TP: MESURE DE PERFORMANCES

1. Objectifs

Comme l'indique le titre de ce TP nous allons allons apprendre de façon plus ou moins exhaustive comment mesurer les performances d'un système. Notre travail aura donc pour objectifs de:

- √ Découvrir quelques benchmarks classiques: un benchmark est une application ou suite d'application permettant de mesurer les performances d'un système pour le comparer à d'autres. On peut utiliser des microbenchs (petite application synthétique écrite pour l'expérimentation) ou des macrobenchs. Parmi les macrobenchs classiques nous pouvons citer PARSEC, SPEC CPU et KERNBENCH, qui sont ceux avec lesquels nous allons travailler.
- √ Collecter des métriques et comparer des environnements: une métrique est une donnée quantifiable permettant de mesurer la performance d'un système. Il peut s'agir du temps d'exécution d'une application, du nombre de cycles CPU, etc.
- √ Choisir de façon pertinente le benchmark à utiliser: en fonction de la ressource la plus utilisée ou la plus importante pour le système étudié, le type d'application à utiliser n'est pas le même. Il est capital de savoir quelles applications sont le plus impactées par un système. Ainsi on distingue des applications:

  ➢ CPU intensive: qui consomment beaucoup de CPU (SPEC CPU, PARSEC)

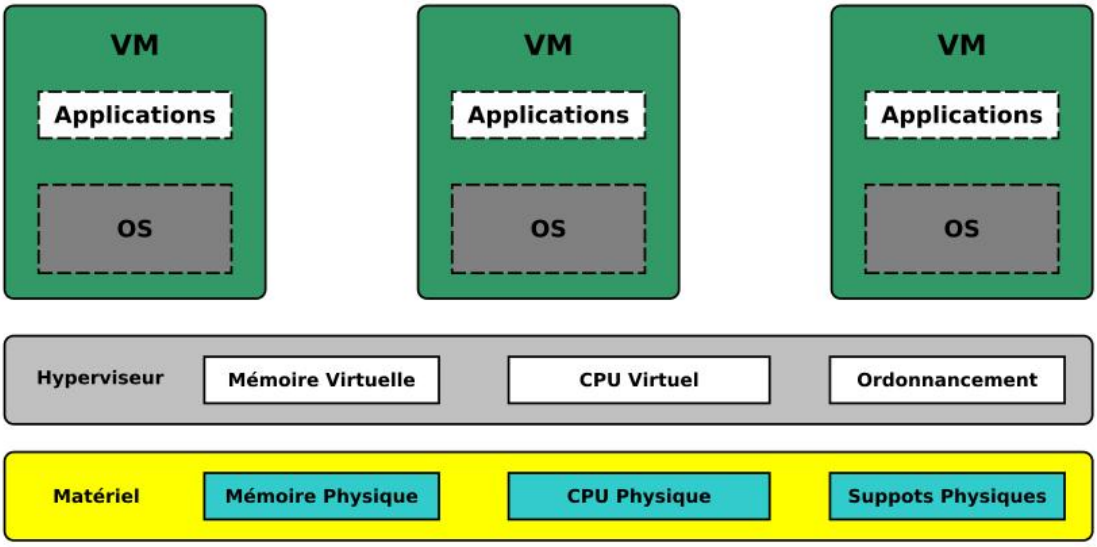
  ➢ Memory intensive: qui consomment beaucoup de mémoire (SPEC MPI)

- Network intensive: (Netperf, SPEC WEB)

2. Atteinte des objectifs

Nous allons mesurer les performances d'un environnement virtualisé en le comparant à un environnement natif ou bare metal.

La virtualisation est l'ensemble des techniques matérielles ou logicielles qui permettent de faire fonctionner simultanément sur une seule machine physique (machine hôte) plusieurs systèmes d'exploitation (OS) appelés Machines Virtuelles (VMs). L'architecture d'un environnement virtualisé est la suivante :



Le niveau de pénétration de la virtualisation est assez élevé, avec plusieurs organisations ayant un taux de virtualisation de leurs serveurs excédant les 75% [*Gartner Inc. Technologie de virtualisation et logiciels de VM*]. Cette utilisation de plus en plus importante des techniques de virtualisation dans les DCs doit son mérite aux nombreux avantages qu'elles offrent. Ceux-ci sont relatifs à :

- La réduction des coûts
- L'isolation et l'amélioration de la sécurité

- La minimisation des interruptions et l’amélioration de la disponibilité des services
- L’optimisation des performances
- La simplification des opérations de sauvegarde

(Plus de détails dans votre cours M2 sur la virtualisation).

3. Description et exécution des benchmarks

A. PARSEC

a. Description

PARSEC est une abréviation pour *Princeton Application Repository for Shared-Memory Computers*. C’est une suite de benchs composée d’applications multithread, permettant d’évaluer les systèmes multicores et multiprocesseurs.

La version actuelle de la suite (3.0) contient les 13 programmes suivants, issus de nombreux domaines tels que le codage vidéo, l'analyse financière, le traitement d'images, etc. :

- *Blackscholes*
- *Bodytrack*
- *Canneal*
- *Deadup*
- *Facesim*
- *Ferret*
- *Fluidanimate*
- *Freqmine*
- *Raytrace*
- *Streamcluster*

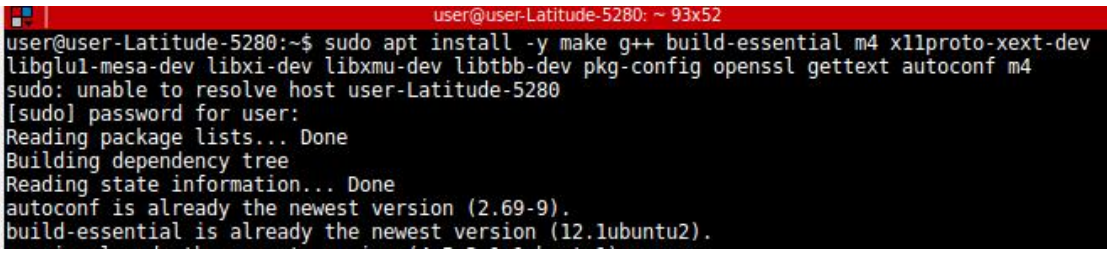
- *Swaptions*
- *Vips*
- *X264*

Pour plus de détails sur la suite et sur chaque programme : *https://parsec.cs.princeton.edu/.*

b. Exécution

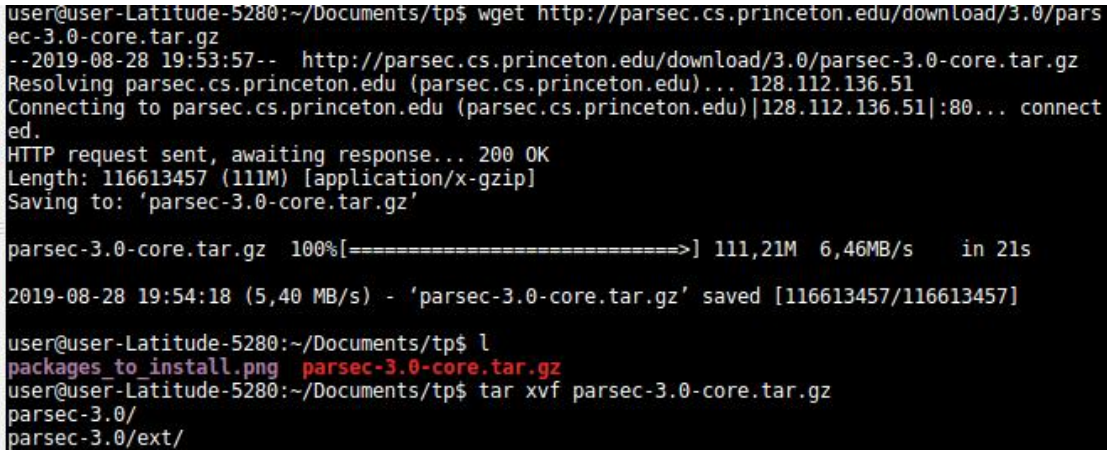
i. *Prérequis*

L'utilisation de PARSEC nécessitera un certain nombre d'outils et de packages (s'ils ne sont pas encore présents dans votre environnement de travail). Les installer comme sur l'image suivante :



ii. Téléchargement de PARSEC

Il faut maintenant télécharger PARSEC et décompresser le paquet (créer un dossier PARSEC et s'y déplacer) :



iii. Configuration

Renommer le dossier (en y enlevant la version), s'y déplacer et installer l'environnement :

```
user@user-Latitude-5280:~/Documents/tp$ mv parsec-3.0 parsec
user@user-Latitude-5280:~/Documents/tp$ cd parsec/
```

```
user@user-Latitude-5280:~/Documents/tp/parsec$ source env.sh
```

Nous pouvons maintenant utiliser l'outil *parsecmgmt* qui nous permettra d'exécuter les benchs :

```
user@user-Latitude-5280:~/Documents/tp/parsec$ parsecmgmt -a status -p all
[PARSEC] Installation status of selected packages:
[PARSEC]
[PARSEC] parsec.blackscholes [1]:
[PARSEC]    -no installations-
[PARSEC]
[PARSEC] parsec.bodytrack [1]:
[PARSEC]    -no installations-
[PARSEC]
[PARSEC] parsec.canneal [1]:
[PARSEC]    -no installations-
[PARSEC]
[PARSEC] parsec.cmake [1]:
[PARSEC]    -no installations-
[PARSEC]
[PARSEC] parsec.dedup [1]:
[PARSEC]    -no installations-
```

Pour obtenir l'aide sur la commande : *parsecmgmt -h*.

iv. "Building workloads"

Il s'agit ici de "construire" en quelques sortes les différents benchs avant de pouvoir les exécuter :

```
user@user-Latitude-5280:~/Documents/tp/parsec$ parsecmgmt -a build -p all
[PARSEC] Packages to build:  parsec.blackscholes parsec.bodytrack parsec.canneal parsec.cmake
 parsec.dedup parsec.facesim parsec.ferret parsec.fluidanimate parsec.freqmine parsec.glib pa
rsec.gsl parsec.hooks parsec.libjpeg parsec.libtool parsec.libxml2 parsec.mesa parsec.netdedu
p parsec.netferret parsec.netstreamcluster parsec.parmacs parsec.raytrace parsec.ssl parsec.s
treamcluster parsec.swaptions parsec.tbblib parsec.uptcpip parsec.vips parsec.x264 parsec.yas
m parsec.zlib splash2.barnes splash2.cholesky splash2.fft splash2.fmm splash2.lu_cb splash2.l
u_ncb splash2.ocean_cp splash2.ocean_ncp splash2.radiosity splash2.radix splash2.raytrace spl
ash2.volrend splash2.water_nsquared splash2.water_spatial splash2x.barnes splash2x.cholesky s
plash2x.fft splash2x.fmm splash2x.lu_cb splash2x.lu_ncb splash2x.ocean_cp splash2x.ocean_ncp
splash2x.radiosity splash2x.radix splash2x.raytrace splash2x.volrend splash2x.water_nsquared
splash2x.water_spatial

[PARSEC] [========== Building package parsec.blackscholes [1] ==========]
[PARSEC] [---------- Analyzing package parsec.blackscholes ----------]
[PARSEC] parsec.blackscholes does not depend on any other packages.
[PARSEC] [---------- Building package parsec.blackscholes ----------]
[PARSEC] Copying source code of package parsec.blackscholes.
[PARSEC] Running 'env version=pthreads /usr/bin/make':
rm -f blackscholes blackscholes.m4.cpp
/usr/bin/m4 ./c.m4.pthreads blackscholes.c > blackscholes.m4.cpp
```

Vous pourrez rencontrer quelques bugs ou erreurs :

➢ *Erreur smime.pod*

Elle se présente comme ceci :

```
smime.pod around line 272: Expected text after =item, not a number
smime.pod around line 276: Expected text after =item, not a number
smime.pod around line 280: Expected text after =item, not a number
smime.pod around line 285: Expected text after =item, not a number
smime.pod around line 289: Expected text after =item, not a number
POD document had syntax errors at /usr/bin/pod2man line 68.
make: *** [install_docs] Error 255
Makefile:680: recipe for target 'install_docs' failed
[PARSEC] Error: 'env PATH=/usr/bin:/home/josyanne/Documents/tp/parsec/bin:/home/josyanne/bin:
/home/josyanne/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games:/snap/bin:/home/josyanne/Documents/tp/parsec/bin /usr/bin/make install'
 failed.
```

Pour la corriger, il suffit d'utiliser le script suivant (cf. https://yulistic.gitlab.io/2016/05/parsec-3.0-installation-issues/) :

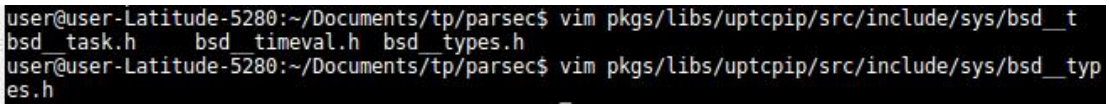
```
user@user-Latitude-5280:~/Documents/tp/parsec$ vim correct_smimebug.sh
user@user-Latitude-5280:~/Documents/tp/parsec$ chmod +x co
config/              correct_smimebug.sh
user@user-Latitude-5280:~/Documents/tp/parsec$ chmod +x correct_smimebug.sh
user@user-Latitude-5280:~/Documents/tp/parsec$ cat correct_smimebug.sh
#! /bin/bash
for i in 0 1 2 3 4 5 6 7 8 9
do
    echo "Replacing '=item $i' to '=item C<$i>'"
    grep -rl "=item $i" * | xargs sed -i "s/=item $i/=item C<$i>/g"
done
user@user-Latitude-5280:~/Documents/tp/parsec$ ./correct_smimebug.sh
Replacing '=item 0' to '=item C<0>'
Replacing '=item 1' to '=item C<1>'
Replacing '=item 2' to '=item C<2>'
Replacing '=item 3' to '=item C<3>'
Replacing '=item 4' to '=item C<4>'
Replacing '=item 5' to '=item C<5>'
Replacing '=item 6' to '=item C<6>'
Replacing '=item 7' to '=item C<7>'
Replacing '=item 8' to '=item C<8>'
sed: no input files
Replacing '=item 9' to '=item C<9>'
sed: no input files
```

➢ Erreur __mbstate_t

L'erreur s'affiche de la façon suivante :

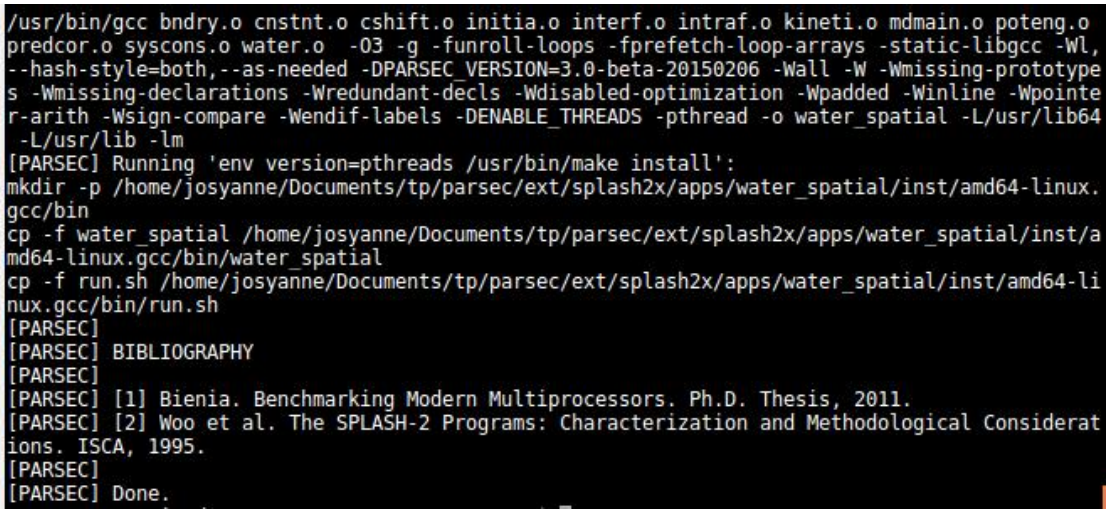
```
/usr/include/wchar.h:94:3: error: conflicting types for '__mbstate_t'
 } __mbstate_t;
   ^
In file included from ../include/machine/bsd_endian.h:37:0,
                 from ../include/sys/bsd_types.h:44,
                 from ../include/sys/bsd_param.h:64,
                 from if_host.c:48:
../include/sys/bsd__types.h:105:3: note: previous declaration of '__mbstate_t' was here
 } __mbstate_t;
```

Pour la corriger il faut ouvrir le fichier *bsd__types.h* dont le chemin est présenté sur la capture d'écran suivante :



Ensuite commenter la déclaration de *__mbstate_t* :

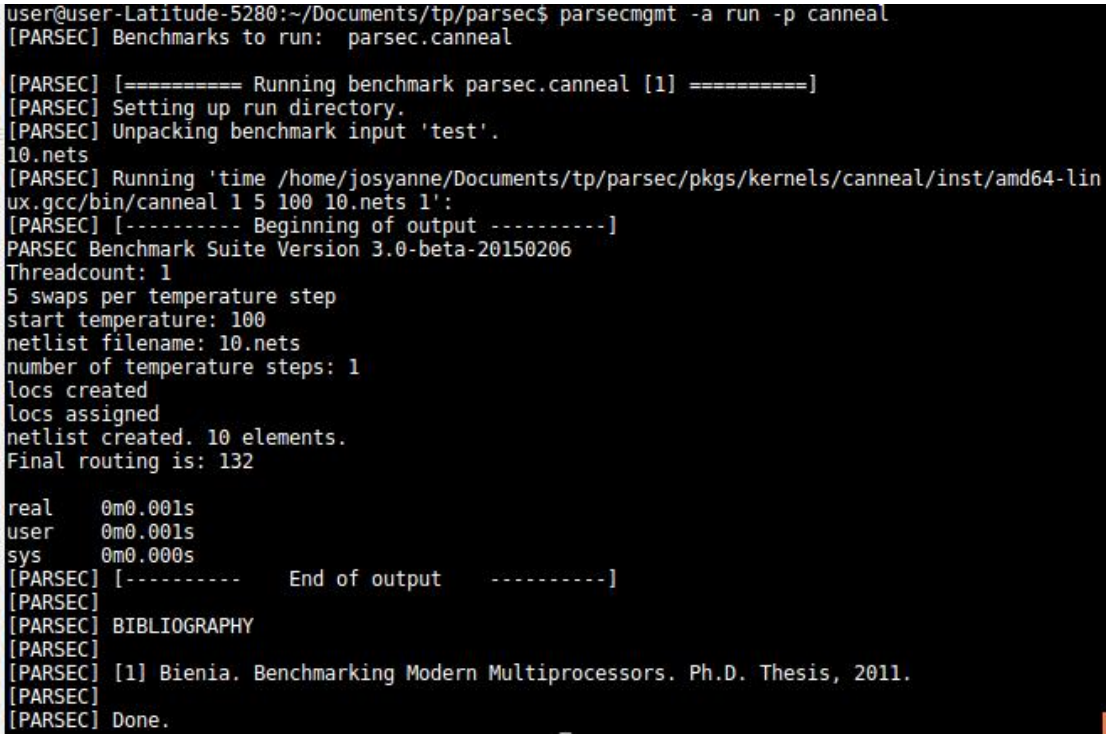
Vous pouvez ensuite refaire le build : *parsecmgmt -a build -p all* :



Il est également possible de configurer ("build") un package de façon indépendante (sans utiliser le *all*). Pour exemple la commande suivante permet de build le package canneal : ***parsecmgmt -a build -p canneal***.

v. "Running workloads"

Après le build on peut maintenant exécuter les benchs. La capture d'écran suivante montre comment exécuter le bench canneal :



Pour exécuter les benchs il est possible d'utiliser des ***inputs*** :

- ***Test*** : permet d'exécuter le programme avec une quantité de données aussi petite que possible
- ***Simdev*** : sollicite toutes les parties du système requises pour un jeu de données plus grand, avce le même chemin d'exécution que des entrées réelles
- ***Simsmall*** : semblable à des entrées réelles avec un temps d'exécution d'environ 1s
- ***Simmedium*** : semblable à des entrées réelles avec un temps d'exécution d'environ 5s
- ***Simlarge*** : semblable à des entrées réelles avec un temps d'exécution d'environ 15s
- ***Native*** : semblable à des entrées réelles avec un temps d'exécution d'environ **15min**

Pour obtnir ces inputs il faut les télécharger également :

wget http://parsec.cs.princeton.edu/download/3.0/parsec-3.0-input-sim.tar.gz

wget http://parsec.cs.princeton.edu/download/3.0/parsec-3.0-input-native.tar.gz

tar xzf parsec-3.0-input-sim.tar.gz

tar xzf parsec-3.0-input-native.tar.gz

N.B: il faut les décompresser à l'extérieur du dossier parsec, i.e. au même niveau d'arborescence).

Nous allons à présent exécuter le bench streamcluster avec l'input simlarge :

```
user@user-Latitude-5280:~/Documents/tp$ parsecmgmt -a run -p streamcluster -i simlarge
[PARSEC] Benchmarks to run:  parsec.streamcluster

[PARSEC] [========== Running benchmark parsec.streamcluster [1] ==========]
[PARSEC] Deleting old run directory.
[PARSEC] Setting up run directory.
[PARSEC] No archive for input 'simlarge' available, skipping input setup.
[PARSEC] Running 'time /home/josyanne/Documents/tp/parsec/pkgs/kernels/streamcluster/inst/amd
64-linux.gcc/bin/streamcluster 10 20 128 16384 16384 1000 none output.txt 1':
[PARSEC] [---------- Beginning of output ----------]
PARSEC Benchmark Suite Version 3.0-beta-20150206
read 16384 points

real    0m5.445s
user    0m5.440s
sys     0m0.004s
[PARSEC] [----------    End of output    ----------]
[PARSEC]
[PARSEC] BIBLIOGRAPHY
[PARSEC]
[PARSEC] [1] Bienia. Benchmarking Modern Multiprocessors. Ph.D. Thesis, 2011.
[PARSEC]
[PARSEC] Done.
```

Pour plus de détails :

https://parsec.cs.princeton.edu/download/tutorial/3.0/parsec-tutorial.pdf.

B. SPEC CPU

a. Description

SPEC (Standard Performance Evaluation Corporation) CPU est une suite de benchs qui exigent beaucoup de ressources processeur, et qui mettent à l'épreuve la mémoire et le compilateur d'un système. Cette suite a été conçue à partir d'applications réelles pour fournir une mesure comparative des tâches de calcul nécessitant beaucoup

de ressources.

SPEC CPU est une suite payante et la version que nous possédons et avec laquelle nous allons travailler est SPEC CPU 2006. Cette version comporte une trentaine de benchmarks parmi lesquels *perlbench*, *gcc*, *calculix*, *sphinx*, etc.

Pour mesurer les performances d'un système en utilisant les benchs de SPEC CPU 2006, nous prenons en compte 2 paramètres:

➢ *SPECspeed* : il s'agit de la mesure la vitesse à laquelle le système exécute une tâche unique

➢ *SPECrate* : il s'agit plutôt de la mesure du nombre de tâches que le système peut exécuter en un laps de temps donné. Il permet de mesurer le débit auquel un système exécute un nombre de tâches donné.

b. Execution

i. Prérequis

Il faudra installer au préalable les paquets ***make*** et ***gfortran***.

Créer un dossier SPEC_CPU et s'y déplacer. SPEC CPU est téléchargé sous la forme d'un fichier iso que nous allons par la suite monter vers un répertoire :

➢ ***sudo mkdir /media/specpu2006*** (création du répertoire vers lequel nous allons monter l'iso)

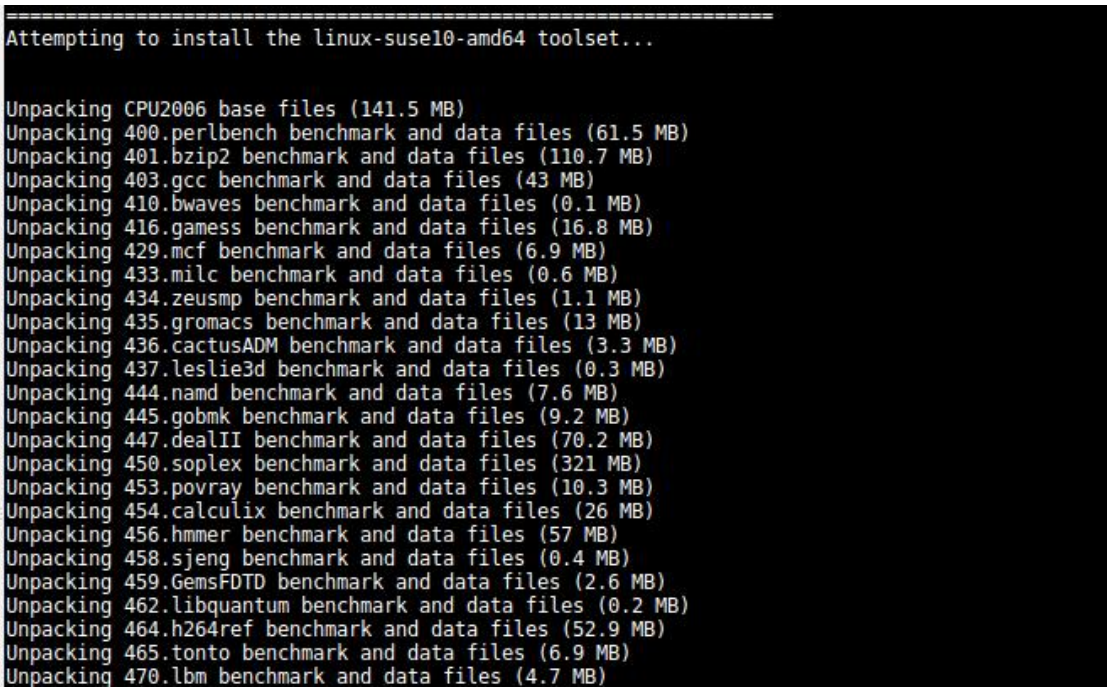
➢ ***sudo mount cpu2006-1.2.iso /media/specpu2006***

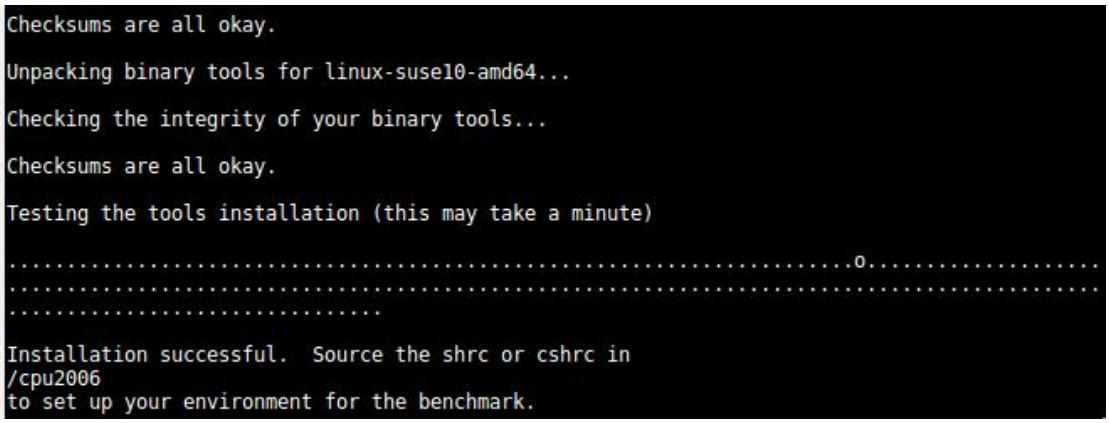
ii. Configuration et exécution

Tout d'abord il est bon de noter qu'il faudra exécuter toutes ces

commandes en mode root (sudo).

Se déplacer ensuite dans le répertoire ***/media/cpu2006*** précédemment créé et lancer le script d'installation avec la commande ***./install.sh*** :





Ensuite se déplacer dans le répertoire que vous avez spécifié pendant l'installation précédente (***/cpu2006***) et lancer le script shrc : ***. ./shrc*** *(bien remarquer qu'il y a 2 points)* ou alors ***source shrc,*** ensuite ***bin/relocate.***

Nous pouvons dès à présent configurer (build) un benchmark afin de pouvoir l'exécuter. Nous allons essayer avec le bench ***milc*** :

```
root@user-Latitude-5280:/cpu2006# runspec --config=Example-linux64-amd64.cfg --action=build --tune=base milc
runspec v6674 - Copyright 1999-2011 Standard Performance Evaluation Corporation
Using 'linux-suse10-amd64' tools
Reading MANIFEST... 22481 files
Loading runspec modules.................
Locating benchmarks...found 31 benchmarks in 6 benchsets.
Reading config file '/cpu2006/config/Example-linux64-amd64.cfg'
Running "specperl /cpu2006/Docs/sysinfo" to gather system information.
Loading "http://www.spec.org/auto/cpu2006/current_version" for version check: OK
Retrieving flags file (/cpu2006/config/flags/Example-gcc4x-flags-revA.xml)...
Retrieving flags file (/cpu2006/config/flags/Example-linux-platform-revA.xml)...
Benchmarks selected: 433.milc
Compiling Binaries
  Building 433.milc base gcc43-64bit default: (build_base_gcc43-64bit.0000)
```

```
LINK: /usr/bin/gcc   -O2 -fno-strict-aliasing  -DSPEC_CPU_LP64        <objects>   -lm        -o options
C: LD="/usr/bin/gcc"
O: COPTIMIZE="-O2 -fno-strict-aliasing"
P: PORTABILITY="-DSPEC_CPU_LP64"
C: MATH_LIBS="-lm"
C: LDOUT="-o options"

Build successes: 433.milc(base)

Build Complete

The log for this run is in /cpu2006/result/CPU2006.009.log

runspec finished at Fri Aug 30 15:49:39 2019; 5 total seconds elapsed
```

Après une configuration réussie, exécuter le bench comme sur la capture d'écran suivante :

```
root@user-Latitude-5280:/cpu2006# runspec --config=Example-linux64-amd64.cfg --size=ref --noreportable --tune=base --iterations=1 milc
runspec v6674 - Copyright 1999-2011 Standard Performance Evaluation Corporation
Using 'linux-suse10-amd64' tools
Reading MANIFEST... 22481 files
Loading runspec modules.................
Locating benchmarks...found 31 benchmarks in 6 benchsets.
Reading config file '/cpu2006/config/Example-linux64-amd64.cfg'
Running "specperl /cpu2006/Docs/sysinfo" to gather system information.
Retrieving flags file (/cpu2006/config/flags/Example-gcc4x-flags-revA.xml)...
Retrieving flags file (/cpu2006/config/flags/Example-linux-platform-revA.xml)...
Benchmarks selected: 433.milc
Compiling Binaries
  Up to date 433.milc base gcc43-64bit default


Setting Up Run Directories
  Setting up 433.milc ref base gcc43-64bit default: created (run_base_ref_gcc43-64bit.0000)
Running Benchmarks
  Running 433.milc ref base gcc43-64bit default
/cpu2006/bin/specinvoke -d /cpu2006/benchspec/CPU2006/433.milc/run/run_base_ref_gcc43-64bit.0000 -e speccmds.err -o speccmds.stdout -f speccmds.cmd -C -q

/cpu2006/bin/specinvoke -E -d /cpu2006/benchspec/CPU2006/433.milc/run/run_base_ref_gcc43-64bit.0000 -c 1 -e compare.err -o compare.stdout -f compare.cmd
Success: 1x433.milc
Producing Raw Reports
mach: default
  ext: gcc43-64bit
    size: ref
      set: int
      set: fp
        format: raw -> /cpu2006/result/CFP2006.010.ref.rsf
Parsing flags for 433.milc base: done
Doing flag reduction: done
        format: flags -> /cpu2006/result/CFP2006.010.ref.flags.html
        format: ASCII -> /cpu2006/result/CFP2006.010.ref.txt
        format: CSV -> /cpu2006/result/CFP2006.010.ref.csv
        format: HTML -> /cpu2006/result/CFP2006.010.ref.html, /cpu2006/result/invalid.gif, /cpu2006/result/CFP2006.010.ref.gif

The log for this run is in /cpu2006/result/CPU2006.010.log

runspec finished at Fri Aug 30 16:17:52 2019; 393 total seconds elapsed
```

C. KERNBENCH

a. Description

C'est un benchmark qui permet de mesurer le débit d'exécution d'un processeur. Il est conçu pour comparer plusieurs kernels sur la même machine ou pour étudier le matériel (processeur). KERNBENCH

effectue une compilation du noyau (Linux à l'occurrence) sous différentes configurations:

- La moitié du nombre de CPUs
- 4 fois le nombre de CPUs (configuration par défaut)
- Le nombre maximal de jobs (configuration optimale)

A la fin le bench affiche des statistiques utiles pour la moyenne de chaque exécution.

b. Exécution

i. Prérequis

- Code source de linux (créer un dossier KERNBENCK et s'y déplcer) :

  wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.2.11.tar.xz

  tar -xJf linux-5.2.11.tar.xz

- Les outils *time, gawk :* ***sudo apt install time gawk***
- Le code de kernbench, qui est un script à télécharger à l'adresse : ***https://github.com/linux-test-project/ltp/blob/master/utils/benchmark/kernbench-0.42/kernbench.*** Dans le répertoire KERNBENCH créer un fichier *kernbench.sh* et y copier le code précédent.

ii. Exécution

Pour lancer l'exécution il faut se déplacer dans le dossier linux et lancer le script kernbench.sh :

```
user@user-Latitude-5280:~/Documents/tp/KERNBENCH$ cd linux-5.2.11/
user@user-Latitude-5280:~/Documents/tp/KERNBENCH/linux-5.2.11$ ../kernbench.sh
4 cpus found
Cleaning source tree...
Caching kernel source in ram...
No old config found, using allnoconfig
Making mrproper
Making allnoconfig...
Kernel 4.15.0-58-generic
Performing 5 runs of
make -j 2
make -j 4
make -j
```

```
All data logged to kernbench.log
Warmup run...
Half load -j 2 run number 1...
Half load -j 2 run number 2...
Half load -j 2 run number 3...
Half load -j 2 run number 4...
Half load -j 2 run number 5...
Average Half load -j 2 Run (std deviation):
Elapsed Time 46.502 (0.346944)
User Time 83.61 (0.537587)
System Time 8.352 (0.123774)
Percent CPU 197 (0)
Context Switches 4734 (632.998)
Sleeps 17805.2 (49.691)

Optimal load -j 4 run number 1...
Optimal load -j 4 run number 2...
Optimal load -j 4 run number 3...
Optimal load -j 4 run number 4...
Optimal load -j 4 run number 5...
Average Optimal load -j 4 Run (std deviation):
Elapsed Time 37.202 (0.800918)
User Time 104.662 (22.2012)
System Time 10.01 (1.75545)
Percent CPU 282.9 (90.65)
Context Switches 21995.4 (19915.7)
Sleeps 18156.1 (394.308)

Maximal load -j run number 1...
Maximal load -j run number 2...
Maximal load -j run number 3...
Maximal load -j run number 4...
Maximal load -j run number 5...
Average Maximal load -j Run (std deviation):
Elapsed Time 37.286 (0.51549)
User Time 112.72 (21.3595)
System Time 9.92133 (1.41669)
Percent CPU 312.4 (84.5744)
Context Switches 32980.4 (23741.6)
Sleeps 17970.9 (416.501)
```

4. Procédé expérimental et Interprétation des résultats

a. Procédé expérimental

Pour chacun de ces benchs, nous allons les exécuter dans un environnement natif puis dans un environnement virtualisé, et nous

allons réalisé une suite de 5 exécutions pour chaque bench.

Il faut créer un dossier distinct pour chaque environnement et pour chaque bench, pour ne pas s'emmeler les pinceaux au moment de l'analyse, et dans un fichier excel, on notera à la fin de chaque exécution la métrique prise en compte.

b. Interprétation des résultats

Pour interpréter les résultats, il faut répondre à ces questions :

- ➢ Quel graphe est il pertinent pour le résultat à interpréter? (Histogramme, Courbe, etc?).
- ➢ Comment compiler tous autres résultats de vos camarades? Comment les présenter pour pouvoir les comparer?

  Les exécutions étant faites sous les mêmes conditions et dans les mêmes environnements (mêmes machines, CPU, même taille de RAM, etc.), vous devriez normalement tous avoir les mêmes résultats (ce qui ne sera pas forcément le cas!!). Alors qu'est ce qui pourrait expliquer les écarts possibles entre vos résultats?