POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 4A (S7) et 3A (S6) **Programmation Avancée 2010/2011**

TP à la découverte de GCC

Objectifs/Instructions

Découvrir certaines caractéristiques de gcc, certaines options de compilation ainsi que les notions de pile et de cache. Intégrer des commandes d'autres langages dans un programme C.

Avant de commencer, récupérez sur la page web de Laure Gonnord l'archive `tp_gcc.tgz`.

Doc La documentation GCC est très complète : http://gcc.gnu.org/

Un excellent survey sur l'optimisation « pratique » de programme C http://leto.net/docs/C-optimization.php

1 Allocation statique/dynamique

Compilez et exécutez les programmes `alloc_statique.c` et `alloc_dyn.c`.

- Comparez les temps d'exécution avec l'aide de la commande Unix `time` (man time).
- Augmentez la taille de la matrice dans `alloc_statique.c`. Que se passe-t-il ? Pourquoi ?
- Mêmes questions avec `alloc_dyn.c`.
- Modifiez le code de `alloc_dyn.c` en ajoutant une vérification de la valeur de retour de `malloc`.

2 Parcours en ligne/en colonne

Copiez le fichier `alloc_dyn.c` en `parcours_mat.c`. Comparez (sur de grandes tailles de matrices) les temps d'exécution des parcours ligne par ligne et colonne par colonne.

3 Les options d'optimisation de gcc

Reprenez le fichier `alloc_dyn.c` avec N=4000, ajoutez une boucle de façon à effectuer 100000 fois l'initialisation. Comparez les temps d'exécution en compilant sans option d'optimisation, puis avec l'option `-O2`, puis avec l'option `-O3` et enfin `-Os`. Comparez aussi les tailles des binaires. Allez voir dans la doc ce que fait l'option -Os.

4 Générer un exécutable à partir de plusieurs langages de programmation

Objectif de l'exercice : montrer qu'une fois compilé, il est possible d'utiliser du code écrit dans un autre langage (ici Fortran).

Préambule : Fortran est un langage de programmation trés populaire aupres des physiciens et mathématiciens. Un grand nombre de code industriels sont encore écrits en Fortran.

Exercice 1 Écrire un petit programme en C qui contient une fonction, qui est appelée dans votre main. Compiler votre programme en générant uniquement le fichier `.o`. Regarder l'aide sur la commande `nm`. Utiliser cette commande sur votre fichier objet.

Exercice 2 Soit le programme Fortran suivant (`exemple.f` dans l'archive)

```
program main
call mafonction
return
end

subroutine mafonction()
return
end
```

Compiler ce programme en générant uniquement le fichier .o (même syntaxe que pour le fichier C hormis qu'on appelle gFortran à la place de gcc). Utiliser nm sur le fichier objet créé. Commentaires sur les symboles générés ?

Exercice 3 Comme le montre l'exercice précédent, le compilateur Fortran rajoute systématiquement un underscore (_) à la fin des symboles de fonctions.

1. **Appeler du C dans un programme Fortran.** Soit le programme Fortran suivant (`essai.f` dans l'archive) :

   ```
   program main
   call foncc
   call foncFortran()
   return
   end

   subroutine foncc()
   call mafonction
   return
   end

   subroutine foncFortran()
   write(*,*)'Hello from Fortran'
   return
   end
   ```

   Écrire dans un fichier C, la fonction qui peut être appelée par le programme Fortran. Pour cette fonction, faites juste un affichage de "Hello from C"

   Pour compiler, il faut :

   ```
   gcc -c monFichier.c gFortran -o mixCF essai.f monFichier.o
   ```

   Exécuter le programme, que se passe-t'il ?

2. **Appeler du Fortran dans une fonction C qui est elle-même appelée par un programme Fortran** Dans le programme `essai.f` enlever la ligne `call foncFortran` et faites en sorte que cette fonction soit appelée dans votre fonction C (les questions 1 et 2 ont leur importance ici).

3. **Passage de paramètres.** Rappel : en C, le mécanisme par défaut de passage des paramètres est par valeur, alors que pour le Fortran c'est un passage par adresse. Pour forcer le passage par valeur en Fortran, on doit écrire `%VAL(nomdemavariable)`

   Modifier vos fichiers .f et .c pour que le programme Fortran appelle la fonction C en passant deux arguments (2 entiers) : un argument par adresse et un par valeur. Dans votre fonction C, vous afficherez la valeur des arguments passés.

   Indication : en Fortran la déclaration d'un entier se fait de la manière suivante `integer mavariable = 2`

   Modifiez les valeurs des arguments dans votre fonction C. Afficher le résultat en Fortran. Le résultat est-il conforme à vos attentes ?

4. **Réciproque**
   - Écrire un programme C qui appelle une fonction Fortran. Vous compilerez chacun des fichiers séparemment avant de générer l'executable à partir des deux fichiers objets.
   - Modifier votre programme pour tester le passage de paramètres.

5. **Avec du C++** Soit le fichier `fichCpp.cpp` suivant (dans l'archive aussi !) :

```
#include <iostream>

using namespace std;

  void mafonctionCpp(int *a) {
    *a=10.0;
    std::cout << "Hello from C++ " << std::endl
  }
```

   Compilez ce fichier avec `g++`, faites un `nm` sur le fichier .o. Que constatez vous ? Pour éviter ce phénomène (qu'on appelle le name `mangling`), il faut rajouter une directive dans le fichier cpp. Compilez le fichier `fichCpp2.cpp` et faites un `nm` dessus. La signature de la fonction est-elle conforme à vos attentes ?

   Modifiez ensuite votre fichier c pour que la `mafonctionCpp` soit aussi appelée.

Codes sources (section1)

```
/*allocation statique de matrice*/

#include <stdio.h>
#define SIZE 40

int main()
{
  int M[SIZE][SIZE];
  int i,j;

  for(i=0;i<SIZE;i++)
    {
      for(j=0;j<SIZE;j++)
        {
          M[i][j] = i+j;
```

```
        }
    }

  return 0;
}

#include <stdio.h>
#include <stdlib.h>
#define N 4000

/*allocation dynamique*/
int main(void)
{
  int i,j;

  /*une matrice est un tableau de tableaux*/

  /* allocation memoire*/
  int** m = (int**) malloc(N * sizeof(int));

  /* allocation memoire pour chaque sous-tableau */
  for (i = 0; i < N; i++)
    m[i] = (int*) malloc(N * sizeof(int));

  /*init de la matrice */

  for (int k = 10000)
  for (i = 0; i < N; i++)
    {
      for (j = 0; j < N; j++)
        m[i][j] = i+j;
    }

  /* desallocation memoire de chaque sous-tableau*/
  for (i = 0; i < N; i++)
    free(m[i]);

  /* desallocation memoire finale de m */
  free(m);

  return 0;
}
```