MASTER ACSI 2ème Année (UPMC)
Stage effectué à l'IRISA de RENNES

Rapport de Spécification

Synthèse automatique sur plateforme FPGA

Auteur : Hélène DAROLLES

Encadrant de Stage : Dominique LAVENIER

le 26 juin 2006

Table des matières

Première partie
Définition et analyse du problème

1 Introduction

Le but de ce stage est d'adapter l'outil de synthèse GAUT à l'architecture ReMIX.
Nous allons dans un premier temps définir et analyser le problème en expliquant le fonctionnement du système ReMIX puis celui de l'outil GAUT. Nous proposerons ensuite une solution de principe et identifierons les tâches à accomplir. Enfin, nous décrirons en détails la procédure de recettes utilisée pour valider notre projet.

2 Projet ReMIX

Le projet ReMIX a pour objectif de proposer une architecture matérielle destinée à accélérer la recherche d'information dans les masses de données. Son coeur est une mémoire de très grande capacité étroitement couplée à des ressources reconfigurables.

Cette architecture permet aussi d'accéder rapidement aux données (comparativement à des supports de stockage de type disque magnétique) et d'envisager des stratégies de recherche innovantes orientées, notamment, sur l'indexation des données.

La programmation du système ReMIX est prise en charge par un *framework* qui guide le concepteur depuis la spécification d'une nouvelle application jusqu'à sa mise en oeuvre sur le support reconfigurable.

2.1 Plateforme ReMIX

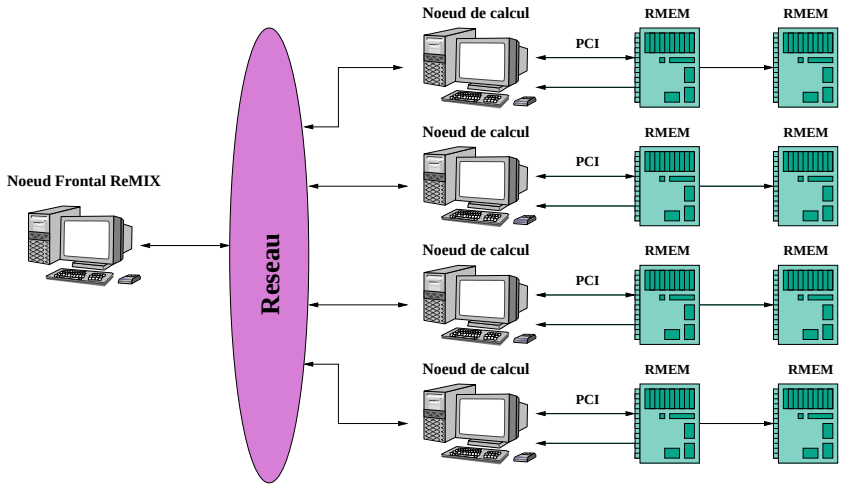


FIG. 1 – *Plateforme REMIX : architecture générale*

Le système ReMIX se présente sous la forme d'un cluster de 4 noeuds de calcul commandés par une machine hôte (noeud frontal).

Chaque noeud de calcul est un PC enrichi de deux cartes *mémoires reconfigurables* : les cartes RMEM, que nous étudierons plus en détails par la suite. Ces cartes embarquent chacune 64 Go de mémoire flash et disposent de ressources reconfigurables conséquentes pour effectuer des traitements à la volée en sortie de la mémoire.

Le système complet dispose donc d'une mémoire de 512 Go associée à quelques 240 000 cellules logiques.

2.2 Carte RMEM

La carte RMEM est le coeur du système ReMIX. Sa conception a eu pour objectif d'associer sur un même support une mémoire de grande taille et des ressources reconfigurables utilisées pour réaliser un traitement à la volée des données au fur et à mesure de leur lecture.

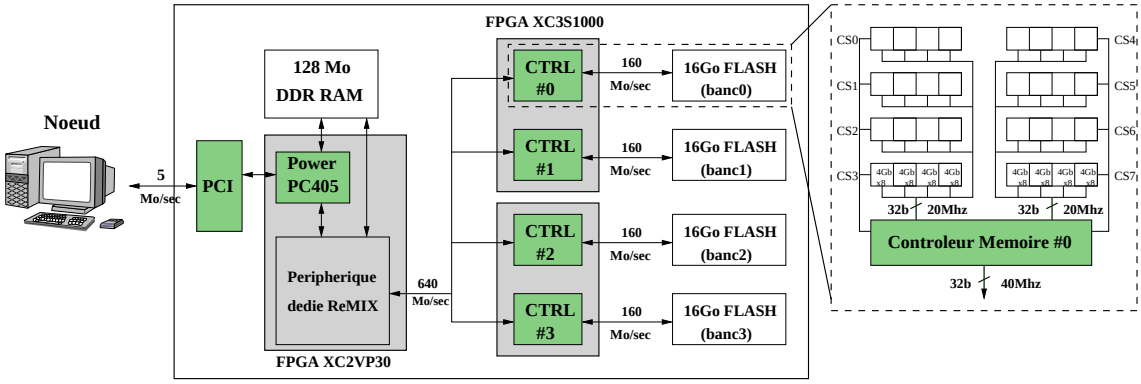


FIG. 2 – *Architecture d'une carte RMEM*

La figure ci-dessus décrit son architecture au niveau système. La carte RMEM se base sur une carte de développement pour FPGA, à laquelle a été associée une carte d'extension mémoire Flash-NAND.

La carte de développement est basée sur une interface PCI et dispose d'un FPGA Virtex-II Pro associé à 128 Mo de SDRAM. Le Virtex-II Pro embarque l'équivalent de 30 000 cellules logiques, 136 blocs mémoire de 18 Kbits et deux coeurs de processeur PowerPC405.

La carte mémoire est composée de 64 boîtiers mémoire de 1 Go; chaque boîtier intègre deux composants Flash qui peuvent être utilisés de manière concurrente. Ceux-ci sont regroupés en 4 bancs indépendants de 16 Go, chaque banc étant organisé en une matrice de 8 x 4 composants dans laquelle on peut accéder à 8 composants en parallèle.

Chaque carte mémoire contient également deux circuits FPGA utilisés pour implémenter les contrôleurs des 128 composants Flash et pour gérer l'interface entre chacun des 4 bancs mémoire et le circuit Virtex-II Pro de la carte PCI.

Le Virtex-II Pro intègre une architecture de type système sur puce construite autour du processeur PowerPC et d'un bus CoreConnect. Le système intègre plusieurs périphériques (contrôleur mémoire SDRAM, DMA, etc.) connectés au bus sytème, parmi lesquels un contrôleur dédié aux échanges de données (lecture et écriture) entre la carte mémoire Flash et la mémoire SDRAM de la carte PCI.

C'est ce périphérique d'interface que nous allons étudier et que nous désignerons, par la suite, périphérique ReMIX.

2.3 Périphérique ReMIX

Le rôle de ce périphérique, dont l'architecture est représentée ci-dessous, est double :

- Il gère les opérations de lecture et d'écriture des données de/sur la carte mémoire Flash;
- Il intègre un bloc *opérateur* matériel spécialisé pour chaque application qui traite à la volée les données lues à partir de la mémoire Flash.

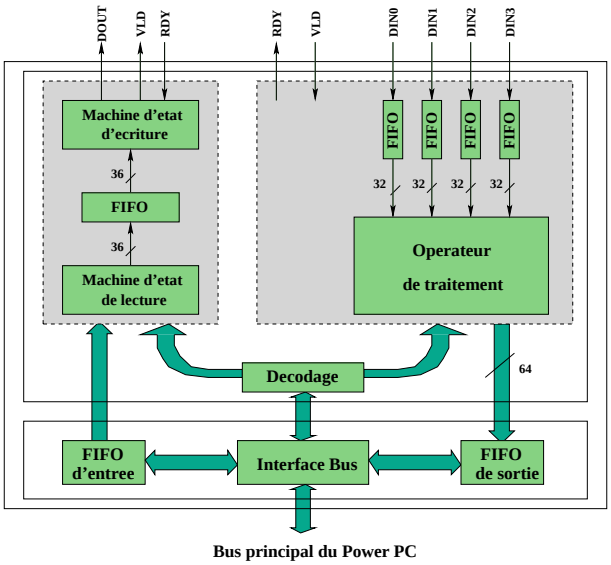


FIG. 3 – *Schéma de principe du périphérique dédié ReMIX*

L'opérateur reconfigurable peut recevoir simultanément 4 flux de données (4 x 32 bits) en provenance des 4 bancs de la mémoire Flash et produit en résultat un flux de données sur 64 bits. La synchronisation, en entrée comme en sortie, est assurée par des mémoires de type FIFO. 15 000 cellules logiques et 90 blocs mémoire RAM de 18 Kbits sont disponibles pour implémenter l'opérateur ReMIX, dont le fonctionnement est directement contrôlé par le PowerPC au travers du bus système.

Grâce à l'accès en parallèle de plusieurs composants Flash, la bande passante mémoire en entrée de l'opérateur est de 640 Mo/sec, soit deux ordres de grandeur de plus que celle disponible sur l'interface PCI de la carte ReMIX. Le bus PCI est donc un goulot d'étranglement qui limite l'intérêt d'une utilisation directe de cette mémoire par l'hôte, puisque l'on dispose alors d'une bande passante trés réduite (inférieure par un ordre de grandeur à celle d'un disque dur), ce qui réduit sensiblement les gains liés au temps d'accès aléatoire faible de la mémoire Flash (toujours par rapport au disque).

En pratique, ce type d'échange est exclu : le principe de fonctionnement de ReMIX consiste à utiliser l'opérateur matériel pour filtrer les données directement à la source et ainsi ne transmettre que l'information utile. Dans ce contexte, et en supposant que le taux de filtrage de l'opérateur est suffisant, la bande passante délivrée par l'interface PCI est suffisante. On peut résumer le déroulement d'un traitement sur cette architecture, c'est à dire la recherche et le traitement d'un objet dans la base d'index, comme suit :

1. un ou plusieurs ordres de lecture sont envoyés depuis l'hôte (via le processeur PowerPC) au contrôleur Flash par le périphérique d'interface ReMIX;
2. l'opérateur ReMIX est paramétré par l'hôte (toujours via le PowerPC) en vue du traitement des données qui seront lues à partir de la mémoire Flash;
3. les données lues, disponibles dans les FIFOs à l'entrée de l'opérateur, sont alors filtrées à la volée par ce dernier qui génère un flux de résultats;
4. ce flux résultat est transféré vers la mémoire SDRAM par l'utilisation d'un DMA pour être finalement envoyé à l'hôte par l'interface PCI;
5. une fois le traitement terminé, l'opérateur signale la fin des calculs au PowerPC par un signal d'interruption.

2.4 Opérateur ReMIX

2.4.1 Interface de l'opérateur

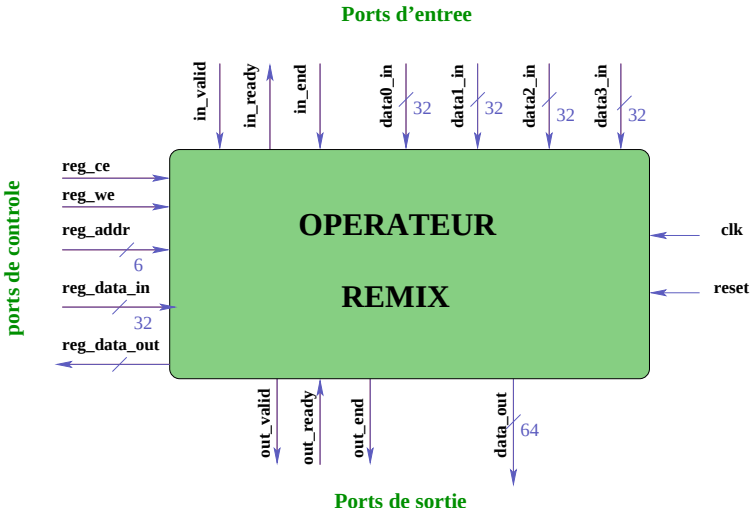


FIG. 4 – *Interface de l'opérateur ReMIX*

2.4.2 Alimentation des ports d'entrée

Signification des signaux :

- **in_valid** : activé lorsque toutes les données en entrée sont valides sur les ports in_data(0) à in_data(3). Il reste à l'état haut tant que la donnée n'a pas été acquitée sur les ports d'entrée des données.
- **in_ready** : activé lorsque l'opérateur acquitte les données valides en entrée.
- **in_end** : activé lorsque la dernière donnée arrive sur le port d'entrée des données et le reste tant que in_valid est à l'état haut.

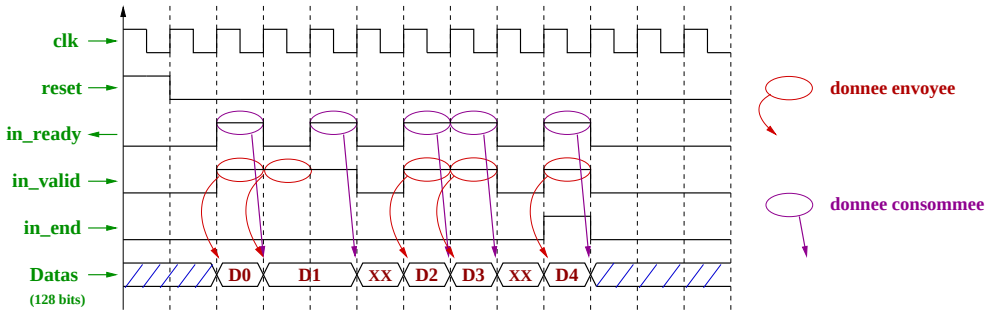


FIG. 5 – *Alimentation des ports d'entrée de l'opérateur*

2.4.3 Récupération des résultats sur les ports de sortie

Signification des signaux :

- **out_valid** : activé lorsque la donnée en sortie de l'opérateur est valide.
- **out_ready** : activé lorsque la donnée en sortie est acquittée.
- **out_end** : désigne la fin d'un cycle du filtre.
  Activé pendant que le dernier résultat sort de l'opérateur.

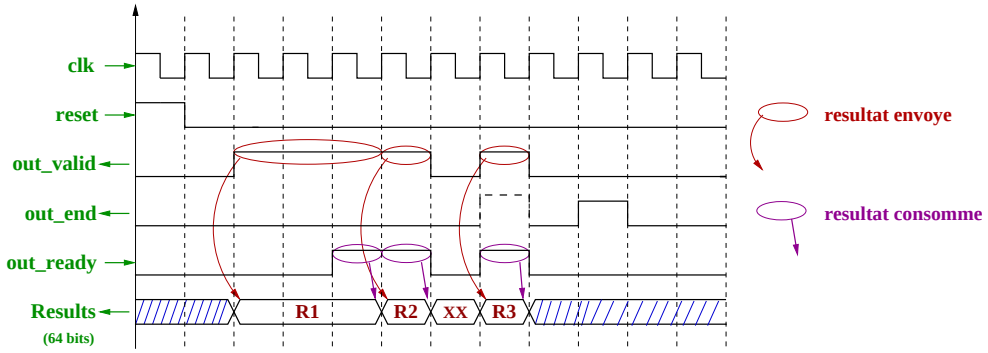


FIG. 6 – *Alimentation des ports de sortie de l'opérateur*

2.4.4 Ports de contrôle de l'opérateur

Signification des signaux :

- **reg_addr** : adresse de la donnée.
- **reg_data_in** : 32 bits de données destinés au port du registre de l'opérateur.
- **reg_data_out** : 32 bits de données destinés à être écrits dans le registre.
- **reg_ce** : activé lorsqu'on veut permettre l'accès au registre (et seulement si reg_we, reg_addr et reg_data_in sont prêts).
- **reg_we** : à l'état haut : accès au registre en écriture.
  à l'état bas : accès au registre en lecture.

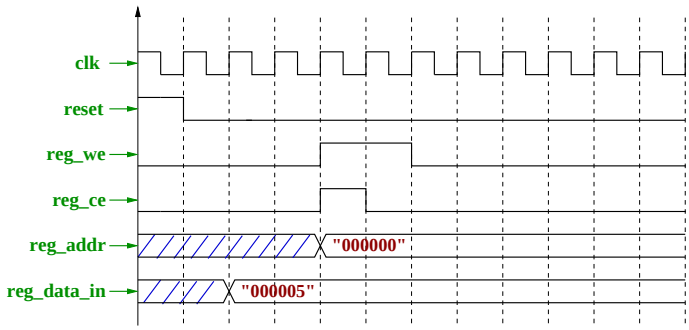


FIG. 7 – *Chronogramme des signaux de contrôle*

2.5 Le problème de la programmation d'un opérateur

Il faut cependant noter que la conception automatique des opérateurs sur la structure reconfigurable de ReMIX n'est pas prise en charge : une interface système assure simplement les échanges entre le processeur d'un noeud de calcul et l'opérateur de la carte RMem. L'opérateur doit être conçu indépendamment.

Ainsi il faut décrire en VHDL les traitements pour chaque nouvelle application. C'est un processus que seul un expert en architecture peut réaliser. C'est donc un frein sérieux à l'usage de ReMIX puisqu'il réduit le nombre d'utilisateurs potentiels.

Cet inconvénient peut être supprimé par l'utilisation d'un outil de synthèse de haut niveau.

3 GAUT : un outil de synthèse de haut niveau

3.1 Principe Général

L'outil de synthèse de GAUT développé au LESTER (Université de Bretagne Sud - Lorient) est un outil de synthèse de haut niveau prenant en entrée une description algorithmique (dans langage dérivé du C, que nous étudierons par la suite et qui sera nommé GAUT-C) et génèrant une architecture matérielle décrite en VHDL et destinée à être synthétisée par des outils de synthèse logique du commerce (ISE/Foundation de Xilinx, Quartus de Altera,...).

Cet outil est un outil complet constitué de tout le flot de synthèse comportemental nécessaire pour générer, à partir d'une description comportementale, la description RTL structurelle associée.

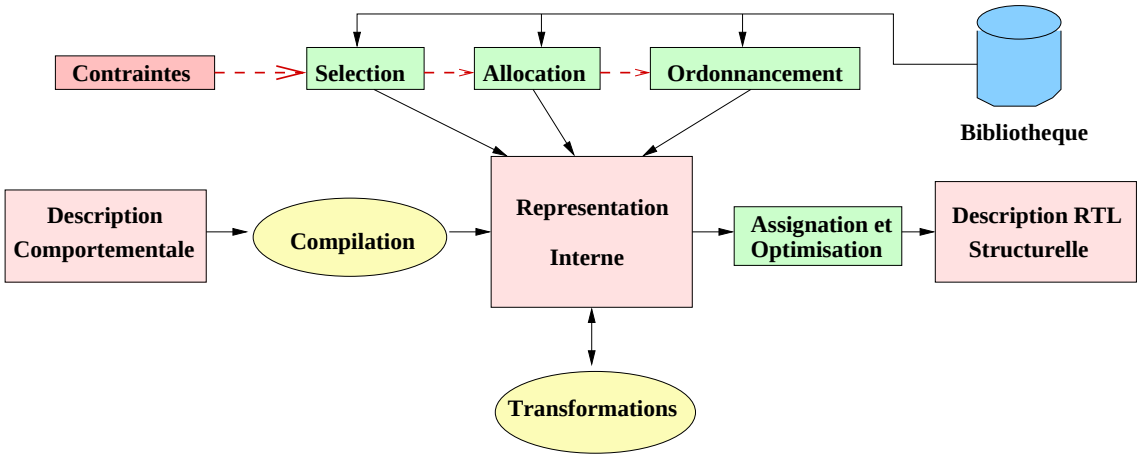


FIG. 8 – *Flot de synthèse de GAUT*

La **Compilation** permet d'analyser le code de la spécification, de le paralléliser et de le transformer afin d'obtenir une représentation exploitable.

La **Sélection** permet de choisir, à partir d'une bibliothèque complète d'opérateurs, les composants dont les caractéristiques répondent aux contraintes de l'application (fonction de coût et heuristique).

L'**Allocation** permet de déclarer la quantité exacte d'opérateurs que l'on désire utiliser ainsi que les intervalles de temps où ils sont utilisables.

L'**Ordonnancement** permet d'affecter une date d'exécution à chaque opération de l'application.

L'**Assignation** permet d'affecter une opération à un opérateur. Cette étape doit minimiser les problèmes de connectique interne à l'unité de traitement posés par la circulation des données : réduction des registres.

Voici ci-dessous le flot de conception de l'outil GAUT qui, à partir d'un algorithme décrit en GAUT-C, génère une architecture composée d'une unité de traitement, d'une unité de mémoire et d'une unité de communication.

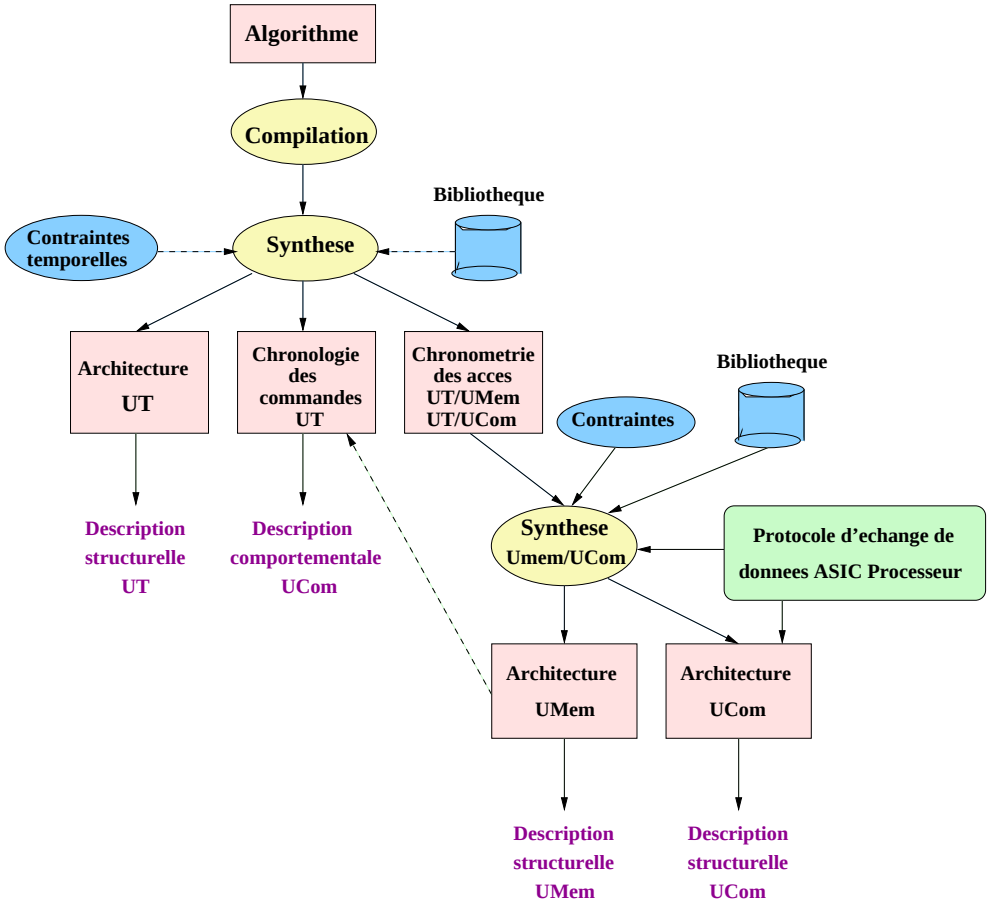


FIG. 9 – *Flot de conception de GAUT*

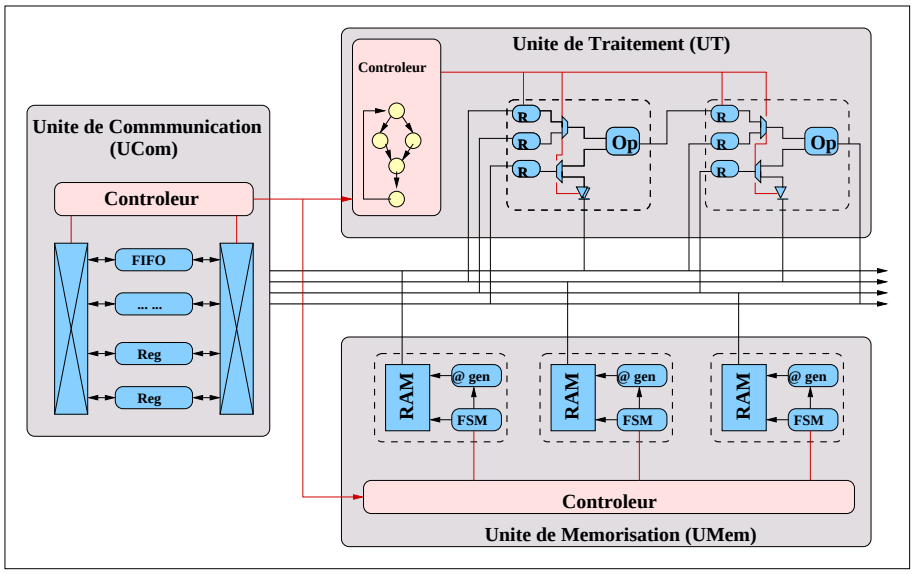


FIG. 10 – *Architecture générée par GAUT*

3.2 Etapes de synthèse avec GAUT

3.2.1 Compilation

La compilation consiste à analyser le code de la spécification, à le paralléliser et à le transformer afin d'obtenir une représentation interne exploitable.

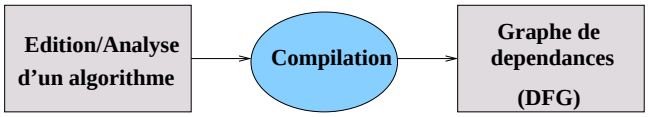


FIG. 11 – *Compilation avec GAUT*

Le compilateur de Gaut élimine les fausses dépendances, les redondances (constantes, expressions communes, invariant de boucle), du contrôle (déroulage de boucles, multiplexage des if) et optimise le code (code mort, remplacement d'opérations).La syntaxe GAUT-C acceptée est la suivante :

- C standard avec certaines restrictions et particularités;
- pas de pointeurs mais des références (&) et des indirections (*);
- pas de fichiers ni de flux;
- Types : int et dérivés (largeur de chemin de données de la bibliothèque), enum, struct (éclatement en scalaire) et tableaux (stockés en mémoire);
- Boucles : domaine d'itération fini et statique;
- Fonctions : opérations de la bibliothèque technologique;
- Procédures : mises en ligne (inline), pas de récursivité;
- Déclarations E/S : référence pour une sortie scalaire, qualificatif **const** pour une entrée tableau.

Pour illustrer la synthèse avec GAUT, nous avons écrit un algorithme trivial en GAUT-C qui fait la somme de 2 entrées (additionneur simple).

```
#define N 4
typedef vecteur[N];

int main (const vecteur E1, const vecteur E2, vecteur res)
{
   int i;
   for(i=0;i<4;i++)  res[i] = E1[i] + E2[i];
   return 0;
}
```

Le DFG (Data Flow Graph) de l'additionneur obtenu est le suivant :

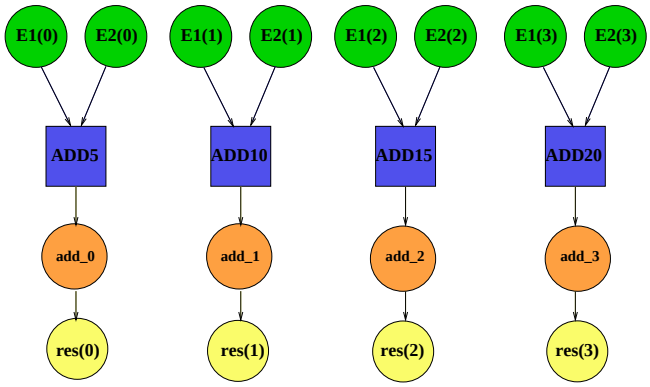


FIG. 12 – *Data Flow Graph de l'additionneur*

Analyse du DFG :
- les cercles **verts** représentent les variables d'entrée de la fonction principale;
- les carrés **bleu** sont les opérateurs faisant l'addition 2 à 2 entre les entrées pour chaque itération;
- les cercles **orange** indiquent les unités de mémorisation des résultats en sortie des opérateurs;
- les cercles **jaune** représentent le résultat;

3.2.2 Synthèse VHDL

Après l'analyse, on effectue la synthèse du code en ajoutant des contraintes sur les E/S et sur les mémoires de la façon suivante :

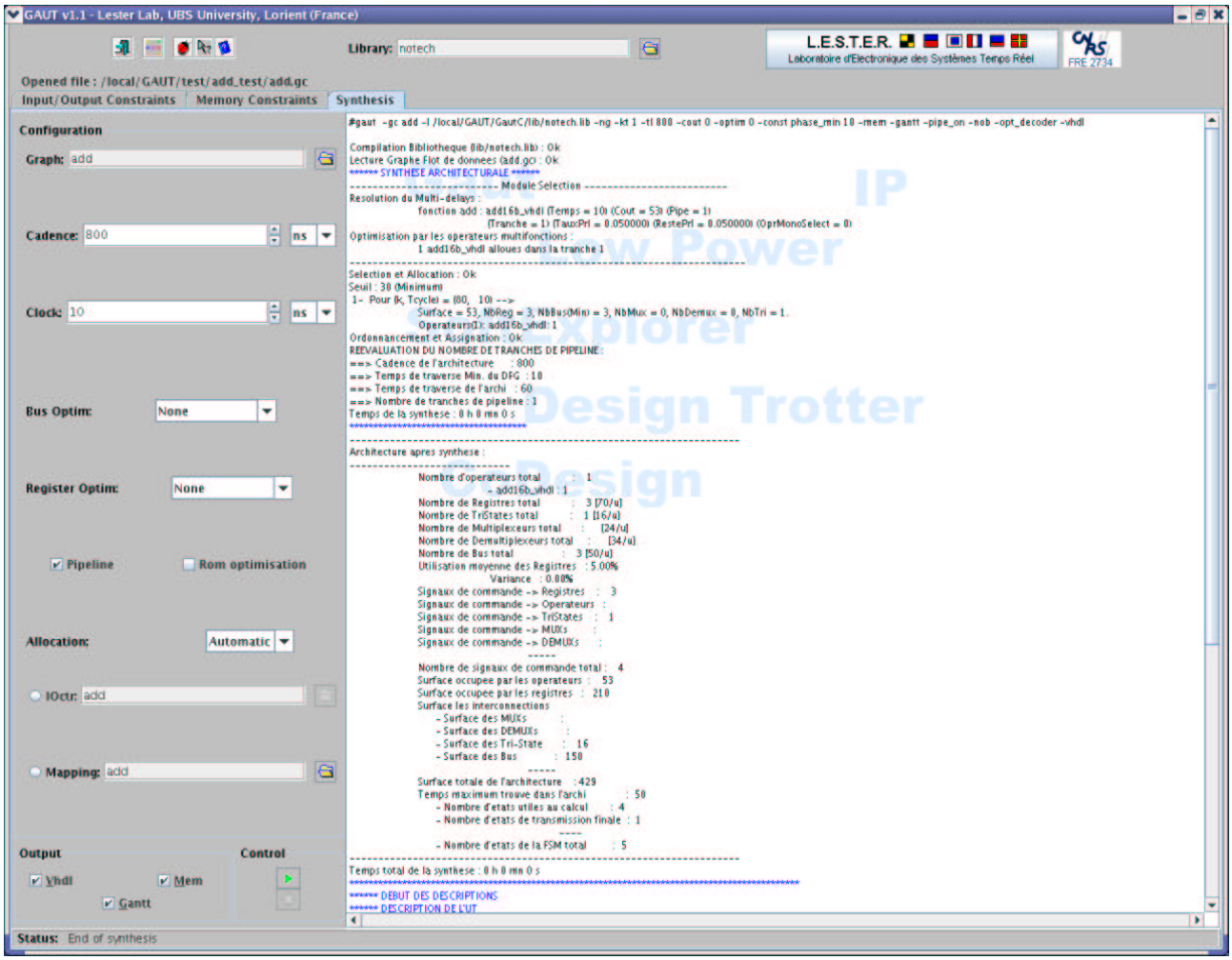


FIG. 13 – *Synthèse comportementale avec GAUT*

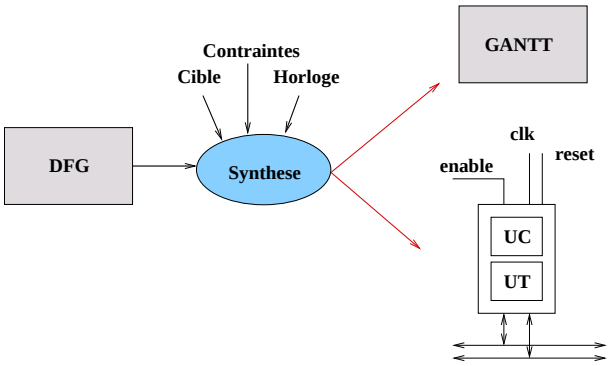


FIG. 14 – *Schéma de la Synthèse de l'Unité de traitement (UT)*

La **cible** représente la bibliothèque de composants RTL combinatoires :
- caractérisés en temps/surface
- outil de caractérisation automatique pour Xilinx/Altera.

Les **contraintes** peuvent être :
- temporelles :
  - cadence de réitération;
  - contraintes sur les E/S (ordre, date) : génération d'un wrapper.
- de mémorisation : mapping mémoire (banc, adresse)
  - des constantes;
  - des vecteurs vieillissants;
  - des variables rebouclées (adaptation);
  - des variables si durée de vie >seuil.

3.2.3 Résultats

Ainsi, la synthèse avec GAUT génère un tableau de **Gantt**, qui donne l'état de chaque composant à chaque coup d'horloge, ainsi que l'architecture de l'additionneur :

	... 0	... 10	... 20	... 30	... 40
register.2	E2(0)	E2(1)	E2(2)	E2(3)	
register.1	E1(0)	E1(1)	E1(2)	E1(3)	
add16b_vhdl.1	add5	add10	add15	add20	
register.3		res(0)	res(1)	res(2)	res(3)

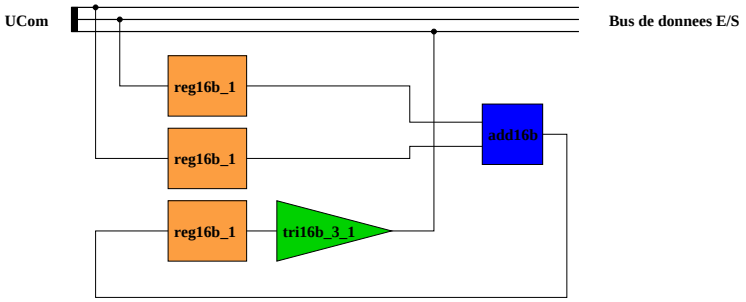


FIG. 15 – *Tableau de gantt et architecture finale obtenue pour l'additionneur*

On peut remarquer que l'architecture obtenue contient pour chaque E/S un registre et pour la sortie un tristate suivant le registre.

Deuxième partie

Proposition d'une solution de Principe

Pour adapter l'architecture en VHDL produite par l'outil GAUT à l'interface de l'opérateur ReMIX, nous allons devoir développer une interface permettant de récupérer les données en entrée de l'opérateur, les appliquer de façon transparente à l'architecture de GAUT et récupérer les résultats en sortie de l'architecture de GAUT et les diffuser à l'opérateur ReMIX.

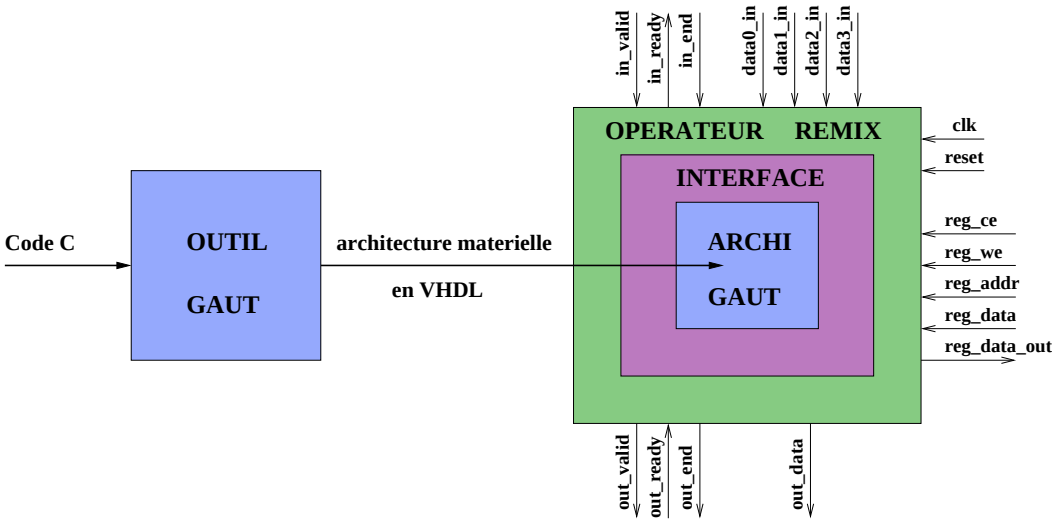


FIG. 16 – *Schéma du principe*

En principe, l'interface (carré mauve) combinée à l'archi GAUT (carré bleu) doit être une architecture compatible avec l'environnement ReMIX.

L'achitecture de cette interface devra pouvoir gérer l'alimentation des bus de données en entrée de l'architecture générée par GAUT à partir du flux de données en entrée de l'opérateur ReMIX, lui envoyer les paramètres utilisés et récupérer les résultats sur les Bus de sortie pour les transférer à l'opérateur ReMIX.

A terme, nous devons être capable de savoir si oui ou non l'interface entre l'opérateur ReMIX et l'architecture génére par GAUT sera facile à automatiser.

Troisième partie

Identification des tâches à accomplir et planning de Réalisation

4 Tâches à accomplir

4.1 Prise en main des outils

4.1.1 Prise en main de ReMIX

Dans un premier temps, nous allons devoir prendre en main la plateforme ReMIX afin de bien interfacer le résultat de GAUT pour l'opérateur ReMIX.

Pour ce faire, nous devons implémenter à la main un opérateur simple qui fait la somme de toutes les données qu'on reçoit en entrées et sort cette somme lorsqu'elle dépasse un certain seuil. Cela nous permettra d'analyser le fonctionnement de l'opérateur.

4.1.2 Prise en main de GAUT

Puis nous allons devoir prendre en main l'outil de synthèse GAUT, en implémentant un algorithme simple en GAUT-C et en analysant les étapes de sa synthèse avec cet outil.

4.2 Environnement de Test

Afin de faciliter la validation des implémentations de l'opérateur ReMIX, nous allons réaliser un environnement de test qui effectuera les étapes suivantes :

- phase de configuration :
  Elle permettra de paramétrer l'opérateur en vue d'un traitement des données qui seront lues à partir de la mémoire flash (envoie à l'opérateur de valeur de seuil, de valeur de référence à comparer ...etc.), et d'alimenter les ports de données en entrée de l'opérateur à partir des fifos d'entrée.
- phase de traitement :
  Phase pendant laquelle nous attendrons la fin du traitement des données par l'opérateur.
- phase de récupération des résultats :
  Cette phase gèrera le flux de résultats en sortie de l'opérateur au moyen d'une fifo.

Cette environnement de test fera l'objet d'une documentation, afin de faciliter son utilisation par d'autres développeurs.

4.3 Opérateur Génomique

Nous devrons ensuite implémenter un opérateur génomique représentatif de l'application.

Cet opérateur devra comparer un flux de séquences d'Acides Aminés par rapport à une séquence de référence en établissant un "score" de similitudes et de sortir une donnée sur 64 bits contenant l'identifiant de la séquence de référence sur 32 bits ainsi que les adresses sur 32 bits associées aux séquences proches de la séquence référence (celles dont le "score" dépasse un certain seuil).

4.3.1 Implémentation manuelle

Nous devrons dans un premier temps implémenter cet opérateur manuellement en VHDL. Nous validerons cet opérateur à l'aide de l'environnement de test réalisé précédemment.

4.3.2 Implémentation GAUT

Nous écrirons ensuite un algorithme en GAUT-C dont la fonction sera la même que celle implémentée manuellement.

Après la synthèse du code en GAUT-C, nous obtiendrons une architecture GAUT de l'opérateur.

4.3.3 Analyse

Nous analyserons ensuite l'architecture générée par GAUT obtenue.

4.4 Spécification à partir de l'analyse

Nous pourrons établir la spécification de l'interface GAUT/ReMIX à partir de l'analyse effectuée.

4.5 Développement de l'interface

En s'appuyant sur la spécification, nous pourrons développer l'interface entre l'architecture GAUT et l'opérateur ReMIX.

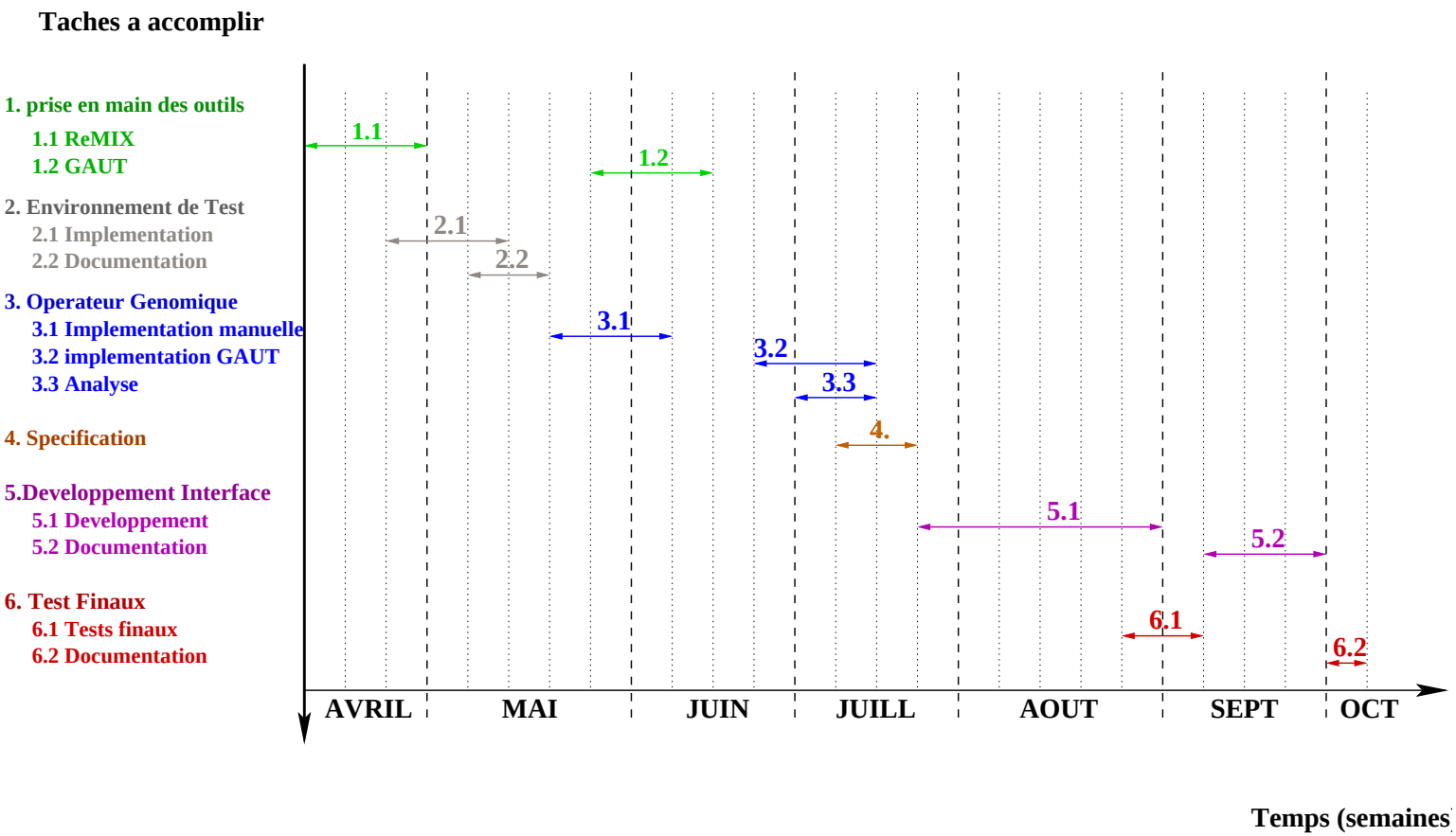
Cette interface fera l'objet d'une documentation pour permettre aux autres utilisateurs de l'exploiter.

4.6 Tests finaux

Avec un jeux de données conséquent et représentatif de l'application, nous testerons l'interface ainsi développée.

Nous fournirons en plus une documentation contenant tous les tests réalisés qui servira de support de test.

5 Planning de réalisation



Quatrième partie
Procédure de Recettes

6 Validation de l'environnement de Test

Pour pouvoir valider notre travail et vérifier nos résultats nous allons implémenter un environnement de Test. Celui-ci permettra, selon l'application souhaitée, de configurer correctement notre opérateur, puis de lui envoyer les données à traiter et enfin de récupérer les résultats en sortie de l'opérateur.

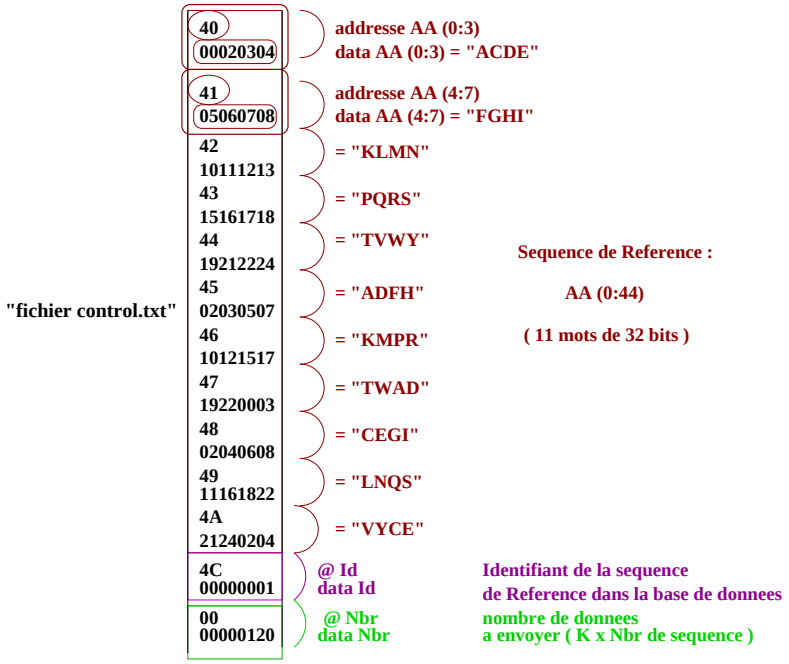
Après avoir effectué différents tests, une documentation détaillée pour l'utilisation de cet environnement de test sera réalisée.

Ainsi, les utilisateurs de l'environnement de test le valideront eux-mêmes au travers des remarques et des commentaires qu'ils pourront faire.

7 Validation de l'opérateur Génomique

Pour vérifier si l'implémentation manuelle de l'opérateur génomique est correcte, nous utiliserons l'environnement de test. Nous configurerons l'opérateur à l'aide du fichier de contrôle.

Voici un exemple de fichier contrôle afin de montrer son format de données :



Ainsi, pour cet exemple, la séquence d'Acides Aminés de référence est la suivante :

séquence Ref = VYCE LNQS CEGI TWAD KMPR ADFH TVWY PQRS KLMN FGHI ACDE

Après avoir paramétré l'opérateur ReMIX, nous allons lui envoyer un flux de séquences d'Acides Aminés en entrée, issues d'une base de données, qui seront comparées une à une à la séquence de référence.

Le résultat comportera des données sur 64 bits contenant l'identifiant de la séquence de référence ainsi que l'adresse, dans la base de données, de la séquence retenue. Ces résultats seront écrits dans le fichier "out.txt" qui aura le format suivant :

Identifiant de la sequence de Reference	@ de la sequence a comparer depassant le seuil
00000001	10000001
00000001	20000001
00000001	20000002
	•••

"fichier out.txt"

8 Validation de l'interface GAUT/ReMIX

Le test final consitera à tester l'interface entre l'architecture générée par GAUT, associée à l'algorithme en GAUT-C représentant le fonctionnement de l'opérateur génomique, avec l'opérateur ReMIX.

Nous configurerons l'opérateur à l'aide du même fichier de contrôle avec lequel l'opérateur génomique implémenté manuellement a été testé.
De plus, nous enverrons exactement les mêmes séquences à comparer que celles utilisées pour la validation de l'opérateur génomique implémenté manuellement.

Ainsi nous allons devoir vérifier que les résultats obtenus avec l'architecture générée par GAUT interfacée avec l'opérateur ReMIX sont bien les mêmes que ceux obtenus avec l'opérateur génomique fait à la main.