ホームページでの公開

フリーな翻訳環境の準備とオープンソース利用のススメ
－ MacOSX 10.2の場合 －

芝池英樹（京都工芸繊維大学）

Hideki.Shibaike@Dad.Kit.Ac.Jp

http://Shibaike.Dad.Kit.Ac.Jp/

オープンソースとは？

- オープンソースソフトウェア／フリーソフトウェア（OSS/FS）は次のようなライセンスを持つ．
  - どんな用途にも使える，
  - 誰でも修正できる，
  - オリジナルも修正版も自由に再配布できる．

http://oss.mri.co.jp/reports/wheeler/oss_fs_why.html

オープンソースでの公開のための状況分析と必要条件

- フリーなコンパイラ環境が存在することが望ましい.
- 潜在的にはFortranソースの蓄積が質・量共に卓越しており期待出来る.

  → GNUのFortranコンパイラが利用可能.
- グラフィック処理・GUI利用では，Object指向言語で記述されたソースが好ましい.

  → Javaが利用可能.

各種プラットフォームと GCC & Java

	GCC (C, C++, Fortran)	Java (J2SE)	Supplement
Solaris 8.X	v 3.3.2	v 1.4.1_01	-
Linux	v 3.3.2	v 1.4.1_01	-
MacOSX 10.2	v 3. 1	(v 1.3)	Project Builder
Windows	v 3.3.2	v 1.4.1_01	Cygwin or MinGW

GCC: http://gcc.gnu.org/
Java2 Standard Edition: http://Java.Sun.Com/j2se/1.4/ja/
Cygwin: http://cygwin.com/
Minimalist GNU For Window: http://www.mingw.org/

Apple's developer tools

- Apple's developer tools (Dec2002DevTools)を先ずインストール.
  - GCC3.1， Project Builder(統合開発環境), Interface Builder（ユーザインターフェイス開発ツール）などが含まれる.

    http://developer.apple.com/techpubs/macosx/DeveloperTools/devtools.html
  - 会員登録（無料）すれば，会員エリアでダウンロード可能（無料，約301MB）.

    http://connect.apple.com/
  - CDイメージ内Developer.mpkgのダブルクリックでインストールは完了.

RPMのインストール#1

参考：http://www-jlc.kek.jp/~fujiik/macosx/10.2.X/memo/RPMonX.html

- Root ユーザを利用可能にする.
- TarファイルからRPMをインストール.

```
# su
# gnutar -zxvf rpm-4.0.2-0.34e-bin.tar.gz -C /
# setenv PATH $PATH¥:/usr/local/bin
# rpm --initdb
```

http://www-jlc.kek.jp/~fujiik/macosx/10.2.X/tgz/rpm-4.0.2-0.34e-bin.tar.gz

RPMのインストール#2

- 以下のRPMファイルをインストール.

```
# rpm -Uvh system-10.2-1b.ppc.rpm --nodeps
# rpm -Uvh gettext-0.10.35-4c.ppc.rpm
# rpm -Uvh *-1.0.2-5b.ppc.rpm
# rpm -Uvh db3*-3.2.9-2b.ppc.rpm
```

- 多くのパッケージで必用となるので,
  以下のRPMファイルもインストール.

```
# rpm -Uvh dlcompat-20021001-1a.ppc.rpm
# rpm -Uvh readline*-4.3-3a.ppc.rpm
```

http://www-jlc.kek.jp/~fujiik/macosx/10.2.X/RPMS/ppc/

MacOSX 10.2 へのG77の導入

- G77は除外されているので，追加インストールが必用.
- コンパイル済みバイナリーファイル（RPM形式）が利用可能.
- インストールは以下の手順による.

```
# rpm -ivh gcc-with-g77-1175-1a.ppc.rpm
# rpm -ivh gcc2-with-g77-7-1a.ppc.rpm
# rpm -ivh gcc-with-g77-symlinks-1175-1a.ppc.rpm
```

http://www-jlc.kek.jp/~fujiik/macosx/10.2.X/RPMS/ppc/

http://www-jlc.kek.jp/~fujiik/macosx/10.2.X/memo/G77onX.html

G77利用上の注意事項

- MacOSXではオプション-fPICが暗黙に指定されるので，明示的指定は不要.
- 現状では”implicit none”は受け付けられないので，コメント行にする必用がある.
- Complex型文関数を利用している場合は，オプション-O0指定する必用がある.
- 分割コンパイル時はオプション-Wallを指定.
- G77でよく利用されているFCFLAGSの一例.

  FCFLAGS = -fno-automatic -fno-second-underscore -fno-f90 -fugly-complex -fno-globals -fugly-init -Wno-globals

  http://gcc.gnu.org/onlinedocs/g77/

C/Java演算のベンチマーク

- C/Javaプログラムでの数値演算速度差のベンチマーク結果が公開されている.
- 「10000*10000回」繰り返し演算して処理時間を計測.
  - カラの関数の呼び出し，int型の加減乗除算とシフト，float型の加減乗除算，double型の加減乗除算，cosの呼び出し，sqrtの呼び出し，３次元ベクトルの単位ベクトルの計算.
- テストされたマシン環境；
  - Windows2000：Athlon 1100MHz，Mem 512MB
  - OS X10.2.2：eMac(G4 700MHz，Mem 384MB)

http://www001.upp.so-net.ne.jp/y_yutaka/labo/math_algo/calcbench.html

主なベンチマーク結果（実数）

倍精度浮動小数点演算の比較

環境	倍精度浮動小数点加算	倍精度浮動小数点減算	倍精度浮動小数点乗算	倍精度浮動小数点除算
Win : VC++ 最適化なし	1031 ms	1262 ms	1001 ms	1863 ms
Win : VC++ 最適化あり	361 ms	370 ms	361 ms	1001 ms
Win : BCC 最適化なし	2083 ms	2093 ms	1903 ms	2814 ms
Win : BCC 最適化あり	1913 ms	1903 ms	1913 ms	2543 ms
Win : Java	2264 ms	1962 ms	1963 ms	2544 ms
OSX : gcc3.1 最適化なし	7440 ms	7450 ms	7430 ms	7460 ms
OSX : gcc3.1 最適化あり	290 ms	280 ms	290 ms	290 ms
OSX : CW8.3 最適化なし	4040 ms	4030 ms	4020 ms	4040 ms
OSX : CW8.3 最適化あり	430 ms	440 ms	430 ms	440 ms
OSX : Java	6256 ms	6983 ms	6980 ms	7002 ms

数値演算関数の呼び出しの比較

環境	三角関数(cos)	平方根(sqrt)	三次元ベクトルを単位ベクトル化
Win : VC++ 最適化なし	13800 ms	4897 ms	9804 ms
Win : VC++ 最適化あり	9394 ms	2183 ms	9443 ms
Win : BCC 最適化なし	11066 ms	3144 ms	14491 ms
Win : BCC 最適化あり	12869 ms	5758 ms	16404 ms
Win : Java	31295 ms	2273 ms	13500 ms
OSX : gcc3.1 最適化なし	27110 ms	29980 ms	43830 ms
OSX : gcc3.1 最適化あり	23420 ms	26940 ms	28280 ms
OSX : CW8.3 最適化なし	24900 ms	27690 ms	35080 ms
OSX : CW8.3 最適化あり	23580 ms	26980 ms	29170 ms
OSX : Java	57057 ms	27282 ms	41198 ms

ベンチマーク結果の特徴#1

- C言語では，OS環境や最適化の有無に関係なく，**「関数呼び出し」に時間がかかる**.
- Javaでは逆に，整数の四則演算よりも関数呼び出しが早い.
- OS Xでは，浮動小数点演算が加減乗除算に関係なく，処理速度は安定している.
  - 最適化した場合，整数の加減乗除算・シフト演算も，処理速度が安定している.
  - これは最適化によりループがまとめられた結果かもしれない.

ベンチマーク結果の特徴#2

- 全体的に数値演算関数（cos/sqrtなど）の実行にかなりの時間を消費している。
  - **数値演算関数の処理自身がボトルネックになりうる**
  - 特にOS Xでの平方根(sqrt)の計算が顕著.
- **特にWindows環境では，除算が重い.**
- 単・倍精度の浮動小数点演算の速度差はあまりない.

速度アップのポイント

- 関数の呼び出しは極力控える.
- 除算は，乗算やシフトで可能な限り代用し，極力使用を避ける.
- 整数演算でできる部分は整数で(OS X環境では大差無し).
- 数値演算関数は，ルックアップテーブルなどを使うようにして，頻繁な呼び出しを避ける.

オープンソース化への課題
2 次元境界要素計算パッケージ（Be2fcm）のソース公開に当たって

- オープンソースな数学・科学技術計算パッケージ（ Netlib等）利用への切替.
  - 連立方程式計算，各種数学関数等.

  http://www.netlib.org/master_counts2.html
- 入会地の悲劇の回避.
  - 著作権の保護と適切な引用の確保.
- 公開性vs計算効率
  - ソースコードの二重化

まとめ

- オープンソースなライブラリィ公開が重要.
- オープンソースのインフラとしてフリーな翻訳環境や開発環境も必要.
- フリーなインフラとしてはGCC(C,C++,F77)とJavaが有効.
- MacOSX 10.2上でのGCC(C,C++,F77) の導入方法を解説.
- WindowsとMacでのJavaとCの計算性能のベンチマーク結果を紹介.
- オープンソース実現のための問題点を整理.