HAL
open science

Optimisation des performances du programme mpiBLAST pour la parallélisation sur grille de calcul

Mohieddine Missaoui

To cite this version:

Mohieddine Missaoui. Optimisation des performances du programme mpiBLAST pour la parallélisation sur grille de calcul. 2006. hal-00678054

HAL Id: hal-00678054

https://hal.science/hal-00678054

Submitted on 12 Mar 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Optimisation des performances du programme mpiBLAST pour la parallélisation sur grille de calcul

Mohieddine MISSAOUI *

Rapport de Recherche LIMOS/RR-06-10

20 novembre 2006

* Contact : missaoui@isima.fr

Résumé :

Le programme mpiBLAST est un programme d'alignement local de séquences destiné aux architectures parallèles telles que les clusters ou les grilles de calcul. La performance du mpiBLAST est optimale lorsqu'on utilise la totalité des nœuds du cluster l'ISIMA. La meilleure performance est obtenue lorsque (n+1) nœuds sont utilisés pour le calcul tandis que la base de données est fragmentée sur n nœuds. Nous avons démontré une légère chute de performance lorsque le nombre de nœuds utilisés pour le calcul est très supérieur au nombre de fragments de la base de données. Le programme parallèle d'alignement de séquence mpiBLAST est adapté pour l'application que nous développons et permet d'aller jusqu'à 8 fois plus vite que sur une architecture classique. L'architecture du cluster est une étape de transition vers l'architecture de grille de calcul.

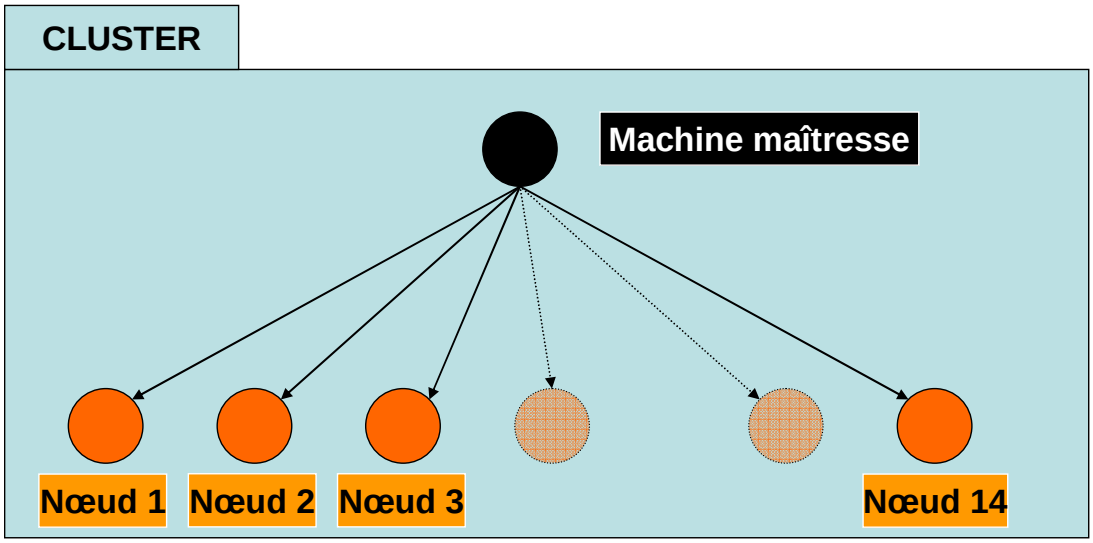
Introduction et objectifs

Le travail suivant entre dans le cadre de l'étude de faisabilité de la migration d'une application dédiée à la conception de sondes vers le cluster de l'ISIMA (http://www.isima.fr). Cette partie consiste essentiellement à trouver des points d'optimisation des algorithmes utilisés dans cette application et qui consomment beaucoup de temps. Ces algorithmes intègrent des appels nombreux à des programmes d'alignement et de comparaison de séquences (nucléiques et protéiques). L'objectif est donc d'étudier le comportement des versions parallèles de ces programmes destinés aux architectures telles que les Clusters ou les grilles de calcul.

Les tests effectués sur le cluster de l'ISIMA montreront l'intérêt ou non d'utiliser des algorithmes parallélisés pour cette application.
Afin de dresser un bilan de comparaison préliminaire, il est important de décomposer le problème en plusieurs tâches.
La première étape consiste à étudier le programme d'alignement de séquences BLAST (Basic Local Alignement Search Tool) qui est un programme de comparaison de séquences (étude d'homologies) très utilisé pour identifier dans une banque de données génomique les séquences ayant des homologies avec une séquence donnée.

Matériels et méthodes

L'architecture utilisée pour les tests de performance est une architecture de cluster classique qui comporte une machine maîtresse et 14 nœuds.
La configuration du cluster se compose de la machine maîtresse qui est une machine biprocesseurs Intel (R) Xeon TM 2.40Ghz, 2Go de RAM, et 150Go de mémoire disque qui tourne sous Linux 2.4 avec la version 7.3 de Red Hat.
Les nœuds sont des machines identiques avec chacune 4 processeurs Intel(R) Xeon(TM) 2.40Ghz, 2.5Go de RAM et 65Go de mémoire disque qui tourne sous Linux 2.4 avec la version 7.3 de Red Hat.



Architecture du cluster

L'importance d'un programme d'alignement de séquence tel que BLAST, et l'arrivée de la programmation parallèle qui a pour objectif de diminuer le temps de calcul des projets de grande envergure comme le « Human Genome Project » a permis l'apparition de logiciels de parallélisation du BLAST [4] [5].

Une étude préalable a été effectuée [1] sur le fonctionnement et l'évaluation des performances du mpiBLAST. Les auteurs ont montré que le programme mpiBLAST permettait d'accélérer les calculs grâce à la fragmentation de la base sur les différents nœuds du cluster et grâce aussi au calcul effectué par chaque nœud sur un seul fragment de la base. Un bilan complet des performances du mpiBLAST a été fait. Les tests de performance ont été effectués sur un cluster Linux.

Une nouvelle stratégie d'optimisation a pu démontrer l'intérêt de diminuer le nombre d'entrées/sorties [2]. En jouant sur l'optimisation d'entrées/sorties et la communication entre les différents nœuds et la machine maîtresse du cluster Linux utilisé, la performance du mpiBLAST croit considérablement.

Dans un premier temps, nous avons testé sur un seul nœud le comportement du programme BLAST local [3]. Pour cela nous avons installé le package BLAST du ncbi (version 2.2.15). Nous avons ensuite formaté la base de données protéique avec la commande *formatdb* du même package. On obtient ainsi trois fichiers nécessaires au lancement du programme BLAST (*protDB.fasta.phr*, *protDB.fasta.pin*, *protDB.fasta.psq*) et un fichier log (*formatdb.log*).

Ensuite, nous avons lancé le programme BLAST sur cette base de données de *692Mo* avec une séquence protéique unique en entrée de longueur d'environ 1500 acides aminés. Le test est lancé sur le noeud01 du cluster de l'ISIMA. Le temps d'exécution est de 34.4s.
Ce temps d'exécution servira comme temps de référence pour les tests qui seront présentés dans la partie suivante.

Tests de performance

Pour étudier la performance du mpiBLAST (http://mpiblast.lanl.gov/) sur le cluster de l'ISIMA, il fallait installer les logiciels permettant de lancer ce genre de programme sur une plate-forme de calcul parallèle. Le logiciel MPICH (programme utilisant la librairie MPI: *Message Passing Interface*) permettant de lancer des calculs distribués sur notre cluster a été installé et configuré au préalable. Le programme mpiBLAST a également été installé sur les différents nœuds du cluster. Une étape de configuration et de personnalisation des programmes a été nécessaire avant de commencer les tests.

Avant de lancer le programme mpiBLAST sur les nœuds du cluster, il a fallu configurer les fichiers de configuration et la base de données protéiques à tester.
D'abord, nous avons configuré le fichier *mpiblast.conf* qui sera utilisé par la commande *mpiformatdb* et la commande *mpiblast*. En effet, le fichier contient les informations nécessaires sur l'emplacement de la base et des futurs fragments associés. Lors du lancement du mpiBLAST sur le cluster, la commande lit dans ce fichier les chemins vers les fragments et la base formatée. Pour assurer un fonctionnement correct du cluster la base sera formatée sur un disque partagé par les différents nœuds sachant que le système d'exploitation est Linux. Les fragments nécessaires au lancement du BLAST sont en revanche copiés (comme indiqué dans le fichier de configuration) vers le répertoire local de chaque nœud. A chaque nœud utilisé on associe un ou plusieurs fragments de base.

Ensuite, il faut configurer le fichier nécessaire au lancement du mpiBLAST et qui contient les noms de machines (nœuds participant au calcul) *machines.LINUX* : ce fichier comporte uniquement les différents noms des nœuds présents dans le cluster. Pour utiliser un nœud il suffit d'ajouter une ligne dans ce fichier comportant le nom de la machine hôte. Lors du lancement d'une commande qui s'exécute sur le cluster, il faut préciser avec l'option –*machinefile* le chemin du fichier.

L'étape suivante consiste à lancer le mpiBLAST sur le cluster de l'ISIMA.

Lancement des tests

Le programme mpiBLAST est lancé sur les différents nœuds du cluster avec la commande *mpirun* livrée avec MPICH (http://www-unix.mcs.anl.gov/mpi/mpich2/). Les tests sont effectués de façon linéaire en variant de manière croissante le nombre de nœuds utilisés et les fragments de la base de données qui seront distribués sur les noeuds.
Avant de lancer cette commande, il est nécessaire de lancer la fragmentation de la base (qui sera la même pour tout les tests faisant varier le nombre de nœuds afin de garder une cohérence des résultats). Il est important également de faire les tests dans les mêmes conditions d'exécution au niveau du cluster. En effet, il faut s'assurer par exemple qu'aucun programme qui ralentirait l'exécution du mpiBLAST n'est en cours d'exécution. Nous avons exécuté les différentes commandes en l'absence de ce genre de programmes.
Nous avons fragmenté successivement la base en 2, 3, 4… et ce jusqu'à 12 fragments. La séquence d'entrée appelée *query* représente une séquence unique dans un fichier indépendant. Un plan d'exécution est un ensemble de lancement du programme mpiBLAST pour un nombre donné de fragments de la base de données en variant de 2 à N avec N = nombre total des nœuds sur le cluster. En effet, pour chaque plan d'exécution nous avons lancé une série de tests, avec pour chaque test un nombre de nœuds incrémenté de 1.
La base de données contre laquelle on lance les tests est rigoureusement la même utilisée pour le BLAST standard.

Soit X l'ensemble des plans d'exécution, et soit N le nombre total de nœuds du cluster. Soit $x \in X$ un plan d'exécution donné et $n < N$ le nombre de fragments de la base de données. A chaque nouveau plan d'exécution x utilisant n fragments, la base de données est totalement supprimée et une nouvelle commande de fragmentation avec (n+1) fragments est lancée et ce jusqu'à 12 fragments. De la même manière, pour un plan d'exécution x avec n fragments, nous lançons les calculs sur i nœuds, i variant de 2 à 13, sachant que i peut être supérieur au nombre des nœuds du cluster car les nœuds utilisées sont des machines biprocesseurs.

Résultats

Les tests effectués sur le cluster sont fait de manière à assurer la cohérence des données obtenues. En effet, un plan d'exécution est lancé plusieurs fois jusqu'à stabilisation du temps de calcul. Ce paramètre est important car l'utilisation d'un tel calcul se fera de façon répétitive des milliers de fois voire de millions de fois dans le logiciel qui utilisera ces commandes. On pourra parler alors de la limite vers laquelle tend le temps de calcul. Lors du lancement du mpiBLAST il y a deux phases principales d'exécution [1] :

- phase de copie des fragments sur les nœuds en question
- phase de lancement du blast sur chaque nœud

La première phase est exécutée uniquement la première fois où on lance le programme mpiBLAST. En revanche, à partir du deuxième lancement, la phase de lancement du programme est directement entamée car un test d'existence de fragment est fait avant de passer à la copie. En effet, si l'algorithme détecte la présence d'un fragment dans le répertoire local du nœud il envoie un processus à la machine maîtresse pour lui dire de lancer directement le programme sans faire de copie [1]. On obtient donc un temps de calcul inférieur lors du lancement de la même commande avec le même plan d'exécution la deuxième fois. Une troisième phase d'envoi et de concaténation des résultats est évidement effectuée mais elle est négligeable par rapport au temps de calcul du BLAST [1].

L'étape suivante a consisté à faire plusieurs tests pour chaque plan d'exécution (voir figure 1). D'abord, il est important donc de prendre la valeur moyenne du test (la valeur vers laquelle tendent les temps d'exécution pour un nombre de fragments donné et un nombre de nœuds donné). Ensuite, remettre en question les valeurs qui paraissent aberrantes à cause de l'utilisation, en cours d'exécution du programme mpiBLAST, du cluster par d'autres programmes. Nous avons gardé les valeurs cohérentes des tests de performance du mpiBLAST.

Résultat des tests et discussion

La figure 1 présente le plan d'exécution du mpiBLAST pour 4 fragments de la base de données : on calcule le temps d'exécution du programme sur le cluster en utilisant 2, 3, 4…jusqu'à 13 nœuds. On remarque que la courbe est caractéristique et est représentative de tous les plans d'exécution. Les autres courbes correspondantes aux autres plan d'exécution sont sensiblement les mêmes.

Le lancement s'est fait sur les 13 nœuds du cluster de l'ISIMA. La fragmentation de la base s'est faite sur plusieurs plans d'exécution comme expliqué dans la partie précédente.

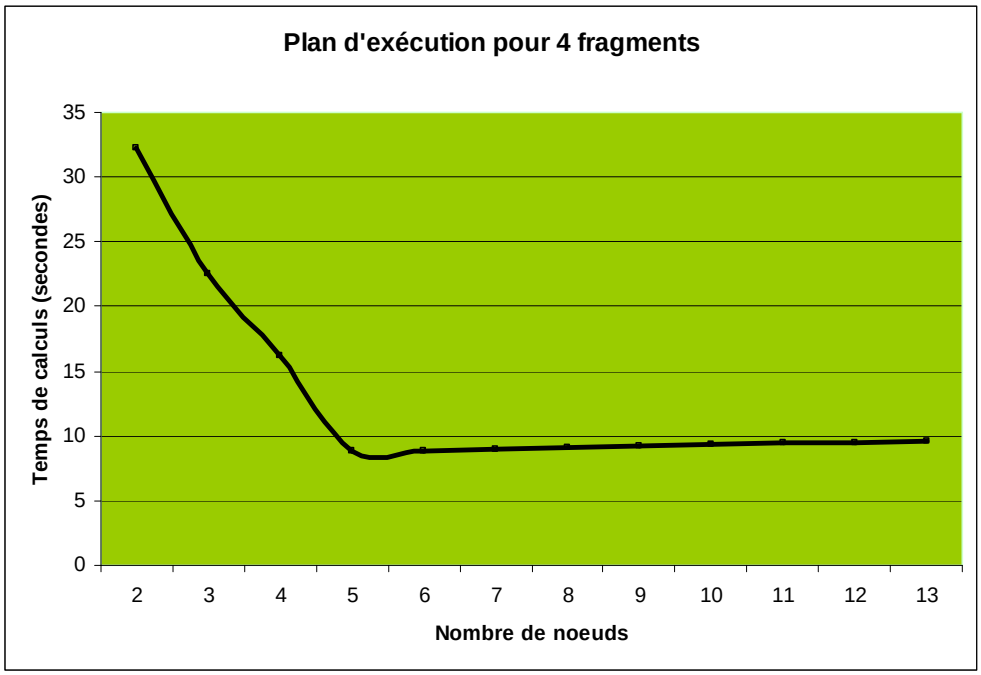


Figure 1 : courbe d'un plan d'exécution

L'interprétation des résultats par nombre plan d'exécution montre l'uniformité des tests. En effet, pour chaque plan d'exécution (n fragments) le temps de calcul en fonction du nombre de nœuds mis à contribution dans le calcul possède une courbe convexe. Le temps d'exécution diminue quand le nombre de nœuds augmente. Ce résultat paraît parfaitement logique. Un résultat surprenant a été observé : il s'agit du comportement légèrement croissant à partir de (n+2) nœuds pour n fragments. Plus le nombre de nœuds augmente moins le mpiBLAST est performant. Bien que l'augmentation soit sensiblement faible, elle représente tout de même un défaut pour le cluster. En effet, nous remarquons une diminution de la performance du mpiBLAST en augmentant le nombre de nœuds utilisés.

Une hypothèse qui pourrait expliquer ce comportement est le temps de communication du au lancement des processus inclus dans l'algorithme du mpiBLAST. L'algorithme envoi à partir de la machine maîtresse des processus à tous les nœuds qui sont utilisés lors du plan d'exécution. Le temps en plus par rapport (n = [6..13]) au temps minimal de calcul est probablement celui des processus qui renvoient leurs états d'exécution à la machine maîtresse sans avoir effectué de BLAST.
Les résultats sont confirmés par une deuxième courbe figure 2 qui varie légèrement d'un plan d'exécution à un autre suivant le nombre des nœuds utilisés.

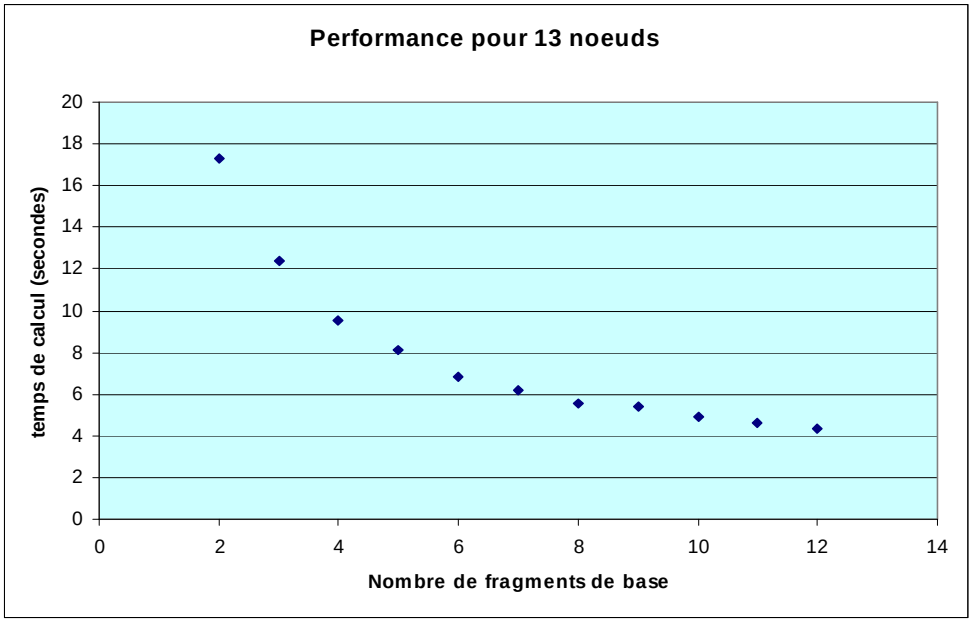


Figure 2 : Performance du mpiBLAST en fonction du nombre de fragments de base

En effet, lorsqu'on augmente le nombre de fragment le programme mpiBLAST devient plus performant jusqu'à ce qu'on atteigne le point minimal. D'après la figure 2, On constate une diminution du temps de calcul en fonction du nombre de fragments de la base de données. Autrement dit, l'algorithme du mpiBLAST devient de plus en plus performant lorsqu'on fragmente de façon uniforme les fragments sur les nœuds du cluster utilisés. On voit que la meilleure performance est obtenue pour 12 fragments quand on utilise 13 nœuds.

D'après [1], la meilleure performance est obtenue lorsqu'on lance le programme mpiBLAST sur i nœuds avec (i-1) fragments de base. Ce résultat est parfaitement confirmé pour notre cluster (voir figure 3).
L'objectif de ce travail est de pouvoir identifier le point critique où la performance atteinte est optimale pour pouvoir lancer des blasts sur le cluster. Le travail fait parti d'un projet plus conséquent qui consiste à développer des algorithmes capable de lancer de tels jobs sur le cluster de l'ISIMA dans un premier temps. L'étape suivante consiste donc à utiliser une architecture de grille de calcul (comprenant entre autre le cluster de L'ISIMA) pour augmenter encore plus la performance des programmes. Ces algorithmes seront développés pour la conception de sondes sur grille de calcul et entre dans le cadre d'un projet régional : INSTRUIRE (http://www.auvergrid.fr/).

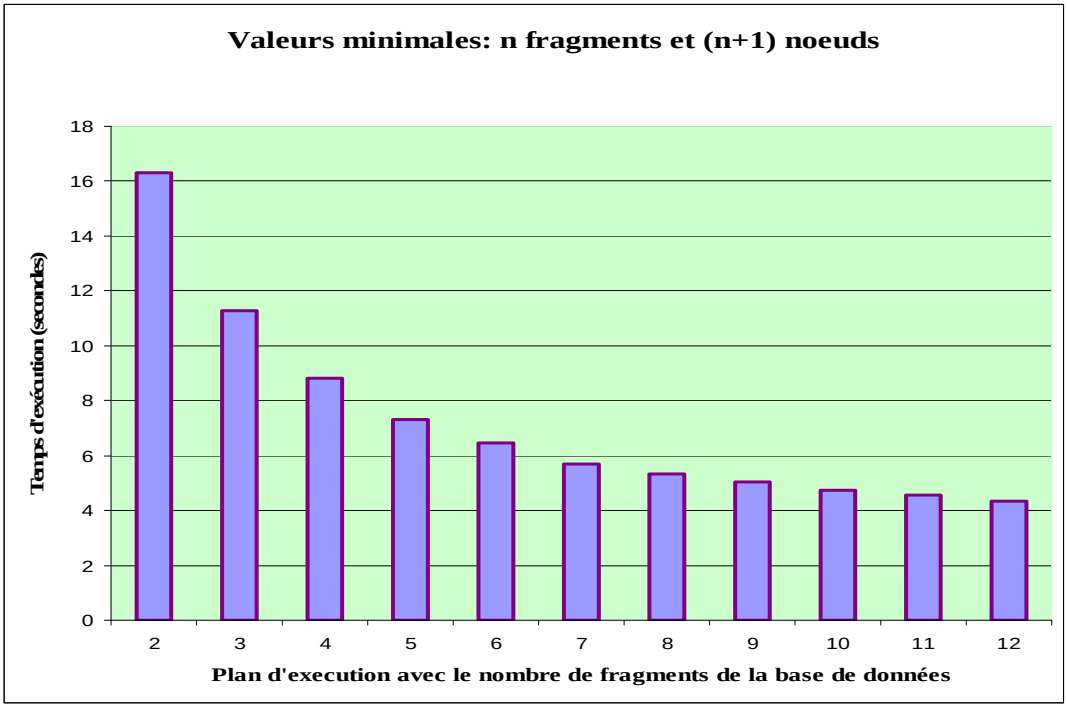


Figure 3 : Valeurs minimales pour les plans d'exécution

On peut constater que la meilleure performance est obtenue pour 13 nœuds et 12 fragments de bases. Le déploiement des algorithmes sera donc défini de façon à optimiser le temps de calcul.

Soit un plan d'exécution donné: pour i fragments le meilleur temps d'exécution est obtenu pour (i+1) nœuds. Car un nœud est utilisé pour la planification des jobs faite par un processus du mpiBLAST : c'est la conception même de l'algorithme [1].

Conclusion et perspectives

Nous avons pu démontrer que sur le cluster de l'ISIMA, la meilleure performance du mpiBLAST est atteinte lorsqu'on utilise la totalité des nœuds du cluster pour la fragmentation de la base. Il faut également faire participer tous les nœuds du cluster au calcul. Ce travail nous a permis de démontrer que la recherche d'homologie dans l'application peut être 8 fois plus rapide lorsqu'elle est exécutée sur le cluster que sur une seule machine. Ce module pourra donc être parallélisé, ce qui nous permettra d'augmenter les performances de notre application. Le travail à venir consiste à faire une étude similaire avec CLUSTALW le programme d'alignement multiple (global) de séquences utilisé pour l'identification d'espèces encore plus proches. Une version dédiée au cluster a été développée [6]. L'objectif est d'utiliser cette version pour notre application de recherche de sondes. Il s'agit dans un premier temps d'étudier les performances de CLUSTALW-MPI sur le cluster de l'ISIMA et en déduire l'utilité.

Remerciement : Je tiens à remercier Pr. D.R.C Hill pour ses conseils, je remercie également Pascale Gouinaud. Ce travail rentre dans le cadre du projet INSTRUIRE.

Références bibliographiques:

[1] Darling AE, Carey L, Feng W. The design, implementation, and evaluation of mpiBLAST. *Proceedings of ClusterWorld Conference and Expo in Conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003*, San Jose, CA, June 2003.

[2] Heshan Lin Xiaosong Ma Chandramohan, P. Geist, A. Samatova, N. Efficient Data Access for Parallel BLAST. Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International

[3] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. Journal of Molecular Biology, 215:403-410, 1990.

[4] R. Bjornson, A. Sherman, S. Weston, N. Willard, and J. Wing. Turboblast: A parallel implementation of blast based on the turbohub process integration architecture. In IPDPS 2002 Workshops, April 2002.

[5] R. Braun, K. Pedretti, T. Casavant, T. Scheetz, C. Birkett, and C. Roberts. Parallelization of local BLAST service on workstation clusters. Future Generation Computer Systems, 17(6):745-754, April 2001.

[6] Kuo-Bin Li ClustalW-MPI: ClustalW analysis using distributed and parallel computing. BIOINFORMATICS Vol. 19 no. 12 2003, pages 1585–1586, March 2, 2003