2009.11.6(ver.3)

gcc 備忘録
（gcc と PGI コンパイラの使用法）

１．gcc 関係

Windows 環境と比べ、Linux 環境（今回は Fedora）を利用して数値計算する利点は、メモリが 2GB を越えるプログラムを実行できることである（Wondows 環境では実行ファイルが作成されない。）ただし、Linux でもコンパイル時にオプションは必要である。

以下に関連する項目をまとめる。

(1)gcc でのコンパイルの前に必要な作業

・Fedora9 では[アプリケーション]→[システムツール]→[端末]としてコマンドラインによる操作ができるようにすること。

・コンパイルするソースファイル(***.c または***.cpp)があるディレクトリにあらかじめ移動しておくこと。cd コマンドを使う

```
$ cd test↵
```

で元のディレクトリの階層下にある test ディレクトリに移動する。

```
$ cd ..↵
```

で上のディレクトリに移動する。

(2)gcc でコンパイル

・以下の例では test1.c をコンパイルして test1 という実行ファイルを出力する。

```
$ gcc -o test1 test1.c↵
```

※-o をつけないと a.out という実行ファイルが作成される。
※実行ファイルの名前と同名のディレクトリやファイルがあるとコンパイルできない。

・c++をコンパイルする場合は以下のようにする。拡張子は".cpp"となる。

```
$ g++ -o test1 test1.cpp↵
```

または gcc で-lstdc++をオプションでつける。

```
$ gcc –o test1 test1.cpp –lstdc++↵
```

・math.h を使っている場合は-lm をオプションでつける。

```
$ gcc -o test1 test1.c –lm↵
```

・メモリが 2GB を越えるコンパイルの方法は以下のようにする。-mcmodel=medium をつけること。

$ g++ -o test7 2di-2slot-k-slx2.cpp -lm -mcmodel=medium↵

※オプションをつけないと”relocation truncated to fit”のエラーメッセージが出る。

・-O1～3 をオプションでつけると自動最適化してくれるので劇的に計算時間が短くなる。O2 か O3 がよい。

$ gcc -o test1 test1.c –lm –O2↵

(3)プログラムの実行

・コンパイルして生成された実行ファイルを実行するには以下のようにする（./をつける）

$ ./test1↵

・プログラムの引数を渡す必要がある場合、引数オプションを忘れるとセグメント違反が出る。以下は test9 という実行ファイルに引数をつけた例である。

$ ./test1 1 test2/test2 0 1 100 120 11 10↵

(4)Perl の利用

Perl を使うとパラメータを引数として渡すことでプログラムを繰り返し実行することができる。Perl ファイル（拡張子が.pl）を作って実行させる場合は以下のようにする。
テキストエディタで例えば test.pl というファイルを作り、以下のように記述する。

・【例１】

```
$i=1;
system ("./test9 $i");          ※$i が実行ファイルの引数。system でコマンドの実行。
```

・【例２】

```
$i="test10";                    ※文字列も利用できる。
system ("mkdir $i");            ※test10 というディレクトリの作成
system ("./test9 1 $i/test2 0 1 100 120 11 10");
```

・【例３】for 文を使ったループは以下のように記述する。

```
for ($i=1;$i<=10;$i=$i+1){      ※10 回ループする。
   system ("mkdir test$i");     ※test1～test10 のディレクトリの作成
   system ("./test9 1 test$i $i");
}
```

perl を実行するには以下のようにする。

$ perl test.pl

２．PGI コンパイラ関係

(1) PGI C++ Workstation のインストール手順（release7.2 32bit/64bit 版の場合）

正式なインストールマニュアルはあるのだが、自分には必要がないことまで書いているので読んでいて煩わしい。そこで、PGI コンパイラの導入とインストール手順を以下にまとめる。Fedora にインストールすることを前提に説明するが、マニュアルにおける RedHat Linux の場合とほぼ同じである。

別途、ライセンスキー（license.dat ファイル）を取得する必要があるが、ここでは説明していない。ただし、取得には PGI 社のホームページにアクセスできるネットワーク環境が必要である。

7.2 以降はバージョンが上がっても、手順は同じである。

インストール前に gcc などの開発環境がインストールされていることが前提となっている。Fedora の場合はインストールされているので、気にしなくてもよい（はず）。不安があれば[ソフトウェアの追加/削除]であらかじめインストールする。

①release7.2 ではインストールを/opt/pgi ディレクトリに行う。インストールはルート権限で行う。

※ルートまたはユーザアカウントのどちらでログインしてもインストールは可能である。しかし、ファイルの移動などの操作が必要なる場合は、ルートでログインした方が煩わしくないかもしれない。

②インストールは英語環境で行う。以下のように確認と変更を行う。

```
# env | grep LANG↵          （確認→LANG=en_US と表示されれば O.K.）
# export LANG=en_US↵        （英語環境に変更（bash の場合））
```

※インストールだけではなく、使用する際も英語環境を推奨している（サポート対象が英語環境のみ）。

③ソフトウェアを http://www.softek.co.jp/SPG/ftp.html からダウンロードして/tmp/pgi に保存し，解凍する。

```
# tar xvzf pgilinux-724.tar.gz↵
```

※保存先はどこでもよいが，/opt/pgi だけはダメ。

※コマンド実行後、/tmp/pgi に解凍される。どこのディレクトリに保存してもよいが、解凍することを考えると、***/pgi というディレクトリを作っておいた方が分かりやすい。

④ファイルを展開したディレクトリ（今回の場合/tmp/pgi）に移動して，以下のようにインストールする。

```
# ./install
```

⑤インストーラが起動する。ライセンス許諾に関してメッセージが表示されるが長い。エンターキーで改行するが最後に承諾するか問われる。うっかりしていると空入力のままエンターキーを押してしまう。100%に近づいてきたら注意が必要。

- Please choose install option? → 1↵ （Single system install を選択）
- Installation derectory ? [/opt/pgi] → /opt/pgi↵
- Do you wish to install MPICH1? (y/n) → y↵ （MPITCH ライブラリのインストール）
- Remote execution method ? [rsh,ssh] → ssh↵ （とりあえず ssh にした）
- 名前，username，email を聞かれるらしいが聞かれなかった。
- Do you want the files in the install directory to be read-only ? [y/n] → n↵

※installation complete が表示されるとインストール終了。
※バージョンによって機能が追加されると、項目が変わる。そこはマニュアルを参照にする。

⑥license.dat ファイルの追加を行う。license.dat ファイルを/opt/pgi/ディレクトリに置くこと。

※ライセンスキーの発行は PGI 社のホームページ上で行う。
https://www.pgroup.com/account/login.php?redirect
※インストールが完了しないと lisence.info ファイルにある FLEXnet hostid と Hostname が分からないため、ライセンスキーの発行ができない。
※ライセンスキーが発行されたら license.dat ファイルを自動生成してくれる。
※試用版は https://www.pgroup.com/webuser/login.php からログインして，Create trial keys をクリックする。FLEXlm hostid（lisence.info ファイルに書いてある）を入力すると，lisence.dat ファイルを自動作成へのリンクをクリックすると自動的に作成され，ダウンロードされる。

⑦PGI compiler のパスを通す。いちいちコマンドラインで打ち込んでもよいが、再起動時にまた打ち込まなければいけないので、あらかじめ.bashrc ファイルを書き換えた方が便利。
使用するユーザのディレクトリにある.bashrc ファイルの最後に以下を追加すれば、ログイン時に自動的に読み込まれる。なお、root の.bashrc ファイルを書き換えることは万が一のことを考えると推奨しない。
以下は release7.2，64bit 版の場合だが、バージョンが変わっても数字が変わるだけである。

```
export PGI=/opt/pgi
export PATH=$PGI/linux86-64/7.2/bin:$PATH
export PATH=$PGI/linux86-64/7.2/mpi/mpich/bin:$PATH
export MANPATH=$MANPATH:$PGI/linux86-64/7.2/man
export LM_LICENSE_FILE=$PGI/license.dat
```

※ここは間違えないで書くこと。大文字、小文字、誤字、脱字などに気をつける。場合によってはログインできなくなる。root でやってしまったら恐ろしい。
※一緒に英語環境になるよう設定してしまうのも手。以下をさらに追加する。

```
export LANG=en_US↵
```

⑧パスを通すために一度ログアウトして，再びログインする（.bashrc ファイルを書き換えていることが前提）。

⑨言語環境を確認。必要に応じて元に戻す。

⑩最後にコンパイラの確認。コマンドラインで以下のようにする。起動するとバージョンなどの情報が表示されれば問題なし。

```
$ pgCC –dryrun –V x.c↵
```

⑪ここまで来ればインストール作業は終わりであるが、ライセンスサーバ・デーモン(lmgrd)を起動するする必要がある。自動起動させるには管理者権限で以下のようにする。OS を再起動すれば起動されているはず。

```
#   cp $PGI/linux86-64/7.2/bin/lmgrd.rc   /etc/rc.d/init.d/lmgrd-pgi
#   /sbin/chkconfig --add lmgrd-pgi                Fedora ではパスが通っていないため/sbin を入れる
                                                   ※Fedora11 では/sbin は不要
```

⑫ライセンスマネージャの設定と起動に関しては以下のホームページが参考になる。

▽株式会社ソフテック（http://www.softek.co.jp/SPG/Pgi/versionup.html#006）

ライセンスマネージャを手動で起動する場合は以下のようにする。

```
# /etc/rc.d/init.d/lmgrd-pgi start↵
```

⑬Fedora11 においてライセンスマネージャが起動しない事態となった。license.dat ファイルで指定されたホスト名が unknown とされることが原因。/etc/hosts ファイルを開き，指定されたホスト名を追加することで解決した。

PGI コンパイラは購入から 1 ヶ月間初期サポートしてもらえる。今回は大変助かりました。

(2)コンパイル・オプション

test1.cpp をコンパイルし、test2 という実行ファイルを作成する場合を例に説明する。2GB 以上のメモリを必要とする場合は gcc と同様に"–mcmodel=medium"を指定する。

コンパイル・オプションについて以下のページを参考にした。説明を鵜呑みにしないで、オプションをつけたら試すが鉄則。なぜなら、これまで問題なく計算できていたプログラムが、オプションを追加すると計算できなくなることもあるからである。ver.7.2 以上であれば以下の内容は適用されるはず。

▽PGI コンパイラの使用法（オプションの使用）
(http://www.softek.co.jp/SPG/Pgi/comp-tips.html)
その中でも以下が一覧になっているので見やすい。
▽PGI コンパイル・オプション一覧（http://www.softek.co.jp/SPG/Pgi/TIPS/option1.html)
▽ビジュアルテクノロジー：技術コラム
(http://visualtechnology.jp/jp/topics/column/tech_column/)

PGI コンパイラの使い方の基本は以下の通り。C と C++で違う。gcc の使い方も参考になる。

```
$ pgcc  –o test2 test1.cpp   （C 言語）
$ pgCC  –o test2 test1.cpp   （C++）
```

これをベースにコンパイラ・オプションを追加していく。ここで説明するのは以下の３つ。
①プロセッサに対する最適化
②自動並列化
③性能の最適化（gcc の-O2、-O3 オプションに相当）

以下にそれぞれの説明を述べるが、結局は組み合わせて使う。

①プロセッサに対する最適化

```
$ pgCC –mcmodel=medium –tp core2-64 –o test2 test1.cpp↵
```

-tp：ターゲットプロセッサタイプの指定。PGI6.2 以降の場合は以下のようにする。
Core2Duo　→　core2（64bit の場合は core2-64）
Core2Quad　→　penryn（64bit の場合は penryn-64）

※64bit か 32bit かはインストールされている OS の環境による。64bitOS で"core2"、"penryn"を指定してもエラーが出てコンパイルしない。

②自動並列化

マルチコア CPU で計算する以上、並列化はとても魅力であるが、最も計算速度が上がるオプションの組み合わせを自分で試行錯誤する必要がある。

並列化の効果は当然であるが、プログラムに依存する。私のプログラムではフラグなしで実行しても余り並列化の効果がなかったが、オプション・フラグをつけると効果は劇的に上がった。ただし、計算速度が速くても正しく計算されていなければ意味はないのだが。

コア数が増えるほど、計算速度が速くなるのが理想だが、両者に線形性があることは希のようだ。実行前に CPU 数を指定することを忘れるとシングルコアで動作する。

$ pgCC –mcmodel=medium –tp core2-64 –Mconcur –Minfo –o test2 test1.cpp↵

-Mconcur：自動並列化（-Mconcur=option1,option2,･･･として以下のオプションの指定ができる。上記のようにフラグなしでも使用は可能。ただし、フラグなしでは私の場合はほとんど効果がなかったので、デフォルトでオフになっている”cncall”と”innermost”を追加した。）
dist:[block/cyclic]：デフォルトは block。ブロック分割とサイクリック分割を選択。
cncall：デフォルトは不許可。サブルーチン、関数を含むループが安全であることを知らせる。
innermost：デフォルトは不許可。再内側ループの並列化を許可する。

-Minfo：並列化のインフォメーションを表示する（省略可）

※プログラム実行前に以下のように CPU 数を指定すること（Core2Duo の場合は 2、Core2Quad の場合は 4)。ちなみにコアを２つ使う場合，CPU 時間は２倍でカウントされている。
$ export NCPUS=2↵
$ ./test2↵

③性能の最適化

$ pgCC –mcmodel=medium –tp core2-64 –fast –Mconcur –Minfo –o test2 test1.cpp↵

-fast：-O2 –Munroll=c:1 –Mnoframe –Mlre と同等
-fastsse：-fast –Mvect=sse –Mscalarsse –Mcache_align –Mflushz と同等(SSE 使用)
-O3：-O2 を上書きするために-fast の後につける。-O2 よりもアグレッシブに最適化する。

また、-fast または-fastsse のあとに以下のオプションがつけられる。
-fastsse –Mipa=fast,inline　：IPA 最適化を有効、自動インライン化を有効。
-fastsse –Minline=levels:10 --no_exceptions：明示的に自動インライン化を有効。

以上述べた①～③を組み合わせると以下のようになる。これ以外でも最適なオプションがあるかもしれないが、オプションが多すぎて試していないものがほとんどである。プログラムによってオプションを追加、削除することになると思うが、色々と試してみるとよいだろう。

$ pgCC –mcmodel=medium –tp core2-64　–Mconcur=cncall, innermost –Minfo -fastsse –Minline=levels:10 --no_exceptions –O3 –o test2 test1.cpp

デュアルコア以上の PC において、②で述べた並列化の効果がなければ、シングルコアで複数のプログラムを動作させる方がお得である。この場合、単純に PC のスペックに依存する。となれば、FSB が高い CPU と M/B で構成され、なおかつ CPU のクロックが高い PC が有利になる（あとはコストパフォーマンスの問題になるだろう)。