

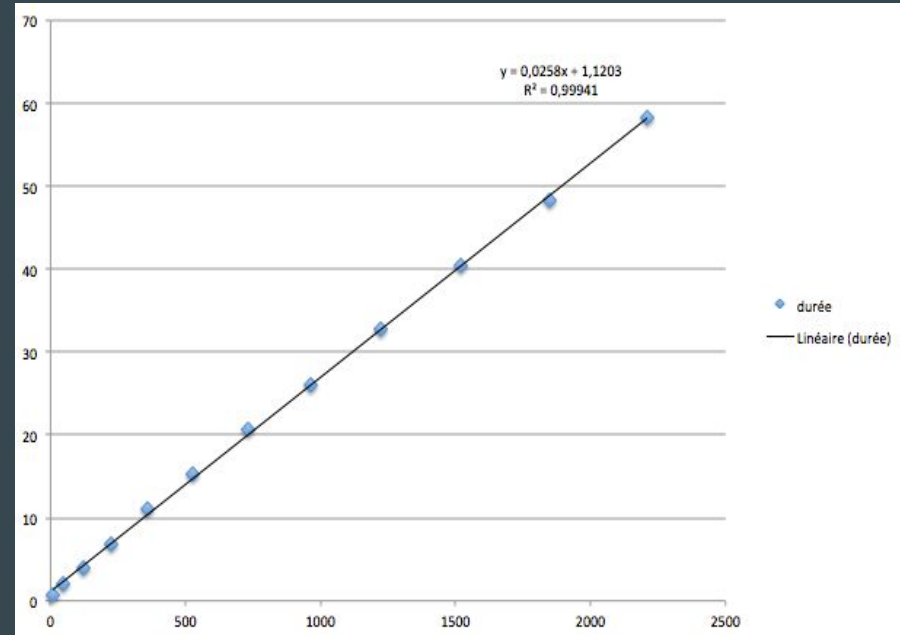
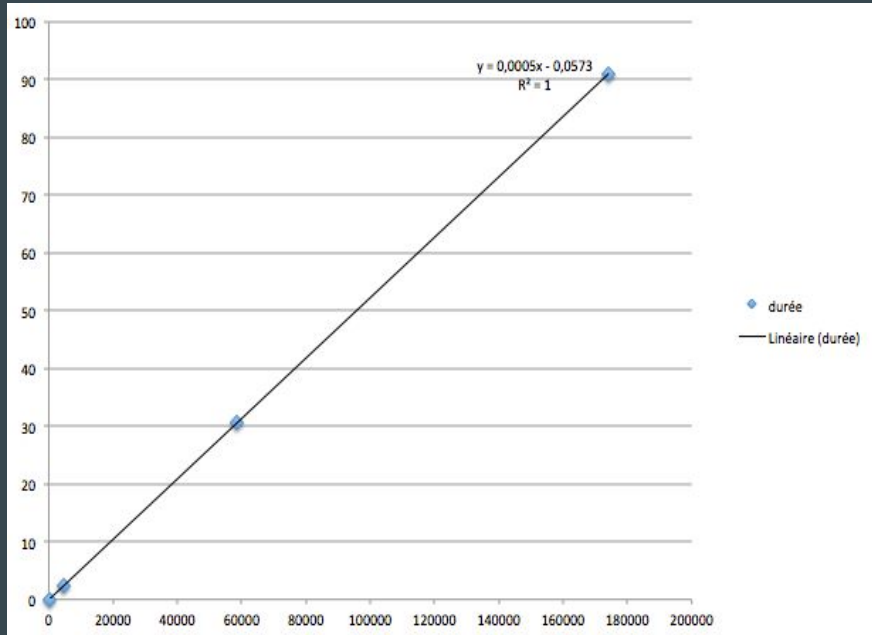
Parallélisation de la recherche de patterns dans une séquence

...

Soutenance de projet - INF560 - Raphaël Montaud et Hugo
Dobbelaere

1) Première approche théorique

$$c = m * n^2$$



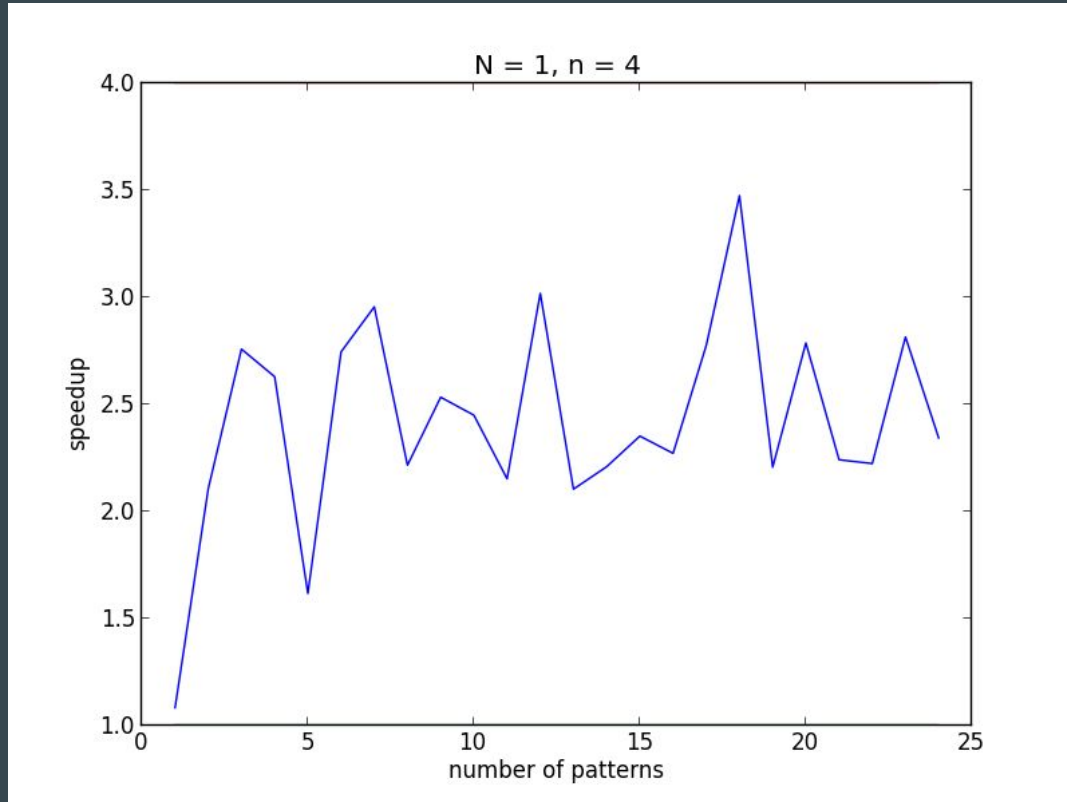
2) Parallélisation sur les patterns

On a k tâches à répartir entre r process.

- *trier les tâches par ordre décroissant*
- *tant qu'il reste des tâches:*
 - *donner la plus grande tâche restante au process qui a le moins de travail*

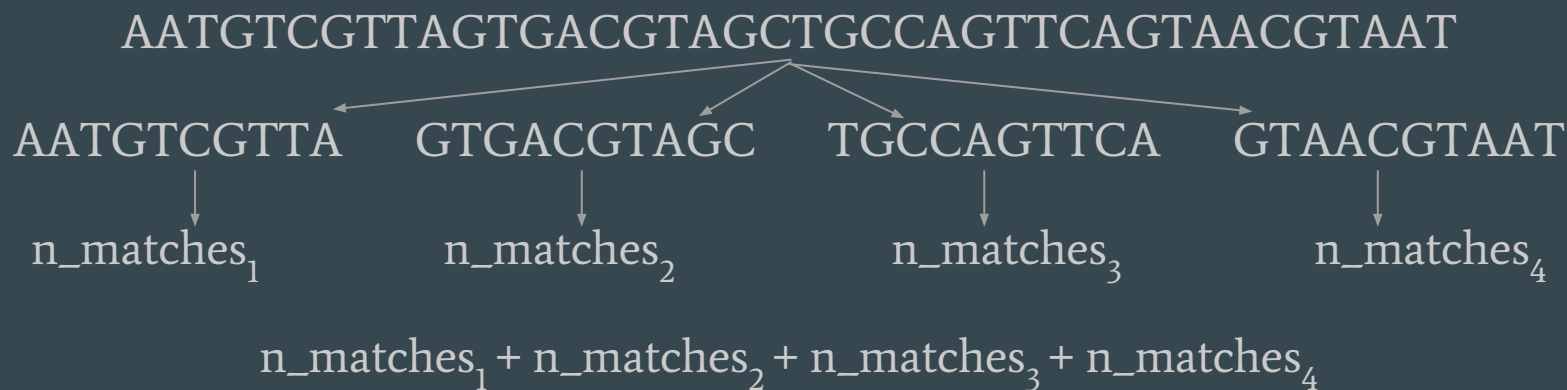
Optimal ? Converge vers une optimisation optimale, mais en pratique...

3) Parallélisation sur les patterns (2)

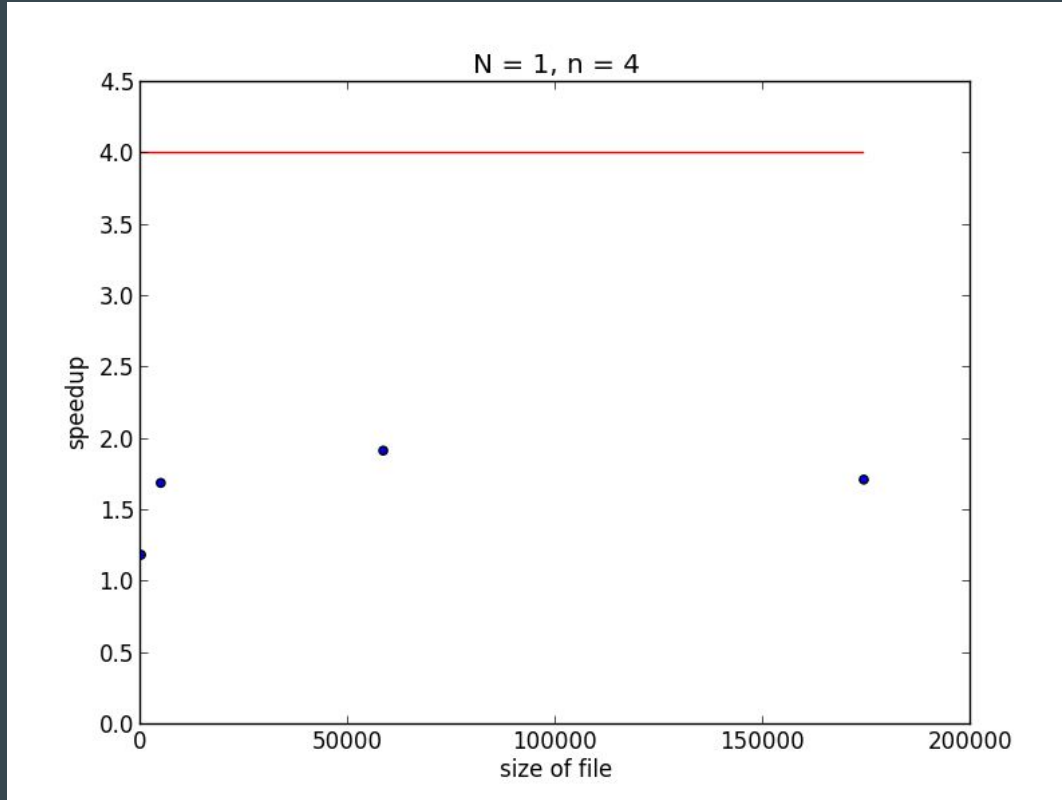


4) Parallélisation sur la séquence: MPI

- Envoyer des morceaux de fichiers à chaque rang: *scatter*
- Prendre de soin de “déborder”.
- calcul de la distance de Levenshtein
- Réduction

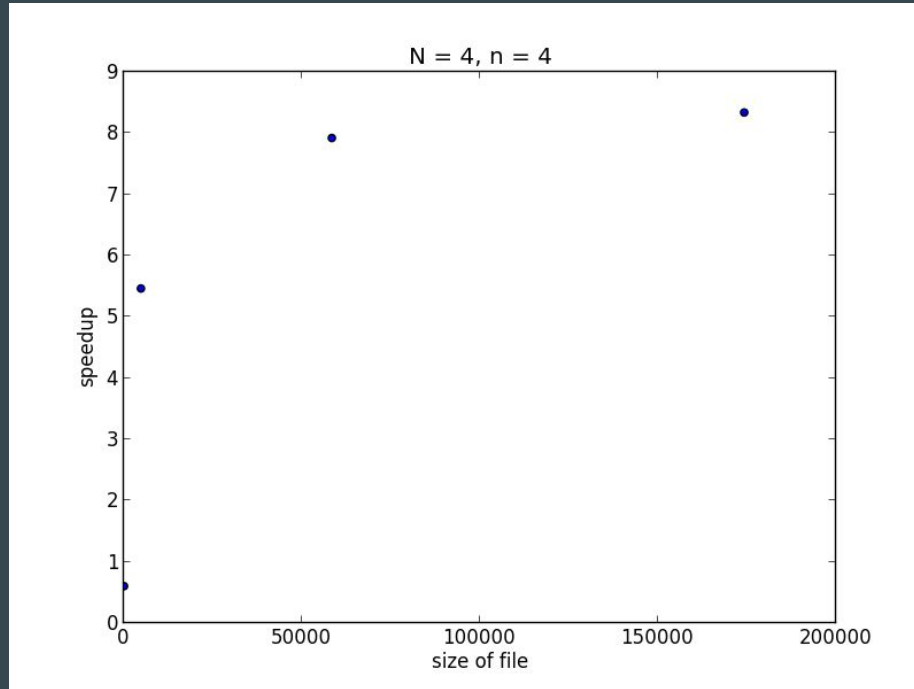


4) Parallélisation sur la séquence: MPI (2)




4) Parallélisation sur la séquence: MPI+OpenMP

Chaque rang MPI multiprocess ses calculs.



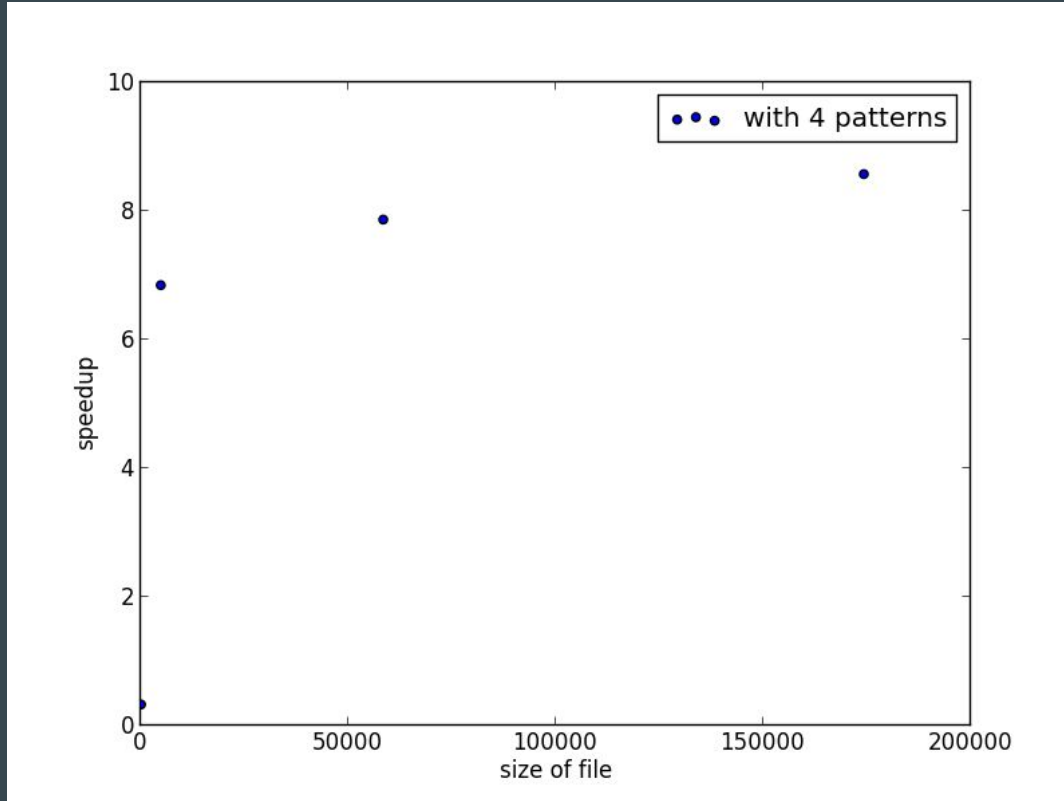
5) Parallélisation sur la séquence: Cuda

Reformater les fonctions pour aller avec le mode opératoire de Cuda:

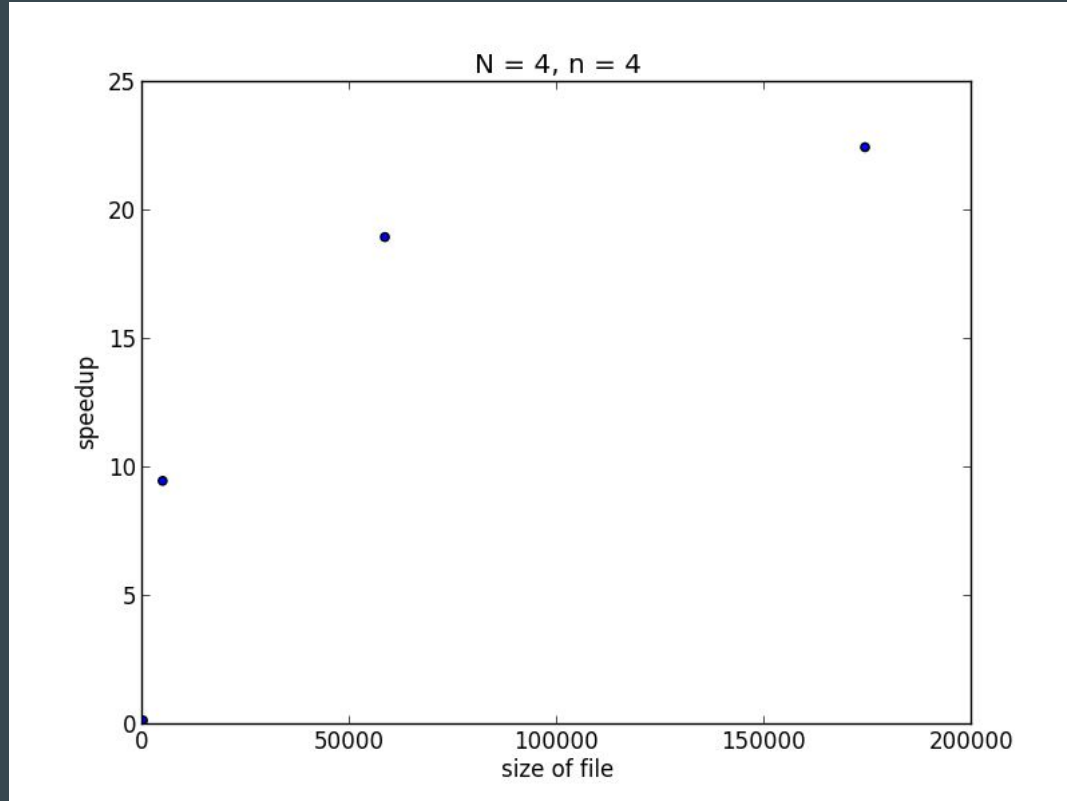
```
__global__  
void cuda_levenshtein(char *s1, char *s2, int len, int * result, int n_max, int i_0) {
```

```
int levenshtein(char *s1, char *s2, int len, int * column) {
```


5) Parallélisation sur la séquence: Cuda (2)



5) Parallélisation sur la séquence: MPI+CuDa



6) Conclusion

Sur un cluster de 30 machines, chacune ayant à disposition un GPU et 8 coeurs.

Implémentation	Temps de calcul (s)	Accélération
Normale	1767	1
MPI+OpenMP	17	104
MPI+Cuda	8	221

7) Améliorations possibles

- Rendre automatique la lecture des outils à disposition et leur utilisation
- Implémentation hybride MPI+Cuda+OpenMP: sur ce cluster on espère réduire de 30% le temps de calcul.

8) Un mot sur nos méthodes

- Git et github
- Automatisation de la compilation en python
- Tests en python
- Interface python plus simple (qui nous a servi notamment pour les figures)