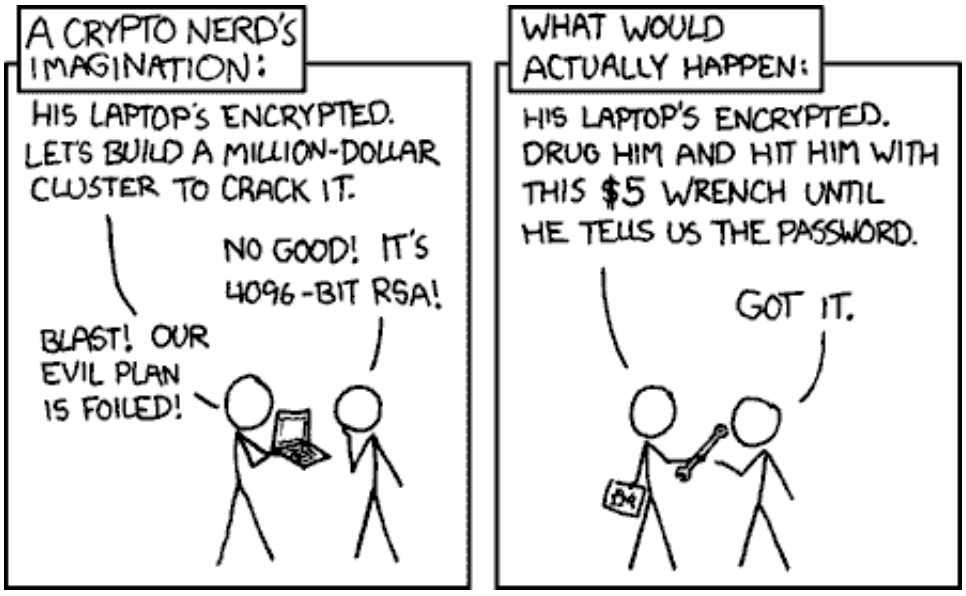
Programmation concurrente
Password Cracker

Claudio Sousa - David Gonzalez

1 novembre 2016



1 Introduction

1.1 Division des tâches

Chaque membre du group a fait seul le programme dans son entièreté. Le travail rendu est la mise en commun du code des differents programmes.

1.2 Machine cible

lscpu:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 60
Stepping:              3
CPU MHz:               800.000
BogoMIPS:              7183.36
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              8192K
NUMA node0 CPU(s):     0-7
```

2 Thread performance comparison

2.1 Méthodologie

Afin de produire les données statistiques de performance, le cracker a été modifié afin de s'éxécuter pendant 10s et imprimer le nombre de mot de passes encryptées par seconde. 20 testes ont été fais pour chaque nombre de threads utilisé ($2^{\{0..8\}}$)

2.2 Données

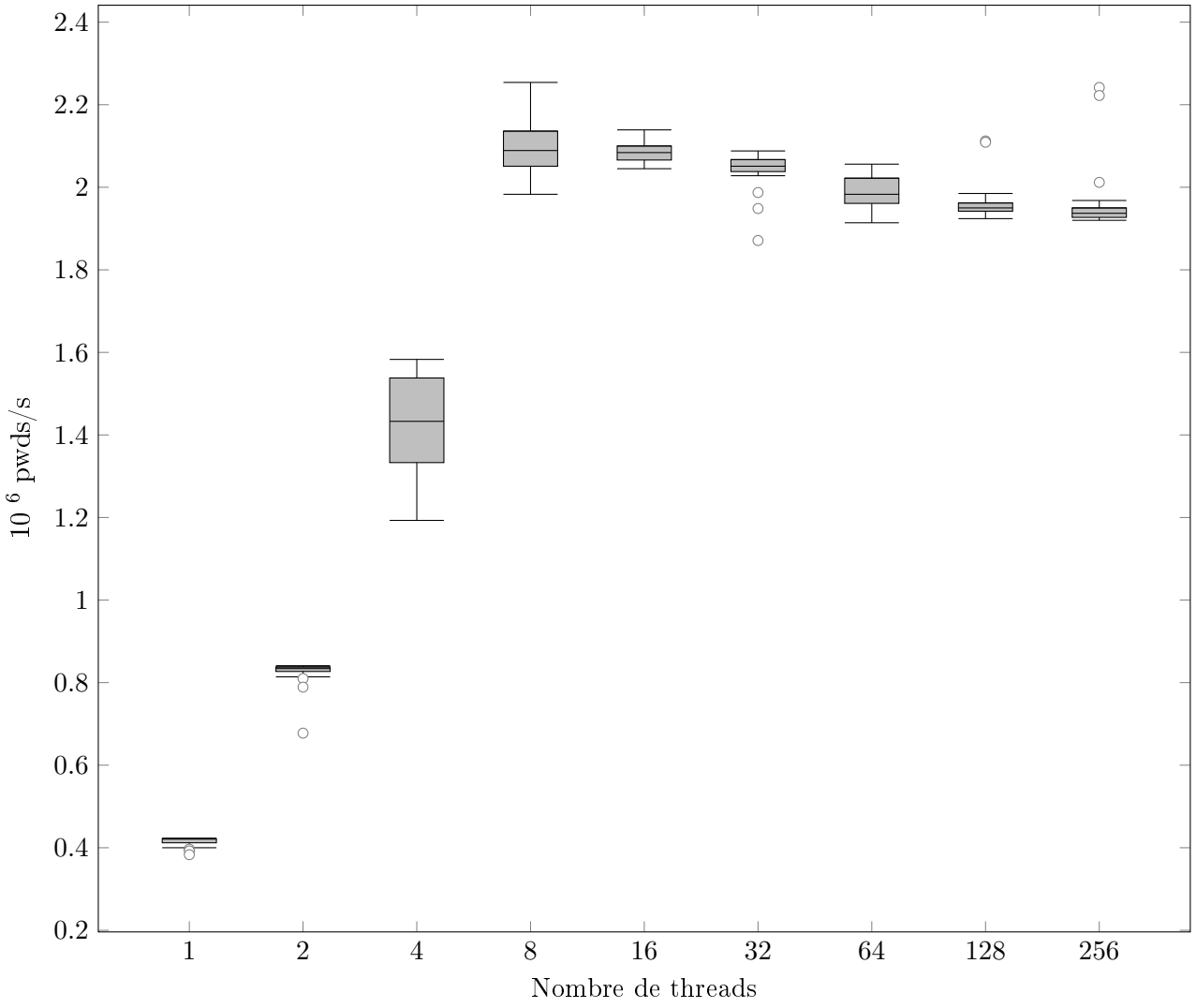


Figure 1: Comparaison des performances avec un nombre différent de threads

Nombre de threads	Median	Q3	Q1	Q4	Q0
1	0.421	0.422	0.412	0.423	0.4
2	0.834	0.838	0.827	0.841	0.814
4	1.433	1.538	1.333	1.583	1.193
8	2.089	2.136	2.051	2.254	1.983
16	2.084	2.1	2.066	2.139	2.045
32	2.051	2.067	2.038	2.088	2.028
64	1.983	2.022	1.961	2.056	1.914
128	1.95	1.962	1.942	1.985	1.924
256	1.937	1.95	1.927	1.968	1.92

Table 1: Comparaison des performances avec un nombre différent de threads

2.3 Questions & réponses

Le gain de vitesse est linéaire avec le nombre de threads?
Le gain de performance (mots de pass crackés par second) avec le nombre de threads est essentiellement linéaire (on ne double pas, mais presque) jusqu'au nombre de CPUs logiques sur la machine hôte.

L'utilisation de threads supplémentaires n'augmente pas la performance. Au contraire, la performance diminue. En effet, si nous avons déjà un thread par CPU logique, nous avons déjà un taux d'utilisation de 100% pour chaque CPUs. Augmenter alors le nombre de threads utilisé ne permettra pas d'augmenter le taux d'utilisation des CPUs, et créera inévitablement une baisse de performance due au temps passé à changer de contexte.

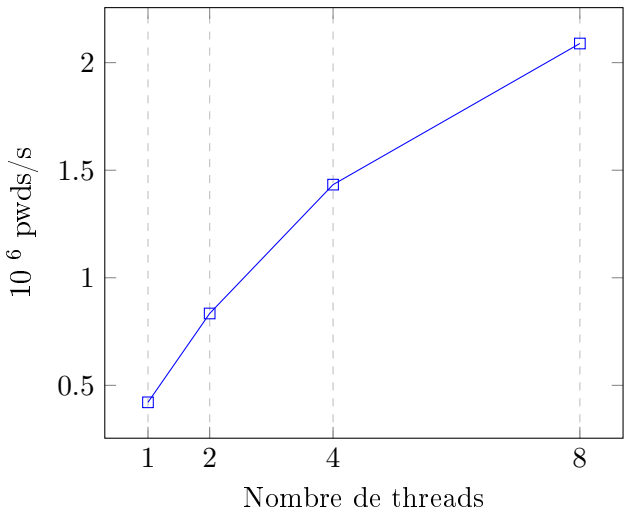


Figure 2: Comparison de la performance par thread avec axe des abscisses avec échelle linéaire

Quel est l'impact de l'hyper-threading sur les performances?
Le hyper-threading permet de doubler le nombre de CPUs. Dans notre cas, malgré que la machine n'ait que 4 CPUs physiques, l'hyper-threading permet d'avoir 8 CPUs logiques à disposition.

Chaque CPU logique possède son propre jeu de registres, mais la paire partage la même unité de calcul. Nous avons pu constater empiriquement que la ressource partagée entre chaque paire de CPUs logiques (l'ALU) ne semble pas être utilisée dans sa totalité par un seul CPU logique. En effet, l'utilisation du $2^{\text{ème}}$ CPU logique (sur le même CPU physique) permet, d'après nos résultats, de quasiment doubler les performances.

Mot de passe trouvé
groumf (après 2h20m53s)