Décroissance numérique et langages de programmation : mesurer l'empreinte mémoire de la toolchain Rust et son évolution au fil des versions

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

Niveau d'études : Licence ou M1.

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, s'est assez bien vérifiée en pratique depuis les années 70. Mais 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 effet 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 pas supporter longtemps une trajectoire de croissance exponentielle. Les expert-es du changement climatique (par ex : le GIEC) proposent au contraire des scénarios où les émissions décroissent pour atteindre la neutralité carbone autour de 2050. En termes de technologies numériques, cela revient à envisager un futur où la demande globale en calcul n'augmente plus indéfiniment mais au contraire diminue d'année en année.

Beaucoup de chercheur-ses pensent que ce changement de paradigme doit être attentivement préparé et anticipé, et qu'il est urgent de réfléchir à un numérique qui serait à la fois souhaitable pour la société et soutenable sur le long terme. C'est notamment le cas de l'équipe Phénix, au laboratoire CITI à la Doua. Nos travaux empruntent à plusieurs disciplines, allant des sciences humaines et sociales (par ex : sociologie, prospective, science politique) à des contributions purement techniques (par ex : mesures de performances, logiciels pour machine nue, compilation) en passant par le champ des STS (Science and technology studies).

Dans ce projet en particulier, nous travaillons à quantifier et comparer la consommation en ressources informatiques (stockage, temps CPU, empreinte mémoire) de différents langages de programmation. En effet, comme historiquement les langages ont co-évolué avec des machines de plus en plus grosses, ils sont devenus à la fois de plus en plus sophistiqués et de plus en plus gourmands. Mais serait-ce possible d'en garder les «bonnes idées» tout en réduisant leur empreinte ? Si la puissance des machines diminue nettement dans le futur, faudra-t-il alors inventer de nouveaux langages ? Ou au contraire, revenir à des implémentations plus anciennes (le GCC de 2025 n'a pas grand-chose en commun avec le GCC de 2005) ? Parmi les langages actuels, lesquels seraient compatibles avec des machines plus petites ? En fait, cette question se décline en deux sous-questions : quels langages produisent des exécutables pouvant s'exécuter sur des machines plus petites ? Et lesquels seraient utilisables sur des machines de développement plus petites ? Pour éclairer objectivement toutes ces questions, nous travaillons à mesurer finement la consommation en ressources de divers langages.

Objectifs du stage

Très en vogue depuis une dizaine d'années, le langage de programmation Rust est prometteur en matière de frugalité. En effet, Rust offre des garanties de sûreté comparables aux

langages de haut niveau tels que Java, tout en donnant à l'utilisateur-ice un contrôle fin sur l'accès au matériel et la gestion de la mémoire. Le langage a déjà fait ses preuves pour cibler des architectures embarquées et/ou minimalistes, car il produit notoirement des exécutables optimisés en temps et sobres en mémoire. Mais la possibilité de développer en Rust sur des petites machines pose encore question, car son compilateur est très gourmand en ressources.

Partant de l'hypothèse que la consommation de mémoire vive est le premier point bloquant pour exécuter un logiciel sur un ordinateur peu puissant, nous avons développé un outil de mesure détaillée la de consommation mémoire ainsi qu'un ensemble de scripts permettant de lancer cet outil sur divers programmes : de petits benchmarks synthétiques de quelques dizaines de lignes, jusqu'à des gros logiciels comme les chaînes d'outils GCC, LLVM, etc. Nous avons également des scripts pour analyser et visualiser les mesures obtenues.

L'objectif de ce stage est de mettre en oeuvre cette méthodologie sur les implémentations successives de la chaîne d'outils Rust, pour répondre à des questions comme :

— Les progrès accomplis par le langage se traduisent-ils en pratique par une augmentation ou une réduction des besoins ? En particulier, comment se comparent l'implémentation historique du langage (écrite en ocaml) et les implémentations postérieures au *boostrap* (c'est à dire elles-mêmes écrites en Rust) ?
— Quelles phases de la chaîne sont les plus gourmandes : compilation, optimisation édition de liens, autres ? En particulier, les analyses et optimisations tardives (par ex *Link-Time Optimization*) offrent-elles un rapport coût/bénéfice favorable ? Sont-elles facultatives, comme en C/C++, ou au contraire sont-elles nécessaires en Rust pour obtenir un exécutable correct ?

Le travail s'appuyera notamment sur l'article récent de Graydon Hoare *«retrobootstrapping rust for some reason»* `https://graydon2.dreamwidth.org/317484.html` où le créateur du langage décrit de façon détaillée comment reconstruire les versions successives du langage les unes à partir des autres.

Comment candidater ?

Envoyer un mail à `matthieu.moy@univ-lyon1.fr`, `guillaume.salagnac@insa-lyon.fr` avec votre CV, quelques lignes de motivation, et éventuellement des questions. Vous pouvez joindre tout document que vous jugez utile (rapport de stage précédent, etc.).

Encadrement

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