Chaire d'innovation technologique — Liliane Bettencourt

M. Gérard Berry
Membre de l'Institut (Académie des sciences)
et de l'Académie des technologies,
Professeur associé

Pourquoi et comment le monde devient numérique

1. Introduction

Le cours « pourquoi et comment le monde devient numérique » a été donné dans le cadre de la chaire annuelle d'innovation technologique Liliane Bettencourt. Il a été le tout premier cours d'informatique jamais donné au Collège de France. J'ai choisi de le conduire selon un point de vue non classique, plutôt grand public que technique. En effet, mes collègues informaticiens et moi-même observons depuis longtemps un phénomène étonnant : si tout le monde constate que le numérique transforme le monde de façon très rapide et très profonde, dans ses composantes quotidiennes, scientifiques, industrielles, et culturelles, ses racines restent largement inconnues ou incomprises. Il sort chaque mois des livres sur la « révolution numérique », analysant ses impacts de tous ordres, certains prêchant l'enthousiasme et d'autres la résistance ; si tous parlent des effets, pratiquement aucun ne parle du *contenu* et donc des causes, souvent même en revendiquant cette omission. Or le non-savoir accepté est une cause de majeure de dépendance, et est donc dangereux à terme.

Cette situation troublante doit être opposée à celle de la fin du 19e siècle et du début du 20e, où les grands progrès scientifiques considérables de l'époque étaient popularisés et vulgarisés dans d'excellents livres et revues. Ayant été lecteur assidu de ce genre d'ouvrage, j'ai pensé que la même approche « leçon de choses » s'appliquerait parfaitement au numérique, à condition de trouver le bon niveau de description. J'avais déjà pratiqué des conférences ou des enseignements selon ce point de vue en plusieurs circonstances. J'en citerai deux : l'association « Art Science Pensée » de ma ville de Mouans-Sartoux, et l'école Montessori des Pouces Verts dans la même ville. Ces expériences m'ont permis d'entrevoir ce que les gens ne comprenaient pas, et

donc de commencer à construire un discours leur permettant de comprendre — et d'aimer — le sujet. Mon cours au Collège peut être vu comme une extension de cette approche. J'espère qu'il aura contribué à lever le voile.

1.1. Conduite du cours

Le cours s'est composé de la leçon inaugurale et de huit séances, chacune consacrée à un grand sous-domaine de l'informatique : algorithmes, circuits, langages de programmation, systèmes embarqués, chasses aux bugs, réseaux, images, cryptologie et conclusion. D'autres sujets importants comme les bases de données et systèmes d'informations n'ont pu être traités faute de temps. Chaque séance s'est composée de deux parties : le cours proprement dit, et un séminaire donné par une ou plusieurs personnalités extérieures, chercheurs ou industriels. Cette division m'a paru importante pour que ma relative ignorance de chaque sujet soit compensée par la grande connaissance d'un spécialiste, et pour que mon cours général soit complété par une intervention concernant un aspect plus spécifique mais important en terme de compréhension ou d'impact. Sauf un (cryptologie), tous les cours et séminaires ont été mis en ligne en vidéo. Il faut noter que 28 étudiants de diverses universités se sont inscrits au cycle de cours, validé pour leur formation. Après le cours, un colloque informatique et bio-informatique a été consacré à cinq sujets spécifiques, chacun présenté par un grand spécialiste du domaine.

1.2. Répétitions de la leçon inaugurale

J'ai répété quatre fois la leçon inaugurale en d'autres lieux : à l'Université de Grenoble, dans le cadre de la journée dédiée à Louis Bolliet (16 mai) ; à Sophia-Antipolis, sur invitation de l'école Polytech Nice et de l'INRIA (26 mai) ; au Lycée Marcel Bloch de Strasbourg, devant des classes de première, dans le cadre des journées de l'Académie des sciences en Alsace (28 mai) ; au colloque INIST / CNRS de Nancy (18 juin). Je la répèterai encore en octobre à l'INRIA Rocquencourt (9 octobre) puis à l'INRIA Rennes (24 octobre), sur invitation de ces organismes, et je donnerai un séminaire similaire à la CCI d'Amiens (30 septembre). La leçon inaugurale a également été projetée dans diverses grandes écoles sans ma présence.

1.3. Actions sur l'enseignement

Lors de la leçon inaugurale, j'ai insisté sur le retard considérable pris sur le thème général de l'informatique par l'enseignement pré-universitaire en France, peu compatible avec l'entrée dans le 21e siècle. Cette affirmation a rencontré un écho certain dans diverses instances, et mon action se poursuit de plusieurs façons. J'ai répété ma leçon inaugurale dans un lycée à Strasbourg, et je compte le faire dans d'autres. J'ai donné une conférence « la révolution numérique dans les sciences » lors de la remise des prix des Olympiades de mathématiques (12 juin). J'ai également été chargé d'une mission sur l'enseignement de l'Informatique au lycée par l'Académie des sciences. A ce titre, j'ai rencontré diverses associations de professeurs, et j'ai

donné un séminaire à l'université d'été des professeurs de mathématiques à St Flour en août 2008. J'ai enfin présidé le jury des prix scientifiques collectifs de l'Ecole Polytechnique (30 juin).

1.4. Retombées médiatiques

Le cours a eu un impact médiatique important. Au niveau radiophonique, j'ai participé aux émissions suivantes :

— ***Les matins de France Culture***, Ali Baddou, 17 janvier, le matin de la leçon inaugurale.

— ***Mediapolis***, Michel Field et Olivier Duhamel, Europe nº 1 , 2 mars.

— ***Travaux publics***, Jean Lebrun, France Culture, 12 mars.

— ***Science publique***, Michel Alberganti, France Culture, 4 avril.

— ***Le pudding***, Nicolas Errera et Jean Croc, Radio Nova, 13 et 20 avril.

— ***Place de la toile***, Caroline Broué et Thomas Baumgartner, France Culture, 23 mai.

J'ai aussi participé à des émissions radio plus courtes (France Bleue, RFI,) et à deux émissions de télévision (France 3 et LCI). Le cours a été annoncé et suivi par des articles dans nombreux journaux, revues et sites Internet.

2. Bref résumé de la leçon inaugurale

Après avoir rappelé les très nombreux effets du passage au numérique et l'ignorance généralisée de ses causes, j'ai présenté ma vision sous la forme d'une conjonction de quatre points :

1. L'idée de numériser de façon homogène toutes sortes de données et de phénomènes.

2. Les fantastiques progrès de la machine à informations, faite de circuits et logiciels.

3. Ceux de la science et de la technologie de son utilisation.

4. L'existence d'un espace d'innovation sans frein.

L'idée de numérisation systématique est née dans les années 1950, avec un contribution fondamentale de Shannon qui a défini son sens et ses limites. La numérisation permet de s'affranchir de l'ancestrale dépendance d'une information vis-à-vis de son support : papier pour l'écriture, disque vinyle pour le son, film argentique pour l'image, etc. Une fois numérisées, toutes les informations prennent la forme unique de suites de nombres. On peut alors leur appliquer deux types d'algorithmes combinables avec une totale liberté :

— des algorithmes *génériques*, indépendants du contenu, pour stocker, copier, comprimer (faiblement), encrypter et transporter l'information sans aucune perte, ce qui est impossible avec les représentations analogiques classiques.

— des algorithmes *spécifiques* à un type de données : compression forte et amélioration d'images et de sons, recherche dans les textes, recherche de chemins optimaux dans des graphes, algorithmes géométriques en robotique ou en imagerie médicale, algorithmes de calcul scientifique, la liste est interminable.

La possibilité d'appliquer effectivement ces algorithmes à bas coût repose sur les progrès exponentiels des circuits électroniques et les avancées scientifiques dans leur conception et dans celle des logiciels. Plutôt que le terme « ordinateur », qui évoque trop précisément l'utilisation d'un clavier et d'un écran, j'utilise le terme général de « machine à information » En effet, la plupart des circuits et logiciels sont maintenant enfouis dans des objets de toutes sortes, de façon invisible à l'utilisateur.

Insistons sur le point 4 ci-dessus, l'espace d'innovation sans frein : dans des sciences physiques ou biologiques, il y a souvent loin de l'idée à la réalisation. Tout progrès demande d'abord la compréhension d'un monde préexistant extrêmement complexe. En informatique, la situation est bien différente. L'évolution des machines à informations et de leurs applications ne se heurte pas à la complexité de la nature, puisqu'elle en synthétise en quelque sorte une autre. La distance entre l'idée et l'application est très courte, et la vraie limite à l'expansion des innovations numériques est celle de l'imagination humaine. De nombreux exemples sont fournis par la profusion d'idées nouvelles contribuant à l'expansion du Web : pour l'innovation majeure qu'est le moteur de recherches, il a suffi de quelques mois pour passer de l'idée à la mise en service.

Dans les sciences, le numérique conduit à une révolution généralisée, qui poursuit celle réalisée par l'introduction des mathématiques. En effet, l'informatique étend la notion mathématique de *mise en équations* en la notion bien plus générale de *mise en calculs*, et amplifie les possibilités très limitées du calcul manuel par celles quasi infinies du calcul automatique.

La leçon s'est terminée par l'expression de l'inquiétude sur les insuffisances massives de l'enseignement, et de façon plus générale de l'information scientifique et technique dans ce domaine crucial pour l'avenir.

3. Cours algorithmes

Les algorithmes sont les éléments centraux de l'informatique, et l'algorithmique en est la science. Son but est de construire des algorithmes efficaces en fonction d'un jeu de primitives de base, et d'étudier leur coût ou *complexité*, mesuré suivant divers paramètres : temps de calcul, mémoire ou surface de circuit utilisée, énergie dépensée, etc.

Le coût d'un algorithme dépend fortement de la nature machine d'exécution. L'algorithmique classique s'est concentrée sur les machines de type von Neumann, où une seule instruction est exécutée à chaque instant. L'algorithmique moderne

s'intéresse aussi aux machines parallèles, qu'elles soient synchrones comme les circuits digitaux, ou asynchrones comme les supercalculateurs multiprocesseurs et maintenant les processeurs multicœurs.

Il existe des milliers d'algorithmes pour résoudre des milliers de problèmes de natures extrêmement variées : traitement de textes (recherche, orthographe, etc.), traitement d'images, de sons et de films (amélioration, compression, analyse, recherche d'objets, etc.), traitement d'objets géométriques (imagerie médicale, synthèse d'image, conception assistée par ordinateur, robotique), calcul scientifique et mathématique, gestion de communications et de réseaux, contrôle en temps-réels de transports ou d'usines, etc. Des applications comme l'imagerie médicale utilisent une conjonction d'algorithmes complémentaires de types divers. Cette variété n'exclut pas une théorie générale : la plupart des algorithmes reposent sur un petit nombre de principes centraux illustrés dans le cours : dichotomie (diviser pour régner), procéder récursivement ou itérativement, exploiter l'aléatoire, etc.

3.1. Tri de listes

Notre premier exemple est le tri-fusion sur machine séquentielle, qui illustre à lui seul de nombreux principes. On coupe la liste donnée en deux parties égales (à 1 près), on trie récursivement ces deux listes avec le même algorithme, puis on fusionne les deux listes triées en comparant itérativement leurs têtes. Le couple récursion/dichotomie assure que le coût *maximal* en temps est de $n \log(n)$ pour une liste de taille n, ce qi est théoriquement optimal. Cependant, ce coût reste le même quelque soit la liste de départ, même si elle était déjà triée. D'autres algorithmes permettent d'améliorer le coût en *moyenne*, ce qui est important en pratique.

3.2. Addition entière

Notre second exemple est l'addition de deux entiers. Les algorithmes de l'école sont de coût linéaire dans le nombre de chiffres à cause de la propagation des retenues. Nous montrons comment faire mieux sur un circuit, en additionnant deux nombres p et q par dichotomie/récursion : en supposant qu'ils ont $2n$ chiffres en base 2, on coupe p et q au milieu des poids forts et faibles, posant $p = p' + 2^n p''$ et $q = q' + 2^n q''$. On calcule alors en parallèle les trois sommes $s' = p' + q'$, $s''_0 = p'' + q''$ et $s''_1 = p'' + q'' + 1$. Chaque somme est calculée récursivement avec un additionneur du même modèle La somme s' des poids faibles contient n + 1 bits. On en extrait le nombre t' formé des n premiers bits et le n + 1^{e} bit de poids fort r', qui est la retenue intermédiaire. Le résultat final s est alors défini ainsi : $s = t' + 2^n s''_0$ si $r' = 0$, ou $s = t' + 2^n s''_1$ si $r' = 1$. La sélection entre s''_0 et s''_1 se fait en temps unitaire dans un circuit à l'aide de multiplexeurs parallèles. Grâce à la dichotomie et à la récursion, le coût en temps de l'algorithme est logarithmique, ce qui est beaucoup mieux que le coût usuel linéaire.

L'addition dichotomique illustre deux principes fondamentaux de l'algorithmique des circuits, *l'anticipation* et *l'échange temps-espace*. Pour les poids forts, on anticipe le

calcul de la retenue intermédiaire ; quand celle-ci est connue, les deux poids forts possibles sont déjà disponibles. Le gain de temps est considérable, mais au prix d'une perte en espace, puisqu'on utilise trois sous-additionneurs parallèles au lieu des deux strictement nécessaires, et d'une dépense supplémentaire d'énergie, puisque l'un des deux résultats de poids forts pré-calculés sera simplement jeté aux oubliettes.

3.3. Algorithmes géométriques

Notre troisième exemple concerne le calcul de l'enveloppe convexe d'un ensemble de points du plan. Cet algorithme est fondamental en géométrie algorithmique, par exemple pour calculer l'intérieur d'un objet. Il se fait en plusieurs étapes : calcul du point le plus bas, tri des angles entre ce point et les autres points, puis parcours itératif des points dans l'ordre ainsi établi pour déterminer les segments de l'enveloppe convexe. L'algorithme est illustré par une animation dans la vidéo et les transparents du cours. Son analyse est remarquable sur un point : l'étape chère de l'algorithme est le tri des angles, en *n log2(n)* s'il y a *n* sommets. Les autres étapes sont linéaires, ce qui n'est pas évident *a priori*. L'analyse d'algorithmes recèle beaucoup de surprises, même pour des algorithmes simples !

Les *diagrammes de Voronoï* constituent une structure fondamentale de la géométrie algorithmique. Etant donné un ensemble *E* de points du plan, le diagramme de Voronoï associé découpe le plan en polygones contenant chacun un point *p* de *E*, et tels que chaque point de l'intérieur d'un polygone est plus près de *p* que de tout autre point *p'* de *E*. La structure duale, formées de segments qui joignent les points de *E* perpendiculairement aux arêtes des polygones, est appelée triangulation de Delaunay. Les arêtes des polygones de Voronoï sont les médiatrices des côtés des triangles de Delaunay, eux-mêmes caractérisés par le fait que leur cercle circonscrit ne contient aucun autre point de l'ensemble. Les diagrammes de Voronoï ont de très nombreuses applications. Ils ont été introduits par Descartes pour étudier la répartition des constellations dans le ciel, utilisés par John Snow pour modéliser la propagation des épidémies dans les années 1850, et le sont maintenant à grande échelle pour le calcul de maillages efficaces en calcul numérique, pour la représentation des objets 3D en synthèse d'image, etc.

Nous montré leur construction à l'aide d'un petit logiciel fourni par l'INRIA, également idéal pour illustrer les notions *d'algorithme incrémental* et *d'algorithme itératif*. Ajouter un nouveau point à un diagramme existant se fait de façon incrémentale au voisinage de ce nouveau point, sans perturber le reste. L'efficacité est excellente, ce qu'on voit en cliquant et bougeant la souris à grande vitesse. On peut aussi minimiser l'énergie d'un diagramme, comptée comme l'intégrale des carrés des distances de chaque point du domaine au point caractéristique du polygone auquel il appartient. Cette énergie n'est pas minimale quand le point caractéristique d'un polygone est distinct de son centre de gravité. L'algorithme de Lloyd fournit une méthode itérative de minimisation par déplacement des points initiaux vers le centre de gravité. Il est très beau à voir fonctionner sur la

démonstration. Il se rencontre effectivement dans la nature, par exemple pour expliquer le déplacement des soles cachées sous le sable pour guetter leur proie lorsqu'il faut laisser de la place à une nouvelle arrivante.

3.4. Le problème SAT et les algorithmes NP-complets

Le cours a ensuite présenté le problème SAT de satisfaction Booléenne, dont l'objet est de savoir si une formule Booléenne (écrite avec des variables et les opérations *vrai*, *faux*, *et*, *ou* et *non*) peut être rendue vraie par un jeu de valeurs des variables. Ce problème élémentaire est en fait très difficile et mal compris. Il est central pour la vérification formelle des circuits et programmes, et il fait l'objet d'une véritable compétition industrielle. Il de plus caractéristique de la classe des problèmes *NP-complets*, qui en comprend des centaines d'autres (emploi du temps, horaires de train, etc.). Pour ces problèmes, on ne connaît que des algorithmes exponentiels dans le cas le pire. Savoir s'il existe des algorithmes toujours polynomiaux pour SAT et les problèmes NP-complets est considéré comme un des problèmes les plus difficiles des mathématiques actuelles.

3.5. Machines chimiques et nombres premiers

Les algorithmes parallèles peuvent être très différents des algorithmes séquentiels. Nous avons présenté une variante parallèle du crible d'Eratosthène pour le calcul des nombres premiers. Intuitivement, on met tous les nombres sauf 1 dans un grand chaudron, et on les laisse être agités par le mouvement brownien. Un nombre p peut manger ses multiples quand il les rencontre, et tous les combats peuvent se faire en parallèle. Les nombres premiers sont les seuls à survivre. Ce modèle ultra-simple s'étend en une *machine chimique* générale, introduite par J.P. Banâtre et Le Métayer puis perfectionnée par G. Boudol et l'auteur. Les machines chimiques sont utilisées pour modéliser les réseaux de types Internet, la migration des calculs dans les réseaux, et certains phénomènes biologiques (cf. section 11).

3.6. Séminaire : algorithmes probabilistes sur de grandes masses de données

Le séminaire donné par Philippe Flajolet, directeur de recherches à l'INRIA, a été consacré aux algorithmes probabilistes sur de grandes masses de données. Il a illustré un autre grand principe algorithmique, *l'exploitation positive de l'aléa*, reposant ici sur l'utilisation de *fonctions de hachage pseudo-aléatoires*. Une telle fonction va associer une information de taille fixe (par exemple 32 ou 64 bits) à n'importe quelle information d'entrée (mot, texte, image, son, etc.). Etant pseudo-aléatoire, elle va répartir équitablement les informations d'entrée dans les valeurs hachées, en cassant leurs régularités potentielles. P. Flajolet a montré l'efficacité de ces algorithmes dans un grand nombre de problèmes apparemment dissociés : le comptage approché de mots dans un livre avec très peu de mémoire et sans dictionnaire, le classement d'un grand nombre de documents selon leur similarité, la détection d'intrusions et de virus dans un réseau, etc. Bien que ces algorithmes soient très simples, leur analyse fait appel à des mathématiques particulièrement sophistiquées.

4. Cours circuits

Les circuits électroniques sont le moteur du monde numérique. Depuis les années 1970, ils se sont développés suivant la *loi de Moore*[1] : le nombre de transistors sur une puce double tous les 18 mois. Cette loi exponentielle est due aux progrès de la physique des transistors et à l'amélioration continuelle des techniques de fabrication. Elle a des conséquences étonnantes : un microprocesseur comportait quelques milliers de transistors en 1975, quelque centaines de milliers en 1985, une dizaines de millions en 1995, et un milliard en 2005, tout cela à prix décroissant. Elle se poursuit pour l'instant, et sera probablement freinée plus par des considérations énergétiques et économiques que par des problèmes physiques de miniaturisation.

Mettre autant de transistors sur une puce permet de fabriquer des circuits hétérogènes appelés *systèmes sur puce* (System on Chip ou SoCs en anglais), qui intègrent des processeurs de calcul, des accélérateurs (graphique, cryptage, etc.), des mémoires, des processeurs d'entrées/sorties, des émetteurs récepteurs radio, tous reliés par des réseaux hiérarchiques. Il s'en fait des milliards par an, et le volume ne cesse d'augmenter. Ils se trouvent dans les objets de tous genres : téléphones, voitures, trains, avions, vélos, maisons, prothèses médicales, jouets, etc. On prévoit qu'il y aura de l'ordre de mille circuits par humain d'ici une quinzaine d'années, la plupart reliés en réseau.

4.1. Les principes des circuits digitaux synchrones

Les circuits *digitaux synchrones* fonctionnent sur le mode binaire 0/1 en étant cadencés par une horloge. Ce sont les plus importants et les seuls étudiés ici. Ils sont complétés par les circuits analogiques, utilisés pour la radio et la communication avec le monde physique, et par quelques circuits asynchrones sans horloges.

Un circuit digital synchrone se comporte électriquement comme un réseau acyclique de portes logiques reliées par des fils qui peuvent être portés à deux potentiels stables 0 et 1 Volt. Les portes calculent les fonctions Booléennes *et*, *ou*, *non*, briques de base avec lesquelles on peut coder tout autre calcul. Le circuit comporte aussi des mémoires, commandées toutes en même temps par l'horloge. Le principe est simple et quasiment inchangé depuis les premiers ordinateurs: pendant un cycle d'horloge, l'environnement maintient les entrées à 0 ou 1, et les mémoires maintiennent leurs sorties vers les portes à 0 ou 1. Les signaux se propagent dans le réseau de portes jusqu'à devenir stables aux sorties du circuit et aux entrées des mémoires. Au front montant suivant de l'horloge, l'environnement échantillonne les sorties, et les mémoires basculent : leurs entrées au front deviennent leurs nouvelles sorties pour tout le cycle suivant. La fréquence maximale de l'horloge est déterminée par la nécessité d'avoir stabilisé tous les fils au front montant.

1. Du nom de Gordon Moore, co-fondateur d'Intel.

Les portes sont organisées en blocs architecturaux. Par exemple, un microprocesseur comporte des décodeurs d'instructions, des unités de calcul arithmétique et logique, des unités de gestion mémoire et des entrées/sorties. Il lit un programme en langage machine et des données, et exécute les calculs spécifiés. L'accélération par *pipeline* est un concept architectural fondamental identique à celui développé par Henry Ford pour ses usines de voitures : chaque opération est découpée en phases qui se suivent dans le temps, et toutes les phases d'instructions consécutives s'exécutent en parallèle. L'accélération par *spéculation* repose sur des calculs réalisés en avance de phase mais potentiellement inutiles, comme expliqué pour l'addition à la section 3.2. Les *mémoires caches* permettent de pallier la lenteur des mémoires externes (RAM) en utilisant des tampons locaux. Des exemples animés sont montré dans la vidéo et les transparents. Beaucoup de détails fins doivent être réglés pour un fonctionnement efficace et sûr des pipelines, spéculations et caches. L'architecture des processeurs modernes est vraiment aux limites des capacités humaines, et celle des systèmes sur puce de tous ordres la rejoint en termes de difficultés.

4.2. Electronic Design Automation : la synthèse de circuits

La conception de circuits aussi gigantesques ne se fait évidemment plus à la main, Les étapes ont été successivement automatisées, donnant naissance à une industrie spécifique, l'EDA (*Electronic Design Automation*). Les outils EDA permettent une synthèse continue et automatique, allant des niveaux abstraits du design fonctionnel vers le dessin concret des masques lithographiques qui servent à fabriquer les circuits. On part de langages de haut niveau pour générer des portes logiques, qu'on va dimensionner, placer et router sur le rectangle de silicium, déterminant alors à quelle vitesse peut tourner l'horloge. Il faut en fait une dizaine d'étapes logicielles, dont le bon fonctionnement est aussi vérifié par logiciel. L'outillage est très cher et très gourmand, et sa complexité ne fait qu'augmenter.

Comme le circuit est fabriqué en une fois, le moindre bug de conception peut le rendre entièrement inopérant. Il faut donc être sûr de son bon fonctionnement avant la mise en fabrication. La vérification fonctionnelle se fait soit par simulation, soit avec les méthodes formelles étudiées en 7.2. Elle revient couramment à 70 % du coût total, et est de moins en moins exhaustive. C'est donc un goulot d'étranglement fondamental.

4.3. Séminaire : la fabrication des circuits

La fabrication des circuits a été présentée dans le séminaire donné par Laurent Thénié, de Cadence Systems Design. Il a détaillé le principe lithographique de la fabrication, les longues et nombreuses étapes nécessaires, et les usines et leurs machines. Une génération de circuits est caractérisée par la taille du transistor, actuellement 65 ou 50 nanomètres. Des complications considérables sont induites par le caractère non-linéaire des gravures modernes, qui reposent sur des images bien plus fines que la longueur d'onde de leurs sources lumineuses. Enfin, il faut

tester un à un tous les circuits fabriqués, car un seul problème de fabrication (par exemple, une seule poussière) suffit à rendre un circuit inopérant. Les tests sont faits à partir de longues séquences fournies par les outils EDA.

La fabrication est de fait très dissociée de la conception : les outils EDA définissent exactement ce qui doit être fabriqué et testé, et le fabriquant n'est pas du tout concerné par ce que fait le circuit.Un problème majeur est que le prix de l'usine augmente considérablement avec chaque génération, et atteindra bientôt des dizaines de milliards d'euros. Bien que l'objet qu'elle fabrique soit l'un des plus légers, l'industrie est une des plus lourdes !

4.4. Le futur des circuits

De nombreuses difficultés de tous ordres risquent de ralentir l'évolution exponentielle : sur le plan technique, citons la difficulté de monter encore en fréquence, les limites posées aux circuits par la chaleur dissipée, et les problèmes posés par la gestion d'horloges multiples ; sur le plan humain, citons la complexité de la conception, qui peut demander des milliers de personnes, le coût extrême des investissements, et la nécessité de rentabiliser une fabrication par de très grands volumes.

De nouvelles solutions apparaissent pour contourner ces problèmes. Le FPGA (Field Programmable Gate Array) a été introduit dans les années 1980. C'est un méta-circuit transformable à volonté en circuit précis par téléchargement d'une configuration binaire. Un même circuit physique peut être ainsi rapidement configuré en un très grand nombre de circuits logiques par l'utilisateur, au prix d'un coût plus élevé et d'une densité moindre. D'autres propositions étudient des circuits à grand nombre de processeurs simples programmables par logiciel. Mais bien les programmer reste un grand défi intellectuel. Enfin, il ne sera bientôt plus possible de fabriquer des circuits zéro défaut. On s'oriente vers des formes de redondance permettant de ne pas utiliser des parties de circuits non fonctionnelles.

Pour gagner des ordres de grandeurs en taille et complexité de circuits, l'industrie de l'EDA évolue vers *l'Electronic System Design* ou ESL. Il s'agit de grimper encore des niveaux d'abstraction, par exemple pour synthétiser complètement un circuit à partir de spécifications de type logiciel et prouver mathématiquement son bon fonctionnement[2].

5. Cours langages de programmation

Les langages de programmation sont l'instrument indispensable pour l'écriture des programmes. Leur genèse précède celle des ordinateurs : les logiciens comme Frege, Russel, Turing et Church se sont intéressé au problème de la définition précise du langage mathématique, qui est de même nature. Les premiers langages ont été les langages machines des ordinateurs, bientôt rendu symboliques sous le

2. Ce qui est le sujet principal du travail de l'auteur.

nom d'assembleurs. Un tournant a été apporté dans les années 1950 par FORTRAN (Formula Translator), qui a augmenté le niveau d'abstraction en utilisant directement les expressions mathématiques comme (A + B) *C pour programmer, avec comme primitives l'affectation de type X = (A + B) *C, la mise en séquence d'instructions, les boucles DO, et l'appel de fonctions définies elles-mêmes par des programmes. Il faut alors un programme appelé *compilateur* pour traduire ces programmes de plus haut niveau en langage machine. Ce schéma est toujours valable.

5.1. Le monde Darwinien des langages

FORTRAN sera suivi dans les années 1960-70 par des langages plus riches et développés plus scientifiquement, comme ALGOL et PASCAL, et par le langage C né avec Unix. En parallèle, des langages très différents verront le jour comme LISP, dédié au calcul pour l'intelligence artificielle. Ce langage fonctionnel (sans affectation) et d'exécution interactive sera suivi par SMALLTALK, SCHEME, puis ML, qui a donné naissance au populaire et rigoureux langage Caml développé à l'INRIA, et au puissant langage fonctionnel Haskell. L'augmentation permanente de la taille des programmes a conduit à l'ajout de systèmes de modules ou de classes objet permettant de définir et de maintenir de vraies architectures logicielles (Modula, ADA, C++, Java, Caml, etc.).

Tous les langages précités permettent de programmer n'importe quel algorithme sur une machine monoprocesseur. Avec les réseaux et la commande de processus temps-réel est apparu la nécessité d'écrire des programmes parallèles s'exécutant sur une ou plusieurs machines, d'où la naissance de langages asynchrones comme CSP puis ADA, de langages synchrones comme Esterel et Lustre (cf. 6.3), et l'ajout de techniques de programmation parallèles dans les langages classiques (en général au moyen de *threads*, ou flots d'exécutions coordonnés). D'autres langages sont plus spécifiques. Basic, petit langage interactif des années 60, est devenu Visual Basic, présent dans toutes les applications bureautiques de Microsoft. APL est dédié à l'écriture très compacte d'algorithmes matriciels. FORTH, originellement destiné aux programmes temps-réel, a introduit les codes compacts embarqués (byte-code), repris par Java et Caml. Prolog a introduit la programmation par règles logiques et fait naître la famille des langages à contraintes. Les langages de scripts, inaugurés par le shell d'Unix, sont devenus très populaires pour les manipulations systèmes et le Web (tcl, perl, python, php, etc.).

Si l'on ajoute la présence de plusieurs dialectes pour chaque langage (des centaines pour LISP), on voit que le paysage linguistique est extrêmement complexe. L'évolution est Darwinienne : la survie d'un langage dépend de la création au bon moment d'un groupe grandissant d'utilisateurs écrivant de nouvelles applications : C est né d'Unix, Java a crû avec le Web, etc. Les guerres de religion ont abondé : les gens du fonctionnel ne supportaient pas l'impératif et réciproquement, les amateurs de langages interactifs ne supportaient pas de devoir attendre la fin d'une compilation, etc. Tout ceci s'est bien calmé, car concevoir un nouveau langage et

l'ensemble de son outillage (compilateur, débogueur, etc.) demande un investissement de plus en plus considérable.

5.2. Pourquoi tant de langages ?

Le nombre des langages est une conséquence normale de l'existence de nombreux styles pour exprimer un algorithme ou une classe d'algorithmes. Chaque style exprime un niveau d'abstraction particulier : le style fonctionnel décrit le résultat à obtenir plus que la façon de le calculer, le style impératif fait exactement l'inverse, le style logique définit les algorithmes par des combinaisons de règles individuellement simples, le style objet encapsule les données dans des classes fournissant des méthodes de traitement locales, le style parallèle abandonne l'idée de suivre un programme avec un doigt, le style graphique propose une programmation plus visuelle que textuelle ; la gestion de la mémoire est soit explicite soit implicite, les pointeurs sont autorisés ou non, etc. Par ailleurs, il existe plusieurs styles syntaxiques capables à eux seuls de faire se battre les gens : C est piquant et impératif, LISP est rond et récursif, CAML est mathématique et récursif, etc. Comme en art, chaque style a son intérêt et combiner deux styles n'est jamais simple.

5.3. Syntaxe, types et sémantiques

Tout langage a trois composantes fondamentales : une syntaxe, qui définit les programmes bien écrits, un système de types, qui assure avant d'exécuter les programmes qu'on n'additionnera pas des choux et des carottes, et une sémantique, qui définit le sens des programmes.

L'analyse syntaxique est un problème magnifiquement réglé depuis longtemps. Le typage évolue encore. Il est fondamental pour détecter les erreurs avant l'exécution et donc avant qu'elles n'aient des conséquences. FORTRAN n'avait que des nombres entiers et flottants. LISP n'avait pas de types du tout. Pascal, C, Ada, C^{++}, etc. ont incorporé des systèmes de types de plus en plus riches, permettant de définir des structures de données complexes tout en détectant beaucoup d'erreurs. ML a fait un pas important en introduisant un système de types qui garantit mathématiquement l'absence d'erreur à l'exécution, avec inférence de types pour éviter d'écrire les types à la main, et généricité pour écrire par exemple des files d'attente bien typées fonctionnant sur types paramétriques. La théorie des types est bien établie mathématiquement mais peut encore progresser pratiquement.

La sémantique des programmes définit ce qu'ils doivent faire. Elle est souvent traitée un peu par dessus la jambe : exécutez donc votre programme pour voir ce qu'il fait ! Pourtant, avoir une sémantique claire est indispensable pour avoir des programmes portables d'une machine à une autre, être sûr de ce qu'ils font, et les prouver formellement. La communauté théorique sait maintenant définir complètement la sémantique d'un langage, pourvu qu'il soit conçu pour. Par exemple, des langages comme Caml, Lustre et Esterel sont définis de façon mathématique et ne peuvent donner lieu à des erreurs d'interprétation.

5.4. Séminaire : du langage à l'action

Xavier Leroy, Directeur de Recherches à l'INRIA a présenté la compilation de programmes de haut niveau en code machine. Il en a détaillé les différentes étapes : analyse syntaxique, analyse des types, allocation des données, en mémoire, mécanismes d'exécution par piles, allocation de registres, optimisation, et génération de code. Il a montré qu'optimiser un programme demandait souvent de spéculer pour compenser la lenteur des mémoires, en dépliant les boucles et en réordonnançant le code expansé obtenu pour pré-charger les données avant d'en avoir réellement besoin. Il a enfin montré la possibilité d'écrire des compilateurs mathématiquement prouvés correct, exploit que son groupe a été le premier à réaliser et qui pourra avoir des impacts pratiques considérables à l'avenir.

6. Cours systèmes embarqués

Un système embarqué est un système informatique logiciel et matériel enfoui dans un objet afin de contrôler son activité et sa sécurité, d'offrir des services à ses utilisateurs et de communiquer avec d'autres objets. Les systèmes embarqués sont extrêmement nombreux et variés : téléphones portable, avions, trains, voitures, et même vélos, appareils photos et lecteurs MP3, objets domotique, stimulateurs cardiaques et prothèses intelligentes, etc. Le domaine est en croissance très rapide et va aller en s'unifiant avec celui des réseaux, car les objets seront de plus en plus reliés entre eux, soit par des communications spécifiques, soit tout simplement par Internet.

Bien que leurs principes fondamentaux soient les mêmes que ceux des ordinateurs à clavier et écrans, la façon de concevoir et de mettre au point les systèmes embarqués est très différente. Ils sont le plus souvent soumis à des contraintes d'autonomie, de fiabilité et de sécurité bien supérieures, surtout s'ils sont destinés à fonctionner en environnement hostile : il est clair que le taux de bugs acceptable pour le pilotage d'un avion soit être absolument minimal.

6.1. Parallélisme et déterminisme

Les systèmes embarqués doivent le plus souvent concilier deux caractéristiques fondamentales : le *parallélisme* et le *déterminisme*. Le parallélisme résulte de la présence de nombreux composants devant agir de façon simultanée et coordonnée. Ainsi, dans un système de pilotage, il faut simultanément gérer les capteurs, calculer la trajectoire, corriger la commande en fonction de cette trajectoire, solliciter les actionneurs, etc. Le déterminisme est une contrainte sur le comportement du système, qui exprime que toute exécution à partir de la même séquence d'entrées doit produire le même comportement. Il est clair que la réaction d'une voiture à un coup de frein ou à un mouvement du volant doit être toujours la même dans les mêmes conditions de route. Bien gérer le parallélisme déterministe est fondamental.

Le parallélisme est partagé par bien d'autres applications non embarquées, par exemple par les applications Internet. La situation est différente pour le

déterminisme. S'il est naturel pour les systèmes embarqués, il ne l'est pas pour d'autres applications parallèles. Par exemple, on ne peut évidemment pas demander à un moteur de recherche de répondre toujours de la même façon à une question donnée, puisqu'il doit au contraire se mettre à jour en continu.

6.2. Méthodes classiques de programmation

Les méthodes classiques pour la programmation des systèmes embarqués sont fondées sur une extension directe des méthodes de programmation séquentielle au cas parallèle. Le parallélisme peut être introduit de deux manières :

1. Au moyen d'un système de tâches séquentielles gérées par un ordonnanceur. Celui-ci peut être dynamique, conduisant à du non-déterminisme, ou pré-calculé et statique, alors déterministe. Mais le raisonnement sur un système de tâche et sa vérification sont difficiles.

2. Par introduction directe dans un langage séquentiel : *tasks* en ADA, *threads* en Java, etc. Dans ce cas, les exécutions des différentes actions parallèles se font de façon asynchrone et non-déterministe, ce qui est contraire à la contrainte de déterminisme. Dans certains cas (Spark ADA), des restrictions permettent d'assurer le déterminisme.

6.3. Les méthodes synchrones

Les méthodes synchrones sont fondées sur un principe complètement différent, né au début des années 1980 dans trois laboratoires français. Elles supposent que les différents composants du système sont cadencés par la même horloge logique et savent communiquer en temps zéro. Cette simplification théorique est analogue à celle de la vision logique des circuits électriques synchrones. Elle permet le développement de langages et formalismes graphiques parallèles particulièrement simples et élégants, ayant des sémantiques mathématiques parfaitement définies. Le développement des théories mathématiques associées dans les 25 dernières années a montré comment compiler efficacement les programmes synchrones de façon efficace, soit en programmes C ou ADA séquentiels, soit en circuits électroniques, et comment les vérifier formellement.

Les langages Lustre (CNRS Grenoble) et Esterel (Ecole des Mines et INRIA Sophia-Antipolis) sont développés et commercialisés par la société de l'auteur, avec leurs ateliers logiciels complets : éditeurs, simulateurs, générateurs de code, vérifieurs, etc. Ils sont utilisés par de nombreuses sociétés industrielles dans les domaines avioniques, ferroviaires, nucléaire, industrie lourde et électronique grand public. A titre d'exemple, plusieurs millions de lignes de code générées par SCADE assurent des fonctions essentielles dans l'Airbus A380 : pilotage, commande des réacteurs, freinage, etc.

6.4. Séminaire : La certification, ou comment faire confiance au logiciel pour l'avionique critique

Gérard Ladier, responsable des méthodes et de la qualité logicielle chez Airbus Industries, a présenté les méthodes de certification du logiciel critique en avionique. La norme imposée est la DO-178B, qui définit les processus de vérification à appliquer à ces logiciels. Cette norme reconnaît clairement la spécificité du logiciel par rapport aux autres composants de l'avion, et donc le besoin d'un traitement spécifique. La vérification se fait par une conjonction d'activités concernant le processus de développement :

— revues des codes et documentations ;

— tests intensifs, autonomes ou sur le simulateur avion ;

— caractérisation de la couverture des exigences fonctionnelles (haut niveau) et du code (bas niveau) apportée par les tests ;

— traçabilité complète entre toutes les étapes, des exigences fonctionnelles au code et aux tests.

Elle fait intervenir des autorités extérieures, qui vérifient la validité et la complétude des vérifications faites par le constructeur.

G. Ladier a montré comment le processus de certification logicielle est mis en place chez Airbus, et comment il a permis d'obtenir des résultats exceptionnels : zéro bug détecté en vol sur l'A320 en plus de 50 millions d'heures de vol. Une nouvelle norme DO-178C est définie par un groupe de travail international de 120 personnes fonctionnant sur le principe du consensus total. Elle mettra davantage l'accent sur un approche orientée produit, avec la qualification d'outils de haut niveau, conception fondé sur des modèles abstraits, méthodes orientées objet, génération automatique de code et vérification formelle.

7. Cours « la chasse aux bugs », ou la vérification des programmes

L'ensemble des cours a insisté sur la notion centrale de bug en informatique. L'ignorer ou la sous-estimer, c'est s'exposer à de graves conséquences. La vérification des circuits et logiciels a pour but d'assurer la conformité aux spécifications, et donc l'absence de bugs. Elle effectue de deux manières complémentaires, par test ou par calcul formel. Les tests reposent sur une exécution directe du circuit ou du programme dans son environnement réel ou simulé, avec observation de son comportement. Les stimuli sont produits manuellement ou de façon aléatoire guidée, et l'on observe les résultats, la couverture du code et celle des spécifications pour mesurer ce qui a été vérifié. Comme le nombre de cas d'exécution est *a priori* non borné, le test ne peut jamais être exhaustif.

A l'opposé, la vérification formelle s'effectue sans exécuter le programme, mais en *calculant dessus* manuellement ou automatiquement à l'aide de différentes logiques associées au langage de programmation. Elle permet des garanties à 100 %, mais se heurte à des problèmes de complexité algorithmique dès que les questions sont trop dures ou les objets trop gros.

7.1. Qu'est-ce qu'une spécification ?

La notion même de vérification pose immédiatement une difficulté essentielle : que veut réellement dire spécifier une application ? Dans les bons cas, les choses sont simples : spécifier qu'une liste de nombres est triée est trivial : il s'agit de fournir une liste triée comportant les mêmes éléments avec la même multiplicité. La *correction totale* du programme sera alors assurée. Mais les choses sont beaucoup plus complexes pour le pilotage d'un avion ou le bon fonctionnement d'un microprocesseur ou d'un système d'exploitation. On s'intéresse alors à des *spécifications partielles*, exigeant le respect de propriétés individuellement simples. Citons quelques exemples : l'ascenseur ne voyagera jamais la porte ouverte et finira par passer par tous les étages ; le train d'atterrissage ne pourra jamais être rétracté quand l'avion est au sol ; le système d'exploitation ne pourra jamais se geler et devenir perpétuellement inactif. Une conjonction de telles propriétés peut quelquefois devenir une spécification complète.

7.2. La vérification des systèmes d'états finis

Le cas le plus favorable pour la vérification est celui des *systèmes d'états finis*, qui ne peuvent prendre qu'un nombre fini d'états distincts. Ils sont très nombreux dans les applications industrielles : circuits à mémoire limitée, interfaces homme-machine, systèmes de contrôle/commande embarqués, etc. Leur vérification formelle a fait récemment des progrès considérables, grâce aux avancées majeures sur le problème SAT discutées en 3.4, ainsi qu'à l'introduction des techniques de vérification par *model checking* pour raisonner sur l'évolution des systèmes dans le temps[3]. Une propriété comme « l'ascenseur ne voyagera jamais la porte ouverte » se montre maintenant sans difficulté à l'aide de systèmes de vérification standard, ce qui a été démontré dans le cours avec l'outil Esterel Studio. Ce type de vérification devient clef pour les systèmes embarqués critiques.

En conception assistée des circuits, pratiquement toutes les transformations élémentaires intervenant dans les étapes de synthèse mentionnées en 4.2 sont vérifiées formellement. Des propriétés de sous-systèmes critiques comme les caches et les pipelines sont aussi vérifiées formellement autant que possible mais restent très difficile. Pour toutes les applications industrielles, le grand problème ouvert reste la prédictibilité du coût de la vérification, industriellement nécessaire pour allouer les budgets correspondants.

7.3. L'indécidabilité de la vérification des systèmes généraux

Les programmes généraux peuvent être d'états infinis, et donc hors de portée des techniques précédentes. Prenons une propriété élémentaire comme « le programme *p* s'arrête-t-il quand on lui donne une donnée *d* ? ». Turing a montré que cette

3. Joseph Sifakis, du CNRS Grenoble, a obtenu le prix Turing 2007 pour la co-invention du model-checking.

propriété est *indécidable* en général, c'est-à-dire non vérifiable par un algorithme prenant p et d comme données et s'exécutant toujours en temps fini.

Montrons ce résultat historique. On utilise d'abord le principe de numérisation de Gödel, voyant tout programme et toute donnée comme des nombres entiers p et d, et toute machine φ comme une fonction partielle sur les entiers définie si et seulement si p s'arrête sur d. Ecrivons $\varphi_p(d)$ pour l'application de p à d. On généralise le vieux paradoxe du menteur[4] de la façon suivante : supposons qu'il existe un programme a tel que le calcul $\varphi_a\ (2^p3^q)$ s'arrête toujours et réponde 0 si $\varphi_p(d)$ s'arrête et 1 si $\varphi_p(d)$ ne s'arrête pas ; le programme a fournirait alors un test total d'arrêt de p sur d. Construisons un autre programme m ainsi :

$$\varphi_m(d) = 0 \text{ si } \varphi_a\ (2^d3^d) = 1$$
$$= \varphi_m(d) \text{ si } \varphi_a\ (2^d3^d) = 0$$

Par construction, $\varphi_m(d)$ boucle si et seulement si et seulement si $\varphi_d(d)$ ne boucle pas. Poser $d = m$ fournit immédiatement une contradiction, démontrant par l'absurde que a ne peut exister.

Beaucoup d'autres propriétés élémentaires peuvent être montrées indécidables par réduction au problème de l'arrêt, par exemple l'équivalence de deux programmes p et q pour toutes données.

7.4. Systèmes d'aide à la preuve interactive

Bien qu'elle pose une frontière fondamentale à la vérification, l'indécidabilité est contournable dans beaucoup de cas, car beaucoup de propriétés peuvent se prouver par récurrence. Par exemple, la preuve d'arrêt de l'algorithme de tri présenté en 3.1 est simple, puisque chaque appel récursif réduit la longueur des listes arguments. En pratique, les preuves de terminaison ou de correction d'algorithmes généraux peuvent être très techniques. Elles ont donc été mécanisées très tôt dans des systèmes *d'assistance à la démonstration*. Partant de systèmes assez rudimentaires comme Boyer-Moore (U. Texas) et LCF (Stanford), on est allé vers des systèmes beaucoup plus sophistiqués comme ACL-2 (MIT), HOL (Cambridge), Isabelle (Cambridge), PVS (Stanford Research Institute) ou Coq (INRIA).

Des succès importants ont été obtenus récemment : la vérification des arithmétiques flottantes des microprocesseurs d'Intel et AMD, la vérification d'un compilateur C en Coq par Xavier Leroy mentionnée en 5.4, la vérification de protocoles de sécurité pour cartes à puce, et l'authentique exploit qu'est la vérification formelle du théorème des 4 couleurs par Georges Gonthier[5]. A l'avenir,

4. L'homme qui affirme « je mens » ne peut ni mentir ni dire la vérité.

5. Il est intéressant de noter que la partie la plus dure de cette preuve était celle d'un lemme considéré comme « folk result » par les mathématiciens qui avaient publiée la première preuve agréé par la communauté.

il est hautement souhaitable de pouvoir disposer de bibliothèques d'algorithmes vérifiés formellement par ces méthodes.

7.5. Lien avec les langages de programmation

L'objectif de la vérification formelle est de prouver la correction des programmes que l'on écrit effectivement. Or, on ne peut rien prouver sur un programme si on ne sait pas dire exactement ce qu'il fait. Il faut donc que le langage dans lequel il est écrit ait une *sémantique formelle* au sens présenté en 5.3. Diverses méthodes de définitions sémantiques sont adaptées à la vérification formelle. Citons la logique de Floyd-Hoare à base de pré- et post conditions (prédicats valides avant et après l'exécution d'une instruction), celle de Scott fondées sur la théorie des treillis, et la sémantique opérationnelle structurelle de Plotkin fondée sur la définition d'une logique directement associée aux instructions du langage. Conçues théoriquement dans les années 1970-1980, ces logiques voient leurs applications fleurir dans les systèmes d'aide à la vérification.

7.6. Séminaire : l'interprétation abstraite

Patrick Cousot, professeur à l'Ecole Normale Supérieure, a présenté l'interprétation abstraite, qu'il développe avec son équipe depuis trente ans. L'idée est de faire calculer symboliquement le programme selon les mêmes lois d'exécution mais sur des données plus abstraites, comme des intervalles ou des polygones au lieu de nombres individuels. On peut alors approximer supérieurement l'ensemble des calculs possibles en temps fini, et détecter l'impossibilité de certaines erreurs critiques : accès à un tableau hors de ses bornes, division par zéro, débordement arithmétique (erreur qui a conduit à l'explosion d'Ariane 501).

L'interprétation abstraite repose sur un cadre général mathématiquement sophistiqué et sur le développement d'une collection de domaines abstraits (intervalles, octogones, ellipsoïdes, etc.). Son implémentation demande une ingénierie subtile pour bien combiner les propriétés des différents domaines. Elle a été industrialisée par plusieurs sociétés (Polyspace, AbsInt, etc.). L'équipe de P. Cousot a développé le logiciel Astrée, utilisé par Airbus pour vérifier l'absence d'exception arithmétique dans le code de pilotage de l'A380, ce qui constitue un des plus grands succès actuels de la vérification formelle.

7.7. Séminaire : preuve et calcul, des rapports intimes

Gilles Dowek, professeur à l'Ecole Polytechnique, est l'auteur du livre « Les métamorphoses du calcul », Grand Prix de philosophie de l'Académie Française en 2007. Il a discuté les relations entre calcul et preuve, évidemment centrales pour la vérification des programmes. Il a montré que les algorithmes ont historiquement précédé les démonstrations, comme le prouvent de nombreuses tablettes d'argile de diverses origines, et que ces algorithmes étaient probablement prouvés corrects de façon non écrite. La notion de démonstration formelle a été introduite plus tard par

les Grecs, probablement à partir d'algorithmes déjà existants ou inventés pour résoudre des problèmes précis, avant d'être développée pour son intérêt propre et de fonder les mathématiques. Des théorèmes bien connus comme Thalès et Pythagore correspondent effectivement à des problèmes algorithmiques. L'analyse de la preuve d'Euclide pour l'algorithme du calcul du pgcd par soustraction montre sa relation très étroite avec les méthodes modernes de preuves de programmes, qui s'intéressent bien sûr à des objets beaucoup plus gros, avec des démonstrations corrélativement beaucoup plus grosses qu'il est nécessaire de vérifier elles-mêmes par ordinateur.

8. Cours réseaux

Les réseaux connectent les machines à informations entre elles. Ils ont été introduits dans deux mondes différents : celui des télécommunications, passé au numérique dans les années 1960/70, et celui des ordinateurs. Les réseaux de télécommunication avaient alors pour seul objectif la transmission de la voix, avec multiplexage des communications sur les fils. Leur développement a permis des progrès techniques essentiels en matière de transmission d'informations, dont la transmission par paquets de bits, maintenant généralisée. Les réseaux d'ordinateurs avaient un objectif initial bien différent : la transmission de fichiers entre machines distantes. Un tournant a été l'introduction publique d'Internet au milieu des années 1990, après son expérimentation chez les militaires et les chercheurs. Ce réseau des réseaux a très vite gagné en capacité, et des applications de transfert de voix y sont apparues dans les années 2000. Elles balayent maintenant les télécommunications fixes. Dans le même temps, l'industrie des télécommunications a évolué vers les services mobiles (GSM, etc.), et fourni des entrées rapides sur Internet ADSL et le câble. Cette histoire complexe repose sur des bases scientifiques et techniques que nous décrivons ici.

8.1. La transmission point à point

La brique de base est la transmission de l'information numérique d'un point à une autre par une liaison avec ou sans fil. Elle peut se faire un bit à la fois ou plusieurs bits en parallèle, avec ou sans transmission d'horloges, par échantillonnage de niveaux ou détection de fronts, etc. La tendance moderne est la transmission sérielle bit par bit, les bits étant codés par des fronts suffisamment nombreux pour reconstituer l'horloge de transmission sans la transmettre explicitement. Par exemple, le *code de Manchester différentiel* associe au moins un front par bit, avec toujours un front en milieu de cycle plus un front en début de cycle pour 0. La multiplicité des fronts fournit une quasi-horloge qui permet un décodage aisé. On peut aussi multiplexer les communications sur un seul fil en travaillant en parallèle sur de nombreuses bandes de fréquences. L'ADSL en utilise 255.

8.2. Rattrapage d'erreurs : les turbocodes

La transmission point à point n'est jamais parfaite et peut altérer les bits. La théorie de l'information de Shannon définit la quantité maximale d'information qu'on peut

transmettre sur une ligne ayant un rapport signal sur bruit donné. La théorie des codes correcteurs fournit le moyen de corriger les erreurs. Son principe est d'ajouter au message des bits redondants, qui permettent de reconstituer le message initial même altéré par la transmission. La question est de savoir combien de bits il faut ajouter en fonction du rapport signal/bruit et comment les calculer. De nombreux codes aux propriétés mathématiques diverses ont été proposés. La TNT (Télévision Numérique Terrestre), qui représentait l'état de l'art en 1994 quand sa norme a été définie, utilise deux codes successifs. On savait alors ce double codage non optimal au sens de Shannon, mais on ne pensait pas pouvoir faire mieux.

L'introduction des *turbocodes* par Claude Berrou et Alain Glavieux dans les années 1990 a résolu le problème en pratique. Au lieu de séquentialiser les deux codes, ce qui est asymétrique, on les met en parallèle. Au décodage, chaque information apportée par un code aide l'autre, comme toute découverte sur une horizontale ou une verticale d'un problème de mots croisés aide l'autre direction. Cette restauration de la symétrie permet d'atteindre pratiquement la limite de Shannon, et les turbocodes sont en voir de généralisation.

8.3. Les réseaux locaux

Les réseaux locaux ont une portée de l'ordre de la centaine de mètres. On les trouve dans les bâtiments, les usines, les automobiles, trains ou avions. Leurs temps de transmissions élémentaires sont faciles à borner, ce qui est impossible dans les réseaux globaux.

La question centrale est de régler les conflits entre stations désirant parler sur le réseau, qui produisent des collisions de messages. La première solution est d'éviter les collisions. Dans les réseaux à jetons, on fait tourner un jeton virtuel unique entre les stations, sous forme de message spécial ou de bits spéciaux dans les messages, et seule la station qui possède le jeton peut parler. C'est apparemment simple, mais insérer dynamiquement une station dans le réseau peut être complexe, et connecter deux réseaux l'est encore plus car il faut tuer un des deux jetons. Une autre technique pour éviter les collisions est de discriminer *a priori* les émetteurs en leur attribuant une fenêtre temporelle précise, déterminée par exemple par une station de base à l'aide de tirages aléatoires. C'est la technique TDMA utilisée dans le téléphone GSM.

Les réseaux à collision comme Ethernet acceptent les collisions entre stations parlant sur le réseau. Une station détecte une collision en comparant ce qu'elle émet avec ce qu'elle entend. Elle arrête alors son émission normale et émet une trame de brouillage détectable par les autres stations. Chaque station émettrice attend un temps aléatoire avant de reparler, avec l'idée que l'aléatoire va séparer les émetteurs et supprimer les collisions. Si une collision se reproduit immédiatement, chaque station double la taille de sa fenêtre aléatoire, etc. Une méthode plus efficace serait que chaque station tire au sort si elle reparle immédiatement ou pas,

diminuant probabilistiquement et itérativement la taille de l'ensemble en collision par 2 (protocole de Kapetanakis).

Toutes les techniques précitées présentent un certain degré d'indéterminisme qui peut être inacceptables pour les réseaux devant offrir des garanties de temps réel, par exemple pour synchroniser les freins et la suspension d'une voiture. Des réseaux locaux plus déterministes sont en cours de déploiement en automobile ou avionique : citons TTP et FlexRay, qui utilisent des protocoles fondés sur le temps absolu et la synchronisation d'horloges, et des variantes déterministes d'Ethernet.

8.4. Les réseaux globaux

Il existe deux grandes technologies de réseaux globaux : l'allocation de ressources et le routage dynamique. L'allocation de ressources est la base de la téléphonie classique. Une transmission commence par l'établissement d'une communication, avec choix d'un chemin d'acheminement et allocation de ressources dans chacun des nœuds du chemin. Une fois l'allocation faite, la transmission des paquets est très simple et la qualité garantie. Cependant, l'établissement de la communication est complexe et doit être repris à zéro cas de panne sur le chemin ; il peut être refusé en cas de saturation des ressources ; des ressources sont bloquées inutilement en cas de silence sur la ligne ; enfin, le système passe difficilement à l'échelle gigantesque des liaisons mondiales actuelles.

Le routage dynamique des paquets a été introduit initialement dans le réseau Cyclades de L. Pouzin, et repris dans le réseau ARPA puis dans le réseau Internet. La transmission repose sur un système d'adressage mondial hiérarchique et universel (adresses IP = Internet Protocol). Chaque paquet contient l'adresse complète de son destinataire et est transmis par des routeurs sans allocation de ressources. Les routeurs maintiennent à jour des tables de routage, qui leurs disent à quel autre routeur ou équipement terminal transmettre un paquet reçu en fonction de son adresse. Les tables de routage sont échangées dynamiquement sur le réseau entre les routeurs, assurant ainsi une gestion quasi-optimale de tous les changements dans le réseau. Celui-ci n'a jamais de vision exacte de sa propre configuration. Il n'y a donc pas de garantie de qualité, mais un mécanisme « best effort » simple, souple et extensible. Les réseaux de ce type sont robustes aux pannes, mais restent vulnérables aux attaques, par exemple par surcharge des routeurs. Ils tendent à supplanter les réseaux de télécommunications classiques, qui peuvent cependant servir aussi à transporter les paquets Internet.

Pour gérer des centaines de milliards d'objets sur le réseau, il faudra introduire un adressage beaucoup plus riche (celui d'IP v6, qui entre progressivement en fonction), augmenter considérablement les bandes passantes par tous les moyens utilisables (radio, fibre optique, satellites, etc.), augmenter la résistance aux attaques, etc.

Mais le plus spectaculaire reste l'extraordinaire panoplie d'innovations qu'ont permis les réseaux : courrier électronique, sites Web, commerce électronique,

moteurs de recherche, travail coopératif, etc. Nous en avons sélectionnées deux pour le séminaire associé au cours.

8.5. Séminaire : qu'est-ce qu'un moteur de recherches ?

François Bourdoncle, co-fondateur de la société Exalead, a présenté l'anatomie d'un moteur de recherche, qui comprend trois sous-fonctions : le rapatriement des pages à collecter sur Internet, leur indexation, et le calcul et la présentation des réponses aux questions. Le premier moteur Altavista, tenait dans une grosse machine et indexait 100 millions de pages. Les moteurs modernes utilisent des milliers de PCs et indexent des dizaines de milliards de pages multilingues et bientôt multimedia. La collecte des pages exploite la structure particulière du graphe des pages Web en « nœud papillon », avec les grands portails comme centres d'aiguillage. L'indexation repose sur un codage efficace des listes inverses mots vers documents. F. Bourdoncle a expliqué les algorithmes de construction et de consultation de l'index, ainsi que le calcul de l'ordre des pages dans la réponse qui a construit la suprématie de Google. Il a enfin discuté les immenses enjeux économiques associés.

8.6. Séminaire : le pair à pair et la transmission épidémique d'information

François Massoulié, de Thomson, a présenté la transmission d'information de pair à pair, qui est dominante pour la distribution (légale ou illégale) des films ou de la musique sur Internet, et occupe maintenant 80 % de la bande passante du Web. L'idée est de ne pas utiliser de serveur centralisé, mais de transformer chaque récepteur d'un flux en un relais de transmission pour ce flux. Les virus ont été la première application de cette technique ; leur nom montre bien qu'ils se propagent de façon épidémique. F. Massoulié a détaillé le problème central du choix des paquets d'information à relayer à chaque instant par un terminal. Des variations subtiles d'algorithmes peuvent donner des résultats très différents. Le meilleur algorithme semble être d'envoyer le dernier paquet utile de l'émetteur à une cible aléatoire, ce qui est loin d'être intuitif. Cette diffusion fournit un débit arbitrairement proche de l'optimal, avec délai optimal.

Le pair à pair est très simple et très efficace, bien que reposant sur des décisions aléatoires et décentralisées. La recherche s'oriente maintenant vers d'autres applications, comme la réalisation de grands calculs distribués sur de très nombreux processeurs distants.

9. Cours images

Le cours image s'est composé d'une courte présentation suivie de trois séminaires décrits ci-dessous.

9.1. Séminaire : de l'imagerie médicale au patient virtuel

Nicholas Ayache, directeur de recherches à l'INRIA, a présenté les progrès de la médecine et de la chirurgie induits par l'imagerie médicale et la modélisation

numérique du corps humain. Les nouvelles techniques d'imagerie (Scanner, IRM, TEP, etc.) fournissent des images aux caractéristiques distinctes et complémentaires. Il est possible de fusionner ces images à l'aide d'algorithmes mathématiques pour obtenir des images résultantes 3D ou 4D donnant beaucoup plus d'informations que les images séparées. Ceci améliore le diagnostic et fournit une aide à la thérapie médicamenteuse ou chirurgicale : planification, simulation, contrôle de la réalisation et suivi. N. Ayache a présenté de nombreux exemples d'applications: neurochirurgie et chirurgie du foie assistées par ordinateur, anatomie algorithmique du cerveau ou du cœur, suivi de l'évolution des tumeurs, modèles électromécaniques du cœur avec simulations de pathologies, etc. Le séminaire s'est terminé par la présentation des nouvelles techniques d'imagerie microscopique *in vivo*, qui permettent par exemple d'explorer en temps réel les alvéoles pulmonaires au niveau des cellules individuelles.

9.2. Séminaire : pourquoi le numérique révolutionne la photographie

Ce séminaire a été donné par Frédéric Guichard, directeur scientifique de la société *DxO Labs*. La photographie numérique repose sur un couple optique/logiciel, où le logiciel joue le rôle le plus important. En numérique, il est possible de défaire par logiciel les déformations des objectifs et capteurs : distorsions géométriques, aberrations chromatiques, certains types de flous, amélioration du contraste, réduction du bruit. C'est ce que fait le logiciel DxO Optics Pro, dont la puissance a été montrée par de nombreux exemples.

Cette nouvelle approche révolutionne l'optique. Plutôt que de concevoir des zooms chers à beaucoup de lentilles ayant des distorsions modérées mais complexes, il sera meilleur de concevoir des zooms très simples et bon marché, ayant des distorsions plus fortes mais faciles à défaire. On peut aussi fabriquer des objectifs numériques à grande profondeur de champ, en profitant du fait qu'on peut rendre une photo globalement nette si une seule de trois couleurs primaires (rouge, vert, bleu) est nette. Au lieu d'essayer de diminuer l'aberration chromatique longitudinale de l'objectif, qui fait que les couleurs ne focalisent pas à la même distance, on l'augmente par des matériaux adaptés, séparant au mieux les hyperfocales de chaque couleur. On obtient ainsi une bonne netteté de 20 cm à l'infini, chose impossible en optique non-numérique.

9.3. Séminaire : traitement d'image pour télévision haute définition

Stéphane Mallat, professeur à l'Ecole Polytechnique et fondateur de la société *Let It Wave*, a présenté divers algorithmes fondamentaux pour la télévision haute définition (TVHD) et leur implémentation matérielle. La TVHD demande des puissances de calcul considérables qui se comptent en téra-opérations par secondes (10^{12} *ops*). Un bon exemple est le redimensionnement spatio-temporel des images, qui change le nombre de points et la fréquence temporelle de leur flux, passant par exemple une séquence vidéo standard en résolution supérieure et à 100 Hz. Les interpolations adaptées à l'expansion dans l'espace sont bien connues en

photographie. L'expansion dans le temps est plus complexe, car elle demande une interpolation dynamique fine entre les images données pour créer les images manquantes, sous peine de sautillements désagréables. S. Mallat a présenté des techniques d'estimation de mouvement pixel par pixel, bien plus fines que les techniques classiques qui opèrent sur des blocs, et a montré comment les implémenter efficacement sur des circuits dédiés. Il a également montré les faiblesses des algorithmes de compressions classiques de type MPEG et comment les éliminer à l'aide d'une nouvelle transformée en bandelettes.

10. Séminaire : la cryptologie

Jacques Stern, professeur à l'Ecole Normale Supérieure et président de la société Ingenico, a présenté les concepts fondamentaux de la cryptologie, qui est la science du transfert sûr de l'information entre parties. Un élément essentiel en est le chiffrement des messages, qui doit obéir à plusieurs critères : facilité de chiffrement et difficulté de déchiffrement, sécurisation des échanges de clefs, etc. Il y a deux types principaux de protocoles : ceux à clef partagées, efficaces mais susceptibles d'attaques lors du partage de la clef, et ceux à clef publique, dont le fameux RSA (Rivest-Shamir-Adleman), très employé sur Internet. J. Stern a présenté les propriétés mathématiques des algorithmes sous-jacents, puis les nouveaux algorithmes fondés sur les fonctions elliptiques, qui peuvent être plus efficaces que ceux fondés sut les grands nombres premiers comme RSA. Il a montré les nouveaux types d'attaque et leurs contre-mesures. Il a enfin expliqué un cas pratique : la validation d'une transaction de carte bleue sur un terminal portable.

11. Colloque informatique et bio-informatique

Le 23 mai 2008 un colloque informatique et bio-informatique a terminé le cours. La matinée a été consacrée à la bio-informatique, qui est en plein bouillonnement, et sera incontestablement un des très grands sujets d'avenir de la recherche scientifique. Les échanges entre biologie et informatique sont doubles. D'abord, la biologie n'est évidemment pas qu'une affaire de molécules. Celles-ci servent à transporter information et énergie dans le corps, notions elles-mêmes physiquement reliées et objets de l'informatique. En retour, la biologie devrait à terme fournir de nouvelles idées en termes de moyens de calcul. Malgré leur extraordinaire efficacité, ceux des ordinateurs restent en effet bien rudimentaires, puisque limités à faire très vite et très exactement des opérations individuellement stupides. La biologie offre une vision exactement inverse : faire lentement, de façon pas très fiable mais massivement parallèle et probabiliste des opérations bien plus variées. Comme les circuits actuels dépassent allègrement le milliard de transistors, les limitations de taille sont de moins en moins présentes. Passer à des modes de calculs de type plus biologique est donc particulièrement tentant ; mais il faudra se méfier des idées simplistes, qui ont toujours échoué par échec du passage à l'échelle.

S'abstraire de la nature précise des molécules pour comprendre de façon plus globale les chemins d'information auxquels elles participent est la première étape

indispensable pour ne pas être noyé par les détails. Dans le premier exposé, Philippe Kourilsky, professeur au Collège de France, a analysé comment cela pourrait se faire pour le système immunitaire vu comme un grand système d'informations. Mais le système immunitaire un très grand système d'informations faisant intervenir des milliers de processus et de médiateurs dans de nombreux types de communications, et sa compréhension informationnelle prendra certainement beaucoup de temps.

L'informatique fournit de nouveaux moyens d'étude des processus biologiques. Le séquençage du génome est un exemple bien connu. Le second exposé, par François Fages, directeur de recherches à l'INRIA, en a présenté un autre : l'étude des réactions chimiques dans la cellule au moyen de techniques originellement introduites pour la vérification formelle des circuits et programmes (cf. 7.2). L'idée est de présenter ces réactions dans le cadre conceptuel de la machine chimique (cf. 3.5) devenue ici *BIOCHAM*, d'instrumenter les équations de cette machine avec des lois cinétiques probabilistes, et de faire calculer des systèmes de résolution de contraintes Booléennes ou numériques pour répondre à beaucoup de questions inaccessibles à l'étude manuelle sur les chaînes réactionnelles.

Le cerveau est évidemment un des organes les plus fascinants au niveau bio-informatique. Alexandre Pouget, professeur à l'université de Rochester et en année sabbatique au Collège de France, a exposé la nouvelle vision des mécanismes globaux de calcul dans le cerveau fournie par les neurosciences computationnelles. A travers une série d'exemples, il a montré que le cerveau fait essentiellement des évaluations probabilistes rapides et des choix assez simples, en particulier pour les activités motrices demandant de prendre des décisions non-triviales. Le cerveau brille également par ses capacités d'apprentissage, dues à la fois à la modification permanente des connections synaptiques en fonction des sollicitations externes et à l'ajustage permanents des critères d'évaluation probabilistes employés. La modélisation informatique de ces mécanismes apportera des contributions fondamentales à leur compréhension et à leur simulation informatique.

L'après-midi du colloque a été consacrée à des sujets plus directement informatiques. Alberto Sangiovanni-Vincentelli, professeur à Berkeley et directeur du GIE Parades à Rome, a présenté l'évolution inéluctable du Web vers l'intégration des objets physiques. D'ici une quinzaine d'années, il pourrait y avoir de l'ordre d'un millier d'objets informatisés et mis en réseau par être humain. Le Web comptera alors des téra-clients (téra = 10^{12}), qui iront des objets audio-visuels classiques aux prothèses médicales, en passant par tous les composants des maisons ou des systèmes de transports (cf. 6). Les pucerons électroniques correspondants effectueront des fonctions de surveillance, de conduite, d'optimisation, et de communication. Cette nouvelle extension du monde numérique demandera une nouvelle approche pluridisciplinaire, intégrant informatique, nanotechnologies, et biotechnologies, et donc de nouvelles techniques de constructions d'objets et de systèmes.

Martin Abadi, professeur à l'Université de Santa Cruz (Californie) et chercheur à Microsoft Research, a enfin présenté les problèmes posés par la sécurité des

données et des interactions informatiques. Ces problèmes sont devenus cruciaux avec la généralisation des échanges sur support banalisé de type Internet avec ou sans fil. Les solutions reposent sur des protocoles de sécurité utilisant des techniques cryptographiques. Mais les codes secrets sont loin de suffire à garantir la sécurité. Leur utilisation doit être encadrée par des protocoles très fins pour résister aux tentatives d'intrusions malignes de toutes sortes, fruits d'imaginations sans limites. L'étude serrée des protocoles et standards à clefs publiques ou privées successivement proposés a permis d'augmenter considérablement leur sécurité. Le sujet reste évidemment très actif en raison de son importance stratégique.

12. Conclusion

Ce cours général sur le monde numérique a représenté un défi important. Il n'était évidemment pas simple de présenter de façon pertinente les sujets individuellement très vastes de chacun des cours en deux heures. Mais j'ai jugé que c'était indispensable, car il faut absolument tordre le cou à l'ignorance sur un sujet qui façonne autant notre vie quotidienne, et donc faire mieux connaître la science informatique.

Remerciements : je remercie le Collège de France, institution merveilleuse et seul endroit (au monde ?) où donner ce genre de cours est possible, ainsi que la Fondation Bettencourt-Schueller pour son soutien à la chaire d'innovation technologique. Je souhaite enfin remercier très chaleureusement Marie Chéron et Cécile Barnier qui m'ont aidé (supporté, comme on dit en anglais) pendant tout le cours, Marion Susini et son équipe pour la rapidité et la qualité de l'installation des vidéos sur le Web, et tous les personnels du Collège qui m'ont aidé avec gentillesse et efficacité.

Bibliographie

G. Berry, *Pourquoi et comment le monde devient numérique*, Fayard/Collège de France, 2008.

D. Harel, *Algorithmics, The Spirit of Computing*, Addison-Wesley Publishing co., 2004.

F. Anceau. Y. Bonnassieux, *Conception des circuits VLSI*, Dunod, 2007.

G. Dowek, J.-J. Lévy, *Introduction à la théorie des langages de programmation*, Les éditions de l'Ecole Polytechnique, 2006.

L. Zaffalon, *Programmation synchrone de systèmes réactifs avec Esterel et les SyncCharts*, Presses Polytechniques Romandes, 2005.

G. Dowek, *Les métamorphoses du calcul*, Le Pommier, 2007.

J.-M. Sepulchre, *DxO pour les photographes*, Eyrolles, 2008.

J. Stern, *La science du secret*, Editions Odile Jacob, 1998

Publication

G. Berry, *Pourquoi et comment le monde devient numérique*, Fayard/Collège de France, 2008.