

# TP Evaluation de performances

M1 INFO CSE-PC

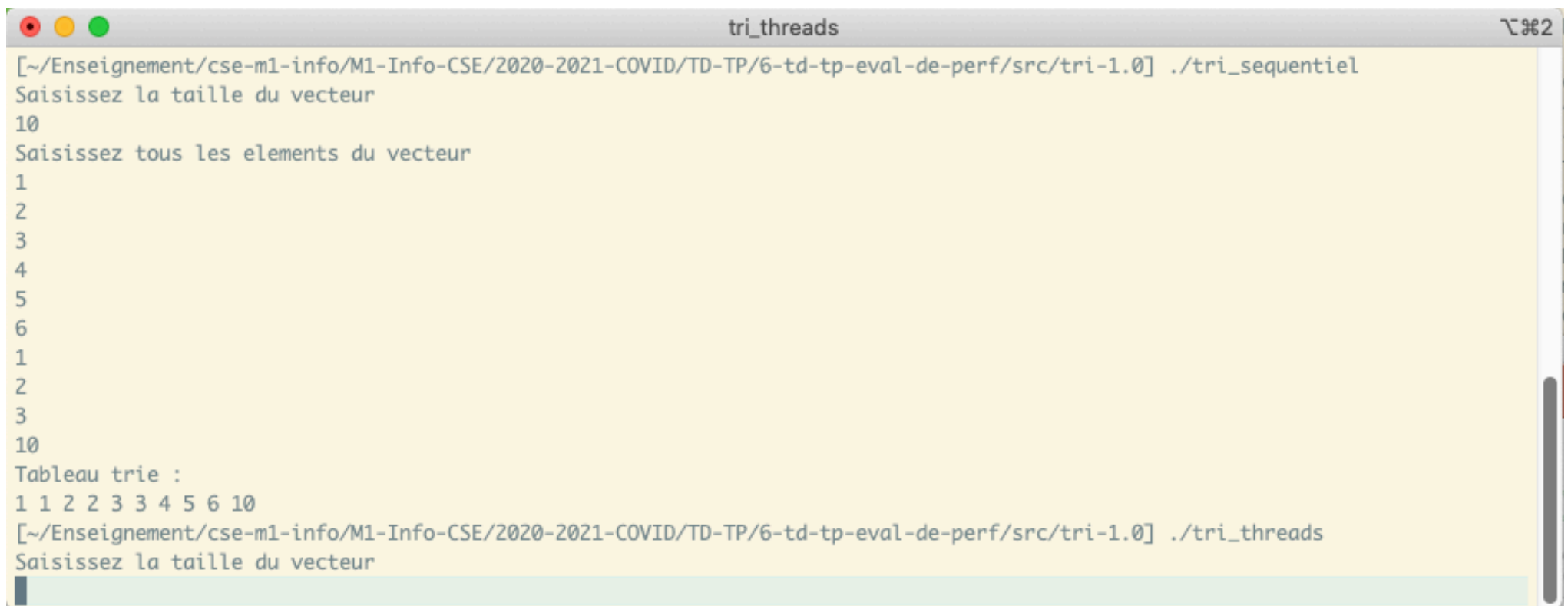
2020-2021

# Objectifs du TP

- ▶ **Continuer à jouer avec les threads**
- ▶ **Etudier l'accélération d'un programme**
  - ▶ Identifier les facteurs pertinents i.e. répondre à la question "Qu'est-ce qui influence l'exécution de mon programme?"
- ▶ **Se sensibiliser à la problématique de l'évaluation des performances**
  - ▶ Il n'est finalement pas si simple de dire si un programme marche "bien" ou pas
  - ▶ D'ailleurs, cela veut dire quoi : bien marcher?

# Programmes fournis (1)

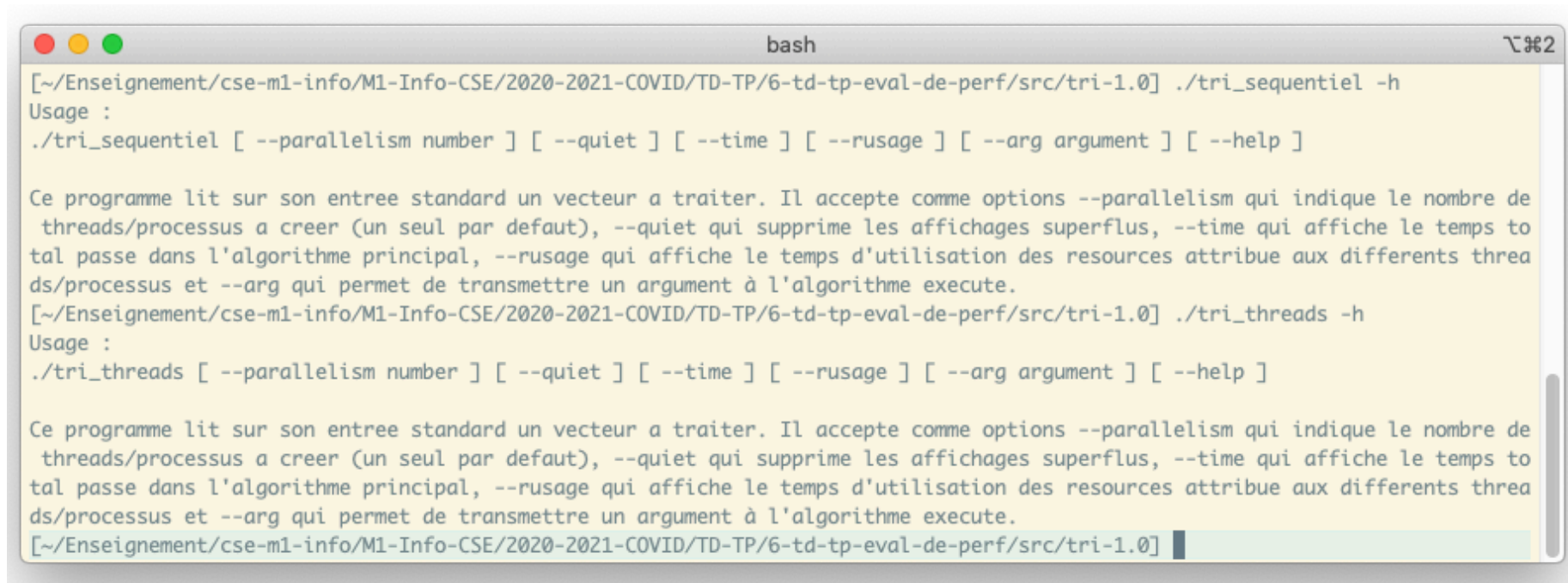
- ▶ **Nous vous fournissons un programme de tri**
  - ▶ Version séquentielle et parallèle



```
tri_threads    2
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0] ./tri_sequentiel
Saisissez la taille du vecteur
10
Saisissez tous les elements du vecteur
1
2
3
4
5
6
1
2
3
10
Tableau trie :
1 1 2 2 3 3 4 5 6 10
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0] ./tri_threads
Saisissez la taille du vecteur
```

## Programmes fournis (2)

- Vous pouvez voir les paramètres acceptés par les deux programmes



```
bash
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0] ./tri_sequentiel -h
Usage :
./tri_sequentiel [ --parallelism number ] [ --quiet ] [ --time ] [ --rusage ] [ --arg argument ] [ --help ]

Ce programme lit sur son entree standard un vecteur a traiter. Il accepte comme options --parallelism qui indique le nombre de
threads/processus a creer (un seul par default), --quiet qui supprime les affichages superflus, --time qui affiche le temps to
tal passe dans l'algorithme principal, --rusage qui affiche le temps d'utilisation des ressources attribue aux differents threa
ds/processus et --arg qui permet de transmettre un argument à l'algorithme execute.
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0] ./tri_threads -h
Usage :
./tri_threads [ --parallelism number ] [ --quiet ] [ --time ] [ --rusage ] [ --arg argument ] [ --help ]

Ce programme lit sur son entree standard un vecteur a traiter. Il accepte comme options --parallelism qui indique le nombre de
threads/processus a creer (un seul par default), --quiet qui supprime les affichages superflus, --time qui affiche le temps to
tal passe dans l'algorithme principal, --rusage qui affiche le temps d'utilisation des ressources attribue aux differents threa
ds/processus et --arg qui permet de transmettre un argument à l'algorithme execute.
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0]
```

## Programmes fournis (3)

### ► Nous vous fournissons également un générateur de vecteurs

- Pour vous éviter de taper à la main à chaque fois
- Surtout si vous voulez tester avec des vecteurs de plusieurs milliers/millions d'éléments 😊

```
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0] ./creer_vecteur -h
Usage :
./creer_vecteur [ --seed number ] [ --size number ] [ --min value ] [ --max value ] [ --help ]

Ce programme affiche sur sa sortie standard la taille donnée (default 1000) puis autant d'entiers que cette taille. Chacun de ces entiers est généré aléatoirement dans l'intervalle [min;max] (default [0;1000]). Le générateur aléatoire est initialisé avec la valeur donnée pour seed ou, à défaut, avec l'horloge système.
```

### ► Le générateur produit

- Un vecteur d'entiers
- D'une certaine longueur (par défaut 1000)
- Compris entre MIN et MAX (par défaut entre 0 et 1000)
- Générés aléatoirement

# Le vrai intérêt du générateur

- ▶ **Le temps de tri d'un vecteur dépend de**
  - ▶ L'algorithme bien sûr mais là il est imposé
  - ▶ La taille du vecteur
  - ▶ La distribution des valeurs dans le vecteur
- ▶ **Si on veut comparer le temps de tri entre un tri séquentiel, un tri parallèle avec 2 threads, un tri parallèle avec 4 threads , etc**
  - ▶ Vaut mieux utiliser le même vecteur!!!
  - ▶ Avec le même algorithme (évident!)
  - ▶ Avec la même taille de données (évident!)
  - ▶ Avec la même distribution – moins évident, on risque de ne pas y penser

## L'usage typique que vous ferez

1. Générer un vecteur d'une certaine taille et le sauvegarder dans un fichier

```
./creer_vecteur 1000000 > million.v1
```

2. Passer ce vecteur au tri séquentiel\*

```
./tri_sequentiel < million.v1
```

3. Passer le même vecteur au tri parallèle\*

```
./tri_threads -p 2 < million.v1 -t
```

4. Comparer les temps d'exécution\*

**\* ATTENTION : il y a quelques détails encore à traiter**

## Nous allons donc étudier

- ▶ Le temps que prend le tri
- ▶ Comparer le temps séquentiel au temps parallèle
- ▶ Raisonner sur l'accélération i.e.  
si et combien de fois le programme parallèle est plus rapide comparé au programme séquentiel
- ▶ Intuitivement, nous aimerions gagner en rajoutant des threads
  - ▶ *"Plus il y en aura, plus vite cela ira!"*
- ▶ **Mais...**  
**vous savez déjà que ce n'est pas vrai dans tous les cas**



## Les étapes de l'étude

- 1. Instrumenter les programmes =  
les modifier pour mesurer le temps de tri**
- 2. Mesurer le temps d'exécution  
pour différentes valeurs des paramètres pertinents**
- 3. Représenter via des graphiques**
- 4. Analyser**

## ETAPE 1 : Instrumenter les programmes

- ▶ Les programmes n'incluent pas les instructions qui permettent la mesure du temps
- ▶ **C'est à vous de les rajouter**
- ▶ En utilisant les fonctions `gettimeofday` et `getrusage`
- ▶ Il faut comprendre la différence entre les deux fonctions
- ▶ Il faut décider quelle partie du code mesurer

```
temps1 = mesure_temps...  
traitement...  
temps2 = mesure_temps...  
printf("Le temps de traitement est :...", temps2-temps1);
```

## ETAPE 2 : Mesurer le temps d'exécution pour différentes valeurs des paramètres pertinents

### ► Il faut mesurer le temps d'exécution plusieurs fois!

```
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < 10.v1 -t  
Saisissez la taille du vecteur  
Saisissez tous les elements du vecteur  
Tableau trie :  
85 295 336 337 404 428 645 756 807 914  
TEMPS : 3 micro secondes  
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < 10.v1 -t  
Saisissez la taille du vecteur  
Saisissez tous les elements du vecteur  
Tableau trie :  
85 295 336 337 404 428 645 756 807 914  
TEMPS : 4 micro secondes
```

10 éléments

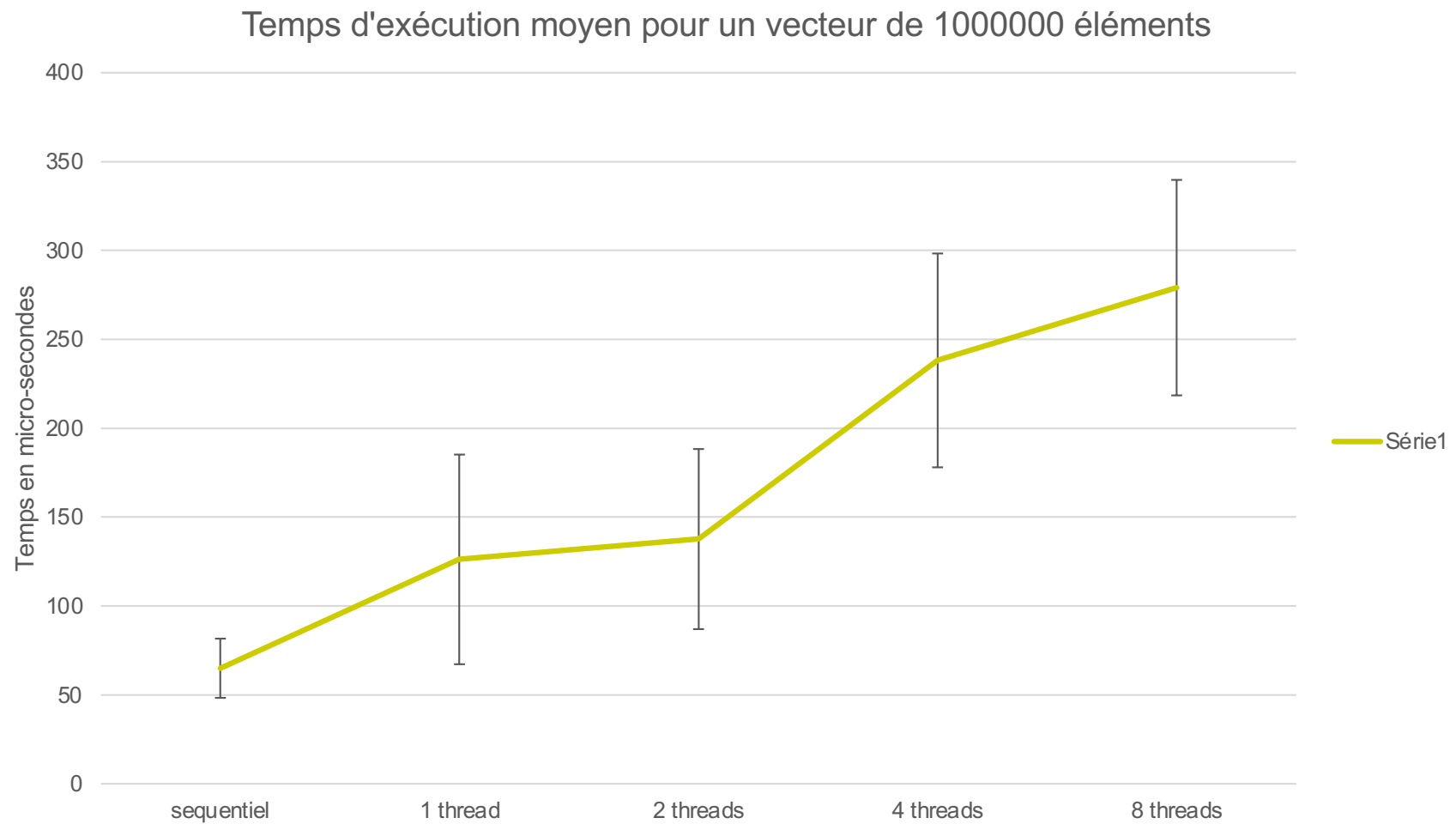
```
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < million.v1 -t -q  
TEMPS : 100 micro secondes  
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < million.v1 -t -q  
TEMPS : 65 micro secondes  
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < million.v1 -t -q  
TEMPS : 80 micro secondes  
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < million.v1 -t -q  
TEMPS : 99 micro secondes  
[~/Enseignement/cse-m1-info/M1-Info-CSE/2020-2021-COVID/TD-TP/6-td-tp-eval-de-perf/src/tri-1.0-vania] ./tri_sequentiel < million.v1 -t -q  
TEMPS : 104 micro secondes
```

1000000 éléments

## ETAPE 2 : Mesurer le temps d'exécution pour différentes valeurs des paramètres pertinents

- ▶ **Il faut mesurer le temps d'exécution plusieurs fois!**
  - ▶ Le temps d'exécution dépend de l'état actuel de la machine sur laquelle s'exécute le programme
    - Caractéristiques matérielles
    - Charge – quels programmes, quel ordonnancement
    - Environnement (température extérieure, refroidissement de la machine, ...)
  - ▶ Il faut effectuer un nombre suffisamment grand de mesures **(30)**
    - Calculer la **moyenne**
    - **Et l'écart-type**
      - pour caractériser la représentativité de la moyenne i.e. la variabilité des valeurs
      - Ainsi, 95% des valeurs sont contenues dans l'intervalle [moyenne – 1,96\*écart, moyenne+1,96\*écart]

## ETAPE 2 : Exemple



## Etape 2 : Automatiser les expérimentations

- ▶ **Itérations = nombre de fois où vous aller répéter la même expérience (mesure)**
- ▶ **Nous vous conseillons d'automatiser le lancement des tests et la récupération des résultats**
  - ▶ Vous n'allez pas attendre avec votre doigt levé devant l'écran 😊
- ▶ **Un exemple de script simple**

```
#!/bin/bash

ITERATIONS=30

for i in `seq 1 ${ITERATIONS}`;
do
    ./tri_sequentiel < 10million.v1 -t -q >> seq.10million.v1
done
```

## ETAPE 2 : Suite

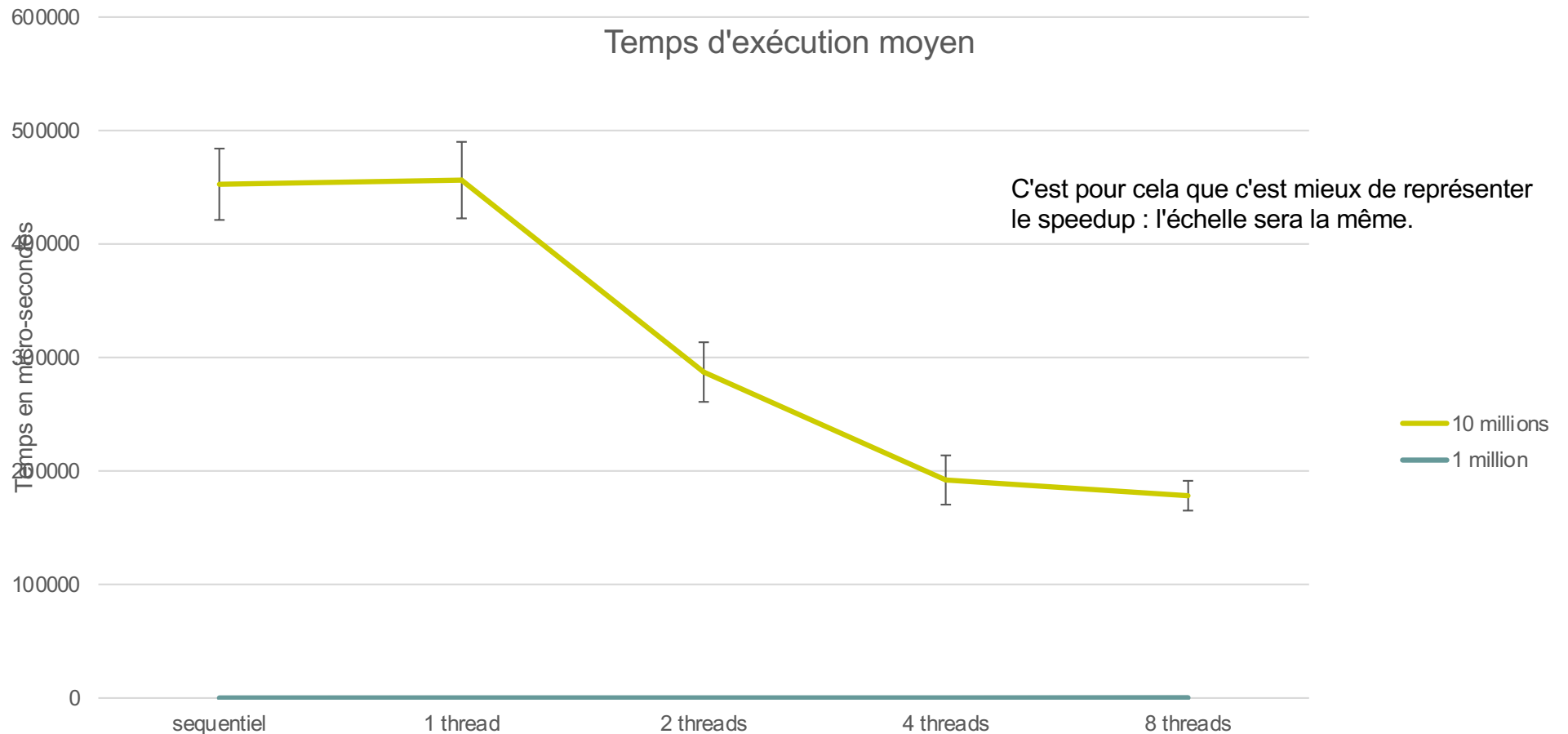
- ▶ **Votre objectif est de comparer des temps d'exécution**
- ▶ **Dans quels cas? Sous quelles conditions?**
- ▶ **Les résultats dépendent**
  - ▶ De la taille du vecteur => il faut faire varier la taille
    - Petit, moyen gros
    - Intuitivement, avec le petit, le parallèle serait plus long, et avec le grand, ce sera l'invers
    - Attention à ce que cela veut dire petit, moyen, gros – n'hésitez pas à mettre des tailles qui vous semblent vraiment très grandes ( $10^6$ ,  $10^9$ , ...)
  - ▶ Du nombre de threads => faire varier 1, 2, 4, 8, 16, 32, ...
  - ▶ De l'architecture physique de la machine
    - Nombre de cœurs physiques, NUMA, hyper-threading
    - mandelbrot, goedel (accessible depuis mandelbrot), votre machine

## ETAPE 3 : Les graphiques

- ▶ **Faire de bons graphiques est très dur**
  - ▶ Avoir trop de graphiques rend l'analyse compliquée
  - ▶ Mettre trop d'information sur un seul graphique rend l'analyse compliquée
- ▶ **Vos graphiques**
  - ▶ Doivent clairement indiquer quel type de résultat ils présentent (typiquement le titre)
  - ▶ Clairement indiquer ce qui est utilisé pour les axes, avec les éventuels métriques
    - Nombre (threads), Temps (secondes), ...



# Un autre (pas très bon) exemple



## ETAPE 4 : L'analyse

- ▶ **Si vous arrivez à cette étape, vous avez déjà fait énormément de choses**
- ▶ **Il ne faut pas s'arrêter, ce serait dommage!**
- ▶ **A partir des graphiques, essayez de conclure sur**
  - ▶ Sur quelle machine, quelle taille de vecteur est intéressant de paralléliser
    - Quelle est l'accélération?
  - ▶ Est-ce le même comportement sur toutes les machines?
  - ▶ Est-ce que cela ralentit au-delà d'un certain nombre de threads? Peut-on corrélér ceci avec l'architecture de la machine?
  - ▶ ...
- ▶ **Alors, conclusion globale : il faut paralléliser quand? 😊**