RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Réalisation matérielle d'automates pondérés pour la recherche de motifs génomiques

Mathieu Giraud, Dominique Lavenier

équipe Symbiose
IRISA, Campus de Beaulieu
35042 Rennes Cedex – France
mgiraud@irisa.fr

Résumé
Nous montrons que l'encodage linéaire *(one–hot)* permet de simuler efficacement les automates finis ainsi que les automates pondérés grâce au parallélisme massif de solutions matérielles comme celui des circuits reconfigurables FPGAs. On peut matérialiser des automates à t transitions au moyen de $\mathcal{O}(t)$ cellules matérielles et résoudre des problèmes de recherche de motifs de manière pipelinée en acceptant un caractère par cycle. Les automates pondérés modélisent les erreurs de substitution utilisées dans l'interrogation de bases de données génomiques. Ils nous mènent à envisager des traitements nouveaux et accélérés au moyen de R-disk, une architecture spécialisée offrant des traitements à la volée à la sortie de dispositifs de stockage.

Mots-clés : recherche de motifs, encodage linéaire, automates finis, automates pondérés.

1. Introduction

Bases de données génomiques. L'automatisation des outils de séquençage conduit depuis une vingtaine d'années à une croissance exponentielle des bases de données génomiques. En septembre 2003, la banque européenne *EMBL Nucleotide Sequence Database*, qui regroupe toutes les séquences nucléiques séquencées du domaine public dans le monde, contient $42 \cdot 10^9$ bases réparties en $28 \cdot 10^6$ entrées. Cette quantité double chaque année [10].
Les biologistes cherchent quotidiennement à obtenir à partir de ces données brutes un aperçu des fonctions métaboliques et donc finalement une compréhension de certains mécanismes biologiques. Cette compréhension peut être utilisée par exemple en pharmacologie (présélection informatique de substances actives sur certains protéines) ou en génie génétique. Elle passe cependant par des traitements lourds sur les données.
Domaines et motifs protéiques. La fonction métabolique d'une protéine dépend fortement de sa structure 3D, elle-même étant fonction de la séquence 1D des acides aminés qui la composent. Ceux-ci sont éléments de l'alphabet à 20 lettres $\Sigma_{AA} = \{A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y\}$. La comparaison des protéines entre elles et leur regroupement en *familles fonctionnelles* passe par l'étude des similitudes entre leurs séquences. La plupart du temps, les protéines d'une même famille proviennent d'un même ancêtre et comportent des *domaines* allant jusqu'à quelques dizaines d'acides aminés bien conservés au cours de l'évolution. Ces domaines sont souvent caractéristiques d'une fonction donnée et peuvent être définis par des *motifs* utilisant la syntaxe PROSITE [12] comme dans la famille de ribonucléases [FYW]-x-[LIVM]-H-G-L-W-P. Les crochets désignent le choix entre plusieurs acides aminés et le x un *gap* représentant un acide aminé quelconque. Il est possible d'avoir des gaps de longueur variable comme dans le motif C-x(6,8)-[LFY]-x(5)-[FYW]-x-[RK]-x(8,10)-C-x-C-x(6,9)-C où le premier gap a une longueur comprise entre 6 et 8.
Recherche de motifs. La recherche de motifs sur les bases de données issues des séquençages permet de prévoir une fonction aux nouvelles protéines découvertes afin de guider les expériences biologiques. On souhaite autoriser des erreurs dans la reconnaissance de ces motifs, ce qui demande une forte puissance

de calcul. Enfin, la recherche de motifs s'applique aussi aux données nucléiques sur l'alphabet $\Sigma_n = \{A,C,G,T\}$: l'ADN, après une étape de transcription en ARN messager, est traduit en protéines par les ribosomes au moyen du *code génétique* qui associe à chaque triplet de nucléotides de Σ_n un acide aminé de Σ_{AA}. Ceci conduit à 6 *phases de lecture* différentes (3 dans chaque sens de lecture) qui peuvent être décodées et traitées de manière parallèle.
Dans le cadre du projet R-disk (partie 5.1), une architecture parallèle pour filtrer les données à la volée dès la sortie des disques, on souhaite réaliser un filtrage de données génomiques par des motifs exprimés sous la forme d'automates finis ou d'automates pondérés. Nous visons à mettre en place une plateforme traitant des motifs à plusieurs dizaines de caractères, avec des erreurs de différents types (substitutions, insertions, suppressions) et des temps de réponse inférieurs à la minute pour que les biologistes puissent élaborer leur motifs de manière quasi-interactive.

Après avoir défini notre problème et passé en revue quelques solutions existantes (partie 2), nous présenterons l'encodage linéaire pour la recherche de motifs appartenant aux langages rationnels (partie 3). Nous étendrons cet encodage aux automates pondérés (partie 4), puis nous terminerons par une estimation des performances de cette approche sur l'architecture R-disk (partie 5).

2. Recherche de motifs et solutions existantes

2.1. Le problème étudié

Soit un alphabet fini Σ. Les éléments de Σ sont appelés les caractères. Un mot est une suite finie de caractères $w = w_1 w_2 \dots w_n \in \Sigma^*$. Un langage $\mathcal{L}$ est une partie de Σ^*. Étant donné un mot w et un langage $\mathcal{L}$, le problème de la *recherche de motifs (pattern matching)* [8] consiste à trouver tous les sous-mots v de w qui appartiennent à $\mathcal{L}$. Ce problème pouvant avoir $\mathcal{O}(n^2)$ solutions (comme a^* dans le mot a^n), on se limitera à trouver toutes les positions dans le mot initial terminant les sous-mots reconnus, à savoir déterminer l'ensemble $\mathrm{Pos}(\mathcal{L},w) = \{\, j \in [1;n] \mid \exists i \in [1;j],\ w_i w_{i+1} \dots w_j \in \mathcal{L} \,\}$. Nous traiterons successivement le cas où $\mathcal{L}$ est rationnel (section 3) ou reconnu par un automate pondéré (section 4).

2.2. Solutions algorithmiques

Approche brute ou indexée. Lorsque $\mathcal{L}$ est réduit à un singleton $\{v\}$, la recherche de motifs devient un problème de recherche d'un mot fixé v dans un mot de longueur n. Il est possible d'utiliser une approche *brute* d'énumération de la séquence, ce qui amène toujours à un temps d'au moins $\mathcal{O}(n)$ (avec quelques accélérations comme les algorithmes de Knuth-Morris-Pratt ou de Boyer-Moore), ou une approche *indexée*, qui consiste à effectuer des pré-traitements (souvent longs bien qu'en $\mathcal{O}(n)$) dans le but de simplifier les requêtes ultérieures (idéalement en $\mathcal{O}(1)$). Les arbres à suffixes et les DAWG *(directed acyclic word graphs)* donnent de tels résultats sur de nombreux problèmes. Ces techniques restent valides pour des *dictionnaires* où $\mathcal{L}$ est un ensemble fini de mots.
L'approche indexée, bien qu'algorithmiquement plus satisfaisante, reste souvent limitée à des cas de reconnaissance exacte et s'étend mal à des classes de motifs plus complexes, notamment par la gestion des erreurs. Enfin, les bases de données étant régulièrement mises à jour, les pré-traitements demandés peuvent être assez réguliers et coûteux. L'architecture R-disk dans laquelle s'inscrit cette étude prend ainsi le parti d'une approche brute, sans pré-traitements.
Simulation d'automates. Le langage $\mathcal{L}$ sera ici défini par des automates. Pour simuler un *automate fini* (voir partie 3.1) à r états et t transitions sur un mot de longueur n, on peut commencer par le déterminiser, ce qui risque de produire un nombre d'états exponentiel. D'autres méthodes, directes, utilisent un vecteur de taille r et aboutissent à une complexité en $\mathcal{O}(r^2 n)$ ou $\mathcal{O}(tn)$. Il est enfin possible d'utiliser le parallélisme au niveau du bit pour gagner un facteur constant. Quant aux *automates pondérés* (partie 4.2), une simulation directe est nécessaire car ils ne sont pas tous déterminisables. Mark G. Eramian a proposé en 2002 un algorithme [1] résolvant le problème en temps $\mathcal{O}(tn)$.

2.3. Solutions logicielles

agrep : Parmi les logiciels déjà existants, on peut citer *agrep* qui utilise le parallélisme au niveau du bit pour chercher des motifs à l'intérieur de longs fichiers[9]. Les motifs simples peuvent être très performants (traitement de plus de 100 Mo/s de données sur un Pentium IV 2 GHz). Il est possible de rechercher des expressions régulières avec au plus 4 erreurs, mais cela conduit à des performances très

dégradées (par exemple 9 Mo/s pour des motifs de 20 lettres).
en biologie : Quelques programmes existent comme PATTERNp ou PATTINPROT[1]. PATTINPROT permet par exemple une recherche avec erreurs mais n'autorise que deux gaps de longueur variable. Ces programmes se limitent à des syntaxes de type PROSITE qui sont loin d'utiliser les possibilités des langages rationnels.

3. Encodage linéaire et automates finis

Dans cette partie, le langage $\mathcal{L}$ est rationnel et reconnu par un automate fini (indéterministe) $\mathcal{A}$. Nous montrons la pertinence de l'encodage linéaire en partant des travaux de Sidhu et Prasanna [4]. L'automate fini $\mathcal{A}_1$ ci-contre, reconnaissant le langage $\mathcal{L}_1 = a\,(b|c)^*\,(ca\,|\,bc)$, sera pris pour exemple.

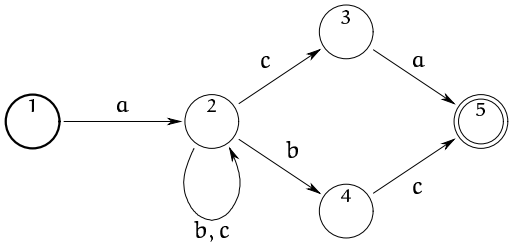

FIG. 1 – *L'automate fini $\mathcal{A}_1$*

3.1. Automates finis

Un *automate fini*[2] *(NFA)* est un quintuplet $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, où Q est un ensemble fini d'états, Σ un alphabet fini, $\delta \subset Q \times \Sigma \times Q$ la fonction de transition, $I \subset Q$ et $F \subset Q$ les ensembles d'états initiaux et finaux. Un mot $w = w_1 w_2 \dots w_n$ est reconnu par $\mathcal{A}$ si et seulement si

$$\bigvee_{\substack{q_0 \in I,\ q_n \in F,\\ q_1,\dots,q_{n-1} \in Q}} ((q_0,w_1,q_1) \in \delta) \wedge ((q_1,w_2,q_2) \in \delta) \wedge \dots \wedge ((q_{n-1},w_n,q_n) \in \delta)$$

Tout langage reconnu par un automate fini est appelé *langage rationnel*. Dans le cas particulier où, pour n'importe quels $q \in Q$ et $\alpha \in \Sigma$ fixés, il existe au plus un état q' tel que $(q,\alpha,q') \in \delta$, l'automate est *déterministe* et on peut noter la fonction de transition sous la forme $\delta(q,\alpha) = q'$.

3.2. Encodages des états

Deux solutions classiques existent pour réaliser matériellement un automate fini selon le mode de représentation de ses $|Q|$ états [3] :

- **l'encodage logarithmique** utilise un vecteur d'états de taille $\log_2 |Q|$ en représentation binaire (naturel, Gray, ou autre). Pour $|Q| = 5$, on considérera par exemple les valeurs $\{000, 001, 011, 101, 111\}$.
- **l'encodage linéaire** utilise un vecteur d'états de taille $|Q|$ où seulement un élément est "à chaud" *(one–hot)* comme dans $\{00001, 00010, 00100, 01000, 10000\}$.

Ces encodages mènent à deux représentations différentes en matériel [5]. Nous verrons ci-dessous que l'encodage linéaire se contente de "plaquer" l'automate sur du matériel. Quant à l'encodage logarithmique, il ne s'applique qu'aux automates déterministes. On commence par initialiser l'état q_0 à l'état initial, puis on calcule à chaque cycle le futur état de l'automate q_i à partir de son état précédent q_{i-1} et du caractère courant w_i. On sait si le mot $w_1 \dots w_j$ a été reconnu par l'automate en vérifiant si l'état q_j appartient à F.

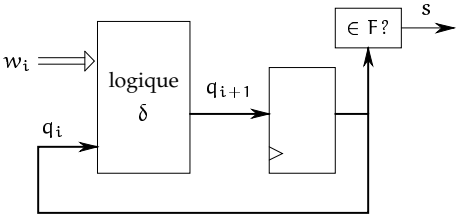

FIG. 2 – *Encodage logarithmique*

L'approche logarithmique convient bien aux modèles de calculs usuels. La taille de l'ensemble dépend surtout de la partie logique et peut être très réduite en logiciel ou en matériel, notamment si la numérotation des états est pertinente. Cette approche est surtout utile en modèle de calcul conventionnel où une instruction est réalisée par cycle.

[1] Ces programmes sont par exemple accessibles à partir de *www.infobiogen.fr/services/deambulum/fr/prog3.html#MOT*
[2] Les automates considérés ici sont sans ε-transitions.

3.3. Encodage linéaire, indéterminisme et pipeline

L'encodage linéaire se réalise directement en matériel. Reetinder Sidhu et Viktor K. Prasanna ont montré en 2001 que celui-ci conduit à une représentation directe des automates finis effectuant une recherche de motifs [4]. Ils construisent un automate à partir d'une expression rationnelle le décrivant. Notre présentation s'effectue sous un angle différent en partant directement d'un automate donné et en le traduisant en matériel. On suppose que l'alphabet à traiter se représente par k bits. Des valeurs typiques seront $k = 8$ pour un texte en ASCII ou $k = 5$ pour l'alphabet Σ_{AA} des acides aminés.

L'ensemble de l'architecture est vue comme un registre à décalage et simule un automate fini. Cet automate peut être indéterministe puisqu'il est possible d'activer plusieurs états en même temps *(multi–hot)*. L'initialisation de l'automate, non représentée, consiste à mettre toutes les bascules à zéro sauf les états initiaux. On traite ensuite à chaque cycle un caractère w_i :

- Chaque transition étiquetée par un caractère α est matérialisée par un *comparateur*. Celui-ci reçoit les k bits du caractère en cours w_i par un bus de données. Il le compare à α pour donner le bit $d_i = (\alpha = w_i)$. Le comparateur peut aussi être une fonction quelconque $\langle k \mapsto 1 \rangle$ (k entrées binaires, une sortie binaire) qui reconnaît n'importe quel sous-ensemble du jeu de caractères. C'est ici le cas de la transition étiquetée par b,c : on aura alors $d_i = (w_i \in \{b,c\})$.
- La transition ne renvoie le résultat de la comparaison que dans le cas où elle est active, c'est à dire $s_i = e_i \wedge d_i$.
- Chaque état est matérialisé par une *bascule* qui fait un OU de l'ensemble des sorties des q transitions qui pointent vers elle $s_i^1, s_i^2, \ldots s_i^q$. Ce résultat e'_{i+1} sera, au cycle suivant, l'entrée d'autres transitions.

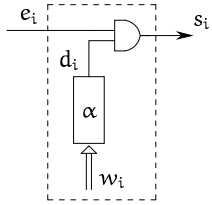

FIG. 3 – *Encodage linéaire*

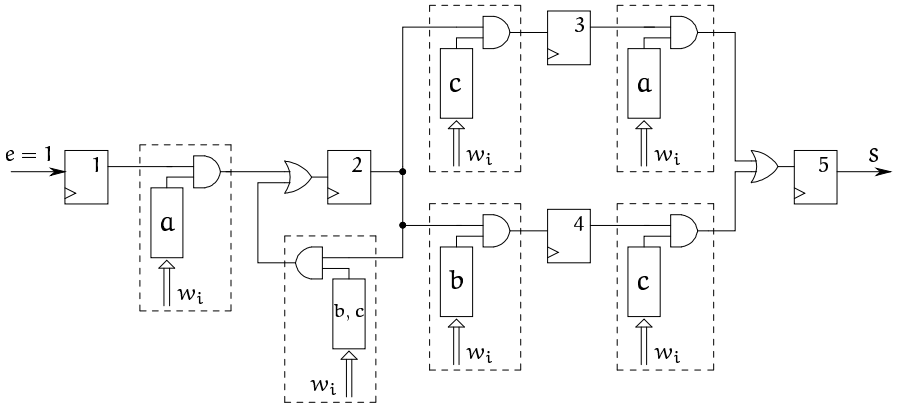

FIG. 4 – *Encodage linéaire de l'automate fini* $\mathcal{A}_1$

Le maintien de l'état d'entrée à $e = 1$ permet enfin de chaîner l'ensemble en recherchant à partir de n'importe quelle position i du mot w un sous-mot $w_i \ldots w_j$ appartenant à $\mathcal{L}$. La sortie s de l'automate représente l'activité des états finaux : un 1 au cycle j signifie qu'un mot $w_i \ldots w_j$ à été reconnu et donc que j appartient à $\mathrm{Pos}(\mathcal{L},w)$. Le problème de la recherche de motifs est ainsi résolu en acceptant un caractère par cycle.

La surface totale de l'automate est en $\mathcal{O}(t)$, où $t = |\delta|$ est le nombre de transitions. Il est ainsi souvent affirmé que l'encodage linéaire ne reste valide que pour les automates où le nombre d'états est limité. Cependant, Dunoyer, Pétrot, et Jacomme [3] ont montré que, pour certains automates de taille conséquente, l'encodage linéaire reste meilleur en consommation *et en surface* par rapport à des encodages logarithmiques.

4. Encodage linéaire et automates pondérés

Il est possible de faire circuler entre les cellules d'un automate avec encodage linéaire des données plus riches que l'activation en $0-1$ comme par exemple un *poids*. Le langage $\mathcal{L}$ sera celui reconnu par un automate pondéré.

4.1. Motivation biologique

Décompte des erreurs simples. Supposons que l'on cherche une suite d'acides aminés avec "au plus $\mathcal{E}$ erreurs". Une solution est de faire circuler entre les cellules de l'automate le nombre d'erreurs en cours. Par exemple, dans le cas où une transition de l'automate attend l'acide aspartique (D), on peut permettre de passer sur cette transition avec un autre acide aminé en appliquant une pénalité de 1. Le décompte total des pénalités à l'état final est comparé à $\mathcal{E}$ pour savoir si le motif a été reconnu ou non.

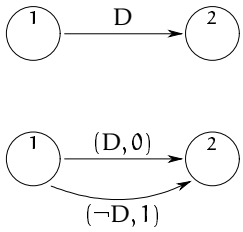

FIG. 5 – *Erreur simple*

Substitutions. On peut étendre le modèle d'erreur simple en attribuant un score quelconque à chaque transition. Les biologistes utilisent fréquemment des matrices de score comme BLOSUM [13] pour rechercher des similarités entre des séquences. Ces scores sont des probabilités de substitution, passées au logarithme et normalisées, qui traduisent les similarités physico-chimiques : l'acide aspartique (D) et l'isoleucine (I) étant des acides aminés très différents, un score négatif -3 est associé à la substitution de l'un par l'autre. Inversement, l'acide aspartique (D) et l'acide glutamique (E) sont tous deux hydrophiles et chargés négativement; ils se substituent avec un score positif $+2$. La transition initiale est ainsi remplacée par un *fuseau* de 20 transitions (nous en avons ici représenté quatre). Là encore, le score total sera comparé à un seuil fixé.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	**9**																				**C**
S	-1	**4**																			**S**
T	-1	1	**5**																		**T**
P	-3	-1	-1	**7**																	**P**
A	0	1	0	-1	**4**																**A**
G	-3	0	-2	-2	0	**6**															**G**
N	-3	1	0	-2	-2	0	**6**														**N**
D	-3	0	-1	-1	-2	-1	1	**6**													**D**
E	-4	0	-1	-1	-1	-2	0	2	**5**												**E**
Q	-3	0	-1	-1	-1	-2	0	0	2	**5**											**Q**
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	**8**										**H**
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**									**R**
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	**5**								**K**
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**							**M**
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**						**I**
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**					**L**
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**				**V**
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**			**F**
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**		**Y**
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	**11**	**W**
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**	

FIG. 6 – *Matrice BLOSUM62*

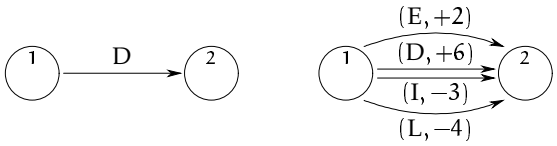

FIG. 7 – *Score de substitution*

4.2. Automates pondérés

La prise en compte des substitutions se fait dans le modèle des automates pondérés. Ceux-ci affectent des poids aux transitions et décrivent des classes de langages strictement supérieures aux langages rationnels.

On considère un semi-anneau $(\mathbb{K}, \oplus, \otimes)$. Un automate pondéré [1, 2] est un quintuplet $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, où Q est un ensemble fini d'états, Σ un alphabet fini, $\delta : Q \times \Sigma \times Q \mapsto \mathbb{K}$ la fonction de transition, $I \subset Q$ et $F \subset Q$ les ensembles d'états initiaux et finaux[3]. La transition $\delta(q_1, \alpha, q_2) = x$ signifie ainsi qu'un poids x est associé à la transition $q_1 \mapsto q_2$[4]. Tout mot $w = w_1 w_2 \dots w_n$ est reconnu par $\mathcal{A}$ avec un certain poids $P(w)$ défini par

$$P(w) = \bigoplus_{\substack{q_0 \in I,\ q_n \in F,\\ q_1, \dots, q_{n-1} \in Q}} \delta(q_0, w_1, q_1) \otimes \delta(q_1, w_2, q_2) \otimes \dots \otimes \delta(q_{n-1}, w_n, q_n)$$

[3] Plus généralement, il est possible de définir des fonctions de distribution initiale et de distribution finale de la forme $\beta : Q \mapsto \mathbb{K}$.

[4] Pour q_1 et α fixés, plusieurs états q_2 peuvent vérifier $\delta(q_1, \alpha, q_2) \neq \bar{0}$: l'automate est indéterministe.

On définit un ensemble de reconnaissance $\mathbb{J} \subset \mathbb{K}$ et on convient que le mot w sera reconnu par $\mathcal{A}$ lorsque $P(w) \in \mathbb{J}$. Les automates finis ne sont qu'un cas particulier d'automates pondérés sur le semi-anneau booléen $(\{0,1\}, \vee, \wedge)$ avec l'ensemble de reconnaissance $\mathbb{J} = \{Vrai\}$. Les autres semi-anneaux couramment utilisés sont $(\mathbb{R}^+, +, \times)$ (calcul des probabilités), $(\mathbb{R} \cup \{-\infty\}, \oplus_{\log}, +)$ (passage au logarithme) et $(\mathbb{R} \cup \{-\infty\}, \max, +)$ (approximation de Viterbi).
Dans ce dernier cas et avec l'ensemble de reconnaissance $\mathbb{J} = \mathbb{R}^+$, l'automate ci-dessous $\mathcal{A}_2$ reconnait le sous-ensemble des mots de $\mathcal{L}_1$ contenant plus de b que de c. Ce langage $\mathcal{L}_2$ n'est d'ailleurs pas rationnel.

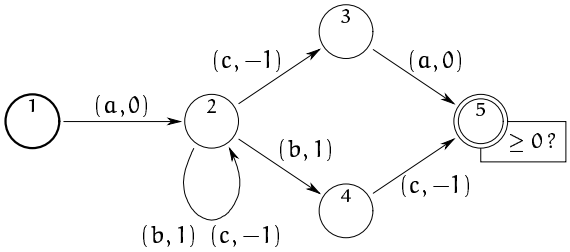

FIG. 8 – *L'automate pondéré $\mathcal{A}_2$*

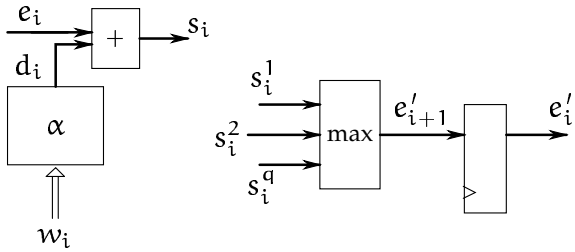

FIG. 9 – *Encodage linéaire pour les automates pondérés*

4.3. Encodage linéaire pour les automates pondérés

Les automates pondérés se réalisent en encodage linéaire similairement à la partie 3.3. L'information qui circule dans l'automate est désormais un poids à p bits pour lequel les bascules et les chemins de données sont dimensionnés[5]. Chaque transition effectue une incrémentation ou une décrémentation du poids en fonction du caractère d'entrée. Le comparateur devient une fonction $\langle k \mapsto p \rangle$ calculant $d_i = f_\alpha(w_i)$, l'opérateur ET un additionneur ($s_i = e_i + d_i$) et l'opérateur OU un maximum.
Comme pour les transitions reconnaissant un ensemble de caractères, il suffit de faire pour l'ensemble du fuseau représenté à la figure 7 une seule fonction $\langle k \mapsto p \rangle$. Remarquons enfin que cette matérialisation est valable pour n'importe quel semi-anneau et opérateurs.

5. Contexte d'utilisation et estimation des performances

5.1. Le projet R-disk

Le projet R-disk [6, 7] est une architecture spécialisée dans le traitement de données en grand volume. Son principe est de fournir une puissance de calcul flexible et reconfigurable à la sortie de dispositifs de stockage :

- les données sont réparties sur plusieurs nœuds reliées par un réseau Ethernet;
- chaque nœud est composé d'une carte comprenant un disque et un processeur reconfigurable offrant des traitements à la volée;
- une station hôte diffuse les requêtes et collecte les résultats.

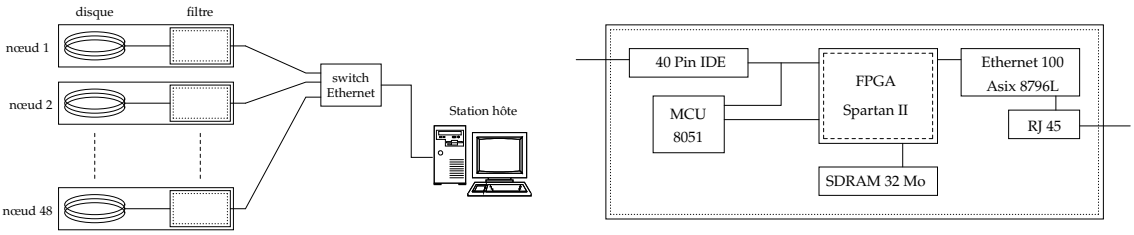

FIG. 10 – *Le projet R-disk*

[5] Ces p bits se répartissent en $p-1$ bits représentant un entier signé en complément à deux, et en 1 bit représentant $-\infty$ pour l'initialisation de l'automate, pour les transitions inexistantes, ainsi que pour les dépassements de capacité.

Les ressources de calcul présentes sur la carte sont principalement utilisées pour filtrer des données selon la requête de l'hôte [6]. Chaque carte lit de manière continue les bases de données en provenance de son disque (avec un débit de 16 Mo/s dans un premier temps, l'interface IDE étant en mode PIO). Une partie de ces données est acceptée par le filtre et renvoyée sur le réseau. Pour ne pas encombrer celui-ci, il est possible de n'envoyer que les identificateurs des données acceptées.

Le processeur reconfigurable utilisé est un composant FPGA à bas coût, le Spartan II de Xilinx. Celui-ci contient 1176 cellules logiques *(cell logic bloc, CLB)*, chacune comprenant en particulier 4 tables de scrutation *(look-up table, LUT)* de 16 bits pouvant réaliser n'importe quelle fonction $\langle 4 \mapsto 1 \rangle$. On suppose que les deux tiers du FPGA sont disponible pour le filtre, soit un total de 784 CLBs faisant un peu plus de 3000 LUTs.

5.2. Réalisation sur FPGA de l'encodage linéaire

L'encodage linéaire se réalise à bas coût dans un circuit FPGA en exploitant massivement son parallélisme, son grand nombre de bascules disponibles ainsi que la propagation locale des données.

LUTs. Le dimensionnement habituel des LUTs convient bien au problème de recherche de motifs protéiques ($k = 5$). Puisque 2 LUTs réalisent n'importe quelle fonction $\langle 5 \mapsto 1 \rangle$ (comparaison d'un acide aminé), les transitions étiquetées par une union de caractères (telles que b,c dans l'automate $\mathcal{A}_1$) ne prennent pas plus de place qu'une simple transition. La juxtaposition de $2p$ LUTs permet, quant à elle, des fonctions $\langle 5 \mapsto p \rangle$ qui matérialisent n'importe quel fuseau avec des poids de substitutions comme celui de la figure 7.

Contrôle. Il n'y a aucun contrôle global coûteux, et il est possible d'utiliser les entrées *Clock Enable* disponibles sur les bascules pour geler tout l'automate en cas de problème d'approvisionnement des entrées ou d'évacuation des sorties.

5.3. Occupation du FPGA et vitesse

Occupation du FPGA. La logique nécessaire pour matérialiser un automate par encodage linéaire se répartit en logique de comparaison (fonction $\langle 5 \mapsto 1 \rangle$ pour les automates finis et fonction $\langle 5 \mapsto p \rangle$ pour les automates pondérés) et en logique de calcul des poids (un opérateur ET/$+$ pour chaque transition ainsi que les éventuels OU/max). Les bascules nécessaires à la mémorisation de chaque état se trouvent déjà à la sortie des LUTs de logique de calcul.

Type d'automate	Logique de comparaison	Logique de calcul des poids	Total par transition	Nombre maximum de transitions pour 3000 LUTs
fini	2 LUTs	$\leq$ 1 LUT	$\leq$ 3 LUTs	≥ 1000
pondéré (p bits)	2p LUTs	$\leq$ 3p LUTs	$\leq$ 5p LUTs	$\geq 600/p$

FIG. 11 – *Occupation du FPGA*

On peut ainsi représenter au moins 75 transitions avec un poids de 8 bits. Il est possible de répartir les transitions disponibles sur plusieurs automates différents traités parallèlement. Un cas particulier peut être la recherche de motifs protéiques sur les banques nucléiques où on peut sextupler l'automate pour pouvoir traiter les 6 phases.

Vitesse. Sidhu et Prasanna [4] ont conclu que, à partir d'un certaine taille, les solutions FPGA sont plus intéressantes que *agrep*. Leurs conclusions restent valables pour les automates pondérés puisque ceux-ci effectuent encore plus d'opérations (additions, maximums) dans le même temps de calcul.

Nous avons mené quelques expériences sur un poste de travail récent (Pentium IV 2 GHz, 728 Mo RAM). Dans le cas le pire, le programme *agrep* accepte des données avec un débit inférieur à 9 Mo/s, tout en étant limité à 4 erreurs de substitution sans score arbitraire. Ces débits doivent être divisés par 6 si l'on souhaite traiter toutes les phases.

De notre côté, nous avons programmé une simulation directe des automates pondérés qui obtient des débits allant de 22 Mo/s pour des automates à 10 états à moins de 2 Mo/s pour des automates à 100 états. Ces automates pondérés restent en plus déterministes, et le traitement des 6 phases baisse là encore le débit.

[6] Les filtres peuvent accepter un sur-ensemble des données voulues (en éliminant par exemple 99% des données négatives) et laisser le traitement final à la station hôte.

En comparaison, la solution que nous proposons peut traiter un débit constant de données, 16 Mo/s pour l'instant, tant que l'automate pondéré tient dans le composant FPGA. Ce débit en sortie des disques correspond à moins d'un acide aminé (5 bits) par cycle aux fréquences de travail du FPGA (pour l'instant 40 MHz). La réalisation des automates pondérés que nous avons décrite accepte un acide aminé par cycle et exploite donc ce débit disque de 16 Mo/s. Des temps de traitement de quelques secondes sont ainsi attendus sur des banques protéiques comme Swiss-Prot [11] (45 Mo de données protéiques).
Une parallélisation massive est enfin obtenue par la présence de plusieurs nœuds R-disk. Le prototype actuellement en fabrication contiendra 48 nœuds. Son débit maximum sera donc de 768 Mo/s, et les bases de données comme EMBL ($42 \cdot 10^9$ bases soit environ 10 Go) pourront être traitées en moins d'une minute.

6. Conclusions et perspectives

L'encodage linéaire permet de réaliser effectivement en matériel les automates finis ainsi que les automates pondérés. Moyennant un espace de $\mathcal{O}(t)$, ces réalisations fournissent une solution au problème de recherche de motifs en acceptant un caractère par cycle. À notre connaissance, il n'existe pas d'autre réalisation matérielle des automates pondérés.
Matérialisés sur un circuit FPGA, les automates pondérés peuvent utiliser sans surcoût des matrices de scores telles que BLOSUM pour autoriser les substitutions. D'autres types d'erreurs (insertions, suppressions) et d'autres classes de langages (répétitions, palindromes) gagneraient à être réalisées en matériel grâce à l'encodage linéaire pour proposer de nouveaux outils performants aux biologistes.

Bibliographie

1. Eramian (M.G.) – Efficient simulation of nondeterministic weighted finite automata, Fourth Workshop on Descriptional Complexity of Formal Systems (DCFS 2002), octobre 2002
2. Mohri (M.) – Edit-Distance of Weighted automatas – Seventh International Conference, CIAA 2002, juillet 2002
3. Dunoyer (J.), Pétrot (F.), Jacomme (L.) – Stratégies de codage des automates pour des applications basse consommation : expérimentation et interprétation, Journées d'étude FTFC, novembre 1997
4. Sidhu (R.), Prasanna (V.) – Fast regular expression matching using FPGAs, IEEE Symposium on Field Programmable Custom Computing Machines (FCCM01), avril 2001
5. Alfke (P.), New (B.) – Implementing state machines in LCA devices, Xilinx XAPP 27, 1994
6. Guyetant (S.), Derrien (S.), Lavenier (D.) – Architecture parallèle reconfigurable pour la génomique, SympA 2002, Hamamet, avril 2002
7. Lavenier (D.), Guyetant (S.), Derrien (S.), Rubini (S.) – A reconfigurable parallel disk system for filtering genomic banks, ESRA 2003, 2003
8. Crochemore (M.), Hancart (C.) – Automata for Matching Patterns – Handbook Formal Languages, Springer, 1997
9. Wu (S.), Manber (U.) – Fast text searching allowing errors, Communications of the ACM, vol 35, no 10, pp. 83 – 91, octobre 1992
10. The EMBL Nucleotide Sequence Database, *www.ebi.ac.uk/embl*
11. Swiss-Prot Protein Knowledge Base, *www.expasy.org/sprot*
12. Bucher (P.), Bairoch (A.) – A generalized profile syntax for biomolecular sequences motifs and its function in automatic sequence interpretation, 2nd International Conference on Intelligent Systems for Molecular Biology (ISMB 94), pp. 53–61, 1994
13. Henikoff (J. G.), Henikoff (S.) – Amino acid substitution matrices from protein blocks, Proc. Natl. Acad. Sci. USA 89, pp. 10915–10919, novembre 1992