Numérique responsable : comment minimiser l'empreinte écologique du développement logiciel ?

Mots-clefs : Durabilité, Sobriété numérique, Langages de programmation, Compilation, Informatique minimaliste ou embarquée.
Niveau d'études : Licence (mais le sujet peut être adapté à d'autres niveaux).

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 mondiales de gaz à effets de serre) 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 microélectronique).

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'outils capable de se compiler elle-même est dite *méta-circulaire*.

En 2022, nous avons démarré 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. Les premières expérimentations portent sur la consommation mémoire pour exécuter un programme, pour compiler ce programme, et pour compiler la chaîne d'outils qui a servi à le compiler, pour plusieurs langages.

La liste des langages (et écosystèmes associés) n'est pas fixée, mais dans l'immédiat nous avons étudié :

— Le langage C, qui n'est probablement pas le meilleur candidat, mais pourra servir de base aux comparaisons. Les premiers programmes et compilateurs C tournaient sur des architectures très peu puissantes (en 1973 !), donc l'histoire a déjà prouvé que le langage C était utilisable dans un contexte minimaliste, même si les technologies ont beaucoup évolué depuis.
— C++, dont les caractéristiques sont similaires à celles de C, mais avec des constructions de plus haut niveau intéressantes pour le programmeur, mais qui peuvent entraîner une consommation mémoire plus importante.
— 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. Ce langage a déjà fait ses preuves pour l'exécution de programmes sur des architectures minimalistes, mais la possibilité de développer des programmes Rust sur ces architectures pose encore question, car le compilateur Rust est par défaut très gourmand en ressources.
— Le langage Python, qui a l'avantage d'être relativement haut niveau et ne pose pas le problème des ressources utilisées par la compilation (vu qu'il s'agit par défaut d'un langage interprété), mais qui est assez gourmand en puissance de calcul et en mémoire à l'exécution.
— Le langage Java, réputé pour être très gourmand en mémoire à l'exécution, mais qui a l'avantage d'être très répandu et d'avoir un écosystème très riche.

1 Objectif du stage

L'objectif de ce stage est de poursuivre l'étude de la consommation mémoire, en comparant pour un langage donné les différentes options disponibles pour la compilation et l'exécution. Par exemple, nous avons déjà pu montrer expérimentalement que la compilation d'un programme C même très simple demandait une quantité de mémoire très importante avec les outils modernes (GCC, clang), et nettement plus basse avec un compilateur plus minimaliste comme tcc (Tiny C Compiler), mais il reste du travail pour comparer les différentes options de compilation disponibles pour chaque chaîne d'outils. Intuitivement nous nous attendons à ce que la compilation optimisante (par exemple, `gcc -O3`) soit plus gourmande en mémoire que la compilation non-optimisée (par exemple `gcc -O0`), mais nous voudrions confirmer cette intuition et surtout quantifier la différence en la mesurant expérimentalement. Notre base de programmes d'exemples est encore limitée et développer d'autres exemples de programmes simples sera également un objectif du stage.

Ces expérimentations se feront en parallèle avec une étude bibliographique sur l'informatique frugale. Le poids accordé à l'étude expérimentale et à l'étude bibliographique pourra être adapté en fonction des motivations du ou des étudiant(e)s recruté(e)s.

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 `https://perso.citi.insa-lyon.fr/gsalagnac/`
— Michael Rao, directeur de recherche CNRS au LIP `https://perso.ens-lyon.fr/michael.rao/`