جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

République Algérienne Démocratique et Populaire.

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université A. Mira de Béjaia.
Faculté des Sciences exactes
Département Informatique

Centre de Recherche sur l'Information Scientifique et Technique

Mémoire de fin de cycle
Pour l'obtention du diplôme de Master en Informatique
Option : Réseaux et Système Distribués.

Thème

Algorithme de recherche locale pour le problème d'assemblage de fragments d'ADN sur processeurs graphiques (GPU)

Encadré par

Dr. SIDER Abdrahmane (Université de Béjaïa)
Dr. BENDJOUDI Ahcen (CERIST)
Dr. MEHDI Malika (CERIST)

Réalisé par

BOUHALI Raouf
DJERROUD Yacine

Devant les jurys

Présidente
M^{me} : TIGHIDET Soraya

Examinateurs
Mr : LARBI Ali
Mr : ALLAM Khaled

Promotion "2012 - 2013"

Remerciements

Au nom de DIEU, clément, miséricordieux, que les grâces de DIEU, sa paix et ses salutations soient sur notre prophète Mohamed.

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force, la patience , la foi et le courage d'accomplir notre but et d'achever ce Modeste travail.

En second lieu, Nous voudrions présenter nos remerciements à nos encadreurs Dr SIDER Abdrahmane, Dr BENDJOUDI Ahcen et Dr MEHDI Malika, afin de leur témoigner de notre gratitude pour leur patiences, leur disponibilités , leur conseils et leur soutiens qui nous a été précieux tout au long de ce travail pour le mener à bon port.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail et de l'enrichir par leurs propositions.

Que tout enseignant nous ayant fait bénéficier de son savoir, trouve ici l'expression de notre profonde gratitude.

À toute personne ayant contribué, de prés ou de loin, à ce travail, nous disons merci.

Merci à tous

Table des matières

Introduction Générale

Introduction générale :

Le séquençage d'ADN est devenu une pratique indispensable dans les domaines de médecine, de phylogénétique et du crime (police technique). Il permet par exemple d'identifier un suspect sur la scène de crime, retrouver les liens de parenté entre deux personnes, établir des relations fonctionnelles entre les virus, les symptômes et les maladies, etc. L'assemblage de fragments ADN est une partie principale de ce processus.

Le problème d'assemblage de fragments ADN est un problème de la classe NP-difficle, dont il n'existe aucun algorithme connu, à la fois correct, efficace et complet. Des centaines de travaux de recherche depuis environ un demi-siècle, visent à mettre en œuvre des méthodes de résolution de ce problème, basées essentiellement sur des métaheuristiques. D'un autre côté, le progrès exponentiel de la capacité de calcul des GPU en fait un outil intéressant pour des applications scientifiques de traitement intensif.

Nous proposons dans ce document un algorithme basé sur la recherche locale, implémenté en utilisant les langages C/C++, accompagnés d'un modèle de parallélisation destiné à une future implémentation sur GPU à l'aide de nVIDIA CUDA.

Pour cela, nous avons structuré notre mémoire en deux grandes parties.

La première partie est composée de 3 chapitres. Elle contient un ensemble de généralités, couvrant le spectre de connaissances liées à notre travail, à savoir : les métaheuristiques, la bio-informatique et le calcul sur processeurs graphiques.

Le chapitre I introduit les métaheuristiques ainsi que quelques notions fondamentales sur l'optimisation combinatoire. Il est suivi d'un deuxième chapitre qui donne un aperçu sur l'historique du séquençage d'ADN avec également les informations essentielles au sujet du processus d'assemblage de fragments ADN ainsi que les spécificités d'un tel problème. Le troisième chapitre est consacré aux concepts liés aux GP-GPU et à nVIDIA.

La partie II se focalise sur la problématique étudiée et contient 2 chapitres : le chapitre IV qui passe en revue de quelques travaux essentiels constituant notre recherche bibliographique ; et

le cinquième et dernier chapitre qui expose notre contribution et avance une auto-critique des méthodes en question.

Chapitre I
Introduction aux métaheuristiques

I.1. Introduction

Les métaheuristiques sont des algorithmes d'optimisation s'appuyant sur des techniques de choix pouvant piloter une ou plusieurs heuristiques de manière abstraite, sans faire appel à un problème spécifique. Ce chapitre est consacré à la présentation de quelques métaheuristiques, en mettant un accent sur les méthodes de recherche locale. Nous définirons, dans un premier temps, des notions essentielles sur l'optimisation combinatoire et sur la recherche heuristique. Nous donnerons ensuite un aperçu général des principales métaheuristiques sous différentes classifications connues.

I.2. Problèmes d'optimisation combinatoire

Les problèmes d'optimisation combinatoires sont des problèmes mathématiques qui consistent à trouver une meilleure solution satisfaisant toutes les contraintes ou les critères donnés. La meilleure solution est définie par une fonction objective. On qualifie généralement de combinatoires les problèmes dont la résolution se heurte à une explosion du nombre de solutions possibles à explorer.

De très nombreux problèmes réels quotidiens sont complexes et indéterministes, et pour lesquels la théorie de la complexité nous disant que l'on ne pourra pas à priori trouver un algorithme de résolution à la fois rapide, correct et complet. l'intérêt majeur de l'optimisation combinatoire est de trouver des méthodes qui sont fiables et efficaces permettant de résoudre ces problèmes pratiques. **[5]**

Pour résoudre un problème combinatoire, on peut explorer l'ensemble des solutions de façon exhaustive et systématique, et tenter de réduire la taille du problème en utilisant des techniques d'optimisation visant à éliminer le plus tôt possible des sous-ensembles de solutions en prouvant qu'ils ne contiennent pas de solution et des heuristiques de choix visant à se diriger le plus rapidement possible vers les meilleures solutions. Ces approches sont complètes dans le sens où elles sont toujours capables de trouver la solution optimale ou, le cas échéant, de prouver l'absence de solution ; en revanche, le temps de résolution dépend de l'efficacité des techniques d'élagage et des heuristiques considérées, et est exponentiel dans le pire des cas. **[5]**

Plusieurs problèmes d'OC sont dits NP-difficiles et ne peuvent être résolus de façon optimale par des algorithmes exacts. Les métaheuristiques ont prouvé leur efficacité à résoudre un grand nombre de ces problèmes en leur trouvant des solutions approchées en un temps raisonnable. Cependant, face à des instances de grande taille, elles ont besoin d'un temps de calcul

et d'une quantité d'espace mémoire considérables pour être performantes dans l'exploration de l'espace de recherche.

I.3. Méthodes heuristiques

Une heuristique est une méthode, conçue pour un problème d'optimisation donné, qui produit une solution non nécessairement optimale lorsqu'on lui fournit une instance de ce problème. **[13]**

Les méthodes heuristiques sont utilisées pour des problèmes de grandes tailles pour lesquels les méthodes exactes prennent un temps exponentiel. Généralement, on n'obtient pas la solution optimale mais une solution approchée, assez bonne. Une méthode heuristique se déplace dans l'espace de solutions, de voisin en voisin, en se basant seulement sur les informations locales (la solution courante et son voisinage), avec objectif d'atteindre un optimum au-delà duquel aucun mouvement local n'est possible. Cette méthode de par sa simplicité est caractérisée par une moindre consommation de mémoire, mais elle ne dirige pas généralement la résolution du problème vers un optimum global. **[3]**

Heuristiques systématiques et randomisées :

Les heuristiques systématiques explorent le voisinage pour choisir la solution suivante. Elles sont:

- **Meilleur voisin :** choisir la solution voisine qui est évaluée le mieux.
- **Premier voisin :** choisir la première solution voisine qui améliore la fonction objective.
- **Heuristique multi-étages :** tout d'abord, on choisit un sous-ensemble des solutions du voisinage ; et puis, choisir dans cet ensemble une solution qui améliore le mieux la fonction objectif.

L'heuristique du meilleur voisin perd le temps pour explorer le voisinage afin de trouver la meilleure solution mais le nombre d'itérations pour atteindre le local optimal peut être moins que celui de l'heuristique First Neighbor. Multistage heuristic fait équilibre les avantages et désavantages des deux heuristiques. **[3]**

Les heuristiques randomisées sont caractérisées par un choix aléatoire de la prochaine solution

- **Amélioration aléatoire** (Random Improvement) : choisir au hasard une solution voisine et se déplacer vers cette solution si elle améliore la fonction objective.

- **Heuristique Métropolis** (Metropolis Heuristique) : choisir au hasard une solution voisine de la solution courante, on se déplace vers cette solution si elle n'accroît pas la fonction objective. Si elle diminue la fonction d'objective, elle est acceptée avec une certaine probabilité, déterminée de façon différente selon chaque programme.

L'emploi de méthodes de recherche heuristiques se justifie par l'inexistence ou l'inefficacité de méthodes exactes, et la difficulté ou l'impossibilité d'une exploration exhaustive de l'espace des solutions.

I.4. Méthodes métaheuristiques

Une métaheuristique est un ensemble de concepts qui peuvent être utilisés pour définir des méthodes heuristiques qui peuvent être appliquées à un large éventail de problèmes différents. En d'autres termes, une métaheuristique peut être considérée comme une bibliothèque algorithmique générale qui peut être appliqué à différents problèmes d'optimisation avec relativement quelques modifications pour les adapter à un problème spécifique. **[4]**

Une métaheuristique est définie comme un processus itératif qui guide une heuristique subordonnée en combinant intelligemment différents concepts pour explorer et exploiter l'espace de recherche, des stratégies d'apprentissage sont utilisées pour structurer des renseignements pour trouver des solutions efficaces presque optimales.

Tandis que les méthodes heuristiques choisissent la solution voisine en basant sur les informations locales, les méthodes métaheuristiques collectent les informations au cours de l'exécution et dirigent la recherche vers l'optimisation globale. En effet, les métaheuristiques ont besoins de plus de mémoire que les méthodes heuristiques afin de garder certaines informations collectées au long de la résolution du problème. **[3]**

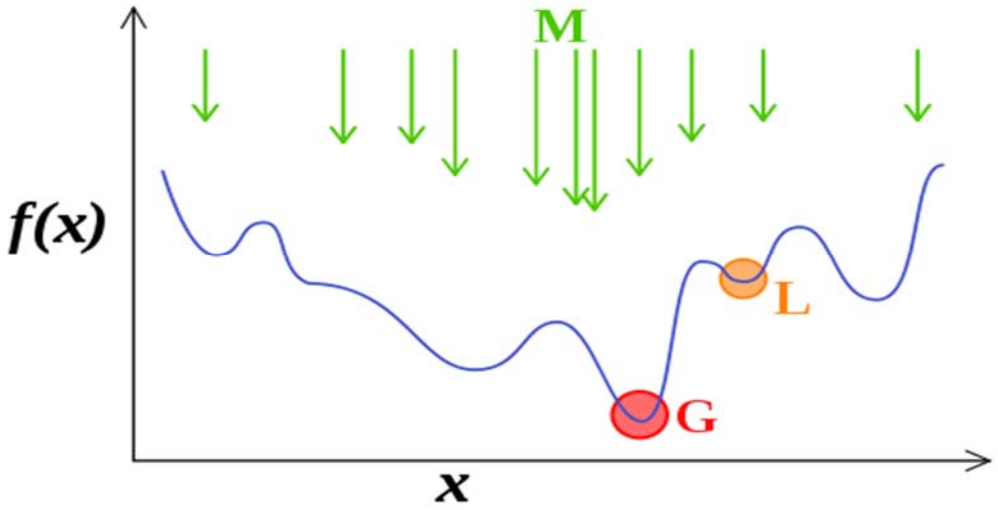

Figure I.1 : Représentation du minimum local et global d'une fonction

Les métaheuristiques (M) sont souvent des algorithmes utilisant un échantillonnage probabiliste. Elles tentent de trouver l'optimum global (G) d'un problème d'optimisation difficile, sans être piégé par les optima locaux (L).

I.4.1 Classification des métaheuristiques [8]

Il existe plusieurs façons de classer les métaheuristiques. Ce diagramme tente de présenter où se placent quelques-unes des méthodes les plus connues. Un élément présenté à cheval sur différentes catégories indique que l'algorithme peut être placé dans l'une ou l'autre classe, selon le point de vue adopté.

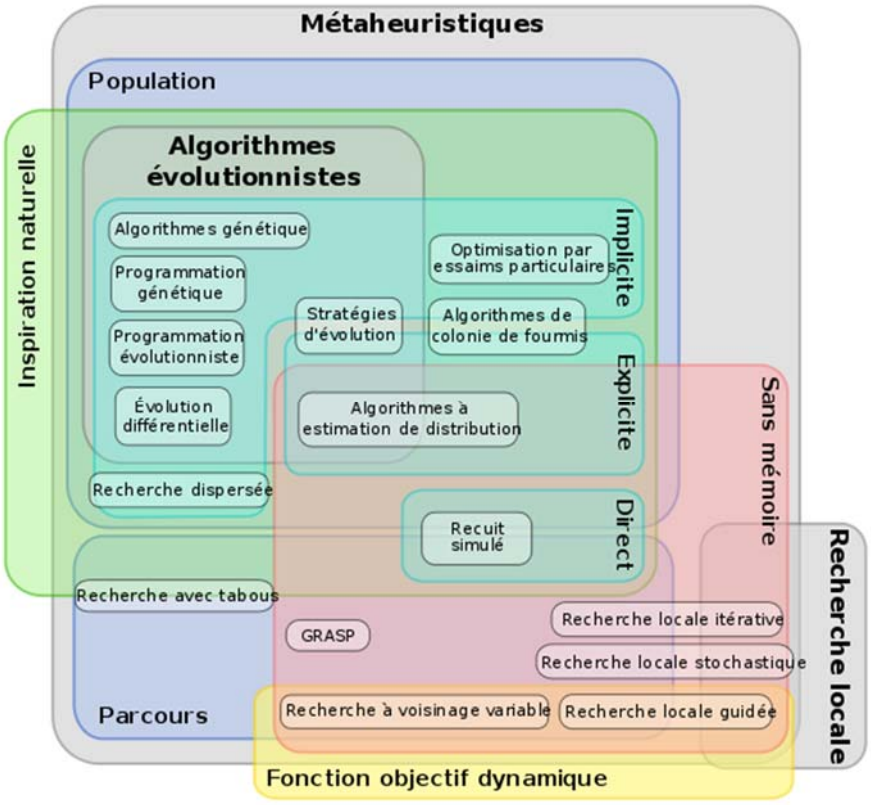

Figure I.2 : Classification des métaheuristiques **[9]**

a. Les métaheuristiques inspirer de processus naturels

On peut distinguer les métaheuristiques qui s'inspirent de phénomènes naturels et celles qui ne s'en inspirent pas. Par exemple, les algorithmes génétiques et les algorithmes de colonie de fourmis s'inspirent respectivement de la théorie de l'évolution et du comportement des fourmis à la recherche de nourriture. Par contre, la méthode Tabou n'a semble-t-il pas été inspirée par un phénomène naturel. Une telle classification ne semble cependant pas très utile et est parfois difficile à réaliser. Il existe de nombreuses métaheuristiques récentes qu'il est difficile de classer dans l'une des 2 catégories.

b. Les métaheuristiques trajectoire et avec population de solution

Une autre façon de classifier les métaheuristiques est le nombre de solutions qu'elle manipule. Il en existe donc celles qui optimisent une population de solutions (ex : recherche par faisceau) et celles qui ne manipulent qu'une seule solution à la fois (ex : escalade).

c. Les métaheuristiques statiques et dynamiques

Les métaheuristiques peuvent également être séparées selon leur manière d'utiliser la fonction objectif, certaines métaheuristiques dites statiques travaillent directement sur la même fonction objectif alors que d'autres, dites dynamiques, font usage d'une fonction g obtenue à partir de f en ajoutant quelques composantes qui permettent de modifier la topologie de l'espace des solutions, ces composantes additionnelles pouvant varier durant le processus de recherche.

d. Selon la structures de voisinage exploitées

Une autre classification des métaheuristiques en fonction du nombre de structures de voisinages utilisées. Étant donné qu'un minimum local relativement à un type de voisinage ne l'est pas forcément pour un autre type de voisinage, il peut être intéressant d'utiliser des métaheuristiques basées sur plusieurs types de voisinages.

e. Les métaheuristiques avec ou sans mémoire

Certaines métaheuristiques font usage de l'historique de la recherche au cours de l'optimisation, alors que d'autres n'ont aucune mémoire du passé. Avec les algorithmes sans mémoire, l'action à réaliser est totalement déterminée par la situation courante. On différencie généralement les méthodes ayant une mémoire à court terme de celles qui ont une mémoire à long terme.

f. Les métaheuristiques avec diversification ou intensification

Une dernière classification est selon l'utilisation de la diversification et de l'intensification. On sous-entend généralement par la diversification une exploration assez large de l'espace de recherche alors que le terme intensification vient plutôt mettre l'accent sur l'exploitation de l'information accumulée durant la recherche et sa concentration sur une zone précise de S. Il est important de bien doser l'usage de ces deux ingrédients afin que l'exploration puisse rapidement identifier des régions de l'espace de recherche qui contiennent des solutions de bonne qualité, sans perdre trop de temps à exploiter des régions moins prometteuses.

I.4.2 Principales métaheuristiques

Dans notre description des principales métaheuristiques, nous allons nous appuyer sur la classification, faisant la différence entre les méthodes de parcours de l'ensemble de solutions et les méthodes basées sur une population (de solutions), population qu'on fait évoluer en essayant de l'améliorer par rapport à l'objectif recherché.

Les méthodes de parcours ou encore trajectoire manipulent une seule solution à la fois et tentent itérativement d'améliorer cette solution. Elles construisent une trajectoire dans l'espace des solutions en tentant de se diriger vers des solutions optimales.

On peut en citer par exemple :

- La recherche locale.
- Le recuit simulé [Kirkpatrick et al., 1983].
- La recherche tabou [Glover, 1986].
- La recherche à voisinages variables (VNS) [Mladenovic et Hansen, 1997].

Les méthodes qui travaillent avec une population de solutions : en tout temps on dispose d'une base de plusieurs solutions, appelée population. L'exemple le plus connu est l'algorithme génétique.

I.4.2.1 Les Méthodes par voisinage ou trajectoire

Le principe de ces méthodes est simple : il s'agit de l'exploration de l'espace des solutions en partant d'un point et en se déplaçant de proche en proche ; les voisins d'une solution seront des solutions proches de celle-ci, i.e. qu'on peut obtenir par exemple en transformant à peine la solution courante. Par exemple, dans le problème du voyageur de commerce, les voisins d'une tournée sont les tournées obtenues en supprimant deux arcs non contigus de la tournée et en reconnectant les deux sous-tournées obtenues. Pour appliquer les méthodes par voisinage, il faut donc avoir au préalable défini comment déterminer les voisins d'une solution. Bien sûr, pour un même problème, plusieurs voisinages peuvent être définis.

On présente brièvement ici quelques méthodes basées sur la notion de voisinage :

a. Algorithmes de recherche locale

L'approche de la recherche locale est très efficace pour résoudre les problèmes NP-difficile ou NP-complet surtout pour ceux ayant un espace de recherche avec un voisinage de très grande taille, elle garantit une solution en temps abordable grâce à sa rapidité et à son principe relativement simple. Dans certain cas, la qualité de la solution trouvée n'est pas bonne. **[3]**

Les grands avantages d'algorithmes de recherche locale sont : **[12]**

- Utilisation d'un petit espace de mémoire.

- Parcours de l'espace de recherche d'une façon intelligente, recherche des régions différentes de l'espace de recherche.
- Production d'une solution optimale ou acceptable avec un temps de calcul raisonnable.

La recherche locale est une métaheuristique utilisée pour résoudre des problèmes dans plusieurs domaines. **[6]**

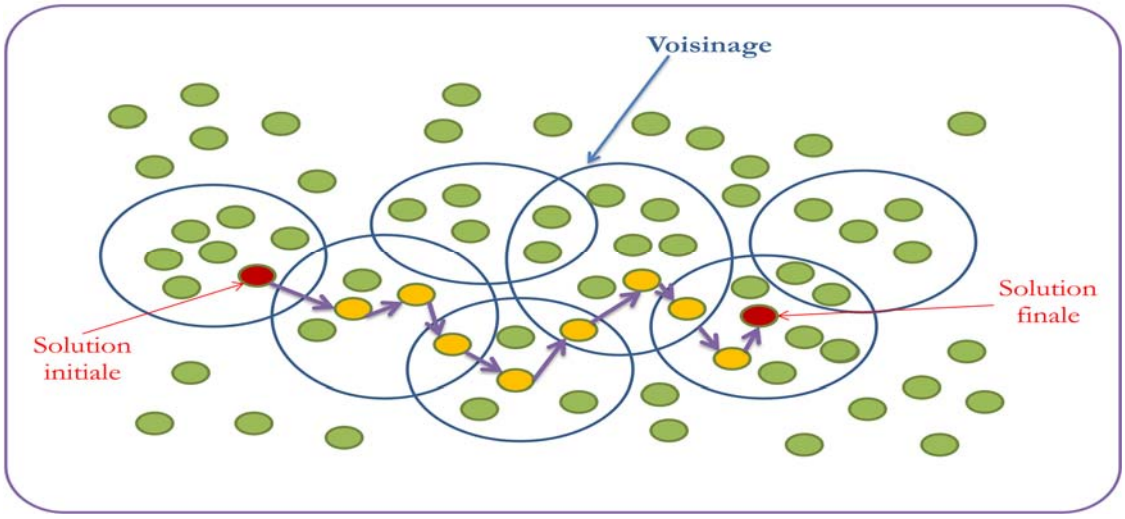

Figure I.3 : Procédure générale de la recherche locale

b. Principe de recherche locale

Un algorithme de recherche locale démarre à partir d'une solution initiale réalisable et se déplace itérativement aux solutions voisines en appliquant une série de modifications locales. Le critère distinctif des solutions candidates est la fonction objectif (fitness function), dont le rôle est de quantifier l'amélioration résultant de l'application d'un mouvement donné à la solution courante. Le processus de recherche est répété jusqu'a ce qu'aucune amélioration ne soit possible. Un avantage de la recherche locale est sa moindre utilisation de mémoire. **[2], [3]**

Le passage d'une solution vers une autre se fait par la définition d'une structure de voisinage, qui varie selon problème à résoudre. On définit l'espace de recherche comme l'espace dans lequel la recherche locale s'effectue. Cet espace peut correspondre à l'espace des solutions possibles du problème étudié. Habituellement, chaque solution candidate a plus d'une solution voisine, le choix de celle vers laquelle se déplacer est pris en utilisant seulement l'information sur les solutions voisines de la solution courante, d'où le terme de recherche locale. **[6]**

Une recherche locale garde juste certains états visités en mémoire, le cas le plus simple est celui de *hill-climbing* qui garde juste un état (l'état courant) et l'améliore itérativement jusqu'à converger à une solution. Le cas le plus élaboré est celui des algorithmes génétiques qui gardent un ensemble d'états (population) qu'il fait évoluer jusqu'à obtenir une bonne solution. **[1]**

En général, il y a une fonction objective à optimiser, dans le cas de hill-climbing. Cette fonction permet de déterminer l'état successeur. Dans le cas des algorithmes génétiques, on l'appelle la fonction de fitness qui intervient dans le calcul de l'ensemble des états successeurs de l'état courant. En général, une recherche locale ne garantie pas de solution optimale. Son attrait est surtout sa capacité de trouver une solution acceptable rapidement. **[1]**

Recherche par escalade (Hill-Climbing)

C'est une méthode de recherche locale qui consiste à se déplacer dans l'espace de recherche toujours en choisissant la meilleure solution parmi voisine. On démarre d'une solution si possible bonne et on balaie l'ensemble de ses voisins. S'il n'existe pas de voisin meilleur que notre solution, on a trouvé un optimum local et on arrête. Sinon, on choisit le meilleur des voisins et on recommence. La convergence vers un optimum local pouvant être très lente, on peut éventuellement fixer le nombre d'itérations maximum, si on veut limiter le temps d'exécution. **[5], [7].**

L'inconvénient de cette méthode gloutonne est qu'elle suppose quel l'optimisation locale d'une solution mène à l'optimum global, or ce n'est généralement pas le cas. La recherche se heurte au premier (et seul) optimum qu'elle trouve. Selon le paysage des solutions, l'optimum local peut être très bon ou très mauvais par rapport à l'optimum global. Si la solution de départ est donnée par une heuristique déterministe, l'algorithme sera déterministe. Si elle est tirée au hasard, on a un algorithme non déterministe et donc plusieurs exécutions différentes sur la même instance pourront donner des solutions différentes et de qualités différentes. **[7]**

Si les voisins sont très nombreux, il y a de fortes chances de trouver l'optimum global mais visiter un voisinage peut être très long, on visitera une grande partie de l'espace des solutions. Si le voisinage est très restreint, on risque de rester bloqué dans un optimum local.

On distingue différents types d'escalade en fonction de la stratégie de génération de la solution de départ et du parcours du voisinage : l'escalade déterministe, l'escalade stochastique et l'escalade vers le premier meilleur (first best).

```
S : ensemble de solution
X0 : solution initiale
X0+1 : solution voisine

1: Répéter :
2: Choisir X0+1 dans V(X0)
3: Si f (X0+1) <f (X0) alors X0 = X0+1
4: Jusqu'à ce que f (X0+1) >= f (X0), ∀ X0+1∈ S
5: Fin.
```

Algorithme I.1 : Algorithme de la recherche par escalade

Méthode de Recuit Simulé (simulated annealing)

Le Recuit Simulé est une méthode de recherche locale qui est une amélioration de l'algorithme hill-climbing pour minimiser le risque d'être piégé dans des optimums locaux. Au lieu de chercher le meilleur voisin immédiat du nœud courant, on accepte avec une certaine probabilité un mouvement vers voisin immédiat moins bon, en espérant d'échapper à l'optimum S'il est local. Cette probabilité diminue avec le temps, ce qui permet de revenir à un optimum quitté s'il est global.

En partant au hasard d'une solution quelconque, on tire arbitrairement une solution voisine. Si elle est meilleure, elle est adoptée ; sinon elle est acceptée avec un coût ; La probabilité d'acceptation d'une solution est fonction du coût engendré et du temps écoulé depuis le lancement de l'algorithme : au début, on accepte facilement un changement qui produit une solution plus coûteuse comparé à la fin où cela devient peu probable. **[1], [2]**

Méthode de recherche Taboue

La recherche avec tabou est une méthode de recherche locale qui a été proposée par Fred Glover en 1986. Le recuit simulé minimise le risque d'être piégé dans des optimums locaux, mais n'élimine pas la possibilité d'osciller indéfiniment en revenant à un nœud antérieurement visité. La recherche taboue propose un mécanisme pour pallier à ce problème. **[1]**

L'algorithme possède une mémoire permettant de garder trace des N solutions les plus récemment explorées (N étant un paramètre de l'algorithme). A chaque étape, le meilleur voisin est sélection, à condition qu'il ne figure pas dans la liste des solutions récentes, d'où vient le nom « tabou » (interdit). La méthode taboue peut donc être vue comme une généralisation de la recherche d'optimum local (si N=0, on retombe dans le cas du hill-climbing) **[7]**

c. Recherche locale itérée

La méthode de recherche locale itérée repose sur la combinaison d'une méthode de recherche locale et de perturbations. La recherche locale utilisant une structure de voisinage assez simple ne permet pas d'échapper aux optimums locaux. Les perturbations basées sur l'utilisation d'un voisinage plus large permettent de modifier une grande partie de la solution et ainsi de tenter d'échapper aux optimums locaux.

A partir d'une solution initiale donnée, une méthode de recherche locale est appliquée. A partir de la solution obtenue s, une perturbation est effectuée jusqu'à la solution initiale. La méthode de recherche locale est alors appliquée sur cette solution pour l'améliorer. Si la nouvelle solution est meilleure que le précédent, elle sera alors le point de départ de la prochaine itération. Sinon, un voisin différent de s est utilisé. Ce processus est répété tant qu'il existe un voisin améliorant le candidat s. **[14], [15]**

Les points importants de cette méthode sont la génération de la solution initiale, la méthode de recherche locale, la perturbation et enfin le critère d'acceptation.

L'algorithme 1 illustre le schéma général d'une recherche locale itérée. Dans les cas pratiques, Il est nécessaire d'implémenter les procédures, et ce en spécifiant : **[14], [15]**

- si la solution initiale est être générée aléatoirement ou par une méthode constructive gloutonne ;
- le voisinage utilisé par la méthode de recherche locale, souvent très simple pour la plupart des problèmes ;
- le mouvement de perturbation (un mouvement aléatoire dans un voisinage d'un degré plus grand que celui de la recherche locale) ;
- et enfin le critère d'acceptation choisi (par exemple : accepter toute amélioration et rejeter les dégradations).

```
   Entrées: solution initiale s0;
   Sorties: meilleure solution s*;
 1: s* = s0
 2: s = RechercheLocale(s0)
 3: répéter
 4:     s' = Perturbation(s,historique)
 5:     s'' = RechercheLocale(s')
 6:     s = CritèreAcceptation(s,s'',historique)
 7:     si f(s) < f(s*) alors
 8:          s* = s
 9:     finsi
10: jusqu'à critère d'arrêt vérifié
```

Algorithme I.2 : Principe de recherche locale itérée

d. Recherche locale guidée

La Recherche Locale Guidée, proposée par Voudouris en 1997 consiste à utiliser une technique de recherche locale dans laquelle la fonction objectif varie durant le processus de recherche, le but étant de rendre les optimums locaux déjà visités moins attractifs.

La fonction de coût est modifiée avec un ensemble de pénalités. Lorsque la recherche locale a atteint un optimum, les pénalités les plus représentatives dans la solution courante sont augmentées et les autres sont diminuées. Ensuite, une nouvelle recherche locale est lancée en utilisant la fonction de coût modifiée. **[15]**

e. Recherche à voisinage variable

D'une manière analogue à la recherche locale guidée, il est possible de changer la nature du voisinage lorsque la solution est bloquée dans un optimum local. Cette stratégie correspond à la recherche à voisinage variable (Variable Neighborhood Search, VNS).

La recherche à voisinage variable est une métaheuristique qui a été proposée par Hansen et Mladenovic [Hansen and Mladenovic, 2003]. La méthode consiste à changer dynamiquement le

type de voisinage utilisé. Il existe différentes versions de recherche à voisinage variable. La méthode la plus simple consiste à choisir aléatoirement, à chaque itération, un type de voisinage puis à appliquer une recherche locale suivant ce voisinage jusqu'à obtenir un optimum local. Cette procédure est réitérée après une éventuelle perturbation. **[15]**

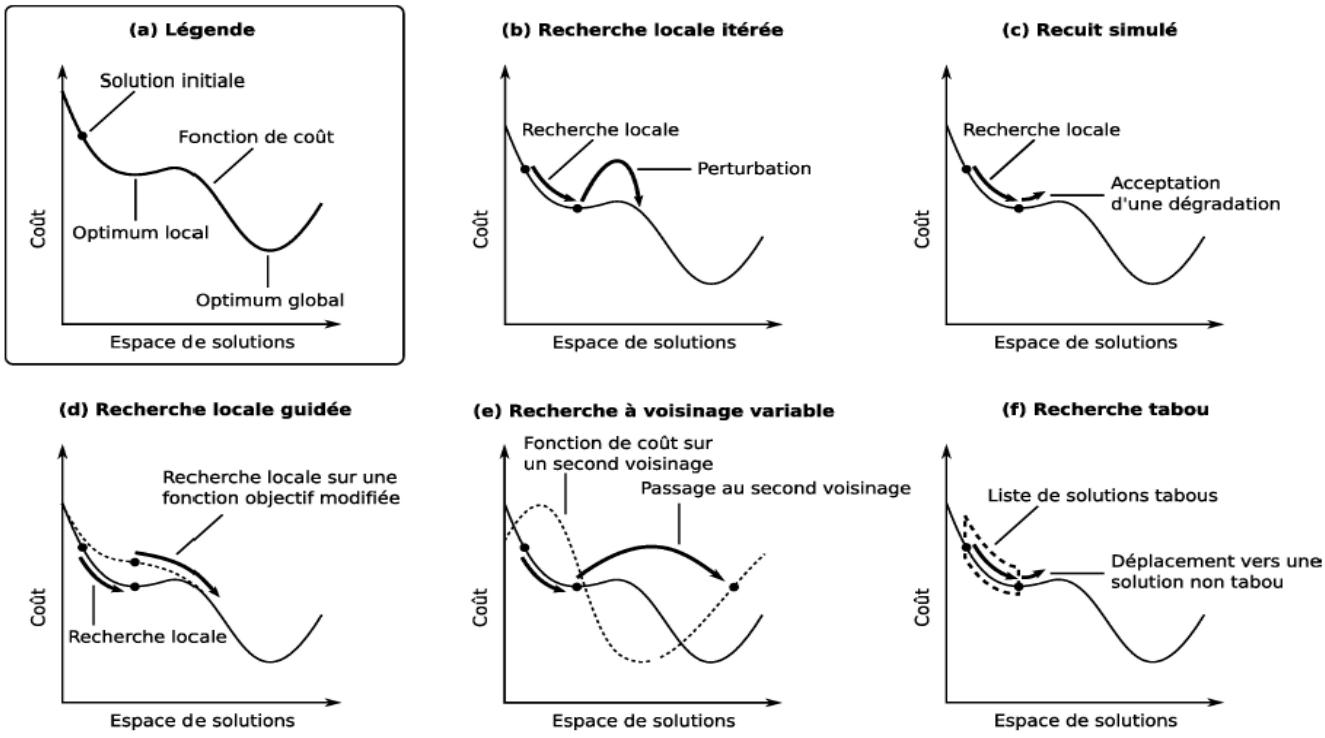

Figure I.4 : Principe de différentes métaheuristiques de trajectoire

I.4.2.2. Les métaheuristiques à base de population

Ces métaheuristiques manipulent une population de solution en les faisant évoluer localement et en les hybridant afin de produire de nouvelles solutions meilleures. Elles travaillent sur un ensemble de points de l'espace de recherche. Cette famille de méthodes sont inspirées de la biologie. Parmi ces algorithmes, on peut citer les algorithmes évolutionnistes comme les Algorithmes Génétiques, les colonies de fourmis etc.

a. Les algorithmes évolutionnistes

Les algorithmes évolutionnistes forment une classe importante des métaheuristiques à base de population. Ils sont fondés sur la métaphore de l'évolution et de la sélection naturelle. Le principe est de faire évoluer une population d'individus (solutions) en appliquant des procédures de croisement, de mutation et de sélection. Chaque nouvelle génération d'individus est obtenue

en croisant et mutant les individus de la génération courante, puis en appliquant une sélection sur la population composée des nouveaux et des anciens individus.

Le principe des approches évolutionnistes est illustré par l'algorithme 2. Dans cet algorithme, P représente la population courante et P' les nouveaux individus obtenus par croisement et mutation. La procédure de croisement consiste à créer de nouvelles solutions en combinant les composantes de solutions initiales. La mutation d'une solution engendre une solution dont les composantes ont été légèrement modifiées par rapport à la solution initiale. Enfin, la sélection consiste à choisir les meilleures solutions pour former la nouvelle génération. Ce choix est souvent réalisé de manière probabiliste en fonction du coût (ou fitness) des solutions.

```
1: Générer la population initiale P
2: Évaluer les individus de P
3: tant que le critère d'arrêt n'est pas atteint faire
4:     Croiser les individus de P pour obtenir P'
5:     Muter les individus de P'
6:     Évaluer les individus de P'
7:    Sélectionner la nouvelle génération P à partir de P et P'
8: fin
```

Algorithme I.3 : Principe des approches évolutionnistes

Il existe trois principales catégories d'approches évolutionnistes : la programmation évolutionniste, les stratégies d'évolution, et les algorithmes génétiques. **[11]**

Algorithmes génétiques

L'opérateur de croisement est considéré comme étant le plus important des opérateurs. La mutation est appliquée avec des probabilités très faibles et agit en tant qu'opérateur d'arrière-plan. Le codage des solutions consiste en une représentation généralement binaire des individus.

b. Stratégies d'évolution

Le codage des solutions peut être réalisé par des structures de données plus complexes que dans les algorithmes génétiques. Par ailleurs, les opérateurs de mutation ont une place aussi importante que les opérateurs de croisement.

c. Programmation évolutionniste

Elle est fondée essentiellement sur l'opérateur de mutation et n'utilise pas d'opérateur de croisement. Comme les Stratégies d'évolution, le codage des solutions peut faire intervenir des structures de données complexes. Cette approche a été développée initialement pour faire évoluer des automates à états finis.

Métaheuristiques	Nombre de solutions utilisées simultanément		Nombre minimum de structures de voisinage exploitées		
	Mono solution	Population	Aucune	Une	Plusieurs
Recherche locale itérée	X			X	
Recherche locale guidée	X			X	
Escalade	X			X	
Recuit simulé	X			X	
Recherche tabou	X			X	
Recherche à voisinage variable	X				X
Algorithme évolutionniste		X	X		

Figure I.5 : Classification des principales métaheuristiques

I.5 Les métaheuristiques hybrides

Les origines des métaheuristiques hybrides reviennent aux travaux de Glover, J. J. Grefenstette et Mühlenbein et al. Les méthodes hybrides sont des méthodes de recherche combinant au moins deux méthodes de recherche distinctes afin de renforcer les avantages des méthodes constitutives et d'affaiblir leurs inconvénients. Cela explique pourquoi il est difficile de trouver un classement exhaustif de toutes les métaheuristiques existantes.

Quelle que soit la méta-heuristique utilisée, elle présente des avantages et des inconvénients. Plusieurs améliorations sont proposées pour obtenir la méthode la plus efficace possible. L'hybridation des méthodes permet de trouver une amélioration car les avantages et les inconvénients de chaque méthode se compensent. Elle exploite la puissance de plusieurs algorithmes, et les combine en un seul méta-algorithme. **[19]**

Les métaheuristiques hybrides sont devenues plus populaires car les meilleurs résultats trouvés pour plusieurs problèmes d'optimisation combinatoires ont étés obtenus avec des algorithmes hybrides. **[17], [19].**

I.6 Les caractéristiques des métaheuristiques

Pour résumer, on peut dire que les propriétés fondamentales des métaheuristiques sont les suivantes : **[11]**

- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale ;
- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales ;
- Les techniques qui constituent des algorithmes de type méta-heuristique vont de la simple procédure de recherche locale à des processus d'apprentissage complexes ;
- Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité ;
- Les métaheuristiques possèdent éventuellement des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace de recherche ;
- Les concepts de base des métaheuristiques peut être décrit de manière abstraite, sans faire appel à un problème spécifique.

- Les métaheuristiques peuvent faire appel à des heuristiques qui tiennent compte de la spécificité du problème traité, et qui contrôlées par une stratégie de niveau supérieur.
- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.

I.7 Conclusion

Dans ce chapitre, nous avons présenté quelques méthodes d'optimisation combinatoires, principalement des métaheuristiques. Ces dernières représentent des modèles généralisés sur lesquels est basé un grand nombre d'implémentations efficace dans la résolution de problème NP-difficiles. Elles sont devenues très populaires grâce à leur simplicité d'emploi dans différents domaines.

L'hybridation de plusieurs métaheuristiques reste une option intéressante lorsque les méthodes connues ne sont pas assez fiables et efficaces dans la résolution d'un problème donné.

Le chapitre suivant est consacré à la définition et la caractérisation du problème d'assemblage de fragments ADN, et montre les technologies anciennes et nouvelles utilisées pour le séquençage d'ADN ainsi l'intérêt du type d'assemblage étudié dans notre travail.

Chapitre II

Le problème d'assemblage de fragments ADN

II.1 Introduction

L'assemblage de fragments ADN (DNA Fragment Assembly Problem, abrégé DNA-FAP) est une partie principale de la méthode du séquençage aléatoire (shotgun sequencing). Ce chapitre présente des concepts importants pour la compréhension du problème d'assemblage de fragments ADN. Il explique de manière générale le processus de séquençage ADN et ses étapes, expose les différentes méthodes et les techniques d'assemblage et met en lumière les caractéristiques de ce problème du domaine de la bio-informatique.

II.2 L'analyse d'ADN

L'analyse d'ADN est devenue de nos jours une pratique indispensable de la police technique et scientifique. Elle permet par exemple d'identifier des victimes, d'établir la présence de suspects sur le lieu du crime ou de vérifier le lien de parenté entre deux personnes. Elle consiste à comparer des segments choisis de molécules d'ADN de différents individus afin de fournir des indices où il n'y a pas de témoins. Elle est également utilisée pour trouver des traitements à des maladies génétiques. En biologie, l'analyse d'ADN est devenue un outil important pour la classification des espèces.

L'objectif principal de l'analyse d'ADN est de déterminer la séquence complète du génome et de son contenu génétique.

II.2.1. Les étapes de l'analyse d'ADN

L'analyse d'ADN passe par 6 étapes : **[21]**

- Duplication de la séquence d'ADN : création de plusieurs copies identiques de la séquence d'ADN.
- Sonification : découpage aléatoire de l'ADN en petits fragments.
- Séquençage : consiste à déterminer l'ordre d'enchaînement des nucléotides.
- Appel des bases : la lecture des bases A, T, C et G à partir des résultats de séquençage (électrophorégramme) ;
- Agencement (layout) : positionnement des fragments selon les chevauchements calculés ;
- Consensus : détermination de l'ADN final.

Les trois premières étapes de séquençage de l'ADN se produisent dans le laboratoire où de grands fragments d'ADN sont dupliqués puis fragmentés en petits fragments qui sont séquencés individuellement (méthode de Sanger).

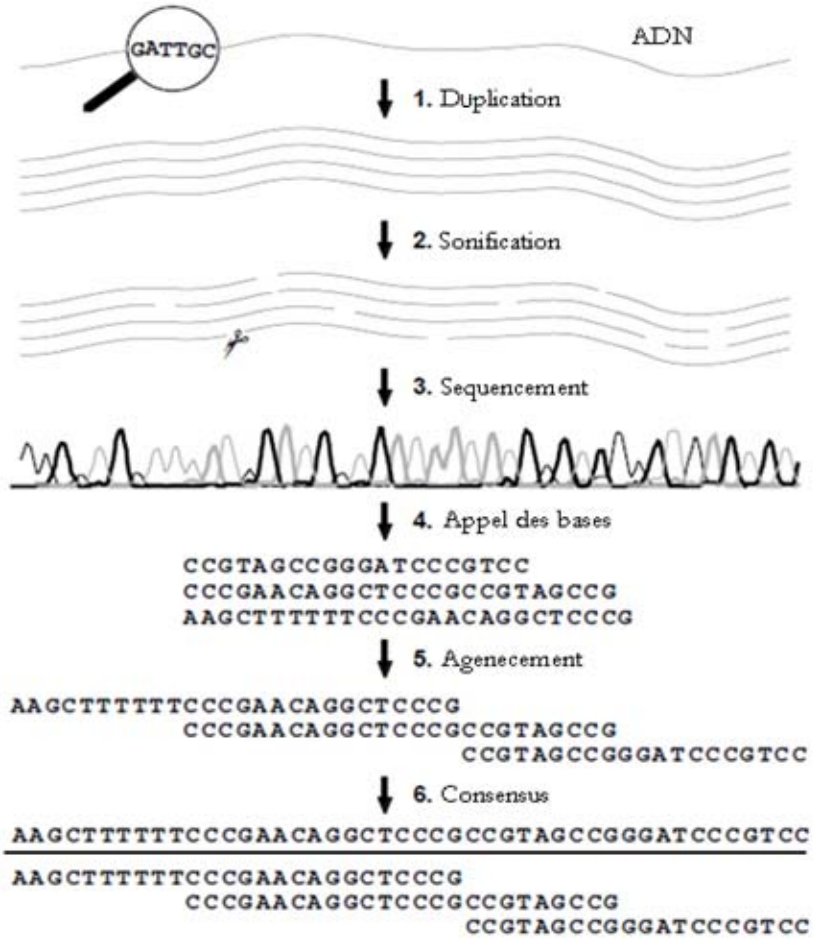

Figure 1 : Les étapes de l'analyse d'ADN **[22]**

II.3. Electrophorèse

L'électrophorèse est une technique permettant de séparer des molécules (d'ADN dans le cas du séquençage) selon leur taille. Cette séparation a lieu sur un gel (gel d'agarose) traversé par un courant électrique. L'ADN étant chargé négativement, la migration est alors possible (vers la borne positive) grâce au courant électrique. La plus petite taille de fragment est celle qui migre le plus loin sur le gel et inversement. **[23]**

II.3.1. Principe électrophorèse sur gel

L'ADN est une molécule acide, dans un tampon pH 8,3 elle sera chargée négativement. Le principe est de faire migrer l'ADN dans un gel d'agarose immergé dans un tampon soumis à

un courant électrique. Dans ces conditions, les molécules d'ADN se dirigent vers le pôle positif (anode) plus ou moins vite en fonction de la taille des fragments. Plus les fragments sont petits, plus ils se déplacent rapidement dans le gel. Après migration, l'ADN est coloré par l'Azure A pour distinguer les quatre bases azotées A, T, C et G.

La vitesse de migration des fragments sera déterminée par :

- la résistance du gel à la migration des particules, proportionnelle à leur taille ;
- la charge des fragments, proportionnelle à leur taille ;
- la concentration du gel en agarose.

Les fragments d'ADN vont donc se séparer en fonction de leur taille. Si le facteur le plus important est la résistance du gel a la migration de l'ADN, ce sont les plus petits fragments qui migreront le plus loin.

C'est une technique très utilisée dans la détection des maladies génétiques. Une mutation dans une séquence d'un gène peut supprimer ou ajouter un site de restriction et ainsi entraîner la formation d'un segment plus grand ou de deux segments à la place d'un. **[23]**

II.4. Séquençage de fragments d'ADN

Séquencer un brin d'ADN, revient à lire l'enchaînement, ou séquence, des nucléotides constitutifs de cette molécule. En effet, on détermine la succession des bases nucléiques. Les méthodes de séquençage actuelles ne permettent pas de séquencer un grand nombre de bases. Or la séquence du génome humain comprend environ 3,2 milliards de paires de bases. Il sera donc impossible de séquencer d'un coup la totalité du génome. Les biologistes sont incapables de manipuler des molécules d'ADN d'un million de bases. Pour obtenir suffisamment de séquences chevauchantes et pour réduire les erreurs de séquençage, il faut atteindre un certain niveau de redondance, c'est-à-dire produire une quantité de séquences aléatoires représentant plusieurs fois la longueur de la séquence d'intérêt. Ceci conduit à un nombre très important de séquences à réaliser. Si par exemple 10 copies de la séquence d'ADN sont disponibles, chaque base de la séquence cible a été lue 10 fois en moyenne. Même à 10X, des "trous" peuvent donc subsister, laissant la séquence finale très légèrement incomplète. **[24]**

Le processus de séquençage d'un génome est basé sur la fragmentation aléatoire pour obtenir des morceaux d'ADN, de quelques paires de bases, faciles à manipuler. Ces petits fragments sont alors séquencés individuellement. La séquence complète du génome est

reconstruite à partir de ces séquences unitaires sur la base des chevauchements. Les fragments ayant une grande partie chevauchante sont plus d'ADN dont elles dérivent ont une partie de leur longueur en commun. La fragmentation étant aléatoire, les molécules d'ADN de l'échantillon ne sont pas toutes fragmentées aux mêmes endroits. **[24]**

On distingue deux méthodes de séquençage des génomes entiers. Dans les deux cas, l'ADN génomique est tout d'abord fragmenté par des méthodes soit enzymatiques (enzymes de restriction), soit physiques (ultrasons) : **[25]**

- La stratégie de séquençage ordonnancement hiérarchique ("shotgun hiérarchique") ou clone par clone **:** consiste à classer les fragments génomiques obtenus avant de les séquencer.
- la stratégie du séquençage aléatoire global ("whole genome shotgun") **:** ne fait pas de classement des fragments obtenus mais les séquences sont dans un ordre aléatoire. L'analyse permet ensuite de réordonner ces fragments génomiques par chevauchement des séquences communes.

La principale différence est que l'ordonnancement hiérarchique essaie d'aligner un jeu de clones de grande taille (~ 100 kb) alors que, dans la méthode globale, le génome entier est réduit en fragments de petite taille, séquences puis alignés.

Ces deux approches reposent sur le principe de séquençage de la méthode de Sanger.

a. La stratégie du séquençage aléatoire global [25]

Cette stratégie a été utilisée dès les débuts du séquençage, en 1982. C'est la stratégie retenue aujourd'hui pour tous les génomes bactériens qui sont de petite taille (de l'ordre de plusieurs millions de nucléotides) et n'ont pas beaucoup de séquences répétitives. Il s'agit d'une méthode de séquençage d'ADN génomique fragmenté, mise au point par le groupe privé Cèlera Genomics sur des génomes bactériens, puis sur le génome de la Drosophile et enfin sur les génomes de l'Homme et de la Souris.

On fragmente le génome de façon aléatoire par sonication, on sélectionne par électrophorèse des fragments de 1 à 2 kb et on les insère dans un plasmide (vecteur de clonage). Les extrémités d'une partie de ces fragments sont ensuite séquencées, puis ces séquences sont assemblées sur la base de leurs chevauchements pour essayer de produire une séquence complète.

On obtient ainsi des blocs de séquences continues ou « contigs », entre lesquels il reste des brèches. La séquence totale est réalisée à force de recouvrements et d'assemblages.

Le problème est cependant plus complexe pour les grands génomes comme le génome humain, du fait du nombre énorme de fragments à séquencer (70 millions pour le génome humain, trois ans de travail) et des difficultés d'assemblage en particulier du fait de la présence de séquences répétées.

b. La stratégie clone par clone [25]

Cette stratégie a été adoptée par plusieurs consortiums internationaux pour le séquençage du génome humain et d'autres génomes de grande taille. L'ADN génomique est après extraction, fragmenté, généralement par des procédés de sonication, en fragments maniables (c'est-à-dire clonables dans des grands vecteurs) de 50 à 200 kb, puis cloné dans un vecteur adapté comme les chromosomes artificiels bactériens. Le nombre de clones doit permettre une redondance de 5 à 10 fois le génome total. Le chevauchement et l'ordonnancement des clones est alors réalisé soit par l'utilisation des profils de restriction c'est-à-dire on digère les clones grâce à des enzymes de restriction, puis on recherche si différents clones présentent des fragments de même taille, Il y a alors de bonnes chances que ces clones possèdent une région génomique en commun, que les enzymes de restriction coupent aux mêmes endroits; soit par l'hybridation des clones entre eux : si deux clones peuvent s'hybrider, c'est qu'ils possèdent des séquences communes.

Une fois tous les grands clones ordonnés et positionnés le long du génome humain, on dispose d'une carte physique du génome. On peut alors sélectionner un ensemble minimal de grands clones chevauchants (un chemin de recouvrement) en vue de les séquencer.

Après avoir effectué cet ordonnancement des clones (donc des fragments génomiques), les clones sélectionnés sont individuellement fragmentés, séquences et assemblés.

Les avantages de l'ordonnancement hiérarchique sont de deux ordres :

- une plus grande facilité d'assemblage des séquences grâce aux données des cartes physiques, génétiques et aux chevauchements des BACs.
- une ouverture vers un travail en collaboration entre plusieurs laboratoires, chacun séquençant une région de chromosome.

L'inconvénient majeur est la présence, à la fin, de portions d'ADN absentes de l'analyse à cause généralement des difficultés de clonage de certaines séquences répétées très fréquentes.

II.4.1. Historique des technologies de séquençage

Le séquençage de l'ADN a été inventé dans la deuxième moitié des années 1970. Deux méthodes ont été développées indépendamment, l'une par l'équipe de Walter Gilbert, (Maxam et Gilbert 1977) aux États-Unis, et l'autre par Frederick Sanger (Sanger, Nicklen et al. 1977) en Grande-Bretagne. Ces deux méthodes sont fondées sur des principes diamétralement opposés : l'approche de Sanger est une méthode par synthèse enzymatique sélective, tandis que celle de Maxam et Gilbert est une méthode par dégradation chimique sélective. Pour cette découverte, Gilbert et Sanger ont été récompensés par le prix Nobel de chimie en 1980 (Kolata 1980).

C'est toutefois l'automatisation de la méthode Sanger à la fin des années 80, avec le développement des marquages fluorescents et de l'électrophorèse capillaire, qui a ouvert la voie du séquençage à haut débit. Par ailleurs, la première méthode non-Sanger ayant fait ses preuves dans le domaine de séquençage a été introduite en 1988 Ronaghi, Uhlen et al. 1998, Hyman et al. 1988), il s'agit de pyroséquençage, une technique principalement basée sur l'addition d'un seul nucléotide qui est révélé en temps réel par détection de la luminescence.

Le début du 21 siècle est marqué par l'avènement de la nouvelle génération des techniques de séquençage, il s'agit en faite de méthodologies nouvelles ou adaptées de séquençage, découlant des avancées technologiques issues de l'évolution des connaissances en physique, informatique, chimique, nanotechnologie et en biotechnologie. Ces innovations technologiques visent à réduire le coût et le temps nécessaires pour le séquençage génome complet. Deux exemples concernant des technologies novatrices ayant fait leurs preuves dans le séquençage à haut débit de génome complet, à savoir la technologie 454 et la technologie Solexa/Illumina. **[28]**

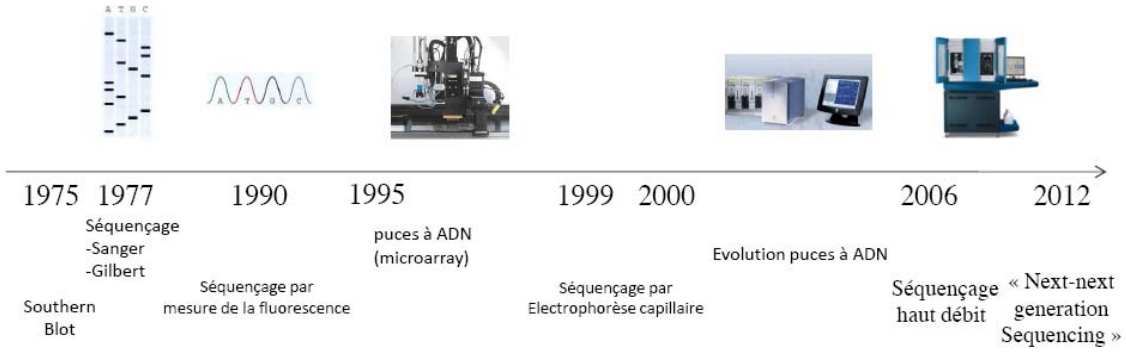

Figure 2 : Historique des technologies de séquençage. **[27]**

II.4.2. Les premières technologies de séquençage

a. Principe de la Méthode de Maxam & Gilbert (1977)

Cette méthode utilise des réactifs qui détruisent des bases nucléotidiques spécifiques. Elle est basée sur une dégradation chimique de l'ADN et utilise les réactivités différentes des quatre bases A, T, G et C, pour réaliser des coupures sélectives. En reconstituant l'ordre des coupures, on peut remonter à la séquence des nucléotides de l'ADN correspondant. **[28]**

On peut décomposer ce séquençage chimique en six étapes successives :

- Marquage : Les extrémités des deux brins d'ADN à séquencer sont marquées par un traceur radioactif (^{32}P). Cette réaction se fait en général au moyen d'ATP radioactif et de polynucléotide kinase.
- Isolement du fragment d'ADN à séquencer : Celui-ci est séparé au moyen d'une électrophorèse sur un gel de polyacrylamide. Le fragment d'ADN est découpé du gel et récupéré par diffusion.
- Séparation de brins : Les deux brins de chaque fragment d'ADN sont séparés par dénaturation thermique, puis purifiés par une nouvelle électrophorèse.
- Modifications chimiques spécifiques : Les ADN simple-brin sont soumis à des réactions chimiques spécifiques des différents types de base. Ces différentes réactions sont effectuées dans des conditions très ménagées, de sorte qu'en moyenne chaque molécule d'ADN ne porte que zéro ou une modification.
- Coupure : Après ces réactions, l'ADN est clivé au niveau de la modification par réaction avec une base, la pipéridine.
- Analyse : Pour chaque fragment, les produits des différentes réactions sont séparés par électrophorèse et analysés pour reconstituer la séquence de l'ADN. Cette analyse est analogue à celle que l'on effectue pour la méthode de Sanger. **[28]**

b. Principe de la Méthode de Sanger et Nicklen (1977)

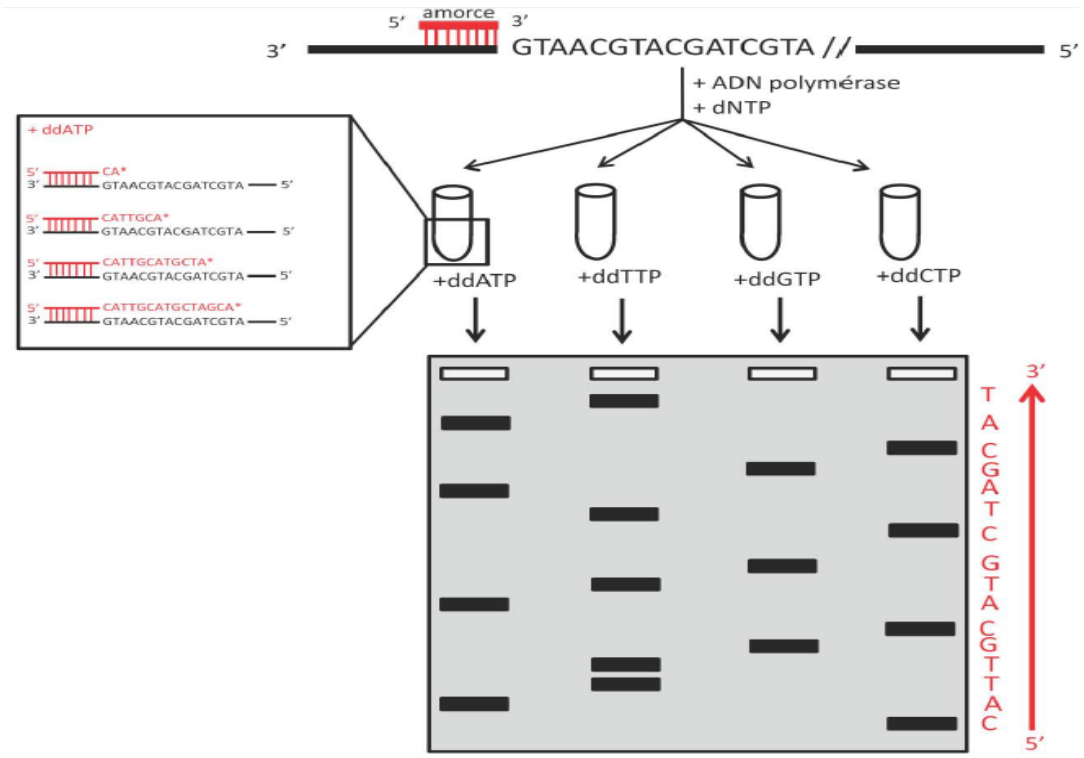

Figure 3 : Principe du séquençage selon la méthode de Sanger. **[26]**

Cette méthode est basée sur l'interruption de la synthèse enzymatique d'un brin d'ADN complémentaire. L'ADN à séquencer est cloné et de nombreuses molécules d'ADN simple brin sont produites. Une courte amorce d'oligonucléotides (généralement synthétisée chimiquement et éventuellement marquée) est fixée à l'ADN et sert de point de départ pour la synthèse du brin complémentaire. La polymérase est alors ajoutée avec les 4 nucléotides normaux : d-ATP, d-CTP, d-GTP et d-TTP et une faible concentration de 4 nucléotides analogues dans des incubations séparées. Les analogues sont des didésoxynucléotides (ddNTP), dont l'ajout à l'ADN permet d'arrêter la polymérase, dû l'absence du groupement OH, nécessaire à l'extension du brin complémentaire.

Les 4 incubations contiennent donc un mélange de fragments partiellement synthétisées d'ADN double brin marqué. La longueur des fragments d'ADN varie en fonction du point d'intégration du didésoxynucléotide. Comme cette intégration est aléatoire, l'ensemble des molécules dans un mélange représente l'ensemble des positions pour une base particulière. Les 4 mélanges sont analysés simultanément sur un gel d'électrophorèse. Celui-ci contient un composé qui entraîne la dénaturation de l'ADN double brin et le processus est mené sous un voltage fort

pour éviter la réassociation des brins. Comme pour la méthode précédente, les bandes sont révélées par autoradiographie et la séquence est lue directement sur le gel.

Une adaptation de cette technique consiste à marquer les didésoxynucléotides par un fluorophore spécifique, permettant ainsi de n'avoir qu'un seul mélange au lieu de 4. Les fragments d'ADN synthétisés portent ce fluorophore terminal appelé terminateur d'élongation. Cette approche est proposée par Smith, Sanders et al. (1986). C'est la technique actuellement utilisée dans certains séquenceurs automatisés. **[28]**

1. Comparaison des deux méthodes :

	Methode Maxam et Gilbert	**Methode de Sanger**
Principe	méthode chimique	méthode enzymatique
Avantages	• Conduit à moins d'erreurs car on n'utilise pas d'enzymes. • Lecture plus rapide car on a directement la séquence du brin matrice	• Plus rapide • Moins lourde • Simplifiée avec l'utilisation de kits de séquençage • Utilisée en automate
Inconvénients	• Technique lourde coûteuse • Utilisation de produits chimiques comme le DMS, l'hydrazine.	• Possibilité d'erreurs dans la polymérisation • Lecture indirecte : la séquence produite est celle complémentaire à l'ADN matrice

Tableau 1 : Comparatif des méthodes de Sanger et de Maxam et Gilbert

Au cours des années, la technique de Sanger s'est imposée, principalement parce qu'elle était plus facile à automatiser.

c. Automatisation de la technique de Sanger

La méthode de Sanger a été largement développée grâce à plusieurs avancées technologiques importantes : **[28]**

- La mise au point de vecteurs de séquençage adaptés, comme le phage M13 développé par Joachim Messing au début des années 1980 (Messing 1983).
- Le développement de la synthèse chimique automatisée des oligonucléotides qui sont utilisés comme amorces dans la synthèse. L'introduction de traceurs fluorescents à la place des marqueurs radioactifs utilisés initialement.
- Ce progrès a permis de sortir le séquençage des pièces confinées, réservées à l'usage des radioisotopes.
- L'adaptation de la technique PCR (polymerase chain reaction) pour le séquençage.
- L'utilisation de séquenceurs automatiques de gènes.
- L'utilisation de l'électrophorèse capillaire pour la séparation et l'analyse.

On utilise 4 marqueurs fluorescents liés aux 4 ddNTPs. Les quatre réactions de séquençages peuvent donc être conduites dans le même tube puisque les marquages sont différents.

d. Séquenceur capillaire

Dans les séquenceurs capillaires (augmentation de la résolution et rapidité de séparation), et la détection se font en temps réel. Les bandes en migrant passent devant une fenêtre fixe où elles sont excitées par un laser. La lumière émise est enregistrée et le résultat affiché sous forme d'un graphe où figure aussi la séquence.

Ces séquenceurs fonctionnent sur 96 canaux ce qui permet d'obtenir 96 séquences en parallèle en moins de 2h et les plus récents fonctionnent sur 384 voire 1024 canaux.

Les séquenceurs capillaires ont apporté une plus grande automatisation. La technique de Sanger est celle qui est mise en œuvre dans les premiers séquenceurs automatiques (Le premier séquenceur automatisé a été mis au point en 1987, par la compagnie Applied Biosystems - ABI). L'automatisation requiert en général l'emploi :

- d'un système d'électrophorèse piloté par ordinateur ;
- des marqueurs fluorescents de différentes couleurs qui sont révélés après excitation par un laser à l'aide d'une caméra CCD.

- des logiciels permettant l'analyse des signaux sortant de l'appareil et leur mise en forme sous forme de résultats (électrophorégramme et séquence).
- d'un robot passeur permettant d'enchaîner les échantillons les uns à la suite des autres (notamment passage de plaques de réaction à 96 puits (12x8)).

La méthode de Sanger automatisée est considérée comme étant la stratégie de séquençage haut-débit de première génération. **[28]**

e. Le pyroséquençage

Depuis 7 ans, de nouvelles méthodes ont été mises au point parmi lesquelles le pyroséquençage haut-débit, qui est de loin la technique non Sanger qui a connu le plus de succès. introduite depuis 1988, par Hyman et al., (Hyman 1988) et amélioré par un groupe suédois (Ronaghi et al. , 1996 – 1998), le pyroséquençage permet un séquençage rapide et à moindre coût qu'un séquençage par la méthode de Sanger. En effet, cette technique ne nécessite pas de clonage et permet une lecture directe de la séquence obtenue après le séquençage. Le pyroséquençage présente par ailleurs des limitations dont la principale est liée à son utilisation pour le séquençage des génomes complets, puisque la longueur de la séquence lue ne dépasse pas les 100 pb. **[28]**

II.4.3. Nouvelles technologies de Séquençages à haut débit

Le terme « Next generation sequencing » regroupe l'ensemble des technologies ou plateformes de séquençage développées depuis 2005 par quelques sociétés de biotechnologies. Deux technologies élégantes qui sont disponibles sur le marché, et qui sont représentative de la nouvelle génération des séquenceurs à haut débit sont : la technologie 454 et CRT Solexa/Illumina.

a. La technologie 454 Life Sciences

Cette technique permet de traiter avec un seul instrument le GS20, plus de 20 millions de bases nucléotidiques par cycle de quatre heures, ce qui correspond à plus de 100 fois la capacité des instruments reposant sur les techniques actuelles à l'échelle macroscopique. Une machine GS20 génère avec plus de 99% d'exactitude, autant de données que 100 séquenceurs capillaires à haut débit.

La technologie de 454 Life Sciences est fondée sur l'intégration de plusieurs techniques : le pyroséquençage, les technologies des plaques en fibre optique picolitré qui contient 1.6 millions

de puits (1 pL = 10^{-12} L), la PCR en émulsion (emPCR) dans des microréacteurs (300.000 réactions PCR en parallèle), ainsi que des technologies informatiques de pointe pour l'acquisition, le traitement et l'analyse des images.

b. La technologie CRT Solexa/Illumina

Pour cette technique, l'amplification de l'échantillon à analyser ne s'effectue pas en solution mais sur un support solide. La réaction de séquençage est alors réalisée directement sur le support où l'ADN a été amplifié. Elle se déroule position après position en ajoutant un mélange contenant toutes les bases associées chacune à un fluorophore différent. L'extrémité de ces bases est protégée pour empêcher l'addition de bases supplémentaires à chaque cycle d'incorporation. Une lecture laser permet alors de détecter simultanément toutes les positions incorporées. Le clivage des fluorophores permet ensuite l'incorporation de la base suivante. La lecture est effectuée ainsi cycle après cycle. Cette méthode permet l'acquisition en parallèle de plus de 3 milliards de séquences de 100 bases de long. Chaque position étant lue l'une après l'autre, les erreurs principales de cette technologie sont des erreurs de substitution d'une base par une autre. **[29]**

II.4.4. Evolution du séquençage [26]

Les progrès techniques apportés ces dernières années, le séquençage est devenu plus rapide et plus efficace. La plus importante amélioration apportée par les nouvelles technologies de séquençage est la quantité de données qui peuvent être générées en une seule réaction. En effet, il est possible de séquencer plusieurs millions d'échantillons en une seule fois, 384 séquences au maximum peuvent être générées avec les séquenceurs les plus récents utilisant la méthode de Sanger automatisée. Au fur et à mesure que de nouvelles techniques de séquençage sont mises au point, la quantité de données générées augmente sensiblement alors que le temps nécessaire à un séquençage continue de diminuer.

II.5. L'assemblage de fragments ADN

L'assemblage de fragments ADN est un processus qui permet de trouver la chaîne la plus courte contenant les séquences issues de la fragmentation d'un ADN initial. Le programme informatique qui réalise ce travail est appelé assembleur. Sa tâche est de combiner les lectures obtenues après séquençage en séquences contigües (ou contigs), et ce sur la base de chevauchements, puisque deux séquences chevauchantes proviennent vraisemblablement de la même région de l'ADN d'origine.

II.5.1. Les difficultés de l'assemblage

Le processus d'assemblage présente éventuellement plusieurs difficultés. Ces erreurs inévitables qui se produisent durant l'étape de séquençage sont classées comme suit : **[21]**

- Orientation inconnue : une fois la séquence d'ADN d'origine est fragmentée, l'orientation des fragments est perdue et l'extrémité à sélectionner devient inconnue. Si deux fragments ne se chevauchent pas, il est toujours possible que leurs inverses marquent un chevauchement ;
- Erreurs d'appel de bases : elles se produisent suite à une fausse interprétation des résultats d'un séquençage (électrophorégramme). Il existe trois types d'erreurs d'appels de base : des substitutions, des insertions et des suppressions de bases.
- Couverture incomplète : l'ADN résultant contient des régions vides (incomplètes). Ce phénomène se produit lorsqu'il n'est pas possible de réunir un ensemble de fragments en un seul contig (de l'anglais *contiguous sequence*). Les fragments qui possèdent une séquence commune chevauchante, assez grande, étant donné un seuil minimal (threshold), font partie d'un même contig.
- Régions répétées : pour un assembleur deux lectures se chevauchant parfaitement proviennent de la même région génomique. Or, certaines séquences peuvent se retrouver de multiples fois dans un génome entrainant la formation de contigs chimériques et de trous dans l'assemblage final. Les assembleurs actuels ne gèrent pas d'une manière assez efficace les longues séquences répétées, ce qui nécessite souvent un post-traitement de l'ADN au niveau laboratoire.
- Chimères : l'apparition de nouveaux fragments suite à une fusion accidentelle de deux fragments non-adjacents sur l'ADN d'origine;
- Contamination : lorsqu'une purification (isolation) de l'ADN n'est pas bien réalisée, on parle de contamination. La purification a un impact direct sur le séquençage, les fragments contaminés doivent donc être éliminés avant l'assemblage.

II.5.3. Types d'assemblage

Afin de mieux comprendre les types d'assemblage, imaginez que quelqu'un nous donne un grand nombre de pièces de puzzle et qu'il nous demande de les assembler, sachant que nous avons toutes les pièces nécessaires pour former une image complète. Si on nous laisse sans

indices, nous irons simplement chercher les pièces qui se complètent et les joindre pour former une image. Mais si on nous donne un aperçu de l'image finale, alors la résolution du puzzle sera moins difficile et plus rapide.

L'assemblage de fragments ADN ne diffère pas trop de l'assemblage des pièces d'un puzzle. On distingue deux types d'assemblage de séquences ADN : l'assemblage par cartographie (mapping) et l'assemblage de novo.

a. Assemblage par cartographie

Tout assemblage se basant sur un génome de référence est un assemblage par cartographie (mapping). Son principe est de comparer les fragments issus de la phase de séquençage avec une séquence ADN déjà connue. Ce génome doit avoir de grandes ressemblances avec l'ADN que l'on cherche à séquencer. Trouver une solution à ce problème revient donc à chercher, pour chacun des fragments la position qui marque un chevauchement maximal avec le génome de référence.

L'un des plus grands exemples d'application d'un assemblage par cartographie est dans la généalogie. Le génome humain est composé d'environ 3.2 milliards de bases nucléiques, où seulement 1% de cette séquence diffère d'une personne à une autre. Pour rechercher les liens de parenté entre deux individus A et B, il est plus intéressant de réaliser une cartographie d'ADN afin de pouvoir mesurer le degré de ressemblance et de savoir si A est le parent de B ou pas.

b. Assemblage de novo

L'assemblage de fragments ADN sans aucun génome de référence est appelé **de novo.** Ce type d'assemblage se base sur la vérification des chevauchements deux à deux entre les fragments. Trouver la solution exacte est équivalent à effectuer un nombre de comparaison exponentiel par rapport au nombre de fragment, une chose irréalisable en un temps abordable sauf dans le cas de petites instances du problème. C'est pour cette raison qu'on emploie des métaheuristiques pour arriver à des résultats plus proches, éventuellement identiques aux résultats recherchés.

Un assemblage de novo est le plus utilisé car le nombre de génomes séquencé est de loin inférieur au nombre d'espèces connues. Il est utile dans le cas où l'ADN provient d'une source inconnue (exemple : des cheveux trouvés sur une scène de crime … difficile de savoir s'ils proviennent d'un humain, d'un chien , d'un chat ou autre être vivant).

II.5.4. Les approches connues pour un assemblage de novo

a. Chevauchement–Agencement–Consensus

Overlap-Layout-Consensus (abrégée OLC) est l'une des premières méthodes utilisées pour assembler une séquence génomique avec succès (Miller et al. 2010). Son principe est de fusionner des séquences chevauchantes, en commençant par celles ayant les scores les plus élevés, jusqu'à obtention d'une séquence unique. Comme son nom l'indique, elle est constituée des 3 phases : **[26]**

1. **Chevauchement (overlap) :** elle sert à calculer le meilleur alignement possible entre deux lectures en tenant compte des éventuelles erreurs de séquençage. Chaque alignement est pondéré par un score qui va tenir compte du pourcentage d'identité nucléotidique et de la longueur de cet alignement entre deux lectures.
2. **Agencement (layout) :** Cette phase consiste à des fragments basé sur le score de similarité calculé. Pour la plupart des assembleurs, des paires de lectures sont sélectionnées en fonction de leur score (les scores les plus élevés en premier) et, si l'alignement est considéré comme cohérent, les deux séquences sont fusionnées pour former un contig. Il s'agit de l'étape la plus difficile.
3. **Consensus :** la dernière phase dite de *consensus*, sert, comme son nom l'indique, à établir la séquence consensus des contigs obtenus lors de l'étape de *layout*. La méthode la plus simple est de déterminer à chaque position quelle base est la plus représentée.

b. Recherche du super-chemin eulérien dans un graphe de de Bruijn

Cette méthode d'assemblage a été mise au point pour s'affranchir de l'étape complexe de *layout* utilisée par la méthode *overlap-layout-consensus* (Pevzner et al. 2001). Les lectures qui doivent être assemblées sont préalablement fragmentées en sous-séquences chevauchantes de taille constante. Ces sous-séquences sont appelées k-mer (avec *k* correspondant à la longueur). Deux k-mers vont être assemblés si leurs séquences ne divergent que par le premier nucléotide de l'un et le dernier de l'autre. Ainsi une identité parfaite doit être partagée entre les deux séquences qui se chevauchent. De cette façon il est théoriquement possible d'établir un lien entre chacun des k-mers (la présence des répétitions empêchant ce cas idéal dans la pratique) et de reconstituer une séquence génomique. **[26]**

La fragmentation des lectures alors que leur longueur est importante pour la résolution des répétitions. Cependant, cette méthode demande peu de ressources computationnelles par

rapport au nombre de lectures qu'il est possible d'assembler. En effet, la recherche de chevauchements par la méthode « overlap-layout-consensus » nécessite de calculer des alignements entre chaque lectures. Ainsi, plus la quantité de données à analyser augmente, plus le temps et les ressources informatiques nécessaires à l'assemblage augmentent. Pour l'approche de « de Bruijn », cette étape de *layout* n'a pas à être opérée pour l'assemblage, car une identité de séquence parfaite doit exister entre deux lectures pour qu'elles se chevauchent. Cette approche d'assemblage est devenue la référence pour assembler des séquences issues de techniques de séquençage produisant un grand nombre de données comme celles d'Illumina/Solexa ou de SOLiD. **[26]**

Toutefois, ce principe d'assemblage est particulièrement sensible aux erreurs car une identité parfaite entre K-mer est requise pour valider un lien. Ainsi en pratique, l'assemblage de séquences par cette méthode demande des lectures possédant peu d'erreurs pour être efficace.

c. Recherche de la plus courte super-chaîne commune (Shortest Common Supersequence – SCS)

Cette méthode consiste à trouver la super séquence la plus courte d'un ensemble de séquences. Tout d'abord, un score d'appariement est calculé pour chaque paire de séquences. Puis les paires appariables sont assemblées suivant l'ordre décroissant des scores.

La méthode SCS présente tout de même des inconvénients majeurs : **[30]**

- Ne tolère pas les erreurs ;
- Les orientations des fragments doivent être connues ;
- La plus courte chaîne n'est pas toujours celle recherchée par les biologistes dans de séquences à régions fréquemment répétées ;

II.6.5. Facteurs à prendre en compte par un assembleur

L'utilisation d'un assembleur particulier sera conditionnée par de nombreux paramètres tels que la stratégie de séquençage utilisée, la présence de « paired-end » ou encore la quantité de données. Les assembleurs utilisant la méthode « overlap-layout-consensus » sont plutôt utilisés avec des lectures allant de 100 à 800 pb (principalement avec un séquençage Sanger ou un pyroséquençage haut-débit). La méthode graphique « De Bruijn » s'emploie plutôt avec des lectures d'une taille allant de 25 à 100 pb. Cette dernière est la méthode d'assemblage la plus utilisée à l'heure actuelle grâce au succès de la plateforme Illumina/Solexa. Toutefois, avec les

améliorations des techniques de séquençage, la taille des lectures ne cesse d'augmenter et la méthode « overlap-layout- consensus » devrait redevenir la référence dans le futur. **[26]**

La plupart des assembleurs ont démontré leur efficacité sur l'assemblage de génomes bactériens car ils sont plus simples à assembler que les génomes eucaryotes, notamment par la plus faible longueur de leurs génomes et leur plus faible teneur en répétitions. Il faut toutefois noter que certains assembleurs ont été testés sur des données simulées (Butler et al. 2008) alors que d'autres ont montré leur efficacité sur des données réelles (MacCallum et al. 2009), ce dernier critère étant plus déterminant dans le choix de l'utilisation d'un assembleur. **[26]**

Les deux séquences assemblées pour cette exemple sont découpées en 4-mer chevauchants. Il est à noter que le 4-mer CCGG qui est une répétition n'est représenté qu'une fois par la méthode graphique de « De Bruijn ». Une identité de séquence parfaite entre les 3 premiers nucléotides de l'un et les 3 derniers de l'autre est recherchée entre ces sous-séquences. S'il y a concordance exacte (comme par exemple pour les 4-mer soulignés en rouge), l'assembleur va relier les deux fragments. Le consensus obtenu fournit la séquence initiale. **[26]**

II.7. Conclusion

Nous avons vu dans ce chapitre que les méthodes destinées à la résolution du problème d'assemblage de fragments ADN sont multiples et varient selon la caractérisation des génomes séquencés, le développement des technologies de séquençage, et l'utilité du séquençage d'un génome spécifique.

Les chapitres de la partie II parlent en détail des solutions d'assemblage étudiées, et de la nouvelle approche introduite par cet ouvrage. Le prochain chapitre est consacré aux processeurs graphiques et au calcul GP-GPU.

Chapitre III

CUDA et le calcul sur processeur graphique

III.1 Introduction

Les processeurs graphiques (GPU), dont l'utilisation était jadis destinés à un usage restreint aux applications graphiques tels que le dessin 3D et les jeux vidéos, trouvent de plus en plus d'intérêt majeur de la part des scientifiques et chercheurs. Leur immense capacité à traiter en parallèle des collections de données très larges en fait un outil idéal pour tout processus de calcul haute performance (HPC). Ce chapitre introduit d'abord quelques notions sur le calcul sur processeurs graphiques, puis prend en détails le cas de CUDA, l'architecture des GP-GPU créée par nVIDIA.

III.2 Historique

Depuis l'apparition dans les années 80 des accélérateurs graphiques matériels, leur évolution n'a cessé de s'accélérer. Les premières cartes graphiques accessibles au grand publiques sont apparues dans le milieu des années 90. C'est Nvidia qui a introduit le terme GPU, remplaçant le terme VGA. **[49]**

Pendant trente ans, l'un des principaux moyens d'améliorer les performances des ordinateurs a consisté à accroître la fréquence d'horloge des processeurs. Trente ans plus tard, les fréquences d'horloge de la majorité des processeurs d'ordinateurs étaient entre 1 GHz et 4 GHz, mille fois plus que celle du premier ordinateur personnel. Au cours des dernières années, cependant, les constructeurs ont été contraints de rechercher d'autres alternatives car plusieurs limites fondamentales des circuits intégrés font qu'il n'est pas possible de se contenter d'augmenter infiniment la fréquence d'horloge d'un processeur. La limite physique de la taille des transistors, ont donc imposé la recherche d'autres solutions. Les améliorations apportées aux supercalculateurs suggèrent une excellente question dans la recherche de puissance supplémentaire des ordinateurs, au lieu de se contenter d'augmenter les performances d'un unique cœur de traitement, pourquoi ne pas en mettre plusieurs ? De cette façon, les ordinateurs pourront continuer à améliorer leurs performances sans devoir continuer à augmenter leurs fréquences d'horloge. **[34]**

Aujourd'hui, les principaux constructeurs de microprocesseurs ont déjà annoncé qu'ils prévoyaient de produire des CPU de 12 et 16 cœurs, ce qui confirme bien que l'ère de l'informatique parallèle est arrivée dans les cœurs. L'évolution des processeurs centraux en termes de fréquences d'horloge et de nombre de cœurs. Pendant ce temps, le traitement graphique subissait une révolution importante. **[34]**

Au milieu des années 1990, des sociétés comme NVIDIA et ATI Technologies ont commencé à produire des accélérateurs graphiques. Ces développements ont fait du graphisme 3D une technologie qui deviendrait essentielle dans les années à venir. **[34]**

La clé de la réussite du GPU-computing réside dans sa performance énorme par rapport à la CPU, aujourd'hui, il ya un écart de performance d'environ sept fois entre les deux lorsque l'on compare la bande passante maximale théorique et la performance gigaflops (figure III.1). Cet écart de performance a ses racines dans les contraintes physiques par-cœur et les différences architecturales entre les deux processeurs. En même temps que les processeurs atteint le plafond de la série de performance, les GPU ont été en croissance exponentielle des performances due au parallélisme massif, comme le calcul de la couleur d'un pixel à l'écran peut être effectuée indépendamment de tous les autres pixels, le parallélisme est un moyen naturel d'augmenter les performances en GPU. **[35]**

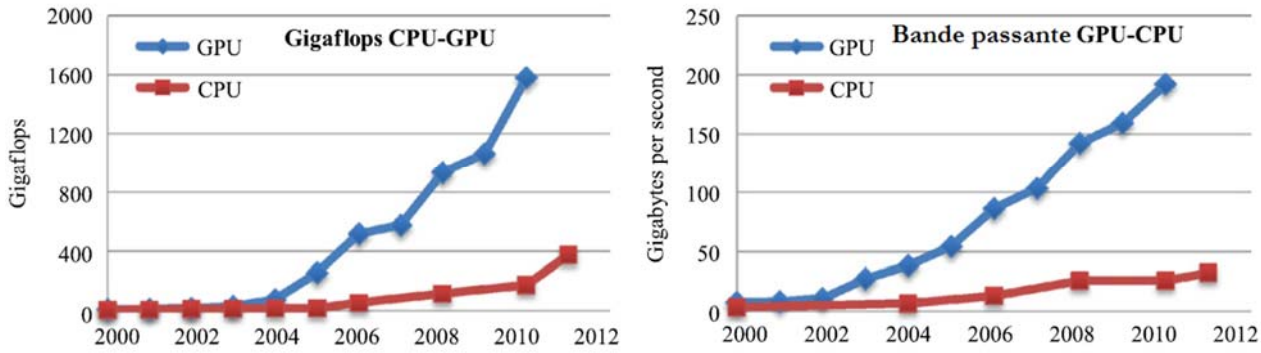

Figure III.1 : Comparaison historique des performances théoriques maximales en termes de gigaflops et la bande passante disponible pour le plus rapide GPU NVIDIA et les processeurs Intel. **[35]**

III.2.1 Les processeurs graphiques GPU

Le GPU (Graphic Processing Unit) est un processeur présent sur les cartes graphiques des ordinateurs, il est spécialisé dans le rendu graphique et la production d'images. Ils ont évolué ces dernières années vers de puissantes architectures parallèles multi-cœurs avec des performances de calculs qui ne cessent d'augmenter (Figure. III.2). **[45], [46]**

Actuellement les processeurs graphiques sont incontournables dans le calcul haute performance, grâce à l'arrivée de langages Cuda (Nvidia) dédiés facilement utilisables et leur grande puissance de calcul. Les GPU ont récemment acquis plus de flexibilité dans leur

utilisation, ne sont plus réservés qu'aux calculs graphiques, mais utilisés comme coprocesseur pour des calculs scientifiques, ce que l'on nomme le GPGPU (General-purpose computing on graphics processing units). À l'heure actuelle, il est utilisé dans de nombreux domaines : finance, bio-informatique, chimie, etc. **[47]**

Un CPU est capable de traiter tout type de tâches. Le GPU, quant à lui va traiter un nombre de tâches plus limitées, mais va être capable de l'appliquer à un très grand nombre de données. Leur architecture utilise une forme de parallélisme nommée SIMD (Single Instruction Multiple Data) c'est-à-dire une instruction, plusieurs données, où chaque unité de calcul du processeur effectue le même traitement sur des données différentes. En général, les unités de calculs sont regroupées sur un ou plusieurs multiprocesseurs. Les tâches qui s'exécutent sur un même multiprocesseur peuvent donc communiquer entre elles. Enfin les cartes graphiques disposent d'une mémoire globale accessible à toutes les tâches. **[45]**

Actuellement, L'architecture des GPUs s'appuie sur des multiprocesseurs SIMT (Single Instructions, Multiple Threads) c'est-à-dire une instruction, plusieurs threads, s'apparente a une architecture SIMD (Simple Instructions, Multiple Data). La principale différence entre ces deux architectures est la gestion avancée de tâches permettant des changements rapides de contexte dans le cas des architectures de type SIMT. Ces dernières sont capables d'exécuter plusieurs tâches effectuant la même opération en même temps. **[46]**

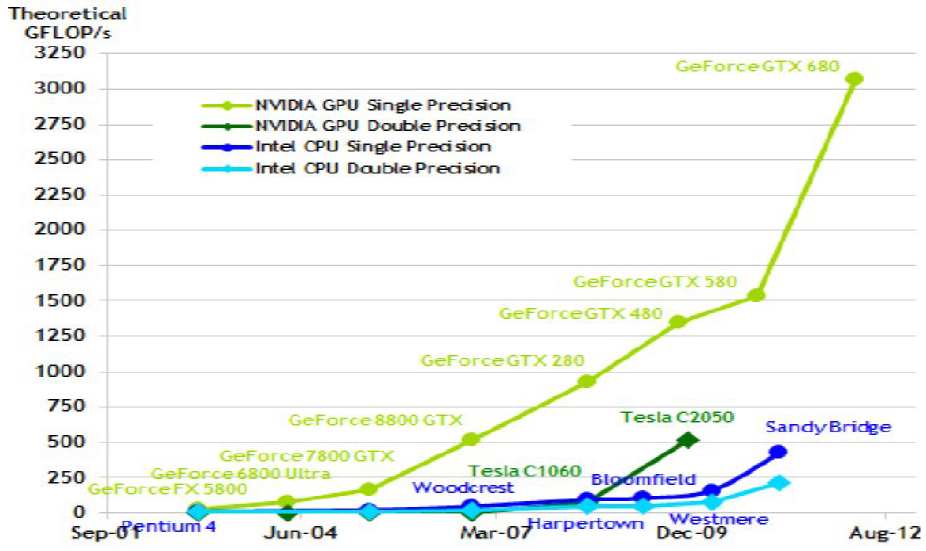

Figure III.2 : Evolution des performances de calcul des CPUs et GPUs [**NVIDIA**].

Les processeurs graphiques sont utilisées avec une meilleur performance pour des applications hautement parallèles qui nécessitent des milliers de tâches indépendantes. Pour amortir le coût de ses communications avec la machine hôte il est nécessaire d'occuper totalement et durablement le processeur graphique. **[45]**

III.2.2 GPU vs CPU

Le processeur graphique (GPU) est bien distinct de son prédécesseur le processeur central (CPU). On remarques la différence importante du nombre d'unités de calcule en vert (**Figure III.3**) appelés threads sur GPU et processus sur CPU.

Cependant un ensemble de threads partage le même espace de mémoire virtuelle alors qu'un processus possède la sienne. Cela induit un temps d'accès différents aux données. Là où chaque processus peut accéder de manière simultanée aux données, les threads doivent le faire de manière séquentielle. De plus, les processus ont un espace de mémoire cache dédié bien plus important, ce qui réduit leur temps d'accès aux instructions et aux données. **[48]**

Figure III.3 Architecture CPU et GPU [**NVIDIA**].

Un CPU dispose d'un nombre réduit d'unités de calcul, mais d'un cache d'une grande taille (plusieurs Mégaoctets) et une unité de contrôle importante. Cela est dû à la gestion de plusieurs tâches très différentes en parallèle qui nécessitent beaucoup de données. Ainsi, les données sont stockés en cache pour accélérer leur accès, et l'unité de contrôle va optimiser le flux d'instructions pour maximiser l'occupation des unités de calcul et optimiser la gestion du cache.

L'architecture GPU dispose d'un grand nombre d'unités de calcul qui disposent de peu de cache (quelques Kilooctets) et de faibles unités de contrôle. Cela lui permet de calculer de façon massivement parallèle le rendu de ces petits éléments indépendants, tout en ayant un débit important de données traitées.

III.3 API CUDA

III.3.1 Présentation

Depuis 2003 Nvidia a développé des outils logiciels pour aider les programmeurs à tirer partie de ses cartes graphiques afin d'avoir accès à un grand nombre de threads, à partir desquels il est possible d'exécuter en parallèle plusieurs processus. **[39]**

La société Nvidia a proposé son propre langage de programmation CUDA dédié à ces processeurs. Il s'agit d'une extension de C/C++ mais n'apportant que peu de modifications : 9 nouveaux mots clés, 24 nouveaux types et 62 nouvelles fonctions, ces extensions fournissant des primitives d'accès aux processeurs graphiques c'est-à-dire permettant les lectures et écritures mémoire arbitraires et accès à des mémoires locales et des synchronisations entre threads. Un programme CUDA utilise des kernels pour traiter des flux de données. Ces flux de données peuvent être par exemple, des vecteurs de nombres flottants, ou des ensembles d'image pour du traitement vidéo. Kernel est une fonction exécutée en parallèle par un certain nombre de threads sur GPU. C'est à partir du développement de l'architecture Tesla que NVidia a réalisé l'utilité de pouvoir traiter le GPU comme un CPU. L'architecture GT200 (Graphics Tesla) est la seconde architecture de type Tesla , la première était G80. On passe d'un circuit composé de 690 millions de transistors à un circuit de 1,4 milliards de transistors. **[44]**

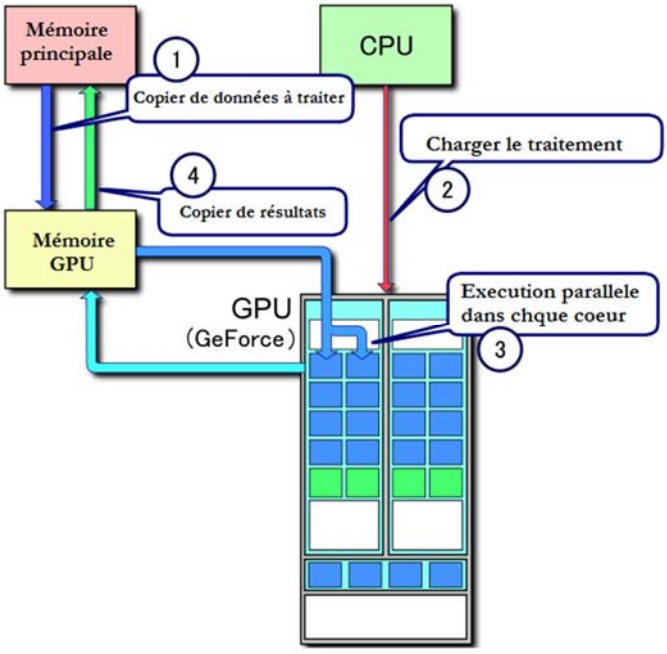

Figure III.4 : Organigramme de flux de traitement sur CUDA **[41]**

1. Copier des données de mémoire principale vers la mémoire GPU.
2. CPU charge le processus de GPU.
3. Exécution parallèle dans chaque cœur de la GPU.
4. Copier le résultat de la mémoire GPU vers la mémoire CPU.

III.3.1 Architecture CUDA

L'architecture CUDA comprend un traitement unifié permettant ainsi à chaque unité arithmétique et logique (ALU) du circuit d'être gérée par un programme afin de réaliser des traitements généraux. Ces ALU implémentent la norme IEEE pour l'arithmétique virgule flottante simple précision et ont été conçues pour utiliser un jeu d'instructions adapté à ces traitements généraux au lieu d'être spécialisées dans les traitements graphiques. En outre, les unités d'exécution du GPU peuvent désormais lire et écrire n'importe où en mémoire et également accéder à un cache appelé mémoire partagée. Toutes ces fonctionnalités de l'architecture CUDA ont été ajoutées afin de créer un GPU qui excelle dans les traitements généraux, avec de bonnes performances pour les traitements graphiques classiques. **[36]**

CUDA est prévu pour s'exécuter sur un GPU, mais il est aussi disponible sur CPU, en émulation. Les performances sont alors bien moindres, mais cela peut être utile pour tester ses applications sans GPU compatible. L'API CUDA est de haut niveau : Nous ne nous occupons donc pas du GPU directement. CUDA en est une couche d'abstraction qui permet de contrôler le GPU. **[43]**

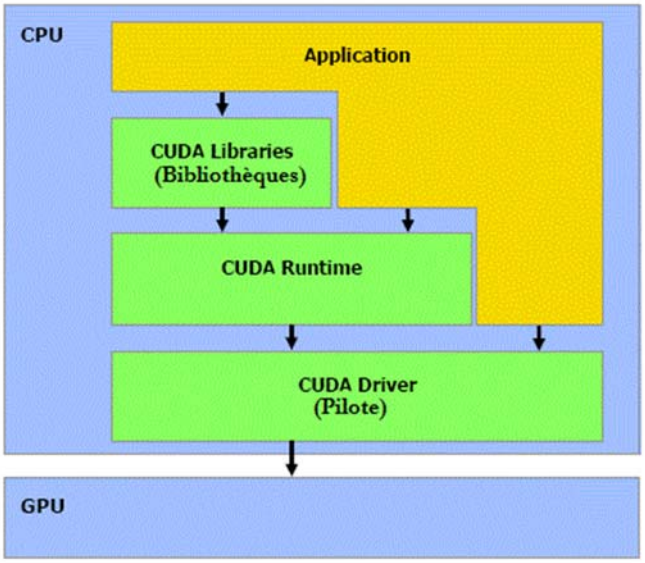

Figure III.5 : Architecture CUDA **[41]**

CUDA est donc représenté par un pilote (driver), un runtime, des librairies, une API basée sur une extension du langage C et le compilateur qui va avec.

Le pilote CUDA se charge du rôle d'intermédiaire entre le code compilé et le GPU, il transmet les calculs de l'application au GPU. Le runtime CUDA est lui un intermédiaire entre le développeur et le pilote qui facilite le développement en masquant certains détails. CUDA propose soit de passer par l'API runtime soit d'accéder directement à l'API Pilote. Il est possible de voir l'API runtime comme le langage de haut niveau et l'API pilote comme un intermédiaire entre le haut et le bas niveau qui permet d'optimiser manuellement le code plus en profondeur. **[42]**

L'architecture CUDA de nouvelle génération, portant le nom de Fermi, est l'architecture de calcul par le GPU la plus avancée jamais construite. Avec plus de trois milliards de transistors et jusqu'à 512 cœurs CUDA, Fermi délivre des fonctions et des performances de super calcul, pour 1/20ème de la puissance des serveurs traditionnels basés uniquement sur le CPU. **[37]**

III.3.2 Architecture Fermi

Fermi généralise l'utilisation simultanée du GPU et du CPU en accélérant une grande gamme d'applications informatiques. Conçue pour le langage C++ et livrée avec un environnement de développement Visual Studio, l'architecture Fermi optimise la programmation parallèle et accélère les performances d'une gamme d'applications, en permettant d'améliorer les performances de calcul en matière de physique, d'analyse des éléments finis, de calculs scientifiques à haute précision, d'algèbre linéaire, de tri et de recherche d'algorithmes. **[37]**

La première architecture Fermi est mis en place avec 3,0 milliards de transistors, offre jusqu'à 512 cœurs CUDA. Les 512 cœurs CUDA sont organisées dans 16 SMs(Streaming Multiprocessor) Multiprocesseur **(Figure III.8)**, 32 cœurs chacun. Une interface hôte qui connecte le processeur graphique à la CPU via PCI-Express. Les multiprocesseurs sont regroupés par 4 blocs appelés GPC (Graphics Processing Cluster) indépendants **(Figure III.7)**. **[45]**

Figure III.6 : Architecture Fermi **[Nvidia]**

Les 16 SM (multiprocesseurs) de Fermi sont positionnés autour d'un cache L2 commun. Chaque SM contient une partie d'orange qui est le GigaThread engine (répartiteur de tâches) se charge de la répartition des blocs pour exécution sur chaque SM, les parties vert sont des unités d'exécution, et les parties en jaune sont des fichier de registres et de la mémoire cache L1. **[45]**

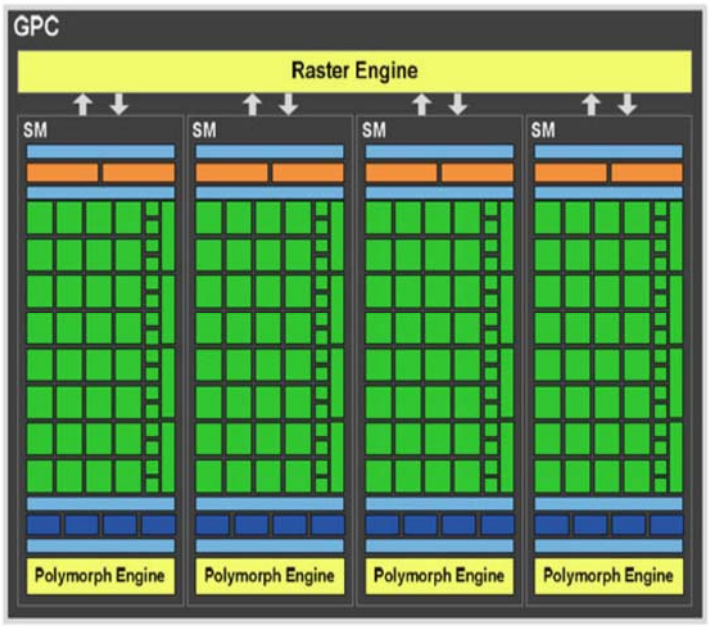

Figure III.7 : Architecture Fermi - GPC **[Nvidia]**

Chaque GPC se compose de 4 SM (multiprocesseurs) comprenant 32 cœurs, soit un total de 4 GPC x 4 SM x 32 = 512 Cœurs.

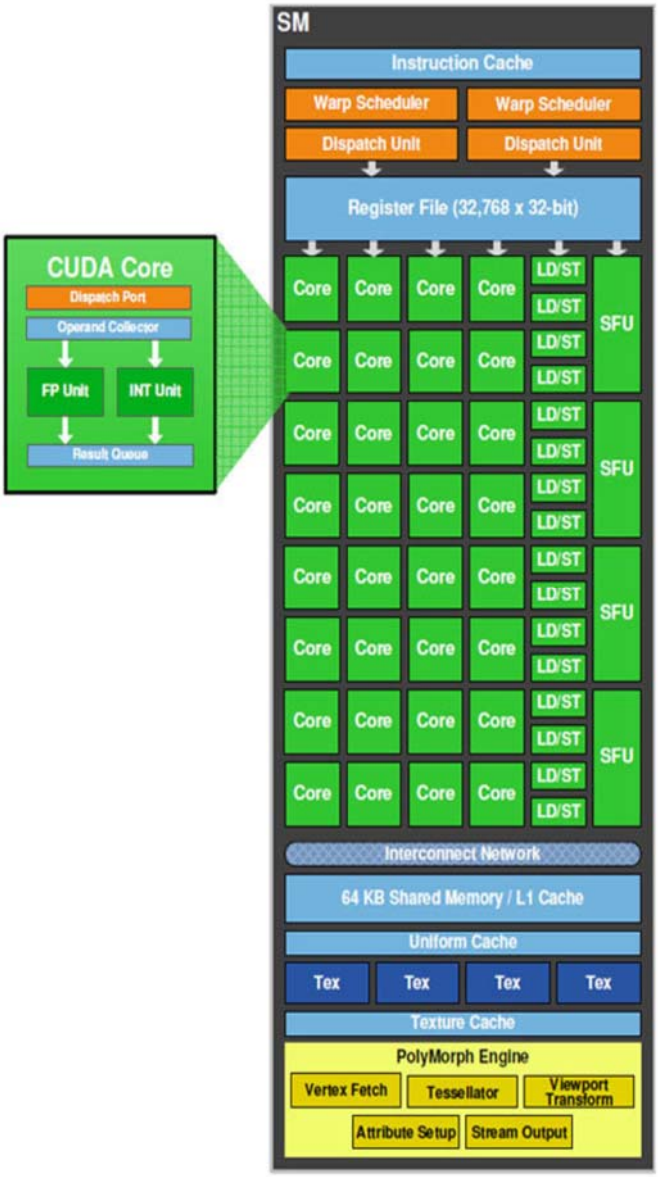

Figure III.8 : Architecture Fermi - Multiprocesseur **[Nvidia]**

Chaque processeur CUDA dispose d'une unité arithmétique et logique (ALU) et d'unité à virgule flottante (FPU). Avant GPU utilisé IEEE 754-1985 arithmétique en virgule flottante. L'architecture Fermi met en œuvre la nouvelle norme à virgule flottante IEEE 754-2008, fournissant la multiplication-addition (FMA) instruction fusionnée à la fois simple et double précision arithmétique. FMA est plus précise que l'exécution des opérations séparément. **[40]**

Chaque thread est exécuté par un processeur mais il faut tenir compte de l'organisation des threads en blocs de l'organisation des processeurs en multi-processeurs.

III.3.2.1 L'organisation hiérarchique des threads

Sur le GPU, les threads que l'on peut lancer en parallèle sur la carte graphique sont organisés en groupe de threads. La figure III.9 résume cette hiérarchie des threads avec un exemple en 2 dimensions.

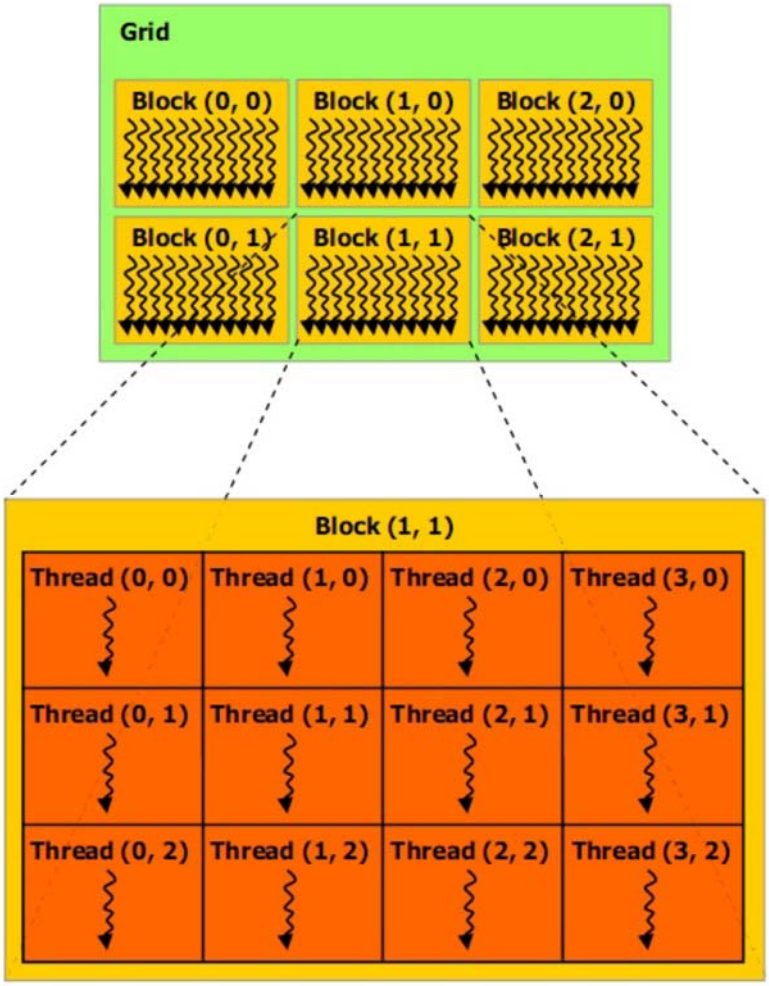

Figure III.9 : L'organisation des threads, blocks et grid dans une carte graphique **[Nvidia]**

Au plus **bas niveau**, il y a le thread. C'est la plus petite unité de traitement que l'on peut lancer sur la carte graphique. Ce sont les threads qui exécutent les programmes en parallèle. Leur nombre est impressionnant : on peut atteindre les 8 millions sur une simple carte GeForce 9300 GE. **[45]**

Au **niveau intermédiaire**, il y a le block : c'est un regroupement de threads. Les threads d'un même block partagent une mémoire commune très rapide. La position d'un thread dans un block est repérée par ses coordonnées sur 2 ou 3 dimensions. **[45]**

Au **plus haut niveau**, il y a le grid : c'est l'ensemble des blocks de la carte graphique. En fait, du point de vu des processus parallèles, le grid, c'est la carte graphique. Les mémoires qui lui sont liées, sont accessibles à tous les threads. De même que pour les threads dans un block, la position d'un block dans un grid est repérée par ses coordonnées sur 2 ou 3 dimensions. **[45]**

a. Threads

Les threads d'un même bloc sont exécutés sur un même multiprocesseur où chaque multiprocesseur possédant une mémoire partagée, cela permet aux threads d'un même bloc de communiquer par cette mémoire. Un multiprocesseur peut se voir attribuer plusieurs blocs suivant les ressources disponibles (registres), mais le nombre de threads par bloc est limité, cette limite dépend de l'architecture. Les threads d'un même bloc sont exécutés instruction par instruction par groupe de 32 threads consécutifs qu'on nomme warp. Par exemple le multiprocesseur (8 processeurs sur Tesla) exécute la première instruction du kernel sur les 8 premiers threads simultanément puis passe au 8 suivants. Une fois l'instruction exécutée sur les 32 threads, on recommence avec l'instruction suivante jusqu'à la fin du kernel. Un multiprocesseur exécute donc les instructions des threads suivant le modèle SIMT : Single Instruction Multiple Threads. **[38]**

b. Exécution et synchronisation de Warp

Les différents warps d'un même bloc ne sont pas exécutés en parallèle. Il n'y a aucune garantie sur l'ordre d'exécution des instructions entre threads de différents warps. Il peut y avoir des problèmes d'accès concurrents aux données en mémoire partagée si 2 threads de 2 warps différents manipulent la même donnée. Une barrière de synchronisation entre threads d'un même bloc est disponible. Lorsqu'un warp arrive à la barrière, il est placé dans une liste d'attente, une fois tous les warps arrivés à la barrière, leur exécution se poursuit après la barrière. Dans le cas d'une structure conditionnelle, la barrière doit être placée dans les deux branches, sinon blocage possible. **[38]**

c. Ordonnancement (Scheduling)

Si un warp doit attendre le résultat d'une longue opération (par exemple accès mémoire globale), celui-ci est placé dans une file d'attente et un autre warp dans la liste des warps prêts à

l'exécution peut être exécuté. Ce mécanisme permet de masquer les opérations ayant une latence importante et d'optimiser l'utilisation des processeurs.

d. blocs

Chaque bloc est placé sur un multiprocesseur. Plusieurs blocs d'un même kernel peuvent s'exécuter en parallèle sur différents multiprocesseurs. Suivant l'architecture, des blocs de kernels différents peuvent s'exécuter simultanément sur des multiprocesseurs différents. Sur l'architecture Fermi, des blocs de kernels différents peuvent s'exécuter simultanément.

III.3.2.2 L'organisation des mémoires [44],[50]

Les données qui doivent être traitées en parallèle par les threads sont premièrement transférées depuis la mémoire de l'ordinateur (*host memory*) vers celle du GPU (*global device memory*). Malheureusement, travailler uniquement avec la mémoire globale ne permet pas de tirer partie de la puissance totale du GPU car les accès mémoires ont une latence importante (de l'ordre de la centaine de cycles d'horloge) et une bande passante limitée.

Bien que l'ILP (*Instruction Level Paralellism*) obtenu par les nombreux threads permette de masquer les temps de latence, on peut se retrouver dans une situation de congestion où de nombreux threads ne peuvent progresser ce qui contraint les SM à rester en attente de traitement. Afin d'éviter cet état de congestion, CUDA introduit un certain nombre d'autres méthodes d'accès à la mémoire qui permettent d'éviter les accès à la mémoire globale afin d'améliorer de manière significative les performances du GPU comme figuré sur le schéma suivant :

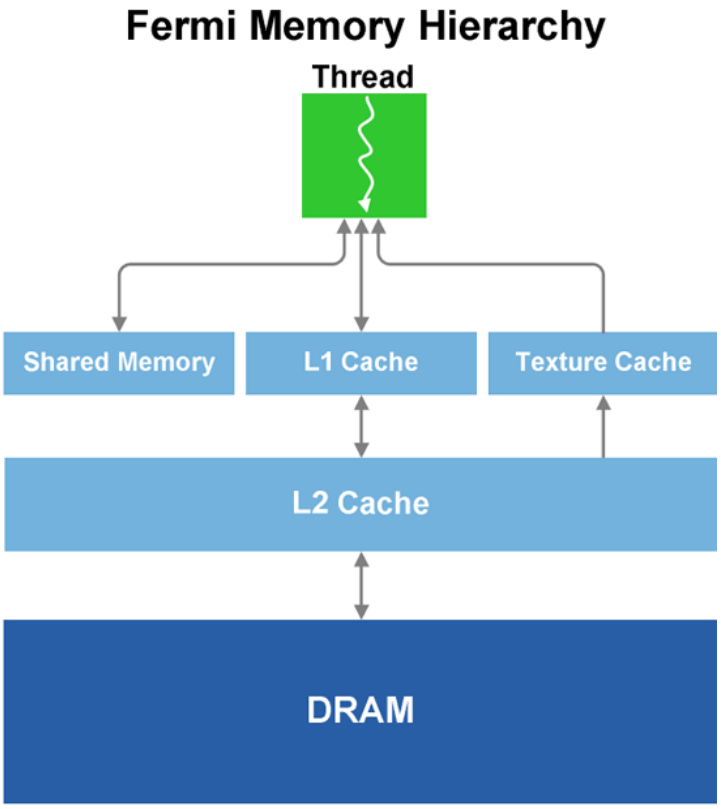

Figure : Hiérarchie de la mémoire fermi **[45]**

la bande passante mémoire des registres et de la mémoire partagée est très importante par rapport à celle de la mémoire globale du GPU.

Type	BP (Go/S)
Registre	8 000
Shared	1 600
Global	177

III.3.2.2.1 Mémoire Globale :

La mémoire globale du GPU (*Global Memory*) est la mémoire utilisable de n'importe quel endroit de CUDA, avec les mêmes performances, cette mémoire n'est pas cachée et il faut attendre 400 à 600 cycles avant d'y accéder. Ce qui laisse un multiprocesseur inactif pendant ce temps. La mémoire globale est en général de la DRAM. Elle peut être utilisée comme mémoire principale des ordinateurs. Cette mémoire est la plus importante et elle varie de 512 ko à 3 Go (Maximum 8 Go sur les cartes Tesla en 2012). Elle sert à stocker les données en entrée et sortie des kernels et est *off-chip*, c'est à dire située sur le PCB (*Printed Circuit Board*) du GPU mais pas à l'intérieur du circuit de calcul, cette mémoire à un problème, il s'agit de la latence. Elle monte jusqu'à 30 ns, ce qui représente quand même déjà 30 cycle.

Comme pour la programmation traditionnelle avec le CPU, l'alignement mémoire est important et influe sur les performances. Avec CUDA on parle de coalesced memory access, ce que l'on peut traduire en français par accès mémoires voisins ou contigüs. Le fait de ne pas aligner les données peut diviser la bande passante mémoire par un facteur N (4/32=1/8 sous Fermi).

Le fait d'aligner les données est imposé par les warp qui réalisent les accès mémoires en parallèle : si les données sont voisines alors l'utilisation de la bande passante est maximale car on a besoin que d'un seul accès mémoire. Par contre, l'accès à des données non voisines peut mener à une dégradation des performances de telle manière à ce que les accès mémoires soient sérialisés (et non plus parallèles)

Un premier facteur limitatif concerne la bande passante de la mémoire globale. Par exemple pour le G80, la BP est de 86,4 Go/s. Si on transfère des réels simple précision, la BP est de 86,4/4 = 21,6 Go/s.

Il existe deux types de chargements depuis la mémoire :

- **Avec cache (caching loads) :** il s'agit du mode par défaut, l'accès aux données passe par les deux caches L1, L2 puis la mémoire globale si les données ne résident pas dans les caches.
- **Sans cache (noncaching loads) :** lorsque des données doivent être accédées depuis des adresses non voisines il est préférable d'interdire l'utilisation du cache L1.

III .3.2.2.2. Mémoire cache :

C'est des petites quantités de mémoire, mais très rapide, qui se place entre le CPU et la mémoire centrale. Ils ne sont utilisés que pour les instructions fréquemment utilisées et les données. Il en existe deux niveaux : L1 et L2, exceptionnellement un troisième, L3, réservés généralement aux serveurs. Cependant, ces mémoires très rapides ne sont pas présentes en grande quantités sur nos CPU, Le cache fonctionne 10 fois plus vite que la RAM, avec un temps d'accès de 5 à 10 fois inférieur.

Les caches sont utilisés de manière transparente par le matériel. Ils se font les miroirs des données en mémoire. Ils transportent les données où elles sont nécessaires quand cela est demandé. Ces données ne sont remplacées que quand des données plus urgentes arrivent. Si les données demandées par le CPU sont disponibles sur le cache, celui-ci les lui envoie, le CPU ne

doit pas attendre. Par contre, si elles ne le sont pas, la demande est effectuée en aval, sur des mémoires plus lentes et le CPU doit attendre.

a. Mémoire cache L2 :

Il s'agit du premier niveau de cache après la mémoire globale, la plupart des données qui transitent entre mémoire globale et thread de calcul sont placées dans ce cache d'une taille de 768 ko sur Fermi.

b. Mémoire cache L1 :

On dispose sur l'architecture Fermi d'un cache configurable de 64 ko qui peut être utilisé en tant que cache L1 ou comme mémoire partagée :

- cache L1 de 48 ko + 16 ko de mémoire partagée.
- cache L1 de 16 ko + 48 ko de mémoire partagée.

Le cache L1 possède des caractéristiques tel que:

- il est conçu pour un accès spatial et non temporel.
- les écritures en mémoire globales n'affectent pas le cache L1.
- possède une latence de 10 à 20 cycle.

III.3.2.2.3. Mémoire partagée (Shared Mémory) :

La mémoire partagée (*Shared Memory* ou *smem*) est commune à un bloc de threads. Un bloc de thread peut donc échanger ou stocker temporairement de l'information afin d'augmenter les performances d'un calcul.

Elle est organisée soit sous forme de 16 ko ou 48 ko par SM comprenant 32 bancs mémoire de 32 bits de manière à ce que 32 threads puissent accéder en parallèle aux données.La mémoire partagée bénéficie à de nombreux problèmes, mais n'est pas appropriée à tous les cas de figure. Cette mémoire est présente sur le chipset, ce qui lui permet d'être assez rapide, plus que la mémoire locale. En fait, pour tous les threads d'un warp, accéder à cette mémoire est aussi rapide que d'accéder à un registre, tant qu'il n'y a pas de conflit entre les threads.

Pour permettre une bande-passante assez élevée, la mémoire partagée est divisée en modules de mémoire, les banques, qui peuvent être accédée simultanément. Ainsi, *n* lectures ou écritures qui

tombent dans des banques différentes peuvent être exécutées simultanément dans un warp, ce qui permet d'augmenter sensiblement la bande passante, qui devient *n* fois plus élevée que celle d'un module.

Cependant, si deux demandes tombent dans la même banque, il y a un conflit de banques et l'accès doit être sérialisé. Le matériel divise ces requêtes problématiques en autant de requêtes que nécessaire pour qu'aucun problème n'ait lieu, ce qui diminue la bande passante d'un facteur équivalent au nombre de requêtes total à effectuer. Dans le cas d'un espace en mémoire partagée, les banques sont organisées pour que des mots successifs de 32 bits soient assignés à des banques successives. Chaque banque a une bande passante de 32 bits tous les deux cycles d'horloge. Un warp a une taille de 32 threads et il y a 16 banques. Une requête en mémoire partagée pour un warp est divisée en deux : une partie pour le premier demi-warp, une autre, pour l'autre moitié. Ce qui a pour conséquence qu'il ne peut y avoir de conflit entre chaque demi-warp. La mémoire partagée atteint un total de 16 ko, 1 ko pour chaque banque.

La mémoire partagée est capable de diffuser à plusieurs threads une donnée (**multicast**), par exemple si N threads d'un warp accèdent à la même adresse alors on ne fait qu'un seul accès mémoire.

III.3.2.2.4. Mémoire locale :

La mémoire locale est une abstraction, il s'agit, suivant l'architecture :

- soit d'une partie de la mémoire globale du GPU (Compute Device < 2.0) : ce qui causait une perte de performance sensible;
- ou la mémoire cache L1.

Les variables automatiques déclarées dans un kernel sont remplacées par un registre. Cependant le choix entre un registre ou la mémoire locale n'est pas précise, le compilateur peut choisir d'utiliser la mémoire locale dans les cas suivants :

- il n'y a plus de registres disponibles;
- une structure serait susceptible d'utiliser trop de registres;
- on ne peut pas déterminer si un tableau est indexé avec des valeurs constantes.

III.3.2.2.5. Mémoire constante :

La mémoire constante est cachée, la lecture depuis cette mémoire ne coûte qu'un cycle. Pour tous les threads d'un demi-warp, la lecture depuis la mémoire constante est aussi rapide que depuis un registre, aussi longtemps que tous les threads lisent le même emplacement mémoire. Le coût de lecture augmente linéairement avec le nombre d'adresses différentes demandées par les threads. Il est recommandé que tous les threads d'un warp utilisent la même adresse et non seulement ceux de demi-warps, vu que les périphériques futurs le requerront pour un fonctionnement optimal. Chaque multiprocesseur dispose d'une mémoire réservée aux constantes, d'une taille de 8 ko.

III.3.2.2.6. Mémoire texture

Il s'agit d'une mémoire qui peut agir comme un cache liée généralement aux traitements graphiques. Cet espace mémoire est caché, le coût de la lecture est donc très faible. Elle est optimisée pour un espace à deux dimensions, ainsi, les threads d'un même warp qui lisent à des adresses proches auront des performances optimales. Il y a 8 ko de texture par SM. Aussi, elle est prévue pour des demandes de flux avec une latence constante.

La lecture des mémoires du périphérique par le mécanisme des textures peut être une alternative avantageuse à la lecture depuis les mémoires globale ou constante.

Les textures permettent vraiment de simplifier le traitement d'images : elles permettent la mise en oeuvre de filtrages bilinéaires et trilinéaires très facilement et l'accès aléatoire aisé aux pixels.

III.3.2.2.7. Registre

Les registres sont une ressource limitée qui influe sur les performances des kernels.

Par exemple pour le G80, chaque SM possède 8192 registres et peut gérer un maximum de 768 threads. Si on utilise de manière optimale chaque SM, chaque thread se voit attribuer un maximum de 8192 / 768 = 10,66 registres. Cependant, si chaque thread utilise plus de 10 registres, le nombre de threads qui peuvent êtres exécutés de manière concurrentielle dans chaque SM sera diminué, on passera par exemple de 768 threads à 512.

Sur Fermi, on dispose de 32 k (32768) registres. Si on utilise l'ensemble des threads, soit 1536, on disposera de 32768/1536 = 21 registres par thread. par défaut, le nombre maximum de registres associés à un thread est de 63.

III.3.2.3. Les caractéristiques des mémoires CUDA

Mémoire	Localisation	Cachée	Accès	Portée
Registre	chipset	Non	Lecture/Ecriture	1 seul thread
Locale	DRAM	Non	Lecture/Ecriture	1 seul thread
Partagée	chipset	Non indiqué	Lecture/Ecriture	Tous les threads du bloc
Globale	DRAM	Non	Lecture/Ecriture	Tous les threads + hôte
Constante	DRAM	Oui	Lecture	Tous les threads + hôte
Texture	DRAM	Oui	Lecture	Tous les threads + hôte

Figure : Les caractéristiques des mémoires CUDA

Les registres et la mémoire partagée (shared memory) sont des mémoires dites on-chip et ont un accès rapide.

L'accès mémoire concerne :

- les registres accessibles par les threads en lecture / écriture;
- la mémoire globale accessible par les threads en lecture / écriture;
- la mémoire partagée accessible par bloc de threads en lecture / écriture;
- la mémoire constante accessible par grille en lecture.

III.3.3 Host et Device

Host ou CPU c'est l'ordinateur qui sert d'interface avec l'utilisateur et qui contrôle le « device » utilisé pour exécuter les parties de calcul intensif basé sur un parallélisme de données. L'hôte est responsable de l'exécution des parties séquentielles de l'application. GPU est le processeur graphique, « General-Purpose Graphics Processor Unit », pouvant réaliser du travail générique qui peut être utilisé pour implémenter des algorithmes parallèles.

Device est le GPU connecté à « l'hôte » et qui va exécuter les parties de calcul intensif basé sur un parallélisme de données. Le device, ou périphérique, est responsable de l'exécution de la partie parallèle de l'application. kernel est une fonction qui peut être appelée depuis « l'hôte » et qui est exécutée en parallèle sur le « device » CUDA par de nombreuses threads.

III.3.3.1 Communication entre « l'hôte » et le « CUDA device »

L'échange de données entre l'hôte et le device est réalisé en copiant des données entre la DRAM, dynamic ram, et la mémoire DRAM globale du device. De la même façon qu'en C, le programmeur doit allouer de la mémoire dans la mémoire globale du device pour les données et libérer cette mémoire une fois l'application terminée.

Les **appels systèmes CUDA** suivants permettent de réaliser ces opérations :

Fonction	**Description**
cudaThreadSynchronize()	bloque jusqu'à ce que le « device » ait terminé les tâches demandées précédemment.
cudaChooseDevice()	retourne le « device » qui correspond aux propriétés spécifiées.
cudaGetDevice()	retourne le « device » utilisé actuellement.
cudaGetDeviceCount()	retourne le nombre de « device » capable de faire du GPGPU.
cudaGetDeviceProperties()	retourne les informations concernant le « device ».
cudaMalloc()	alloue un objet dans la mémoire globale du « device ». Nécessite deux arguments :

	l'adresse d'un pointeur qui recevra l'adresse de l'objet, et la taille de l'objet
cudaFree()	libère l'objet de al mémoire globale du « device »
cudaMemcpy()	copie des données de l'hôte vers le « device ». Nécessite quatre arguments : le pointeur destination, le pointeur source, le nombre d'octets et le mode de transfert.

III.3.4 Les "en-têtes" CUDA

On appelle "en-têtes" les qualificateurs des fonctions et des variables dans CUDA. Pour les fonctions, elles servent à définir la manière dont est exécutée une fonction. Pour les variables, elles servent à indiquer dans quelle mémoire la variable est stockée : selon son emplacement, l'accès et la modification ne se fera pas de la même manière et les performances ne seront pas les mêmes. **[48]**

III.3.4.1 Les qualificateurs de fonctions

Les qualificateurs de fonctions indiquent comment une fonction est appelée ou exécutée, suivant par exemple la syntaxe : __qualificateur__ void ma_fonction (arg_1, arg_2) ; Il existe 3 qualificateurs :

- __host__, une fonction appelée par le CPU et exécutée sur le CPU, c'est l'option par défaut lorsqu'on ne met rien.
- __global__, une fonction appelée par le CPU et exécutée sur le GPU.
- __device__, une fonction appelée par le GPU et exécutée sur le GPU.

On peut combiner __host__ et __device__, dans ce cas la fonction peut être exécutée sur le CPU et sur le GPU. **[48]**

III.3.4.2 Les variables

Les qualificateurs de variables indiquent le type de mémoire qui stocke la variable, suivant par exemple la syntaxe : __qualificateur__ int ma_variable ; Il existe 3 qualificateurs :

- __device__, la variable est stockée dans la mémoire globale du GPU, elle est accessible par tous les threads.
- __constant__, la variable est stockée dans la mémoire constante, elle est accessible par tous les threads.
- __shared__, la variable est stockée dans la mémoire partagée, elle est accessible seulement aux threads du même block.

On peut combiner __device__ avec les deux autres qualificateurs, leurs propriétés s'additionnent. **[48]**

III.4 Conclusion

Nous avons vu dans ce chapitre un bref historique sur l'évolution GPU comparée à celle des CPUs, puis nous avons mis en exergue l'architecture CUDA adaptée par les nouvelles technologies nVIDIA, allant de la structure des processeurs et mémoires graphiques jusqu'aux nouveautés introduits pour les développeurs. Le chapitre III conclut la partie I du document. La partie II est consacrée à la recherche bibliographie et aux détails de notre contribution.

Chapitre IV
Etude bibliographique des travaux réalisés

IV.1 Introduction

Dans les chapitres I, II et III qui constituent la partie I, nous avons parcouru les notions essentielles à la bonne compréhension de la problématique étudiée, qui s'étend sur un large spectre de connaissances en métaheuristiques, en bio-informatique et en GPU-computing.

Le début de la deuxième partie du document est marquée par ce quatrième chapitre qui est le fruit des études bibliographiques réalisées sur des travaux récents et antérieurs de référence, représentatifs de l'ensemble des efforts de recherche scientifique dans le domaine d'assemblage de novo de fragments ADN.

De nombreux travaux ont étés menés par des scientifiques, financés par des instituts gouvernementaux et privés, ou dans le cadre d'une collaboration entre plusieurs laboratoires, cherchant à la fois à trouver de nouvelles solutions au problème ou une amélioration des solutions existantes, et à mieux caractériser le problème d'assemblage de fragments ADN suivant les progrès technologiques du séquençage ainsi que la nature et l'origine des génomes à séquencer et leur teneur en erreurs. Les budgets importants consacrés pour la réussite de ces projets sont considérés comme étant des investissements à moyen et à long-terme.

IV.2 Choix des travaux étudiés

Notre proposition visant à résoudre le problème DNA-FAP, classé NP-difficile, est applicable pour un assemblage de novo suivant l'approche OLC (Overlap-Layout-Consensus) évoquée dans le chapitre II. Cependant, les travaux étudiés ne suivent pas tous cette même approche, mais représente la variété des solutions proposées du début du 21 siècle. La majorité des anciennes méthodes ont été conçues de façon primitive en prenant en compte des contraintes matérielles qui ne sont plus imposées aujourd'hui. Elles laissent donc place à des méthodes plus récentes, plus performantes et mieux adaptées au problème en question. Le choix des propositions étudiées est principalement focalisé sur les méthodes de recherche locale ou celles qui en encapsulent une.

D'autres travaux qui cherchent, non pas à résoudre le problème directement, mais à améliorer la qualité des résultats par le traitement des erreurs de séquençage, ne sont pas étudiées ici, car ils impliquent une charge de calcul supplémentaire. De plus, certaines de ces erreurs ne sont présentes que dans les anciennes banques de gènes, et sont donc de moins en moins improbables, notamment grâce aux progrès en termes de précision des nouvelles technologies de

séquençage automatique à haut débit. Nous en citerons quand-même des références majeures à la fin du chapitre.

IV.3 Approches basées sur la théorie des graphes

Nous introduisons ici des travaux connus utilisant la recherche du super-chemin eulérien dans un graphe de de Bruijn, deux propositions représentant à la fois la première méthode de ce genre ainsi qu'une autre assez récente.

IV.3.1 Travaux de Pevzner et *al.* (2001) [52]

Pavel A. Pevzner, Haixu Tang et Michael S. Waterman, chercheurs dans deux universités de Californie (San Diego et Los Angeles) aux USA, ont publié en 2001 un travail de référence, représentant complètement une nouvelle approche pour l'assemblage de fragments ADN.

Après environ 20 ans de recherches de nouvelles méthodes basées sur le paradigme Overlap-Layout-Consensus, Pevzner et *al.* ont proposé une nouvelle méthode qui se base sur la théorie des graphes, et ce afin de pallier à quelques points faibles des approches déjà connues (OLC et SCS) à l'époque, dont principalement :

- La phase chevauchement retrouve certaines similarités ne donnant pas des informations sur l'alignement des fragments. Une meilleure approche serait de montrer des similarités multiples, suffisantes pour juger de la qualité du chevauchement. Cette approche est cependant irréalisable dû à la charge énorme de calcul engendrée par une telle opération.
- Les assembleurs OLC dans leur majorité n'arrivent pas à résoudre le problème des régions répétées même avec des instances provenant de cellules procaryotes (voir chapitre II). Les biologistes travaillant dans les centres de séquençage sont conscients de cette difficulté, et sont donc amenés à mener des travaux expérimentaux additionnels pour réparer les séquences issues de l'assemblage.

Ce travail abandonne l'approche classique OLC en faveur d'un nouveau modèle qui réduit le problème d'assemblage de fragments ADN à la recherche d'un super-chemin eulérien dans un graphe de de Bruijn.

Les assembleurs existants connus tel que CAP3, Phrap et TIGR produisent des erreurs de l'ordre 17, 14 et 9 respectivement, et avec des améliorations ce score a été réduit à 5, 4 et 2 respectivement (Assemblage du génome *N. Miningitidis*. Alors que l'assembleur proposé, nommé EULER, élimine complètement les erreurs d'appel de bases grâce à des mécanismes d'alignement

multiple de séquences. Il produit donc des donnés sans erreurs, plus faciles à assembler par une recherche de super-chemin eulérien (Eulerian Superpath Problem - ESP).

IV.3.1.1 Alignement multiple de séquences et correction d'erreurs

L'approche de Idury-Waterman (1995) **[53]** pour l'alignement multiple de séquences est basée sur la comparaison non pas de séquences complètes, mais de *l-tuples,* séquences de petite taille, issues de la séquence d'origine et qui diffèrent l'une de l'autre d'une seule position. Ainsi, une chaîne de longueur N produit $(N - l + 1)$ l-tuples de longueur l. La comparaison de l-tuples est plus rapide avec une probabilité d'erreur moindre vu leur petite taille.

Le problème posé par cette méthode est qu'une erreur de séquençage sur une seule base implique la génération de $2l$ l-tuples portant la même erreur (voir figure 1). Cela a un impact négatif sur l'alignement global et nécessite un processus de correction d'erreur, et donc engendre une charge plus grande en termes de calcul.

- **Problème de correction d'erreurs :**

Etant donnés une séquence S, un seuil maximal d'erreurs Δ et une longueur l, introduire jusqu'à Δ correction telles que le cardinal du spectre de S, notée $|S_l|$, est minimisé. Le spectre de S contient l'ensemble des l-tuples de S et de $\bar{S}$ (inverse complémentaire de S).

Les algorithmes de correction d'erreurs fonctionnent généralement suivant le principe de distance d'édition. La distance d'édition entre deux séquences A et B est définie par le nombre de modifications à appliquer sur A afin d'obtenir B. Deux séquences ayant une distance d'édition assez petite (de l'ordre d'une seule erreur) sont susceptibles d'être à l'origine une seule et même séquence, ce qui constitue le critère de base pour tout outil d'alignement.

Dans un problème d'alignement multiple de séquences, une erreur empêche l'alignement parafait de deux chaines A et B ayant une distance d'édition non nulle. si $k_A$et $k_B$sont les nombres de l-tuples générés de A et de B respectivement, alors le nombre de l-tuples à une position donnée positions sera considérablement réduit à la présence d'une seule erreur. Les l-tuples affectés sont dit *faibles*, contrairement aux l-tuples solides, qui s'alignent en grand nombre sur une position correcte.

La méthode Idury-Waterman vise à augmenter le nombre l-tuples solides et à réduire le nombre des l-tuples faibles pour assurer un alignement de qualité. La correction des erreurs une

par une n'est pas une option intéressante en termes de complexité, et sa réalisation nécessite un temps considérable.

EULER propose un prétraitement visant à corriger les erreurs présentées par la figure 1. Il consiste en la recherche de modifications qui résultent en une réduction du cardinal $|S_l|$d'une valeur de $2l$ (figure 3). La quasi-totalité des erreurs sont cette nature et une correction de ce genre permet à EULER d'obtenir des données correctes (avec un taux d'erreur 0.11%).

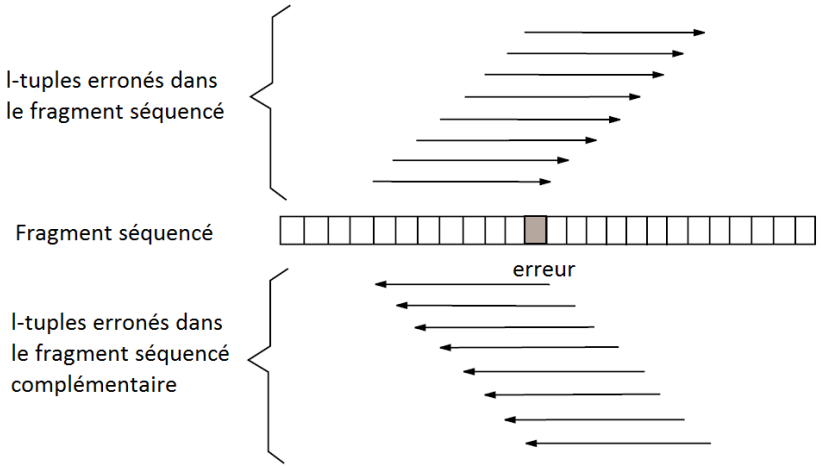

Figure 1 La présence d'une erreur affecte 2l l-tuples

Figure 3 : Impact de la présence d'une erreur sur le processus d'alignement

IV.3.1.2 Recherche du super-chemin eulérien :

Soit $S = \{s_1, \ldots, s_n\}$ un ensemble de lectures (fragments). Soit le graphe de de Bruijn $G(S_l)$, dont les nœuds sont de S_l, ensemble de tous les l-tuples de S. Les arcs du graphe sont des (l – 1)-tuples qui relient ses nœuds (l-tuples). Un (l – 1)-tuple u correspond à un arc de G s'il relie deux l-tuples v et w, i.e : u est identique à la fin de v et au début de w (figure IV.2)

Si S contient exactement une séquence s_1, alors le graphe G correspond à un chemin visitant chaque sommet de G une et une seule fois. La recherche du super-chemin eulérien est un problème assez connu qui peut être efficacement résolu.

L'approche suppose une multiplicité unitaire des sommets de G (pas de l-tuples répétés) et la présence du complément direct de chaque séquence (permet de construire la solution en amont et en aval).

- **Problème du super-chemin eulérien (ESP)**

Etant donné un graphe eulérien G et une collection de chemins dans ce graphe, trouver un chemin eulérien dans ce graphe qui contient tous ces chemins comme sous-chemins.

Afin de résoudre le problème du super-chemin eulérien, des transformations sont effectuées sur le graphe G et le système de chemins P de ce graphe pour obtenir un nouveau graphe G1 et un nouvel ensemble P1. Une telle transformation est dite équivalente s'il existe une correspondance un-à-un entre les super-chemins de (G, P) et de (G1, P1). L'objectif est d'effectuer une série de transformations équivalentes (G,P) → (G1, P1) → … → (Gk, Pk) qui mène à un système de chemins Pk où chaque chemin étant un seul nœud. Puisque la toutes les transformations de (G, P) à (Gk, Pk), alors chaque solution du problème du parcours eulérien dans (G, P) en donne une solution dans (Gk, Pk).

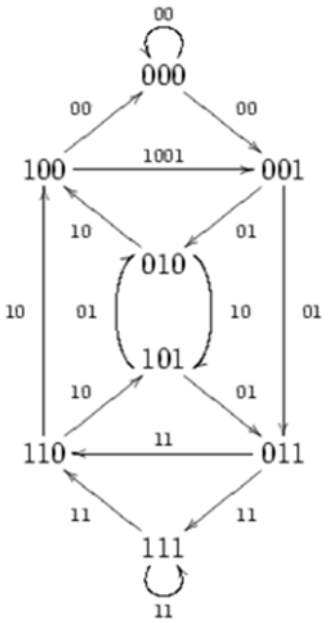

Figure 2 : Graphe de de Bruijn sur un alphabet binaire {0,1} dont les sommets sont des 3-tuples et les arcs sont des 2-tuples.

La transformation équivalente proposée dans EULER est une opération de détachement (voir figure 4) qui supprime deux arcs x et y de G et ajoute un nouvel arc *z* les remplaçant. L'objectif et de réduire le nombre d'arcs du graphe tout en gardant la construction de chemins équivalents à ceux appartenant à P. La figure 4 montre un exemple de ces transformations.

Les arcs yi ont une multiplicité unitaire tandis que les xi possèdent une multiplicité supérieure à 1. Le détachement réduit la multiplicité des arcs xi et baisse systématiquement leur degré entrant et sortant ce qui permet d'effectuer un parcours plus rapide du graphe lors de la recherche du super-chemin eulérien.

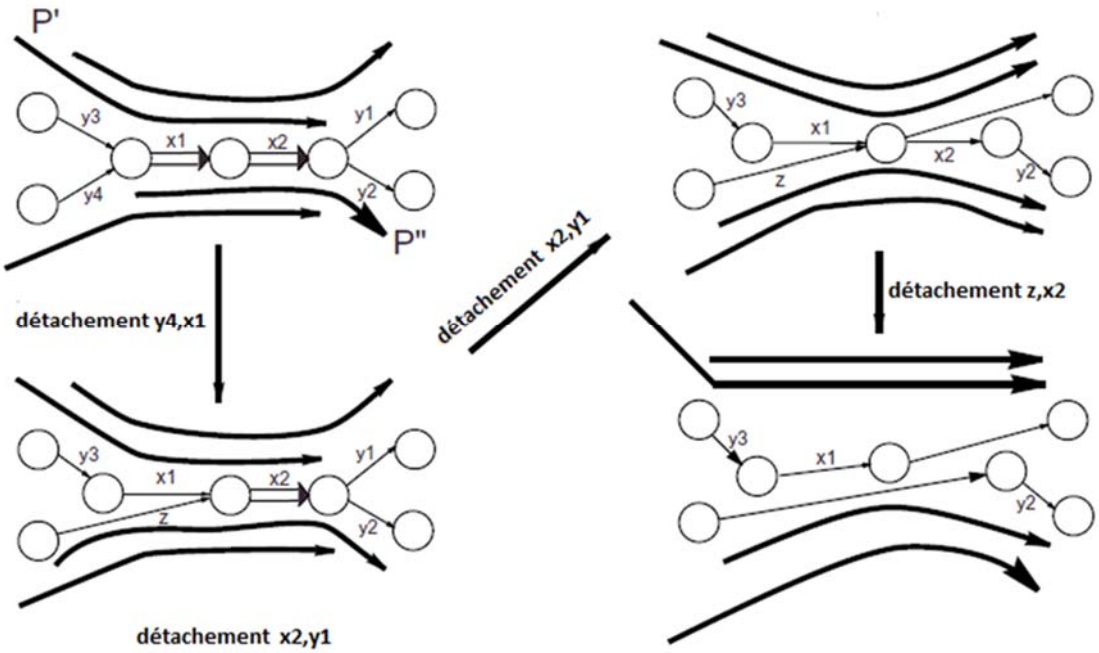

Figure 3 Exemples de la transformation de (G, S) par détachement, utilisée par EULER

IV.3.1.3 Critiques de la méthode :

Les auteurs eux-mêmes critiquent un coté de leur méthode et attirent l'attention sur un point important concernant l'efficacité du processus de correction d'erreurs. En effet, le mécanisme de correction d'erreurs cherche à faire coïncider les l-tuples séparés d'une petite distance d'édition afin de pouvoir construire un graphe de de Bruijn sans erreurs. Mais il arrive que des l-tuples provenant de différents fragments se fassent altérer certaines de leurs positions non-erronées. Cela produit, le cas échéant, des données corrompues non-correctes. Si le taux d'erreurs est important, la recherche du super-chemin eulérien serait alors réalisée sur un graphe corrompu et peut résulter en l'assemblage d'une super-séquence chimérique.

IV.3.1.4 Conclusion :

La nouvelle méthode de Pevzner, Tang et Waterman, qui réduit DNA-FAP à un problème de recherche de super-chemin eulérien, présente aujourd'hui une famille d'algorithmes d'assemblage. Cette approche n'élimine pas les difficultés de la résolution d'un tel problème, mais le rend un peu plus clair et plus facile à traiter. Les scientifiques et les chercheurs sont capables de considérer les grands groupes de sommets et d'arcs comme de simples éléments d'un nombre assez petit, au lieu de se focaliser sur chaque élément du brin d'ADN.

IV.3.2 Travaux de Xu et *al.* (2012) [54]

La proposition de Baomin Xu, Jin Gao et Chunyan Li, chercheurs deux universités chinoises (Beijing Jiaotong et de Tangshan), est l'une des approches originales de la famille de la théorie des graphes. Il s'agit d'une simplification de la méthode de Pevzner et al. et ses successeurs, tout en tirant avantage de la robustesse des ressources contemporaines de calcul.

La solution proposée suppose un environnement d'exécution distribué, ce qui permet donc de réaliser un calcul massivement parallèle au niveau de plusieurs machines distantes, et de pallier à la contrainte de complexité mémoire d'une telle méthode. Son implémentation est réalisée à l'aide du Framework MapReduce.

IV.3.2.1 Framework MapReduce

MapReduce est un modèle de programmation pour le traitement de grandes collections de données avec un algorithme parallèle et distribué sur un cluster.

Un programme MapReduce comprend une procédure **Map()** qui réalise le filtrage et le tri des données, et une procédure **Reduce()** qui orchestre l'exécution de tâches parallèles et le regroupement des résultats. Le modèle s'inspire des fonctions **map** et **reduce** généralement utilisées dans la programmation fonctionnelle, bien que leur objectif dans MapReduce n'est pas le même que celui de leur forme originelle.

IV.3.2.3 Principe de la méthode

L'algorithme de recherche du super-chemin eulérien n'est pas appliqué sur les lectures (fragments séquencés), mais il est précédé de deux autres opérations. La méthode de Xu et *al.* se résume en trois étapes :

a. Découpage des fragments en k-mers :

La première étape consiste à prendre en entrée les lectures d'en produire des k-mers. Un k-mer (le suffixe ***mer*** étant repris du mot ***polymer***) est une séquence nucléotidique relativement petite (de 20 à 25 bases). Les expériences dans cet article sont réalisées sur des k-mers de 20 nucléotides. Chacun des k-mers, provenant des différents fragments, se voit attribué un numéro de séquence unique.

b. Construction d'un graphe de de Bruijn :

Cette étape vise à trouver les relations d'adjacence entre les k-mers. Tous les couples (n°séquence, k-mer) sont stockés dans un fichier de séquences (appelé **`kmerfile`**). La construction d'un graphe de de Bruijn est très coûteuse en temps de calcul et en mémoire de stockage, et pour cela, il a été choisi de réaliser ce calcul dans un environnement distribué.

D'abord, le fichier de séquence **`kmerfile`** est stocké sur un système de fichiers distribué (HDFS : Hadoop Distributed File System) afin que les processus aient un accès aux informations de la totalité des k-mers. L'algorithme fait appel aux deux fonctions du framework MapReduce : Map et Reduce.

- La fonction **`Map()`** découpe l'ensemble des tuples (n°séquence, k-mer) contenu dans **`kmerfile`** en sous-ensembles de tailles égales, et associe chacun de ces derniers à un processus qui se chargera de construire le sous-graphe de de Bruijn correspondant.
- Ensuite, la fonction **`Reduce()`** construit le graphe de De Bruijn par fusion des sous-graphes de de Bruijn calculés localement, et ce en cherchant les relations d'adjacence existant entre chaque paire de k-mers hétérogènes (provenant de graphes locaux distincts). **`Reduce()`** permet d'obtenir un graphe de de Bruijn complet, dont toutes les relations d'adjacence sont stockées dans le fichier **`adjfile`**.

c. Trouver un super-chemin eulérien dans le graphe de de Bruijn.

L'idée de base est de choisir un nœud de départ et d'ensuite trouver un chemin en se basant sur la liste des voisins et des règles de compatibilité. Chaque étape lors de la recherche du chemin est basée sur les relations entre les nœuds (prédécesseurs et successeurs), et la prochaine étape dépend strictement de l'étape précédente. Ce calcul, comparé à l'étape 2, n'est pas coûteux pour trouver un intérêt dans sa parallélisation.

- **Compatibilité de chemins :**

Afin de résoudre le problème des régions répétées, la recherche du chemin eulérien est munie d'un prédicat permettant un choix déterministe du k-mer adjacent au k-mer actuel constituant le parcours eulérien souhaité. Ce prédicat détermine si les chemins sont compatibles.

Etant donné un k-mer k et son adjacent k' ($k' \in$ liste des adjacents de k), Le critère de compatibilité consiste à vérifier si $k \in$ liste des adjacents de k'. Ce mécanisme garantit la qualité

de la solution car le chemin eulérien est construit en amont et en aval et peut donc être parcouru dans les deux directions.

IV.3.2.3 Critique de la méthode

La méthode de Xu et *al.*, comparée à celle de Pevzner et *al* :

- remplace la génération de l-tuples par un découpage en k-mers, ce qui n'augmente pas, certes, la taille du problème mais qui peut résulter en une couverture incomplète, notamment dans le cas de présence de k-mers répétés qui s'alignent sur la même position du graphe de de Bruijn alors qu'ils ne proviennent pas d'une même région de l'ADN ;
- n'utilise pas un mécanisme de correction pour les erreurs d'appels de base qui, comme vu précédemment, rendent difficile le restant du processus d'assemblage. Ce deuxième point faible devient cependant presque non-existant avec les échantillons à une redondance élevée.

IV.3.2.4 Conclusion :

Le travail de Xu et al., l'un des algorithmes les plus récents qui, représente une nouvelle génération de la famille des approches ESP pour l'assemblage de novo. La puissance du calcul parallèle distribué a permis à de telles méthodes une amélioration importante en termes de fiabilité et d'efficacité.

IV.4 Approches basées sur le paradigme OLC :

Overlap–Layout–Consensus (Chevauchement–Agencement–Consensus) est la deuxième famille des algorithmes d'assemblage de fragments ADN. Sa première implémentation étant apparue en 1982, ce modèle, basé sur une représentation plus proche du DNA-FAP, connaît de plus en plus de proposition de la part des scientifiques et des chercheurs dans le domaine. (voir

IV.4.1 Travaux d'Alba et *al.* (2007) [55]

L'algorithme PALS, proposé par Enrique Alba et Gabriel Luque, chercheurs du département d'Informatique de l'université de Malàga (Espagne), est un modèle assez simple qui peut servir de base à différents programmes d'assemblage de novo par l'approche Overlap–Layout–Consensus.

IV.4.1.1 Principe de la méthode :

Dans l'approche adoptée pour la résolution du problème, PALS (Problem Aware Local Search), on suppose que les fragments adjacents sont ceux ayant une valeur plus élevée de la fonction de pertinence (fonction qui calcule le chevauchement de deux fragments donnés). On cherche donc à minimiser le nombre de contigs tout en maximisant les valeurs de la fonction de pertinence (fitness function).

a. Génération de la solution initiale :

Une solution initiale présente une permutation de l'ensemble des fragments en entrée (issus d'un séquençage). Pour cela, deux méthodes incrémentales de génération sont proposées : la méthode aléatoire et la méthode gloutonne.

- Génération incémentale aléatoire : les fragments sont numérotés arbitrairement puis les chevauchements sont calculés pour chaque paire de séquences successives.
- Génération incrémentale gloutonne : un fragment initial est sélectionné, puis d'une manière itérative, le prochain fragment choisi est celui qui maximise le chevauchement avec le dernier fragment choisi.

b. Calcul des mouvements candidats : Pour chaque paire de fragments *i* et *j* de la solution actuelle, on calcule la variation de la fonction de pertinence Δf et du nombre de contigs Δc après l'inversion de la sous-permutation entre *i* et *j*. Les calculs sont effectués uniquement sur les positions affectés par le mouvement (*i, i-1, j* et *j+1*). Ce même calcul est effectué pour tous les mouvements possibles.

c. Sélection et application d'un mouvement : parmi les mouvements calculés dans l'étape 2, seul un mouvement est sélectionné et appliqué selon une politique de sélection :

- **Meilleur :** On choisit le mouvement qui minimise le nombre de contigs et maximise le degré de chevauchement ;
- **Premier :** On choisit le premier mouvement qui n'augmente pas le nombre de contigs (qui ne produit pas une mauvaise solution) ;
- **Aléatoire :** On choisit un mouvement d'une manière aléatoire.

IV.4.1.2 Critique de la méthode :

PALS, étant une méthode de recherche locale possède des points faibles qui peuvent remettre en cause sa fiabilité :

- L'algorithme dans sa globalité suppose une méthode très efficace de génération de solution initiale. En effet, le départ d'une solution ayant un grand nombre de contigs n'aide pas trop dans la résolution effective du problème et engendre dans la majorité des cas la terminaison de l'algorithme avec une mauvaise solution.

  Avec une grande instance du problème ayant une couverture moyenne de 4. Si la génération de solution initiale donnerait un grand nombre de contigs en sortie, ce qui implique un long traitement dans la phase de consensus (les chevauchements sont remplacés par des alignements, soient en moyenne 3 fois plus complexes en nombre d'opérations de comparaison) ;
- Les tests sont réalisés sur des benchmarks artificiels, avec un taux d'erreur nul dans la plupart des cas. PALS n'est pas adapté au traitement de fragments réels qui présentent un taux d'erreur plus élevé et des régions communes plus redondantes.

IV.4.1.3 Conclusion

La méthode d'Alba et *al.* se caractérise par un choix judicieux de la structure de voisinage (inversion d'un segment de fragments) qui donne le meilleur équilibre entre le coût de calcul, le coût d'application et la probabilité d'amélioration. Elle peut cependant ne pas donner de bons résultats à cause d'un mauvais paramétrage de ses sous-méthodes (génération, calcul de mouvements, sélection). PALS a été reprise dans plusieurs propositions récentes et encapsulée dans des méthodes de recherche plus globale tels que les algorithmes génétiques et les méthodes d'optimisation par essaims particulaires.

IV.5 Autres travaux

Il existe une variété de travaux de recherche menés dans le domaine d'assemblage de fragments ADN, ne visant pas à trouver directement des solutions, mais cherchant à améliorer la qualité des instances à traiter. Nous en citons par exemple :

- Le travail de Zheng et al. (2004) : qui propose une méthode de correction d'erreurs de séquençage, meilleure que celle de Pevzner et *al.*, vue précédemment, utilisée en phase de pré-assemblage. **[56]**

- La contribution de J.S Firoz, M.S. Rahman et T.K. Saha (2012) : proposant un algorithme d'optimisation par colonie d'abeilles, efficace à la fois avec des instances sans et avec erreurs ;
- La méthode de Haixiang Shi et *al.* (2012) : cette approche fournit un mécanisme de correction d'erreurs basé sur les scores de qualité, et ce à travers un calcul sur GPU à l'aide de CUDA.
- La méthode de S.Nurk, P.Pevzner et al (2013)., avec la participation de 9 organismes différents : elle vise à résoudre le problème des séquences chimériques. Ce problème est difficile à détecter et le nombre de méthodes auquel il existe peu de propositions

IV.6 Conclusion

Nous avons présenté dans ce chapitre différentes méthodes issues de deux écoles d'assemblage de novo : certaines basées sur le paradigme Overlap-Layout-Consensus et d'autres qui explicitent l'équivalence (DNA-FAP ⇔ ESP). Notre proposition, détaillée dans le chapitre suivant, fait partie de la famille OLC.

Chapitre V
Proposition et Résultats

V.1 Introduction

Après avoir étudié quelques assembleurs connus de fragments ADN, nous introduisons dans ce dernier chapitre notre contribution, en passant d'abord par la présentation des modèles séquentiels mis en œuvre, puis des modèles parallèles proposés avec CUDA.

V.2 Instances de tests

Nous avons effectués nos tests sur des instances générés artificiellement par l'outil GenFrag. Ces benchmarks sont issus d'un travail de Michael L. Engle et Christian Burks (1992), et ont été utilisés comme mesure pour la comparaison des assembleurs en termes d'efficacité et de qualité de solution. Les fragments générés possèdent un seuil minimal de chevauchement égal à 30pb, ce qui rend l'assemblage un peu difficile pour les instances de grande taille. **[57]**

Les séquences choisies, dont les informations sont disponibles sur le site du NCBI, sont présentées par le tableau suivant :

Code d'accession	Nom	Longueur (bp)
X60189	HUMMHCFIB	3835
M15421	HUMAPOBF	10089
J02459	bacteriophage lambda	~ 20k
BX842596	Neurospora crassa	77292

Tableau V.1 : Benckmarks de tests

V.3 Modèles séquentiels proposés :

L'objectif de notre travail est de proposer un algorithme de recherche locale pour la résolution du DNA-FAP. L'idéal est d'avoir un algorithme qui aboutit à un assemblage parfait, mais ces algorithmes sont connus pour être complexes en temps d'exécution. C'est pour cela que l'utilisation des métaheuristiques est l'option la plus favorable, bien qu'elles ne donnent pas toujours les résultats exacts souhaités.

V.3.1 Recherche locale simple

La première idée était de mettre en œuvre une méthode de recherche locale qui s'inspire des travaux d'Alba et al (2007). Nous avons pour cela étudié les structures possibles à utiliser pour le voisinage. **[55]**

V.3.1.1 Structures de voisinage

Il existe une variété de mouvements pouvant constituer la base d'un voisinage efficace, certains étant élémentaires, d'autres une généralisation de mouvements simples. Nous proposons ici 5 mouvements différents, illustrés par la figure V.1 :

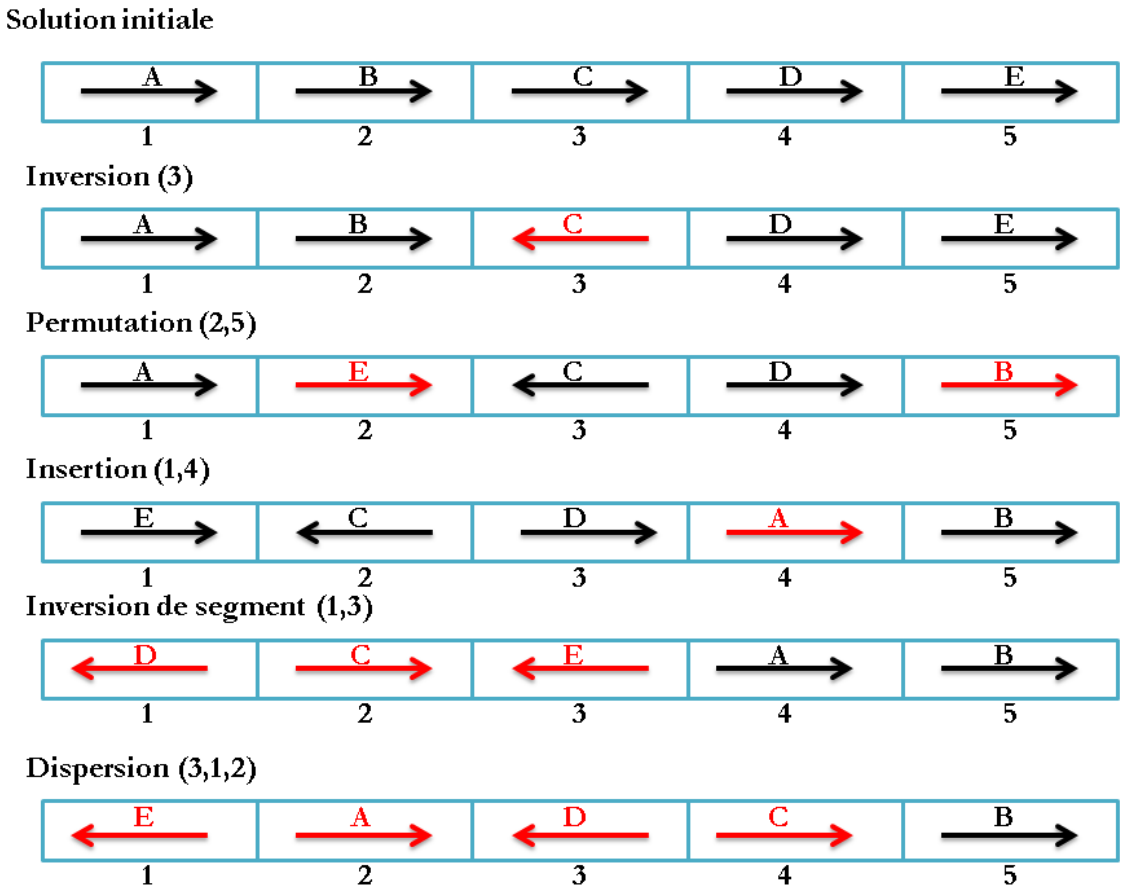

Figure V.1 : Exemple d'application des mouvements sur une solution de taille 5

flèches : fragment avec orientation

éléments en rouges : modifiés par le mouvement précédent

a. **Inversion (flip) :** consiste à inverser l'orientation d'un fragment. Si le fragment est positionné [gauche → droite] son orientation change en [droite ← gauche] et inversement ;
b. **Permutation (swap) :** étant donnés deux fragments de positions i et j respectivement, échange les positions des deux séquences ;
c. **Insertion :** Etant données deux positions i et j, insère le fragment n°i à la position j et décale tous les fragments situés entre les deux positions en direction de i ;

d. **Inversion de segment :** étant données deux positions i et j, prend tout le segment de fragments entre i et j compris, et y applique une inversion telle une seule séquence contiguë. Ce mouvement est réalisé à base de permutations symétriques et d'inversion de tous les fragments ;
e. **Dispersion :** étant donnés deux positions i et j, et une longueur l, applique une insertion en j du segment de taille l, dont le premier fragment est situé en i. Il s'agit bien évidemment d'une généralisation de l'insertion de plusieurs fragments contigus ;

Les mouvements élémentaires (inversion, permutation et insertion) se caractérisent par leur coût minime d'application comparés aux mouvements combinés (inversion de segment et dispersion), mais possèdent une faible probabilité d'amélioration de la solution. Bien que les mouvements combinés soient complexes en temps d'évaluation et d'application, ils possèdent une probabilité moyenne d'amélioration, permettant ainsi de faire avancer le processus de recherche locale. Une comparaison plus générale est présentée dans le tableau V.1.

Mouvement	**Coût d'estimation (chevauch.)**	**Coût d'application**	**Probabilité d'amélioration**	**Taille du voisinage (sur N fragments)**
Inversion	2	1 affectation	Très faible	N
Permutation	4	3 affectations	Faible	$\frac{N^2}{2}$
Insertion	3	Plusieurs affectations	Moyenne	$N * (N-1)$
Inversion de segment	2	Plusieurs affectations	Moyenne	$\frac{N}{2} * (N-1) - 1$
Dispersion	3	Plusieurs affectations	Moyenne	$\frac{N * (N-1)}{4}$

Tableau V.2 : Récapitulatif des mouvements applicables

V.3.1.2 Paramètres de la recherche locale :

Nous avons choisi pour notre algorithme de recherche local un ensemble de configurations définies comme suit :

- La génération de la solution initiale utilise une méthode d'amélioration aléatoire (random improvement).
- Le calcul des mouvements candidats utilise une structure de voisinage à base d'inversion de segment. D'après le tableau V.1, c'est le meilleur mouvement qui fournit un bon équilibre entre coût d'estimation total (coût * taille du voisinage) et la probabilité d'amélioration. L'insertion reste une option intéressante mais légèrement plus coûteuse car elle implique le calcul d'un chevauchement de plus. La dispersion, généralisation de l'insertion, n'est pas favorable pour un voisinage.
- La sélection se fait par la stratégie du meilleur d'abord (Best)

V.3.1.3 Résultats :

Avec l'instance **X60189_4** (39 fragments générés), le nombre de contigs n'est pas assez petit. La recherche locale s'arrête dans les meilleurs cas à 11 contigs et dans le pire des cas à 19 contigs, ce qui n'est pas suffisant.

Afin de mesurer la performance de l'implémentation, nous avons utilisé la plus grande instance **BX842596_7** (773 fragments de longueur environ 608pb). La première version du code a donné malheureusement 15 secondes par itération, pour un total de 700 à 800 itérations (plus de 2h d'exécution). La méthode n'est ni fiable ni efficace et nécessite donc une révision.

V.3.1.4 Application d'ingénierie de performance :

Les programmes écrits à des fins pédagogiques favorisent la simplicité et la clarté au détriment de performance, à l'inverse des programmes de calcul intensif qui cherchent à réduire au maximum le temps d'exécution à cause de la charge de calcul qui peut causer une lenteur cumulée pour des instances à grande échelle, ce qui est le cas de notre application.

En ayant recours à certaines pratiques d'ingénierie de performance, nous avons pu accélérer la performance du programme d'un facteur de l'ordre de 40, un nombre qui peut encore augmenter. Dans ce qui suit quelques-unes de ces pratiques :

a. Profiling :

Le profiling permet de mesurer la performance du programme, en fournissant des informations sur les calculs les plus coûteux, ainsi que les parties les plus visitées du code dont l'amélioration engendre une accélération plus grande.

b. Linéarisation de fonctions :

Les appels de fonctions consomment du temps à cause des opérations de chargement des paramètres sur la pile. Si un sous-programme est trop utilisé, il est conseillé de remplacer ses instructions d'appel par des instructions simples insérées directement dans le code. Cette opération augmente la taille du code à cause de la redondance de certaines de ses parties, un problème facile à traiter grâce à l'utilisation de macros. t de ne pas encombrer le code source et de le garder esthétique.

c. Elimination de branchements conditionnels :

L'utilisation de structures de sauts conditionnels (if..else, while, …) rend inconnue l'instruction suivante à exécuter. Alors que le compilateur effectue des optimisations sur l'exécution des séquences connues d'instructions indépendantes ou dépendantes (architecture pipeline), un saut conditionnel élimine cette possibilité. Des opérations simples, telles que le minimum/maximum de deux nombres, sont à base de branchements conditionnels mais peuvent être écrites seulement à l'aide d'opérateurs arithmétiques, relationnels et bit à bit (bitwise). Un calcul complètement intrinsèque à l'UAL remplace les opérations de saut et économise quelques cycles d'horloge.

d. Modification des structures de données traitées :

- Le stockage des fragments (séquences de caractères A, T, C et G) dans les deux orientations permet de supprimer le calcul engendré par les positions des fragments inversés ;
- Le parallélisme niveau instruction : puisqu'une machine 32 bits possède des registres d'une taille de 4 octets, la comparaison de 2 paires de 4 bases (soit la taille d'un int en langage C) est au moins 4 fois plus performante que la comparaison de 2 bases (2 char, 8 bits chacun). Le codage d'une base sur 4 bits (5 positions A T C G –) ou sur deux bits (4 positions A T C G) permet d'atteindre des rapports de performance de 8/1 et 16/1 respectivement.
- Boucle sentinelle : les boucles ayant deux critères d'arrêt impliquent un grand nombre de comparaisons. L'idée est de supprimer une vérification et d'ajouter un élément à la fin de la structure de données manipulée qui satisfait toujours la condition.
- Elimination de mouvements équivalents et inutiles : pour une inversion de segment, le mouvement (i, j) est équivalent à mouvement $(j' + 1, i' - 1)$. Il suffit de calculer l'ensemble des mouvements pour lesquels $(i \leq j)$. Les insertions (i, i) sont également un exemple de mouvements inutiles qui ne modifient pas la solution.

V.3.2 Recherche locale itérative :

La recherche locale n'était pas efficace pour des raisons citées précédemment (voir les travaux d'Alba et *al.*, chapitre IV). Nous avons donc implémenté une procédure de recherche locale itérative qui encapsule l'algorithme de recherche locale.

V.3.2.1 Utilisation d'une perturbation :

La recherche locale itérative (introduite en chapitre I) ajoute, en plus de l'itération d'un processus de recherche local, un mécanisme de sortie des optimums locaux : il s'agit de la perturbation. Les étapes d'exécution du nouveau modèle sont détaillées par l'algorithme V.1 :

```
Sorties: meilleure solution s*;
 0: GénérerSolution(s0)
 1: Evaluer(s0)
 2: s* = s0
 3: s = s*
 4: répéter
 5:     Perturbation(s)
 6:     Evaluer(s)
 7:     RechercheLocale(s)
 8:     si f(s) > f(s*) alors
 9:         s* = s
10:     finsi
11: jusqu'à critère d'arrêt vérifié
```

Algorithme V.1 : Recherche locale itérative

- GénérerSolution() est la même que celle de la recherche locale.
- La perturbation applique aléatoirement des insertions sur la solution. Elle cause, certes, une dégradation de la qualité de la solution mais permet de continuer une autre exécution de la recherche locale dans l'espoir de tomber sur un meilleur optimum.
- f(s) est l'évaluation calculée par la fonction Evaluer(s) et possède deux paramètres : le nombre de contigs (à minimiser) et le chevauchement total (à maximiser) de la solution.
- Le critère d'arrêt est une condition sur le nombre de contigs.

V.3.2.2 Résultats :

La recherche locale itérative dépend fortement de la méthode de recherche locale qu'elle encapsule. Si cette dernière n'est pas très efficace, alors la perturbation ne fait qu'augmenter les chances d'atteindre un nombre de contigs égal à 11 pour l'instance **X60189_4** citée précédemment.

IV.3.2.3 Introduction d'un deuxième voisinage :

L'algorithme précédent utilise une recherche locale avec un voisinage d'inversion de segment. La proposition actuelle est d'ajouter au code de l'algorithme V.1 une recherche locale mais avec un autre voisinage : il s'agit de l'insertion de fragments (algorithme V.2).En effet, l'application d'un mouvement d'inversion de segment à une itération i engendre la disparition de certains mouvements candidats suite à un changement d'orientation. La continuation de la recherche locale avec des mouvements d'insertion permet d'accepter de nouvelles améliorations de la solution et d'atteindre un nombre de contigs plus bas.

```
   Sorties: meilleure solution s*;
 0: GénérerSolution(s0)
 1: Evaluer(s0)
 2: s* = s0
 3: s = s*
 4: répéter
 5:     Perturbation(s)
 6:     Evaluer(s)
 7:     SFM_RechercheLocale(s)
 8:     IM_RechercheLocale(s)
 9:     si f(s) > f(s*) alors
10:         s* = s
11:     finsi
12: jusqu'à critère d'arrêt vérifié
```

Algorithme V.2 : Recherche locale itérative à double voisinage

Les préfixes ajoutés SFM et IM représentent respectivement SegmentFlipMovement (mouvement d'inversion de segement) et InsertionMovement (mouvement d'insertion).

Le calcul de mouvements d'insertion prend plus de temps, mais on privilégie la qualité au détriment de la performance afin d'atteindre un nombre de contigs acceptable pour l'algorithme de consensus (inférieur à 10, idéalement 1), soit 9 pour **X60189_4**.

V.4 Modèle d'exécution parallèle sur GPU :

L'utilisation d'un processeur graphique pour effecteur certaines parties du calcul en parallèle permet de multiplier les performances du programme d'un facteur très important, mais cela nécessite des connaissances du problème en question et des portions de traitements pour lesquels une exécution sur une GPU est favorable. Dans ce qui suit, nous présentons deux modèles destinés à une implémentation de notre méthode de recherche locale itérative à double voisinage.

V.4.1 Calcul parallèle des mouvements candidats de chaque itération :

Le calcul des mouvements candidats est la partie la plus exécutée dans l'algorithme. Elle contient pour chaque mouvement, selon la structure de voisinage choisie, un nombre de chevauchements à calculer entre les fragments, à base de simples comparaisons. Un parallélisme au niveau du calcul de mouvement est intéressant vu la taille des structures de voisinage utilisées. A titre d'exemple avec le voisinage d'inversion de segment :

- Pour l'instance la plus petite X60189_4 : 39 fragments donnent 780 mouvements (1561 au total).
- Pour l'instance la plus grande BX842596_7 : 773 fragments donnent 259150 mouvements (598302 au total).

❖ Inconvénients :

- Il est difficile d'assurer une grande occupation du processeur graphique à cause du grand nombre de branchements conditionnels (longueurs de chevauchement différentes). Ce problème cause une divergence au niveau des Warps, qui ne sont plus lancés massivement en parallèle et impliquent un plus long temps d'ordonnancement dédié au Gigathread et au Warp Schedulers de CUDA.
- Le kernel du calcul d'un mouvement candidat possède un nombre important de variables, destinées à être stockées dans des registres. Une limite est imposée sur le nombre de registres disponibles par block (32768 registres dans cas d'un GPU nVIDIA Tesla C2075), ce qui réduit le nombre de threads que peut contenir un bloc.

V.4.2 Calcul parallèle des chevauchements distinctement :

La clé de la performance dans CUDA est de savoir manipuler les mémoires. Il faut que le temps consommé par les instructions de copie mémoire (de la mémoire globale vers la mémoire partagée par exemple) et d'ordonnancement de threads, soit négligeable par rapport au temps de calcul effectif. Pour atteindre cet objectif, le calcul effectué par les thread doit être le plus uniforme possible.

L'idée est d'effectuer des calculs partiels des mouvements, dont la brique de base est le calcul de chevauchement. On cherche à réduire les accès mémoires divergents et faire en sorte que plusieurs threads accèdent à une même donnée en lecture concurrente (les GPU de l'architecture CUDA sont des PRAM CREW).

Un seul chargement en mémoire du fragment i signifie que tous les chevauchements calculés avec ce fragment doivent être initiés dans le même block, i.e :

$$\{0 \leq k \leq n : \ (i+1,k)\ ou\ (k,i-1)\}$$

V.5 Expérimentation

Nous avons implémenté le modèle parallèle n°1, dont le calcul du mouvement candidat représente la brique de base du traitement massif sur GPU (kernel). Cette section représente l'essentiel du travail réalisé. Nous avons procédé à des tests visant à comparer la performance du programme séquentiel (CPU uniquement) et du programme parallèle (CPU+GPU). Les expérimentations sont réalisés sur les benchmark cités précédemment, principalement les instances les plus grandes, et dans un environnement d'exécution bien défini.

V.5.1 Environnement d'exécution

- **CPU :** Processeur Intel Xeon® E5606, nombre de cœurs : 4, fréquence d'horloge : 2.13 GHz.
- **RAM :** 8 GB DDR3.
- **GPU :** nVIDIA Tesla C2075 ; nombre de cœurs CUDA : 448 cœurs, fréquence d'horloge : 1.15 GHz, mémoire graphique : 6GB GDDR3.
- **Système d'exploitation :** GNU/Linux, version du kernel : 2.6.28-11-generic, distribution : Ubuntu 9.04 Jaunty Jackalop (Debian 5.0), édition 64 bits.

V.5.2 Essais réalisés

Les essais effectués dans leur majorité servent à comparer la performance des deux programmes écrits, mais il y a cependant une série de tests permettant de déterminer la bonne configuration du programme parallèle. Les séries d'essais sont résumées dans le tableau V.3.

Essai n°	Instances	Exécution	Métrique	Remarques
1	38524243_7 (773 fragments)	GPU seulement	Temps d'exécution de 200 itérations	Variation de la taille du bloc CUDA
2	38524243_4 (442 frgs) 38524243_7 (773 frgs)	CPU vs GPU	Temps d'exécution	Nombre d'itérations 50, 100, 150, 200
3	x60189_4 (39 frgs) x60189_6 (66 frgs) m15421_5 (127 frgs) m15421_7 (177 frgs) j02459_7 (352 frgs) 38524243_4 (442 frgs) 38524243_7 (773 frgs)	CPU vs GPU	Temps moyen par itération	Le temps est évalué sur une exécution jusqu'au premier optimum trouvé

Tableau V.3 : Résumé des expérimentations

- L'essai n°1 vise à déterminer la taille de bloc CUDA idéale pour une exécution optimale du programme GPU. La technologie CUDA limite le nombre des registres utilisés par bloc de threads. Si un kernel déclare un grand nombre de variables et effectue plusieurs traitements, il occupera en conséquence un grand nombre de registres. La taille d'un bloc détermine le nombre d'instances du kernel (threads) qui s'y exécutent. Si la taille est trop élevée, les threads auront recours à la mémoire partagée. Par contre, si la taille est trop petite, les registres disponibles au niveau du bloc ne seront pas exploités au maximum.
- Les essais n°2 et 3 ont pour but de comparer la performance des deux implémentations de notre algorithme (CPU et GPU). L'essai 2 mesure les temps d'exécution sur les deux échantillons les plus grands (38524243 [~77kbp]) pour déterminer les performances de pointe. Tandis que l'essai 3 mesure le temps moyen d'une itération en variant la taille de l'instance de départ, et ce afin d'avoir une approximation de la vitesse d'exécution par rapport à un échantillon quelconque.

V.5.3 Résultats et discussion :

A. Influence du nombre de threads par bloc sur la performance :

Il est évident que les programmes de calcul intensif prennent moins de temps pour s'exécuter sur une GPU comparé au temps nécessaire pour une exécution sur CPU. Cet écart naturel est dû à l'évolution exponentielle des GPU suivant la loi de Moore (la capacité de traitement des processeurs double tous les 18 mois) mais avec un facteur plus élevé que celui des CPU. Cependant, il existe un nombre considérable d'améliorations que peut subir un programme de GPU-Computing afin de le rendre plus performant.

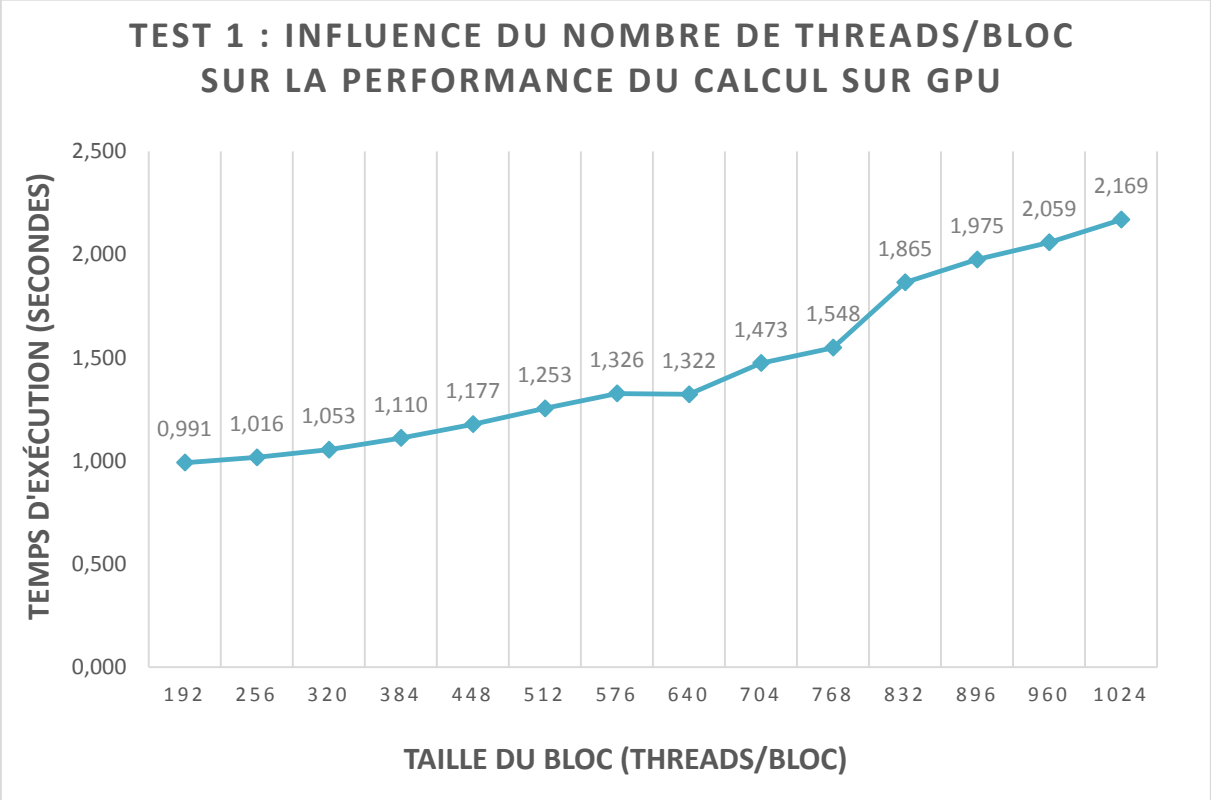

Figure V.2 : Influence du nombre de threads/bloc sur la performance du calcul sur GPU

Les résultats du test n°1 (figure V.2) montrent bien que la variation du nombre de threads par bloc, une des configurations de l'exécution sur CUDA, joue un rôle important dans la détermination de la durée d'un traitement GPU. Ce test est réalisé sur l'instance du problème la plus grande parmi les benchmarks disponibles (38524243_7), et dont chaque itération nécessite le calcul de 259150 mouvements candidats.

Pour un nombre maximal de 1024 threads/bloc, l'exécution de 200 itérations du programme prend 2,169 secondes. Alors que pour 192 threads/bloc, le temps d'exécution diminue

à 0,991 secondes. La courbe possède une allure croissante entre ses deux valeurs. En effet, le kernel défini pour le calcul d'un mouvement candidat réserve trop d'espace d'adressage local (variables, résultats intermédiaires), ce qui consomme rapidement les registres disponibles en nombre limité par bloc (32768 registres dans une architecture Tesla C2075). Un nombre élevé de threads issus de ce kernel implique un accès concurrent aux ressources locales, et un recours à des ressources plus grandes mais moins rapides, disponibles aux niveaux supérieurs de l'architecture (mémoire partagée). La réduction de la taille du bloc diminue la charge des registres et réduit donc le temps d'attente des threads, ce qui améliore le temps d'exécution total.

B. CPU vs GPU : Performance de pointe

La vraie puissance d'une métaheuristique quelconque se mesure par sa capacité à résoudre de grandes instances pendant un temps raisonnable. Le test n°2 avait pour but de comparer les performances des deux programmes CPU et GPU avec les deux plus grandes instances du problème DNA-FAP, et pour un nombre borné d'itérations. Les résultats du test sont illustrés par les figures V.3 et V.4

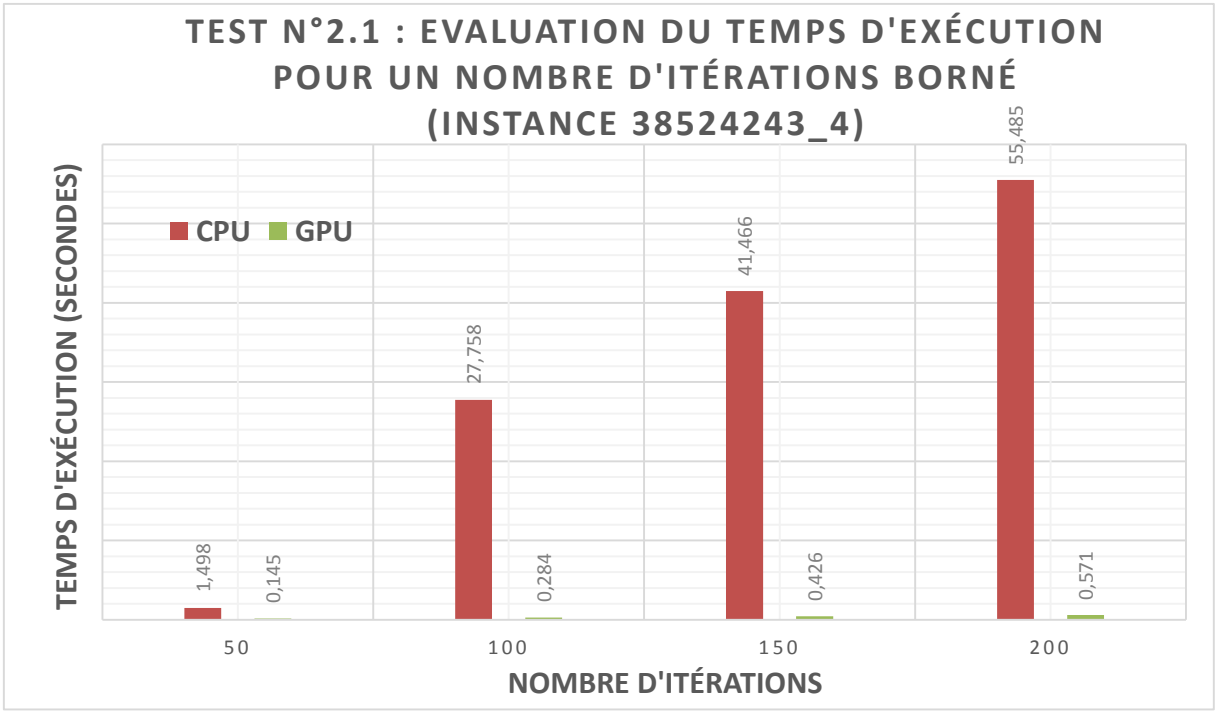

Figure V.3 : Temps CPU vs GPU pour 50, 100, 150 et 200 itérations [38524243_4]

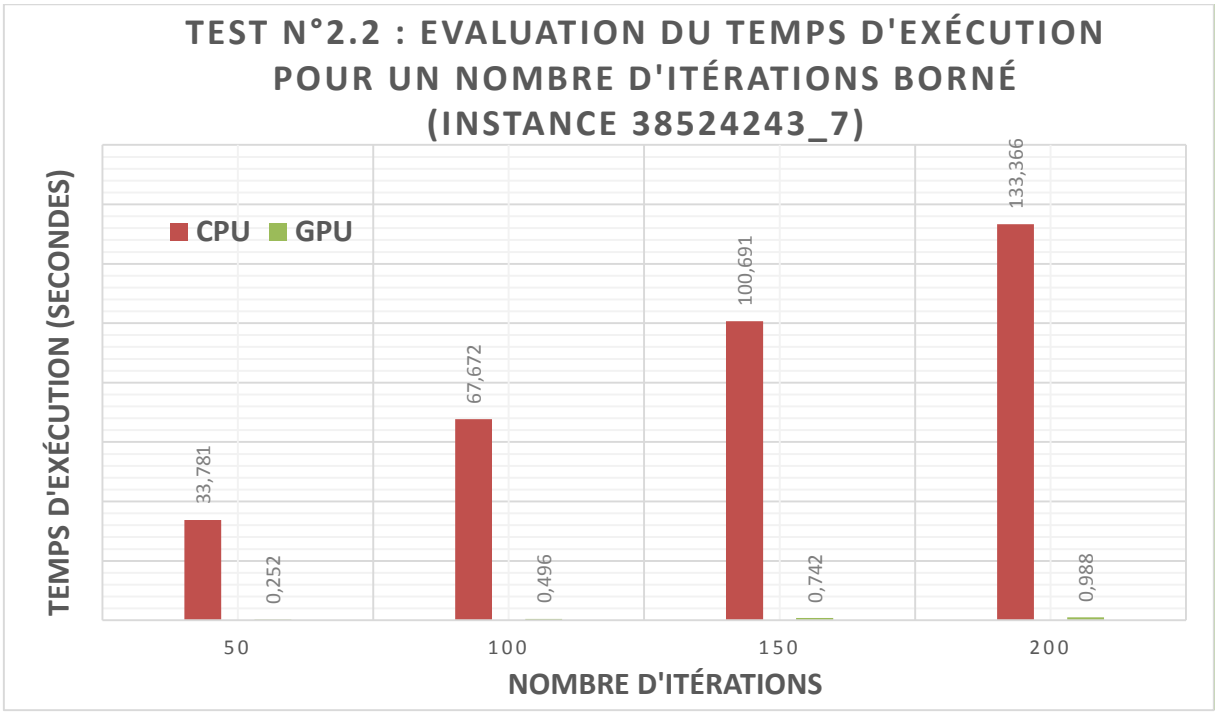

Figure V.4 : Temps CPU vs GPU pour 50, 100, 150 et 200 itérations [38524243_7]

Le temps nécessaire pour l'exécution avec [38524243_7] est entre 33,7s pour 50 itérations et 133,3s pour 200 itérations, une durée trop longue par rapport aux algorithmes existants (PALS de E.Alba et G.Luque [2007] prend 27s pour la résolution complète de la même instance). Par contre, l'exécution du même calcul sur GPU est très rapide et prends des temps qui ne dépassent pas l'échelle de la seconde (de 252ms pour 50 itérations à 988ms pour 200 itréations). Les mêmes remarques sont à noter pour l'instance [38524243_4]. L'accélération due à l'emploi d'un processeur graphique de l'architecture CUDA croît d'une manière exponentielle avec le nombre d'itérations exécutées.

C. CPU vs GPU : Temps moyen par itération

La dernière série de test représente l'exécution de l'algorithme, i.e. jusqu'à l'arrivée à un optimum. Le temps et le nombre d'itérations sont enregistrés pour chaque exécution pour déterminer le temps moyen nécessaire à l'achèvement d'une seule itération. Le diagramme en bâtons (figure V.5) regroupe les résultats de ces tests, réalisés sur les différentes instances disponibles.

Pour [X60189_4], une itération prend 1,53ms sur CPU et 1,42 en GPU. Les valeur sont proches car la fraction de calcul parallèle pour cette instance, considérée la plus petite du jeu de

tests, n'est pas très importante ; Le programme passe par conséquent moins de temps dans le kernel et ne gagne que quelques dizaine de microsecondes par rapport à son équivalent séquentiel.

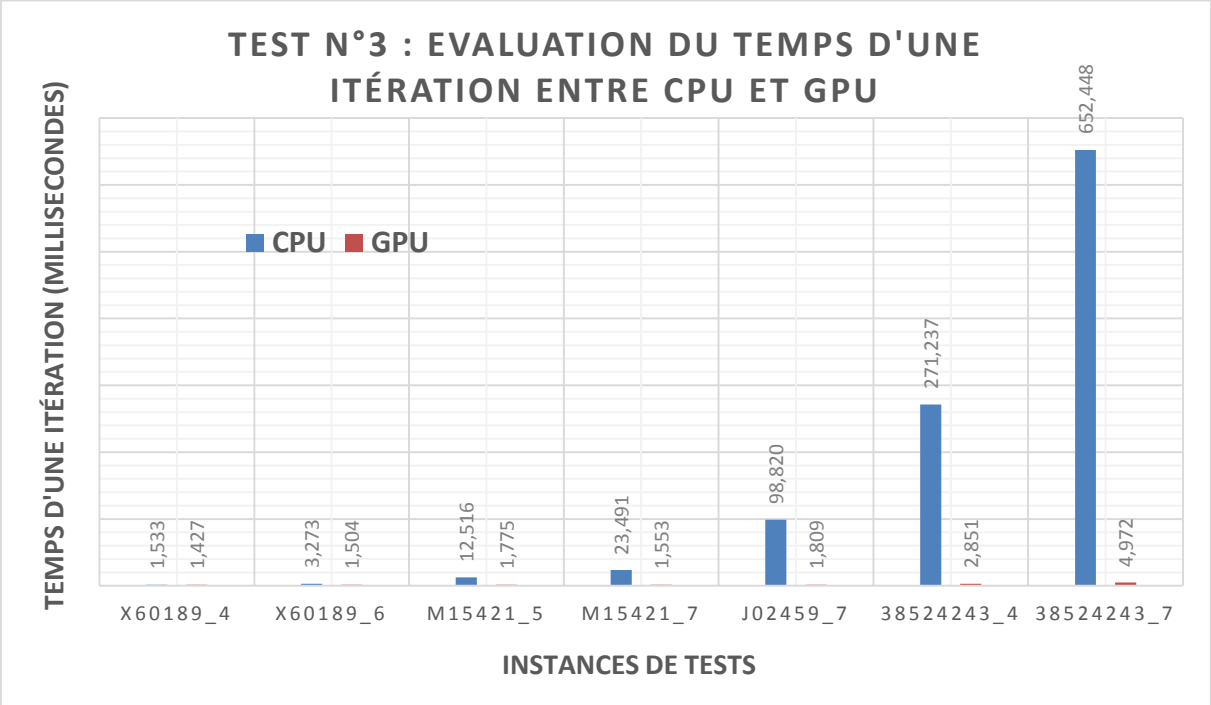

Figure V.5 : CPU vs GPU Temps moyen par itération pour différentes instances

L'écart entre CPU et GPU dépasse le double pour la deuxième instance [X60189_6] et devient plus grand en augmentant la taille du problème. La plus grande instance [38524243_7] ayant 773 fragments, prend 652,4ms par itération sur CPU, soit 131 fois le temps nécessaire pour que le même traitement s'achève sur GPU (4,97ms seulement).

V.5 Conclusion :

Nous avons présenté dans ce chapitre l'essentiel de notre proposition visant à fournir une méthode simple et efficace pour la résolution du problème d'assemblage de fragments ADN. Un processus de recherche locale itérative à plusieurs voisinages peut s'avérer complexe en temps d'exécution mais meilleur en qualité de solution qu'une simple recherche locale itérative mono-voisinage. Nous avons également proposé pour cette méthode deux modèles de parallélisme, puis implémenté le premier modèle en CUDA C. Les résultats des expérimentations nous ont permis d'optimiser le paramétrage du programme parallèle d'un côté, et nous ont doné d'un autre coté un aperçu réel sur l'écart de performance entre CPU et GPU. Il est à noter que les règles de

performance appliquées pour un programme CPU sont en grande partie remises en cause lors de l'évaluation d'un traitement GP-GPU, là où même des calculs supprimés (jugés inutiles) peuvent réduire la performance d'une implémentation.

Conclusion générale

Conclusion générale :

Le problème d'assemblage de fragments ADN est un problème qui est de nos jours assez connu. Avec le développement de technologies de séquençage, l'apparition de nouvelles banques de gènes ayant des caractéristiques spécifiques, ainsi que la croissance de la capacité des calculateurs contemporains, les biologistes, médecins, policiers et même particuliers, exigent des méthodes de plus en plus exactes et performantes.

Les méthodes conçues pour la résolution de DNA-FAP suivant un assemblage de novo sont très variées. Souvent classés en 3 familles, SCS, OLC et ESP, elles présentent le fruit de plus d'une cinquantaine d'années de recherche scientifique dans ce domaine. Les méthodes anciennes faisaient face à des contraintes qui ne sont plus présentes aujourd'hui, et les nouvelles méthodes trouvent toujours d'autres difficultés, de par la croissance très rapide du séquençage et son ouverture au large public. Bien que les prix actuels pour séquencer un génome humain complet aient considérablement baissé, ils restent tout de même loin d'être démocratisés.

Nous avons proposé dans ce document une nouvelle approche pour l'assemblage de fragments ADN qui atteint un bon niveau de performance et une qualité de solution acceptable pour des assemblages à petite échelle, avec deux modèles destinés à une implémentation sur CUDA, exploitant sa capacité à traiter d'immenses collections de données d'une manière parallèle. Nous avons implémenté et testé l'un de ces modèles pour des assemblages de différentes tailles, et fait des conclusions générales sur l'amélioration due à l'introduction de GPU dans le traitement d'un côté, et exploré quelques règles d'optimisation propres à l'architecture CUDA.

Perspectives :

Ce travail de recherche est loin d'être achevé. Il existe encore des points qu'il est intéressant d'explorer, citons par exemple :

- Amélioration de la performance au niveau séquentiel par le biais d'optimisation de structures mémoires, des révisions des segments de codes redondants, etc.
- Validation de la deuxième proposition citée par une implémentation des tests standards, plus rigoureux, afin de mesurer l'efficacité d'allégation du kernel et de l'uniformisation partielle du traitement.

- Utilisation de benchmarks réels pour mieux comprendre et mieux gérer les problèmes communs de séquences réelles.
- Proposition d'implémentations parallèles de quelques méthodes de recherche par population (algorithmes génétiques, essaims particulaires, colonie de fourmis, etc.) qui combinent la diversification et l'intensification des approches. Cela permettra de positionner notre algorithme dans un large éventail de choix, et déterminer ce qui est plus adéquat à la résolution de DNA-FAP.

Glossaire

Technique qui consiste à isoler et purifier un fragment d'ADN après l'avoir inséré et multiplié à l'identique dans une cellule hôte (souvent une bactérie). Le terme de clone fait alors référence à l'ensemble des cellules bactériennes dérivées par division mitotique de la seule cellule initiale ayant incorporé le fragment. **[32]**

ddNTP

Un didésoxyribonucléotide (ddNTP) est un nucléotide modifié (ne possède pas de groupe 3'-OH). Une fois incorporé dans un brin d'ADN, un didésoxyribonucléotide ne permet pas la liaison de nucléotides supplémentaires et empêche ainsi la croissance du brin en question.

dNTP

Un **dNTP** est l'appellation courante du mélange des quatre désoxyribonucléotides : dATP (désoxy adénine tri-phosphate), dCTP (désoxy cytosine tri-phosphate), dGTP (désoxy guanine tri-phosphate), dTTP (désoxy thymine tri-phosphate). Il est utilisé au cours d'une PCR (amplification d'ADN) comme élément de synthèse de l'ADN complémentaire.

Enzyme/Site de restriction

Une enzyme de restriction est une enzyme qui coupe l'ADN à la reconnaissance des séquences nucléiques spécifiques, ces zones s'appellent sites de restriction. Les enzymes de restriction se retrouvent souvent dans les bactéries et les archéobactéries, et fournissent un mécanisme efficace de défense contre les virus. Plus de 3000 de ces enzymes ont été étudiés en détail et environ 600 sont disponible commercialement. **[20]**

Gène

Unité élémentaire fonctionnelle dans le génome. Un gène correspond à une instruction à effectuer par la cellule. Le génome d'une bactérie en contient quelques milliers, celui des plantes ou des vertébrés (et donc de l'homme), de l'ordre de 30.000. Certains gènes codent des ARN aux fonctions cruciales, mais la plupart codent la séquence de protéines, qui sont synthétisées via des ARN messagers. **[19]**

Génome

Ensemble de l'information génétique d'un organisme, présente dans chacune de ses cellules. Le support matériel du génome est l'ADN (sauf chez certains virus, où il s'agit d'ARN), sous la forme de longues molécules nommées chromosomes. **[19]**

Nucléotide

Les nucléotides sont les molécules dont l'enchaînement constitue les brins d'ADN. Un nucléotide est formé d'une partie constante " un phosphate et un sucre " et d'une partie variable, une base azotée qui peut prendre quatre formes (voir Base). **[19]**

Oligonucléotide

Court fragment d'ADN, constitué de quelques (une ou plusieurs dizaines) molécules élémentaires (nucléotides). **[31]**

Pb

Paire de bases Dans un ADN double brin, paire de nucléotides complémentaires utilisé comme unité de mesure de la longueur d'une séquence.

PCR (réaction en chaîne de la polymérase)

C'est une technique d'amplification enzymatique qui permet à partir d'un fragment d'ADN, d'obtenir un grand nombre (plusieurs millions) de copies identiques de ce même fragment.

Séquence, séquençage

Le séquençage est l'opération qui consiste à déterminer l'ordre, ou séquence, dans lequel se succèdent les bases le long d'une molécule d'ADN. Une séquence se présente donc comme un texte plus ou moins long, écrit uniquement avec les quatre lettres A, T, G et C (voir Base). **[19]**

Sonde

Séquence d'ADN utilisée pour détecter par hybridation moléculaire la présence d'une séquence complémentaire. **[32]**

Vecteur

Molécule d'acide nucléique dans laquelle il est possible d'insérer des fragments d'acide nucléique étranger, pour ensuite les introduire et les maintenir dans une cellule hôte (utilisé dans les opérations de clonage moléculaire). Les vecteurs couramment utilisés sont les plasmides et les BAC dans lesquels sont insérés des fragments d'ADN de tailles de l'ordre de 1kb et de 100 kb, respectivement. **[31]**

Core

Désigne l'équivalent d'un processeur généraliste semiautonome qui partage un certain nombre de composants avec les autres cores. **[51]**

Streaming Multiprocessor

Grappe de processeurs ALU partageant une unité de contrôle et des mémoires. **[51]**

SIMD

Single instruction Multiple Data, type de processeur vectoriel utilisant une même instruction sur plusieurs données différentes. **[51]**

Banques

Désigne un segment sur une mémoire threads : élément de calcul s'exécutant sur le processeur scalaire. **[51]**

Kernel

Fonctions appliquées à chaque ensemble de données nécessitant l'application du même traitement. **[51]**

Warp

Groupement de 32 threads. **[51]**

Grille, Blocs

Le modèle de programmation considère une grille composée d'un ensemble de blocs qui exécutent simultanément des threads. **[51]**

Références bibliographiques

[1] : Froduald Kabanza, 2013, Université de Sherbrooke, Cours : Intelligence artificielle, La Recherche heuristique locale.

[2] : Damien pellier, 2012, Ecole D'Ingénieur de l'Université Paris Diderot, Cours : Intelligence artificielle et robotique, Algorithmes et recherches heuristiques.

[3] : LAI Hien Phuong, 2008, Institut de la Francophonie pour l'Informatique, Les Modèles Heuristiques et Metaheuristiques en Comet pour le problème Bin Packing.

[4] : Website : http://www.metaheuristics.net

[5] : Christine Solnon, Résolution de problèmes combinatoires et optimisation par colonies de fourmis.

[6] : Abdesslem LAYEB, 2010, Université Mentouri de Constantine, Thèse de docorat: Utilisation des Approches d'Optimisation Combinatoire pour la Vérification des Applications Temps Réel.

[7] : Professeur Sophie TISON, Université Lille 1, Cours : (Méta-)Heuristiques.

[8] : Sebastien Le Digabel, 2013, Ecole Polytechnique de Montreal, Introduction aux métaheuristiques.

[9] : Website : http://fr.wikipedia.org/wiki/M%C3%A9taheuristique

[10] : Tarek CHAARI, 2011, Université de Valenciennes et du Hainaut-Cambrésis, Thèse de doctorat : Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride.

[11] : Blum C and Roli A, 2003, Université Libre de Bruxelles and Université degli Studi de Bologna, Metaheuristics in combinatorial optimization : Overview and conceptual comparison.

[12] : Yves Deville, 2007, Cours : Constraint Based Local Search.

[13] : LAURIERE, 1978, A language and a program for starting and solving combinatorial problems, Artificial Intelligence

[14] : Voudouris, C. and Tsang, E.P.K, 2003, Kluwer Academic, Handbook of Metaheuristics, chapter Guided local search.

[15] : David MEIGNAN, 2008, Université de Technologie de Belfort-Montbéliard, Thèse de doctorat une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques.

[16] : AMROUCHE Hakima, 2012, UNIVERSITE MOULOUD MAMMERI, Thèse : Traitement d'Images et Reconnaissance de Formes.

[17] : E.Talbi, 2002, Laboratoire d'Informatique Fondamentale de Lille, A taxonomy of hybrid metaheuristics . Journal of Heuristics.

[18] : Website: David DUVIVIER, Résolution de problèmes industriels http://www.fucam.ac.be/redirect.php3?id=36647

[19] : Glossaire du Centre National de Séquençage " Genoscope "

[20] : Roberts RJ, November 1976, CRC Crit. Rev. Biochem , "Restriction endonucleases".

[21] : Enrique Alba and Gabriel Luque, 2007, Université de malaga, A New Local Search Algorithm for the DNA Fragment Assembly Problem.

[22] : Enrique Alba, Gabriel Luque, Pascal Bouvry and Bernabe Dorronsoro, 2008, A Self-Adaptive Cellular Memetic Algorithm for the DNA Fragment Assembly Problem.

[23] : Martine POITEVIN, 2008, Thèse de Doctorat de l'Université Paris VI – Pierre et Marie Curie, Contribution au développement d'un microsystème pour la séparation bidimensionnelle de protéines par électrophorèse.

[24] : Pavel A. Pevzner, 2006, Springer-Verlag France, Paris, Bio-informatique moléculaire - Une approche algorithmique.

[25] : FRÉDÉRIQUE BARROY-HUBLER, 2003, INRA Editions Paris, Principes des techniques de biologie moléculaire,

[26] : BRUTO Maxime 2010, Université de Nancy, Assemblage et analyse comparative du génome des souches.

[27] : Valérie Duranton-Tanneur, 2012, Laboratoire de Génétique des Tumeurs Solides, Séquençage à Haut Débit et applications.

[28] : EL FAHIME Elmostafa et Ennaji Mly Mustapha, LES TECHNOLOGIES DE LABORATOIRE - N°5 Juillet-Août 2007, Laboratoire de Virologie, Hygiène et Microbiologie, Évolution des techniques de séquençage.

[29] : S. LE CROM, Direction de l'Evaluation des Médicaments Et des produits Biologiques, 2011, Le sequençage a haut debit methodes et enjeux en medecine, pharmacologie et toxicologie.

[30] : Olivier.Delgrange, 2011, Laboratoire informatique fondamentale Lille, séquençage de génomes - assemblage de fragments d'adn.

[31] : Glossaire de génétique moléculaire et génie génétique website : http://www.myonet.org/GENETIQUE/glossaire.html

[32] : LEXIQUE des termes de génétique et de biologie moléculaire, 2004 - 2006, AGENOP.

[33] : Futura Science, 2013, website : http://www.futura-sciences.com.

[34] : Nvidia, 2010, GF100 whitepaper. World's Fastest GPU Delivering Great Gaming Performance with True Geometric Realism.

[35] : André R. Brodtkorb, Trond R. Hagena, Martin L. Sætra, 2012, Université de Oslo, Graphics processing unit (GPU) programming strategies and trends in GPU computing.

[36] : Jason Sanders et Edward Kandrot ,2011, Cuda by exemple.

[37] : l'architecture cuda de nouvelle génération, website : (http://www.nvidia.fr/object/fermi_architecture_fr.html)

[38] : Sylvain Jubertie, 2011-2012, Laboratoire d'Informatique Fondamentale d'Orléans, NVIDIA CUDA Compute Unified Device Architecture.

[39] : Raksmey PHAN, 2010, Prise en main de la technologie CUDA pour la programmation sur GPGPU.

[40] : NVIDIA, 2009, NVIDIA's Next Generation, CUDATM Compute Architecture Fermi

[41] : Flash informatique N° 02, 2009, les gpu ne sont pas uniquement faits pour les consoles de jeux, website : http://flashinformatique.epfl.ch/spip.php?article1715.

[42] : Article de hardware.fr sur L'API CUDA, website : http://www.hardware.fr/articles/659-4/nvidia-cuda-apercu.html

[43] : Une introduction à CUDA, 2009-2011,
http://tcuvelier.developpez.com/tutoriels/gpgpu/cuda/introduction/

[44] : Jean-Michel Richer, Maître de Conférences en Informatique, 2013, cours : CUDA.

[45] : Guillaume VIGUIE, 2008, Université Toulouse III, Thèse de doctorat : apport de la programmation graphique pour la reconstruction rapide d'images 3d en tomographie par emission monophotonique.

[46] : Mohamed Esseghir LALAMI, Université Toulouse III, Thèse de doctorat : Contribution à la résolution de problèmes d'optimisation combinatoire: méthodes heuristiques et parallèles.

[47] : Thomas Izard, 2011, Université Montpellier 2, Thèse de doctorat : Opérateurs arithmétiques parallèles pour la cryptographie asymétrique.

[48] : Raksmey PHAN, 2009, laboratoire LIMOS, thèse d'ingénieur : Étude des technologies CUDA pour l'Optimisation Discrète en Tournées de Véhicules.

[49] : Alexandre Chariot, 2008, Ecole des Ponts Certis, Thèse de doctorat : Quelques Applications de la Programmation des Processeurs Graphiques à la Simulation Neuronale et à la Vision par Ordinateur.

[50] : Nvidia Developer Zone, website : https://developer.nvidia.com/

[51] : Francis.Lapique, 2009, Les GPU ne sont pas uniquement faits pour les consoles de jeux.

[52] : Pevzner et *al.* 2001, Université de California à San Diego, A New Approach To Fragment Assembly in DNA Sequencing.

[53] : R. Idury and M. Waterman, 1995, Journale de Computational Biology, A new algorithm for DNA sequence assembly.

[54] : Xu et *al*, 2012, Université de Beijing Jiaotong Chine, An efficient algorithm for DNA fragment assembly in MapReduce.

[55] : Alba et al, 2007, Université de Malagé Espagne, A New Local Search Algorithm for the DNA Fragment Assembly Problem.

[56] : Jia Zheng ,2004, Université nationale de Singapour, An Improved Algorithm for Error Correction of Reads in DNA Fragment Assembly.

[57] : Engle ML, Burks C. 1993, Los Alamos National Laboratory, Artificially Generated Data Sets for Testing DNA Sequence Assembly Algorithms.

Résumé :

Le séquençage d'un ADN revient à lire la séquence de bases nucléotidiques A, T, C et G le constituant. Les technologies actuelles ne permettent pas d'effectuer un séquençage direct que sur des génomes de petite taille. C'est ainsi qu'est apparu le séquençage aléatoire (shotgun) consistant en la fragmentation de l'ADN, puis le séquençage individuel des fragments, et enfin l'assemblage des séquences résultantes à base de leurs ressemblances deux à deux. La résolution d'un problème d'assemblage de fragments ADN, de classe NP-difficile, est une partie importante de ce processus.

Plusieurs travaux de recherches ont été menés afin de mettre en œuvre des assembleurs à la fois précis, complets et efficaces. La nature du problème fait du regroupement de ces trois critères un idéal jamais atteint. Les métaheuristiques présentent un outil essentiel permettant d'atteindre des résultats acceptables et relativement bons, et le calcul parallèle permet une amélioration considérable de la performance des solutions proposées.

Nous proposons dans ce travail un algorithme d'assemblage de fragments ADN, qui consiste en une recherche locale itérative à double voisinage, ainsi qu'un modèle de parallélisation destiné à une exécution sur GPU. L'algorithme a été implémenté en utilisant le langage C, et son équivalent GPU est écrit en CUDA C, exécutable sur les technologies nVIDIA. Les tests sont réalisés sur des séquences ADN benchmarks générées artificiellement.

Mots-clés : Assemblage de novo, métaheuristiques, séquençage ADN, GP-GPU, CUDA.

Abstract :

The goal of DNA sequencing is to read the nucleic bases A, T, C and G composing the sequence. Current technology cannot perform direct sequencing except for short-length genomes. That's how began the shotgun sequencing technique, consisting in DNA fragmentation, followed by individual fragment sequencing, and finally the fragments' assembly on a pair resemblance basis. Solving a DNA fragment assembly problem, classified as NP-hard, is an important part of this process.

Many researches have been done in order to build up precise, complete and efficient assemblers at once. The nature of the problem makes regrouping these three criterions an never reached ideal. Metaheuristics are considered an essential tool allowing to reach acceptable and relatively good results, and parallel computing can result in considerable improvements in terms of performance of the proposed solutions.

We propose in this work a DNA fragment assembly algorithm, which consists of a double-neighborhood iterated local search, also a parallel model designed to be run on GPU. The algorithm is implemented using C language, and its GPU-equivalent is written in CUDA C, runnable on nVIDIA technologies. Test are performed on artificially generated benchmark DNA sequences.

Keywords : de novo assembly, metaheuristics, DNA sequencing, GP-GPU, CUDA.

Introduction Générale

Chapitre I
Introduction aux métaheuristiques

Chapitre II

Le problème d'assemblage de fragments ADN

Chapitre III

CUDA et le calcul sur processeur graphique

Chapitre IV
Etude bibliographique des travaux réalisés

Chapitre V
Proposition et Résultats

Glossaire

Références bibliographiques

Conclusion générale