UNIVERSITÉ DU QUÉBEC

PRÉDICTION DE GÈNES PARALLÉLISÉE DE HAUTE PERFORMANCE DANS MATLAB

PAR
SYLVAIN ROBERT RIVARD

MÉMOIRE

PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI
COMME EXIGENCE PARTIELLE DE LA MAÎTRISE EN INGÉNIERIE
AU DÉPARTEMENT DES SCIENCES APPLIQUÉES
MODULE DE GÉNIE ÉLECTRIQUE

JUIN, DEUX MILLE SEIZE

Université du Québec à Chicoutimi
Département des Sciences Appliquées
Module de génie électrique

Ce mémoire de maîtrise intitulé :

PRÉDICTION DE GÈNES PARALLÉLISÉE DE HAUTE PERFORMANCE DANS MATLAB

Présenté par :
Sylvain Robert Rivard

A été évaluée par un jury composé des personnes suivantes :

Dr Hung Tien Bui ing., Ph.D.
Université du Québec à Chicoutimi
Directeur de recherche

Dr Martin Otis ing., Ph.D.
Université du Québec à Chicoutimi
Membre du jury

Dr Giulio Antoniol ing., Ph.D.
École Polytechnique de Montréal
Examinateur externe

Mémoire de maîtrise en ingénierie accepté le : 25 août 2016

Sommaire

Ce travail s'inscrit dans un cadre de recherche global du génome humain. Il s'intéresse particulièrement à l'identification de milliers de gènes qui demeurent toujours inconnus à ce jour. Afin de pouvoir effectuer cette tâche sur une plateforme informatique, les séquences d'acide désoxyribonucléique (ADN) seront traitées comme étant des signaux pour permettre l'usage des techniques en traitement numérique de signaux (TNS). Cette approche permettra de réduire les coûts et surtout, le temps que prennent les chercheurs à trouver un gène impliqué dans une maladie.

Le projet est divisé en deux volets. Le premier volet de cette recherche consiste à réduire de façon importante les temps de calcul de certains algorithmes en bio-informatique. Cette recherche propose une méthode de mise en œuvre des algorithmes de prédiction de gènes en parallèle avec le logiciel MATLAB. Les approches proposées sont basées soit sur l'algorithme de Goertzel ou de FFT en utilisant diverses procédures de parallélisme sur une unité centrale de traitement (CPU) et à une unité de processeur graphique (GPU). Les résultats montrent que l'utilisation d'une approche simple, c'est-à-dire sans modification de l'implémentation dans MATLAB, peut nécessiter plus de 4 h et demie pour le traitement de 15 millions de paires de bases (pb) alors qu'une implémentation optimisée peut effectuer la même tâche en moins d'une minute. Nous avons obtenu les meilleurs résultats avec l'implémentation sur GPU qui a pu compléter l'analyse en 57 s, ce qui est plus de 270 fois plus rapide qu'une approche conventionnelle.

Ce premier volet de recherche propose deux stratégies pour accélérer le traitement des données du génome humain et s'appuie sur les différents mécanismes de parallélisation. Lorsque l'implantation se fait avec un CPU, les résultats indiquent qu'il serait préférable d'utiliser une fonction de bas niveau (MEX) afin d'augmenter la vitesse d'exécution. De plus, l'usage des boucles parallèles (PARFOR) doit être effectué dans un ordre précis pour bénéficier au maximum du parallélisme dans l'implantation de Goertzel. Lorsque l'implantation est exécutée sur le GPU, les données doivent être segmentées en plus petits blocs afin d'optimiser les temps de traitement. En effet, les blocs qui sont trop gros risquaient de dépasser la taille de la mémoire tandis que des blocs trop petits ne permettaient pas à l'usager de bénéficier pleinement du parallélisme.

Dans le second volet, nous avons poursuivi avec l'implantation d'un second algorithme qui permet de cibler les régions susceptibles à la présence de gènes. Cet algorithme se base sur les hexamères qui sont de courtes séquences d'ADN composées de 6 nucléotides. De toutes les variations d'hexamères possibles (4096), seulement 40 de celles-ci se

retrouvent plus souvent dans les régions codantes que non codantes. Les autres hexamères se retrouvent autant dans les introns que dans les exons. Il est donc possible de survoler les séquences d'ADN et, selon la présence ou l'absence de certains hexamères, de prédire quelles régions sont codantes.

Lors de la superposition des deux approches, soit l'analyse en fréquence et l'analyse des hexamères, il est possible de mieux cibler les zones où l'on peut retrouver de nouveaux gènes. Les analyses effectuées avec les différents algorithmes présentent des valeurs qui témoignent de la probabilité de retrouver un gène dans une région donnée. Pour compléter le processus de prédiction, il est important de déterminer un critère à partir duquel le programme décide qu'il s'agit réellement d'un gène. Ce critère est basé sur trois paramètres, soient le seuil positif, le seuil négatif et taille de la fenêtre. Afin de déterminer les valeurs optimales, les trois paramètres ont été balayés et la meilleure combinaison a été identifiée.

L'approche proposée dans ce mémoire permet d'analyser de grandes séquences d'ADN en peu de temps afin d'identifier des zones susceptibles de coder un gène. Ce processus est important puisqu'on estime qu'il reste encore quelques milliers de gènes inconnus qui peuvent être responsables de plusieurs maladies génétiques. Nous espérons que ce travail contribuera à la découverte de nouveaux gènes pour ainsi mieux diagnostiquer certaines maladies génétiques.

Table des matières

Liste des tableaux

Liste des figures

Liste des équations

Liste des sigles et abréviations

A	adénine
AA	acide aminé
ADN	acide désoxyribonucléique
ADNc	ADN complémentaire
ANN	réseau de neurones artificiels (artificial neural network)
ARN	acide ribonucléique
ARNm	acide ribonucléique messager
BIST	(Built-IN Self-Test)
BLAST	Basic Local Alignement Search Tool
BOST	(Built-Off Self-Test)
pb	paire de bases
C	cytosine
cM	centimorgan
CPU	unité centrale de traitement
dbEST	data base Expressed Sequence Tag
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
G	guanine
GPU	unité de traitement graphique
HMM	Hidden Markov Model
Kb	mille paires de bases
Mb	million de paires de bases
NCBI	National Center for Biotechnology Information
OGM	Organismes Génétiquement Modifier
s	seconde
T	thymine
TFD	transformée de Fourier discrète
TNS	traitement numérique des signaux

Remerciements

Je désire remercier tous ceux qui m'ont encouragé durant la réalisation de mon mémoire de maîtrise.

En tout premier lieu, je tiens à remercier le professeur Hung Tien Bui, mon directeur de mémoire de maîtrise, pour m'avoir permis de faire des études supérieures dans un domaine de pointe qui n'en est qu'au début.. Je le remercie de m'avoir accordé sa confiance et de m'avoir promulgué de nombreux conseils tout au long de cette recherche.

Je tiens également à remercier le Dr Martin Otis de l'UQAC et le Dr Rachid Beguenane du Royal Military College of Canada à Kingston, Ontario, pour le dynamisme et l'amour pour leur travail qui fut pour moi une source de motivation.

Un merci spécial à monsieur Jean-Gabriel Mailloux, étudiant au doctorat, qui m'a promulgué de nombreux conseils en programmation. Nous avons développé une grande complicité et une amitié qui sont d'une valeur inestimable. Ce fut un grand plaisir de travailler avec lui. Merci pour la grande patience qu'il m'a démontré ainsi que sa disponibilité et sa bonne humeur au travail. Un merci également à tous mes collègues de laboratoire.

Merci au professeur émérite le Dr Jean-François Moreau, un ami fidèle qui m'a encouragé dès le début de notre rencontre à poursuivre mes études en ingénierie. Il n'a jamais douté de mes capacités à innover en science. Peu de personnes auraient pu tendre l'oreille comme il l'a fait et supporter patiemment les hauts et les bas de l'excitation de mes travaux de recherche.

Finalement, un merci tout particulier à Diane avec qui je partage ma vie. Merci de m'avoir accordé toute sa confiance, de n'avoir jamais critiqué mes décisions et d'avoir toujours eu le bon mot d'encouragement dans les moments difficiles. Je lui dois la réalisation d'un autre de mes rêves qui est cette passion de faire de la recherche. Seul toi as pu me comprendre et me donner un souffle nouveau lors de mes études.

Dédicace

Ce travail de recherche est une suite logique de ma préoccupation sociale que tous les êtres humains ont le droit d'être heureux et de vivre en santé. La première partie de ma carrière en génétique moléculaire m'a permis de mettre au point des stratégies permettant de développer des cultures agronomiques saines, afin que l'ensemble de la population mondiale puisse se nourrir. J'ai également travaillé sur les maladies héréditaires humaines pour que les enfants, les femmes et les hommes puissent vivre en santé.

Ma seconde formation en ingénierie m'a permis d'apporter mon humble contribution à identifier et trouver des solutions pour ceux qui souffrent de maladies héréditaires en développant une approche globale qui servira, je l'espère, à l'identification de plusieurs gènes associés à des maladies néfastes pour la santé des individus.

C'est avec une grande fierté que je dédie cet ouvrage à mes deux petits-enfants, Gabrielle et Olivier à qui je souhaite une longue vie remplie de bonheur et de succès.

Albert Einstein disait :

''La connaissance s'acquiert par l'expérience, tout le reste n'est que de l'information''

Votre grand-papa Sylvain-Robert qui vous aime.

Introduction

La génétique moderne remonte aux travaux de Mendel qui fut le premier à établir les lois de l'hérédité [1] [2] [3]. À l'époque, ses travaux sur le pois cultivé (*Pisum sativum*) passèrent presque inaperçus. La redécouverte des lois de Mendel en 1900 par de Vries a permis à celui-ci de développer la théorie de la mutation [4] [5] [6]. Par la suite, ce sont les travaux de T. H. Morgan qui permirent d'établir la théorie chromosomique de l'hérédité c'est-à-dire que les gènes sont organisés en série linéaire le long des chromosomes. Ses travaux de recherche ont été réalisés en utilisant la mouche à vinaigre, Drosophila *mélanogaster* [7]. L'ensemble de son œuvre fut publié en 1915 [8]. Par la suite, plusieurs chercheurs participèrent à la démonstration que l'ADN était la substance de l'hérédité [9]. C'est en 1953 que la structure de l'ADN a été décrite par Watson et Crick [10]. Dans les années 1960, de nombreux mécanismes de base impliqués dans l'expression des gènes furent expliqués. Du milieu jusqu'à la fin du 20^{e} siècle nous assistons à une explosion dans l'avancement des connaissances en biologie moléculaire.

De ces découvertes fondamentales en biologie découle la mise au point d'une multitude d'outils en biologie moléculaire. L'application de ces découvertes a permis la réalisation de cartes génétiques. Les premières cartes furent appliquée notamment chez les grandes monocultures en agroalimentaire telles que la pomme de terre, la tomate, etc. pour l'identification des gènes de résistance aux maladies[11] [12]. Les chercheurs ont élucidé de nombreux mécanismes de l'expression des gènes, le développement de techniques permettant la manipulation de ces gènes ainsi que de nombreuses applications médicales et en agriculture tels que les OGM (organismes génétiquement modifiés).

Les bactéries ont été les premiers organismes à être génétiquement modifiées [13][14]. Une des grandes avancées à la fin du siècle dernier a été le développement des méthodes pour le séquençage de l'ADN. Ces techniques permettent de déterminer l'ordre des nucléotides (A, T, G, C) c'est-à-dire la séquence exacte d'un fragment d'ADN.

Le projet du génome humain 'The Human Genome Project' (HGP) débutât dans les années 1990, et consistait à déterminer la séquence complète du code génétique (ADN) chez l'homme. La première carte génétique de l'homme a été publiée par Schuler *et coll.* en 1996. Plus de 2000 scientifiques, et au-delà de 20 instituts provenant de six pays collaboraient à la première ébauche du génome humain [15] [16]. L'achèvement du projet de séquençage du génome humain a bénéficié de l'utilisation des installations de séquençage ainsi que de ses ressources considérable afin d'effectuer des projets encore plus ambitieux [17]. Un exemple d'un tel projet a pour objectif de séquencer la totalité

des microbes du corps humain permettant une vue de l'homme comme étant un super-organisme [17] [18].

1 Structure du génome humain

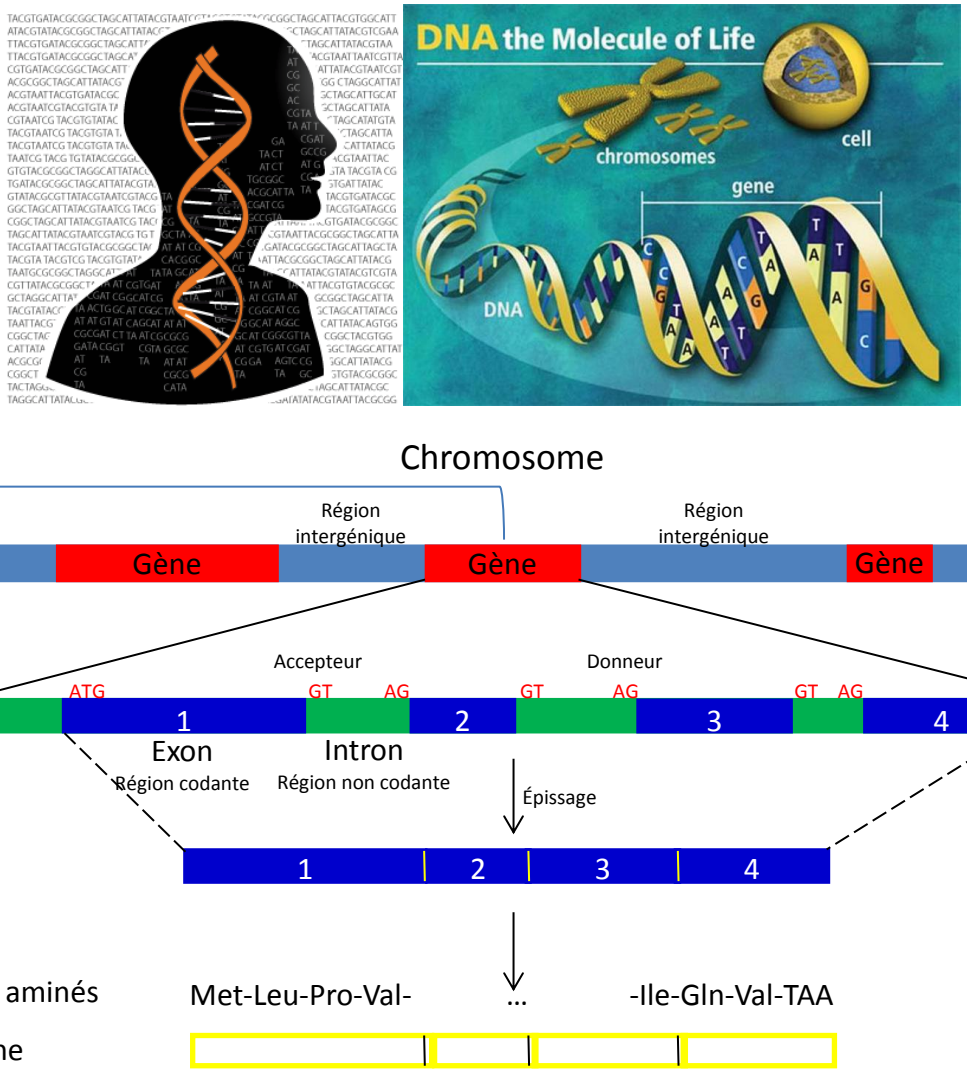


Figure 1: Structure du génome humain.

Chez les eucaryotes[1], dont les humains font partie, l'ADN se trouve dans le noyau des cellules (figure 1). L'ADN est composé de 23 paires de chromosomes. Ces chromosomes sont constitués de gènes et de régions intergéniques. Les gènes des eucaryotes sont composés d'exons et d'introns. Les exons contiennent les informations nécessaires à la synthèse des protéines. Dans les régions codantes (exons), chaque groupe de trois bases (aussi connu sous le nom ***codon***) forme un acide aminé (AA) et ce sont ces AA qui forment les protéines (voir annexe 1).

Tel que mentionné, les AA sont composés d'une série de trois bases, permettant au code génétique d'avoir $4^3 = 64$ variations différentes. Malgré cela, il n'y a que 20 sortes d'AA. On dit donc que le code est dégénéré, puisqu'il y a plusieurs séquences qui encodent pour le même acide aminé (voir tableau 1). De plus, dans l'annexe 1, nous pouvons voir les différentes notations pour le code génétique.

Tableau 1: Code génétique.

Deuxième lettre

Première lettre	T	C	A	G	Troisième lettre
T	TTT Phe TTC Phe TTA Leu UUG Leu	TCT Ser TCC Ser TCA Ser TCG Ser	TAT Tyr TAC Tyr TAA Stop TAG Stop	TGT Cys TGC Cys TGA Stop TGG Trp	T C A G
C	CTT Leu CTC Leu CTA Leu CTG Leu	CCT Pro CCC Pro CCA Pro CCG Pro	CAT His CAC His CAA Gln CAG Gln	CGT Arg CGC Arg CGA Arg CGG Arg	T C A G
A	ATT Ile ATC Ile ATA Ile ATG Met	ACT Thr ACC Thr ACA Thr ACG Thr	AAT Asn AAC Asn AAA Lys AAG Lys	AGT Ser AGC Ser AGA Arg AGG Arg	T C A G
G	GTT Val GTC Val GTA Val GTG Val	GCT Ala GCC Ala GCA Ala GCG Ala	GAT Asp GAC Asp GAA Glu GAG Glu	GGT Gly GGC Gly GGA Gly GGG Gly	T C A G

[1] Les eucaryotes se caractérisent par la présence d'un noyau qui est délimité par une double membrane appelée enveloppe nucléaire. Dans le noyau il y a les chromosomes et la grande majorité des gènes possèdent des introns (région non codante).

1.1 Problématique

Le génome humain complet a été séquencé il y a une douzaine d'années. Cela signifie que l'ordre des trois milliards de nucléotides qui constituent le code génétique est connu. Il reste cependant à identifier tous les fragments qui codent pour des protéines. Le regroupement de ces sections courtes forme ce qu'on appelle un gène.

Les chercheurs du Genoscope ont été parmi les premiers à suggérer, en 2000, un nombre total de gènes humains de l'ordre de 30000 [19], soit une valeur bien inférieure aux estimations (centaine de milliers) qui avaient cours à cette époque [20] [21]. Aujourd'hui, nous estimons avoir trouvé près de 20000 gènes humains. Certains suggèrent qu'il n'y a approximativement que 5000 gènes qui ont un rôle dans les maladies héréditaires. Puisque nous en connaissons déjà près de 1500, il en resterait environ 3500 à découvrir [22].

1.2 Objectif

Ce projet de recherche s'intéresse à la prédiction de gènes de façon informatisée en utilisant des ordinateurs conventionnels. Il existe déjà plusieurs programmes qui peuvent identifier des gènes, mais ceux-ci sont souvent limités par des contraintes telles que la vitesse d'exécution, la taille des séquences à analyser, la précision de l'emplacement du gène et la difficulté de détection des petits gènes (moins de 100 pb).

L'objectif de cette recherche est donc de développer des outils de bio-informatiques afin d'identifier des gènes impliqués dans l'apparition des maladies héréditaires mono géniques. Ces outils seront développés pour être plus simple, plus rapide et conçus pour opérer sur des ordinateurs de bureau. Les groupes de recherches dans le domaine de la génétique pourront ainsi avoir accès aux programmes et les utiliser avec un minimum de formation.

Afin de respecter les contraintes énumérées précédemment, nous devons tenir compte des caractéristiques suivantes :

- Facile à installer localement;
- capacité à former et à tester les programmes de façon indépendante;
- la disponibilité du code source;
- les vitesses de traitements rapides;
- la liberté de restrictions de licences.

2 Revue de la littérature

2.1 Approche classique

Les stratégies en biologie moléculaire pour l'identification d'un gène sont nombreuses, complexes, longues et très coûteuses. Cette section décrit les grandes étapes d'une de ces stratégies pour l'identification d'un gène impliqué dans une maladie héréditaire mono génique. La figure 2 résume les principales étapes de ce processus. Pour commencer, une des étapes importantes, est l'identification des personnes affectées par la maladie étudiée et recenser le maximum d'individus qui ont des liens de parenté avec les personnes recrutées pour le projet. Les personnes concernées seront rencontrées pour obtenir un maximum d'informations.

L'étape subséquente implique une prise de données biologiques telle qu'une biopsie ou des analyses sanguines afin d'établir le bon diagnostic. L'extraction de l'ADN génomique des participants se fait à partir d'une prise de sang (5-10 ml). Ce sang est composé de deux groupes principaux de cellules : les globules rouges qui sont les plus nombreux (plus de 98%) et les globules blancs (moins de 2 %). Les globules rouges n'ont pas de noyau donc ne possèdent pas d'ADN. Seuls les globules blancs possèdent un noyau contenant de l'ADN. C'est à partir de ces cellules que nous obtiendrons l'ADN génomique des participants [23]. Le temps pour effectuer le travail dépendra du nombre d'échantillons prélevé, soit d'environ 1- 2 mois.

Il existe des régions dans l'ADN humain qui sont facilement identifiables et qui sont situés à des endroits bien connus. Ces fragments appelés marqueurs, peuvent être, par exemple, des gènes déjà connus. La banque de données du 'National Center for Biotechnology Information' (NCBI) nous permet de localiser ces régions chromosomiques qui ont déjà été identifiées.

Il est connu que les structures génétiques proches l'une de l'autre sur un brin d'ADN ont tendance à être transmis ensemble de façon héréditaire. Par conséquent, les marqueurs génétiques qui sont situés près du gène responsable de la maladie seront souvent transmis lors de la transmission de la maladie. La stratégie est donc de suivre les marqueurs et de voir quels marqueurs sont le plus souvent (voire toujours) présents chez une personne malade.

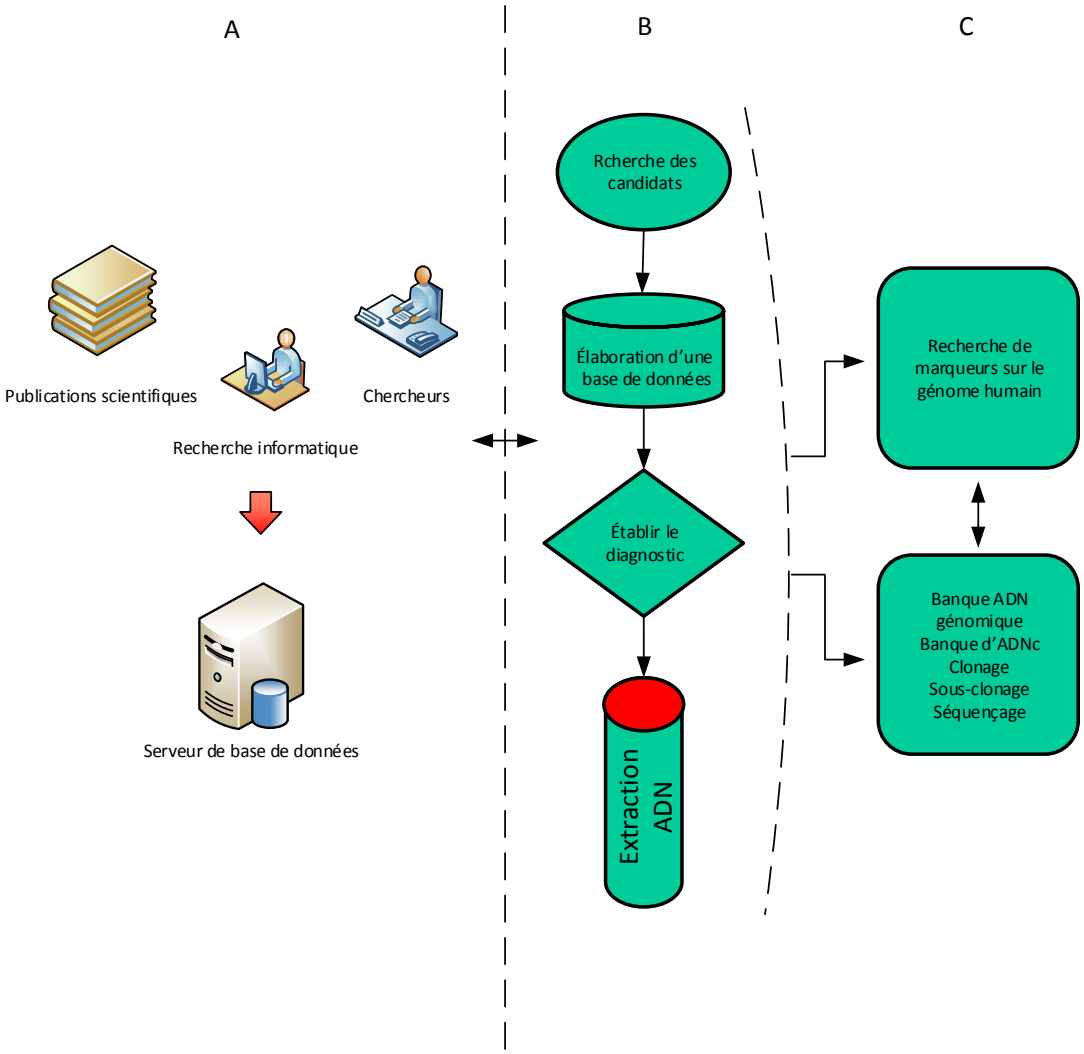


Figure 2: Approche classique. A) Information, B) Candidats et matériel biologique, C) Criblage du génome.

Grâce à la banque de données du NCBI il est possible de sélectionner de nombreux marqueurs moléculaires[2] couvrant l'ensemble du génome humain afin d'identifier lesquelles sont associées à la maladie. Ceci est possible en identifiant les individus atteints et les individus sains d'une même famille et par la suite procéder au criblage du génome. Ce travail est très fastidieux et très couteux. Lorsque la région recherchée est

[2] Un marqueur moléculaire est un marqueur génétique composé de fragment d'ADN qui sert de repère sur les chromosomes.

identifiée, il faut procéder par sous-clonage[3] de cette région chromosomique, l'hybridation moléculaire[4] des différentes banques ADN et finalement effectuer le séquençage des fragments d'ADN afin d'identifier la mutation responsable de la maladie. Cette section de travail qui mène à la découverte d'un nouveau gène peut prendre une ou deux autres années de travail.

2.2 Prédiction de gènes à partir d'une plateforme informatique

La fin du 20e et le début du 21e siècle furent témoins d'extraordinaires découvertes en sciences fondamentales et plus particulièrement en biologie moléculaire et en informatique. Les dérivés de l'informatique sont devenus des outils quotidiens.

De nos jours, plusieurs centres de recherche séquencent des génomes de façon régulière. Cette masse d'information qui afflue sans cesse justifie un besoin de développer des outils informatiques pour l'analyse de ces séquences. Outre son intérêt dans la communauté scientifique, la génomique et la prédiction de gènes sont d'une importance considérable pour la santé humaine et la médecine. Les gènes, qui ont une importance cruciale dans la fabrication des protéines, font l'objet de nombreuses recherches pour différencier les régions codantes de celles non codantes de l'ADN. Depuis le projet du génome humain, plusieurs chercheurs croient qu'il est possible d'effectuer la prédiction de gènes à partir d'une plateforme informatique.

2.2.1 Approches existantes en bio-informatique pour la détection de gènes

Les approches de détection de gènes peuvent être divisées en deux classes : prédiction par similitude (extrinsèque) et prédiction *ab initio* (intrinsèque). Cette dernière est basée principalement sur la reconnaissance des signaux fonctionnels des gènes ainsi que la structure et l'organisation de ceux-ci.

Les approches extrinsèques permettent d'identifier des gènes en fonction de leur similitude à des gènes connus dans des bases de données existantes (ADN génomique, ADNc, dbEST de l'anglais data base Expressed Sequence Tag ou de protéines). Pour ce faire, nous utilisons des algorithmes d'alignement de séquence, comme l'algorithme Basic Local Alignement Search Tool (BLAST), permettant d'identifier les régions similaires entre deux ou plusieurs séquences d'ADN.

---

[3] Le sous clonage consiste à insérer dans un nouveau vecteur un fragment d'ADN déjà isolé.

[4] L'hybridation moléculaire est une technique qui utilise la propriété d'appariement entre les bases complémentaires des deux brins d'ADN (A-T, G-C). Cette technique permet d'identifier et de localiser un fragment d'ADN plus ou moins grand.

Dans ce travail de recherche nous nous intéressons principalement aux méthodes intrinsèques. C'est la raison pour laquelle les sous-sections suivantes décrivent les méthodes intrinsèques les plus importantes. La section 2.2.1.6 discute exceptionnellement d'une méthode extrinsèque. Celle-ci n'a pas été omise due à son importance dans le domaine de la prédiction des gènes.

2.2.1.1 Méthode avec chaîne de MARKOV

Le modèle de Markov est un modèle statistique qui suppose que les états futurs sont déterminés uniquement par les états présents et passés. Pour la prédiction de gènes, le modèle de Markov suppose que la probabilité d'un nucléotide spécifique se trouvant à une position donnée ne dépend que de la valeur des k nucléotides précédentes. Un tel modèle est défini par les probabilités conditionnelles P (X|k nucléotides précédents), soit X_1 le premier nucléotide d'une séquence d'ADN, de loi de probabilité $\lambda = (\lambda_a, \lambda_c, \lambda_g, \lambda_t,)$ où $\lambda_a = P(X_1 = a)$, $\lambda_c = P(X_1 = c)$, $\lambda_g = P(X_1 = g)$, $\lambda_t = P(X_1 = t)$, avec $\lambda_a + \lambda_c + \lambda_g + \lambda_t = 1$ et $\lambda_a, \lambda_c, \lambda_g, \lambda_t \geq 0$ [24] [25]. Il existe plusieurs variantes du modèle de Markov et celles-ci sont présentées dans le tableau 2.

Tableau 2: Modèles de Markov

Matrice de pondération (PWM)	**Modèle de Markov pour trois positions**	**Modèles de Markov cachés (HMM)**	**Champ aléatoire conditionnel Semi-Markov (SMCRF)**
Modèle d'ordre supérieur (WAM)	**Modèle d'interpolation de Markov (IMM)**	**Modèle de Markov cachés généralisé (GHMM)**	**Modèles de Markov Cachés Évolutifs (EHMM)**

Légende: PMW, Positional weight matrices; WAM, Weight array model; IMM, interpolated Markov model; SMCRF, Semi-Markov conditional random field; EHMM, Evolutionary Hidden Markov Models

Afin de construire un modèle de Markov, un ensemble de séquence d'apprentissage est nécessaire. Ces séquences peuvent provenir de n'importe quelle espèce mais il est préférable d'utiliser celles provenant d'espèces apparentées du point vue évolutif [26].

2.2.1.2 Méthode de la courbe-Z

La méthode de la courbe Z est un outil puissant pour la visualisation et l'analyse de séquences d'ADN. Cette méthode a été appliquée sur plusieurs génomes avec et sans introns [27] tel que celui du génome humain [28], celui de la levure [29] ainsi que celui de

la bactérie *Vibrio cholerae* responsable du choléra chez l'homme [30]. Pour une séquence d'ADN donnée, nous avons une représentation unique de la courbe tridimensionnelle (X, Y, Z) [31] [32]. Chaque composante a une fonction biologique spécifique. La composante X affiche la répartition des bases purine-pyrimidine, la composante Y affiche la distribution des bases de type amino-céto tandis que la composante Z affiche la distribution des bases de types hydrogènes (H) liaison faible–forte le long de la séquence. L'un des avantages de la courbe Z est que l'interprétation des résultats est intuitive. Les courbes Z des génomes peuvent être visualisées sur un ordinateur ou sur une feuille de papier, quelle que soit la longueur du génome.

2.2.1.3 Méthode des réseaux de neurones

Un réseau de neurones artificiels (artificial neural network, ANN) est un modèle de traitement de l'information pour présenter des relations d'entrée-sortie complexes. C'est un modèle de calcul dont la conception est schématiquement inspirée du fonctionnement des neurones biologiques. Les réseaux de neurones sont généralement optimisés par des méthodes d'apprentissage de type probabiliste, en particulier bayésien[5]. Les réseaux de neurones mettent en œuvre le principe d'induction, c'est-à-dire l'apprentissage par l'expérience. Donc, plus on entraine notre système, plus il est efficace jusqu'au point où le modèle sature. Cette approche demande beaucoup d'information dès le début afin d'entrainer notre système de réseau neural. Un problème qui revient est le chevauchement de certain exons qui donne souvent de faux positifs (communément appelés erreurs de type II) [33]. Un des désavantages de cette méthode est que la capacité de traitement limitée. En effet, les réseaux de neurones ont un temps d'exécution élevé et sont souvent incapables de traiter de longues séquences d'ADN [34].

2.2.1.4 Méthode des hexamères

Dans les régions codantes, un regroupement de trois bases spécifie le type d'acide aminé qui sera produit. Certaines combinaisons sont plus propices que d'autres de se retrouver dans un gène. Prenons comme exemple les codons-stops. Si l'un de ceux-ci se retrouvait au centre d'un exon, il y aurait arrêt de la transcription empêchant ainsi la production de la protéine décrite par le gène. Il serait donc impossible de retrouver un codon-stop à l'intérieur d'un exon. De plus, nous savons que certains acides aminés peuvent être composés de plusieurs combinaisons de pb (par exemple, l'alanine peut être codée de 4 façons différentes, voir annexe 1). De ces combinaisons, certaines sont plus communes que d'autres. En analysant la distribution des pb dans les régions codantes et non

[5] L'inférence bayésienne est une méthode d'inférence permettant de déduire la probabilité d'un événement à partir de celles d'autres événements déjà évalués. Elle s'appuie principalement sur le théorème de Bayes.

codantes, il serait possible de prédire si une séquence d'ADN est codante ou pas. L'ensemble de ces informations nous amène à une autre approche qui a été développée pour l'identification des gènes, soit l'analyse par hexamères.

Les hexamères sont de courtes séquences d'ADN composées de 6 nucléotides. Le nombre d'hexamères que l'on peut retrouver avec les 4 types de base (A, T, G, C) est de 4^6. Certaines de ces combinaisons se retrouvent de façon préférentielle dans les régions codantes. Il serait possible d'attribuer une pondération aux hexamères selon qu'ils se trouvent plus souvent dans les régions codantes ou pas. Cette pondération pourra donc être utilisée pour identifier les régions codantes, permettant ainsi de cibler des zones qui auront les caractéristiques recherchées pour l'identification des gènes.

Dans une étude, des chercheurs ont évalué 19 algorithmes différents pour l'identification des gènes. Le nombre de gènes détectés avec les hexamères était excellent avec un taux de succès de 91.5% pour des gènes ayant 192 pb. [35]. Compte tenu de sa bonne performance, la méthode des hexamères est une méthode à privilégier. Il faut dire également que les analyses d'hexamères possèdent 4096 combinaisons différentes, ce qui augmente nos chances d'identifier un gène, mais le temps de calcul est plus long que les FFT.

2.2.1.5 Analyse en fréquences

Il a été observé que les arrangements des nucléotides dans les régions codantes ont une certaine périodicité. Ceci est dû à la périodicité de trois bases qui forment les différents AA que nous retrouvons dans les régions codantes [36]. Cette périodicité peut aussi être observée dans le domaine fréquentiel comme étant une région de forte amplitude située au tiers de la fréquence d'échantillonnage, ce qui correspond aussi au tiers du nombre d'éléments considérés [37]. Selon cette observation Tiwari *et al.* (1997) [38] ont analysé plusieurs séquences d'ADN de différents organismes et proposent que toutes les régions codantes possèdent cette même caractéristique.

Pour détecter cette périodicité, et donc identifier les régions codantes, il est possible d'effectuer une analyse de fréquence en utilisant la transformée de Fourier discrète (DFT). Puisque les calculs de la DFT ne sont pas efficaces, la FFT est plus souvent utilisé [39] [40] [38] [41] [42]. Les résultats de la FFT sont ensuite analysés afin de déterminer s'il y a effectivement une périodicité de trois.

Pour effectuer une analyse par FFT, il faut découper le signal en petites fenêtres. Il a été démontré que la forme et la longueur des fenêtres affectent les résultats de prédiction [38] [43] [44] [45]. La forme de la fenêtre est souvent décrite comme étant un filtre de fenêtrage. Ces filtres permettent d'obtenir de meilleurs résultats [46] [47] [48]. L'autre caractéristique importante est la longueur de la fenêtre L pour les analyses des séquences génomiques. Plusieurs tests avec des longueurs variables de la fenêtre ont été effectués avec des séquences génomiques de levures et de virus. Quelques régions codantes ont été identifiées avec une fenêtre inférieure à L=300 pb. Lorsque celle-ci est inférieure à 250 pb on observe une amplification du bruit de fond et des chevauchements entre les régions codantes contigües surviennent avec une longueur de fenêtre supérieure à L=400 pb [38]. Donc, pour ces raisons, plusieurs groupes de recherche ont proposé des analyses avec une fenêtre L=351 pb. La FFT est effectuée pour une taille de fenêtre donnée et cette fenêtre est déplacée jusqu'à ce que la séquence complète soit évaluée.

Bien qu'il existe plusieurs solutions dans la littérature qui donnent de bons résultats, ils sont souvent limitées à de petites séquences de quelques milliers de pb [41] [49] [50]. L'approche par la FFT, par contre, n'a pas cette limite. Par contre, il a été observé que l'utilisation seule de la méthode de Fourier ne permet pas la détection des sites d'épissage. Une autre limitation de la FFT est qu'elle n'est pas en mesure d'identifier les gènes dépourvus de périodicité de 3 [51].

2.2.1.6 Méthodes GENSCAN et TWINSCAN (extrinsèque)

Le programme GENSCAN est un outil informatique qui utilise une méthode extrinsèque [52] pour l'identification des structures exon/introns d'un gène dans l'ADN génomique. Avec ce programme, il est possible de prédire plusieurs gènes dans une séquence et de prédire des gènes partiels même ceux qui sont présents sur l'un ou l'autre des brins. De plus, Flicek *et coll*. 2003 ont mis au point l'algorithme TWINSCAN [53] permettant de repérer les séquences qui sont semblables à des gènes connus chez une autre espèce. Les séquences semblables entre deux génomes (l'homme et la souris, par exemple) permettent d'identifier des régions homologues pour des séquences connues. Le problème réside dans le fait que l'approche est effectuée par homologie ce qui a pour conséquence qu'une comparaison doit être faite avec une base de données existante. GenomeScan est un autre outil qui utilise la méthode extrinsèque et qui permet d'identifier avec précision les structures exon-intron de gènes [54]. Une lacune du programme GenomeScan est son incapacité à reconnaître les différentes variantes de gènes.

3 Approche proposée

L'approche proposée dans ce mémoire est de procéder en deux étapes. Dans un premier temps, nous proposons utiliser l'approche par analyses en fréquences. La simplicité de cette approche laisse croire qu'elle pourrait être utilisée pour une analyse sommaire rapide. La difficulté avec une telle approche est que la délimitation des zones codantes et non-codantes n'est pas toujours bien définie. Nous proposons donc utiliser une deuxième technique, qui est basée sur d'autres caractéristiques du génome, pour effectuer la même tâche. Cette technique, appelée l'approche par hexamère, permet elle aussi de détecter des régions codantes. Ces informations seront amalgamées pour finalement obtenir des résultats satisfaisants. L'approche proposée est présentée à la figure 3.

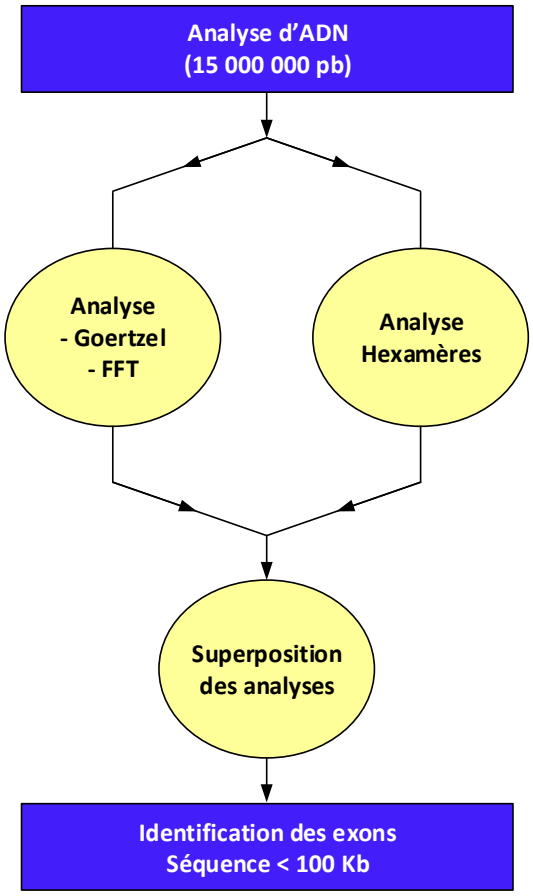


Figure 3: Compilation des données (fréquence et hexamères).

4 Analyse en fréquences pour l'identification d'un gène

4.1 Création des vecteurs

Afin de pouvoir effectuer l'analyse en fréquences sur des séquences d'ADN, ces séquences doivent tout d'abord être converties sous une forme numérique. Plusieurs systèmes de représentation numérique ont été proposés [55] [56] [37] [57] [58]. Le schéma de représentation de Voss [37] est l'un des plus populaires en raison de sa simplicité et des bons résultats qu'il permet d'obtenir. En utilisant cette approche, une séquence d'ADN est transformée en quatre vecteurs binaires correspondant aux quatre bases (A, T, G, C). Dans le vecteur binaire de la base X, la présence de cette base particulière à la position donnée est représentée par un 1 et l'absence de cette base est représentée par un 0 [37]. Le tableau 3 donne un exemple pour illustrer le processus: la séquence ATCTGGA a été décomposée en quatre vecteurs $X_A(n)$, $X_C(n)$, $X_G(n)$ et $X_T(n)$.

Tableau 3: Séquence d'ADN convertie en ses homologues binaires.

Séquence	A	T	C	T	G	G	A
$X_A(n)$	1	0	0	0	0	0	1
$X_C(n)$	0	0	1	0	0	0	0
$X_G(n)$	0	0	0	0	1	1	0
$X_T(n)$	0	1	0	1	0	0	0

4.2 Méthodes d'analyse en fréquences

4.2.1 DFT

Pour faire une analyse en fréquences, il est possible d'utiliser la DFT. Pour analyser le spectre de fréquences d'une séquence d'ADN *f(n)* de longueur N, il faudrait utiliser l'équation suivante :

Équation 1

$$F(k) = \sum_{N=0}^{N-1} f(n) * e^{i\left(\frac{2\pi}{N}\right)nk}$$
$$0 \leq k \leq N-1$$

Dans la formule de la DFT, nous avons la représentation du vecteur en entrée *f(n)* et de l'exponentielle complexe (voir annexe 4). L'équation (1) peut être utilisée pour le calcul d'une DFT pour chacun des quatre vecteurs binaires soit $F_A(k)$, $F_T(k)$, $F_G(k)$ et $F_C(k)$. Pour quantifier une périodicité de 3 dans les exons d'une séquence d'ADN, il faut que le coefficient k corresponde à N/3 (où N est la longueur de la séquence et est choisi pour être un multiple de 3).

La totalité du spectre combine les amplitudes au carré des DFT individuels *i.e.* $F_A(k)$, $F_T(k)$, $F_G(k)$ et $F_C(k)$. Ceci est exprimé dans l'équation (2) :

Équation 2

$$SC[k] = \sum |X_m[k]|^2 , m \in \{A, T, G, C\}$$

4.2.2 FFT

L'algorithme pour effectuer une DFT n'est pas efficace en termes de temps de calcul. Pour effectuer une analyse en fréquence, la DFT est souvent remplacée par la transformation de Fourier rapide (FFT) puisque celle-ci offre de meilleures performances [46].

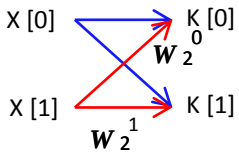
Équation 3

$$F(k) = \sum_{n=0}^{(N/2)-1} x(2n) * W_{N/2}^{nk} + W_N^k \sum_{n=0}^{(N/2)-1} x(2n+1) * W_{N/2}^{nk}$$

$$\boldsymbol{W}\frac{k}{N} = \boldsymbol{e}^{-\boldsymbol{i}(2\boldsymbol{\pi}/\boldsymbol{N})}$$

L'application de la FFT se fait normalement avec une opération papillon, tel qu'illustré à la Figure 4. La figure montre une FFT à 2 points.

Figure 4: Représentation d'une opération papillon.



Exemple de calcul :

$$K[0] = X[0] + X[1]$$
$$K[1] = X[0] - X[1]$$

Si

$$W_n^k = e^{\frac{-2\pi ik}{n}}$$

Alors

$$K[0] = X[0] + X[1] * W_n^k$$
$$K[1] = X[0] - X[1] * W_n^K$$

$$K[0] = X[0] + X[1] * W_2^1$$
$$K[1] = X[0] - X[1] * W_2^1$$

Les opérations en papillon peuvent être combinées afin de calculer des FFT à 4 et à 8 points tel qu'illustré ci-dessous à la figure 5.

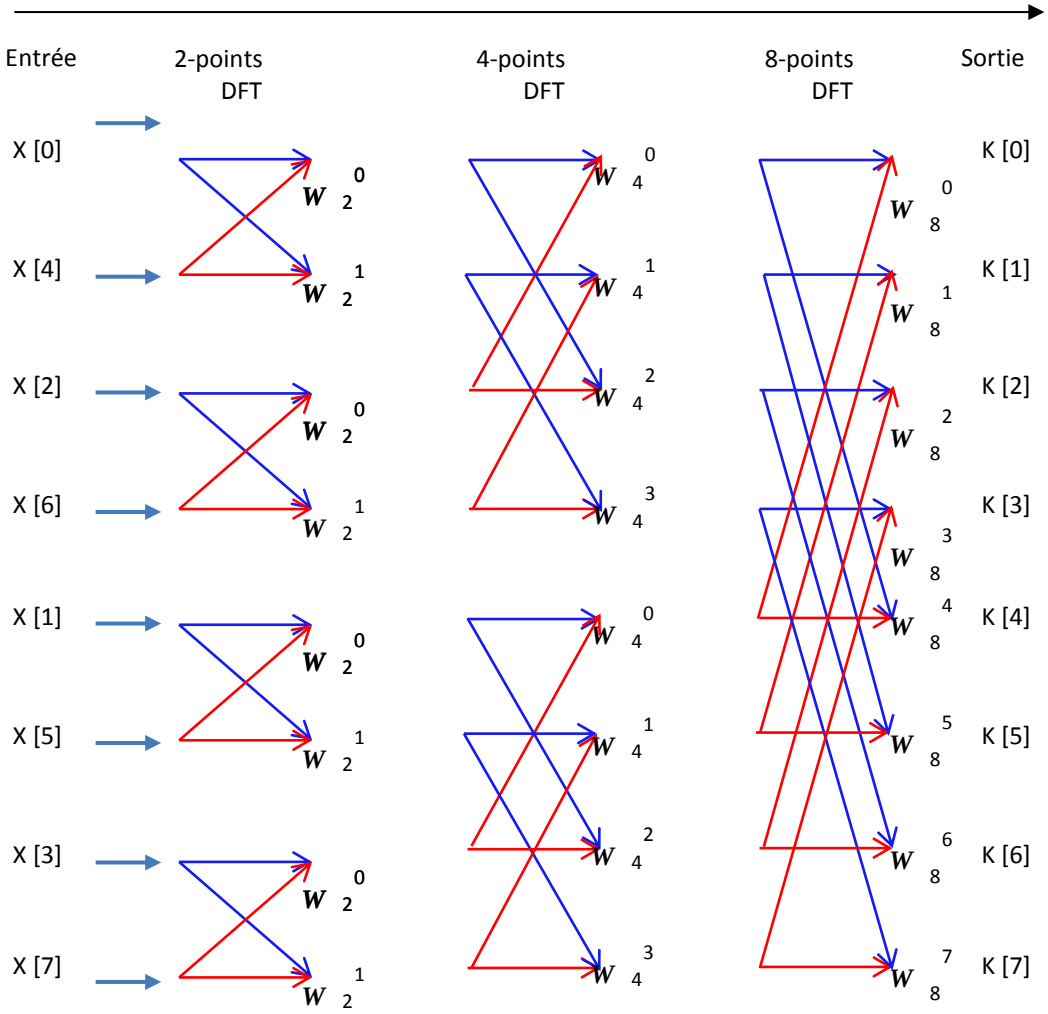


Figure 5: Construction d'un diagramme à 2, 4 et 8 points

4.2.3 Algorithme de Goertzel

En plus de la DFT et de la FFT, il existe d'autres algorithmes qui permettent l'analyse en fréquences. Une de ces technique est l'algorithme de Goertzel [59]. L'algorithme de la DFT et celle de la FFT calculent l'amplitude et la phase d'une grande plage de fréquences, tandis que l'algorithme de Goertzel fait le calcul pour une fréquence spécifique prédéterminée. L'algorithme de Goerzel est implémenté comme un filtre IIR (Réponse Impulsionnelle Infinie) du second ordre. Pour chacune des fréquences, il nécessite seulement le calcul de deux coefficients, un coefficient réel et un coefficient complexe.

De plus, cet algorithme n'a pas besoin d'emmagasiner les N échantillons avant de pouvoir calculer la sortie du filtre, contrairement à la DFT. L'algorithme de Goerzel est un algorithme récursif dont la sortie au temps présent dépend aussi de sa sortie aux temps précédents.

L'application de l'algorithme est faite à partir des équations 4 et 5. La première étape est de calculer une séquence de sortie, y[n], pour un signal d'entrée x[n] :

Équation 4

$$y[n] = x[n] + 2\cos(2\pi f_n)y[n-1] - y[n-2]$$

Après avoir effectué le traitement de N éléments en utilisant l'équation (4), le résultat de l'analyse en fréquences est obtenu avec l'équation suivante, où P_x est la puissance à la fréquence *fs/3* pour la base de type *x :*

Équation 5

$$P_x = y_N^2 + y_{N-1}^2 + y_N * y_{N-1}$$

Les quatre valeurs de P_X, qui correspondent aux quatre bases, sont alors additionnées ensemble pour obtenir la mesure du contenu *spectral.*

La performance des algorithmes de Goertzel est souvent supérieure à celle de la FFT, surtout lorsque le nombre de fréquences à analyser est faible. Par exemple, des chercheurs ont optimisé une nouvelle structure de l'algorithme de Goerzel pour l'analyse des BIST (Built-IN Self-Test) ou BOST (Built-Off Self-Test) et l'ont comparé à la performance avec la FFT. Par rapport à ce dernier, l'implantation avec l'algorithme de Goertzel a permis de réduire par un facteur de 6 le temps d'un essai de dispositifs émetteurs-récepteurs RF [35].

4.3 Fenêtrage

4.3.1 Fenêtre coulissante

Lorsqu'un signal est trop long ou lorsqu'on ne s'intéresse qu'à une section d'un signal il est possible d'utiliser le fenêtrage. Le fenêtrage sert à découper le signal et à le traiter de façon à améliorer les résultats dans le domaine fréquentiel. Dans une analyse typique, une longueur spécifique de base (taille de la fenêtre) est traitée puis la fenêtre est décalée d'une base le long de la séquence et le traitement est répété jusqu'à la fin de la séquence (voir figure 6). Il serait possible d'augmenter le décalage de la fenêtre pour la déplacer de plus d'une base lorsque l'analyse de la fenêtre est terminée. Par exemple, un décalage de deux ou trois bases permettrait d'améliorer le temps de calcul tout en, possiblement, conservant le niveau de précision. Par contre, cette analyse n'a pas été effectuée dans ce mémoire bien qu'elle puisse faire partie de travaux futurs.

Il a été démontré que la forme de la fenêtre et les paramètres de longueur affectent les résultats de prédiction d'un gène [43] [60]. Anastassiou (2000) [44], Kotlar et Lavner (2003) [45] ont utilisé une fenêtre rectangulaire dans l'analyse d'ADN en exécutant une DFT. Gunawan *et coll.* (2007) [61] et Akhtar *et coll.* 2008 [43] ont de leurs côtés utilisé d'autres types de fenêtres tel que Bartlett et Kaiser dans leurs méthodes afin de mieux définir les spectres de fréquence des fenêtres. D'autres fenêtres telles que Gauss, Hamming, Hanning et Blackman pourraient également être utilisées pour obtenir des caractéristiques différentes. Dans le cadre de ce projet, pour limiter le temps de traitement, nous avons sélectionné une fenêtre rectangulaire. N'oublions pas que nous voulons que nos analyses prennent un minimum de temps.

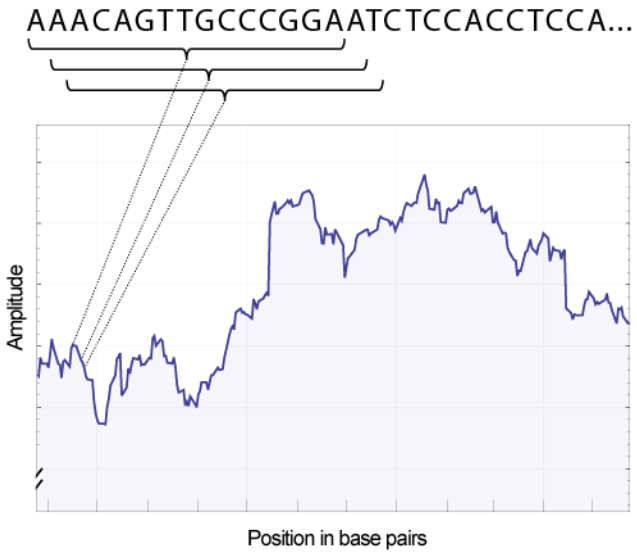


Figure 6: Illustration d'une fenêtre coulissante.

4.3.2 Taille des fenêtres

Une des difficultés lors du fenêtrage est la détermination d'une taille appropriée pour la fenêtre coulissante. Un cadre de petite fenêtre provoque davantage de fluctuation des données, ce qui entraine des erreurs de prédiction, alors qu'un cadre de fenêtre de trop grande taille pourrait nous faire manquer de précision. Plusieurs groupes de chercheurs dont Tiwari *et al.* et Amastassiou ont démontré qu'une fenêtre de 351 était un bon choix [38] [43] [44].

4.4 Mise en œuvre

Pour effectuer l'algorithme de Goertzel, il existe une fonction fournie avec MATLAB. En analysant cette fonction, il est possible de constater qu'elle effectue beaucoup de validations des données avant de faire appel à la fonction GoertzelMEX. En dressant le profil de la fonction Goertzel.m, il a été découvert qu'une longue période de temps est consacrée à l'exécution des commandes de validation au lieu de l'algorithme de Goertzel lui-même. Afin d'améliorer le temps de traitement, il nous a été possible de contourner les commandes de validation et d'exécuter la fonction GoertzelMEX directement. Pour ce faire, les données envoyées à la fonction GoertzelMEX ont besoin d'être formatées correctement puisque le processus de validation n'est plus effectué. Le temps d'exécution est alors considérablement réduit.

MATLAB fournit également une fonction pour calculer la FFT. L'optimisation de la FFT dans MATLAB est extrêmement performante. Dans un algorithme de prédiction de gène typique, l'analyse en fréquence, est effectuée séquentiellement sur une fenêtre de données avant de passer à la suivante. Étant donné que chaque fenêtre est indépendante. Il serait possible de les exécuter en parallèle.

4.4.1 Processeurs multicoeurs

Depuis les dernières années les ordinateurs personnels sont, pour la plupart, équipés d'un processeur avec plusieurs cœurs qui peuvent exécuter plusieurs tâches en parallèle. Afin d'utiliser ces ressources parallèles, MATLAB doit être configuré et l'algorithme doit être adapté. Dans un algorithme typique de prédiction de gènes, l'analyse en fréquence est effectuée de manière séquentielle sur une fenêtre de donnée avant de passer à la fenêtre suivante. Sachant que le résultat de calcul pour une fenêtre est indépendant des résultats des autres fenêtres, il serait possible d'effectuer ces opérations sur plusieurs fenêtres en parallèle. MATLAB permet ce style de parallélisme avec la commande PARFOR. La commande PARFOR est une version parallèle de la boucle FOR qui est couramment utilisée en programmation séquentielle.

4.4.2 GPU

De nombreux ordinateurs modernes sont équipés de cartes graphiques qui contiennent une ou plusieurs unités de traitement graphique (GPU). Chaque GPU contient un grand nombre de processeurs qui peuvent être utilisés pour le traitement parallèle. Il existe plusieurs façons de bénéficier du parallélisme du GPU.

Bien que ces cartes aient historiquement été mises au point pour le traitement vidéo, elles peuvent dorénavant être utilisées pour accélérer les calculs. Afin d'accéder à la puissance de traitement de ces cartes à l'intérieur de MATLAB, plusieurs boîtes à outils sont disponibles.

4.4.2.1 Matlab R2011b

Il existe plusieurs commandes dans MATLAB qui permettent à l'usager d'accéder aux ressources du GPU dont celles de la boîte à outils pour le parallélisme. Malheureusement, cette boîte à outils a plusieurs inconvénients incluant l'absence d'une commande équivalente à PARFOR. Bien que la commande ARRAYFUN offre des fonctionnalités similaires, elle manque de flexibilité lorsqu'une fenêtre coulissante doit être utilisée. Cette boîte à outils MATLAB n'a donc pas été examinée de manière plus poussée.

4.4.2.2 JACKET

JACKET est un logiciel qui permet à un algorithme de MATLAB d'exécuter une partie de leur code sur un GPU. JACKET offre deux méthodes pour accélérer le processus de prédiction de gènes dans MATLAB. La première méthode consiste à utiliser une fonction FFT qui est implémentée sur le GPU. La deuxième méthode, qui peut être utilisée en conjonction avec la première, est de paralléliser les boucles en utilisant la commande GFOR.

4.4.2.3 Gestion des données pour la mise en œuvre sur GPU

Pour utiliser le GPU de façon optimale, la séquence d'ADN choisie doit d'abord être séparée en ***blocs*** d'ADN. Ces blocs d'ADN sont ensuite envoyés en séquence au GPU pour le traitement. Lorsque les blocs d'ADN sont sur le GPU, ils sont divisés à nouveau en plus petits ***fragments*** d'ADN afin d'être traités en parallèle par les GFOR séparément. Ce processus est illustré à la figure 7 de la page suivante.

La difficulté rencontrée dans cette mise en œuvre est de trouver le nombre optimal de GFORs pour traiter une séquence donnée en parallèle. Un grand nombre de GFORs permet d'améliorer le parallélisme, dans un même temps celui-ci consomme aussi les ressources de traitement sur le GPU. Lorsque les ressources sont épuisées, l'algorithme de maintenance de JACKET est appelé à remédier à la situation. Ce processus ralentit l'exécution de l'algorithme et doit être évité lorsque possible. Le but est donc de maximiser le nombre de GFORs sans épuiser les ressources disponibles.

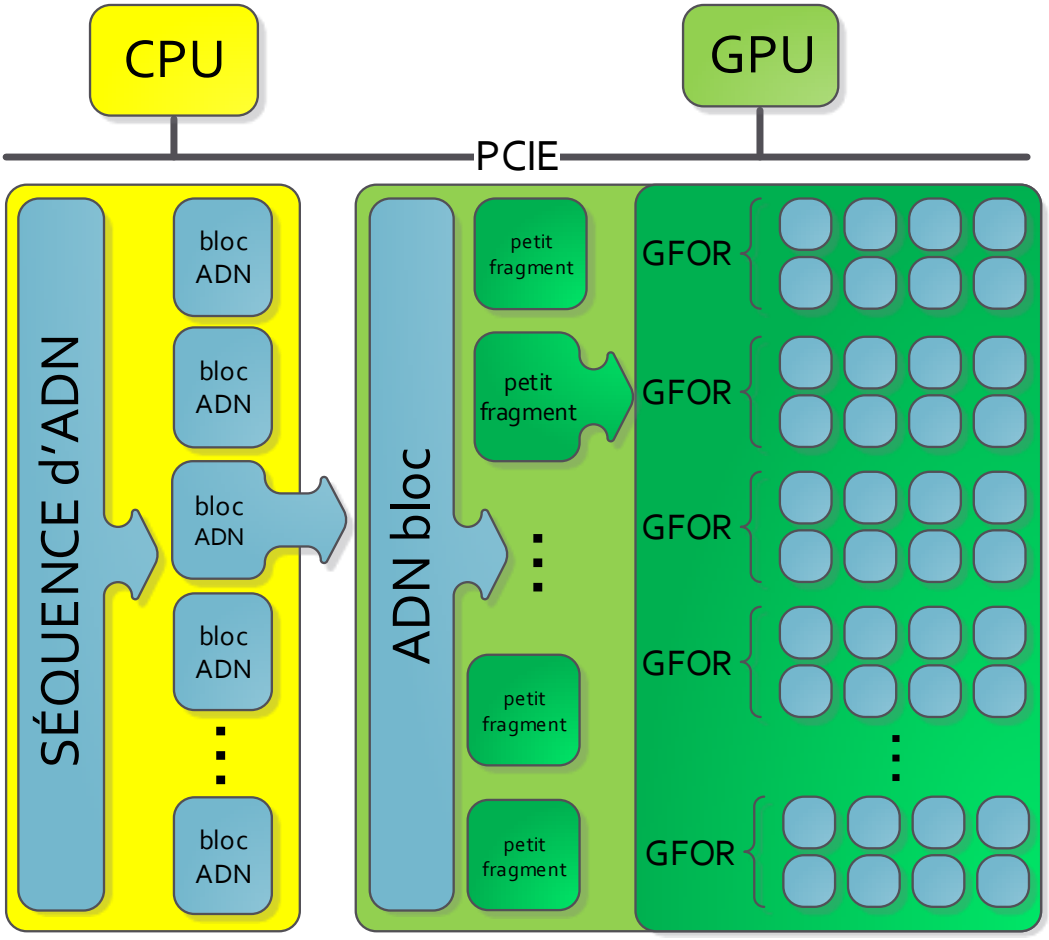


Figure 7: Procédure pour décomposer de grandes séquences d'ADN pour le traitement par GPU.

Une carte de GPU ne dispose que d'un nombre limité de processeurs parallèles (PU_{TOTAL}). Le nombre maximum de boucles GFOR (N_{GFOR1}) est donné par:

Équation 6

$$N_{GFOR1} \leq \frac{PU_{TOTAL}}{4 * (PU_{FFT} + PU_{MISC})}$$

L'équation indique que quatre implémentations GPU de la FFT doivent être présentes au cours de chaque boucle de GFOR pour tenir compte de chacune des matrices de nucléotides. L'implémentation demande PU_{FFT} unités de traitement pour la FFT elle-

même et nécessite des unités supplémentaires de traitement (PU_{MISC}) pour diverses opérations telles que la puissance de deux et l'addition des spectres de puissance.

Comme indiqué précédemment, cette équation doit être respectée de sorte que le nombre d'unités de traitement ne dépasse pas les ressources disponibles. Si il y a dépassement le logiciel réaffectera les ressources du GPU ce qui peut ralentir l'exécution de l'algorithme.

En plus de la limitation de l'unité de traitement, la mise en œuvre doit aussi tenir compte des contraintes de mémoire. Chaque itération nécessite une certaine quantité de mémoire GPU donc, la quantité de mémoire GPU disponible limite également le nombre maximum de GFORs. Cette contrainte peut être résumée comme suit:

Équation 7

$$N_{GFOR2} \leq \frac{MEM_{TOTAL}}{4 * (MEM_{FFT} + MEM_{MISC}) + MEM_{SETUP}}$$

Cette équation est similaire à l'équation (6) à l'exception qu'elle inclut le terme MEM_{SETUP} qui représente la mémoire nécessaire pour gérer les différents GFOR afin de les combiner à la fin. La capacité de mémoire du GPU est un point important à considérer car cela peut engendrer une surcharge dans la communication entre le CPU et le GPU. La transmission de données avec le GPU est plus lente par rapport au temps de traitement du GPU. Afin de minimiser ces transferts de données, il est important de diviser la séquence d'ADN en grands blocs avant de les envoyer au GPU. Cependant, il ne faut pas que la taille des blocs dépasse une certaine limite car de plus grands blocs risquent de produire des erreurs de mémoire lors des calculs par le GPU. Le nombre maximum de GFOR est donné par le minimum entre N_{GFOR1} et N_{GFOR2}:

Équation 8

$$N_{GFOR} = \min(N_{GFOR1}, N_{GFOR2})$$

Dans le cas spécifique du système utilisé, N_{GFOR1} est plus petit que N_{GFOR2}, ce qui signifie que toutes les ressources pour la parallélisation peuvent être utilisées avant que la mémoire GPU ne soit remplie.

À partir des ***blocs*** reçus, ceux-ci seront décomposés en ***fragments***. Afin de déterminer la taille optimale des fragments d'ADN, nous avons effectué une vaste série d'essais pour déterminer la taille de ceux-ci. Les blocs d'ADN de tailles différentes ont été envoyés au GPU et traités par la suite en utilisant des quantités différentes de GFORs. Pour chaque combinaison de dimension de blocs et dimension de fragments, le temps de calcul a été évalué. Les résultats de ces essais sont présentés à la figure 7. Il est à noter que sur la figure 8, le nombre de GFORs n'est pas représenté explicitement mais ce qui est représenté est la taille des fragments d'ADN traité dans les GFOR.

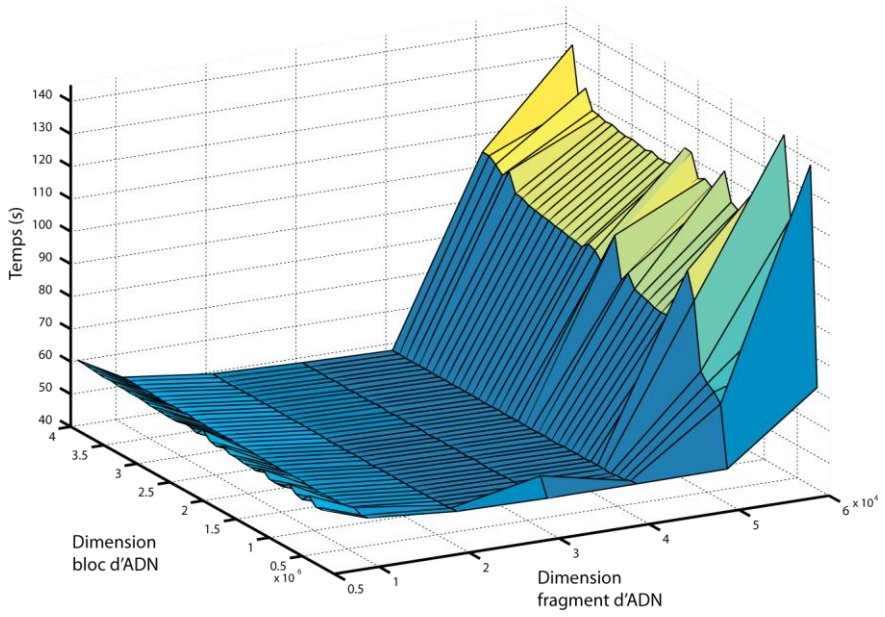


Figure 8: Le temps de traitement d'une séquence de 15 millions de paires de bases en faisant varier la taille des blocs de l'ADN et la taille des fragments d'ADN.

Les résultats présentés dans la figure 8 comprennent tous les transferts de données entre le CPU et la mémoire GPU. Pour faciliter la comparaison, les temps de calcul ont été normalisés pour une séquence d'ADN de 15 millions de pb. Il s'agit là d'une taille maximale estimée dans laquelle un algorithme doit chercher pour trouver un gène candidat entre deux marqueurs génétiques. Ceci a été déterminé en considérant le cas de l'hémochromatose juvénile où les marqueurs les plus proches sont à une distance de 4 cM, soit 4 millions de pb [62]. Dans le cas des autres maladies la plupart des marqueurs sont espacés de moins de 4 cM. Afin de s'assurer de pouvoir traiter tous les cas possibles, l'algorithme proposé permet de faire des analyses allant jusqu'à 15 millions de pb, ce qui est amplement suffisant pour l'identification d'un gène.

Selon la figure 7, il est possible d'observer que l'utilisation d'un fragment d'ADN de taille entre 20 et 40 milles pb donne le temps de traitement minimal. Une taille de fragment d'ADN plus petit ne tirerait pas profit de toute la puissance de calculs en parallèle et donc donnerait des résultats moins performants. Ceci est illustré à la figure 7, où les temps de traitement augmentent à mesure que la taille des fragments tombe à 5 000 pb et pour les gros fragments le temps de traitement augmente rapidement.

4.5 Configuration des tests

Un processeur Intel ® Core ™ i7-2600K processeur (8 Mo de cache, 3,40 GHz, 8 coeurs) a été utilisé avec 8 Go de RAM. La carte graphique utilisée est une GeForce GTX 560 1 Go GDDR5 avec 336 noyaux CUDA. Notons que 336 cœurs CUDA ne permettent pas de profiter directement d'un gain de performance de 336. Ces cœurs sont bien différents de ceux sur un CPU [63]. Pour chacune des implémentations de FFT ou Goertzel, le temps de traitement mesuré comprend également le temps de transfert de données. Il convient de noter que ces tests ne sont pas destinés à quantifier la fiabilité de l'approche choisie puisque ce type d'étude a déjà démontré que l'analyse de fréquence est fiable [64] [46], [65]. Les algorithmes sont disponibles en annexe 3 et 4.

4.6 Résultats

4.6.1 Gène HFE2 pour la validation des analyses

Pour tester l'efficacité des différentes implémentations, plusieurs analyses ont été effectuées sur des séquences d'ADN extraites de la banque de donnée NCBI (version CRCh37.2). Le but de ces analyses est de déterminer si les algorithmes sont en mesure de prédire la présence du gène HFE2 (Hémochromatose juvénile de type 2 - EMBL:AY372521) ainsi que le temps requis pour effectuer ce traitement.

Le gène HFE2 a quatre variantes, chacune ayant un ensemble légèrement différent d'exons. Ces différents exons sont présentés dans le tableau 4. Dans ce tableau, nous voyons qu'il y a quatre exons de tailles différentes avec *3a* et *3b* qui se chevauchent. La stratégie de prédiction de gènes utilisée ici devrait donc être en mesure d'identifier les quatre régions distinctes du gène HFE2 : l'exon 1, l'exon 2, exon 3a/3b et l'exon 4.

Tableau 4: Données relatives du gène HFE2 sur le chromosome 1.

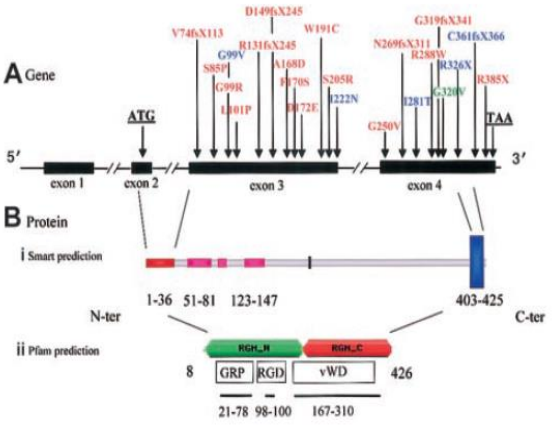
	Exons	Taille des régions codante (pb)
Transcript variant a	1,2,3b,4	2234
Transcript variant b	1,3b,4	2048
Transcript variant c	1, 3a,4	1525
Transcript variant d	1,4	1488

	Exon 1	Exon 2	Exon 3a	Exon3b	Exon 4
Début	145,413,191	145,414,693	145,415,278	145,415,278	145,416,313
Fin	145,413,427	145,414,879	145,415,315	145,415,838	145,417,545
Taille	236	186	37	560	1232

4.6.2 Détection des régions codantes

Pour vérifier la fonctionnalité de la détection des régions codantes, il est possible de tracer les données des analyses en fréquences et de les comparer avec la structure connue du gène [66] [67] (voir figure 9a). Les résultats montrent que toutes les implémentations proposées avec MATLAB produisent les mêmes résultats (voir figure 9b). Dans cette figure, chaque pic de grande taille représente une région probable de codage (exons).

a)



b)

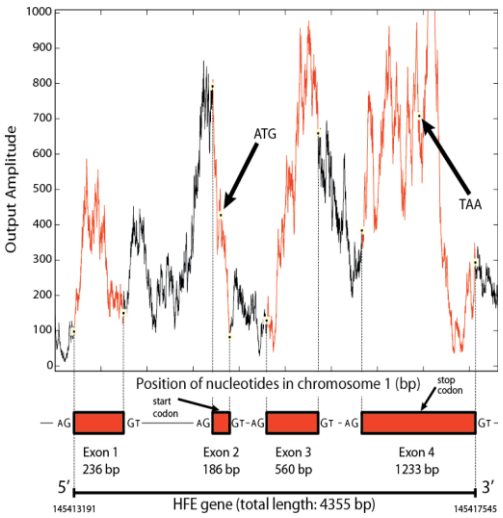


Figure 9: Représentation du gène HFE2, a) structure moléculaire du gène, b) représentation à fréquence de $f_s/3$ pour les différents exons du gène HFE2 avec une fenêtre glissante lors de l'analyse par FFT et l'algorithme de Goertzel.

En comparant les résultats à la structure des gènes présentés dans la figure 8b, il est possible de conclure que l'approche identifie les régions codantes avec succès.

4.6.3 Temps de traitement

Dix différentes implémentations de prédiction de gènes ont été évaluées avec sept différentes tailles de séquences d'ADN. Les résultats sont présentés au tableau 5. Seuls les résultats les plus significatifs sont présentés. Une analyse préliminaire a montré que l'implémentation avec la fonction Goerzel de MATLAB nécessite plus de 9 heures avec le CPU (1 cœur) pour traiter 15 000 000 pb. Ce temps de calcul est représentatif de ce que l'usager moyen obtiendrait puisque MATLAB n'utilise qu'un seul cœur par défaut même lorsque plusieurs cœurs sont disponibles.

La fonction GoertzelMex a été implantée avec deux, quatre et huit cœurs afin de démontrer l'amélioration des performances lorsque des cœurs sont ajoutés au traitement. Pour une séquence de 1 million de pb, les performances avec goertzelMEX ont permis une amélioration par un facteur de 30 comparé à l'implantation avec goertzel.m lorsque les huit cœurs du CPU sont utilisés (Tableau 5, ligne 5). Voici donc une première amélioration du temps de traitement. Une version conçue de toute pièce de l'algorithme de Goertzel a été testée en utilisant les huit cœurs du processeur. Les résultats montrent que, lorsque seul le CPU est utilisé (sans GPU), celui-ci donne les meilleurs résultats (Tableau 5, ligne 6).

Tableau 5: Temps de traitement pour les différentes longueurs de séquence.

				Temps (s) de traitement:						
Fonction	Type de boucle	Traitement	$5x10^3$	$5x10^4$	$2x10^5$	$5x10^5$	$1x10^6$	$5x10^6$	$15\ x10^6$
1 goertzel.m	PARFOR	CPU 8T	1.06	8.29	32.91	82.16	161.89	805.44	TLTC
2 goertzelMEX	FOR	CPU	0.18	1.78	7.11	17.84	35.65	178.30	535.21
3 goertzelMEX	PARFOR	CPU 2T	0.19	0.99	3.86	9.58	19.20	100.39	287.35
4 goertzelMEX	PARFOR	CPU 4T	0.18	0.60	2.36	5.81	11.41	56.27	164.84
5 goertzelMEX	PARFOR	CPU 8T	0.25	0.53	1.95	4.75	9.52	47.49	164.57
6 Goertzel personnalisé	**PARFOR**	**CPU 8T**	0.25	0.37	1.18	2.83	5.56	27.63	87.47
7 FFT de JACKET (séquence complète)	GFOR	GPU	0.03	0.22	0.78	1.90	3.78	18.82	57.68
8 FFT de JACKET (Blocs 1M)	**GFOR**	**GPU**	0.03	0.22	0.78	1.90	3.78	18.90	56.70
9 FFT de Matlab	PARFOR	CPU 8T	0.29	0.42	1.46	3.51	6.95	34.12	109.15

10 Goerztel sur GPU	GFOR	GPU	0.22	0.79	2.82	7.15	14.09	71.01	213.31

La fonction FFT dans MATLAB a également été testée et les délais de traitement sont présentés à la ligne 9 du tableau 5. Les résultats indiquent un temps de traitement qui est beaucoup plus long qu'avec les implémentations de Goertzel. Il est possible de conclure que l'algorithme de Goertzel soit plus efficace que la FFT lorsque le nombre de fréquences ciblées est faible.

Des tests ont aussi été effectués avec le GPU (ligne 7 et 8). Les résultats montrent qu'il faut moins de 1 minute pour compléter les calculs pour 15 000 000 pb. Il est possible d'observer que le traitement en parallèle des analyses a raccourci significativement les temps de calcul. Les tests effectués montrent que le choix de l'algorithme et la mise en œuvre jouent un rôle important dans la faisabilité de la prédiction de gènes dans MATLAB.

Si la fonction goertzel.m avait été utilisée avec un seul cœur, il n'aurait pas été possible d'effectuer les calculs dans un délai raisonnable. Même avec 4 cœurs (8 unité de traitement (threads) activées) et en faisant une extrapolation, le temps de traitement pour 15 millions de paires de base serait d'environ 41 minutes.

L'implémentation avec GoertzelMEX et Goertzel personnalisé permet de réduire le temps de traitement à 90 secondes. Finalement, les meilleurs résultats sont obtenus avec le GPU en 57 secondes (Voir tableau 5).

La stratégie proposée dans ce chapitre permet l'analyse d'une grande quantité de données dans un délai raisonnable tout en étant précis et fiable. Cette première partie du travail montre comment le parallélisme peut être utilisé dans Matlab pour la prédiction de gènes dans une très grande séquence pour produire des résultats qui sont 270 fois plus rapides que l'approche classique [68] Voir l'annexe 7 pour l'intégrale de l'article publié dans BMC.

4.7 Critères de décisions pour l'identification d'un gène

Le graphique de la figure 9 présente le résultat d'une analyse de Goertzel avec une séquence de 3 000 000 pb dans une région avec des gènes connus. L'axe des X contient la position dans le génome tandis que l'axe des Y représente la somme des résultats de l'équation (5). Selon la théorie, une valeur élevée en Y indique une grande probabilité de

région codante. Pour que le logiciel puisse prédire la présence ou l'absence d'un gène, il faut lui fournir des critères non ambigus. Le critère choisi pour cette prise de décision est basée sur 3 paramètres: le seuil positif, le seuil négatif et la dimension de la fenêtre négative. Ces termes seront expliqués dans la section subséquente.

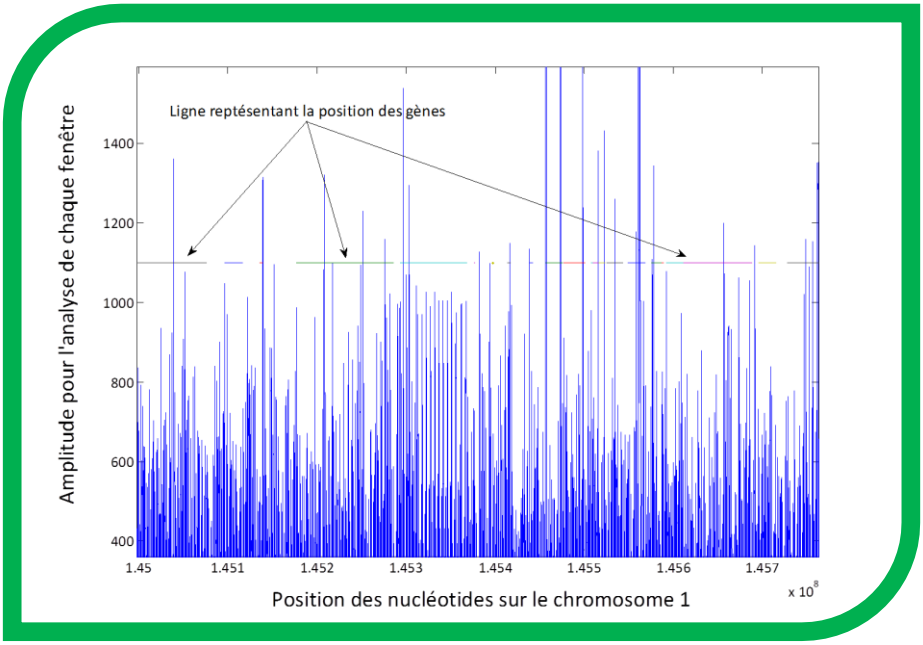


Figure 10: Analyse de FFT.

4.7.1 Seuil positif, négatif et fenêtre négative

La figure 10 contient une ligne horizontale placée en haut de la figure pour indiquer l'emplacement des gènes connus dans la région (autour de Y=1 100). Le ***seuil positif*** est une valeur au-dessus de laquelle l'algorithme prédit la présence d'une région codante. Par contre, sachant que les valeurs trop faibles en Y suggèrent l'absence de gènes, il a fallu un deuxième seuil, appelé ***seuil négatif***, qui permettrait de prédire l'absence de régions codantes. Ainsi, toute valeur en deçà de ce seuil indiquerait l'absence de régions codantes autour de ce point. La région affectée par ce seuil négatif est déterminée par le troisième paramètre, la fenêtre négative. Elle détermine la taille de la région où l'absence de gènes est prédite. Il est donc important de noter que le seuil négatif a priorité sur le seuil positif si celui-ci se trouve à l'intérieur de la zone affectée. La figure 11 montre l'analyse en fréquences d'une région autour du gène HFE2 en mettant en évidence un seuil positif à 400 et un seuil négatif à 350.

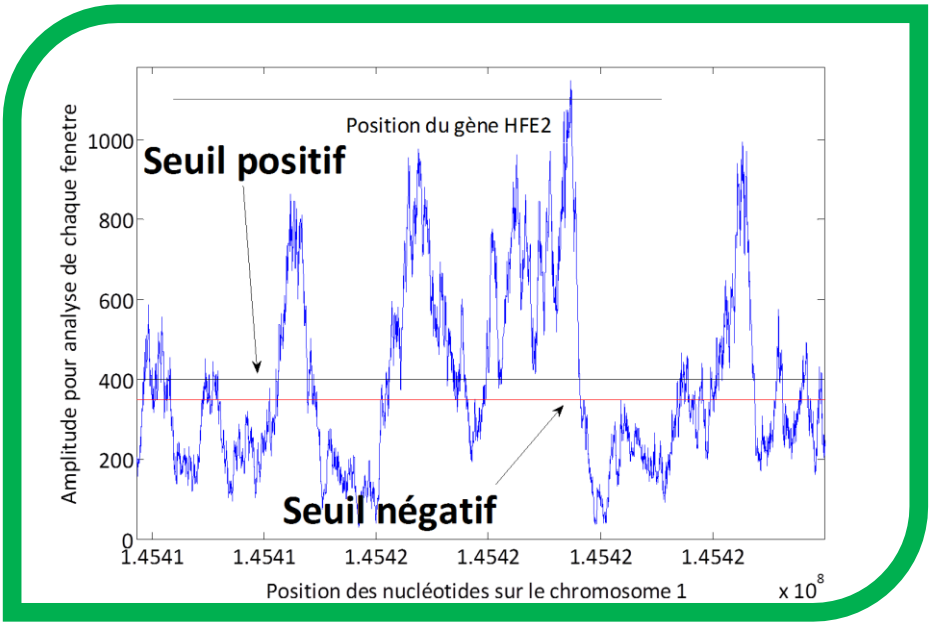


Figure 11: Analyse de FFT montrant les seuils positifs et négatifs.

4.7.2 Processus d'analyse

Afin de déterminer les valeurs optimales de seuils et de fenêtres pour identifier un maximum de gènes positifs et un minimum de faux positifs, il a fallu procéder à un balayage des valeurs possibles pour chacune des trois variables. Ainsi, pour chaque seuil positif et chaque taille de fenêtre donnée, les seuils négatifs ont été évalués. Le tableau 6 montre un exemple d'une analyse ayant deux seuils positifs (25 et 35) dont chacun sera analysé avec différents seuils négatifs (-10, -20, -50, -80 et -100). Pour chaque combinaison de seuils, plusieurs fenêtres négatives couvrant 1, 5 et 10 paires de bases détermineront la région couverte de part et d'autre de chacun des points. Dans cet exemple, 30 combinaisons sont ainsi analysées afin de déterminer celle pouvant identifier correctement un maximum de gènes tout en produisant un minimum de faux positifs.

Tableau 6: Balayage d'une séquence d'ADN avec les seuils et fenêtres.

Seuils positifs	Fenêtres négatives	Seuils négatifs
25	1	-10
25	1	-20
25	1	-50
25	1	-80
25	1	-100
25	5	-10
25	5	-20
25	5	-50
25	5	-80
25	5	-100
25	10	-10
25	10	-20
25	10	-50
25	10	-80
25	10	-100
35	1	-10
35	1	-20
35	1	-50
35	1	-80
35	1	-100
35	5	-10
35	5	-20
35	5	-50
35	5	-80
35	5	-100
35	10	-10
35	10	-20
35	10	-50
35	10	-80
35	10	-100

Le processus de balayage décrit précédemment a été utilisé pour analyser une séquence d'ADN contenant 57 gènes bien connus afin de déterminer la meilleure combinaison de paramètres. L'hypothèse dans cette situation est que les valeurs optimales de seuils positifs, de seuils négatifs et de la taille de la fenêtre seront semblables partout dans le génome. Les résultats de l'analyse sont présentés au tableau 7. En utilisant l'équation empirique décrite en (9), il est possible de calculer un pointage qui devra être maximisé. L'équation met une emphase particulière sur le nombre de gènes détectés et accorde moins d'importance aux faux positifs. Selon ce principe, il a été déterminé que la combinaison qui maximise le nombre de gènes détectés tout en minimisant le nombre de faux positifs se trouve à la dernière ligne du tableau 7.

Équation 9

$$pointage = (Nombre\ de\ gènes\ détectés)^3 / \sqrt{(faux\ positifs)}$$

Tableau 7: Identification des gènes connus et faux positifs avec Goertzel.

Seuils positifs	Fenêtres	Seuils négatifs	Gènes	Faux positifs
200	5	0	60	4648
200	5	50	60	4648
200	5	100	60	4648
200	10	0	60	4648
…	…	…	…	…
300	10	50	59	4091
300	10	100	59	4091
300	25	0	59	4091
300	25	50	59	4091
…	…	…	…	…
100	25	50	61	2191
400	10	350	58	1805
300	25	250	58	1793
400	25	300	58	1683
400	25	350	**57**	**1003**

Cette première partie de la recherche nous permet de cibler des zones qui pourraient contenir des régions d'ADN codant pour un gène. Cependant, la délimitation de ces zones d'expressions reste imprécise. C'est pourquoi nous avons développé une seconde

approche afin d'être plus précis. La seconde partie de ce mémoire traite de cette approche qui est l'analyse des séquences d'ADN à partir d'hexamères.

5 Analyse d'hexamères

5.1 Stratégie complémentaire pour la prédiction de gène

L'analyse en fréquences permet l'identification des exons grâce une propriété (périodicité de trois) qui est souvent présente dans les régions codantes. Pour améliorer le résultat des prédictions, il est possible de procéder à l'analyse de la même séquence utilisant une autre technique basée sur des principes différents et indépendants. La technique choisie, nommée le décompte d'hexamères, permettra d'obtenir des résultats qui pourront être combinés à l'analyse en fréquences afin d'améliorer les prédictions.

5.2 Utilisation des hexamères comme statistique

Les hexamères sont de courtes séquences d'ADN composées de 6 nucléotides. Le nombre d'hexamères que l'on peut retrouver avec les 4 types de bases (A, T, G, C) est de 4^6. Dans les régions codantes, les hexamère correspondent à 2 codons tandis qu'ils représentent simplement 6 nucléotides dans les régions non-codantes. Il est possible d'observer que certains hexamères se retrouvent de façons préférentielles dans les régions codantes tandis que d'autres ne s'y retrouvent que rarement.

Prenons comme exemple un hexamère composé d'un codon stop (TAA, TAG ou TGA). Ce codon est plus rare dans les régions codantes car celui-ci met fin à l'expression d'un exon. De plus, un hexamère composé de deux codons stop ne se retrouvera jamais dans une région codante. D'autres observations ont été effectuées et ont permis de dresser une liste des hexamères qui se trouvent plus fréquemment dans les régions codantes que dans régions non-codantes. À l'aide de cette information, une pondération est attribuée à ces hexamères: une pondération plus élevée est attribuée aux hexamères qui se trouvent plus souvent dans les régions codantes.

De plus, il est possible d'observer que les régions proches des télomères contiennent plus de bases de type G et C. Ces régions sont dites ***riches en G+C*** tandis que les régions plus éloignées sont dites ***pauvres en G+C***. Les AA proches des télomères auraient donc tendance à posséder plus de G et de C dans le codon qui le constitue. La figure 12 montre

un exemple d'un même codon avec une séquence riche et pauvre en G+C. Le codon CGC qui code pour l'arginine dans une séquence riche en G+C change pour AGA dans une séquence pauvre en G+C. La leucine peut être représentée par le codon CTC proche du télomère (riche en G+C) tandis qu'il serait représenté par TTA s'il était plus éloigné (pauvre en G+C). Pour ces raisons, il existe deux pondérations différentes pour l'analyse des hexamères (annexe 5).

Figure 12: Exemple de séquence riche et pauvre en G+C.

CCTGTACACACCACACCGCCTACCTCCCTTCCCCCAAGAAGGGAGTGAGAAAGGCC
Arg

60 % G+C

ATTATTATATTGCAGTACCTAAACTATTAATAATTTTTACAGAAAAAGTGATACTAGTT
Arg

20 %G+C

5.3 Techniques d'analyse

5.3.1 Caractéristiques statistiques des exons dans les gènes humains

Afin de faciliter la découverte de gènes à l'échelle du génome humain, plusieurs outils ont été développés [69] [45]. Parmi ces outils, une base de données composée d'hexamères a été produite pour l'analyse d'ADN. La sélection des hexamères pour établir la pondération fut établie d'après l'article de Zhang [69]. On observe que les hexamères avec une pondération positive se trouvent généralement dans les exons contrairement aux hexamères ayant une pondération négative. Ces régions ayant une pondération négative indiquent une faible probabilité d'y trouver des régions codantes. Bien qu'il y ait 4096 variations d'hexamères, seulement 40 de ceux-ci ont une pondération différente de 0. Les autres hexamères se retrouvent autant dans les introns que dans les exons.

Pour débuter l'analyse, le logiciel commence par classifier les séquences selon qu'ils sont riches ou pauvres en G+C pour toute la longueur de la séquence sélectionnée. L'analyse d'hexamères est effectuée en parcourant une séquence d'ADN et en comptabilisant les hexamères présents. Sachant que les hexamères ont une pondération assignée selon leur probabilité d'être dans une région codante, un calcul peut être effectué afin de déterminer la probabilité d'être dans un exon. Vue le faible nombre d'hexamères considéré dans les pondérations, le temps d'exécution pour effectuer les analyses est négligeable par rapport à l'analyse des FFT. La figure 13 montre le résultat d'une analyse

d'hexamères. De façon générale, plus un compte est positif plus cette région possède les caractéristiques associées à un exon. Sur la figure, il est possible de remarquer qu'il y a des segments de lignes horizontaux autour de la position 100 de l'axe des Y. Ces segments de ligne représentent la présence de gènes existants.

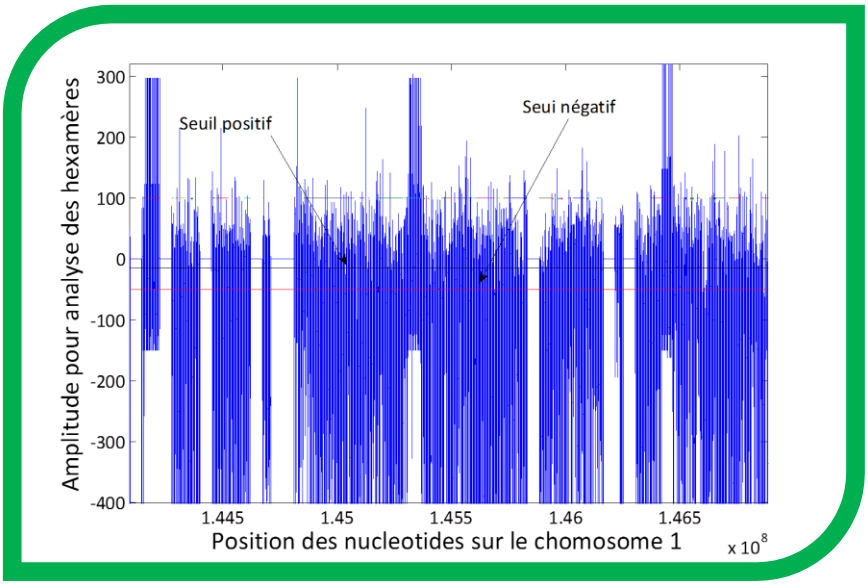


Figure 13: Analyse des hexamères pour une séquence ADN pauvre en G+C.

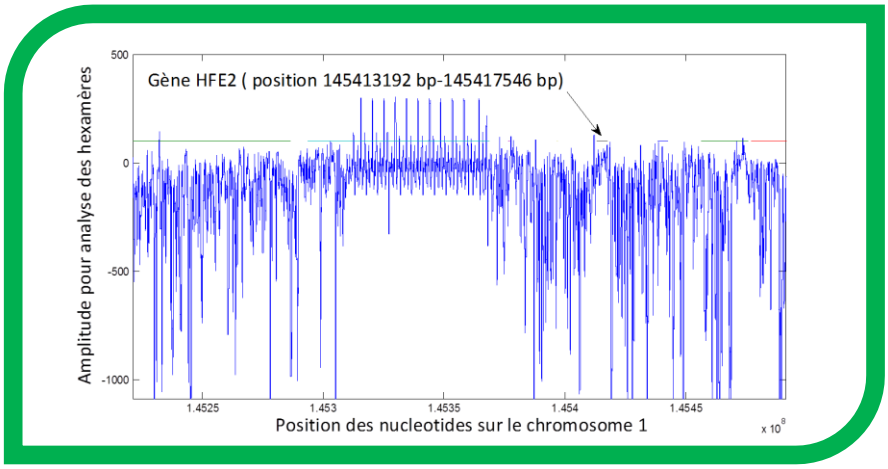


Figure 14: Analyse des hexamères pour une séquence ADN pauvre en G+C.

La figure 14, qui est un agrandissement de la figure 13, nous indique la position du gène HFE2 avec une barre horizontale. Il est possible d'observer que les résultats sont plus

élevés à la position du gène que dans les régions non-codantes adjacentes. Ces résultats concordent avec les résultats escomptés. Les régions ayant des valeurs négatives sont typiquement associées à des introns. Il arrive que les résultats indiquent une amplitude négative pour certains gènes. Cette situation se produit lorsque ces gènes sont classés comme pseudo-gène ou parfois un 'gène' qui se retrouve sur le brin non codant. Ces deux classes de gènes contribuent fortement à augmenter le nombre de gènes qui sont des faux positifs.

Afin d'identifier les régions codantes, des critères non ambigus ont dû être définis. Pour ce faire, les critères définis au chapitre 4 ont été repris, soient le seuil positif, le seuil négatif et la fenêtre négative. Les valeurs associées à ces critères ont été déterminées en utilisant la technique de balayage. Ceci est semblable à ce qui a été effectué avec les analyses de fréquences. Les paramètres ont été balayés dans une séquence connue pour déterminer quelle combinaison serait en mesure de produire les meilleures prédictions tout en limitant les faux positifs. Afin d'optimiser les paramètres, il serait possible de balayer les valeurs de seuils positifs, négatifs et de taille des fenêtres avec des intervalles plus petites. De plus, il serait également possible d'effectuer cette optimisation avec des méthodes bien connues telle qu'une approche convexe ou non-linéaire. Bien qu'intéressant, ceci sort des cadres de ce mémoire.

5.4 Résultats de l'analyse par hexamère

Les tableaux 8 et 9 montrent partiellement les résultats de balayage obtenus avec les différentes combinaisons de seuils et fenêtres pour les deux groupes de pondération. Le but est d'identifier la meilleure combinaison qui permettra d'identifier un maximum de gènes tout en minimisant le nombre de faux positifs. Les encadrés dans les tableaux 8 et 9 indiquent plusieurs combinaisons intéressantes. Dans le tableau 8 il y a 4 combinaisons avec le même nombre de détections soit 57 gènes positifs et 346 faux positifs. L'une des 4 combinaisons peut être utilisée pour les analyses subséquentes. La même règle s'applique pour les résultats du tableau 9.

Tableau 8: Analyse des résultats hexamères pauvre G+C.

Seuil positif	Fenêtre	Seuil négatif	Gènes	Faux positifs
15	50	-10	56	567
15	75	-20	56	534
15	75	-10	56	471
15	100	-20	56	466
5	100	-10	57	382
-5	100	-10	57	363
-30	100	-10	57	346
-20	100	-10	57	346
-10	100	-10	57	346
-15	100	-10	57	346
-15	1	-80	58	1061
-15	1	-100	58	1061
-15	5	-100	58	1061
-15	1	-20	58	1060
-15	1	-50	58	1059
-15	5	-80	58	1059
-15	10	-100	58	1059
-15	10	-80	58	1055
-15	5	-50	58	1052
-30	1	-10	58	1047
…	…	…	…	…
-30	20	-100	60	1121
-30	20	-80	60	1104
-30	10	-50	60	1091
-30	50	-100	60	1054

Tableau 9: Analyse des résultats hexamères riche G+C.

Seuil positif	Fenêtre	Seuil négatif	Gènes	Faux positifs
15	50	-50	57	743
15	20	-10	57	725
15	50	-20	57	699
-5	75	-10	57	667
15	50	-10	57	646
-30	75	-10	57	612
-20	75	-10	57	612
-10	75	-10	57	612
-15	75	-10	57	612
5	1	-10	58	1082
5	1	-20	58	1082
5	1	-50	58	1082
5	1	-80	58	1082
…	…	…	…	…
-15	1	-50	60	1417
-15	1	-80	60	1417
-15	1	-100	60	1417
-15	5	-80	60	1417
-15	5	-100	60	1417
-15	5	-50	60	1414
-15	1	-20	60	1413
-15	10	-100	60	1413
-15	10	-80	60	1412

6. Fusion des données et interprétation des résultats

Deux approches en parallèle ont été appliquées pour la prédiction des exons. Après examen des données avec l'analyse en fréquences et le décompte d'hexamères, l'étape subséquente est de mettre ces données ensemble en combinant les deux approches. L'intersection est l'opération qui a été choisie pour effectuer ce processus. Quatre intersections seront faites : deux intersections avec l'analyse en fréquences et l'analyse des hexamères avec la pondération riche et pauvre en G+C pour les gènes connus et deux autres intersections toujours avec l'analyse en fréquences et les hexamères, mais cette fois-ci pour la détection des faux positifs. La figure 17 nous montre qu'il est important lors de l'analyse avec les hexamères de procéder avec la bonne pondération soit celle pour les

séquences pauvres ou riches en G + C, car ceci permet la détection de faux positifs. La séquence étudiée dans ce cas-ci indique que la pondération pour une séquence pauvre en G + C réduit de la moitié les faux positifs.

L'intersection entre les résultats en fréquence et les analyses d'hexamères nous donne 13918 faux positifs avec la pondération pauvre en G+C et 24952 faux positifs avec la pondération riche en G+C (données non montrées). Ce grand nombre de faux positif vient du fait que l'identification d'une région codante se fait à chaque fois qu'une fenêtre est déplacée d'un nucléotide. Il arrive souvent que plusieurs fenêtres consécutives génèrent des faux positifs. Dans ce cas, ces faux positifs seront regroupés puisqu'ils représentent effectivement la même prédiction. Ce processus de regroupement est illustré sur les figures 15 et 16. En effectuant un agrandissement sur la figure 15, nous pouvons apercevoir que plus la bande est large, plus grand est le nombre de nucléotides regroupés (figure 16). Nous avons analysé une séquence de 3 000 000 pb dans laquelle se trouvent 57 gènes connus. La densité des informations est très grande, afin de mieux voir les résultats, la figure 16 montre un agrandissement de cette zone analysée. Nous pouvons voir clairement que le regroupement des faux positifs est entre les gènes déjà identifié. La figure 16 montre clairement les regroupements de faux positif entre trois gènes déjà bien identifiés.

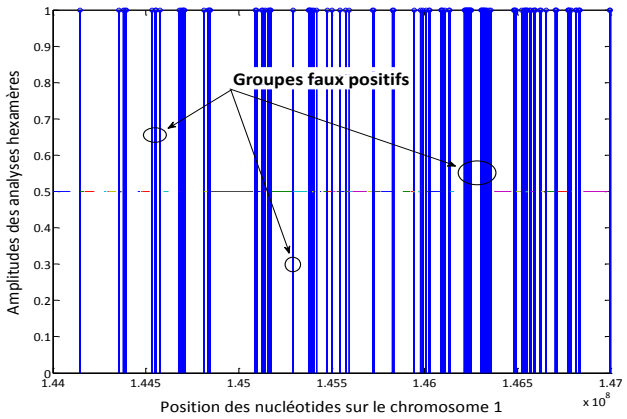


Figure 15: Groupes de faux positifs insertion Goerzel et hexameres pauvre en G+C.

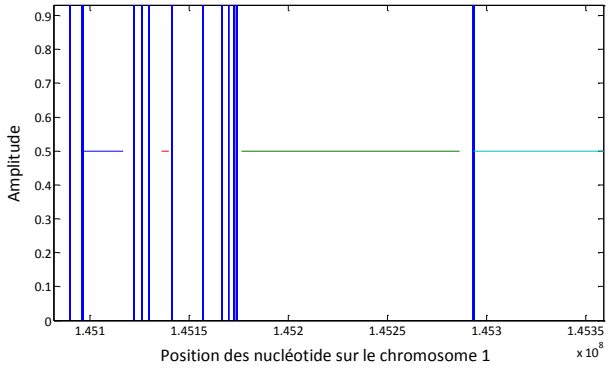


Figure 16: Agrandissement de la figure 15.

L'algorithme utilisé dans l'analyse des hexamères a permis de réduire de beaucoup l'identification de faux positifs par conséquent de diminuer considérablement le nombre. Nous avions identifié au départ 24 952 faux positifs qui ont été réduits de plus de 97% soit à 612 faux positifs pour les séquences riches en G + C. On remarque également que tous les gènes connus dans cette séquence ciblée (57 gènes) ont été identifiés.

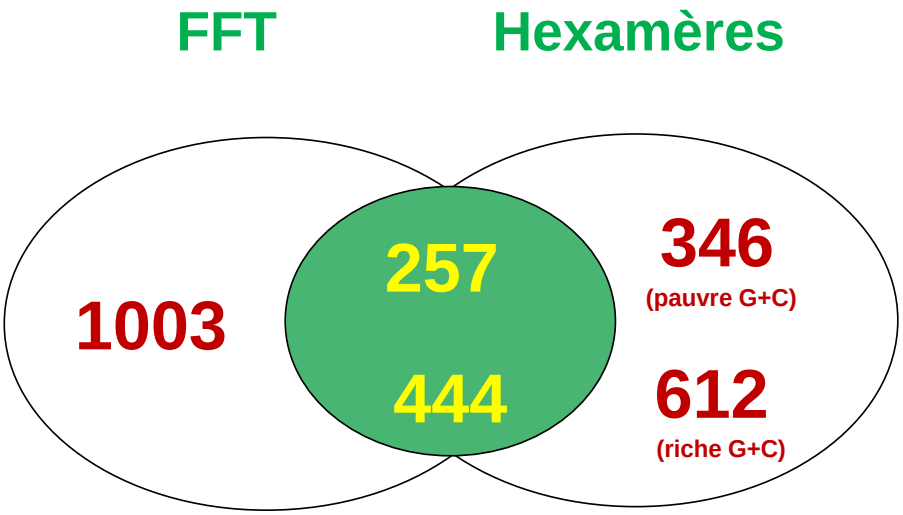


Figure 17:Compilation des analyses FFT et hexamères.

Avec l'analyse de fréquence, l'algorithme a généré 1003 faux positifs. Cependant, lorsque le résultat des analyses en fréquences est combiné au résultat des analyses des

hexamères, le nombre de faux positifs est réduit. On ne retrouve plus que seulement 257 faux positifs avec les hexamères dont la pondération est faite avec une séquence pauvre en G + C et 444 faux positifs avec les hexamères riches en G + C. L'intersection des deux approches permet de diminuer significativement le nombre de faux positifs.

Bien que la combinaison des deux approches ne délimite pas nécessairement de manière précise les limites des régions codantes, celle-ci permet de mieux cibler les exons. Le nombre de faux positif est considérablement réduit par cette approche.

Il est important de souligner le fait qu'il existe des logiciels qui permettent de bien définir le début et la fin d'un gène avec les caractéristiques des gènes telle que la boîte TATA, le site CAT, les donneurs, les accepteurs, etc. Par contre, ces logiciels ne permettent que l'analyse de très petites séquences, soit environ quelques centaines de pb. Afin de bénéficier de ces logiciels, il est primordial de préalablement identifier les régions contenant un ou plusieurs gènes avec l'analyse par FFT et des hexamères.

7. Conclusion

Bien que l'analyse par FFT ne permette pas la démarcation exacte du début ni la fin des exons, ceux-ci peuvent analyser de grandes séquences et cibler les régions d'expressions génétiques c'est-à-dire un gène ou plusieurs gènes continus. L'avantage de l'approche proposée est la possibilité d'effectuer un balayage du génome humain dans des temps raisonnables (15 000 000 pb/ min.) ce qu'aucun autre groupe de recherche n'a réussi à faire jusqu'à présent. Nous avons amélioré de beaucoup le temps de traitement des données. La seconde partie du travail fut de poursuivre les analyses de séquences d'ADN avec un autre algorithme qui est celui de l'analyse par hexamères. La combinaison des deux analyses permet de mieux cibler les régions impliquées dans la structure d'un gène et de réduire significativement le nombre de faux positifs. Une fois, les régions d'intérêt ciblées, il existe des programmes capables d'effectuer des analyses avec de petites séquences pour la démarcation précise des exons. Il est important de noter que ces outils de travail qui sont mis à la disposition des chercheurs ne sont pas définitifs. La précision des régions codantes et la détection de faux positifs pourraient être améliorées en optimisant la pondération pour les analyses d'hexamères. Il serait aussi intéressant de tester l'approche sur d'autres génomes comme ceux des microbiotes qui sont de plus en plus analysés afin de mieux servir la santé des humains.

Références

[1] G. Mendel, “Verhandlungen des naturforschenden Vereines in Brünn, Bd. IV für das Jahr, 1865, Abhandlungen,” in *Versuche über Pflanzenhybriden*, 1865, vol. 4, pp. 3–47.

[2] G. Mendel, “Versuche über Pflanzen-Hybriden (1865) in: Verhandlungen des Naturforschenden Vereines. Abhandlungen. Brünn,” *(http://www.netspace.org./MendelWeb/).*, no. 1865, pp. 3–47, 1866.

[3] E. von Tschermack, “Uber Künstliche Kreuzung Bei Pisum sativum,” *Berichte der Dtsch. Bot. Gesellschaft*, vol. 18, pp. 232–239, 1900.

[4] H. de Vries, “Mass Mutation in zea mays,” *Science (80-.).*, vol. 47, no. 2119, pp. 465–467, 1918.

[5] C. Lenay, “Hugo De Vries: from the theory of intracellular pangenesis to the rediscovery of Mendel,” *Comptes Rendus l’Académie des Sci. - Ser. III - Sci. la Vie*, vol. 323, no. 12, pp. 1053–1060, Dec. 2000.

[6] J. Harwood, “The rediscovery of Mendelism in agricultural context: Erich von Tschermak as plant-breeder,” *Comptes Rendus l’Académie des Sci. - Ser. III - Sci. la Vie*, vol. 323, no. 12, pp. 1061–1067, Dec. 2000.

[7] T. H. Morgan, “Sex Limited Inheritance in Drosophila,” *Science (80-.).*, vol. 32, pp. 120–122, 1910.

[8] T. H. Morgan, A. H. Sturtevant, H. J. Muller, and C. B. Bridges, “The Mechanism of Mendellian Heredity,” *New York Henry Holt Co.*, pp. 1–262, 1915.

[9] O. T. Avery, C. M. MacLeod, and M. McCarty, “Stidies on the chemical nature of the substance inducing transformation of pneumococcal types,” *J.Exp. Med.*, vol. 79, no. 6, pp. 137–158, 1944.

[10] J. D. Watson and F. H. C. Crick, “the Structure of Dna,” *Cold Spring Harb. Symp. Quant. Biol.*, vol. 18, pp. 123–131, Jan. 1953.

[11] S. R. Rivard, M. Cappadocia, and B. S. Landry, “A comparison of RFLP maps based on anther culture derived, selfed, and hybrid progenies of Solanum chacoense.,” *Genome*, vol. 39, no. 4, pp. 611–621, Aug. 1996.

[12] S. D. Tanksley, R. Bernatzky, N. L. Lapitan, and J. P. Prince, “Conservation of gene repertoire but not gene order in pepper and tomato,” *Proc. Natl. Acad. Sci. USA*, vol. 85, no. September, pp. 6419–6423, 1988.

[13] S. N. Cohen, A. C. Y. Chang, H. W. Boyert, and R. B. Hellingt, “Construction of Biologically Functional Bacterial Plasmids In Vitro,” *Proc. Natl. Acad. Sci. USA*, vol.

70, no. 11, pp. 3240–3244, 1973.

[14] D. A. Jackson, R. H. Symonst, and P. Berg, “Biochemical Method for Inserting New Genetic Information into DNA of Simian Virus 40: Circular SV40 DNA Molecules Containing Lambda Phage and the Galactose Operon of Escherichia coli,” *Proc. Nat. Acad. Sci. USA*, vol. 69, no. 10, pp. 2904–2909, 1972.

[15] E. S. Lander, L. M. Linton, B. Birren, and E. Al., “Initial sequencing and analysis of the human genome.,” *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.

[16] J. C. Venter, M. D. Adams, E. W. Myers, and E. Al., “The sequence of the human genome,” *Science*, vol. 291, no. 5507, pp. 1304–1351, 2001.

[17] R. D. Sleator, C. Shortall, and C. Hill, “Metagenomics,” *Lett. Appl. Microbiol.*, vol. 47, pp. 361–366, 2008.

[18] P. J. Turnbaugh, R. E. Ley, M. Hamady, C. M. Fraser-Liggett, R. Knight, and J. I. Gordon, “The Human Microbiome Project,” *Nature*, vol. 449, no. October, pp. 804–810, 2007.

[19] J.-M. Claverie, “GENE NUMBER: What If There Are Only 30,000 Human Genes?,” *Science (80-.).*, vol. 291, no. 5507, pp. 1255–1257, Feb. 2001.

[20] G. D. Schuler, M. S. Boguski, E. A. Stewart, and et al., “A Gene Map of the Human Genome,” *Science (80-.).*, vol. 274, no. 5287, pp. 540–546, Oct. 1996.

[21] F. Antequera, “Cellular and Molecular Life Sciences Structure , function and evolution of CpG island promoters,” *Cell. Mol. Life Sci.*, vol. 60, pp. 1647–1658, 2003.

[22] J. O. Andersson, W. F. Doolittle, and C. L. Nesbø, “Genomics. Are there bugs in our genome?,” *Science*, vol. 292, no. 5523, pp. 1848–50, Jun. 2001.

[23] T. Bienvenu, C. Meunier, S. Bousquet, S. Chiron, L. Richard, A. Gautheret-Dejean, J.-F. Rouselle, and D. Feldmann, “Les techniques d’extraction de l’ADN à partir d’un échantillon sanguin,” *Ann. Biolopgie Clin.*, vol. 57, no. 1, pp. 77–84, 1999.

[24] F. Gao and C.-T. Zhang, “Comparison of various algorithms for recognizing short coding sequences of human genes.,” *Bioinformatics*, vol. 20, no. 5, pp. 673–781, Mar. 2004.

[25] S. R. Eddy, “BIOINFORMATICS REVIEW Profile hidden Markov models,” *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.

[26] D. Boffelli, J. Mcauliffe, D. Ovcharenko, K. D. Lewis, L. Pachter, and E. M. Rubin, “Phylogenetic Shadowing of Primate Sequences to Find Functional Regions of the Human Genome,” *Science (80-.).*, vol. 299, no. 5611, pp. 1391–1394, 2003.

[27] C. T. Zhang, Z. S. Lin, M. Yan, and R. Zhang, “A novel approach to distinguish between intron-containing and intronless genes based on the format of Z curves.,” *J. Theor. Biol.*, vol. 192, no. 4, pp. 467–473, 1998.

[28] M. Yan, Z.-S. Lin, and C. Zhang, “A new Fourier transform approach for protein coding measure based on the format of the Z curve,” *BIOINFORMATICS*, vol. 14, no. 8, pp. 685–690, 1998.

[29] C. T. Zhang and J. Wang, “Recognition of protein coding genes in the yeast genome at better than 95% accuracy based on the Z curve.,” *Nucleic Acids Res.*, vol. 28, no. 14, pp. 2804–2814, 2000.

[30] J. Wang and C.-T. Zhang, “Identification of protein-coding genes in the genome of vibrio cholerae with more than 98% accuracy using occurrence frequencies of single nucleotides,” *Eur. J. Biochem.*, vol. 268, no. 15, pp. 4261–4268, 2001.

[31] C. T. Zhang, R. Zhang, and H. Y. Ou, “The Z curve database: A graphic representation of genome sequences,” *Bioinformatics*, vol. 19, no. 5, pp. 593–599, 2003.

[32] C. T. Zhang and R. Zhang, “Analysis of distribution of bases in the coding sequences by a diagrammatic technique.,” *Nucleic Acids Res.*, vol. 19, no. 22, pp. 6313–6317, 1991.

[33] M. Q. Zhang, “Identification of protein coding regions in the human genome by quadratic discriminant analysis.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 94, no. 2, pp. 565–568, Jan. 1997.

[34] Y. Jiang, B. Cukic, D. a Adjeroh, H. D. Skinner, J. Lin, Q. J. Shen, and B.-H. Jiang, “An algorithm for identifying novel targets of transcription factor families: application to hypoxia-inducible factor 1 targets.,” *Cancer Inform.*, vol. 7, pp. 75–89, Jan. 2009.

[35] A. Tchegho, H. Mattes, and S. Sattler, “Optimal high-resolution spectral analyzer,” *Proc. -Design, Autom. Test Eur. DATE*, pp. 62–67, 2008.

[36] J. W. Fickett, “Recognition of protein coding regions in DNA sequences,” *Nucleic Acids Res.*, vol. 10, no. 17, pp. 5303–5318, 1982.

[37] R. F. Voss, “Evolution of long-range fractal correlations and 1/f noise in DNA base sequences,” *Phys. Rev. Lett.*, vol. 68, no. 25, pp. 3805–3808, 1992.

[38] S. Tiwari, S. Ramachandran, A. Bhattacharya, S. Bhattacharya, and R. Ramaswamy, “Prediction of probable genes by Fourier analysis of genomic sequences,” *Bioinformatics*, vol. 13, no. 3, pp. 263–270, 1997.

[39] B. D. Silverman and R. Linsker, “A measure of DNA periodicity.,” *J. Theor. Biol.*, vol. 118, no. 3, pp. 295–300, Feb. 1986.

[40] S. A. Marhon and S. C. Kremer, “Gene Prediction Based on DNA Spectral Analysis: A

Literature Review.," *J. Comput. Biol.*, vol. 18, no. 4, pp. 639–676, Mar. 2011.

[41] A. R. Fuentes, J. V. L. Ginori, and R. Grau Abalo, "Detection of Coding Regions in Large DNA Sequences Using the Short Time Fourier Transform with Reduced Computational Load," *Lect. Notes Comput. Sci.*, vol. 4225, no. 11, pp. 902 – 909, 2006.

[42] W. Li, T. G. Marr, and K. Kaneko, "Understanding Long-range Correlations in DNA Sequences," *Phys. D Nonlinear Phenom.*, vol. 75, no. 1–3, pp. 392–416, 1994.

[43] Akhtar Mahmood, E. Ambikairajah, and J. Epps, "Digital Signal Processing Techniques for Gene Finding in Eukaryotes," *Digit. Signal Process.*, pp. 144 – 152, 2008.

[44] D. Anastassiou, "Frequency-domain analysis of biomolecular sequences," *Bioinformatics*, vol. 16, no. 12, pp. 1073–1081, 2000.

[45] D. Kotlar and Y. Lavner, "Gene Prediction by Spectral Rotation Measure : A New Method for Identifying Protein-Coding Regions," *Genome Res.*, vol. 13, no. 8, pp. 1930–1937, 2003.

[46] S. Datta, A. Asif, and H. Wang, "Prediction of protein coding regions in DNA sequences using Fourier spectral characteristics," *IEEE/ sixth Int. Symp. Multimed. Softw. Eng.*, pp. 160 – 163, 2004.

[47] S. Data and A. Asif, "A FAST DFT BASED GENE PREDICTION ALGORITHM FOR IDENTIFICATION OF PROTEIN CODING REGIONS," *IEEE*, vol. 5, no. 4, p. v/653–v/656, 2005.

[48] P. P. Vaidyanathan and B.-J. and Yoon, "The role of signal-processing concepts in genomics and proteomics.," *J. Franklin Inst.,* vol. 341, no. 1–2, pp. 111–135, 2004.

[49] S. Datta, A. Asif, and H. Wang, "Prediction of Protein Coding Regions in DNA sequences Using Fourier Spectral Characteristics," *Proc. IEEE*, no. 2, pp. 3–6, 2004.

[50] C. Yin and S. S.-T. Yau, "Prediction of protein coding regions by the 3-base periodicity analysis of a DNA sequence.," *J. Theor. Biol.*, vol. 247, no. 4, pp. 687–694, 2007.

[51] B. Issac, H. Singh, H. Kaur, and G. P. S. Raghava, "Locating probable genes using Fourier transform approach," *Bioinformatic Appl. note*, vol. 18, no. 1, pp. 196–197, 2002.

[52] C. Burge and S. Karlin, "Prediction of complete gene structures in human genomic DNA.," *J. Mol. Biol.*, vol. 268, no. 1, pp. 78–94, Apr. 1997.

[53] P. Flicek, E. Keibler, P. Hu, I. Korf, and M. R. Brent, "Leveraging the mouse genome for gene prediction in human: from whole-genome shotgun reads to a global

synteny map.," *Genome Res.*, vol. 13, pp. 46–54, 2003.

[54] R. Yeh, L. P. Lim, and C. B. Burge, "Computational Inference of Homologous Gene Structures in the Human Genome," *Genome Res.*, vol. 11, pp. 803–816, 2001.

[55] D. Bielińska-Wąz, T. Clark, P. Wąz, W. Nowak, and A. Nandy, "2D-dynamic representation of DNA sequences," *Chem. Phys. Lett.*, vol. 442, pp. 140–144, 2007.

[56] S. S.-T. Yau, "DNA sequence representation without degeneracy," *Nucleic Acids Res.*, vol. 31, no. 12, pp. 3078–3080, Jun. 2003.

[57] P. D. Cristea, "Conversion of nucleotides sequences into genomic signals.," *J. Cell. Mol. Med.*, vol. 6, no. 2, pp. 279–303, 2002.

[58] S. S.-T. Yau, "DNA sequence representation without degeneracy," *Nucleic Acids Res.*, vol. 31, no. 12, pp. 3078–3080, Jun. 2003.

[59] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *Math. Assoc. Am.*, vol. 65, no. 1, pp. 34–35, 1958.

[60] S. Tiwari, S. Ramachandran, A. Bhattacharya, S. Bhattacharya, and R. Ramaswamy, "Prediction of probable genes by fourier analysis of genomic sequences," *Bioinformatics*, vol. 13, no. 3, pp. 263–270, Oct. 1997.

[61] Gunawan T.S., E. J., and A. E., "Boosting approch to exon detection in DNA sequences.," *Electr, Lett.,* vol. 44, pp. 323–324, 2008.

[62] S. R. Rivard, C. Lanzara, D. Grimard, M. Carella, H. Simard, R. Ficarella, R. Simard, A. P. D'Adamo, C. Férec, C. Camaschella, C. Mura, A. Roetto, M. De Braekeleer, L. Bechner, and P. Gasparini, "Juvenile hemochromatosis locus maps to chromosome 1q in a French Canadian population.," *Eur. J. Hum. Genet.*, vol. 11, no. 8, pp. 585–9, Aug. 2003.

[63] Nvidia, "Whitepaper NVIDIA's Next Generation CUDA Compute Architecture," pp. 1–22, 2009.

[64] D. Anastassiou, "GENOMIC SIGNAL PROCESSING," *IEEE Signal Process. Mag.*, no. July, pp. 8–20, 2001.

[65] Anastassiou Dimitris, "Genomic Signal Processing," *IEEE Signal Process. Mag.*, vol. 18, no. 4, pp. 8– 20, 2001.

[66] S. R. Rivard, C. Lanzara, D. Grimard, M. Carella, H. Simard, R. Ficarella, R. Simard, A. P. D'Adamo, C. Férec, C. Camaschella, C. Mura, A. Roetto, M. De Braekeleer, L. Bechner, and P. Gasparini, "Juvenile hemochromatosis locus maps to chromosome 1q in a French Canadian population.," *Eur. J. Hum. Genet.*, vol. 11, no. 8, pp. 585–9, Aug. 2003.

[67] C. Lanzara, A. Roetto, F. Daraio, S. Rivard, R. Ficarella, H. Simard, T. M. Cox, M. Cazzola, A. Piperno, A. P. Gimenez-Roqueplo, and others, “Spectrum of hemojuvelin gene mutations in 1q-linked juvenile hemochromatosis,” *Blood*, vol. 103, no. 11, p. 4317, 2004.

[68] R. Rivard1*, Sylvain-Robert † Mailloux1†, Jean-Gabriel Beguenane2 and H. T. Bui1, “Design of high-performance parallelized gene predictors in MATLAB,” *BMC Res. Notes*, pp. 183–193, 2012.

[69] M. Q. Zhang, “Statistical Features of Human Exons and Their Flanking Regions,” *Hum. Mol. Genet.*, vol. 7, no. 5, pp. 919–932, 1998.

Annexes

Annexe 1

Code à une lettre	Code à trois lettres	Acides aminés	Codons possible
A	Ala	Alanine	GCA, GCC, GCG, GCT
C	Cys	Cystéine	TGC, TGT
D	Asp	Acide aspartique	GAC, GAT
E	Glu	Acide glutamique	GAA, GAG
F	Phe	Phénylalanine	TTC, TTT
G	Gly	Glycine	GGA, GGC, GGG, GGT
H	His	Histidine	CAC, CAT
I	Ile	Isoleucine	ATA, ATC, ATT
K	Lys	Lysine	AAA, AAG
L	Leu	Leucine	CTA, CTC, CTG, CTT, TTA, TTG
M	**Met**	**Méthionine**	**ATG**
N	Asn	Asparagine	AAC, AAT
P	Pro	Proline	CCA, CCC, CCG, CCT
Q	Gln	Glutamine	CAA, CAG
R	Arg	Arginine	AGA, AGG, CGA, CGC, CGG, CGT
S	Ser	Sérine	AGC, AGT, TCA, TCC, TCG, TCT
T	Thr	Tréonine	ACA, ACC, ACG, ACT
V	Val	Valine	GTA, GTC, GTG, GTT
W	Trp	Tryptophane	TGG
X	**X**	**Codon stop**	**TAA, TAG, TGA**
Y	Tyr	Tyrosine	TAC, TAT

Annexe 2

Exemple de code pour utiliser JACKET_seq_FFT.m [6]

Exemple de traitement d'une séquence en utilisant JACKET_seq_FFT.m

```
%% Example code for using JACKET_seq_FFT.m
% Minimalistic example of how to process a sequence using JACKET_seq_FFT.m
%%  Reset MATALB workspace
clear all
close all
clc

%% Define sequence size and start point for analysis
% Define start and end positions of sequence to analyse.
seq_start       = 145413000;
seq_end         = 145417000;
seq_size        = seq_end – seq_start;

%% Read sequence from fasta file
% Matlab bioinformatics toolbox has a function for reading fasta file.
% For diverse reasons (error handling, etc.) it is slow.  In this example, we use the
% bare minimum of commands to load the DNA sequence.  All we want is the
% data without EOL chars.

% !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
%       NOTE:  the fasta file in this example has no headers.
%       If yours does, strip headers accordingly before calling function.
% !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
%
% Location of chromosome 1:
% ftp://ftp.ensembl.org/pub/release64/fasta/homo_sapiens/dna/Homo_sapiens.GRCh37.64.dna.chromosome.1.fa.gz
% Don't forget to strip headers (either in or outside of Matlab)
%
% DNA source file path
Filename = 'Homo_sapiens.GRCh37.64.dna.chromosome.1.fa'; % Change accordingly.
% Open source file
```

[6] http://www.biomedcentral.com/content/supplementary/1756-0500-5-183-S1.m

```
Fid = fopen (filename);
% Read all raw sequence data in CPU memory
Data = fread (fid, '*char');
% Close source file
Fclose (fid);

% Remove all endlines characters
newline = char (10); % This is unix EOL character
raw_sequence = strrep(data',newline,'');

% Clear large temporary values no longer needed
Clear data;

%%  Data analysis using GPU
[ANALYSIS_RESULTS, GPU_TIME] = JACKET_seq_FFT (raw_sequence (seq_start: seq_start
+ seq_size), 351, 118, 20000);
%%  Display results
Plot(seq_start: seq_start + seq_size, ANALYSIS_RESULTS)
```

```
function [ FREQ_DATA, TIME] = JACKET_seq_FFT(SEQUENCE,W_SIZE,FREQ,GFOR_SIZE)
% JACKET_seq_FFT Frequency analysis on a sequence using 'JACKET' s FFT
%
% JACKET_seq_FFT(SEQUENCE,W_SIZE,FREQ,GFOR_SIZE) will run the FFT on
% every sliding window % of size W_SIZE in the entire SEQUENCE.  All results for
% FREQuency  FREQ are stored in  FREQ_DATA.
%
% TIME indicates the GPU processing time, including memory transfers.
%
% GFOR_SIZE defines the chunk size of a sequence that are to be processed by
% GFORs at the same time.  Too large a value will cause out of memory errors or, if
% enough GPU memory is available, will cause slowdowns.  Off the shelf GPUs
% have a sweet spot of around 20 to 60K, depending on the GPU's ram.
```

```
% Clear GPU memory
Clear gpu_hook;

%%  Data formatting in CPU memory
% Find SEQUENCE size
Seq_size = max (size(SEQUENCE));

% A, T, G, C arrays (for storing Os and 1s)
nuc_arrays = false (seq_size, 4);

% Converting sequence to 0-1 arrays
ACTG_numbers = single('ACTG');
for current_vec = 1:4
nuc_arrays(find(SEQUENCE == ACTG_numbers(current_vec)),current_vec) = 1;
end
% Start the clock before GPU processing and communications
tic
% Load the 4 sequences in GPU memory
% For the FFT to work, binary values are read into gsingle arrays
Gnuc_arraya = gsingle(nuc_arrays(:,1));
Gnuc_arrayc = gsingle(nuc_arrays(:,2));
Gnuc_arrayt = gsingle(nuc_arrays(:,3));
Gnuc_arrayg = gsingle(nuc_arrays(:,4));

% Results are to be stored in this (also in GPU memory)
G_final_result = gzeros(4,seq_size,'single');
%% Preparing the different indexes for GFORs
% GFOR restrictions can be found at:
%  htt://wiki.acceleteyes.com/wiki/index.php/GFOR Usage
% Index size for every array used inside a single GFOR
idx = 0:(W_SIZE-1);

% Find the size of the data which will be processed in the very last
% GFOR. This is because our sequence size does not always match 'X times GFOR_SIZE'.
last_one = 0;
for m=1:GFOR_SIZE:((seq_size-W_SIZE)-GFOR_SIZE)
    for k = m+(0:GFOR_SIZE-1)

        last_one=k;
```

```
end
end
% The value of last_one is the start point for the last GFOR
%% GFOR PROCESSING
% Here we run an FFT on every window, then slide the window to the right
% We then extract the results at FREQuency 118 to match the goertzel
% results
for m = 1:GFOR_SIZE:((seq_size-W_SIZE)-GFOR_SIZE)
        gfor k=m+(0:GFOR_SIZE-1)
    ra = fft(Gnuc_arraya(k+idx));
    rc = fft(Gnuc_arrayc(k+idx));
    rt = fft(Gnuc_arrayt(k+idx));
    rg = fft(Gnuc_arrayg(k+idx));

    G_final_result(1,k)=abs(ra(FREQ));
    G_final_result(2,k)=abs(rc(FREQ));
    G_final_result(3,k)=abs(rt(FREQ));
    G_final_result(4,k)=abs(rg(FREQ));
gend
end
% This does the same processing for the last window of data
gfor k = (last_one+1):(seq_size-W_SIZE)
    ra = fft(Gnuc_arraya(k+idx));
    rc = fft(Gnuc_arrayc(k+idx));
    rt = fft(Gnuc_arrayt(k+idx));
    rg = fft(Gnuc_arrayg(k+idx));

    G_final_result(1,k)=abs(ra(FREQ));
    G_final_result(2,k)=abs(rc(FREQ));
    G_final_result(3,k)=abs(rt(FREQ));
G_final_result(4,k)=abs(rg(FREQ));
gend
%% Compile data and bring it back to the CPU memory
FREQ_DATA = single(sum(G_final_result.^2));
```

```
% Record analysis processing time
TIME = toc;

% Clear storage arrays created in CPU memory
clear nuc_arrays;

%% Cleaning GPU memory
clear gpu_hook ;
end
```

Annexe 3

Analyse en fréquence sans GPU

```
%% Procedure pour Analyse en frequence d'une sequence FASTA
% NE PAS OUBLIER MATLAPBOOL
%% RESET
close all
clear all
clc
```

```
%% Choisir le fichier
filename = 'Chr1_144_147M.fasta'
%% Lire le fichier FASTA
SEQUENCE_pour_gzl = load_fasta_for_gzl(filename);
%% Analyse en frequence goertzel
tic
[GZL_DATA(1).DATA, TIME ] = my_gzl_raw( SEQUENCE_pour_gzl ,351);
GZL_DATA(1).WIN = 351;
toc
```

```
%%
tic
[GZL_DATA(2).DATA, TIME ] = my_gzl_raw( SEQUENCE_pour_gzl ,900);
GZL_DATA(2).WIN = 900;
toc
```

```
%%
tic
[GZL_DATA(3).DATA, TIME ] = my_gzl_raw( SEQUENCE_pour_gzl ,1500);
GZL_DATA(3).WIN = 1500;
toc

%%
tic
```

```
[GZL_DATA(4).DATA, TIME ] = my_gzl_raw( SEQUENCE_pour_gzl ,3000);
SEQUENCE_pour_gzl ,3000;
toc
```

```
%% Fonction my_gzl_raw.m
function [PROFILE_DATA, TIME ] = my_gzl_raw( donnee ,fenetre)
%MY_GZL Summary of this function goes here
%   Detailed explanation goes here

%   Detailed explanation goes here
tic
% Find sequence_size
seq_size = max(size(donnee));

% Create four input vectors
% Vecteur A
vecteuracomplet = false(seq_size,1); % False est l'ecriture en steno pour 'logical[0]'
vecteuracomplet(find(donnee == 'A')) = true;% True est l'ecriture en steno pour 'logical[1]'
vecteuracomplet = double(vecteuracomplet);

% Vecteur C
vecteurccomplet = false(seq_size,1);
vecteurccomplet(find(donnee == 'C')) = true;
vecteurccomplet = double(vecteurccomplet);

% Vecteur T
vecteurtcomplet = false(seq_size,1);
vecteurtcomplet(find(donnee == 'T')) = true; % find: Localise tout les ï¿½lï¿½ment non
zï¿½ro.
vecteurtcomplet = double(vecteurtcomplet);

% Vecteur G
vecteurgcomplet = false(seq_size,1);
vecteurgcomplet(find(donnee == 'G')) = true;
vecteurgcomplet = double(vecteurgcomplet);

% Pre-allocate memory for results
PROFILE_DATA = zeros(seq_size-fenetre,1);

% One column means you need to compute a sum every iteration. this is
```

```
% slower than storing 4 times the results, but consumes more memory. On
% large sequences, it is better to go for this approach.

parfor i=1:seq_size-fenetre
   vecteura=vecteuracomplet(i:i+fenetre-1);
   resultata=(my_gzl(vecteura,fenetre));

   vecteurc=vecteurccomplet(i:i+fenetre-1);
   resultatc=(my_gzl(vecteurc,fenetre));

   vecteurt=vecteurtcomplet(i:i+fenetre-1);
   resultatt=(my_gzl(vecteurt,fenetre));

   vecteurg=vecteurgcomplet(i:i+fenetre-1);
   resultatg=(my_gzl(vecteurg,fenetre));

  PROFILE_DATA(i)=sum([resultata;resultatt;resultatg;resultatc]);
end
TIME = toc;
clear v*;
% END OF FUNCTION
end
```

```
%% Fonction my_gzl
% my_gzl
function [ R ] = my_gzl( SEQUENCE ,fenetre)
%MY_GZL Summary of this function goes here
%   Detailed explanation goes here
RC=0;
R1=0;
R2=0;
        for k=1:fenetre
        RC = SEQUENCE(k)-R1-R2;
        R2 = R1;
R1 = RC;
end
R = R1^2+R2^2+R1*R2;
% #codegen
```

Annexe 4

Procédure pour l'analyse d'hexamères d'une séquence d'ADN

```
%% Reinitialiser
close all
clear all
clc
```

%% Importe les sequences d'ADN de longueur variable en format FASTA % Importe la sequences d'ADN de longueur variable du gene HFE2 (Voir NCBI % procédure)position du gene sur le chromosome_1: 145413191pb a 145517545pb

```
SEQUENCE=fastaread('Chr1_144_147M.fasta');
% Importe la sequence d'ADN du gene HFE2 à partir du numero d'ascension de NCBI.
HFE2 = getgenbank('NG_011568');
Hfe2_lines=getExonsPlotLine(HFE2);
% Superpose le gène HFE2 (11355 pb) sur la séquence d'analyse et indique la position des
4 exons de HFE2 sur la figure
hfe2_lines_axe_x = 145408191:145408191+11354;
% Specifier axe des x
axe_x = 144000000:147000000;

% Obtenir la liste des genes de martview
%getGeneList144_147
% Obtenir la liste depuis le fichier FLAT modifie avec juste les indices de
% depart et d'arrive
```

%% Importer le fichier FLAT de sylvain

```
% Ceci remplace la ligne de code ancienne:
% GENES_X = importdata('FLAT_liste_pour_import.flat')'+144000000;fid = fopen('FLAT_LISTE_NOM_COMPLETE.flat');% fileID = fopen(nom du fichier)  % ouvre le fichier, nom du fichier, l'accès en lecture binaire% et retourne un identifiant de fichier de nombre entier  % égal ou supérieur à 3.  Matlab fichier réserves identificateurs 0,1 et 2 pour % l'entrée standard, la sortie standard(l'écran) et l'erreur standard
% respectivement.
RAW_FLAT_IN = textscan(fid,'%d %d %*s'); % %d: entier signé base 10,
```

```
% %s:chaîne de caratères. lire une série de caractères, jusqu'à trouver un
% espace blanc.
% GENES_X = cell2mat(RAW_FLAT_IN)converti les cellules de même type de
% données en un seul module
GENES_X = cell2mat(RAW_FLAT_IN)'+144000000;

clear RAW_FLAT_IN;

fclose(fid);% fclose(fid)ferme un fichier ouvert. fidest un indicateur de fichier entier obtenu par la fonction fopen.

%%


%GENES_X = [GENES_X ; [125;126] ; [125;126]]

GENES_Y = [1 1];

%%  Analyse des hexamers de HFE2 avec differentes source de ponderation dont la fenêtre du glissement est variable (Zhang,M.Q. 1998,vol7, No. 5,919=932, HMG).

% La longueur de la sequence d'analyse est de 1000000 nt.

% Importe les fichiers de ponderations d'excel de l'article

Ponderation_Article_richeGC = importdata('Ponderation_Article_richeGC.xlsx');

Ponderation_Article_pauvreGC = importdata('Ponderation_Article_pauvreGC.xlsx');

%% Analyse des hexamers avec une fenêtre de 500 pb d'une sequence fasta entre 145000000 et 146000000 pb


%%
tic
Fenetre_tmp = 351;
Analyse_Hexa(1).Nom = 'Chr1 HFE2 144 a 147M Ponderation Riche GC';

Analyse_Hexa(1).Fenetre = Fenetre_tmp;

Analyse_Hexa(1).Res =
HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_richeGC.Sheet1',Fenetre_tmp,F
enetre_tmp)

toc
```

```
%%
tic
Fenetre_tmp = 351;
```

```
Analyse_Hexa(2).Nom = 'Chr1 HFE2 144 a 147M Ponderation Pauvre GC';
Analyse_Hexa(2).Fenetre = Fenetre_tmp;
Analyse_Hexa(2).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_pauvreGC.Sheet1',Fenetre_tmp,Fenetre_tmp);
toc
```

```
%%
tic
Fenetre_tmp = 900;
Analyse_Hexa(7).Nom = 'Chr1 HFE2 144 a 147M Ponderation Riche GC';
Analyse_Hexa(7).Fenetre = Fenetre_tmp;
Analyse_Hexa(7).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_richeGC.Sheet1',Fenetre_tmp,Fenetre_tmp)
Toc
```

```
%%
tic
Fenetre_tmp = 900;
Analyse_Hexa(8).Nom = 'Chr1 HFE2 144 a 147M Ponderation Pauvre GC';
Analyse_Hexa(8).Fenetre = Fenetre_tmp;
Analyse_Hexa(8).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_pauvreGC.Sheet1',Fenetre_tmp,Fenetre_tmp);
toc
```

```
%%
tic
Fenetre_tmp = 1500;
Analyse_Hexa(3).Nom Chr1 H= 'FE2 144 a 147M Ponderation Riche GC';
Analyse_Hexa(3).Fenetre = Fenetre_tmp;
Analyse_Hexa(3).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_richeGC.Sheet1',Fenetre_tmp,Fenetre_tmp)
```

```
toc
```

```
%%
tic
Fenetre_tmp = 1500;
Analyse_Hexa(4).Nom = 'Chr1 HFE2 144 a 147M Ponderation Pauvre GC';
Analyse_Hexa(4).Fenetre = Fenetre_tmp;
Analyse_Hexa(4).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_pauvreGC.Sheet1',Fenetre_tmp,Fenetre_tmp);
toc
```

```
%%
tic
Fenetre_tmp = 3000;
Analyse_Hexa(5).Nom = 'Chr1 HFE2 144 a 147M Ponderation Riche GC';
Analyse_Hexa(5).Fenetre = Fenetre_tmp;
Analyse_Hexa(5).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_richeGC.Sheet1',Fenetre_tmp,Fenetre_tmp)
toc
```

```
%%
tic
Fenetre_tmp = 3000;
Analyse_Hexa(6).Nom = 'Chr1 HFE2 144 a 147M Ponderation Pauvre GC';
Analyse_Hexa(6).Fenetre = Fenetre_tmp;
Analyse_Hexa(6).Res = HexamerCoulis(SEQUENCE.Sequence,Ponderation_Article_pauvreGC.Sheet1',Fenetre_tmp,Fenetre_tmp);
toc
%% Sauvegarder les axes des X
clear Fenetre_tmp
save Hexamer
```

```
function [ hexam_scores ] = HexamerCoulis (seq_to_analyse, hexam_score_mat, min_wsz, max_wsz)
%  HexamerCoulis Analyse des Hexamer par Fenetre coulissante
%   On lui donne une sequence, et la matrice des scores de chaque hexamer
%   (4096 valeurs), et la fonction va retourner un tableau de resultats.
%
%   Chaque colonne correspond une taille de fenetre.
%   Chaque ligne correspond la fenetre pour un nucleotide dans la sequence.
%
%   ON PEUT CHANGER LES TAILLES DE FENETRES DANS LE FICHIER HexamerCoulis.m
%% Convertir la seq en upper
seq_to_analyse = upper(seq_to_analyse);
```

```
%% Creer la Hash Table des Hexamer

for x=0:4095
hexa_list{x+1} = base42seq(x);
end
HexaMap = containers.Map(hexa_list,0:4095);
```

```
%% Definir la matrice des scores
% hexam_score_mat = 1:4096;
%% Definir la sequence a analyser
% seq_to_analyse = 'AAAAAAGGGGGG';
% Taille de la sequence
seq_to_analyse_sz = size(seq_to_analyse,2);
% ------------------------------------------------------
% ICI POUR CHANGER LES TAILLES DE FENETRE A TRAITER!!!
% ------------------------------------------------------
% Min et max de la fenetre coulissante
% min_wsz = 6;
% max_wsz = 2000;

% Si le max size de la sequence est plus petit que max_wsz, on prend cette
% valeur
If          ( seq_to_analyse_sz < max_wsz )
max_wsz = seq_to_analyse_sz;
end
disp('------------------------------------------------------')
disp(['Largeur de fenetre minimale utilisee: ' num2str(min_wsz)])
```

```
disp(['Largeur de fenetre maximale utilisee: ' num2str(max_wsz)])
disp(['Taille de la sequence: ' num2str(seq_to_analyse_sz)]) % Liste de toutes les fenetre à traiter windows_sizes = min_wsz:max_wsz;
num_of_wins = size(windows_sizes,2);
% Preallouer la matrice des resultats
% les colonnes sont les résultats pour une fenetre
hexam_scores = zeros(seq_to_analyse_sz,num_of_wins);

% Pour chaque fenetre
for       cur_win = 1:num_of_wins
% Trouver la grosseur de fenetre actuelle
cur_win_sz = windows_sizes(cur_win);
%  Pour chaque nucleotide
Parfor cur_nuc = 1:(seq_to_analyse_SZ+1 = cur_win_sz
% Trouver le total d'hexamer dans la fenetre
total_hexams_inwin = hexamsINseq2(seq_to_analyse(cur_nuc:cur_nuc+cur_win_sz-1),HexaMap);
% Multiplier avec la matrice des scores des hexamers
 cur_score = total_hexams_inwin*hexam_score_mat';

% Store le score total dans la liste de cette fenetre (pour le
% nucleotide actif)

hexam_scores(cur_nuc,cur_win) = cur_score;
end
end
% Fin fonction
end
```

```
% Liste de seuil et faux positifs
Theorie_1=(Analyse_Hexa(2).Res);
% Total des genes:
TOTAL_GENES = length(GENES_X)
% Position de x sur la sequence
% Liste des Seuils
Liste_seuils = 0:10:80;
% Genes qui contiennent des valeurs au dessus du seuil
gene_detectes = zeros(TOTAL_GENES,length(Liste_seuils));
seuil_actif = 1;
% Variables pour les faux positifs
total_faux_positifs = zeros(1,length(Liste_seuils));
seuil_flag = 0; % Pour ne pas détecter des gènes en double, etc.
for       Seuil = Liste_seuils
```

```
        Seuil
        for     x=1:length(Theorie_1)
    position=144000000+x-1;
    % Exemple pour le premier gene de la liste
    % [144000001;]Début de séquence du gène.
        % [144094427;]Fin de séquence du gène
        for gene_courant = 1:TOTAL_GENES
        if (position >= GENES_X(1,gene_courant) & position <= GENES_X(2, gene_courant))
        % On est à l'interieur d'un gene
                        % et notre courbe depasse notre seuil
                        if      (Theorie_1(x) > Seuil)
                                % gene_courant
                                 % position
                                % Theorie_1(x)
                                % Mettre le gene comme étant détecté
                                gene_detectes(gene_courant,seuil_actif) = 1;
     end        % if de si on depasse le seuil
else
% Je ne suis pas à l'intérieur d'un gène
                        % Si je depasse mon seuil
        if (Theorie_1(x) > Seuil)
        % Si mon flag est à 0 (pas de gene detecte precedemment)
        if (seuil_flag == 0)
     % Compte un gene
    total_faux_positifs(seuil_actif) = total_faux_positifs(seuil_actif)+1;
    % Mettre le flag à 1
     seuil_flag = 1;
    end
    else
```

```
    % Si je ne depasse PAS mon seuil
    % Mettre le flag à 0
    seuil_flag = 0;
     end % fin de depasse seuil

     end % if de etre un gene ou non

     end % for de chaque gene

end % for des positions
seuil_actif = seuil_actif+1;
end % for seuil

% Affichage des resultats
[Liste_seuils ; sum(gene_detectes) ; total_faux_positifs]
```

```
%% Base 42seq
function [ my_seq ] = base42seq(my_num,padding)
% BASE42SEQ Converts a decimal number to base 4, then to Letters
%       BASE42SEQ(N,P) returns the sequence in LETTERS for a decimal number N.
%       It will pad with the letter A to return a number as big as P.
%
%  BASE42SEQ(N) returns the sequence with a default padding of 6.
%
%  Reminder: A T G C are 0 1 2 3 in base 4
%
%  Examples
%     base42seq(4095,6) returns 'CCCCCC'
%     base42seq(4095,7) returns 'ACCCCCC'

% Check number of args
narginchk(1, 2);
% If no padding specified, default to 6
     if nargin == 1

padding = 6;
end
the_seq = dec2base(my_num,4);
my_seq = strrep(the_seq,'0','A');
my_seq = strrep(my_seq,'1','T');
my_seq = strrep(my_seq,'2','G');
```

```
my_seq = strrep(my_seq,'3','C');

seq_size = size(my_seq,2);
if          seq_size < padding
           pad_seq = char(65*ones(1,padding));
           pad_seq(end-(seq_size-1):end) = my_seq;
my_seq = pad_seq;
end
```

Annexe 4

Transformée de Fourier discrète.

$$F(k)=\sum_{n=0}^{N-1} f(n)*e^{-i\left(\frac{2\pi}{N}\right)nk}$$

Pour $0 \leq k < N-1$

Dans la formule de TFD nous avons la représentation du vecteur par *f*(n) et la matrice par le nombre complexe.

Exemple de calcul :

Pour k= 0, 1, 2, …, N-1

$$F(0)=\sum_{n=0}^{N-1} f(n)e^{-i\left(\frac{2\pi}{N}\right)n*0}=\sum_{n=0}^{N-1} f(0)$$

$$F(0)=f(0)*1+f(1)*1+f(2)*1+\ldots+f(N-1)*1$$

$$F(1)=\sum_{n=0}^{N-1} f(n)e^{-i\left(\frac{2\pi}{N}\right)n*1}$$

$$F(1)=f(0)e^{-i\left(\frac{2\pi}{N}\right)0*1}+f(1)e^{-i\left(\frac{2\pi}{N}\right)1*1}+f(2)e^{-i\left(\frac{2\pi}{N}\right)2*1}+\ldots+f(N-1)e^{-i\left(\frac{2\pi}{N}\right)(N-1)*1}$$

$$F(2)=\sum_{n=0}^{N-1} f(n)e^{i\left(\frac{2\pi}{N}\right)n*2}$$

$$F(2)=f(0)e^{-i\left(\frac{2\pi}{N}\right)0*2}+f(1)e^{-i\left(\frac{2\pi}{N}\right)1*2}+f(2)e^{-i\left(\frac{2\pi}{N}\right)2*2}+\ldots+f(N-1)e^{-i\left(\frac{2\pi}{N}\right)(N-1)*2}$$

$$F(N-1)=\sum_{n=0}^{N-1} f(n)e^{i\left(\frac{2\pi}{N}\right)n*(N-1)}$$

$$F(N-1)=f(0)e^{-i\left(\frac{2\pi}{N}\right)0*(N-1)}+f(1)e^{-i\left(\frac{2\pi}{N}\right)1*(N-1)}+f(2)e^{-i\left(\frac{2\pi}{N}\right)2*(N-1)}+\ldots+f(N-1)e^{-i\left(\frac{2\pi}{N}\right)(N-1)*(N-1)}$$

$$\begin{pmatrix} 1_{11} & 1_{12} & \cdots & 1_{1n} \\ 1_{21} & e^{-i\left(\frac{2\pi}{N}\right)1*1}{}_{22} & \cdots & e^{-i\left(\frac{2\pi}{N}\right)(N-1)*1}{}_{2n} \\ 1_{31} & e^{-i\left(\frac{2\pi}{N}\right)1*2}{}_{32} & \cdots & e^{-i\left(\frac{2\pi}{N}\right)(N-1)*2}{}_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ 1_{m} & e^{-i\left(\frac{2\pi}{N}\right)(N-1)}{}_{2m} & \cdots & e^{-i\left(\frac{2\pi}{N}\right)(\wedge N-1)*(N-1))}{}_{nm} \end{pmatrix} * \begin{pmatrix} f(0) \\ f(1) \\ f(2) \\ \vdots \\ f(N-1) \end{pmatrix} = \begin{pmatrix} F(0) \\ F(1) \\ F(2) \\ \vdots \\ F(N-1) \end{pmatrix}$$

L'algorithme de calcul de Transformée de Fourier Rapide (TFR) ou en anglais Fast Fourier Transform (FFT), fut développé par James Cooley, du centre de recherche Thomas Watson d'IBM, et John Tukey, des laboratoires Bell Téléphone en 1965. Un des grands avantages de la FFT est qu'elle réduit de façon considérable le nombre de multiplications. La FFT permet de ramener le calcul de la transformée de Fourier discrète de T^2 à $T\log_2 T$ opérations ; cette réduction de complexité suffit à faire passer d'impossibles à facilement résolubles nombre de problèmes. Une dérivation de la transformée de Fourier rapide permet de séparer en deux parties la séquence d'entrée.

$$F(k) = \sum_{n=0}^{N-1} f(n) * e^{-i\left(\frac{2\pi}{N}\right)nk}$$

Nous utiliserons la récursivité en montrant que le calcul d'une transformée de Fourier de taille T se ramène aux calculs de deux Transformées de Fourier de taille T/2 suivi de T/2 multiplications.

Posons n=2n si n est paire et n=2n+1 si n est impaire. F(k), s'écrit alors, en posant N=T/2.

$$F(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n)e^{-i2\pi(2n)k/N} + \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1)e^{-i2\pi(2n+1)k/N}$$

$$F(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n)e^{-i2\pi(2n)k/N} + \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1)e^{-i2\pi(2n)k/N} * e^{-i2\pi k/N}$$

$$F(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n)e^{-i2\pi(2n)k/N} + e^{-i2\pi k/N} \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1)e^{-i2\pi(2n)k/N}$$

Posons $W_N = e^{-i\left(\frac{2\pi}{N}\right)}$ que l'on appelle Facteur de phase angulaire.

Alors l'équation devient :

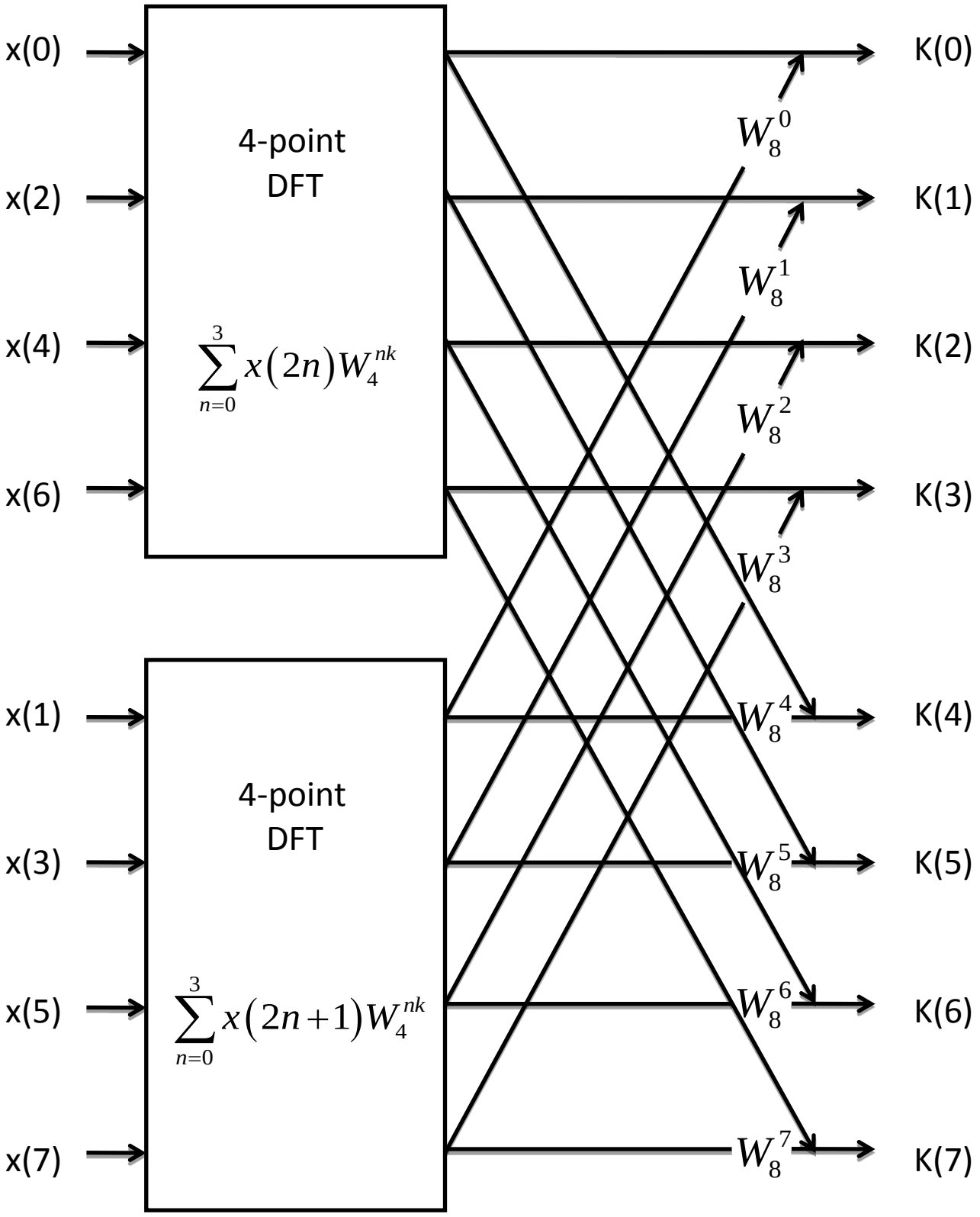
$$F(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n) * W_N^{2nk} + W_N^k \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1) W_N^{2nk}$$

Propriété de l'exponentiel :

$$W_N^2 = e^{-i\left(\frac{2\pi}{N}\right)*2} = e^{-i\frac{2\pi}{\frac{N}{2}}} = W_{\frac{N}{2}}$$

$$F(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n) * W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1) * W_{\frac{N}{2}}^{nk}$$

Implémentation d'une TFR (FFT), TFD (DTF)
8-point utilisant deux TFDs à 4-point



Annexe 5

Pauvre G+C		Riche G+C	
Hexamères	**Pondération**	**Hexamères**	**Pondération**
TTTTTT	-45.0000	TTTTTT	-25.3000
AAAAAA	-26.5000	AAAAAA	-18.9000
ATTTTT	-18.0000	CACACA	-12.5000
TTTTTA	-16.3000	GGGAGG	-11.8000
TATTTT	-15.0000	CCTCCC	-11.1000
TTTTAA	-14.0000	ACACAC	-9.6000
TTATTT	-13.0000	ATTTTT	-9.3000
TTTAAA	-13.0000	AGGCTG	-8.2000
TTTATT	-13.0000	GGAGGG	-8.2000
TTTTCT	-13.0000	GGTGGG	-7.9000
TTAAAA	-13.0000	AGGGAG	-7.9000
TTTCTT	-12.0000	GGGTGG	-7.9000
TTTTAT	-12.0000	CTCCCA	-7.5000
TGTTTT	-12.0000	CAGCCT	-7.5000
AAAAAT	-12.0000	CTGGGA	-7.1000
TTCTTT	-12.0000	GAGGGG	-6.8000
CTTTTT	-11.0000	TGGGTG	-6.4000
TTTTTG	-11.0000	GCTGGG	-6.4000
TAAAAA	-11.0000	GGGTGT	-6.4000
AAAATA	-11.0000	GGGGTG	-6.4000
GAAGGA	6.0000	TGAAGA	5.0000
GAGAAG	6.0000	CAGCTG	5.0000
TGGAAG	6.0000	GTGGAC	5.0000
CAGCTG	6.5000	CTGAAG	6.1000
GATGAA	6.5000	CGAGGA	6.1000
GGAAGA	6.5000	CTGGAC	6.4000
AAGGAG	6.5000	ACCTGG	6.4000
CTGGAG	7.0000	TGCTGC	6.4000
TCCTGG	7.0000	GGAGGA	6.4000
GGAGAA	7.0000	GAGCTG	7.1428
CAGAAG	7.0000	AAGAAG	7.8750
CTGGAA	7.5000	CTGCTG	7.8750
ATGAAG	7.5000	GCTGGA	8.5710
TGATGA	7.5000	GACCTG	8.5710
TGGAGA	11.5000	TGGAGA	8.5710
CCTGGA	11.5000	CTGGTG	9.2850
AGAAGA	11.5000	CTGCAG	9.2850
TGAAGA	12.0000	GAGGAG	9.3846
AAGAAG	16.0000	CTGGAG	11.4000
GAAGAA	18.0000	CCTGGA	13.6000

Annexe 6

Droits d'auteurs

ROYAL MILITARY COLLEGE OF CANADA • COLLÈGE MILITAIRE ROYAL DU CANADA
PO Box 17000, Station Forces • CP 17000, Succursale Forces • Kingston, Ontario • K7K 7B4

2 juin 2015

De la part de : Dr Rachid Beguenane
Department of Electrical and Computer Engineering
Royal Military College of Canada

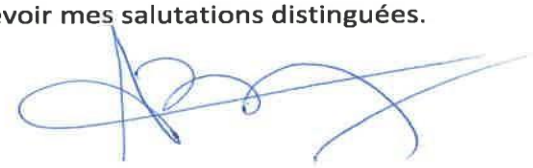
Objet : Autorisation pour inclure dans son mémoire de maîtrise un article scientifique publié par Sylvain Robert Rivard et moi-même

Mesdames, Messieurs,

La présente est pour confirmer que j'autorise M. Sylvain Robert Rivard à utiliser l'article scientifique suivant comme composante de son mémoire de maîtrise en ingénierie.

> Sylvain Robert Rivard, Jean-Gabriel Mailloux, **Rachid Beguenane** and Hung Tien Bui (2012). Design of high-performance parallelized gene prediction in MATLAB. BMC Research Notes, 5 : 183-192.

Veuillez recevoir mes salutations distinguées.



Rachid Beguenane ing., Ph. D.
Associate Professor
Department of Electrical and Computer Engineering
Royal Military College of Canada
Box 17000, Station Forces
Kingston, Ontario, K7K 7B4

UQAC
UNIVERSITÉ DU QUÉBEC À CHICOUTIMI
555, boulevard de l'Université
Chicoutimi (Québec) CANADA G7H 2B1

Chicoutimi, 3 décembre 2015

De la part de : Jean-Gabriel Mailloux
Université du Québec à Chicoutimi

Objet : Autorisation pour inclure dans son mémoire de maîtrise un article scientifique publié par Sylvain Robert Rivard et moi-même

---

Mesdames, Messieurs,

La présente est pour confirmer que j'autorise M. Sylvain Robert Rivard à utiliser l'article scientifique suivant comme composante de son mémoire de maîtrise en ingénierie.

> Sylvain Robert Rivard, Jean-Gabriel Mailloux, Rachid Beguenane and Hung Tien Bui (2012). Design of high-performance parallelized gene prediction in MATLAB. BMC Research Notes, 5 : 183-192.

Veuillez recevoir mes salutations distinguées.

Jean-Gabriel Mailloux
Étudiant Ph.D. UQAC

UQAC
UNIVERSITÉ DU QUÉBEC À CHICOUTIMI
555, boulevard de l'Université
Chicoutimi (Québec) CANADA G7H 2B1

Chicoutimi, 3 décembre 2015

De la part de : Dr Hung Tien Bui

Objet : Autorisation pour inclure dans son mémoire de maîtrise un article scientifique publié par Sylvain Robert Rivard et moi-même

---

Mesdames, Messieurs,

La présente est pour confirmer que j'autorise M. Sylvain Robert Rivard à utiliser l'article scientifique suivant comme composante de son mémoire de maîtrise en ingénierie.

> Sylvain Robert Rivard, Jean-Gabriel Mailloux, Rachid Beguenane and Hung Tien Bui (2012). Design of high-performance parallelized gene prediction in MATLAB. BMC Research Notes, 5 : 183-192.

Veuillez recevoir mes salutations distinguées.

Hung Tien Bui ing., Ph. D.
Prof. agrégé

Annexe 7

Publication acceptée avec comité de lecture

Rivard *et al. BMC Research Notes* 2012, **5**:183
http://www.biomedcentral.com/1756-0500/5/183

BMC Research Notes

TECHNICAL NOTE Open Access

Design of high-performance parallelized gene predictors in MATLAB

Sylvain Robert Rivard[1*†], Jean-Gabriel Mailloux[1†], Rachid Beguenane[2] and Hung Tien Bui[1]

Abstract

Background: This paper proposes a method of implementing parallel gene prediction algorithms in MATLAB. The proposed designs are based on either Goertzel's algorithm or on FFTs and have been implemented using varying amounts of parallelism on a central processing unit (CPU) and on a graphics processing unit (GPU).

Findings: Results show that an implementation using a straightforward approach can require over 4.5 h to process 15 million base pairs (bps) whereas a properly designed one could perform the same task in less than five minutes. In the best case, a GPU implementation can yield these results in 57 s.

Conclusions: The present work shows how parallelism can be used in MATLAB for gene prediction in very large DNA sequences to produce results that are over 270 times faster than a conventional approach. This is significant as MATLAB is typically overlooked due to its apparent slow processing time even though it offers a convenient environment for bioinformatics. From a practical standpoint, this work proposes two strategies for accelerating genome data processing which rely on different parallelization mechanisms. Using a CPU, the work shows that direct access to the MEX function increases execution speed and that the PARFOR construct should be used in order to take full advantage of the parallelizable Goertzel implementation. When the target is a GPU, the work shows that data needs to be segmented into manageable sizes within the GFOR construct before processing in order to minimize execution time.

Findings

Background

Since the early beginnings of the human genome project, numerous research groups have developed computerized approaches to studying human genetics [1,2]. In order to deal with the high volume of accumulated biological data, geneticists and molecular biologists require faster algorithms. This is important as the amount of data in gene research roughly doubles every six months whereas computer processing speeds improve at a much lower rate. Thus, speed increases due to hardware improvement alone cannot keep up with the amount of data to process, and therefore, optimization of the existing tools is required.

In bioinformatics, one of the main interests is studying deoxyribonucleic acid (DNA) sequences, which are of fundamental importance in understanding all living species. DNA molecules are at the base of hereditary information and are composed of four types of nucleotides: adenine (A), thymine (T), guanine (G) and cytosine (C). Certain segments of the DNA strand contain the necessary information for protein synthesis and these are called genes. Genes include two sub-sections called coding regions (exons) and non-coding regions (introns). As of the time of publication, it is estimated that several thousands of all human genes have yet to be discovered [3].

In general, bioinformatics applications need to process massive quantities of data.

In the near future, it is the authors' belief that certain applications will require the processing of a large number of complete genomes. This can occur, for instance, in the case of comparative analyses of individual genomes over an entire population. To facilitate this process and to encourage geneticists to use this promising computerized venue, it is necessary to develop tools with the following characteristics [4]:

- Easy to install locally;
- ability to train and test the programs independently;

* Correspondence: sylvain-robert.rivard@uqac.ca
†Equal contributors
[1]Département des sciences appliquées, Université du Québec à Chicoutimi, 555 blvd de l'Université, Chicoutimi, QC, G7H 2B1, Canada
Full list of author information is available at the end of the article

- availability of the source code;
- fast processing speeds;
- freedom from excessive licensing restrictions.

The process of discovering genes has traditionally been done in genetics laboratories and is often seen as being long and expensive. Since the beginning of the 21st century, a digitized version of the complete human genome has been made publicly-accessible. It has since been speculated that part of gene discovery could be done digitally using computer algorithms. This task is called gene prediction and involves analyzing a DNA sequence and identifying regions that code for protein. While gene prediction is of great interest, the amount of data that needs to be processed is very large at around 6 billion bps and the required mathematical operations are time consuming. It is therefore important to find techniques that are accessible to geneticists that can process very large amounts of data within a short time frame.

Over the last three decades, digital signal processing (DSP) techniques have been developed for the identification of protein coding regions, both in humans and in others species [5,6]. Fickett [7] was the first to propose the use of an autocorrelation function (ACF) to identify the periodicity within the exons of a genomic sequence. Over the years, other methods have been proposed including frequency analysis using the Fast Fourier Transform (FFT), the autoregressive model and the hidden Markov model [2,4].

The purpose of this work is to offer geneticists and molecular biologists the tools for the prediction or identification of new genes using existing complementary strategies. The objectives of this research are to render this approach fast, reliable, accurate and easy to use, thus requiring little training. Furthermore our approach can work with several types of genomes such as humans, plants or other various organisms.

This work focuses on predicting genes using frequency analysis with FFTs and with an equivalent technique known as Goertzel's algorithm [8]. These methods have a proven detection rate that can surpass 80% [9], and the results in [10] further demonstrate the reliability of this method. Specifically, this work's objective is to develop parallel computing techniques that allow for gene prediction algorithms to run at a higher speed on conventional desktop computers. While there are several existing solutions in the literature [11-13] that yield good results, they are often limited to small sequences of a few thousand bps. We believe that the performances of these previously proposed solutions are not yet adequate since linked markers are often separated by millions of bps [14]. This work specifically targets large sequences and makes sure the processing time remains low in order to apply these techniques in actual day to day genetics research. These results are then validated against a specific gene for which the information is well-known. More importantly, the designs proposed herein are implemented using MATLAB rather than developing custom software and/or hardware solutions such as FPGAs or ASICs. The reason is that gene prediction tools often do not match the researcher's needs. Tailoring existing software/hardware solutions for a specific requirement often involves modifying the design which demands a certain level of expertise. We have witnessed first-hand how time consuming the process of tapping FPGA power [15] can be in this scenario, but most importantly, how inflexible it becomes when the needs of the application change. Since MATLAB is a multi-platform tool with a relatively simple yet powerful programming language that does not require compilation, the difficulties encountered with a custom software/hardware approach are mitigated.

Frequency analysis in gene prediction

One of the crucial steps in gene prediction is the identification of coding regions. It has been well-established that these coding regions have repeating nucleotide sequences that exhibit a periodicity of three [16-18]. To detect this periodicity, and thus to identify coding regions, the typical approach is to perform frequency analysis using the Discrete Fourier Transform (DFT). Since the straightforward implementation of the DFT is not computationally efficient, the FFT is often preferred. The FFT has been extensively covered in literature, and its optimization has been thoroughly explored [13,17,19,20]. A gene prediction algorithm based on the FFT can therefore yield good results.

The FFT is an operation that takes S samples of a time-domain or a space-domain signal and outputs the amplitude and the phase information for a series of S frequencies. While all this information may be required in many applications, only the amplitude of one frequency needs to be considered in gene prediction. This frequency of interest, which corresponds to a periodicity of three in the nucleotide sequence, is equal to a third of the sampling frequency ($f_s/3$). The remaining phase and amplitude information provided by the FFT are discarded. Thus, for applications that only require a small number of frequencies, such as gene prediction, the use of the FFT is not optimal. A more efficient approach is to use the Goertzel algorithm which is given by:

$$y[n] = x[n] + 2\cos(2\pi f_n)y[n-1] - y[n-2] \quad (1)$$

In (1), x is the input, y is the output and n is the sample number. Since gene prediction relies on the detection of a periodicity of three in the sequence, the normalized frequency f_n can be replaced by 1/3. In that case, the coefficient of the *y[n-1]* term becomes −1 and the equation can be rewritten as:

$$y[n] = x[n] - y[n-1] - y[n-2] \quad (2)$$

In this form, the Goertzel algorithm no longer requires multiplications or fractional terms. This allows for a faster and more efficient implementation. It should be noted, however, that MATLAB uses double precision arithmetic when performing certain functions, including the Goertzel algorithm. While the full benefits of having such a simple algorithm are not always apparent, preliminary tests have shown a 28% increase in speed.

After processing N elements using equation (2), the final result is obtained with the following equation, where P_X is the power at frequency $f_s/3$ of nucleotide X :

$$P_X = y_N^2 + y_N^2 + y_N \cdot y_{N-1} \quad (3)$$

This process is done for all values of X (A, T, G and C) and the results are added together to provide the final output.

General implementation

A block diagram of the gene prediction algorithm is shown in Figure 1. It shows that the first step in gene prediction is to separate the DNA sequence into four different vectors containing numerical values. This conversion is what allows for the use of DSP in DNA analysis. While numerous techniques can be used to perform this conversion [21], the one used in this work was proposed by Voss [22]. The approach consists of converting the DNA sequence into four binary vectors, each corresponding to a type of nucleotide. These vectors will contain '1' at a given position if the corresponding nucleotide is of that type. If not, that vector position will be '0'. An example of such conversion is shown in Figure 2.

In gene prediction, frequency analysis is performed on a small window at a time. Within this window, an FFT or Goertzel's algorithm is executed in an attempt to detect a periodicity of three. Once this is completed, the window is shifted by one nucleotide before frequency analysis is performed again. This process continues until the whole DNA sequence has been analyzed. At that point, the results from all four vectors are combined to generate the final output. Figure 3 illustrates this process by showing how an output figure is generated. In such a figure, an area with larger amplitudes indicates a higher probability of containing a coding region.

The choice of window size affects the quality of the results. It has been shown that long window sizes remove specificity from the analysis which makes it more difficult to determine the boundary between an intron and an exon. On the other hand, windows that are too small tend to yield noisy results. Mahmood et al. [23] suggest that a size of 351 bps would provide a good balance between noise and specificity; [10] and [19] also demonstrate why 351 is a solid choice.

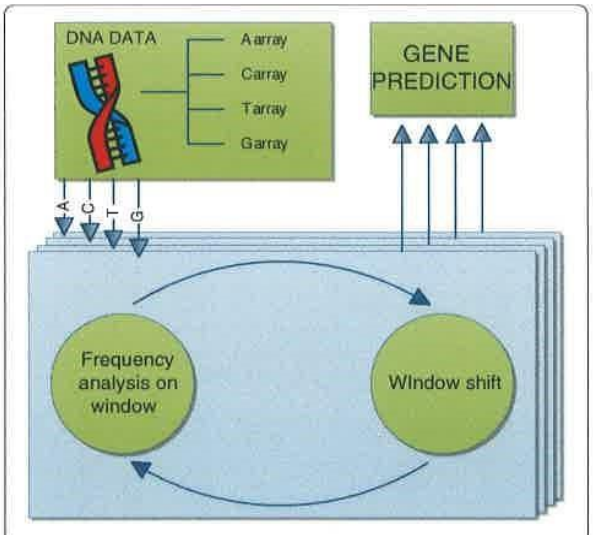


Figure 1 Block diagram of the system.

Design

There are several ways of implementing frequency analysis in MATLAB. They are categorized according to the nature of the algorithm used to perform the frequency analysis and the processing device used to do so. These are described in the following subsections.

Single core implementation

In a typical MATLAB working environment, executed instructions will normally run on a single core. In such an environment, gene prediction can be implemented using any of the following implementations.

Goertzel

The most straightforward approach to implementing the Goertzel algorithm is to use the built-in function *(goertzel)* provided by MATLAB. The function is stored in an .m file (goertzel.m) which contains validation commands as well as a call to a highly-optimized pre-compiled function called goertzelMEX.

SEQUENCE:	A	T	C	T	G	G	A
$x_A(n)$	1	0	0	0	0	0	1
$x_C(n)$	0	0	1	0	0	0	0
$x_G(n)$	0	0	0	0	1	1	0
$x_T(n)$	0	1	0	1	0	0	0

Figure 2 DNA sequence converted into its binary counterparts.

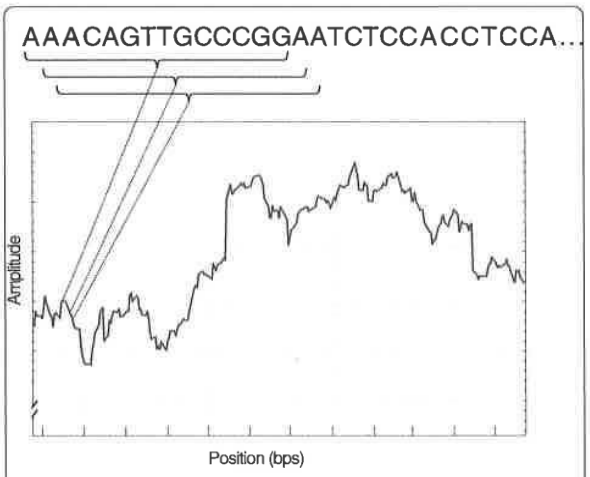


Figure 3 Illustration of the sliding window approach.

By profiling the goertzel.m function, it was discovered that a large amount of time is spent performing validation commands instead of the Goertzel algorithm itself. In an effort to improve processing time, it is possible to bypass the validation commands and execute the goertzelMEX function directly. To do so, the data sent to the goertzelMEX function need to be formatted properly since the validation process is no longer performed. Given the correct data format, the results produced are the same as with the goertzel.m file while the execution time is significantly reduced.

While the goertzel.m and goertzelMEX functions have been provided with MATLAB, the algorithm can be greatly simplified when used in gene prediction. In such case, it is possible to implement the algorithm directly using equations (2) and (3) which is expected to yield better results.

FFT

In addition to the Goertzel algorithm, MATLAB also provides an FFT function (*fft*). Since the hardware implementation and optimization of the FFT have been the topic of many papers, it is expected that a parallelized implementation would yield good results.

Multi-core implementation

In recent years, it has become common for personal computers to be equipped with multiple cores that can often handle a higher number of threads. In order to make use of these cores, it is possible to instruct MATLAB to use more than one processor when the algorithm is adapted for such parallelism.

In a typical gene prediction algorithm, frequency analysis is sequentially performed on a window of data before moving to the next one. Since each window is independent, frequency analysis can be performed on many windows in parallel, if the resources are available. MATLAB allows for these processes to run in parallel on different threads/cores using the PARFOR construct. PARFOR can be used with any of the previously mentioned implementations.

GPU implementation

Many modern computers are equipped with graphics cards that contain one or more GPU. Each GPU contains a large number of streaming processors which can be used for parallel processing. While these cards have historically been developed for video processing, they can now be used to accelerate calculations in MATLAB. To access the processing power of these cards within MATLAB, several toolboxes are available including AccelerEyes' JACKET and the parallel toolbox from MATLAB R2011b. Prior versions of MATLAB provided basic GPU functions, but were too limited to be of interest. For instance, it was not possible to index an array stored in GPU memory. It should be noted that all these technologies call upon NVIDIA's CUDA technology [25] and are not available on Open Computing langage (OpenCL)-only cards.

Matlab R2011b

MATLAB's parallel toolbox offers several commands that allow for algorithms to be executed on a GPU. Unfortunately, there are no commands that are equivalent to PARFOR that can execute multiple FOR loops in parallel on the GPU. While the command *arrayfun* does offer similar functionality, it removes some of the flexibility needed for this particular algorithm given our need for processing a sliding window within an array. This method was therefore not considered further.

JACKET

JACKET is a commercial package that allows MATLAB algorithms to execute parts of their code on GPU. One of the main advantages of using JACKET is the provided GPU counterpart to the PARFOR construct (GFOR). It is therefore a simple task to adapt the MATLAB code to make use of the GPU.

JACKET offers two methods to accelerate the current gene prediction scheme in MATLAB. The first method is to access the FFT function that is implemented on the GPU. The second method, which can be used in conjunction with the first one, is to parallelize the loops using the GFOR command. GFOR, similarly to the PARFOR command, helps to parallelize the operations by distributing them over GPU streaming processors. While this construct does have some limitations (found in JACKET's documentation), they are not prohibitive for the current algorithm.

GPU implementation analysis

To make use of the GPU, the chosen DNA sequence must first be split into DNA blocks. These DNA blocks are then sent in sequence to the GPU for processing. When the DNA block is on the GPU, it is broken down further into DNA fragments in order to be processed by separate parallel GFOR instances. This process is illustrated in Figure 4. The difficulty encountered in this implementation revolves around finding the optimal number of GFORs to run in parallel for a given sequence. While a large number of GFORs may improve parallelism, it also consumes the processing resources on the GPU. When the resources are depleted, JACKET's maintenance algorithm is called to rectify the situation. This process slows down the execution of the algorithm and should be avoided when possible. The goal is therefore to maximize the number of GFORs without depleting the available resources.

Through preliminary tests, it was found that the FFT implementation on the GPU performs much better than any of the GPU Goertzel implementations (Goertzel using the CPU, on the other hand, is fast and is easier to implement). This is expected as FFTs are algorithms that can be parallelized in a straightforward manner whereas the Goertzel algorithm is a recursive one and therefore, does not lend itself easily to parallelization. The GPU implementations will thus only focus on the FFT. Processing times when using Goertzel on a GPU will nonetheless be provided for the purpose of comparison.

A GPU card only has a limited number of parallel processors (PU_{TOTAL}). Knowing that the GPU FFT function requires these resources, the maximum number of GFOR loops (N_{GFOR1}) is given by:

$$N_{GFOR1} \leq \frac{PU_{TOTAL}}{4 \times (PU_{FFT} + PU_{MISC})} \quad (4)$$

The equation states that four GPU implementations of the FFT need to be present during each GFOR loop to account for each of the nucleotide arrays. The implementation requires PU_{FFT} processing units for the FFT itself and requires extra processing units (PU_{MISC}) for miscellaneous operations such as the power of two and the summing of the power spectra.

As previously stated, this equation needs to be respected so that the number of processing units does not exceed the available resources. Otherwise, JACKET will reallocate the GPU resources which slows down the execution of the algorithm.

In addition to the processing unit limitation, the implementation also needs to consider the memory constraints. Since every iteration requires a certain amount of GPU memory, the total amount of GPU RAM available also limits the maximum number of GFORs. This constraint can be summarized as follows:

$$N_{GFOR2} \leq \frac{MEM_{TOTAL}}{4 \times (MEM_{FFT} + MEM_{MISC}) + MEM_{SETUP}} \quad (5)$$

This equation is similar to equation (4) with the exception that it also includes a MEM_{SETUP} term which accounts for the memory required to manage each different GFOR in order to combine them at the end. The maximum number of GFORs is given by the minimum between N_{GFOR1} and N_{GFOR2}:

$$N_{GFOR} = \min(N_{GFOR1}, N_{GFOR2}) \quad (6)$$

In the specific case of the chosen test system, N_{GFOR1} was smaller than N_{GFOR2} which means that all parallel resources can be used before the GPU memory is filled. GPU memory is an important issue to consider because of the communication overhead it involves. Sending data to and from the GPU is slow when compared to the GPU processing time. In order to minimize these data transfers, it is efficient to divide the DNA sequence into large blocks before sending them to the GPU, albeit with some caveats which will later be discussed. It is important, however, to not surpass a predefined limit, since larger blocks will produce out of memory errors while the GPU computes. Fortunately, the test GPU used in this work has 1 GB of RAM which is sufficient to store the optimal DNA block size.

To determine the optimal DNA fragment size, an extensive series of tests have been performed. DNA blocks of different sizes were sent to the GPU and were processed by implementations using different quantities of GFORs. The results of these tests are shown in Figure 5.

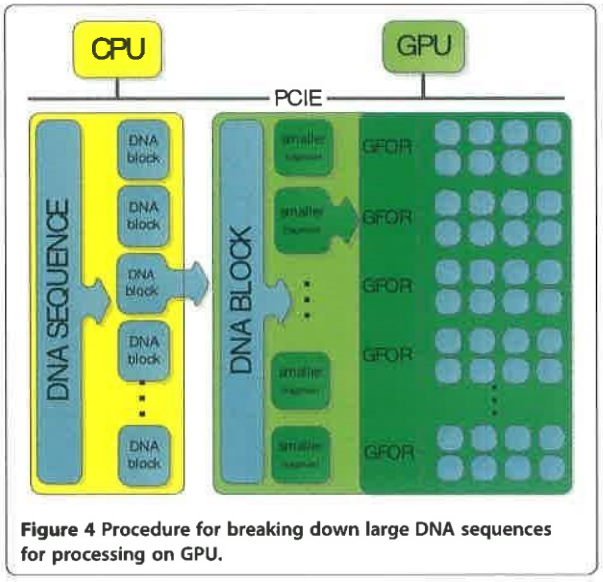


Figure 4 Procedure for breaking down large DNA sequences for processing on GPU.

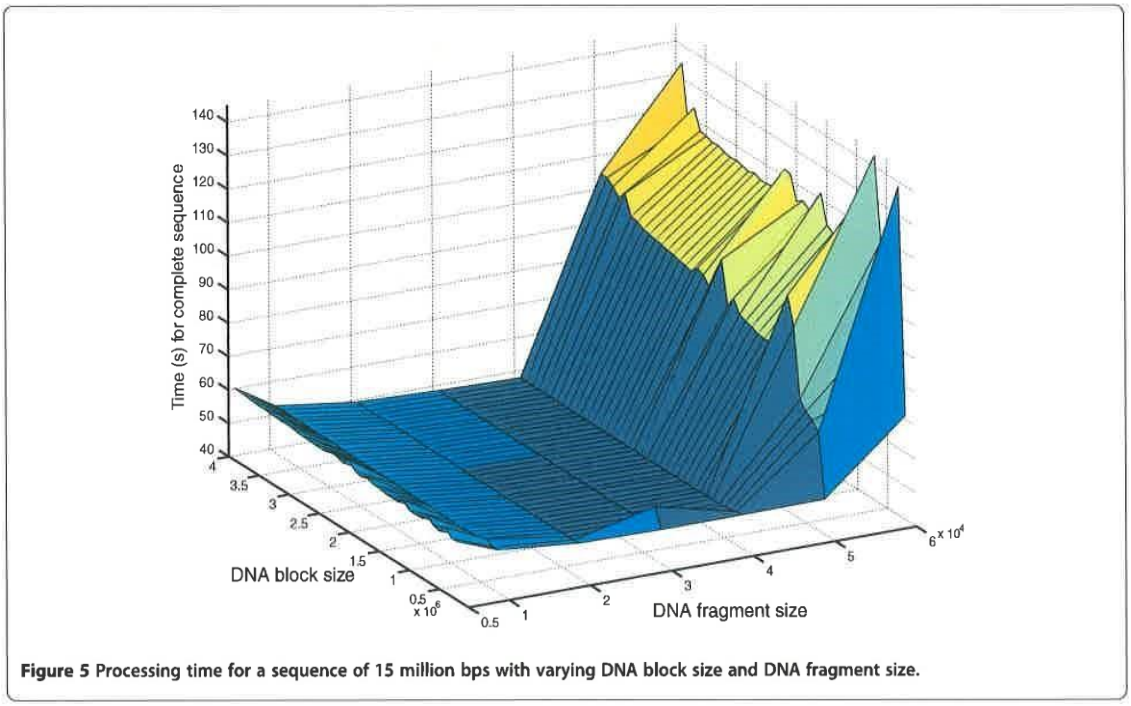


Figure 5 Processing time for a sequence of 15 million bps with varying DNA block size and DNA fragment size.

It is worth noting that, in this figure, the number of GFORs is not shown explicitly. Instead, the DNA fragment size processed by the GFOR instances is shown. The number of parallel GFORs can be deduced from the fragment size as it is proportional to it. The results shown in Figure 5 also include all data transfers between CPU and GPU memory. In an effort to make the comparisons fair, the processing times have been normalized for a large global sequence of 15 million bps. This is an estimated maximum sequence size over which a gene prediction algorithm needs to be executed in order to find a candidate gene between two genetic markers. The basis for this estimation can be found in [14] where the two markers were separated by 4 centiMorgan (cM) which roughly corresponds to 4 million bps. From Figure 5, it can be determined that using DNA fragment sizes between 20,000 and 40,000 bps yields the lowest processing times. A smaller DNA fragment size would not fully benefit from all the available parallel processing power and therefore, perform poorly. This can be shown in Figure 5, where the processing time increases as the fragment size drops to 5,000 bps.

Results

To evaluate the processing time of the different gene prediction implementations, tests were performed on DNA segments retrieved from the NCBI library (GRCh37.2). For the purpose of this work, the HFE2 (hemochromatosis type 2 in the human genome) gene [Accession number EMBL:AY372521] was chosen and large DNA segments containing this gene were extracted from the database. The HFE2 gene has four different variants each having a slightly different set of exons. These different exons are presented in Table 1. The human genome used for this test contains all the different exons whose beginning and ending positions are shown in Table 2.

From this table, it can be seen that, since exons 3a and 3b overlap each other, the gene prediction process should be able to identify a total of four distinct coding regions in the gene: exon 1, exon 2, exon 3a/3b and exon 4.

The goal of this test is to determine whether each implementation can detect coding regions in the HFE2 gene and to evaluate their processing time. This process is repeated for DNA sequences of different sizes in order to determine how each algorithm performs with varying amounts of data.

Table 1 data on HFE2 gene on chromosome 1

	Exons	Total length of coding region (bps)
Transcript variant a	1,2,3b,4	2234
Transcript variant b	1,3b,4	2048
Transcript variant c	1, 3a,4	1525
Transcript variant d	1,4	1488

Rivard et al. BMC Research Notes 2012, 5:183
http://www.biomedcentral.com/1756-0500/5/183

Table 2 HFE2 exon positions on chromosome 1

	Exon 1	Exon 2	Exon 3a	Exon3b	Exon 4
Start	145,413,191	145,414,693	145,415,278	145,415,278	145,416,313
End	145,413,427	145,414,879	145,415,315	145,415,838	145,417,545
Length	236	186	37	560	1232

Lengths are in bps.

For each of these cases, the measured processing time also includes data transfer time. One of the advantages of including data transfer time is that the tests reflect what the user is actually experiencing. In addition, it allows for processing times of larger sequences to be extrapolated linearly. It should be reiterated that these tests are not meant to quantify the reliability of the chosen approach; this type of study has already been well-documented and has shown that frequency analysis is reliable [9,10].

Availability and requirements

For this paper, an Intel® Core™ i7-2600 K processor (8 MB cache, 3.40 GHz, 8 threads) was used with 8 GB of RAM. The graphics card used was a GeForce GTX 560 1 GB GDDR5 with 336 CUDA cores. It is important to note that, while some may think that the availability of 336 CUDA cores could improve the execution speed by a factor of 336 times compared to a CPU implementation, it is not the case. This is due to the fact that CUDA cores and CPU cores serve different purposes and therefore, are architecturally different. Details concerning CUDA cores can be found in Nvidia's documentation [24].

Coding region detection

In the first part of the test, it is important to verify that the coding region detection aspect of each implementation is functional. To do so, it is possible to plot the data of the frequency analyses and compare them with the known structure of the gene [14,25]. MATLAB results show that all implementations proposed in this work produce the same outcome (Figure 6). Each peak in this figure represents a probable coding region and, by comparing the results to the gene structure presented in Tables 1 and 2, it can be shown that the approach is successful at identifying coding regions.

Processing time

Using the test setup described previously, ten different implementations of the gene prediction algorithm have been evaluated using seven different DNA segment sizes. The results of the test are presented in Table 3.

While the most significant results are shown in that table, many other tests had to be run to ensure that no critical points were missed.

The first implementation considered is the goertzel.m approach where the *goertzel* function was executed. It was found that, when parallelism was not used, the processing time became excessive and required over 4.5 h to process 15 million bps. That option was therefore not considered further. For an implementation using eight threads, running goertzel.m for a sequence of 1 million bps takes 162 s on the test machine. For sequences larger than 5 million bps, it was deemed unnecessary to evaluate the performance as the processing speed was already too low. When the DNA sequences take too much time to process, the table indicates that it is too large to consider (TLTC).

The solution using the GoertzelMEX function was implemented using one, two, four and eight threads. This is done so as to show the performance gained when CPU cores are added. Row 5 of Table 3 shows a nearly 18x improvement over using goertzel.m when processing a 1 million bps sequence with the same eight threads. Rows 3 and 4 show that, even with two or four threads, it is already possible to obtain better results than with goertzel.m using eight threads.

A custom Goertzel algorithm was also tested running on eight threads. Its processing time is presented on the sixth row of Table 3. It shows that, when only the CPU is considered, this approach provides the best results.

The gene prediction algorithm was also implemented with MATLAB's FFT function and the processing times are presented in Row 9 of Table 3. The results show that the FFT's processing time is much larger than with the Goertzel implementations. This confirms the fact that the Goertzel algorithm is more efficient than the FFT when the targeted number of frequencies is small.

The two remaining implementations use the FFT function on the GPU. It should be noted that when using the GPU, the FFT is faster than Goertzel and thus the latter is not discussed in greater detail. Nonetheless, results from running Goertzel on the GPU are shown on Row 10 in Table 3. For the first implementation of the FFT, the entire sequence was loaded onto the GPU before processing (Row 7). In the other implementation, the sequence was broken down into smaller blocks of 1 million bps and processed sequentially (Row 8).

It is worth noting that Rows 7 and 8 represent the same operation up to the size of 1 million bps. After that point, the results remain almost identical: sending 15 times a sequence of 1 million bps is only one second faster than sending a large 15 million bps sequence. Our tests have shown that breaking down a large sequence into sizes smaller than 1 million bps results in performance losses whereas sizes larger than 15 million bps yields out of memory errors. According to our results, GPU implementations using the proposed test setup should divide DNA sequences into blocks of 1 to 15 million bps for optimal results.

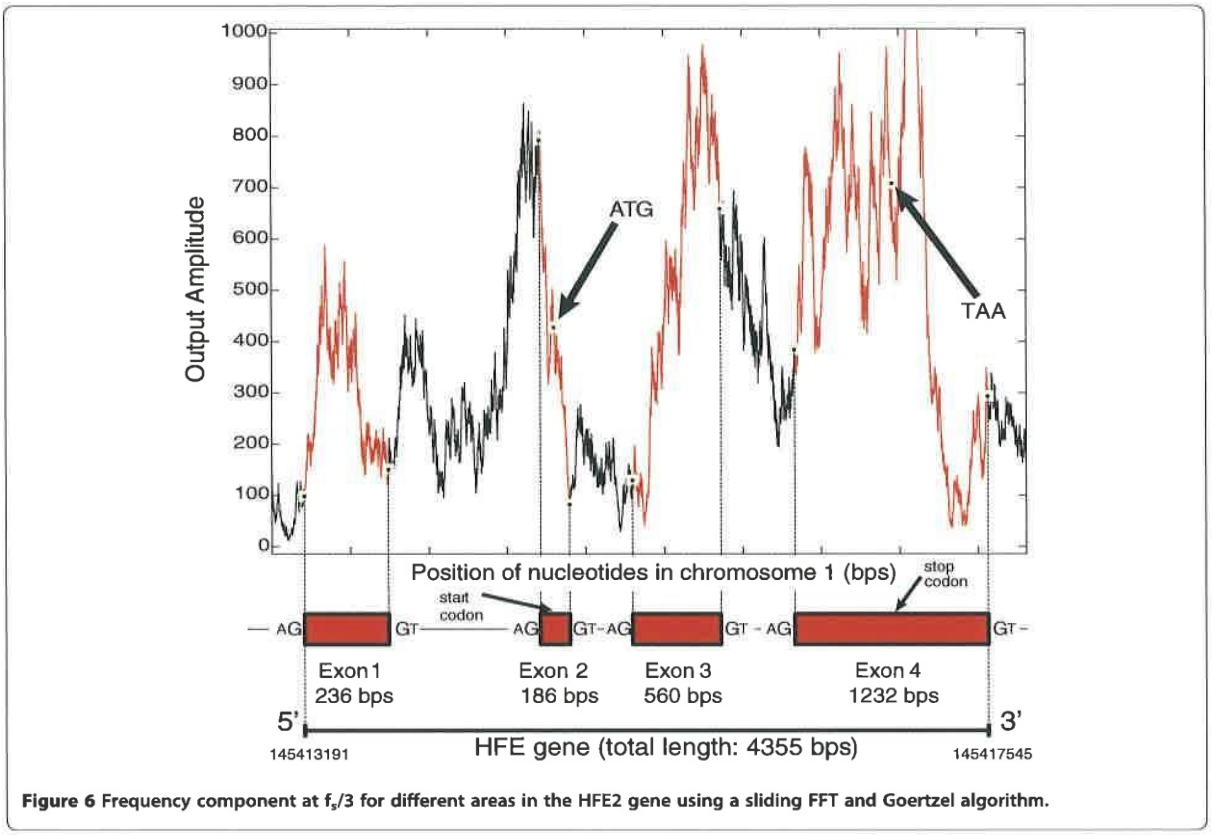


Figure 6 Frequency component at $f_s/3$ for different areas in the HFE2 gene using a sliding FFT and Goertzel algorithm.

Discussion

The tests performed in the previous section show that the choice of algorithm and of implementation plays an important role in the feasibility of gene prediction in MATLAB. Using a straightforward goertzel.m approach with a single thread would not have been possible within a reasonable time frame. Even with eight threads enabled, the extrapolated processing time for 15 million bps would be close to 41 min. When GoertzelMEX and a custom Goertzel algorithm were used, the processing time was reduced to close to 90 s. Finally, with the GPU, the processing time could be reduced to 57 s. These results show that acceleration via a

Table 3 Runtime for varying sequence lengths

			Time (s) for the following sequence sizes:						
Function	**Loop type**	**Processing**	**5,000**	**50,000**	**200,000**	**500,000**	**1,000,000**	**5,000,000**	**15,000,000**
1 goertzel.m	PARFOR	CPU 8 T	1.06	8.29	32.91	82.16	161.89	805.44	TLTC
2 goertzelMEX	FOR	CPU	0.18	1.78	7.11	17.84	35.65	178.30	535.21
3 goertzelMEX	PARFOR	CPU 2 T	0.19	0.99	3.86	9.58	19.20	100.39	287.35
4 goertzelMEX	PARFOR	CPU 4 T	0.18	0.60	2.36	5.81	11.41	56.27	164.84
5 goertzelMEX	PARFOR	CPU 8 T	0.25	0.53	1.95	4.75	9.52	47.49	164.57
6 Custom Goertzel	**PARFOR**	**CPU 8 T**	**0.25**	**0.37**	**1.18**	**2.83**	**5.56**	**27.63**	**87.47**
7 JACKET's FFT (full sequences)	GFOR	GPU	0.03	0.22	0.78	1.90	3.78	18.82	57.68
8 JACKET's FFT (1 M blocks)	**GFOR**	**GPU**	**0.03**	**0.22**	**0.78**	**1.90**	**3.78**	**18.90**	**56.70**
9 Matlab's FFT	PARFOR	CPU 8 T	0.29	0.42	1.46	3.51	6.95	34.12	109.15
10 Custom Goerztel on GPU	GFOR	GPU	0.22	0.79	2.82	7.15	14.09	71.01	213.31

CUDA-enabled graphics card yields the best performance when the sequence to be analyzed is large enough to justify the overhead, yet small enough not to deplete available resources. For a relatively simple yet efficient implementation, a custom Goertzel algorithm using MATLAB's CPU parallelization (Row 6) can provide results that are about 36% slower than with a GPU (Row 8).

While the absolute difference in processing time may seem inconsequential, it should be noted that frequency analysis is often combined with other techniques to make gene prediction more robust. In addition, to improve on the reliability of the approach, it is sometimes relevant to perform frequency analysis for different window sizes. In those cases, the benefits of using a GPU can be justified. Otherwise, a well-designed MATLAB algorithm running on CPU would provide satisfactory results.

The approach proposed in this paper can deal with a large amount of data in a reasonable time while being accurate and reliable given the proven track record ([9,10]) of the algorithm. It also remains easy to use for researchers who are not experts in the field of bioinformatics. Building upon this foundation, the next step is to use these methods to accurately identify new genes involved in monogenic and complex diseases.

Conclusions

In this paper, we presented a number of ways of implementing gene prediction using MATLAB. The different implementations were described and evaluated to test for processing time. We have shown that this approach allows for the processing of very large sequences (15 million bps was used) in a reasonable time. This renders the processing of the entire human genome and other organisms very feasible on a conventional desktop machine. It is the authors' belief that this type of demonstration has never been published before.

In addition, each implementation was also evaluated for shorter DNA sequences to help analyze how the processing time evolves with different sequence lengths. Results show that, with common desktop computers, it is possible to perform gene prediction on sequences of 15 million bps quite rapidly. Using MATLAB's FFT, an eight-core parallel implementation was able to complete the operation in less than five minutes whereas a GPU-accelerated version did it in approximately one minute. Using a CPU, the work shows that direct access to the MEX function increases execution speed and that the PARFOR construct should be used in order to take full advantage of the parallelizable Goertzel implementation. When the target is a GPU, the work shows that data need to be segmented into manageable sizes within the GFOR construct before processing in order to minimize execution time.

The fact that these results can be achieved within the MATLAB environment without calling upon custom hardware/software solutions means that researchers already familiar with MATLAB can readily use this technique without requiring additional IT resources. The source code provided with this paper can be run locally for accurate results at fast processing speeds (Additional file 1, Additional file 2). This can help make gene prediction tools more accessible to geneticists and can help speed the discovery of new genes.

Additional files

Additional file 1: Example code for using JACKET_seq_FFT.m

Additional file 2: Frequency analysis on a sequence using JACKET's FFT.

Competing interests
The authors declare that they have no competing interests

Authors' contributions
HTB and RB proposed the initial concept and participated in the design of the study. The detailed study design was developed by the members of the research team, SRR and JGM who also carried out the computations and simulations to produce the published results. All authors have read and approved the final manuscript

Authors' information
Sylvain Robert Rivard received the B.Sc., M.Sc. and Ph. D. degree in molecular genetics from the Université de Montréal, Québec, Canada, in 1984, 1988 and 1995 respectively. He held three Posdoctorals positions (Colorado State University, Fort-Collins, Colorado, USA; Établissement Français du sang, Brest, France and TIGEM Napoli in Italia). He is currently pursuing the M.A.Sc. degree in electrical engineering from Université du Québec à Chicoutimi (UQAC), Canada. His main research interests include human genetics, computational biology and microelectronics.
Jean-Gabriel Mailloux received the B.Eng. degree in computer engineering from the Université du Québec à Chicoutimi, Canada, in 2005 where he also received the M.A.Sc. in computer engineering in 2008. He is currently pursuing a Ph. D. in computer engineering. His current research interests include computational biology and optimal approaches for testing algorithms in FPGAs and GPUs.
Rachid Beguenane received his D.E.A. and Ph.D. degrees, both in electrical engineering from CNAM, Paris, France in 1991 and 1994 respectively. During this period, he conducted his Ph.D. research work at École des Mines de Douai, France. From 1995 to 1997, he held a teaching and research position at the Professional Institute of Amiens, France. During 1997/1998, he was a post-doctoral researcher at the School of Information Technology and Engineering of the University of Ottawa, Canada. From 1998 to 2002, he has been employed as ASIC/FPGA design engineer by Telexis inc., Nortel Networks, and Prova Scientific Corp. in Ottawa and Montreal, Canada. He mainly designed and verified IC chips for telecommunication industry. From 2002 to 2009, he was associate professor at Université du Québec à Chicoutimi (UQAC), Canada. Since 2009, he is associate professor in electrical and computer engineering department of Royal Military College of Canada. Dr. Beguenane has authored and co-authored more than 60 technical papers.
Hung Tien Bui (S' 04, M' 06) received the B.Eng. degree in 1998 from Concordia University in electrical engineering in Montreal, Canada and the M.A.Sc. degree in 2000 in computer engineering from Florida Atlantic University. He received the Ph.D. degree in electrical engineering in 2006 from École Polytechnique de Montréal. He has been with the Department of Applied Sciences at Université du Québec à Chicoutimi since 2006 where he is currently an assistant professor. His research interests revolve around high performance microelectronics and signal processing applied to computational biology and health sciences.

Acknowledgements
This work was supported in part by NSERC. The authors would also like to thank RESMiQ, CMC Microsystems and AccelerEyes for access to software and other resources. Finally, we would also like to thank M. J-D. Otis for helpful discussions and suggestions.

Author details
[1]Département des sciences appliquées, Université du Québec à Chicoutimi, 555 blvd de l'Université, Chicoutimi, QC, G7H 2B1, Canada [2]Royal Military College of Canada, Kingston, ON K7K 7B4, Canada

Received: 7 October 2011 **Accepted:** 10 April 2012
Published: 10 April 2012

References

1. Fickett J, Tung CS: **Assessment of protein coding measures.** *Nucleic Acids Research* 1992, **20**:6441.
2. Marhon Sa, Kremer SC: **Gene prediction based on DNA spectral analysis: a literature review.** *Journal of computational biology: a journal of computational molecular cell biology* 2011, **18**:639–676.
3. Stein LD: **Human genome: end of the beginning.** *Nature* 2004, **431**:915–916.
4. Makarov V: **Computer programs for eukaryotic gene prediction.** *Briefings in bioinformatics* 2002, **3**:195–9
5. Kauer G, Bloker H: **Applying signal theory to the analysis of biomolecules.** *Bioinformatics* 2003, **19**:2016–2021
6. Tuqan J, Rushdi A: **A DSP approach for finding the codon bias in DNA sequences.** *Selected Topics in Signal Processing, IEEE Journal of* 2008, **2**:343–356
7. Fickett JW: **Recognition of protein coding regions in DNA sequences.** *Nucleic Acids Research* 1982, **10**:5303
8. Goertzel G: **An algorithm for the evaluation of finite trigonometric series.** *The American Mathematical Monthly* 1958, **65**:34–35.
9. Datta S, Asif A, Wang H: ***Prediction of protein coding regions in DNA sequences using Fourier spectral characteristics.*** Miami: IEEE/sixth International Symposium on Multimedia Software Engineering (ISMSE'04; 2004:160–163.
10. Dimitris A: **Genomic Signal Processing.** *IEEE Signal Processing Magazine* 2001, **18**:8–20.
11. Yin C, Yau SS-T: **Prediction of protein coding regions by the 3-base periodicity analysis of a DNA sequence.** *Journal of theoretical biology* 2007, **247**:687–94.
12. Wang L, Stein LD: **Localizing triplet periodicity in DNA and cDNA sequences.** *BMC bioinformatics* 2010, **11**:550
13. Fuentes AR, Ginori JVL, Grau Abalo R: **Detection of coding regions in large DNA sequences using the short time Fourier Transform with reduced computational load.** *Lecture Notes in Computer Science* 2006, **4225**:902–909
14. Rivard SR, Lanzara C, Grimard D, Carella M, Simard H, Ficarella R, Simard R, D'Adamo AP, Férec C, Camaschella C, Mura C, Roetto A, De Braekeleer M, Bechner L, Gasparini P: **Juvenile hemochromatosis locus maps to chromosome 1q in a French Canadian population.** *European journal of human genetics: EJHG* 2003, **11**:585–9.
15. Voyer M, Rivard S-R, Morin L, Bui HT: **Rapid prototyping of the Goertzel algorithm for hardware acceleration of exon prediction** In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)* Rio de Janeiro: IEEE; 2011:85–88
16. Chechetkin VR, Turygin AY: **Size-dependence of three-periodicity and long-range correlations in DNA sequences.** *Physics Letters A* 1995, **199**:75–80
17. Silverman BD, Linsker R: **A measure of DNA periodicity.** *Journal of theoretical biology* 1986, **118**:295–300
18. Issac B: **Locating probable genes using Fourier transform approach.** *Bioinformatics* 2002, **18**:196–197
19. Tiwari S, Ramachandran S, Bhattacharya A, Bhattacharya S, Ramaswamy R: **Prediction of probable genes by Fourier analysis of genomic sequences.** *Bioinformatics* 1997, **13**:263–270.
20. Li W, Marr T, Kaneko K: **Understanding long-range correlations in DNA sequences.** *Physica D: Nonlinear Phenomena* 1994, **75**:392–416.
21. Kwan HK, Arniker SB: **Numerical representation of DNA sequences** *Proceeding of IEEE International Conference on Electro/Information Technology, Windsor, Canada* 2009:307–310
22. Voss RF: **Evolution of long-range fractal correlations and 1/f noise in DNA base sequences.** *Physical review letters* 1992, **68**:3805–3808.
23. Mahmood A, Ambikairajah E, Epps J: **Digital Signal Processing Techniques for Gene Finding in Eukaryotes.** *Digital Signal Processing* 2008, **5099**:144–152
24. Nvidia: **Whitepaper NVIDIA's Next Generation CUDA Compute Architecture.** 2009:1–22.
25. Lanzara C, Roetto A, Daraio F, Rivard S, Ficarella R, Simard H, Cox TM, Cazzola M, Piperno A, Gimenez-Roqueplo AP, *et al*: **Spectrum of hemojuvelin gene mutations in 1q-linked juvenile hemochromatosis.** *Blood* 2004, **103**:4317

doi:10.1186/1756-0500-5-183
Cite this article as: Rivard *et al.*: **Design of high-performance parallelized gene predictors in MATLAB.** *BMC Research Notes* 2012 **5**:183