Laboratoire 6: Mesure de consommation énergétique

Département: **TIC**

Unité d'enseignement: **HPC**

Auteur(s):

- **CECCHET Costantino**

Professeur:

- **DASSATTI Alberto**

Assistant:

- **DA ROCHA CARVALHO Bruno**

Date:

- **22/05/2024**

[Page de mise en page, laissée vide par intention]

Introduction

Dans ce laboratiore nous allons mesurer la consommation énergétique d'un programme en utilisant différents outils de mesures.

Programme de test

Le programme de test est un programme C qui remplit un tableau de taille definie avec des 1, puis effectue une somme de tous les éléments du tableau. Le programme est le suivant:

```c
int main() {
    float *array = (float *)malloc(SIZE * sizeof(float));

    for (size_t i = 0; i < SIZE; i++) {
        array[i] = 1.0f;
    }

    float sum = sum_non_vectorized(array, SIZE);
    printf("Somme non vectorisée : %f\n", sum);

    free(array);
    return 0;
}
```

Deux versions de la fonction de somme sont disponibles, une version non vectorisée et une version vectorisée, elles sont dans les fichiers 'lab06.c' et 'lab06_vec.c' respectivement.

Mesure de consommation énergétique

likwid-powermeter

La première méthode de mesure de consommation énergétique est l'utilisation de l'outil `likwid-powermeter`. Cet outil permet de mesurer la consommation énergétique d'un programme en utilisant les compteurs de performance du processeur.

voici la commande pour mesurer la consommation énergétique des programmes non vectorisés et vectorisés :

```
$ likwid-power.sh
```

Et voici les résultats obtenus :

Mesures	Non-Vectoriser	Vectoriser
Runtime (s)	0.0208941	0.00642727
PKG (Joules)	0.0491333	0.0325317
PKG (Watt)	2.35154	5.06152
PP0 (Joules)	0.0143433	0.0186768
PP0 (Watt)	0.686475	2.90586
PP1 (Joules)	0.173218	0.105957
PP1 (Watt)	8.29028	16.4856
DRAM (Joules)	0.00683594	0.0055542

Mesures	Non-Vectoriser	Vectoriser
DRAM (Watt)	0.327171	0.864162
PLATFORM (Joules)	0.173218	0.105957
PLATFORM (Watt)	8.29028	16.4856

perf

La deuxième méthode de mesure de consommation énergétique est l'utilisation de l'outil `perf`. Cet outil permet de mesurer la consommation énergétique d'un programme en utilisant les compteurs de performance du processeur.

voici la commande pour mesurer la consommation énergétique des programmes non vectorisés et vectorisés :

```
$ perf stat -e power/energy-pkg/ -e power/energy-pp0/ -e power/energy-pp1/ -e power/energy-r
```

Et voici les résultats obtenus :

Mesures	Non-Vectoriser	Vectoriser
Cores (Joules)	0.01	0.07
GPU (Joules)	0.00	0.00
PKG (Joules)	0.01	0.17
PSYS (Joules)	0.07	0.88
RAM (Joules)	0.08	0.03

likwid-perfctr

La troisième méthode de mesure de consommation énergétique est l'utilisation de l'outil `likwid-perfctr`. Cet outil permet de mesurer la consommation énergétique d'un programme en utilisant les compteurs de performance du processeur.

voici la commande pour mesurer la consommation énergétique des programmes non vectorisés et vectorisés :

```
sudo likwid-perfctr -C S0:0 -g ENERGY ./lab06 && sudo likwid-perfctr -C S0:0 -g ENERGY ./lab
```

Et voici les résultats obtenus :

Metric	HWThread 0 (non-vectorized)	HWThread 0 (vectorized)
Runtime (RDTSC) [s]	0.0066	0.0041
Runtime unhalted [s]	0.0041	0.0014
Clock [MHz]	3062.5857	2694.8486
CPI	0.9927	0.5848
Temperature [C]	64	54
Energy [J]	0.0930	0.0372
Power [W]	14.0752	9.0053
Energy PP0 [J]	0.0745	0.0301
Power PP0 [W]	11.2842	6.6105
Energy PP1 [J]	0.0032	0
Power PP1 [W]	0.4898	0
Energy DRAM [J]	0.0143	0.0014

Metric	HWThread 0 (non-vectorized)	HWThread 0 (vectorized)
Power DRAM [W]	2.1718	0.3395

Résultats

Les résultat peuvent varier d'une exécution à l'autre, mais les tendances restent les mêmes.

Analyse des résultats

Les résultats obtenus avec les trois méthodes de mesure de consommation énergétique sont assez cohérents. Les résultats obtenus avec `likwid-powermeter` et `perf` sont assez proches, tandis que les résultats obtenus avec `likwid-perfctr` sont un peu plus élevés. Cela peut être dû à la précision des compteurs de performance utilisés par les différents outils.

Les résultats obtenus avec les programmes non vectorisés sont plus élevés que ceux obtenus avec les programmes vectorisés. Cela est dû au fait que les programmes vectorisés sont plus efficaces et nécessitent moins de cycles d'horloge pour effectuer les mêmes opérations.

Conclusion

Dans ce laboratoire, nous avons mesuré la consommation énergétique de programmes non vectorisés et vectorisés en utilisant trois méthodes différentes. Les résultats obtenus avec les trois méthodes sont cohérents et montrent que les programmes vectorisés consomment moins d'énergie que les programmes non vectorisés.

Environnement d'execution

Le système décrit dispose d'un processeur Intel Core i7-8550U avec 8 threads, répartis sur 4 cœurs physiques. La fréquence du processeur est de 1,80 GHz avec une fréquence mesurée de 1432,548 MHz.

Voici plus ample information sur le processeur:

```
$ cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 142
model name  : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
stepping    : 10
microcode   : 0xf4
cpu MHz     : 1432.548
cache size  : 8192 KB
physical id : 0
siblings    : 8
core id     : 0
cpu cores   : 4
apicid      : 0
initial apicid  : 0
```

```
fpu     : yes
fpu_exception   : yes
cpuid level : 22
wp      : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflus
vmx flags   : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offse
bugs        : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_mult
bogomips    : 3999.93
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual

$ likwid-topology
--------------------------------------------------------------------------------
CPU name:   Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
CPU type:   Intel Kabylake processor
CPU stepping:   10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:        1
Cores per socket:   4
Threads per core:   2
--------------------------------------------------------------------------------
HWThread    Thread      Core        Socket      Available
0       0       0       0       *
1       0       1       0       *
2       0       2       0       *
3       0       3       0       *
4       1       0       0       *
5       1       1       0       *
6       1       2       0       *
7       1       3       0       *
--------------------------------------------------------------------------------
Socket 0:       ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:          1
Size:           32 kB
Cache groups:       ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:          2
Size:           256 kB
Cache groups:       ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:          3
Size:           8 MB
Cache groups:       ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
```

```
NUMA Topology
******************************************************************************
NUMA domains:       1
------------------------------------------------------------------------------
Domain:         0
Processors:     ( 0 4 1 5 2 6 3 7 )
Distances:      10
Free memory:        542.145 MB
Total memory:       7695.82 MB
------------------------------------------------------------------------------
```