Efficacité de la parallélisation :

- Le résultat « idéal » qu'on peut espérer est : **temps_total = N * temps_Nthreads**

  - En variant le nombre de thread entre 1 et 50, on remarque que tant que N est grand le résultat n'est plus celui attendu. Cela est du à l'exclusion mutuelle et du faite qu'on ne peut pas exécuter tous les processus au même temps, et donc **temps_Nthreads** est plus grand que celui espéré.

  - Le surcoût induit par la gestion des threads peux être calculer en utilisant la formule :

    surcoût = moyenne(temps_total) - N*moyenne(temps_nonSynchronisée)

On a donc pour :

N = 1	=>	**surcoût = -605 ms**	N = 3	=>	**surcoût = -2680 ms**
N = 5	=>	**surcoût = -2489 ms**	N = 10	=>	**surcoût = -11694 ms**
N = 15	=>	**surcoût = -22343 ms**	N = 20	=>	**surcoût = -22682 ms**
N = 25	=>	**surcoût = -13364 ms**	N = 30	=>	**surcoût = -1598 ms**
N = 40	=>	**surcoût = 73968 ms**	N = 50	=>	**surcoût = 176335 ms**

Coût de la cohérence :

- Les valeurs affichées dans le cas où il n'y a pas préemption du processeur entre threads sont :

  NB_IT * NB_IT_INTERNES, 2 * NB_IT * NB_IT_INTERNES, ….. , N * NB_IT * NB_IT_INTERNES

- Dans le cas où la gestion des activités partage le temps processeur par quantum de temps entre threads, les valeurs affichées sont aléatoire croissantes et la dernière valeur sera **N * NB_IT * NB_IT_INTERNES**

- La politique suivie par la JVM utilisée pour le test est la deuxième càd la gestion des activités partage le temps processeur par quantum de temps entre threads.

- On remarque que la valeur finale du compteur n'est pas égale au nombre total d'itérations, cela est du à l'accès des processus à la valeur **cpt** au même temps et donc ils l'incrément au même temps ce qui cause l'incrémentation de **cpt** par 1 au lieu de la valeur souhaitée.

- En ajoutant le bloc **synchronized**, on remarque que le résultat est correct. On remarque que le coût de l'utilisation de ce mécanisme en plaçant la boucle interne dans le bloc **synchronized** est moins élevé par rapport à celui en plaçant uniquement l'incrémentation de la boucle interne dans le bloc **synchronized.**

- En utilisant un objet de la classe **java.util.concurrent.atomic.AtomicLong** pour le compteur est bien garantie, et le coût est plus élevé que la solution en ajoutant le bloc **synchronized**.

- En déclarant le compteur comme **volatile** le résultat n'est plus correct, car **volatile** permet au processus d'être notifié au cas où un autre processus a aussi accès à la variable au même temps mais ne l'empêche pas d'y accéder et donc la variable peut être modifier au même temps.

- Pour conclure les trois mécanismes (en plaçant la boucle interne dans le bloc **synchronized**, en plaçant uniquement l'incrémentation de la boucle interne dans le bloc **synchronized**, et en utilisant un objet de la classe **java.util.concurrent.atomic.AtomicLong**) permettent de corriger le resultat, tandis que le mécanisme en déclarant le compteur comme **volatile**, ne permet pas la correction du résultat. Pour ce qui est coût, on peut les ordonner du moins coûteux au plus coûteux comme suit : (exemple pour N = 3)
  1. En plaçant la boucle interne dans le bloc **synchronized** 4558 ms
  2. En utilisant un objet de la classe **java.util.concurrent.atomic.AtomicLong** 131960 ms
  3. En déclarant le compteur comme **volatile** 146972 ms
  4. En plaçant l'incrémentation de la boucle interne dans le bloc **synchronized** 252425 ms