TP6 OSE – DVFS

Modulation dynamique de la tension et de la fréquence

La modulation de fréquence et de tension (DVFS) est mise en œuvre par le noyau Linux à travers le sous-système CPUfreq. CPUfreq implémente plusieurs politiques de modulation de fréquence, appelées gouverneurs, dont chacun a un objectif défini en termes de performance ou économie d'énergie. Dans ce TP, nous allons comprendre le fonctionnement des différents gouverneurs et déterminer à quels cas d'usage ils sont adaptés. Pour cela :

1. Nous commençons par vérifier que les options qui permettent le sous-système de modulation de fréquence (CPUfreq) sont actives (Voir les captures ci-dessous).

Pour cela, dans le répertoire de buildroot, faites un :

$ make linux-menuconfig

Si ces options ne sont pas actives, activez-les, et recompiler le noyau. (Vous pouvez garder la dernière version de noyau que vous avez compilée dans le TP précédent, dans mon cas la 5.3.0).

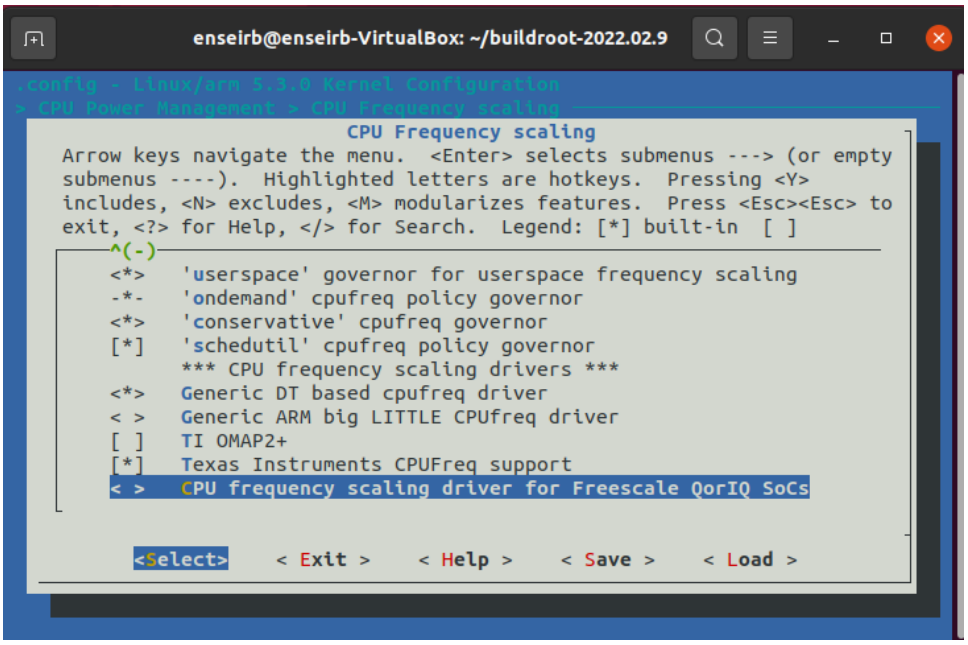

Figure 1 : Options à activer pour avoir CPUfreq

2. Vous devriez voir apparaître le répertoire */sys/devices/sysem/cpu/cpu0/cpufreq*. Explorez ce répertoire.

Dans le fichier *scaling_available_governors*, vous pouvez voir qu'il existe 6 gouverneurs possibles. Parmi eux, les gouverneurs powersave et *performance* sont statiques et permettent de fixer la fréquence respectivement au

minimum possible pour favoriser l'économie d'énergie, ou au maximum possible pour favoriser la performance. Les gouverneurs *conservative*, *ondemand* (et *schedutil)* prennent en compte l'utilisation du CPU (ainsi que la priorité des processeus) pour moduler la fréquence, de façon à ce que la fréquence soit élevée lorsque le CPU est utilisé intensivement et basse sinon. Enfin

Nous allons maintenant observer le comportement des différents gouverneurs dynamiques lors de l'exécution de différents types de tâches :

1. **Tâche intensive en calculs (CPU Bound) :** dans ce type de tâche, le processeur est fortement utilisé.
2. **Tâche intensive en mémoire (Memory Bound) :** ce type de tâche requiert l'utilisation de la mémoire pour stocker des structures lesquelles sont fréquemment accédées.
3. **Tâche intensive en entrées/sorties (I/O Bound) :** les périphériques sont fortement sollicités durant l'exécution des tâches de ce type.

Pour chacune des tâches, nous voulons mesurer le temps d'exécution pour chacun des gouverneurs, et observer l'évolution de la fréquence. Pour cela :

3. Nous devons d'abord créer une tâche dans chacun de ces types :

a) Tâche CPU bound : compléter le code primeNumbers.c qui compte le nombre de nombres premiers compris entre 1 et N.
b) Tâche memory bound : compléter le code prodScal.c qui calcule le produit scalaire entre 2 vecteurs V1 et V2, qui seront générés de manière aléatoire.
c) Tâche I/O bound : compléter le code writeFile.c qui écrit un certain nombre blocks de données dans un fichier. On utilisera par exemple la fonction *exec* qui recouvrira le programme avec l'exécution de la commande dd comme suit :
   dd if=/dev/urandom of=./fileName bs=4096 count=262144
   /dev/urandom est un fichier spécial qui contient des octets aléatoires générés par le noyau.

4. Compléter le code measureWorkloadWithGovernor.c qui prend en entrée un des gouverneurs Performance, Powersave, ondemand ou conservative ainsi que le chemin vers l'exécutable d'une des 3 tâches a), b) ou c) et leurs paramètres d'entrée respectifs. Cette tâche effectue le traitement suivant :
1. Fixer le gouverneur à utiliser selon le paramètre récupéré.
2. Le processus principal lance un processus fils qui s'occupe d'exécuter la tâche a), b) ou c) (en utilisant un *exec*), puis attend la terminaison de son fils. Le processus principal mesure et renvoie le temps d'exécution du processus workload (processus fils).

5. Créer un code ou un script (au choix) qui lance la *measureWorkloadWithGovernor* 10 fois et calcule les moyennes et l'écarts type des temps d'exécution des workloads (a, b ou c) lorsque chacun des gouverneurs sont utilisés.

6. Compléter le programme getFreqTimeSerie.c qui relève toutes les 1 us la fréquence du CPU et qui écrit la fréquence relevée dans un fichier sous la forme (timestamp de la mesure, fréquence). On s'inspire pour cela du TP 2.

7. Une fois que vous vous êtes assurés que vos programmes fonctionnent, compilez les pour la BeagleBone en utilisant le compilateur arm-linux-gcc qui se trouve dans *output/host/bin/*, et effectuez les mesures suivantes :

a) Lancer le script de mesure du temps d'exécution pour chacune des tâches avec chacun des gouverneurs (la question 5).

b) Lancer MeasureWorkloadWithGovernor (la question 4) en fixant à chaque fois le gouverneur utilisé à l'un des 3 gouverneurs dynamiques, et en les lançant en même temps que le programme getFreqTimeSerie (la question 6).
c) Récupérer le fichier des fréquence tracés sur votre VM, en utilisant le protocole tftp sur la carte comme suit :
tftp -p -r < nom du fichier > 192.168.1.2
d) Tracer (à l'aide de gnuplot, Excel, ou tout autre outil de votre choix) les fréquences obtenues avec chacun des gouverneurs pour chacune des tâches. Vous pouvez vous servir du script script_gnuplot.sh.

8. Comparez les temps d'exécution du chacune des tâches selon les différents gouverneurs.

9. Quelles peuvent être les conclusions tirées de ces mesures ?

Les groupes de contrôle (cgroups)

Le mécanisme des cgroups est une fonctionnalité du noyau Linux (intégrée depuis le noyau 2.6.24) qui permet de contrôler et isoler l'utilisation des ressources (processeurs, espace mémoire, utilisation disque, etc.) par groupe de processus.

Les cgroups fournissent plusieurs fonctionnalités : la limitation de ressources, la priorisation, en accordant plus de ressources processeur et bande passante à un groupe, la comptabilité en mesurant les quantités de ressources consommées par les processus, l'isolation des processus en restreignant l'accès aux fichiers à un groupe par exemple. Les cgroups sont utilisé notamment pour la virtualisation au niveau du système d'exploitation.

Pour pouvoir utiliser les différents contrôleurs, il faut monter le système de fichiers cgroup dans le répertoire prévu à cet effet.

1. Exécutez la commande suivante puis explorez le répertoire résultant ;
   mount -t cgroup -o all cgroup /sys/fs/cgroup
2. Identifier les fichiers permettant le contrôle de l'utilisation mémoire des processus. Dans la suite du TP, nous nous intéressons à cette utilisation des cgroup.

Nous allons, à présent, contraindre l'espace mémoire utilisable pour nos tâches. Pour cela nous allons créer le groupe auquel nous appliquerons les contraintes et ajouterons les tâches qui nous intéressent. Pour cela :

3. Créez un group à l'intérieur du répertoire cgroup : cela se fait simplement en créant un répertoire à l'intérieur du répertoire cgroup.
   mkdir nomDuGroupe
   Vous remarquerez que le contenu du répertoire cgroup est reproduit dans le groupe créé.
4. Appliquez une limite d'espace mémoire utilisable par les processus qui appartiendront à ce groupe. Pour cela, il faut modifier le fichier *memory.limit_in_bytes*. Par exemple, pour restreindre l'espace utilisable à 1 Mo, la commande est la suivante :
   echo 1M > nomDuGroupe/memory.limit_in_bytes
5. Nous allons maintenant réexécuter les tâches de comptage des nombres premiers, et de produits scalaires et observer l'impact de cette contrainte sur le temps d'exécution des tâches.
   a. Ajoutez aux programmes primeNumbers.c et prodScal.c les instructions qui permettent d'ajouter le pid de ces tâches dans le fichier *tasks*.
   b. Créez un script qui permet d'appliquer des contraintes mémoires de plus en plus grands au groupe créé, puis mesure le temps d'exécution de la tâche sous ces contraintes. Vous pouvez par exemple démarrer avec une limite de 512 Mo puis de diviser cette taille par 2 à chaque fois.
   c. Lancez ce script avec le comptage de nombre premiers. Que remarquez-vous ?

d. Lancez le script avec le produit scalaire. Que remarquez-vous ?
e. Que se passe-t-il lorsque la somme des tailles des vecteurs V1 et V2 dépasse l'espace mémoire utilisable ? Vous pouvez effectuer ce test en augmentant la taille des vecteurs d'entrée de sorte à dépasser la contrainte utilisée.