IDM et Simulation TP 2

Ingénierie des modèles & Simulation

But du TP: Tester une technique de méta-programmation (génération de code par un autre programme). Application simple à une simulation de Monte Carlo (génération des valeurs pseudo-aléatoires) et évaluer les performances. Le début du TP repart des acquis en 2nd année (introduction à la simulation aléatoire à événements discrets).

1. *Retrouver dernière version du code source en C du générateur « original » Mersenne Twister de Makoto Mastumoto sur sa « Home page » (nommé MT ci-après). Aller dans ses anciennes pages (avant 2010) jusqu'à trouver (MT – Explanations & C code – la dernière version plus efficace 2002 – implémentation en mars 2004). Compiler ce code et comme en 2nd année (re-)**tester que le programme est bien porté. Les sorties doivent être conformes à ce qui est annoncé par l'auteur qui donne un exemple du résultat attendu (fichier .out) - c'est aussi le principe de portabilité lié cette fois à la reproductibilité scientifique.** Le README de Matsumoto explique que le fichier '.out' contient les résultats que l'on doit obtenir pour montrer que le générateur a été bien porté et que l'on reproduit bien la même trace.*

2. Utiliser ce générateur pour produire un (des) code(s) source(s) en C qui écrit simplement une déclaration et une initialisation de tableau de N nombres pseudo-aléatoires, puis le code boucle pour produire un nombre par ligne (puis une virgule).

```
float tabMT-1[] = {
    premier nombre tiré,
    deuxième ombre,
    etc.,
    …
    } ;
```

3. *Reprendre un code de simulation de Monte Carlo pour calculer PI (cf. ZZ2 - à retrouver sur le Net ou à reprogrammer rapidement).* Tester ce code en faisant des appels au générateur MT en faisant 10 réplications séquentielles avec 1 000 000 de points (10^7 points au total). Mesurer le temps de calcul (avec le time Unix) et donner l'écart moyen avec la constante M_PI.

4. Reprendre le code du point 3 et intégrer le code généré au point 2 pour 10 millions de points (compilation séparée) – utiliser les nombres stockés dans le tableau plutôt que de réaliser les tirages en appelant le générateur (optimisation par dite par « lookup table »). La compilation séparée du code qui déclare le tableau est conseillée (vous pouvez néanmoins tout mettre dans un même source). Noter le temps de compilation et d'édition des liens. Attention : le tirage chaque point nécessite 2 nombres pseudo-aléatoires. La zone de swap doit être assez grande pour cette étape de compilation.

5. Comparer les performances en temps machine entre : (1) entre tirer des nombres pseudo-aléatoires avec MT et (2) lire ces nombres pré-calculé dans le tableau généré (prendre un nombre significatif de tirages dans les limites de la taille possible pour une compilation – zone de swap spécifique qui pourrait être étendue).

6. Ajouter un facteur 1000 et mesurer également le temps de calcul (soit 2.10^9 points) en effectuant réellement les tirages. Pourrez-vous utiliser de tableau pré-calculé pour ce milliard de points. Quels seraient les problèmes / contraintes si l'on souhaite utiliser une technique de méta programmation pour la question (générant 20 milliards de nombres) ?

7. Pour ceux qui ont du temps et qui souhaitent découvrir une technique logicielle utilisant au mieux le matériel. Au lieu de générer un code source avec un tableau de nombres issus de MT, on peut écrire ces nombres dans un fichier (binaire) et étudier ou redécouvrir la technique dite du « memory mapping » pour adresser le fichier binaire comme s'il s'agissait d'un tableau en mémoire. Pour le contexte que nous venons de voir aux points 6) et 7) est-ce que cela règle / lève des contraintes. (https://linux.die.net/man/3/mmap)