマンドなどを通じて、最新の情報を広報していますので、これらの情報も確認するようにして下さい。特に、Web ページ等は、編集/印刷/出版までに時間のかかるスーパーコンピューティングニュースによる広報に比べ、より即時性のある情報が得られるようになっています。また、本稿の内容はセンタースーパーコンピューティング部門の Web ページ(<http://www.cc.u-tokyo.ac.jp/>)にも掲載しています。これらも併せて御利用下さい。



---

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 第4版

---

2002年4月 初版発行  
2004年4月 2版発行  
2007年4月 3版発行  
2009年3月 4版発行

編集・発行 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  
〒113-8658 東京都文京区弥生 2-11-16

本書の一部あるいは全部について，東京大学情報基盤センターの許諾を得ずに  
いかなる方法においても無断で複写，複製することを禁じます。

---