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.

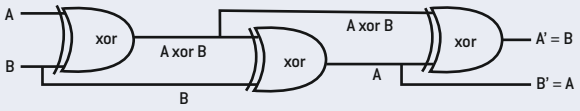
L'information est conservée

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 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.

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*.

Les meilleurs résultats connus

Après les permutations circulaires de variables, les transformations les plus simples sont les permutations quelconques de variables, par exemple :
$(A, B, C, D, \mathcal{E}) \rightarrow (B, \mathcal{E}, D, C, A)$. Un peu d'attention montre que la transformation se décompose en 2 cycles : $A \rightarrow B \rightarrow \mathcal{E} \rightarrow A$ 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.