HAL
open science

Recherche dans les banques d'ADN par indexation parallèle

van Hoa Nguyen, Dominique Lavenier

To cite this version:

van Hoa Nguyen, Dominique Lavenier. Recherche dans les banques d'ADN par indexation parallèle. RVIF, Feb 2006, HCM, Vietnam. inria-00180372

HAL Id: inria-00180372

https://inria.hal.science/inria-00180372

Submitted on 19 Oct 2007

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.

Recherche dans les banques d'ADN par indexation parallèle

Van Hoa Nguyen
Institut Francophone d'Informatique
Hanoi, Vietnam
Email: nvhoa@ifi.edu.vn

Dominique Lavenier
CNRS / IRISA
Rennes, France
Email: lavenier@irisa.fr

***Abstract*— Une des tâches de base de la biologie moléculaire est la recherche de similarités dans les banques d'ADN. Une recherche rapide passe par une mise en oeuvre parallèle des algorithmes. Les méthodes séquentielles développées se parallélisent en général sans problème : les noeuds travaillent indépendamment sur une partie des banques et le résultat final est la fusion de l'ensemble. Un des inconvénients réside dans le parcours systématique des banques puisque leur taille influe directement sur le temps d'exécution. Or les banques d'ADN croissent exponentiellement. Cet article propose une autre méthode, basée sur l'indexation, et qui minimise cet écueil. Les banques d'ADN sont structurées de manière à pointer directement sur l'information recherchée. Les premiers résultats effectués sur la plateforme GRID 5000 montrent un potentiel intéressant par rapport au programme de référence dans le domaine : le programme BLAST.**

I. Introduction

Les progrès récents des biotechnologies conduisent à un accroissement exponentiel des données génomiques. Depuis quelques années, de nombreux génomes - dont le génome humain - ont été entièrement séquencés. En octobre 2005, la base de données GOLD (Genome Online database, http://www.genomesonline.org/) compte 298 organismes dont le génome est complètement connu et plus de 1200 génomes en cours de décryptage. A ce flux d'information vient s'ajouter tous les séquençages réalisés ponctuellement par les laboratoires de recherche sur une partie des génomes d'organismes extrêmement variés. De manière plus pragmatique, on estime que le volume des données génomiques double tous les 16 mois. Ces données sont stockées dans des banques accessibles à l'ensemble de la communauté scientifique.

Cette masse d'information est quotidiennement consultée par les biologistes. Les centres de bioinformatique, comme le NCBI (*National Center for Biotechnology Information*, http://www.ncbi.nlm.nih.gov/) par exemple, offrent une panoplie d'outils permettant d'analyser rapidement l'ensemble de ces donnés. Cependant, une des principales tâches est l'interrogation des banques par rapport à leur contenu brut, et non sur les annotations potentiellement disponibles. En d'autres termes, il s'agit de répondre à la question suivante : est ce que les banques contiennent une ou plusieurs séquences d'ADN qui ressemblent à celle que je suis en train d'étudier ?

Cette forme d'interrogation est essentiellement motivée par le dogme central de la biologie moléculaire. Un gène est une petite partie du génome qui code pour une protéine. Les protéines sont constituées d'une chaîne unique d'acides aminés qui représente une traduction directe, via le code génétique, d'une séquence de nucléotides. Il y a donc une correspondance immédiate entre la séquence d'ADN (le gène) et le produit final (la protéine). Mais ce qui intéresse avant tout le biologiste, c'est de connaître la fonctionnalité des protéines, et par extension, le gène qui conduit à l'expression de cette protéine. Cette fonctionnalité est portée principalement par la structure tridimensionnelle de la protéine. Et la structure est en partie liée à l'enchaînement des acides aminés les uns avec les autres. Ainsi, si on est capable de trouver une ressemblance textuelle entre un gène connu et un gène inconnu, alors leur translation en chaînes d'acides aminés peut transmettre cette ressemblance qui s'exprimera par une structure 3D proche, voire une fonctionnalité identique. Il existe donc un lien très fort entre similarité textuelle et fonctionnelle. Cette hypothèse n'est pas toujours vérifiée. Mais elle se révèle suffisamment forte pour que cette piste soit systématiquement prise en considération chaque fois que l'on est en présence d'une séquence d'ADN potentiellement porteuse d'un gène dont on veut découvrir la fonction.

Statistiquement, l'interrogation des banques génomiques par une recherche basée sur le contenu représente plus de 70% des calculs des serveurs bioinformatiques. Ces derniers doivent donc faire face à l'accroissement extrêmement fort des données. Une des solutions consiste à paralléliser les traitements sur des supercalculateurs ou des fermes de PC. A titre d'exemple, le NCBI, qui gère plus de 100 000 requêtes de ce style chaque jour, parallélise ce service sur un cluster de 280 processeurs [1]. Toutes les améliorations, qu'elles soient algorithmiques ou matérielles, sont donc les bienvenues pour augmenter les performances liées à ces traitements.

Ce papier présente une méthode pour réduire les temps d'exécution d'une recherche par le contenu dans les banques d'ADN. Contrairement aux autres approches, la méthode évite un parcours systématique des banques. Elle repose sur une indexation judicieuse des données permettant de pointer immédiatement sur les zones d'intérêt. Cependant, le prix à payer est un stockage des données beaucoup plus volumineux. Ce handicap est toutefois relatif dans la mesure où la capacité des disques reste très importante par rapport au volume actuel des données génomiques. De plus, la mise en oeuvre parallèle

proposée limite, pour un noeud donné, la capacité locale de stockage.

La suite du papier est organisée de la façon suivante : la section 2 rappelle le principe de la recherche par le contenu des données génomiques. La section 3 expose notre méthode d'indexation. La section 4 détaille sa parallélisation. La section 5 montre l'implémentation qui en a été faite sur un cluster de PC et les résultats obtenus. La section 6 conclue ce papier en indiquant les perspectives de ce travail.

II. Recherche par le contenu

La recherche par le contenu se concentre sur les données brutes des génomes et non sur leurs annotations. En d'autres termes, la matière première est simplement un long texte sur un alphabet à 4 caractères (A, C, G et T) qui représente l'enchaînement des nucléotides de la molécule d'ADN. Une requête est une séquence composée des mêmes caractères. Le résultat est l'ensemble des positions dans la banque où on retrouve des similarités avec la séquence requête.

Soit, par exemple, la séquence SB qui représente une banque :

```
SB = attagagaccaggatagagaccaccattagagcc
     acgagatatagagacccaccaagggatataggagac
```

et une séquence requête SR :

```
SR = cttgccaaggctacgacaggcctca
```

Rechercher une similarité entre ces 2 séquences revient à mettre en évidence un alignement local, c'est-à-dire deux sous séquences qui partagent un grand nombre de caractères identiques :

```
SB = ...ccaccattagagccacgagatatagag...
                || || |||| |
SR =     cttgccaag-gctacgacaggcctca
```

Le résultat d'une interrogation est donc un ensemble d'alignements locaux qui peuvent être quantifiés de manière à les ordonner, du plus pertinent vers le moins pertinent. A chaque alignement est affecté un score qui se calcule en fonction de sa taille et du nombre de caractères identiques. Si, par exemple, on donne un score de +1 pour tous caractères identiques et -1 pour tous caractères différents l'alignement précédent à un score de 6.

```
agagccacgaga
|| || |||| |   9 x 1 - 3 x 1
ag-gctacgaca
```

Le premier algorithme mis au point pour trouver de tels alignements locaux est dû à Smith et Waterman [2]. C'est un algorithme de programmation dynamique dont la complexité est quadratique par rapport à la taille des séquences. Son avantage est qu'il donne un résultat exacte : il trouve le meilleur alignement local entre deux séquences relativement au coût d'appariement (caractère identique) et au coût de mésappariement (caractère différent). Son inconvénient majeur réside dans sa complexité, ce qui l'empêche d'être utilisé couramment pour l'interrogation des banques. En effet, dans ce cas, les temps de calcul peuvent se mesurer en heures alors que le délai généralement supporté pour interroger une banque est d'au plus quelques minutes.

Pour remédier à ce problème, une technique très efficace est apparue au début des années 90 et a été mise en oeuvre dans le programme BLAST, un des programmes bioinformatiques le plus utilisé à ce jour [3]. Elle repose sur une heuristique liée aux données génomiques, et plus précisément sur une caractéristique forte présente dans les alignements : on constate que dans la majorité des cas un alignement partage au moins W caractères *consécutifs identiques* entre les deux séquences. A partir de cette constatation, on peut se concentrer uniquement sur ces zones, que l'on appelle points d'ancrage - ou *hit* en anglais -, et ne rechercher des alignements qu'à ces endroits. L'espace de recherche est alors considérablement réduit, ce qui réduit d'autant les temps de calcul. En contrepartie, on n'est pas assuré d'obtenir le meilleur alignement. Mais cette heuristique est suffisamment robuste pour produire des résultats tout à fait satisfaisants.

La mise en oeuvre s'appuie sur l'indexation de la séquence requête. Dans une première étape, on construit un dictionnaire qui contient tous les mots de W caractères de la séquence requête. A chaque mot est associée la liste des positions dans la requête. Dans une seconde étape, la banque est lue séquentiellement, du début jusqu'à la fin, en considérant chaque mot de W caractères. Pour chaque mot, on recherche s'il est présent dans le dictionnaire. Dès qu'un mot existe, alors on est en présence d'un *hit*. En effet, cela signifie que ce mot de W caractères est à la fois présent dans la requête et dans la banque. A partir de ce *hit*, on essaie d'étendre à droite et à gauche pour produire un alignement plus important.

La figure 1 reprend, sur un exemple, le processus de recherche que l'on peut décrire en 4 étapes : (1) un dictionnaire de mots de 3 caractères est construit à partir de la séquence requête (ATGGACTGGC). Il contient 7 mots différents, le mot TGG apparaissant 2 fois en position 2 et 7 ; (2) tous les mots de 3 caractères de la banque sont confrontés au dictionnaire pour déterminer si une occurrence existe dans la requête. Les premiers mots AAT, ATC, TCG et CGG ne sont pas présents dans le dictionnaire et ne donnent pas lieu à un traitement supplémentaire. Le mot GGA, par contre, appartient au dictionnaire ; (3) lorsqu'un point d'ancrage est détecté (le mot GGA) la requête est alignée sur la banque à cette position ; (4) la dernière étape consiste à construire un alignement par extension à droite et à gauche autour du point d'ancrage.

L'avantage de cette méthode est sa rapidité par rapport aux techniques de programmation dynamique. Cette rapidité est d'autant plus marquée que la taille W du point d'ancrage est importante. En effet, plus W est grand, plus la probabilité de trouver un mot de cette taille est faible, et moins les étapes 3 et 4 sont effectuées. En biologie, on fixe en général cette taille à 10 ou 11.

En revanche, un des inconvénients est le parcours systématique de la banque. La taille de la banque d'ADN de référence (GenBank [9]) est aujourd'hui (sept. 2005) de l'ordre de la centaine de Giga nucléotides. En compressant les

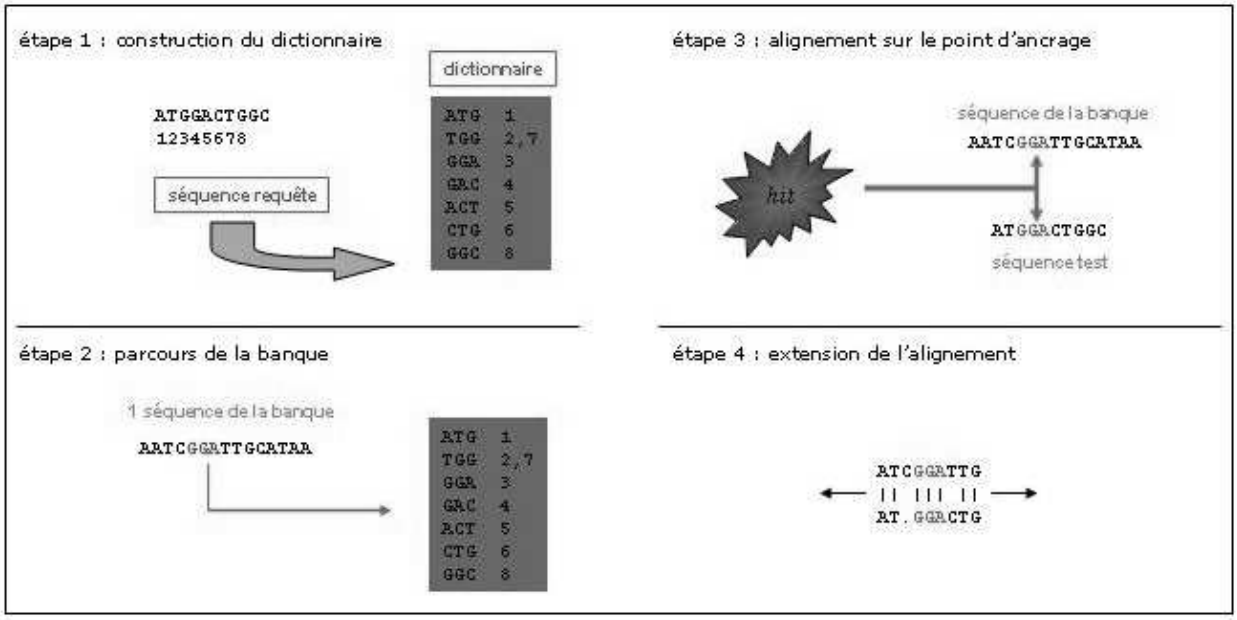

Fig. 1. Les 4 étapes de la recherche d'alignement basée sur l'heuristique des W caractères consécutifs identiques.

données (codage d'un nucléotide sur 2 bits) cela représente 25 Giga octets de données brutes à scanner. Quels que soient les temps de traitement associés à ces données, il est évident que le temps d'exécution minimal est borné par le temps de lecture de la banque.

III. Indexation des banques d' ADN

Pour limiter les temps d'accès, nous proposons non pas d'indexer la séquence requête mais la banque d'ADN. Ainsi, au lieu de parcourir la banque, nous parcourons la requête dont la taille est infiniment plus faible, typiquement quelques milliers de nucléotides (comparé à cent milliards). Le principe de la recherche reste strictement identique : à un mot de taille W de la requête, toutes les occurrences de ce mot dans la banque servent de point d'ancrage. Plus précisément, à tous les mots de taille W de la banque, on associe une liste des positions permettant de positionner la requête à cet endroit.

Dans cette approche, la difficulté vient de la mémorisation du dictionnaire dont la taille dépasse largement les capacités des mémoires centrales des ordinateurs. Indexer 100 Giga octets de nucléotides revient à mémoriser 100 milliards de pointeurs, soit 400 Giga octets s'ils sont représentés sur 32 bits. Il n'y a donc pas vraiment d'autres alternatives que de stocker cette structure de données sur disque. Si ce support ne pose aucun problème en terme de volume de stockage, il est par contre extrêmement lent et très pénalisant dès lors que l'on veut se positionner, dans la banque, aux différents emplacements indiqués par la liste des positions. En effet, se positionner signifie rapatrier une courte séquence en mémoire centrale pour réaliser l'opération d'extension au voisinage du point d'ancrage (étape 4). Cela se traduit par un accès disque dont le coût se mesure en millisecondes.

Par exemple, si on suppose une banque de 100 Giga octets indexée sur des mots de taille 10, cela signifie, qu'en moyenne pour un mot donné, il y a 100 000 occurrences de ce mot dans la banque ($10^{11}/4^{10}$). Si chaque accès coûte 5 ms, il faut plusieurs centaines de seconde pour parcourir la banque, ce qui est évidemment peu efficace, surtout si cette opération doit être répétée pour tous les mots de la séquence requête.

Le schéma d'indexation que nous proposons évite cet écueil : pour chaque mot de taille W nous mémorisons à la fois sa position et son voisinage immédiat, ce qui évite de multiples accès disque. La présence du voisinage immédiat permet de commencer un calcul d'extension et de prendre rapidement une décision sur la possibilité qu'il existe ou non un alignement à cet endroit. C'est en quelque sorte un filtrage qui écarte tous les points d'ancrage qui ne possèdent pas un minimum de similarité imm'ediate avec la séquence requête de part et d'autre de ce point. Dans les faits, la plupart des points d'ancrage sont isolés et cette technique en élimine un nombre extrêmement important.

La figure 2 précise la structure du dictionnaire (désormais appelé index) sur un exemple. La taille des mots indexés est de 2 caractères et le voisinage est de 3 caractères de part et d'autre de ce mot. Ainsi, le mot aa est présent en position 4, 14 et 30 dans la banque et possède à ces positions un voisinage représenté par les couples (agt,gca), (gat,cca) et (cac,gga). Sur la figure 3, seules les positions relatives aux mots aa et ac sont représentés. Il faut bien sur considérer les 16 mots possibles de 2 caractères. Plus généralement, à tous les mots possibles de taille W on associe une liste de triplets (position, voisin-gauche, voisin-droit) ou, pour simplifier (pos, vg, vd). Cette liste de triplets est appelée W-index.

Si la séquence requête contient, par exemple, la sous

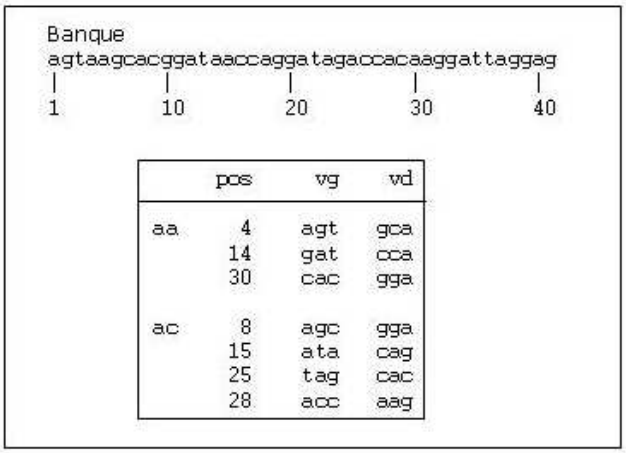

Fig. 2. Structure de l'index. Exemple sur un indexation à 2 caractères avec un voisinage de 3.

séquence gctaacca, le couple de voisins (gct,cca) correspondant au mot central aa sera comparé au W-index référencé par aa. On calculera donc le score des minis alignements suivants :

```
gctaacca        gctaacca        gctaacca
  ||| ||        | ||||||           ||  |
agtaagca        gataacca        cacaagga
```

et on retiendra que seul le second présente suffisamment de similitude pour conduire à un alignement réellement significatif. On ira donc extraire de la banque que l'information relative à la position 14.

Dans ce schéma d'indexation, le temps de calcul total (T_{exec}) relatif à la recherche par le contenu d'une requête peut être scindé en deux parties :

$$T_{exec} = T_{filtrage} + T_{align} \quad (1)$$

$T_{filtrage}$ et T_{align} représentent respectivement les temps de filtrage et d'alignement. $T_{filtrage}$ considère les N mots de W caractères de la séquence requête. Pour chaque mot il faut lire un W-index et pour tous les triplets du W-index calculer le score d'un mini alignement. T_{align} est fonction du nombre (M) de minis alignements dont le score a dépassé un seuil dans l'étape de filtrage précédente. Il faut alors aller chercher les séquences correspondantes dans la banque et calculer un alignement complet.

IV. L'INDEXATION PARALLÈLE

L'indexation parallèle consiste à répartir le traitement précédent sur une machine composée de K noeuds de manière à ce que le temps d'exécution T_{exec} soit le plus court possible. Le modèle retenu est celui représenté par la figure 3.

Dans un premier temps, la requête (de taille N) est divisée en N tâches (exactement N-W+1 tâches). Une tâche de filtrage consiste à lire un W-index et à calculer les minis alignements. L'index est réparti sur chaque noeud à raison de 4W/K W-index par noeud (W est la taille du mot sur lequel est basé l'indexation). En moyenne, un noeud se voit donc affecter N/K tâches. Chaque noeud renvoie une liste de positions qui est fusionnée pour éliminer d'éventuels doublons. Il faut noter que le temps d'exécution d'une tâche de filtrage est directement proportionnel à la taille d'un W-index. A ce stade du traitement, après fusion, on possède M positions dans la banque où on doit calculer un alignement complet. Cette étape est partiellement parallélisée : les noeuds se répartissent les séquences d'ADN et extraient les portions de séquences utiles au calcul des alignements complets. A partir de ces séquences, les alignements sont calculés (séquentiellement) avec le logiciel BLAST. Cette dernière étape présente l'avantage de retourner des résultats qui restent dans un format standard et connu des biologistes.

En réalité, ce schéma d'exécution n'est pas optimal car la charge des noeuds peut être déséquilibrée. En effet, en fonction de la nature de la requête, l'étape de filtrage peut solliciter les noeuds de manière très différente. Tout dépend des mots qui composent cette requête et de la répartition des W-index sur les noeuds. De plus, les tailles des W-index peuvent être extrêmement différentes comme le montre la table ci-dessous calculée à partir d'une banque d'ADN réelle, extraite de GenBank.

	nombre de mots indexés	taille moyenne W-index	taille plus grand W-index	taille plus petit W-index
W=12	16777216	260	124524	0
W=10	1048576	4133	303501	0

Un moyen d'équilibrer les charges est de dupliquer partiellement l'index sur chaque noeud. Ceci implique alors une étape d'ordonnancement préalable pour affecter l'ensemble des tâches de filtrage à tous les noeuds, ainsi qu'une politique d'affectation des W-index aux noeuds. Nous avons choisi de distribuer et de dupliquer arbitrairement les W-index de la manière suivante :

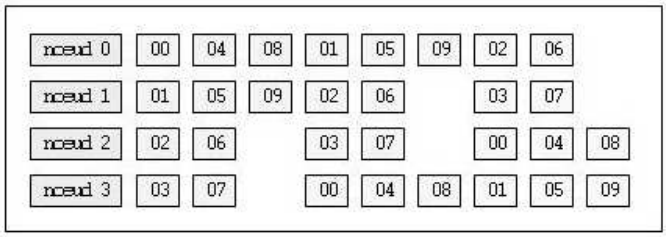

Fig. 4. Exemple de distribution et de duplication de 10 W-index sur 4 noeuds avec un niveau de réplication de 3.

Basée sur cette duplication, l'algorithme d'ordonnancement est simple : il prend séquentiellement toutes les tâches de filtrage issues de la requête ; pour chacune d'elles, il considère l'ensemble des noeuds où le W-index est présent ; puis il

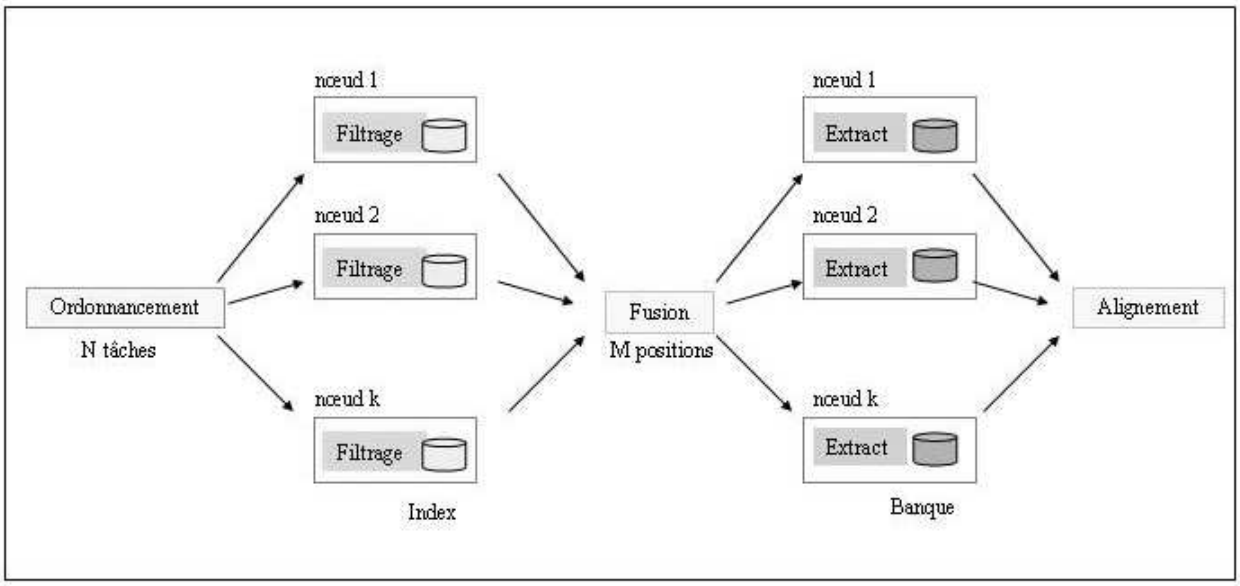

Fig. 3. modèle d'indexation parallèle. Il y a 2 étapes : une étape de filtrage et une étape d'alignement. Les 2 étapes sont parallélisées indépendamment

choisi d'affecter la tâche à celui qui est le moins chargé. Cet ordonnancement est statique : il est effectué au début du calcul et n'est plus remis en question après. Dans ce schéma, le temps de filtrage, est égal au temps que met le noeud le plus lent.

De façon identique, l'étape d'extraction doit faire face au problème de distribution de la banque d'ADN : les M tâches doivent être réparties équitablement pour que chaque noeud ait une charge de travail identique. Comparativement à l'étape précédente, la complexité des tâches est bien moindre : il s'agit juste de sélectionner quelques séquences parmi un grand nombre sans effectuer de traitement particulier. Ainsi, les quelques expérimentations que nous avons fait sur la duplication la banque d'ADN n'apporte quasiment aucun gain significatif sur le temps d'exécution totale car les temps de communication et de synchronisation prennent le pas sur les temps de calcul. Une répartition au hasard des séquences d'ADN sur chacun des noeuds (sans duplication) conduit à des résultats tout à fait satisfaisants et simplifie avantageusement le programme.

V. Expérimentation

Afin de tester notre approche, une expérimentation grandeur nature sur un cluster de PC de 32 noeuds a été effectuée. Le cluster est issu de la plateforme expérimentale GRID5000 [4]. Les noeuds sont des processeurs SUN Fire V20z cadencés à 2.2 GHz avec une mémoire de 2 Goctets. Chaque noeud possède un disque dur local de 73 Goctets (protocole SCSI) et l'ensemble est connecté via un réseau Gigabit Ethernet. Le système d'exploitation est Linux. L'application a été programmée en C avec la librairie MPI.

L'implémentation repose sur un schéma client/serveur. Le serveur reçoit la requête puis se charge de distribuer les tâches aux différents noeuds. Les résultats sont également centralisés sur le serveur. Chaque noeud possède une partie des données (index et banque) sur son disque local.

Le jeu de données est une banque d'EST (*Expressed Sequence Tag*). C'est une banque d'ADN ayant la particularité de contenir de courtes séquences (500 à 1000 nucléotides). Elle est issue de GenBank [9] et contient 16 millions de séquences correspondant à une taille totale de 9×10^9 nucléotides. Elle reflète la réalité biologique quant à la disparité des tailles des W-index. La banque est indexée avec des points d'ancrage de taille 10, et leur position dans la banque est approximée par le numéro de la séquence ou ils se trouvent. Ainsi, lorsque le score d'un mini alignement dépasse le seuil, on renvoie la séquence d'ADN complète (600 nucléotides en moyenne) sur laquelle on recherche ensuite l'alignement.

Le graphique de la figure 5 donne les temps d'exécution mesurés pour l'étape de filtrage en fonction du nombre de noeuds et sans duplication des données. La taille des séquences requête varie de 300 à 3000 nucléotides. On observe que, quelle que soit la taille de la requête, on obtient un facteur d'accélération proportionnel au nombre de noeuds avec, cependant, un fléchissement très net au-delà de 24 noeuds. Ceci est principalement dû à la non duplication des données où probablement plusieurs noeuds se retrouvent avec une charge de travail constante, ce qui borne ainsi les performances de l'ensemble.

Le graphique de la figure 6 évalue l'importance de la duplication des données sur un cluster de 32 noeuds. Au delà d'une réplication de deux, on constate une augmentation modeste des performances. Ainsi, pour cette taille de cluster, une duplication supérieure à deux n'apporte pas réellement de performances supplémentaires.

Nous avons comparé les performances de notre approche (IndexBLAST) avec la version parallèle de BLAST : le logiciel MPI-BLAST [5] (version 1.4.0). Le tableau de la figure 7 résume les temps d'exécution mesurés sur un cluster de 24 processeurs.

Taille requête	Temps filtrage	Nombre de séquences	Temps extraction	Temps fusion	Temps alignement	Temps IndexBLAST	Temps MPI-BLAST
323	0.27	4780	0.90	0.08	0.24	1.82	3.10
616	0.48	8415	0.93	0.13	0.45	2.31	3.08
909	0.58	9625	0.85	0.17	0.51	2.47	3.15
1223	1.01	17904	1.09	0.25	0.95	3.64	3.47
1501	1.03	9327	0.87	0.14	0.51	2.97	3.37
1803	1.08	10376	0.96	0.16	0.61	3.14	3.56
2127	1.28	18648	1.14	0.27	1.05	4.01	3.61
2724	1.31	15083	0.95	0.22	0.89	3.72	3.85
3165	1.65	16727	1.00	0.24	0.94	4.18	4.08

Fig. 7. Comparaison des performances entre IndexBLAST et MpiBLAST. Le temps indiqué dans l'avant dernière colonne (IndexBLAST) est la somme des temps reportés dans les colonnes précédentes (filtrage, extraction, fusion, alignement) plus un overhead qui comprend, entre autre, les temps de communication.

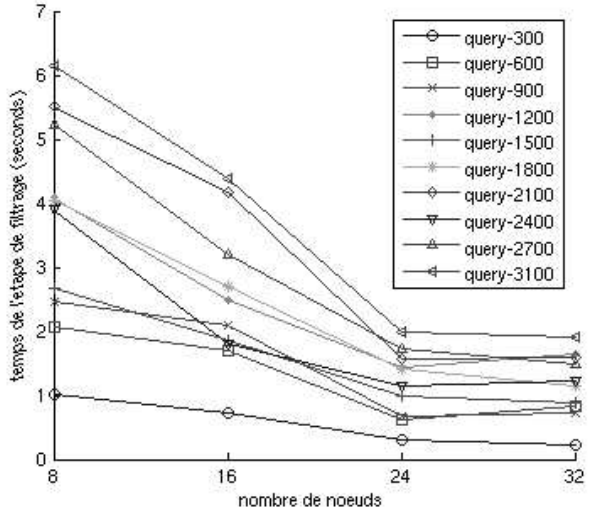

Fig. 5. Temps d'exécution de l'étape de filtrage par rapport à la taille du cluster lorsque l'index n'est pas dupliqué.

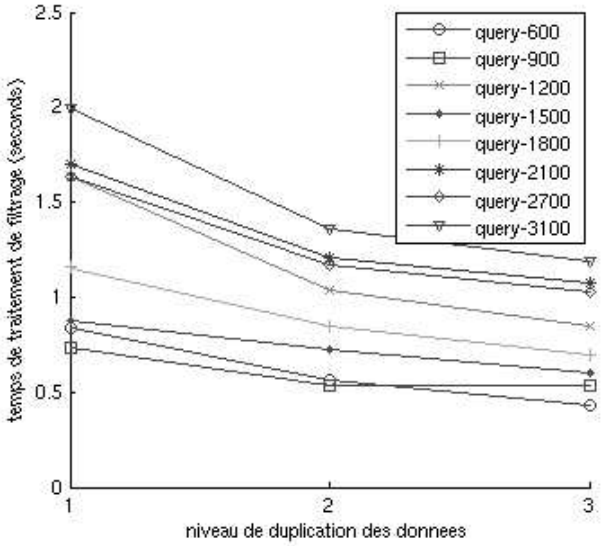

Fig. 6. Temps d'exécution sur un cluster de 32 noeuds en dupliquant l'index 2 et 3 fois.

Les deux dernières colonnes donnent les temps d'exécution sur un cluster de 24 noeuds. Le temps indiqué dans l'avant dernière colonne (IndexBLAST) est la somme des temps reportés dans les colonnes précédentes (filtrage, extraction, fusion, alignement) plus un overhead qui comprend, entre autre, les temps de communication. La 3ème colonne indique le nombre de séquences sélectionnées après filtrage. On remarque que les temps des deux dernières colonnes sont quasiment similaires, ce qui peut sembler, dans une première approche, un résultat décevant. Les mesures des différents temps intermédiaires permettent, cependant, d'être optimiste sur la poursuite de ces travaux en se plaçant dans un contexte plus réaliste où il s'agit de traiter des banques d'ADN de taille beaucoup plus importante. Dans la suite, pour fixer les idées et être en phase avec les données actuelles, nous considérerons une banque 10 fois plus importante (10^{11} nucléotides) par rapport à celle que nous avons utilisée dans notre étude.

Dans cette situation, les temps de MPI-BLAST sont simplement multipliés par 10 car la complexité dépend principalement de la taille de la banque : chaque noeud parcours séquentiellement une partie de la banque dont il a la charge. De la même manière, dans IndexBLAST, les temps de filtrage sont multipliés dans les mêmes proportions, les tailles des W-index étant directement proportionnelles à la taille de la banque. Les temps d'alignement et de fusion sont, quant à eux, proportionnels au nombre de séquences issues de l'étape de filtrage (3ème colonne). Ici, il est plus difficile d'évaluer l'augmentation de ce nombre par rapport à la taille de la banque. Tout dépend, bien sûr, du contenu des banques et de leur variété. A priori, une banque 10 fois plus importante ne signifie pas 10 fois plus d'information du même type. On peut donc espérer un ratio plus faible du nombre de séquences sélectionnées au fur et à mesure de l'augmentation des banques, donc un temps d'alignement et de fusion proportionnellement plus faible. Le temps d'extraction semble constant alors qu'il devrait être fonction du nombre de séquences sélectionnées. En fait, dans l'implémentation, pour éviter des accès disques aléatoires fréquents et coûteux, la banque est mise en mémoire vive.

La majorité du temps est donc consacrée à cette opération, et l'extraction ne représente qu'un très faible pourcentage. Ce temps pourrait être beaucoup plus faible avec une mise en oeuvre plus élaborée : la banque pourrait être maintenue en mémoire et accessible via un mini serveur de requête. Le tableau suivant résume et extrapole les données sur une banque de 100 Giga nucléotides :

taille requête	temps filtrage	temps alignement	temps IndexBLAST	temps MpiBLAST
300	3	1.2	6.6	30
1000	10	3.6	17	33
3000	17	4.8	26	40

Cette extrapolation reste bien sûr à valider. Son but est simplement de montrer que le comportement de notre méthode d'indexation sera d'autant plus intéressante que le volume de données sera important, ce qui sera le cas dans les années futures.

VI. Conclusion et perspectives

Nous avons présenté une méthode d'indexation parallèle pour la recherche de similarités dans les banques d'ADN. Notre objectif est de pouvoir traiter dans des temps raisonnables les masses de données génomiques de demain. Ainsi, plutôt que de parcourir systématiquement les banques nous organisons les données de manière à ne pointer que sur l'information pertinente via un index préalablement construit et stocké sur disque. Les premiers résultats obtenus nous permettent de prévoir des gains significatifs sur les banques d'ADN par rapport au logiciel de référence du domaine : BLAST (ou MPI-BLAST, pour sa version parallèle).

L'implémentation parallèle réalisée dans un premier temps pour valider cette approche peut être largement améliorée. Le schéma proposé (cf. figure 3) comporte des parties séquentielles qui brident les performances de l'ensemble, notamment l'étape finale d'alignement. La principale raison de cette organisation était de produire rapidement (et facilement) des résultats au travers d'un format unanimement exploité par la communauté biologiste. Mais cette tâche peut être avantageusement fusionnée avec l'extraction : une séquence extraite de la banque peut aussitôt être traitée pour localiser les alignements. Ainsi, la parallélisation de cette dernière étape fera décroître de manière significative le temps d'exécution total.

L'analyse des résultats montre également que dans notre approche la majorité du temps est passée dans l'étape de filtrage. C'est donc une partie essentielle sur laquelle nos efforts doivent porter. Pour l'instant, l'indexation qui a été faite reste très primaire. Elle est basée sur l'heuristique de BLAST qui consiste à trouver des points d'ancrage formés de mots de W caractères consécutifs. Or, depuis quelques années, plusieurs équipes de recherche ont mis en évidence des motifs plus subtils (appelés graines espacées [10] [11]) qui permettent soit une sensibilité accrue, soit une indexation moins gourmande en terme d'espace mémoire. On peut ainsi créer des index plus petits, et donc passer moins de temps à les traiter.

Il existe donc plusieurs leviers sur lesquels nous pouvons agir pour accroître de manière très significative les performances de notre approche. La taille de Genbank a dépassé en août 2005 la barre symbolique des 100 Giga nucléotides. Au rythme de production actuelle, la barre du Tera nucléotides devrait être atteinte avant 5 ans. Il faut donc travailler dès maintenant sur les solutions algorithmiques et architecturales qui traiteront efficacement cette masse de données future.

References

[1] Bealer et al. (2004). A Fault-Tolerant Parallel Scheduler for BLAST, Supercomputing 2004, poster exhibit

[2] T.F. Smith, M.S. Waterman. Identification of common molecular subsequences, J Mol Biol. 1981 Mar 25;147(1):195-7.

[3] S.F. Altschul, W. Gish W, W. Miller, E.W. Myers, D.J. , Lipman, Basic local alignment search tool, J Mol Biol. 1990 Oct 5;215(3):403-10.

[4] GRID 5000, http://www.grid5000.org/

[5] A. Darling, L. Carey, W. Feng, The Design, Implementation, and Evaluation of mpiBLAST, ClusterWorld Conference, 2003

[6] W.J. Kent, BLAT: The BLAST-Like Alignment Too, Genome Research, 12 (4) 2002

[7] H.E. Williams, J. Zobel, Indexing and Retrieval for Genomic Databases, IEEE Transactions on Knowledge and Data Engineering, 14 (1) 2002

[8] G. Cooper, M. Raymer, T. Doom, D. Krane, N. Futamur, Indexing Genomic Databases, Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'04), Taichung, Taiwan, 2004

[9] R. Mehnert, K. Cravedi. Public Collections of DNA and RNA Sequence Reach 100 Gigabases, National Library of Medicine, (301) 496-6308, 2005

[10] B, Brejova, D, Brown, T. Vinar. Optimal spaced seeds for homologous coding regions. Journal of bioinformatics and computational biology, 1(4):595-610, 2004

[11] D. Brown. Optimizing multiple seeds for protein homology search. IEEE Transaction on Computational Biology and Bioinformatics (IEEE TCBB), 2(1):29-38, 2005.