

alces, l'orignal (nommé élan en Europe), le plus grand des cervidés.

Jefferson commence sa recherche d'un grand orignal en envoyant à ses amis une étude en 16 points sur les habitudes, la taille et l'histoire naturelle de l'animal. Il fait savoir que si des chasseurs pouvaient lui procurer le squelette d'un orignal géant, il leur serait profondément redevable. Le gouverneur du New Hampshire, John Sullivan, général pendant la guerre de l'Indépendance, lui répond avec enthousiasme et s'attèle à la tâche.

Après son arrivée en France, Jefferson obtient une invitation à rencontrer Buffon. Leur conversation porte sur de nombreux sujets, y compris la théorie de la dégénérescence. Jefferson dit au comte que « le renne européen pourrait passer sous le ventre de notre orignal ». Jefferson sort de la réunion avec l'impression que Buffon céderait si seulement il voyait un énorme orignal.

Enfin, au cours de l'hiver 1786-1787, Jefferson reçoit de bonnes nouvelles : Sullivan s'est procuré les restes d'un orignal auprès d'un certain capitaine Colburn, qui a tué un spécimen de plus de 2,10 mètres dans le Vermont. Il faut 14 jours à une équipe de plusieurs hommes pour amener l'orignal au domicile de Sullivan. Sullivan engage alors un capitaine de navire pour que celui-ci embarque l'orignal lors de son prochain voyage transatlantique.

Tout se déroule selon le plan prévu, mais, pour une raison inconnue, l'orignal reste à quai lorsque le bateau prend la mer. Sullivan envoie la mauvaise nouvelle. Désespéré, Jefferson écrit à un ami que « la caisse, les os, tout a disparu ; ainsi, ce chapitre de l'histoire naturelle restera encore une page blanche ». Il ignore encore que Sullivan a récupéré l'orignal et a engagé un autre navire. Le spécimen arrive à Paris vers le 1^{er} octobre 1787.

Triomphal, Jefferson souhaite apporter l'orignal à Buffon en personne, mais ce dernier, malade, ne reçoit pas de visiteurs. Il envoie donc l'orignal à l'assistant de Buffon. Le comte vit apparemment l'orignal, car Jefferson écrit que cet animal géant avait « convaincu Monsieur Buffon. Il a promis dans son prochain volume de rétablir l'exactitude de ces choses ». Six mois plus tard, cependant, Buffon décédait sans avoir publié aucune révi-

sion de sa théorie. L'*Histoire naturelle*, si influente, était ainsi vouée à promouvoir la théorie d'un Nouveau Monde dégénéré.

L'idée d'une Amérique dégénérée a persisté et évolué pendant une soixantaine d'années avant de laisser place à un anti-américanisme général. Deux factions se sont constituées. En Europe, le philosophe allemand Emmanuel Kant et le poète anglais John Keats acceptaient la dégénérescence dans son ensemble. Keats décrivait l'Amérique comme le seul endroit où « la Grande Nature infaillible semble pour une fois s'être trompée ». Kant, manquant ici de raison pure, écrivait de de Pauw que « même si neuf dixièmes de son matériel ne sont pas étayés ou sont incorrects, cet effort d'intelligence même mérite des éloges et une émulation ».

Le nouveau mythe américain

Sur le continent américain, Jefferson compte parmi ses partisans les écrivains lord Byron, Washington Irving et Henry David Thoreau, et le géographe Jedidiah Morse (le père de l'inventeur du télégraphe électrique, Samuel Morse). Byron appelait l'Amérique « un grand climat ». Irving mettait en pièces la théorie de Buffon dans *The Sketchbook of Geoffrey Crayon*, écrivant qu'il « allait visiter ce pays merveilleux [l'Europe] et voir cette race gigantesque à partir de laquelle j'ai dégénéré ». Thoreau a utilisé son ouvrage *Walking* comme une tribune « pour s'opposer au compte rendu de Buffon sur cette partie du monde et ses productions ». Et Morse discrédita la théorie de la dégénérescence dans les dix premières pages de son manuel de géographie, qui fut lu par la première génération d'écoliers de la nouvelle nation américaine.

Réagissant à cette idée de l'infériorité du Nouveau Monde, ces écrivains américains ont forgé un récit contradictoire, décrivant l'Amérique comme une région magnifique, vaste, riche en ressources et peuplée d'individus robustes. L'identité américaine à ce jour – et les réactions du reste du monde à cette image de soi moderne – pourrait remonter en partie à la destruction par Jefferson, ses pairs et ses successeurs, du mythe d'une dégénérescence biologique en Amérique. ■



2. GEORGES LOUIS LECLERC, comte de Buffon.



3. THOMAS JEFFERSON.

✓ À ÉCOUTER

Jeudi 16 juin 2011, Stéphane Schmitt (REHSEIS), spécialiste de Buffon, évoquera la controverse dans l'émission **La marche des sciences**, sur *France Culture* de 14h à 15h. www.franceculture.com

✓ BIBLIOGRAPHIE

L. A. Dugatkin, *Mr. Jefferson and the Giant Moose : Natural History in Early America*, University of Chicago Press, 2009.

Buffon, *Œuvres*, La Pléiade, Gallimard, 2007.

Buffon, *Histoire naturelle*, 1749-1804, édition en ligne : <http://www.buffon.cnrs.fr/>

LOGIQUE & CALCUL

Le calculateur amnésique

*Un calculateur sans mémoire est-il sérieusement limité ?
Les résultats de l'algorithmique in situ montrent que non :
avec un peu d'astuce, il s'en sortira toujours.*

Jean-Paul DELAHAYE

*J'ai la mémoire qui flanche
Je me souviens plus très bien
De quelle couleur étaient ses yeux
Je crois pas qu'ils étaient bleus
Étaient-ils verts ? Étaient-ils gris ?
Étaient-ils vert-de-gris ?
Ou changeaient-ils tout le temps de couleur
Pour un non, pour un oui ...
Jeanne Moreau, Cyrus Bassiak
et François Rauber*

Si vous avez déjà écrit des programmes informatiques, vous avez nécessairement été confronté au problème suivant : deux nombres sont stockés dans les variables A et B , par exemple $A = 44$ et $B = 13$, et vous voulez échanger le contenu de A et B pour obtenir $A = 13$ et $B = 44$. La méthode classique, que tout le monde réinvente, consiste à utiliser une variable C et à écrire la séquence de trois instructions : $C := A$; $A := B$; $B := C$.

Le signe $:=$ est celui de l'affectation. L'instruction $X := Y$ signifie que nous lisons le contenu de Y et que nous le mettons dans X , ce qui efface le contenu de X sans changer celui de Y . Cette séquence de trois affectations pour l'échange de deux valeurs exige l'utilisation d'une variable C supplémentaire. Or nous pouvons échapper à l'introduction de ce complément de mémoire par la séquence suivante : $A := A + B$; $B := A - B$; $A := A - B$. Le déroulement du calcul donne :

	A	B
Au départ :	44	13
Après $A := A + B$:	57	13
Après $B := A - B$:	57	44
Après $A := A - B$:	13	44

La suite d'instructions est valable avec n'importe quelles valeurs à la place de 44 et 13. Elle fonctionne aussi avec des valeurs booléennes où les signes $+$ et $-$ sont remplacés par xor qui désigne l'opération « ou exclusif » ($0 \text{ xor } 0 = 0$; $0 \text{ xor } 1 = 1$; $1 \text{ xor } 0 = 1$; $1 \text{ xor } 1 = 0$). La suite d'instructions est :

$A := A \text{ xor } B$; $B := A \text{ xor } B$; $A := A \text{ xor } B$, où A et B valent 0 ou 1.

Vérifions que la séquence permute bien les valeurs de A et B en examinant l'effet de chaque opération. La première affectation met $[A \text{ xor } B]$ dans A . La deuxième met donc $[(A \text{ xor } B) \text{ xor } B]$ dans B ; mais, d'après l'associativité du xor , cela vaut $[A \text{ xor } (B \text{ xor } B)]$ c'est-à-dire A , puisque $(B \text{ xor } B) = 0$ et que $A \text{ xor } 0 = A$. La troisième affectation met donc $[A \text{ xor } B]$ dans A ; comme $(A \text{ xor } B) \text{ xor } A = [A \text{ xor } (B \text{ xor } A)]$, cela donne, par commutativité, $[A \text{ xor } (A \text{ xor } B)] = [(A \text{ xor } A) \text{ xor } B] = [0 \text{ xor } B] = B$. Il y a bien échange.

Ce type de calcul est dénommé calcul *in situ* ou *calcul sans mémoire*. L'idée est d'opérer les calculs sans faire appel à des variables de stockage autres que celles qui communiquent les données au programme, que l'on utilise pour calculer et pour placer les résultats une fois les opérations terminées. L'étude de ce calcul *in situ* a pour fonction d'identifier ce que peut calculer un amnésique qui, ne disposant de rien d'autre que l'espace mémoire des données du calcul, le réutilise car il n'a en propre aucune mémoire à sa disposition.

Avant de donner des détails sur les résultats obtenus par les chercheurs qui étudient

cette algorithmique *in situ*, et pour bien en mesurer la difficulté, posons-nous une petite énigme.

Considérons une permutation circulaire de trois données A, B, C , où l'on cherche à passer de $A = 3, B = 50, C = 700$ à $A = 50, B = 700, C = 3$. Nous voulons pour ce faire utiliser un programme *in situ*, c'est-à-dire n'utilisant aucune autre variable que A, B et C , et bien sûr nous voulons qu'il fonctionne pour tout jeu de données.

Permutation circulaire de trois valeurs

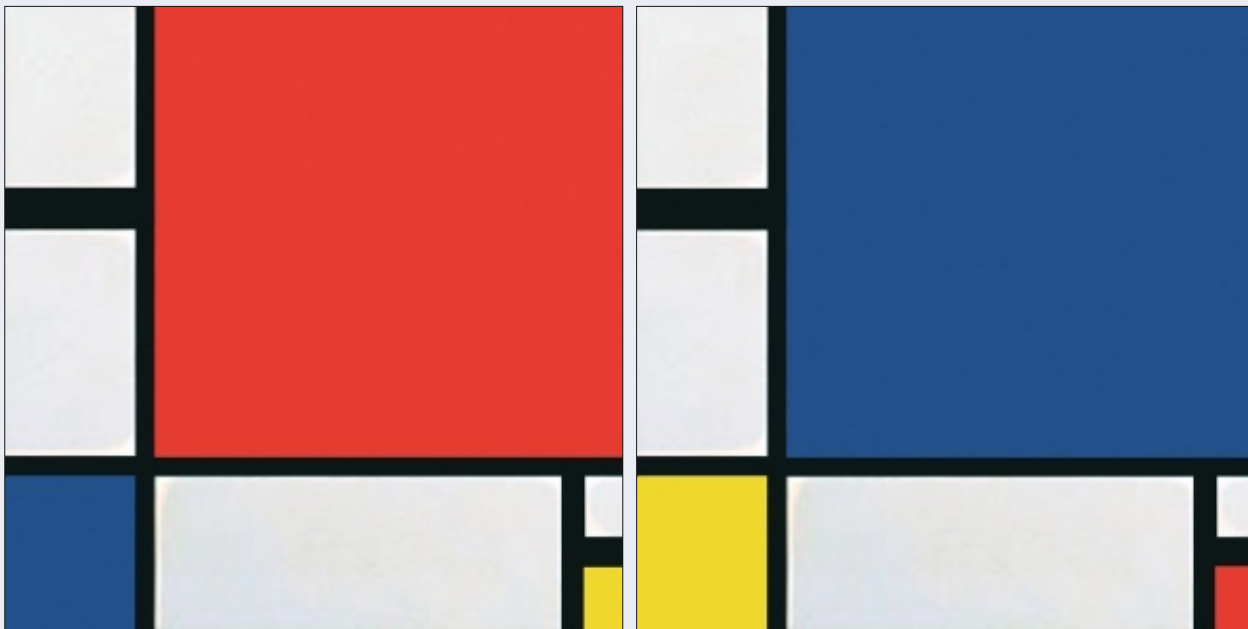
En voici un, avec le détail de l'état de chacune des variables lors du calcul :

	A	B	C
	3	50	700
$A := A + B$	53	50	700
$B := B + C$	53	750	700
$C := A - B + C$	53	750	3
$B := B - A + C$	53	700	3
$A := A - C$	50	700	3

En remplaçant les $+$ et les $-$ par des xor , on obtient, comme précédemment, un programme pour variables booléennes. Une autre méthode aurait consisté à échanger les valeurs de A et B , puis à échanger les valeurs de B et C en utilisant deux fois la méthode vue précédemment. Cela aurait nécessité six affectations alors que notre proposition n'en utilise que cinq. Peut-on faire mieux que cinq affectations ? La solution est donnée dans l'encadré ci-contre.

Expliquons maintenant avec plus de détails ce que les chercheurs dénomment calcul *in situ*. Un ensemble fini S est donné

1. La permutation circulaire



La question posée est : quel est le minimum d'affectations dont on a besoin dans un calcul *in situ* pour échanger circulairement les contenus de trois variables A, B, C , (A, B, C) devenant (B, C, A) ? Dans le tableau de Mondrian et son faux, trois couleurs ont été ainsi permutées. Petit exercice : quel est le vrai, quel est le faux ?

Notons d'abord que trois affectations ne suffiront pas. Il faudrait en effet, dès la première affectation, placer une bonne valeur là où on fait l'affectation, ce qui ferait disparaître la valeur qui y est stockée, dès lors irrécupérable.

En revanche, réaliser la permutation circulaire avec quatre affectations est possible, donc 4 est le nombre minimum d'affectations nécessaires à la permutation circulaire de trois nombres. Voici la solution en quatre affectations :

	A	B	C
Au départ	A	B	C
$A := A + B + C$	A+B+C	B	C
$C := A - B - C$	A+B+C	B	A
$B := A - B - C$	A+B+C	C	A
$A := A - B - C$	B	C	A

Le procédé se généralise pour opérer une permutation circulaire entre n mémoires, c'est-à-dire pour passer de $(A_1 A_2 A_3 \dots A_{n-1} A_n)$ à $(A_2 A_3 \dots A_{n-1} A_n A_1)$.

À la première étape, on place dans A_1 la somme $A_1 + A_2 + A_3 + \dots + A_{n-1} + A_n$, puis, à chaque

étape, toutes les valeurs initiales des variables sont présentes dans $A_2, A_3, \dots, A_{n-1}, A_n$ sauf une, ce qui permet, grâce à A_1 , de la placer au bon endroit.

	A_1	A_2	A_3		A_{n-1}	A_n
Au départ	A_1	A_2	A_3		A_{n-1}	A_n
$A_1 := A_1 + A_2 + \dots + A_n$	$A_1 + A_2 + \dots + A_n$	A_2	A_3		A_{n-1}	A_n
$A_n := A_1 - A_2 - \dots - A_n$	$A_1 + A_2 + \dots + A_n$	A_2	A_3		A_{n-1}	A_1
$A_{n-1} := A_1 - A_2 - \dots - A_n$	$A_1 + A_2 + \dots + A_n$	A_2	A_3		A_n	A_1
.....		...	...		...	...
$A_1 := A_1 - A_2 - \dots - A_n$	A_2	A_3	A_4		A_n	A_1

L'affectation opérée pour A_n place donc bien la valeur initiale de A_1 en A_n , puis l'affectation opérée pour A_{n-1} place donc bien la valeur initiale de A_2 en A_{n-1} , etc. Le calcul est présenté avec des + et des -, et fonctionne avec des nombres entiers, réels ou même complexes.

Le calcul s'adapte aussi avec des nombres calculés modulo k , ou aux variables booléennes (en remplaçant les + et les - par *xor*, ce qui correspond d'ailleurs au + quand on calcule modulo 2). Cette remarque est importante car elle permet de répondre à une objection naturelle, mais essentielle, à la méthode proposée pour opérer une permutation circulaire de n entiers.

Voici l'objection : quand on calcule avec des entiers, la méthode de calcul *in situ* proposée triche, car elle stocke dans A_1 la somme des

valeurs de toutes les variables, ce qui exige un espace mémoire plus grand pour la variable A_1 : s'il y a par exemple 10 variables variant entre 0 et 10, il faut pour stocker A_1 une variable capable de recevoir toutes les valeurs entre 0 et 100.

Voici la réponse à l'objection. Oui, cela est vrai pour la méthode avec des + et des -, c'est-à-dire avec les opérations arithmétiques habituelles entre nombres entiers. Cependant, puisqu'on peut mener un calcul modulo k (où k est un entier fixé), on dispose dans ce cas d'une méthode qui n'utilise pas plus d'espace pour A_1 .

Une autre idée pour répondre à l'objection consiste à écrire les nombres entiers en base 2 et, au lieu de faire des additions ou des soustractions entre deux nombres X et Y , on calcule le nombre dont les chiffres binaires sont ceux obtenus en faisant le *xor* entre les chiffres binaires de X et de Y (exemple $1001 \text{ xor } 0111 = 1110$, autrement dit, $9 \text{ xor } 7 = 14$) et l'on retombe bien sur ses pieds à la fin.

L'espace nécessaire pour stocker A_1 reste encore identique à celui nécessaire pour stocker chaque nombre (p chiffres binaires par exemple si on manipule des nombres de p chiffres binaires). Les procédés décrits (traduits avec des calculs modulo k ou des *xor*) réalisent donc un calcul *in situ* sans avoir à considérer un espace mémoire particulier pour A_1 .

L'AUTEUR



Jean-Paul DELAHAYE est professeur à l'Université de Lille et chercheur au Laboratoire d'informatique fondamentale de Lille (LIFL).

qui peut être un ensemble de nombres, ou de lettres ou de codes numériques chargés de représenter informatiquement des objets réels. On se donne aussi un nombre entier n et une application F de S^n dans S^n , c'est-à-dire une transformation qui à tout n -uplet d'éléments de S en associe un autre. Dans l'exemple de la permutation circulaire, l'application F de S^3 dans S^3 est celle qui, à tout triplet de valeurs (A, B, C) , associe (B, C, A) .

Calcul *in situ* général

Quand on considère des variables booléennes, cela signifie que chaque élément de S est 0 ou 1. En plus du cas booléen, nous envisagerons deux autres cas finis. Si nous calculons avec des nombres entiers écrits en binaire avec k chiffres alors les éléments de S sont $0, 1, 2, \dots, 2^k - 1$; si nous calculons modulo p (nous remplaçons chaque nombre par son reste lors de la division par p), les éléments de S sont $0, 1, \dots, p - 1$.

Dans le cas général, un calcul *in situ* pour une application F est une suite d'affectations analogues à celles que nous avons utilisées

dans les exemples. Ces affectations sont envisagées une à une et chacune attribue une valeur nouvelle à l'une des variables dont l'ancienne valeur est donc perdue... c'est tout le problème ! La valeur nouvelle ne dépend que des valeurs disponibles à cet instant du calcul et nous nous imposons, dans le calcul *in situ*, de n'utiliser aucune variable supplémentaire.

Pour la permutation circulaire de n variables proposée dans l'encadré 1, nous considérons l'opération :

$$F: (A_1, A_2, \dots, A_n) \rightarrow (A_2, \dots, A_n, A_1).$$

La méthode proposée montre que $n + 1$ affectations suffisent pour réaliser cette permutation circulaire, sans que nous n'utilisions aucune autre variable que celles données, et sans que nous ayons à stocker dans les variables des nombres plus grands que $2^k - 1$ dans le cas des nombres en binaire, ou que $p - 1$ dans le cas du calcul modulo p . L'encadré 2 explique que lorsqu'on se donne des variables où l'on peut mémoriser des entiers sans limitation de taille (ce qui est irréaliste en informatique), alors toute affectation F de S^n dans S^n se calcule *in situ* en au plus $n + 1$ affectations.

Cette valeur de $n + 1$, valable pour les permutations circulaires de n variables et pour les transformations F quelconques quand nous utilisons des variables non bornées, n'est pas générale. Pour la transformation $F(A_1, A_2, A_3, A_4) = (A_2, A_1, A_4, A_3)$, on ne sait pas faire mieux que six affectations pour un calcul *in situ* avec des variables bornées.

Les programmes *in situ*, surtout s'ils sont courts, devraient aider à concevoir des puces électroniques moins gourmandes en mémoire, ce qui augmentera la rapidité du calcul. En effet, calculer «localement» est souvent plus efficace pour un processeur que de faire des requêtes aux mémoires externes qu'on doit associer à l'utilisation de variables auxiliaires.

Cette économie de mémoire entraîne un allongement de la taille des programmes. C'est là un nouvel exemple d'une règle générale rencontrée en maintes occasions en informatique : pour économiser de la mémoire, il faut parfois complexifier le calcul. Les algorithmes de compression

2. Calcul *in situ* dans le cas infini

Au départ, nous avons un n -uplet : $(A_1, A_2, \dots, A_n)$. Les variables $A_1, A_2, \dots, A_n$, sont choisies dans un ensemble S non borné, par exemple l'ensemble des entiers.

Une application F attribue de nouvelles valeurs aux variables $A_1, A_2, \dots, A_n$ qui deviennent $B_1, B_2, \dots, B_n$.

Utilisons des variables où nous pouvons mémoriser des entiers sans limitation de taille et voyons comment il a été démontré, dans ce cas particulier, que toute application F peut se calculer *in situ* en au plus $n + 1$ affectations. Pour que le calcul soit *in situ*, nous devons tenir compte du fait que lorsque nous mettons une valeur dans A_i , celle-ci écrasera l'ancienne valeur dès lors indisponible. Pas question donc d'écrire un programme du type :

$$A_1 := E_1(A_1, A_2, \dots, A_n), A_2 := E_2(A_1, A_2, \dots, A_n), \dots, A_n := E_n(A_1, A_2, \dots, A_n).$$

L'idée du calcul est de remplacer A_1 par $A' = 2^{A_1} 3^{A_2} \dots p_n^{A_n}$, où la suite p_i désigne la suite des nombres premiers $(2, 3, 5, 7, \dots)$. Cette étape réunit dans A_1 toute l'information contenue dans les variables $A_1, A_2, \dots, A_n$. Pour récupérer A_i à partir de A' , on calculera le nombre de fois que A' est divisible par p_i . Le fait de placer B_2 dans la variable A_2 ne fait pas perdre l'information A_2 , car celle-ci reste présente, mêlée aux autres valeurs dans A' . On peut alors calculer tous les nouveaux A_i en récupérant toutes les valeurs des anciens A_i dans A' , puis en utilisant l'application adéquate. À l'étape $n + 1$, on calcule A_1 en fonction de A' .

Comme nous avons regroupé et codé (avec la décomposition en facteurs premiers) toutes les valeurs de $A_1, A_2, \dots, A_n$ dans A' , nous n'avons pas besoin de mémoire supplémentaire et nous gardons à portée de main durant le calcul toutes les valeurs initiales des variables mises «au chaud». Bien sûr, cette astuce ne fonctionne pas en informatique où chaque variable ne peut prendre qu'un nombre fini de valeurs.

de données fonctionnent aussi selon cette maxime : en calculant les versions compressées d'images, de musiques ou de vidéos que nous voulons conserver, nous économisons de l'espace. Aujourd'hui, mener des calculs est devenu plus facile et l'algorithmique *in situ* est appelée à se développer et à intervenir au plus profond des puces électroniques. Un brevet a été déposé dans cette optique par les chercheurs du domaine.

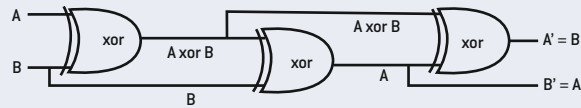
Serge Burckel, aujourd'hui enseignant à l'Université de La Réunion, est l'initiateur des travaux de recherche sur l'algorithmique *in situ* et l'un des principaux auteurs des découvertes sur le sujet. Il pense que le principe du calcul *in situ* a également un sens physique.

L'information est conservée

Pour S. Burckel, l'importance des calculs *in situ* provient de ce que tout système physique isolé qui se modifie mène une sorte de calcul *in situ*. Le fait d'être isolé impose qu'aucune information n'est copiée à l'extérieur du système durant sa transformation. C'est donc que les variables internes qui le décrivent gèrent l'évolution du système d'étape en étape sans que rien d'autre ne soit utilisé. Les lois de la physique ont probablement des formes particulières qui autorisent cela et même qui facilitent l'obtention de ces calculs *in situ*. Comprendre ce qui est faisable par des algorithmes sans mémoire est donc important non seulement pour les applications qu'on devrait en tirer dans la conception des programmes et des puces électroniques, mais aussi pour saisir les principes généraux de la physique.

S. Burckel explique que lorsqu'il a commencé à étudier cette algorithmique de l'économie de mémoire, il a cherché à identifier les transformations d'une suite de n variables booléennes calculables par un programme *in situ*. Prenant des transformations de ce type au hasard, il a découvert, à sa grande surprise, que chacune d'elles se laissait programmer *in situ*. Ce succès sur des exemples était certainement la conséquence d'un

3. Calculer sans croiser de fils



Le croisement de deux fils où circulent des informations est impossible si l'on interdit à un fil de passer sur l'autre (si deux fils se touchent, les informations vont se mélanger). L'utilisation de trois portes logiques *xor* permet l'équivalent d'un croisement de fils. On exploite la permutation *in situ* en trois affectations :

$A := A \text{ xor } B, B := [(A \text{ xor } B) \text{ xor } B] = A; A := [(A \text{ xor } B) \text{ xor } A] = B.$

Plus généralement, si des informations arrivent sur n fils dans un certain ordre et doivent repartir dans un autre (c'est-à-dire une permutation quelconque de n fils, ce que l'on désigne parfois sous le nom de tresse) sans aucun croisement de fils, la programmation *in situ* de la permutation de variables donnera une solution utilisant au plus $3n/2$ portes logiques *xor*.

théorème général qu'il a fini par obtenir : toute transformation de l'ensemble $\{0, 1\}^n$ (les n -uplets de 0 et de 1) dans lui-même se programme *in situ*.

Une longue quête a alors commencé pour, selon les différents types de transformations, élaborer les meilleures listes d'affectations possibles constituant les programmes *in situ*. Nous allons présenter quelques-uns des résultats obtenus.

Les meilleurs résultats connus

Après les permutations circulaires de variables, les transformations les plus simples sont les permutations quelconques de variables, par exemple :

$(A, B, C, D, E) \rightarrow (B, E, D, C, A)$. Un peu d'attention montre que la transformation se décompose en 2 cycles : $A \rightarrow B \rightarrow E \rightarrow A$ et $C \rightarrow D \rightarrow C$.

Autrement dit, on est ramené à deux permutations circulaires, l'une portant sur trois éléments, l'autre sur deux. Puisque chacune se décompose en suites d'affectations (respectivement quatre et trois affectations), ce sera le cas de la transformation qui réunit les deux cycles. Elle pourra donc être obtenue *in situ* en sept affectations.

Nous savons depuis des siècles que toute permutation se décompose en cycles. La méthode de décomposition proposée pour notre exemple se généralise donc sans difficulté : toute permutation possède un programme *in situ*.

✓ BIBLIOGRAPHIE

S. Burckel, E. Gioan et E. Thomé, **Mapping computation with no memory**, dans *Unconventional Computation, 8th International Conference Proceedings*, Lecture Notes in Computer Science n° 5715, pp. 85-97, Springer, 2009.

S. Burckel et E. Gioan, **In situ design of register operations**, *Proceedings of ISVLSI 2008*, IEEE Computer Society Annual Symposium on Very-Large-Scale Integration, 2008.

S. Burckel et M. Morillon, **Quadratic sequential computations**, *Theory of Computing Systems*, vol. 37(4), pp. 519-525, 2004.

S. Burckel et M. Morillon, **Sequential computation of linear boolean mappings**, *Theoretical Computer Science*, vol. 314, pp. 287-292, 2004.

S. Burckel, **Closed iterative calculus**, *Theoretical Computer Science*, vol. 158, pp. 371-378, 1996.

4. Nouvelle décomposition de matrices pour le calcul *in situ*

La décomposition d'une application linéaire F en série d'affectations pour en tirer un programme *in situ* est toujours possible en $2n-1$ étapes, ce qui est très peu.

La lecture de ces $2n-1$ affectations (toutes linéaires aussi) s'interprète comme la décomposition de la matrice de départ en un produit de matrices simples : chacune est obtenue en modifiant une unique ligne à la matrice identité. C'est là un nouveau type de décomposition d'une matrice en produit de matrices simples.

Donnons un exemple avec un calcul modulo 2.

Nous voulons aboutir à l'affectation finale :

$$F(A, B, C, D) = (A + D, A + B + C, A + C + D, A + C).$$

Comme toute application linéaire, on peut l'écrire sous la forme du produit d'une matrice et d'un vecteur colonne.

$$F = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} A+D \\ A+B+C \\ A+C+D \\ A+C \end{pmatrix}$$

L'application est définie par le vecteur colonne indiqué. Toutefois, le calcul ne peut se faire *in situ* d'un seul coup car, dès que l'on a remplacé A par $A + D$, il faut réaménager le calcul en fonction de cette nouvelle valeur de A , et ainsi de suite pour toutes les variables. Cela peut se faire en utili-

$$\begin{aligned} J &= \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} A+D \\ B \\ C \\ D \end{pmatrix} \\ K &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A+D \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} A+D \\ A+B+C+2D \\ C \\ D \end{pmatrix} = \begin{pmatrix} A+D \\ A+B+C \\ C \\ D \end{pmatrix} \\ L &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A+D \\ A+B+C \\ C \\ D \end{pmatrix} = \begin{pmatrix} A+D \\ A+B+C \\ A+C+D \\ D \end{pmatrix} \\ M &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} A+D \\ A+B+C \\ A+C+D \\ D \end{pmatrix} = \begin{pmatrix} A+D \\ A+B+C \\ A+C+D \\ A+C+2D \end{pmatrix} = \begin{pmatrix} A+D \\ A+B+C \\ A+C+D \\ A+C \end{pmatrix} \end{aligned}$$

sant des pseudomatrices unités où une seule ligne est transformée. Ainsi, la matrice F de la transformation est décomposée en un produit des matrices J, K, L et M , explicitées ci-dessus. On obtient avec ces matrices :

$$F(A, B, C, D) = MLKJ(A, B, C, D) = (A + D, A + B + C, A + C + D, A + C).$$

Puisqu'il faut $n + 1$ affectations pour une permutation circulaire entre n variables, il faut dans le pire cas (quand tous les cycles sont des cycles d'ordre 2) $3n/2$ affectations si n est pair, et $(3n - 1)/2$ affectations si n est impair. Dans le cas d'une permutation se décomposant en k cycles non réduits à un point, $c_1, c_2, \dots, c_k$ de longueurs $|c_1|, |c_2|, \dots, |c_k|$, le programme *in situ*, résultat de l'idée indiquée au-dessus, est composé de $k + |c_1| + |c_2| + \dots + |c_k|$ affectations. Ce nombre est le meilleur qu'on puisse obtenir. La décomposition d'une permutation quelconque de variables en un programme *in situ* de $3n/2$ opérations au plus permet d'échanger de l'information entre n fils qui ne peuvent pas se croiser (car placés sur un même plan), en utilisant $3n/2$ portes logiques *xor* au plus (voir l'encadré 3).

D'autres résultats ont été obtenus pour les applications linéaires. Tout d'abord sur des variables booléennes et plus généralement modulo p avec p premier (S. Burckel et Marianne Morillon) et par la suite sur des entiers modulo k , pour tout entier k fixé (Emmanuel Thomé). Un exemple est donné dans l'encadré ci-dessus.

Les résultats obtenus indiquent que toute transformation linéaire peut s'obtenir par un programme *in situ* comportant au plus $2n - 1$ affectations, chacune correspondant à une opération elle-même linéaire (comme dans l'exemple). De plus, on peut imposer l'ordre dans lequel se feront les affectations : 1, 2, ..., n , $n - 1$, ..., 2, 1 (affectation de la première variable, de la deuxième, etc.)

Trouver le simple est compliqué

Toute transformation n'est pas linéaire, aussi fallait-il étudier le cas général. Dans un premier temps, il a été établi que toutes les transformations bijectives (linéaires ou non) F de S^n dans S^n possédaient des programmes *in situ* et que $2n - 1$ affectations suffisent. S. Burckel et Emeric Gioan ont ensuite montré que $5n - 4$ affectations suffisent toujours, quelle que soit la complexité de F et cela même quand F n'est pas bijective. Lorsque le nombre d'éléments de S est une puissance de 2, on peut même descendre jusqu'à $4n - 3$.

Ces résultats récents améliorent un résultat précédent obtenu par S. Burckel et M. Morillon qui indiquait que n^2 affecta-

tions suffisaient dans le cas booléen. Le gain de n^2 à $4n - 3$ est considérable. Malheureusement, le calcul des décompositions est parfois très difficile et lorsque le nombre de variables est grand, on risque de se heurter à une impossibilité pratique. Autrement dit : la complexité des algorithmes pour calculer les décompositions les plus simples est grande. On a là, une fois encore, une illustration de la maxime « Trouver le simple est compliqué ».

Ces résultats ne donnent pas nécessairement les décompositions optimales, qui sont encore plus difficiles à trouver. Les explorations exhaustives sont en théorie envisageables, puisque nous sommes dans un cas fini. Elles garantissent de trouver les programmes *in situ* optimaux, mais elles sont si coûteuses en calcul qu'on ne peut les envisager que si n est vraiment petit (inférieur à 10). De nombreux résultats sont donc encore attendus dans ce domaine à la fois combinatoire, algébrique et algorithmique. Il est à parier qu'à mesure des perfectionnements apportés aux méthodes pratiques pour calculer les décompositions en programmes *in situ*, leur intérêt concret apparaîtra et leur utilisation se généralisera.

Rendez-vous sur www.pourlascience.fr

- Retrouvez gratuitement les actualités scientifiques quotidiennes.
- Recevez nos lettres d'information et réagissez aux articles en vous inscrivant sur le site.
- Recherchez et commandez les archives parues depuis 2004 de *Pour la Science* et des *Dossiers Pour la Science*.
- Accédez aux numéros numériques compris dans votre abonnement en créant votre compte.



À très bientôt sur www.pourlascience.fr !

POUR LA
SCIENCE.fr

Le site de référence de l'actualité scientifique

 ART & SCIENCE

L'erreur de Léonard de Vinci

Un polyèdre à 26 faces, le rhombicuboctaèdre, a défié Léonard de Vinci et un autre peintre de la fin du XV^e siècle, qui ont voulu le représenter. Les deux se sont trompés.

Loïc MANGIN

En 1496, le moine et mathématicien italien Luca Pacioli (1445-1517) arrive à Milan, invité par la puissance locale, le duc Ludovico Sforza (1452-1508). Là, il rencontre Léonard de Vinci (1452-1519) à qui il enseigne les mathématiques. Une solide amitié lie les deux hommes, au point qu'ils partiront ensemble vers Mantoue, en 1499, à la chute de leur protecteur, destitué par les troupes du roi de France Louis XII.

À Milan, Pacioli écrit son grand œuvre *De divina proportione*, publié à Venise en 1509 et dont on connaît trois exemplaires : le premier, dédié à Ludovico, est conservé à Genève, le deuxième est à Milan, le troisième a disparu. Les planches gravées sont de Léonard de Vinci. Dans la première partie de l'ouvrage, *Compendio de divina proportione*, traitant du nombre d'or, on trouve la représentation de 60 types de polyèdres. Le mathématicien et artiste néerlandais Rinus Roelofs s'est récemment intéressé à l'un d'eux et y a découvert une erreur, que le mathématicien belge Dirk Huylebrouck a contribué à rendre publique.

Le polyèdre en question est le rhombicuboctaèdre (voir la figure a, un dessin de Léonard de Vinci qui apparaît dans *De divina proportione*). Il s'agit d'un solide d'Archimède – les faces ne sont pas nécessairement identiques – constitué de 8 faces triangulaires et de 18 faces carrées. En chacun de ses 24 sommets se rencontrent un triangle et trois carrés, et un triangle est toujours entouré de carrés. En outre, le polyèdre est doté d'une symétrie octaédrique, comme le cube. On le nomme *rhombicuboctaèdre*, car 12 de ses faces carrées sont

placées dans les mêmes plans que les 12 faces (en losange) du dodécaèdre *rhombique* qui est le dual du *cuboctaèdre* (huit faces triangulaires et six faces carrées). Rappelons que les sommets du dual d'un polyèdre correspondent aux centres des faces de ce dernier.

À partir de ce polyèdre, on peut en construire un autre, dérivé, en érigeant à partir de chaque face une pyramide qui a donc, outre sa base, trois ou quatre faces. C'est un tel objet qu'a représenté Léonard de Vinci sur une de ses planches de *De divina proportione* (voir la figure d). Un examen attentif révèle

**La pyramide
la plus basse devrait
avoir trois faces visibles
et non quatre.**

l'erreur : la pyramide la plus basse du dessin devrait être à trois faces visibles, et non pas quatre (d, flèche rouge). Un modèle informatique (voir la figure e) et une sculpture de R. Roelofs (voir la figure f, insérée dans la planche originale) confirment le diagnostic. D'autres pyramides (indiquées par des flèches noires) ne semblent pas conformes, mais le doute demeure.

À sa décharge, Léonard de Vinci, qui fut le premier à représenter un tel édifice géométrique, n'avait à sa disposition que les instructions de son ami Pacioli. Il n'avait sans doute pas de modèle, du moins n'en connaît-on aucune trace.

Aucun modèle ? Pourtant, sur le portrait de Luca Pacioli attribué au peintre Jacopo

de' Barbari, le solide suspendu à droite est bien un rhombicuboctaèdre (voir la figure c) ! Regardons de plus près ce tableau, daté de 1495 et dont la paternité est contestée. Pacioli, habillé en franciscain, illustre sur une ardoise marquée « Euclide » un théorème de ce dernier. Sa main gauche est sur les *Éléments* d'Euclide, ouverts au niveau du livre XIII. À droite, sur la table, un dodécaèdre en bois est posé sur un livre, probablement la *Summa de arithmetica, geometria, proportioni et proportionalita* que Pacioli a publiée en 1494, à Venise. L'identité de l'homme, au second plan, est incertaine : certains reconnaissent Guidobaldo 1^{er} de Montefeltro, 3^e duc d'Urbin, d'autres Albrecht Dürer.

Le rhombicuboctaèdre semble être en verre et devait donc être lourd, d'autant plus qu'il est à moitié plein d'eau. Comment un tel récipient pouvait-il être suspendu par un fil si fin ? Des artisans de cette époque étaient-ils capables de fabriquer des joints étanches pour un objet aussi complexe ? Ces questions jettent le doute sur la réalité de ce modèle. Autre indice, plus probant : le rendu du fil sur la toile, et notamment sa réfraction à travers l'eau et le verre, n'est pas réaliste, la version moderne du rhombicuboctaèdre en attestant (voir la figure b) : le fil devrait paraître coupé, à l'inverse de ce que l'on voit sur le tableau.

Ainsi, tant Barbari que Vinci ont dessiné le rhombicuboctaèdre sans modèle. Rendons-leur justice, pour un solide à 26 faces, c'est plutôt réussi ! ■

D. Huylebrouck, *Een fout van Leonardo da Vinci*, EOS, avril 2011.