

CSC5001 - Projet APM

Alexis LE GLAUNEC

Alice ZHEN

Une parallélisation hybride

1. Parallélisation OpenMP
2. Parallélisation MPI
3. Hybridation 1 et 2

Hypothèses de départ

En définissant :

- $N_patterns$: nb de patterns
- N : taille de la séquence d'ADN
- Max_CPU : nb max de CPU
- $Max_pattern$: taille max de pattern

Pour la suite, on suppose que :

$$N \gg N_patterns$$

$$N \gg Max_CPU$$

$$N \gg Max_pattern$$

Parallélisation OpenMP

Idée : découper l'analyse de la
séquence d'ADN entre des
threads

Comment ?



Parallélisation de la boucle for



Choix du scheduling intelligent



Protection des accès
concurrents

Parallélisation avec OpenMP:

```
#pragma omp parallel for schedule(static) private(j)
for ( j = 0 ; j < n_bytes ; j++ )
{
    column = int[size pattern+1];
    int distance = 0 ;
    int size ;

    size = size pattern ;
    if ( n_bytes - j < size_pattern )
    {
        size = n_bytes - j ;
    }

    distance = levenshtein( pattern[i], &buf[j], size, column );

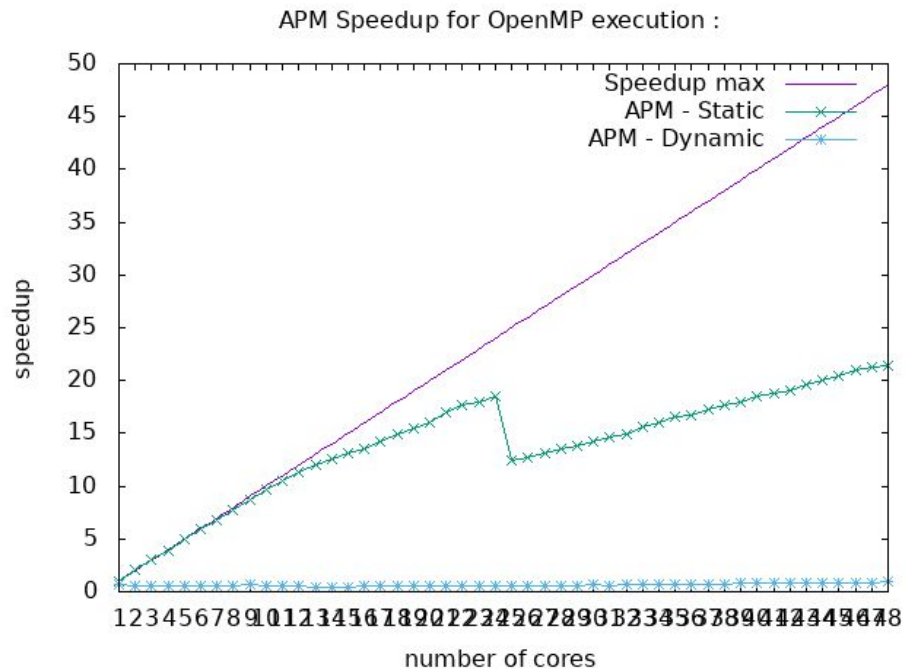
    if ( distance <= approx_factor ) {
        #pragma omp atomic
        n_matches[i]++ ;
    }
}
```

}
Parallélisation de la 2ème boucle for
Choix de schedule static
Column initialisé localement

}
Protection de la section critique

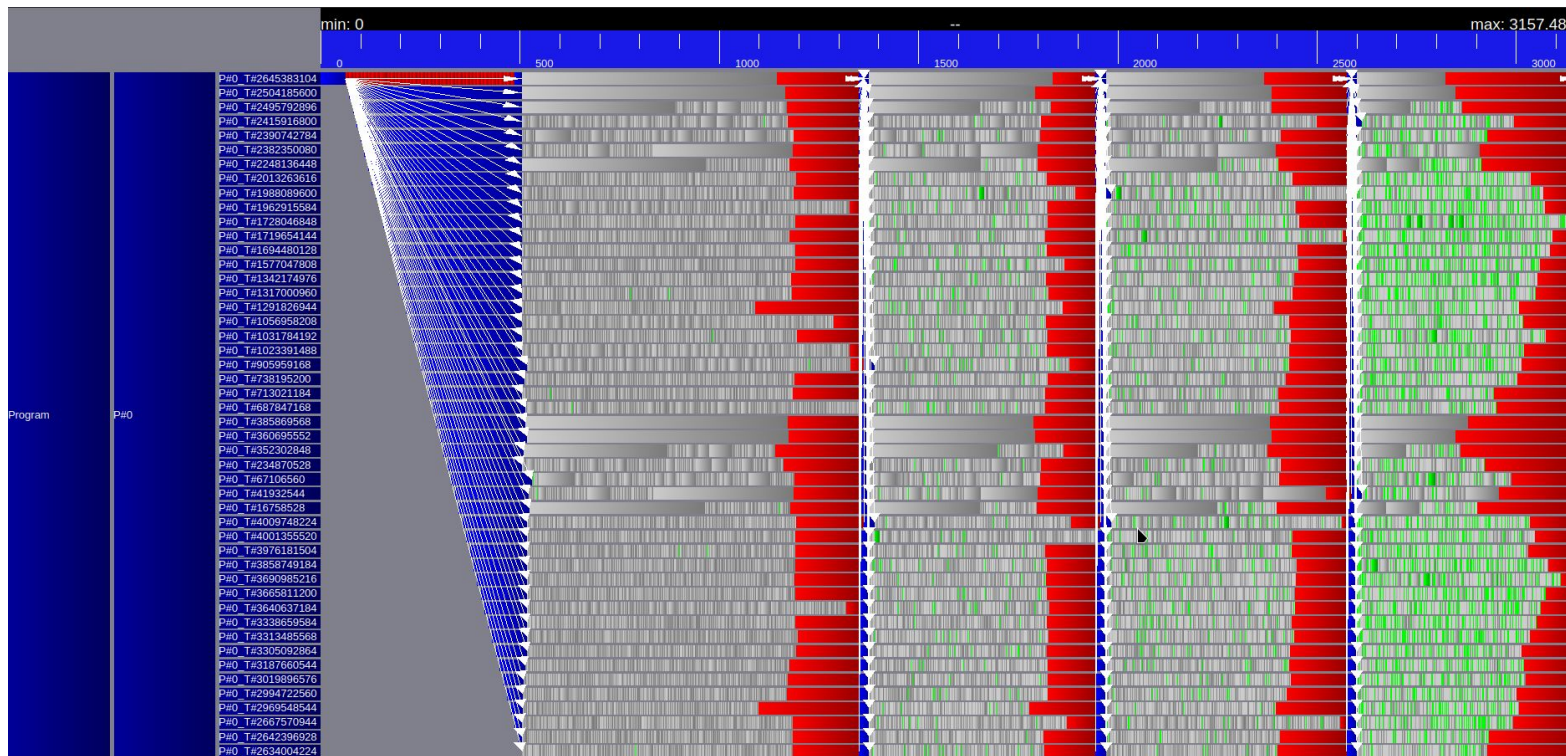
Choix du type d'ordonnement

```
./apmOMP 0 chr2.fa ATCG TTC AAA AT
```



Analyse de trace OpenMP avec EZtrace

```
export OMP_NUM_THREADS=48; eztrace -t "omp" ./apmOMP 0 chr2.fa ATCG TTC AAA AT
```



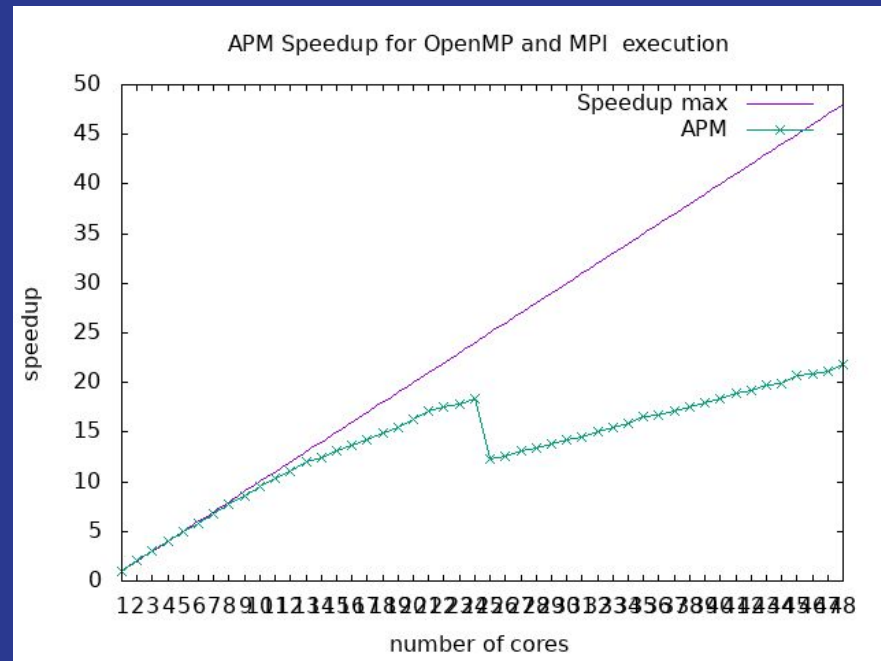
Améliorations possibles :

- Construire un bloc parallèle englobant les deux boucles :
 - ✚ Moins d'attente : pas de synchro pour chaque pattern, et fork/join
 - Mauvais speedUp : ne passe pas à l'échelle...
 - Problème de concurrence non résolu
- Tester les ordonnancements avec des chunk sizes != 1
- Ne plus considérer l'hypothèse $N \gg N_{\text{patterns}}$


```
./apmOMP 0 chr2.fa ATCG TTC AAA AT
```

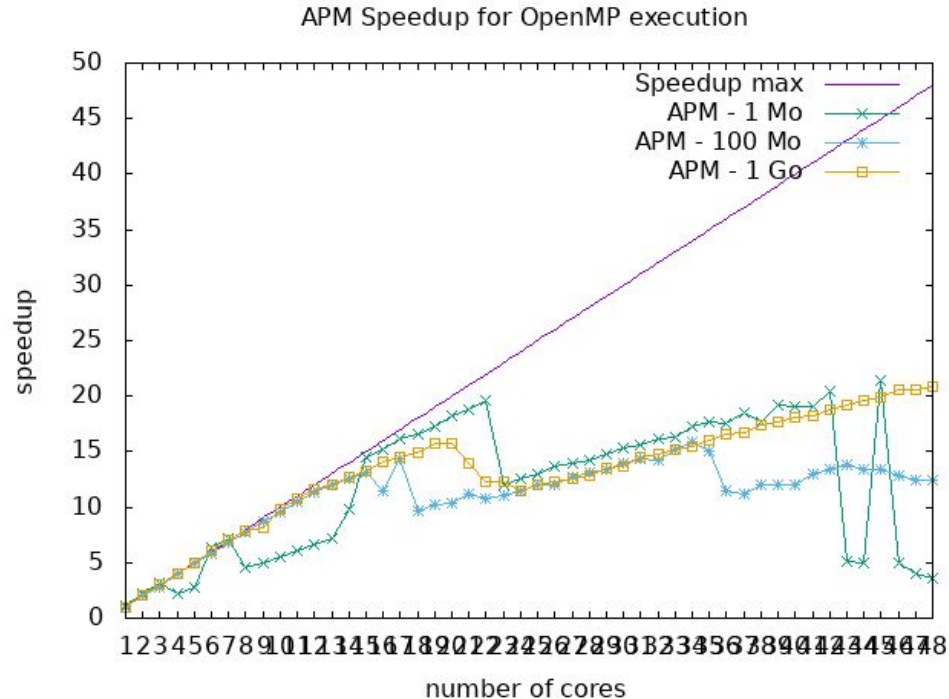
Analyse du speedup OpenMP

- Speedup presque parfait jusqu'à 24 coeurs
- Après :
 - Effet NUMA : hyperthread
 - Taille du problème



Analyse du speedup OpenMP




En testant sur plusieurs tailles de problème :



Parallélisation MPI

Idée : découper la séquence
d'ADN entre les noeuds MPI

Comment ?

-  Maître : subdivise la séquence entre les noeuds et envoie les sous-séquences
 -  Tous les noeuds : exécute l'algorithme APM sur leurs sous-séquences
 -  Mise en commun des occurrences
-

Parallélisation avec MPI: Processus 0

```
buf = read_input_file( filename, &n_bytes ) ;
```

} Lit le fichier

```
int start = 0;
int end = n_bytes / world - 1 + max_len_pattern - 1;
for (i = 1; i < world; i++) {
    start += n_bytes / world;
    end += n_bytes / world;
    if (end > n_bytes || i == world - 1) {
        end = n_bytes;
    }
    diff = end - start + 1;
```

} Divise la séquence en `world` sous séquences

```
MPI_Send(&diff, 1, MPI_INTEGER, i, 0, MPI_COMM_WORLD);
MPI_Isend(&buf[start], diff, MPI_BYTE, i, 0, MPI_COMM_WORLD, &req[i]);
}
diff = n_bytes / world - 1 + max_len_pattern - 1;
```

} Envoie à chaque processus sa sous-séquence

Parallélisation avec MPI: autres processus

```
MPI_Recv(&diff, 1, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
```

} Reçoit la taille de sa sous-séquence

```
buf = (char *)malloc( diff * sizeof ( char ) );
```

```
if ( buf == NULL )
```

```
{
```

```
    fprintf( stderr, "Unable to allocate %ld byte(s) for buf array\n",  
             diff );
```

```
    return EXIT_FAILURE;
```

```
}
```

} Alloue de la mémoire pour recevoir la sous-séquence

```
MPI_Recv(buf, diff, MPI_BYTE, 0, 0, MPI_COMM_WORLD, &status);
```

} Reçoit la sous-séquence à traiter

Parallélisation avec MPI: mise en commun

```
if (rank == 0) {  
    MPI_Status status;  
    int *n_matches_recv = (int *) malloc(nb_patterns * sizeof(int));
```

} P0 alloue de la mémoire pour recevoir les occurrences

```
    for (i = 1; i < world; i++) {  
        /* Clear the buffer */  
        memset(n_matches_recv, 0, nb_patterns * sizeof(int));  
        MPI_Recv(n_matches_recv, nb_patterns, MPI_INTEGER, i, i, MPI_COMM_WORLD,  
&status);  
        for (j = 0; j < nb_patterns; j++) {  
            n_matches[j] += n_matches_recv[j];  
        }  
    }  
    free(n_matches_recv);
```

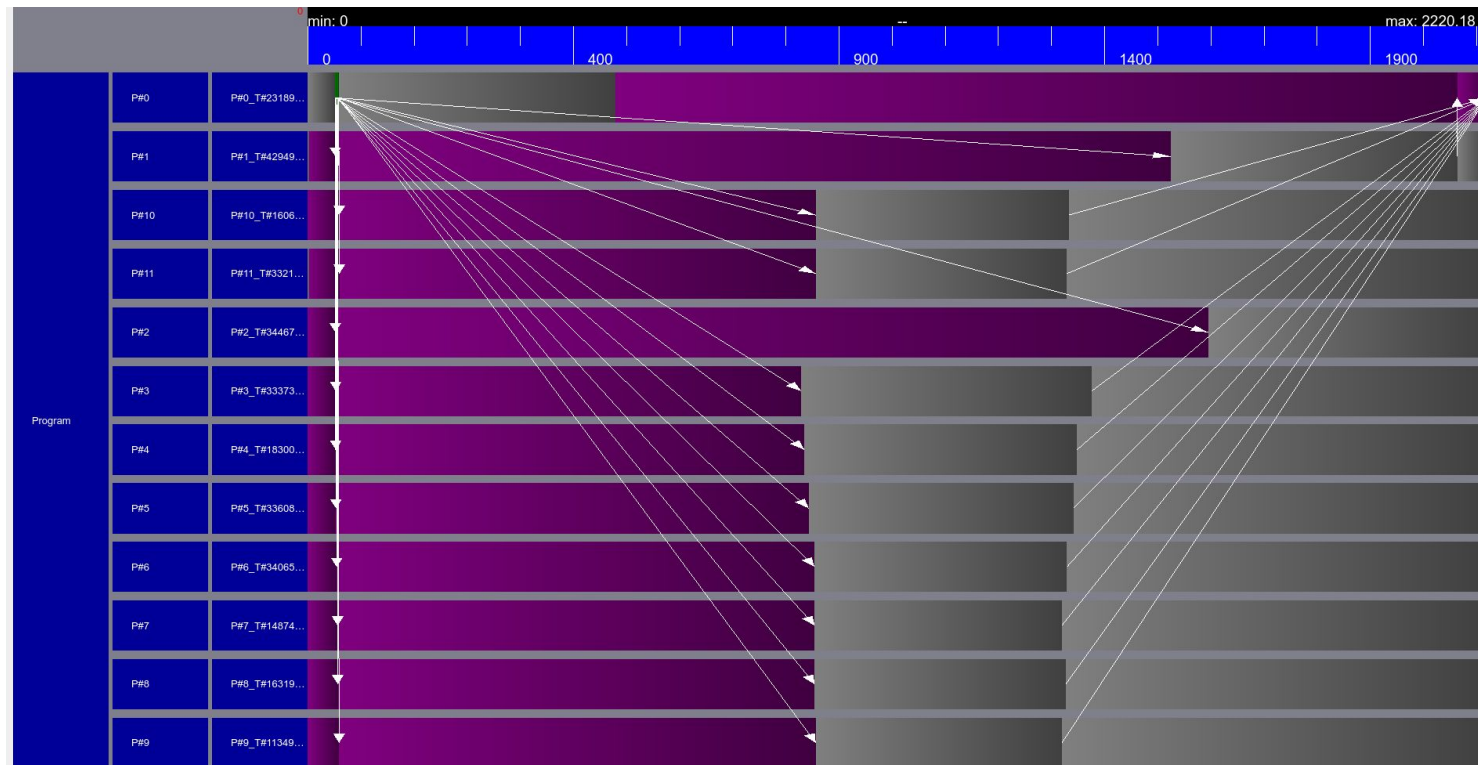
} P0 reçoit de chaque processus les occurrences trouvées et les somme à ses propres résultats

```
} else {  
    MPI_Send(n_matches, nb_patterns, MPI_INTEGER, 0, rank, MPI_COMM_WORLD);  
}
```

} P1-N envoient leurs résultats à P0

Analyse de trace MPI avec EZtrace

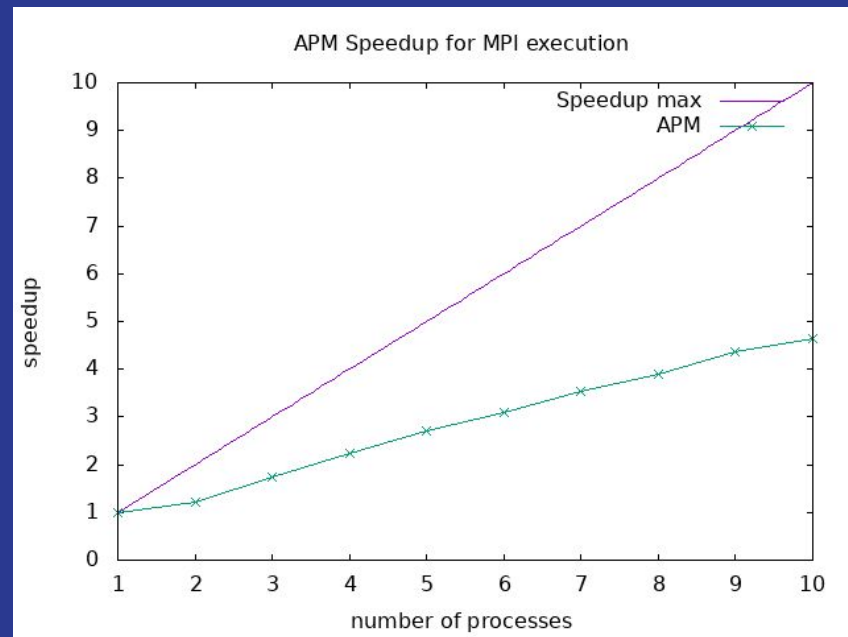
```
mpirun -np 12 -f hosts eztrace -t "mpi" ./apmMPI 0 chrY.fa CATG AAC AGTACA
```



Analyse du speedup MPI

- Speedup: coût de l'envoi de messages à prendre en compte

```
mpirun -np $i -f hosts ./apmMPI 0  
dna/lmo chrY.fa AAT TGA TTTT
```



Hosts: 3a401-01 à 3a401-10

Parallélisation hybride

Idée : mettre en commun les parallélisations MPI et OpenMP

Comment ?



On fusionne les deux codes précédents



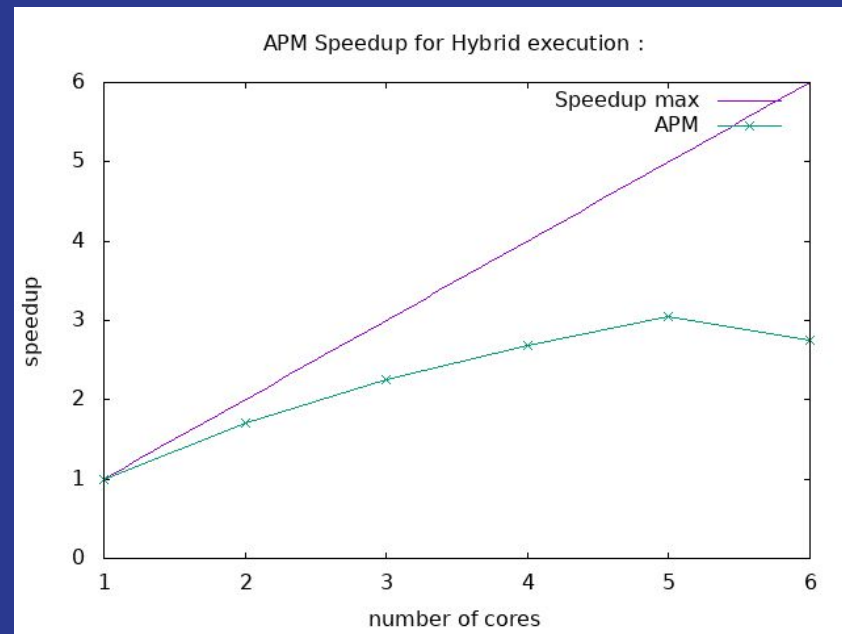
On étudie les performances en fonction :

- ◆ Du nombre de noeuds MPI
- ◆ Du nombre de coeurs utilisés

Analyse du speedup OpenMP

- Problème identique à la version OpenMP
- Overhead MPI faible

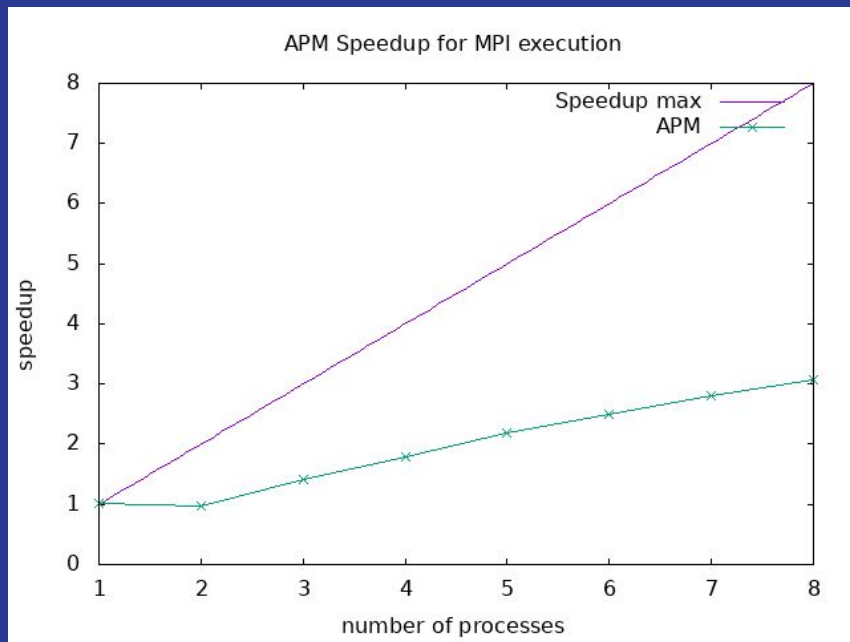
```
mpirun -np 4 -f hosts ./apmParallel 0  
dna/chr2.fa AAT TGA TTTT
```



Analyse du speedup MPI

- Performances analogues à l'algorithme apmMPI
- Courbe un peu moins bonne qu'avec apmMPI
- Seulement sur 8 machines : certains machines ne répondaient pas

```
mpirun -genv OMP_NUM_THREADS 6 -np $i -f hosts  
./apmParallel 0 dna/1mo chrY.fa AAT TGA TTTT
```



Hosts: 3a401-01 à 3a401-08

Conclusion

- Bon speedUp par rapport :
 - Au nb de machines du cluster
 - Au nombre de coeurs par machine
- Problème des hyperthreads
- Gains réels pour des fichiers de grande taille (+100 Mo)
- Angles d'améliorations pour un speedup idéal :
 - Ne pas créer des threads pour chaque pattern
 - Envoi asynchrone pour le noeud MPI master