Décroissance numérique : quels écosystèmes logiciels pour l'informatique frugale ?

Mots-clefs : Durabilité, Sobriété numérique, Langages de programmation, Compilation, Informatique minimaliste ou embarquée.

Contexte

L'informatique évolue depuis des décennies en suivant une croissance exponentielle (en particulier la loi de Moore qui affirme que le nombre de transistors sur une puce double tous les deux ans, qui s'est assez bien vérifiée en pratique depuis les années 70). Cette croissance atteint ses limites : la consommation d'énergie liée au numérique est aujourd'hui importante (2 à 4% des émissions de gaz à effets de serre mondiales) et également en augmentation exponentielle (doublement tous les dix ans environ) et les ressources en termes de minerais rares sont limitées et ne pourront suivre une croissance exponentielle à long terme. Les experts du changement climatique (comme GIEC) proposent au contraire des scénarios où les émissions de gaz à effets de serre décroissent pour atteindre la neutralité carbone autour de 2050.

S'il est tentant de penser que les progrès technologiques suffiront à réduire la consommation d'énergie et l'utilisation de ressources, l'histoire jusqu'ici montre plutôt le contraire : les progrès en efficacité énergétique ont presque toujours été compensés par une augmentation des usages (par exemple, les téléphones mobiles d'il y a 20 ans étaient conçus sur des technologies bien moins efficaces énergétiquement que celles des smartphones, et pourtant avaient une autonomie sur batterie beaucoup plus longue). On parle pour décrire ce phénomène d'« effet rebond ».

Il est donc important de penser à une réduction des usages pour permettre de réels progrès sur l'impact écologique du numérique : envisager un futur où la quantité de calculs réalisés sur l'ensemble de la planète décroît d'année en année au lieu de croître indéfiniment. Il est très difficile de prédire l'évolution du numérique sur les décennies à venir, mais cette décroissance risque d'être imposée à l'humanité faute de ressources (on voit déjà en 2022 des pénuries de composants électroniques qui sont entre autres la conséquence d'évènements climatiques extrêmes dans les pays où se trouvent les usines de microélectroniques).

Beaucoup de chercheurs pensent que ce changement de paradigme doit être préparé et anticipé, et qu'il est urgent de réfléchir à un numérique souhaitable et durable sur le long terme. C'est la motivation de la création de l'équipe Phénix, au laboratoire CITI à la Doua.

Les outils permettant d'exécuter du code sur des architectures minimalistes (microcontrôleur, ordinateurs avec très peu de mémoire, etc.) ont déjà un très bon niveau de maturité. La manière classique de programmer ces architectures est d'utiliser un ordinateur puissant pour le développement et la compilation croisée vers une architecture peu puissante (c'est-à-dire, utiliser un compilateur générant du code pour l'architecture minimaliste, mais qui s'exécute sur un PC classique). Dans une optique de décroissance globale, il serait au contraire souhaitable, et probablement nécessaire, que l'écosystème complet permettant le développement et l'exécution puisse tourner sur une architecture minimaliste. Une chaîne d'outil capable de se compiler elle-même est dite *méta-circulaire*.

1 Objectif du projet

L'objectif de ce projet est de réaliser une étude, à la fois bibliographique et expérimentale, pour comparer les outils de développements et déterminer les avantages et inconvénients de chacun dans un contexte d'informatique frugale.

Bien sûr, il ne s'agit pas d'une étude exhaustive, qui dépasserait de loin le cadre d'un projet comme celui-ci. Nous souhaitons dans un premier temps développer un ensemble d'exemples de code dans plusieurs langages pour avoir quelques points de comparaisons. Il s'agit d'un tour d'horizon en vue d'orienter les discussions et recherches futures de l'équipe.

La liste des langages (et écosystèmes associés) n'est pas fixée, mais nous pensons à :

— Le langage Forth, un langage à pile interprété pour lequel il est relativement simple d'écrire un interprète.
— Le langage Rust, un langage offrant des garanties similaires aux langages de haut niveau comme Java, tout en donnant un contrôle total à l'utilisateur sur l'accès au matériel et à la gestion de la mémoire.
— Le langage C, qui n'est probablement pas le meilleur candidat, mais pourra servir de base aux comparaisons.

Quelques exemples seront développés dans chacun de ces langages (quelques exemples très simples comme un calcul de factorielle, au moins un exemple intensif en calcul comme un solveur de Sudoku, un exemple d'algorithme utilisé dans le cadre de machines communicantes comme un algorithme de chiffrement).

Les critères de comparaison incluent :

— La facilité de programmation et debuggage,
— La quantité de mémoire nécessaire pour compiler,
— La quantité de mémoire nécessaire pour exécuter,
— Le temps nécessaire à la compilation,
— Le temps nécessaire à l'exécution.

Selon les motivations du ou des étudiant(e)s recruté(e)s, le projet pourra s'orienter plutôt vers une étude expérimentale (en essayant concrètement beaucoup d'exemples), ou une étude bibliographique (chercher des articles montrant les points forts et points faibles de chaque écosystème).

Encadrement

— Matthieu Moy, maître de conférences UCBL/LIP, `https://matthieu-moy.fr/`,
— Lionel Morel, maître de conférences INSA/CITI, `http://lionel.morel.ouvaton.org/wp/`.
— Guillaume Salagnac, maître de conférences INSA/CITI, `http://perso.citi.insa-lyon.fr/gsalagnac/`,