1. TRAVAIL A FAIRE

Ce projet consiste à analyser et à optimiser les performances d'un code de multiplication d'une matrice « creuse » par un vecteur. L'analyse et l'optimisation seront faites en utilisant plusieurs compilateurs et différentes options de compilation.

1.1. Stabilisation du système

Afin de garantir une mesure des performances stable et valide, il vous faudra stabiliser votre système en suivant les recommandations suivantes :

1. Si vous effectuez vos mesures sur un laptop, veillez à ce qu'il soit branché au secteur. Les batteries ne fournissent pas assez de puissance (Watts) pour que le CPU puisse atteindre sa fréquence maximale

2. La fréquence du CPU doit être fixe et configurée à sa valeur maximale

3. Aucun autre programme (navigateur, …) ne doit partager le système lors des mesures. Si possible, désactivez le réseau ainsi que l'interface graphique

4. Les threads du programme doivent être fixés sur des coeurs de calculs afin d'éviter le bruit causé par leur migration.

5. Effectuer la mesure des performances sur un système Linux natif. Toute mesure sur un système virtuel est considérée comme invalide car la virtualisation est de par sa nature considérée comme du bruit (point **4**).

Comment fixer la fréquence du CPU sous Linux?

> Avant de fixer la fréquence du CPU, il vous faudra vérifier quels sont les *frequency governors* présents sur votre système Linux en utilisant la commande suivante :

```
$ sudo cpupower -c all frequency-info

    analyzing CPU 0:
      driver: intel_pstate
      CPUs which run at the same hardware frequency: 0
      CPUs which need to have their frequency coordinated by software: 0
      maximum transition latency:  Cannot determine or is not supported.
      hardware limits: 800 MHz - 3.80 GHz
      available cpufreq governors: performance powersave
      current policy: frequency should be within 800 MHz and 3.80 GHz.
                      The governor "powersave" may decide which speed to use
                      within this range.
      current CPU frequency: Unable to call hardware
      current CPU frequency: 1.20 GHz (asserted by call to kernel)
      boost state support:
        Supported: yes
        Active: yes
    analyzing CPU 1:
      driver: intel_pstate
      CPUs which run at the same hardware frequency: 1
      CPUs which need to have their frequency coordinated by software: 1
      maximum transition latency:  Cannot determine or is not supported.
      hardware limits: 800 MHz - 3.80 GHz
      available cpufreq governors: performance powersave
      current policy: frequency should be within 800 MHz and 3.80 GHz.
                      The governor "powersave" may decide which speed to use
                      within this range.
      current CPU frequency: Unable to call hardware
      current CPU frequency: 1.25 GHz (asserted by call to kernel)
      boost state support:
        Supported: yes
        Active: yes
    analyzing CPU 2:
      driver: intel_pstate
      CPUs which run at the same hardware frequency: 2
      CPUs which need to have their frequency coordinated by software: 2
      maximum transition latency:  Cannot determine or is not supported.
      hardware limits: 800 MHz - 3.80 GHz
      available cpufreq governors: performance powersave
      current policy: frequency should be within 800 MHz and 3.80 GHz.
                      The governor "powersave" may decide which speed to use
                      within this range.
      current CPU frequency: Unable to call hardware
      current CPU frequency: 1.20 GHz (asserted by call to kernel)
      boost state support:
        Supported: yes
        Active: yes
    analyzing CPU 3:
      driver: intel_pstate
      CPUs which run at the same hardware frequency: 3
      CPUs which need to have their frequency coordinated by software: 3
      maximum transition latency:  Cannot determine or is not supported.
      hardware limits: 800 MHz - 3.80 GHz
      available cpufreq governors: performance powersave
      current policy: frequency should be within 800 MHz and 3.80 GHz.
                      The governor "powersave" may decide which speed to use
                      within this range.
      current CPU frequency: Unable to call hardware
      current CPU frequency: 1.20 GHz (asserted by call to kernel)
      boost state support:
        Supported: yes
        Active: yes
```

Comme vous pouvez le constater sur la sortie de la commande, deux *governors* sont disponibles : ***performance*** et ***powersave***. Nous pouvons aussi observer que tous les cœurs du CPU sont configurés en mode ***powersave***. En fonction du driver de votre CPU (**`intel_pstate`**, **`acpi_cpufreq`**, …), il est possible que votre système présente plusieurs autres *governors* (***schedutil***, ***userspace***, …).

Pour configurer les cœurs en mode ***performance***, vous devez utiliser la commande suivante :

```
$ sudo cpupower -c all frequency-set -g performance
```

REMARQUE : Lorsque l'on configure les cœurs en mode "performance", les modifications apportées au mode des cœurs ne persistent pas après un redémarrage. Il faut exécuter la

fonction permettant de passer tous les cœurs en mode performance à chaque démarrage de l'ordinateur portable.

Vous pouvez, bien sûr, cibler un cœur en particulier en spécifiant son identifiant à la place de ***all***. Par exemple, la commande suivante permet de modifier le ***governor*** du cœur **n °2** indépendamment des autres cœurs :

```
$ sudo cpupower -c 2 frequency-set -g performance
```

Comment allouer un processus sur un coeur du CPU?

Afin d'éviter de bruiter les mesures par la migration de processus implémentée par l'OS pour améliorer la gestion des ressources CPU, il faut signaler à l'OS que les threads OpenMP du programme toto doit s'exécuter sur des coeurs désignés. Cette technique est généralement nommée : thread/process core pinning. Pour effectuer ce pinning vous pouvez utiliser les variables d'environnement `OMP_PLACES` et `OMP_PROC_BIND`. On utilise généralement (`OMP_PLACES=cores` et `OMP_PROC_BIND=close`) ou (`OMP_PLACES=threads` et `OMP_PROC_BIND=spread`)."

```
$ OMP_PLACES=cores   OMP_PROC_BIND=close  ./toto
$ OMP_PLACES=threads OMP_PROC_BIND=spread ./toto
```

1.2. Analyse des performances

De manière standard, la mesure des performances utilise 2 types de métriques:

- *`Time`*: le temps d'exécution du noyau en **secondes**
- *`GFLOP/s`*: le nombre d'opérations d'arithmétique flottante effectuées par seconde (appelée aussi **intensité arithmétique**)

Ces métriques nous permettent d'évaluer le temps d'exécution ainsi que la quantité de *"travail"* effectuée durant une unité de temps (**1 seconde**). Afin de pouvoir évaluer la performance avec un haut degré de précision, le noyau doit être exécuté plusieurs fois et sa performance est mesurée à chaque exécution.

Pour évaluer et garantir la qualité des mesures effectuées, vous utiliserez le mode Stability de MAQAO One View.

1.3. Compilation

Afin d'explorer et maximiser les possibilités en termes de performance, il vous faudra compiler le code avec **GCC** et **OneAPI (ICPX)**, et avec 2 niveaux d'optimisation (**O3 et Ofast/Fast**) pour chacun des compilateurs. Ensuite il faudra d'abord comparer les performances des deux options (niveaux d'optimisation) pour un même compilateur puis comparer les meilleures options des 2 compilateurs. Le but est d'analyser l'impact des différents compilateurs et des différentes optionsur les performances.

GCC et **OneAPI (ICPX)** s'installent facilement sur tous les systèmes Linux et sont compatibles au niveau de leurs interfaces. Cependant, il est toutefois recommandé de se référer à la documentation (**man gcc** et **man ICPX**). Le diable est dans les détails!!

Dans le makefile, il est utile de rajouter l'option « -fno-omit-frame-pointer » et peut être l'option -lstdc

NOTE : dans OneAPI, ICX est le compilateur C et ICPX le compilateur C++.

1.4. Profileurs

Il faudra utiliser le profiler **MAQAO** (www.maqao.org). Il faudra fournir le répertoire contenant les résultats d'analyse MAQAO. En complément, vous pourrez utiliser d'autres profilers tels que : **Intel VTune**, **AMD µProf**, **Linux perf** ou **MAP (Linaro)**.

Il faudra fournir les données brutes (répertoire des résultats MAQAO au format html) ainsi que des explications relatives aux métriques retournées par ses outils et à leur stabilité.

Vous serez amené à utiliser 4 modes de MAQAO :

- Mode « Normal/Standard » : le paramètre "-create-report=one" permet de lancer un test normal sur MAQAO
- Mode « Stability »
- Mode « Scalability ». Le paramètre "—with-scalability=strong" permet de lancer un test scalabilité sur MAQAO
- Mode « Compare » : la comparaison des options de compilation et des compilateurs devra utiliser le mode Compare de MAQAO.

NE PAS OUBLIER DE CONSULTER LA DOC EN LIGNE : maqao oneview --help

1.5. Optimisation

Les optimisations seront appliquées via compilateur.

GCC et **OneAPI** proposent plusieurs options pour piloter les optimisations et générer du code machine adapté à l'architecture cible. Les niveaux d'optimisation que vous devrez tester sont les suivants: **-O3** et **-Fast/Ofast**. Plus d'informations sont disponibles ici: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. Chaque niveau d'optimisation implémente un ensemble de transformations destinées à produire du code machine performant.

Cependant, le flag d'optimisation **-Fast/Ofast** peut sacrifier la stabilité numérique des calculs au profit d'une meilleure performance en appliquant des transformations sur l'ordre des opérations d'arithmétique flottante. Il faudrait donc effectuer une analyse de la stabilité numérique. Ce point n'est pas à traiter dans le projet.

Afin d'assurer que les optimisations des compilateurs sont adaptées à l'architecture cible, vous pouvez utiliser le flag **-march=** ***TARGET_ARCH*** en spécifiant l'architecture cible. Vous trouverez les identifiants des architectures et plus d'informations

ici: https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html. Il est aussi possible de faire détecter l'architecture par le compilateur en utilisant le flag: **-march=native**.

Pour chaque version et chaque compilateur, il faudra fournir une brève analyse du code assembleur. Cette analyse devra évaluer la qualité du code généré par rapport à la vectorisation, le déroulage, … **MAQAO** fournit des analyses détaillées des codes assembleurs pour une architecture cible donnée.

1.6. Code Source et données

Le code source à tester est disponible à https://academic.liparad.uvsq.fr/iatic4/aob/spmxv/ et à l'adresse miroir `https://academic.exascale-computing.eu/iatic4/aob/`.

Le fichier epi-spmxv-main.tar.gz (28 MB) contient le code et les données à utiliser. Le fichier additionel_matrix.tar.gz (2 GB) contient d'autres matrices mais ne sera pas utilisé dans un premier temps.

Il n'y a pas de readme mais le code est très simple et les seules modifications à effectuer porteront sur le Makefile qu'il faudra changer pour insérer les noms des compilateurs et les options.

La commande à utiliser sera est la suivante. Attention les valeurs des paramètres seront à utiliser impérativement.

./spmxv.exe -f input-matrix/mat_dim_493039.txt -t <nb threads> -r 20000

-f : filepath de la matrice à utiliser comme donnée d'entrée

-t : nombre de threads à utiliser (valeur identique à OMP_NUM_THREADS)

-r : nombre de répétitions. La valeur de 5000 a été spécifiée pour obtenir des temps d'exécution supérieurs à 30 Secondes.

2. TRAVAIL A EFFECTUER

Il faudra utiliser le parallélisme OpenMP présent via directives dans le code.

Vous aurez à effectuer 3 types de mesures :

- Mesure de stabilité sur le code s'exécutant sur le nombre maximum de cœurs physiques disponibles sur la machine. Ces runs doivent être faits en utilisant le mode « stability » de MAQAO.
- Mesure standard sur le code s'exécutant sur le nombre maximum de cœurs physiques disponibles sur la machine. Ces runs doivent être faits en utilisant le mode standard de MAQAO.
- Mesures d'extensibilité (aka scalabilité) du code s'exécutant sur un nombre variable de cœurs depuis un jusqu'au nombre maximum de cœurs physique. Ces runs doivent être faits en utilisant le mode « scalability » de MAQAO. Voir ci-dessous la table de test a effectuer suivant le nombre maximum de cœurs physiques.

Nombre Max de cœurs physique	Nombre de coeurs à tester
2	1,2
4	1,2,3,4
6	1,2,3,4, 6
8	1,2,4,8
16	1,2,4,8 ,16

3. RAPPORT

Il vous faudra fournir :

- le fichier de votre rapport au format pdf envoyé par mail : ce rapport doit être nommé **`NOM_PRENOM_iatic5_rapport_AP`**
- une archive nommée comme suit **`NOM_PRENOM_iatic5_proj.tar.gz`**.

Avant transmission, l'archive et le rapport devront avoir été testés par un antivirus. La présence d'un virus dans l'archive ou dans le rapport entrainera automatiquement la note de 1/20.

Les détails pour ces 2 documents sont indiqués ci-dessous

3.1 DETAILS SUR LE RAPPORT :

Un rapport **EN ANGLAIS** au format **PDF** dont le texte (hors figures et screenshots) doit contenir au plus 20 000 caractères en incluant les espaces. Le nombre total de caractères utilisées devra être indiqué. Ce rapport doit être structuré comme suit :

- **Section 1** : **Environment description**
  - **Hardware description**
  - **Software description :** compilers, OS
- **Section 2** : **GCC Compiler. Max number of available physical cores have to be used**
  - **Option -O3**: results + comments. Screenshot stability + Screenshot Global Metrics (Maqao Standard).
  - **Option -Fast**: results + comments. Screenshot stability + Screenshot Global Metrics (Maqao Standard).
  - **Comparing options.** Screenshot compare mode Global Metrics
- **Section 3** : **OneAPI ICPX Compiler. Max number of available physical cores have to be used**
  - **Option -O3**: results + comments. Screenshot stability + Screenshot Global Metrics (Maqao Standard).
  - **Option -Fast**: results + comments. Screenshot stability + Screenshot Global Metrics (Maqao Standard).
  - **Comparing options.** Screenshot compare mode Global Metrics
- **Section 4 : comparing compilers tested in Sections 2 and 3.**
  - **Compiler Comparison :** results + comments. Screenshot compare mode Global Metrics
- **Section 5 : Scalability**

    - **GCC Compiler Option -Fast :** Scalability runs. Results + comments. Screenshot scalability
    - **ICPX Compiler Option -Fast :** Scalability runs. Results + Comments. Screenshot scalability
    -
- **Section 6 : Conclusion**

3.2 DETAILS SUR LE CONTENU DE L'ARCHIVE :

- les informations (un fichier **cpu.info**) sur l'architecture sur laquelle vous avez effectué vos expériences. Vous pouvez obtenir les informations sur votre système en utilisant les commandes suivantes :

```
$ cat /proc/cpuinfo > cpu.info

ou

$ lscpu > cpu.info
```

- les informations sur la hiérarchie des caches de données de votre système (fichiers: **L?.info**):

```
# Cache L1
$ cat /sys/devices/system/cpu/cpu0/cache/index0/* > L1.info

# Cache L2
$ cat /sys/devices/system/cpu/cpu0/cache/index2/* > L2.info

# Cache L3
$ cat /sys/devices/system/cpu/cpu0/cache/index3/* > L3.info
```

- Les données brutes des mesures de performance i.e. résultats MAQAO format html

4. CONSEIL

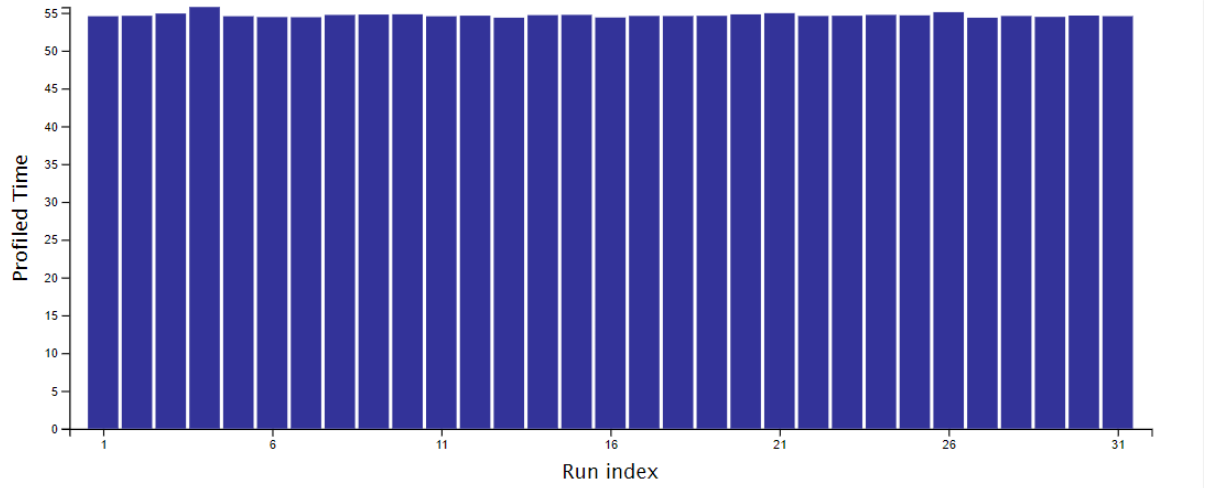
In doubt, RTFM :]

5. REFERENCES

- https://godbolt.org/
- https://www.agner.org/optimize/optimizing_cpp.pdf
- https://maqao.org/

EXAMPLES DE SCREEN SHOT STABILITY

Profiled Time



min	med	avg	max
54.36	54.61	54.67	55.77

EXAMPLE DE SCREENSHOT ONE VIEW GLOBAL METRICS (attention c'est un run parallèle)

▸ **Filter Information**

Global Metrics

Total Time (s)	37.00
Profiled Time (s)	36.66
Time in analyzed loops (%)	85.0
Time in analyzed innermost loops (%)	81.3
Time in user code (%)	85.0
Compilation Options Score (%)	75.0
Array Access Efficiency (%)	65.2

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP + MPI + Pthread		1.17
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.34
No Scalar Integer	Potential Speedup	1.02
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	2.14
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	3.92
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.02
	Nb Loops to get 80%	1