TP : Modèles du hasard & Parallélisation de flux stochastiques

Confrontation avec une « vraie » bibliothèque – CLHEP

Mail : bibliothèque CLHEP, code de calcul intégral, quasi-aléatoire

But principal : se confronter à une bibliothèque de taille professionnelle pour la simulation distribuée : CLHEP

1) Installer la bibliothèque (observer la hiérarchie des répertoires, où sont les sources, les include, les codes de test, regarder les nouveaux répertoires qui sont créés, la création des bibliothèques de leurs chemins d'accès…). Tester cette bibliothèque avec le code C++ donné en fin de TP.

2) Archiver quelques statuts pour le générateur Mersenne Twister 1997 séquences d'une longueur donnée (10 nombres par exemple). Tester la restauration à un statut donné et réfléchir et proposer une façon de lancer **en parallèle** les 10 réplications d'une simulation de Monte Carlo sur un multi-cœur ou sur grille avec des flux indépendants correspondants.

3) Faire un petit code de test de la méthode de « Sequence splitting » (découpage des séquences tous le N nombres avec N très petit) avec le générateur Mersenne Twister.

4) Optionnel pour ceux qui sont rapides : re-documentez vous sur les simulations de Monte Carlo (wiki), puis sur les générateurs quasi-aléatoires

   Essayer le code donné par mail pour un calcul de PI sous X windows (avec ou sans le graphisme) puis le code tel quel :

   a. avec le générateur de base
   b. avec le générateur (quasi) Monte Carlo fourni (attention penser à l'utiliser en 2D)
   c. avec le générateur Mersenne Twister
   d. avec une grille déterministe d'une précision donnée

   Comparer les résultats
   Dans quels cas avez-vous besoin de faires des réplications ?

TIPS : Partir de la version CLHEP modifiée par Romain Reuillon (Random.tgz - CLHEP)

```
./configure --prefix=$PWD                    avec PWD répertoire d'installation
make –j16                          le –j16 exploite le parallélisme 16 cœurs par ex
make install
```

Compilation des programmes : Attention à la version du g++. Regarder la structure et l'emplacement des fichiers installés et ajuster le chemin des includes – I et celui de la bibliothèque – L. Exemple de code et de

```
CLHEP_DIR=chemin vers le répertoire d'installation

g++   -o testRandom testRandom.cc
      -I$CLHEP_DIR/Random/include
      -L$CLHEP_DIR/Random/lib
      -lCLHEP-Random-2.1.0.0
```

Au besoin ajouter -static pour inclure la librairie dans l'exécutable ou avant le lancement du programme compilé ajouter le chemin vers les librairies dans la variable d'environnement LD_LIBRARY_PATH

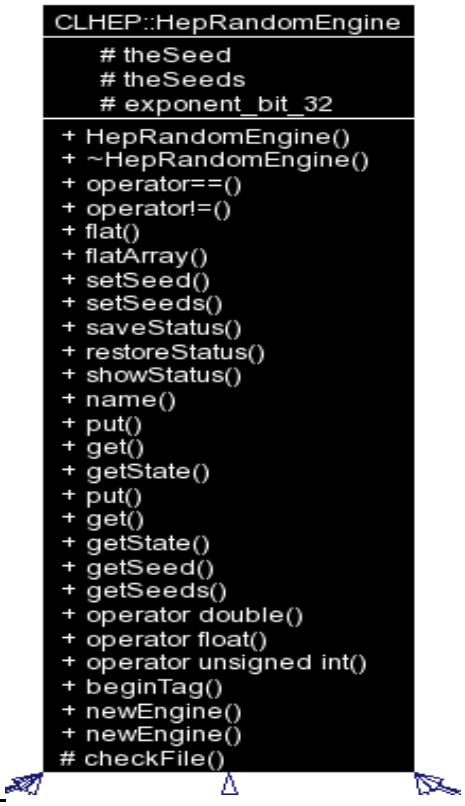
NB – Générateurs : SFMT :

Le Dr. Saito ancien doctorant de Matsumoto a proposé avec lui un générateur avec de meilleure qualité statistique que MT, avec dans certaines versions des périodes à faire passer MT pour un vulgaire LCG sur 16 bits ((2^{216091})-1), et une vitesse de génération deux fois plus rapide sur la plupart des architectures (2006).

URL : http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html

Il a également proposé récemment des générateurs MT pour GP-GPU : MTGP (2010) et TinyMT (2011)

Super classe



Pour tirer un nombre utiliser la méthode : flat()
Pour sauver un statut : saveStatus()

Sous classes : Il est possible de générer des status pour les générateurs suivants

JamesRandom, DRand48Engine, DualRand
Hurd160Engine, Hurd288Engine,
MTwistEngine
NonRandomEngine
RandEngine
RanecuEngine
Ranlux64Engine
RanluxEngine
RanshiEngine
TripleRand

Exemple de code CLHEP :

Objectif générer des nombres dans un fichier binaire
(pour test avec le logiciel DIE HARD de G. Marsaglia)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <limits.h>
#include <unistd.h>

#include "CLHEP/Random/MTwistEngine.h"

int main ()
{
   CLHEP::MTwistEngine * s = new CLHEP::MTwistEngine();

   int          fs;
   double       f;
   unsigned int nbr;

   fs = open("./qrngb",O_CREAT|O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR);

   for(int i = 1; i < 3000000; i++)
   {
     f = s->flat();
     nbr = (unsigned int) (f * UINT_MAX);

     //printf("%f\n", f); ou mieux cout << f << endl;

     write(fs,&nbr,sizeof(unsigned int));
   }

   close(fs);

   delete s;

   return 0;
}
```

TIPS: Pour l'ajout d'un générateur :

Le résultat multiplicatif intermédiaire peut dépasser 32 bits, il faudra penser à utiliser des long long

Ceci peut poser un problème à la compilation (ANSI ISO C99), il faut alors au besoin modifier les fichiers configure.in – Makefile.am et prendre en compte ces modifications par un boostrap (./bootstrap)

Pour toute prise en compte des modifications, penser à supprimer les bibliothèques avec un rm –fr lib* ou plus sagement la commande make adaptée et exécutée dans le bon répertoire. Ensuite vous pouvez relancer le make –j16 et make install.