Valentin Regnault

Nathan Corneloup

Remi Loison

Synthèses de fin de module sur la consommation énergétique du logiciel

Numérique Écoresponsable 2 (NER2)

Table des matières

1 Introduction....1
2 Avant - cahier des charges et pile technologique....4
2.1 Choix des fonctionnalités....4
2.2 Choix de la pile technologique....5
2.2.1 Choix du format des données....6
2.2.2 Choix des outils....6
3 Pendant....10
Implémentation : 10
3.1 Développement : 13
4 Après....14
4.1 Virtualisation....14
4.2 Supervision....15
4.3 Distribution 15
5 Conclusion....16

1 Introduction

Les émissions de gaz à effets de serre provoquent un réchauffement climatique global qui a, et va continuer d'avoir, des conséquences dramatiques pour l'homme. Entre autres, la montée des eaux, des méga-feux, des sécheresses et inondations qui causent une baisse de la production agricole, une chute de la biodiversité, une augmentation de l'intensité des ouragans, cyclones et typhons, ou encore une augmentation de l'intensité, de la fréquence et de la durée des canicules[Green algorithms]. Ces conséquences de nos émissions nous obligent à les réduire. Or, une grande partie d'entre elle sont causées par notre consommation d'énergie. Il va donc falloir dans un futur proche réduire ces consommations.

D'autre part, les ressources fossiles, qui comptaient pour 85 % du mix énergétique mondial en 2018, s'épuisent. Cela aussi nous impose aussi de réduire nos consommations d'énergie.

Parmi l'ensemble des activités humaines qui requiert de l'énergie, nous allons nous intéresser au logiciel. D'après la plaquette d'Écoinfo[Ecoinfo] sur la réduction de la consommation énergétique du logiciel, 5 millions d'heures.coeur de calcul (env. 25 tonnes eq CO2) sont équivalents à 25 Allers-Retours Paris - New York en avion selon l'aviation civile, ou 11 selon Ecolab. Ces émissions pourraient être réduites en améliorant la performance énergétique du matériel sur lequel on effectue ces calculs, mais on s'approche de plus en plus de la limite physique de l'optimisation énergétique des processeurs[Moore's law]. Il y a donc un enjeu de trouver d'autres manières de les réduire, et l'optimisation des logiciels en est une.

Ainsi, cette plaquette d'Ecoinfo[Ecoinfo] explique que "L'efficacité ou la sobriété du logiciel influe fortement sur le besoin de puissance de calcul, mémoire et stockage, ce qui permet de réduire le volume de matériels nécessaires au service numérique, donc de diminuer l'impact de la phase de fabrication des équipements et la consommation énergétique en fonctionnement dans une moindre mesure.". Les personnes qui écrivent des programmes informatiques ont une responsabilité quant à leur consommation, mais aussi quant au dimensionnement des infrastructures qui vont permettre d'exécuter ces programmes. Tirée d'un article sur l'impact écologique des calculs hautes performances dans l'astronomie intitulé « The Ecological Impact of High-performance Computing in Astrophysics, »[Astrophysics], la figure 1 permet de comparer les empreintes carbones de différentes activités dans le domaine de l'informatique. La courbe bleue montre notamment comment le nombre de cœurs utilisés peut radicalement changer les émissions carbones nécessaires pour faire un calcul scientifique. C'est d'ailleurs ce qui est confirmé dans un autre article sur les algorithmes verts[Green algorithms], qui explique que l'utilisation des supercalculateurs pour l'astronomie australienne émet 15 kt eq CO2 par an, soit 22 t eq. CO2 par chercheur. C'est près du double de l'empreinte carbone d'un citoyen australien moyen.

Dans ce document, nous allons étudier les différentes méthodes qui permettent de réduire l'impact du logiciel. Nous n'étudierons que des méthodes numériques, et non pas matérielles, bien que ces méthodes numériques puissent avoir un impact sur les besoins en infrastructures (dimensionnement des serveurs et des autres appareils) et sur leur durée de vie (et donc leur empreinte carbone). Nous procéderons selon le même progression que la plaquette d'Écoinfo[Ecoinfo], en étudiant les méthodes de réduction de la consommation énergétique à 3 étapes du développement d'un logiciel : avant (création du cahier des charges et choix de

la pile technologique) ; pendant (conception, implémentation et impact direct du développement) ; et après (déploiement et maintenance).

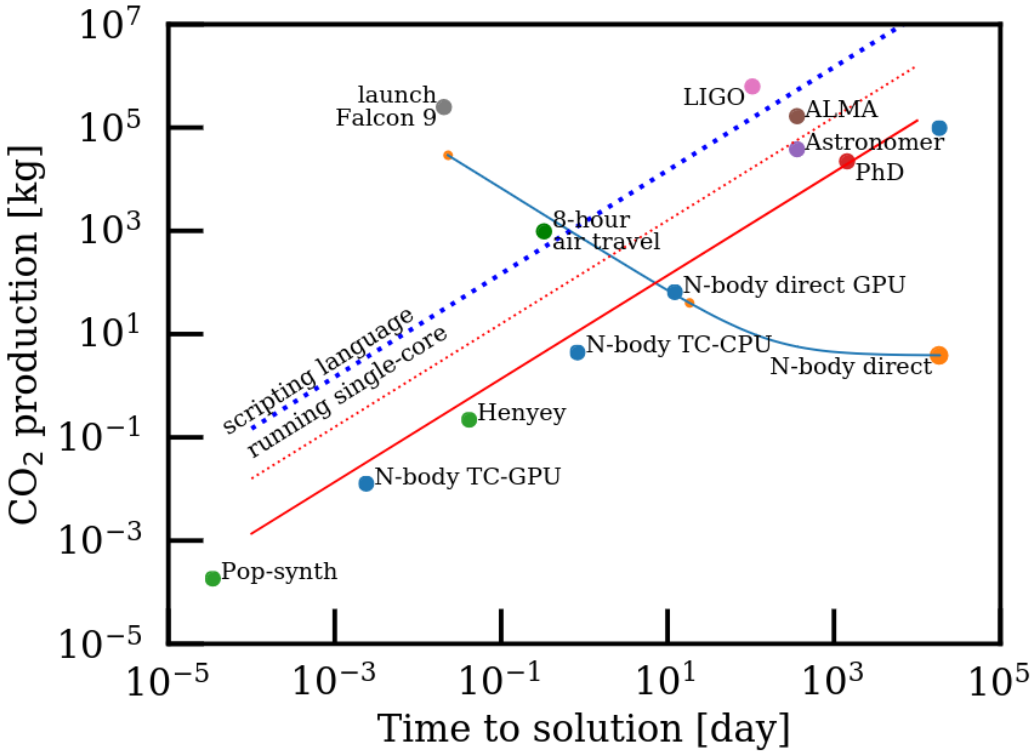


Figure 1: **CO^2 emission (in kg) as a function of the time to solution (in days) for a variety of popular computational techniques employed in astrophysics, and other activities common among astronomers [3,4]. The solid red curve gives the current individual world-average production, whereas the dotted curves give the maximum country average. The LIGO carbon production is taken over its first 106-day run (using ~ 180 kW) [5], and for ALMA a 1-year average [6]. A Falcon 9 launch lasts about 32 minutes during which $\sim 110\,000$ liters of highly refined kerosene is burned. The tree-code running on GPU is performed using $N = 2^{20}$ particles. The direct N-body code on CPU (right-most blue bullet) was run with $N = 2^{13}$ [7], and the other codes with $N = 2^{16}$. All performance results were scaled to $N = 2^{20}$ particles. The calculations were performed for 10 N-body time units [8]. The energy consumption was computed using the scaling relations of [9] and a conversion of KWh to Co_2 of 0.283 kWh/kg. The blue dotted curve shows the estimated carbon emission when these calculations would have been implemented in Python running on a single core. The solid blue curve to the right, starting with the orange bullet shows how the performance and carbon production changes while increasing the number of compute cores from 1 to 10^6 (out of a total of $7\,299\,072$, left-most orange point) using the performance model by [10].**

Figure 1 Différentes exemples de tâche courrante en astrophysique, représenté par leurs emissions en fonction du temps.[Astrophysics]

2 Avant - cahier des charges et pile technologique

2.1 Choix des fonctionnalités

La première étape du développement d'un logiciel est l'analyse du besoin auquel il va répondre. Que ce soit dans le cadre d'un calcul scientifique, du développement d'un logiciel pour le grand public ou d'un script coté serveur, il est important de choisir les fonctionnalités que nous allons implémenter en prenant en compte leur consommation énergétique.

D'après la plaquette d'EcoInfo[Ecoinfo], qui reprend une étude du Standish Group, 70% des fonctionnalités demandées par les utilisateurs ne sont pas ou très peu utilisées. Pourtant, ces fonctionnalités alourdissent le programme, le rendent plus lent, moins fluide à l'utilisation et plus consommateur d'énergie. Il est donc important de commencer par renoncer aux fonctionnalités dont le programme n'a pas besoin. Pour cela, la plaquette recommande de prendre en compte la "pertinence" d'une fonctionnalité, définie comme suit :

pertinence = utilité (le résultat doit répondre à l'attente de l'utilisateur) x rapidité (temps de réponse pour l'utilisateur) x accessibilité (par exemple pour certains handicaps)

D'autre part, les fonctionnalités doivent être choisies en fonction de leur impact énergétique. Pour cela, il est essentiel d'être capable de mesurer cet impact **avant** le développement du logiciel. C'est ce qui est proposé dans l'article "Towards Energy-Efficient Mobile Ad Optimization: An App Developer Perspective"[Ads Optimization] dans lequel les auteurs étudient l'impact d'une méthode permettant de réduire la consommation énergétique des publicités avant son implémentation grâce, entre autres, à un modèle statistique. Pour cela, ils identifient les facteurs sur lesquelles leur méthode va avoir un impact, et cherche un modèle qui représente la consommation énergétique en fonction de ces facteurs. En l'occurrence, pour modéliser la consommation énergétique des publicités, ils prennent en comptes leur nature (vidéo ou image), leur taille (en octet) et leur taux de rafraîchissement. Grâce à des mesures de consommation d'énergie faites en variant ces trois facteurs, ils arrivent à déterminer le lien entre eux. Ils construisent ainsi un modèle statistique (une équation) qui permet d'estimer la consommation énergétique d'une publicité en fonction de ces trois facteurs. Ils estiment ensuite l'impact que peut avoir leur méthode d'optimisation sur les trois facteurs et en déduisent une approximation des économies d'énergies obtenues grâce à leur méthode. Cette démarche (identification des facteurs, puis mesures pour

déterminer le lien entre ces facteurs, puis construction d'un modèle, et enfin estimation de l'impact) peut s'appliquer à différentes fonctionnalités et devrait faire partie de la création d'un cahier des charges. On peut estimer le coût énergétique qu'une fonctionnalité peut engendrer avant de l'avoir développée, et choisir les fonctionnalités que l'on va implémenter en fonction de cela.

Il est important de faire ces estimations en prenant en compte 3 ordres de consommation direct :

- Ordre 1 : combien consomme cette fonctionnalité

- Ordre 2 : comment cette fonctionnalité impact la consommation des autres fonctionnalités et du reste du système

- Ordre 3 : comment cette fonctionnalité impact l'usage par l'utilisateur de l'ensemble du logiciel.

Une illustration des 3 ordres est donnée dans un article sur la consommation énergétique des claviers virtuels de téléphones[Android Keyboard], intitulé « Greenspecting Android Virtual Keyboard ». Depuis l'invention des smartphones, les claviers ont été dématérialisés, et sont devenus des applications. Sur les smartphones Android, il existe plusieurs applications de claviers virtuels, et cette étude cherche à déterminer lesquelles consomment le plus, et quelles fonctionnalités de ces claviers virtuels consomment le plus. Notamment, l'article s'intéresse à la prédiction de mots. La plupart des claviers offrent la possibilité de compléter et/ou corriger le mot que l'utilisateur est en train d'écrire grâce à des algorithmes qui prédisent le mot le plus probable dans le contexte. La consommation d'ordre 1 de cette fonctionnalité est tout simplement le surcoût énergétique nécessaire pour exécuter cet algorithme. Mais cette prédiction permet de réduire le temps nécessaire pour écrire un texte, et donc de réduire le temps d'utilisation du clavier et du smartphone en général. Comme le smartphone est moins utilisé, il consomme moins. Il est alors important d'étudier l'impact de cette fonctionnalité sur le temps d'utilisation (ordre 2) et de savoir si le gain de temps d'utilisation compense le surcoût engendré par la prédiction de mot. Dans le cas de la prédiction de mot, la réponse est mitigée, la prédiction de mot ne compense pas toujours le surcoût de consommation requis pour calculer les suggestions, dans le cas de certains claviers, et inversement pour d'autres. Enfin, l'ordre 3 qui n'est pas étudié dans cet article consisterait à se demander si la prédiction de mots nous incite à écrire plus de texte, et donc engendre plus de consommation d'énergie.

2.2 Choix de la pile technologique

Le choix de la pile technologique, c'est-à-dire le choix des outils et technologie que l'on va utiliser pour créer notre logiciel (quel langage de programmation, quelles librairies de code, quels frameworks, quels formats de données), est une étape clef qui a un impact conséquent sur la consommation énergétique du logiciel.

2.2.1 Choix du format des données

Pour limiter la consommation énergétique de notre logiciel, le moyen le plus efficace est de ne pas en avoir besoin et de ne pas le créer. La première étape est donc de rechercher si d'autres programmes ont déjà effectué la tâche que l'on souhaite accomplir (en particulier dans le cas des calculs scientifiques). Si ce n'est pas le cas, alors il est intéressant de créer notre propre programme. Une bonne pratique est de mettre à disposition le résultat de nos calculs et notre code publiquement, de manière que d'autres personnes puissent les réutiliser. Il faut donc faire attention aux formats des données que l'on utilise et produit, pour que ces formats ne soient pas un frein à la réutilisation de nos résultats. Pour cela, la plaquette ÉcoInfo[Ecoinfo] recommande d'utiliser la méthode F.A.I.R (Findable, Accessible, Interoperable, Reusable). Notamment, la plaquette mettre l'accent sur l'interopérabilité des données : c'est leur capacité à être combinées, à être compatible avec, d'autres jeux de données. L'utilisation de format standard et indépendant des outils est recommandé. Par exemple, il est conseiller d'exporter des données d'une base de donnée en CSV ou autre format standard plutôt que dans un format spécifique à la technologie de base de donnée que l'on a utilisé.

D'autre part, mettre ces données en open source, à disposition libre et gratuite de toutes celles et ceux qui souhaitent les utiliser, permet d'éviter à d'autres utilisateurs de recalculer ces mêmes données et de créer des doublons.

2.2.2 Choix des outils

Dans l'article "The Ecological Impact of High-performance Computing in Astrophysics"[Astrophysics] l'auteur étudie l'impact des langages de programmation compilés par rapport aux langages interprétés. Les résultats, présent dans la figure 2, parlent d'eux-mêmes. Les langages interprétés comme

python (sans utilisation de librairies compilées ou d'environnement d'exécutions optimisés) nécessitent jusqu'à 100 fois plus de temps et d'énergie pour effectuer exactement la même tâche. Leur utilisation doit donc être strictement réservée à des calculs légers et à des prototypes, et jamais à des tâches lourdes ou des applications à destinations du grand public.

Cependant, le constat n'est pas tout noir : la plupart des langages interprétés disposent aujourd'hui d'environnements d'exécutions optimisés (JIT) et/ou des librairies compilées sont disponibles pour faire les opérations les plus intensives (numpy ou numba en python). On voit d'ailleurs sur la figure 2 que l'utilisation de numba permet de rapprocher drastiquement l'impact de python à celui de langages plus performant comme le C. D'autre part, les langages interprétés permettent généralement de faciliter la maintenance et les mises à jour. Dans le cas d'applications légères, ils peuvent permettre une meilleure flexibilité et une facilité de réutilisation, ce qui peut permettre à terme de réduire leurs impacts énergétiques.

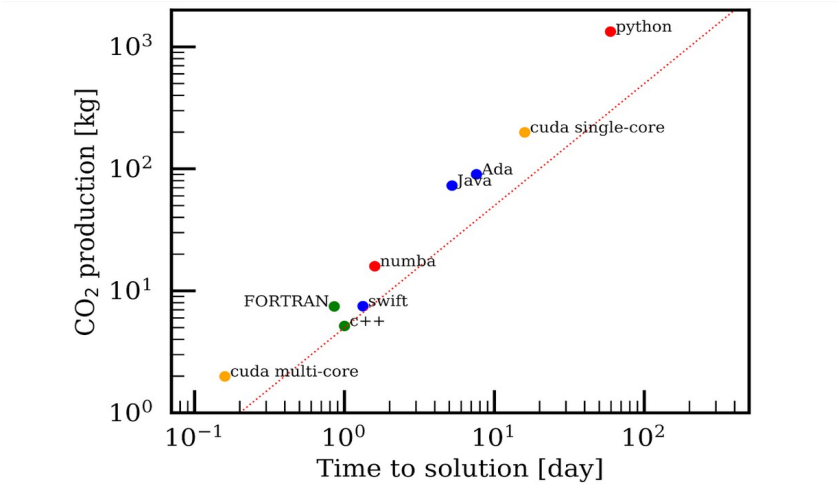
Dans la figure 3 issue de la plaquette d'Écoinfo, on observe qu'une écrasante majorité des émissions de CO2 des appareils numériques à destination du grand public (ordinateurs, smartphones…) sont dues à leur fabrication. Les créateurs de logiciels sur ces plateformes doivent prendre leurs responsabilités.

Ce qui pousse les consommateurs au renouvellement de son matériel informatique, c'est d'une part l'obsolescence matérielle (l'appareil devient défectueux avec le temps) mais aussi l'obsolescence logicielle. L'obsolescence logicielle désigne le fait de devoir changer d'appareil, car il n'est plus compatible avec les applications. Ainsi, les mises à jour ne sont pas faites pour les vieux appareils, qui ne disposent petit à petit plus des mêmes fonctionnalités que les nouveaux. Au bout d'un moment, cela les rend complètement incompatibles avec les applications. L'obsolescence logicielle désigne aussi le fait que les applications modernes consomment plus de ressources et fonctionnent donc moins bien sur les anciens appareils moins performants.

Pour éviter ce phénomène, il faut en conséquence choisir des outils qui nous permettront de mettre à jour notre logiciel sur des vieux appareils aussi longtemps que possible. Pour cela, l'usage d'outils cross-plateform est recommandé. Ces outils de création d'application fonctionnent sur le principe "Write once, run everywhere". Ils permettent d'écrire le code de nos applications de manière à ce qu'il soit compatible avec les différents appareils de différents constructeurs, avec différents systèmes d'exploitation. En général, ces outils permettent aussi de

garantir une compatibilité avec des vielles versions des systèmes d'exploitations et donc de pallier l'obsolescence logicielle. De plus, ces outils cross-platforms évitent le surcoût économique qui serait nécessaire si les entreprises devaient réécrire leurs applications pour les différents systèmes d'exploitations et les différentes versions de ceux-ci.

Figure 2 - Emissions de CO2 (en kg) en fonction du temps pour trouver la solution à une tâche d'astrophysique (en jour) [Astrophysics]



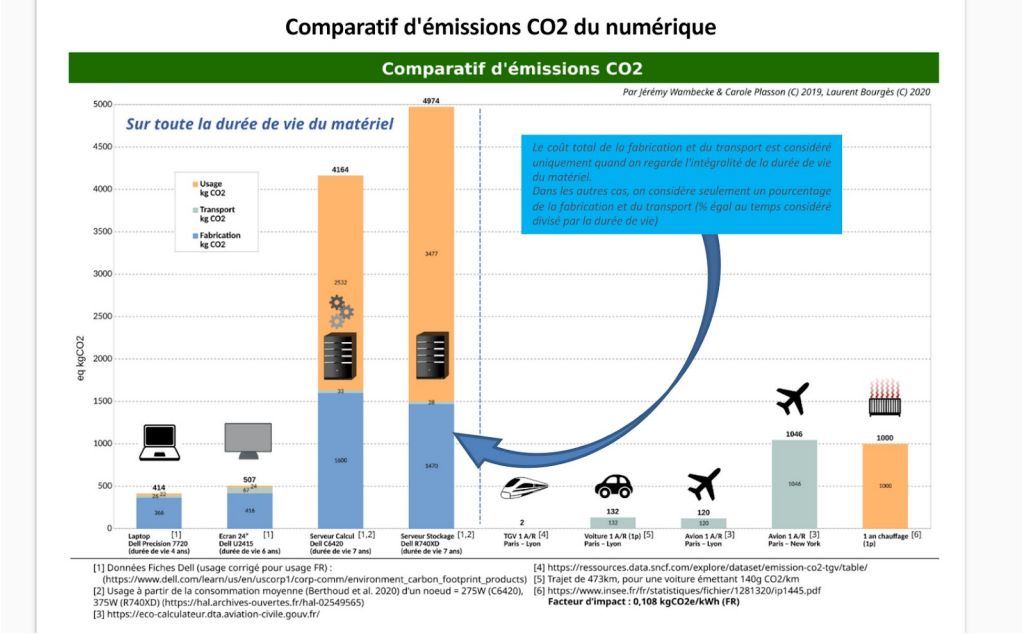


Figure 3[Ecoinfo]

Cependant, ces outils viennent souvent avec un défaut majeur : les performances. En effets, des outils comme React Native, Electron.js ou Ionic ont des performances bien inférieures à celle que l'on attendrait d'application native, c'est-à-dire écrite spécialement pour une plateforme. Cela relativise largement cette solution à l'obsolescence logicielle : en effet, si ces outils cross-plateforms permettent de supporter des vieux appareils, mais que ceux-ci n'ont pas les ressources nécessaires pour exécuter les applications qu'ils produisent, alors ils sont inutiles pour résoudre ce problème. C'est le cas des outils cross-plateform qui utilisent la stratégie des web-renderers (ionic, electron) et exécute des sites webs comme s'ils étaient des applications. L'illusion fonctionne sur les appareils modernes, mais les vieux modèles exécutent généralement ce genre d'applications au ralenti. Cependant, d'autres stratégies, comme celle de Flutter, permettent d'obtenir des performances décentes, et donc de supporter des vieux appareils sans problème. Il est alors intéressant d'utiliser ce genre d'outils pour prolonger la durée de vie des appareils.

Enfin, l'article sur l'impact de la consommation énergétique de l'astrophysique [Astrophysics] conclue par une réflexion sur les efforts qui doivent être fournis par les créateurs de logiciel. Ainsi, dans le cas des langages interprétés, même si ces langages sont très simples d'utilisation et permettent de faciliter le développement, ce n'est pas une raison pour les utiliser dans toutes les situations. Il est de la responsabilité des développeurs de faire des efforts en matière de consommation énergétique, quitte à allonger le développement. Ces efforts supplémentaires peuvent d'ailleurs être mis à profit pour l'ensemble de la communauté grâce à travers une philosophie du libre. Ecoinfo[Ecoinfo] place cette

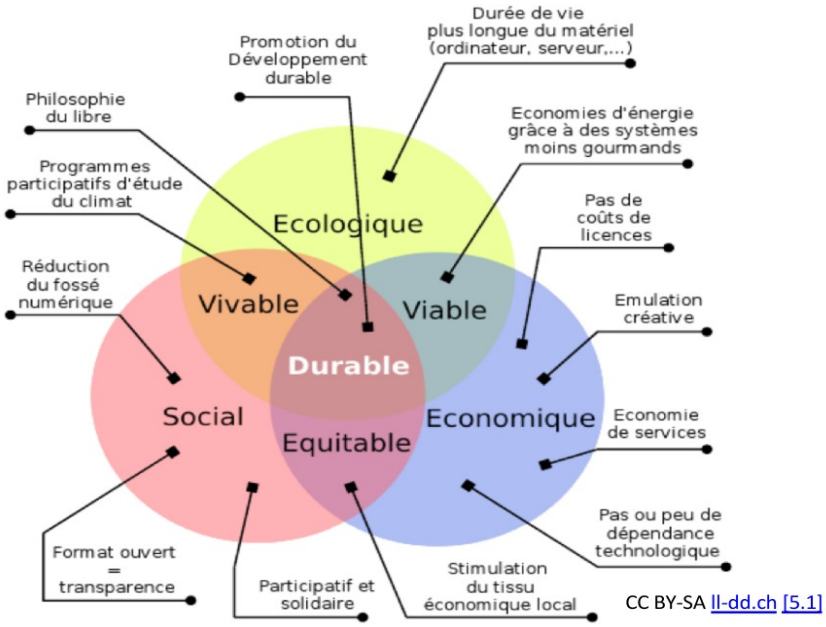


Figure 4[Ecoinfo]

philosophie du libre à la croisée des enjeux économiques, sociaux et écologiques dans la figure 4.

3 Pendant

3.1 Implémentation

Une fois que les fonctionnalités et les technologies ont été choisies, il est important de faire attention à l'implémentation. Ainsi, implémenter une fonctionnalité dans un logiciel avec une technologie peut être fait de différentes manières, plus ou moins efficace énergétiquement.

L'article "The Ecological Impact of High-performance Computing in Astrophysics"[Astrophysics] étudie par exemple comment la sous-utilisation ou la surutilisation des cœurs de calculs qui peut impacter l'efficacité énergétique d'un programme. Dans un processeur d'ordinateur, les cœurs de calculs permettent d'effectuer plusieurs calculs en même temps (à chaque instant, un cœur peut effectuer un calcul, plus il y a de cœurs, plus on peut faire de calculs en parallèle). Cela permet de réduire significativement le temps de calculs, mais nécessite plus de complexité dans la création du logiciel. L'article s'interroge sur l'impact énergétique de l'utilisation des cœurs de calculs. Les résultats sont exposés dans la figure 4.

Les résultats montrent que faire l'effort d'implémentation lors de la création de logiciel, et ainsi de passer d'un calcul à la fois dans un seul cœur à plusieurs calculs dans plusieurs cœurs, permet de réduire la consommation énergétique. Cela reste vrai jusqu'à 64 cœurs. Au-delà, l'utilisation de plus de cœurs engendre une surconsommation d'énergie : pour une même tache, on prend moins de temps, mais on consomme plus d'énergie.

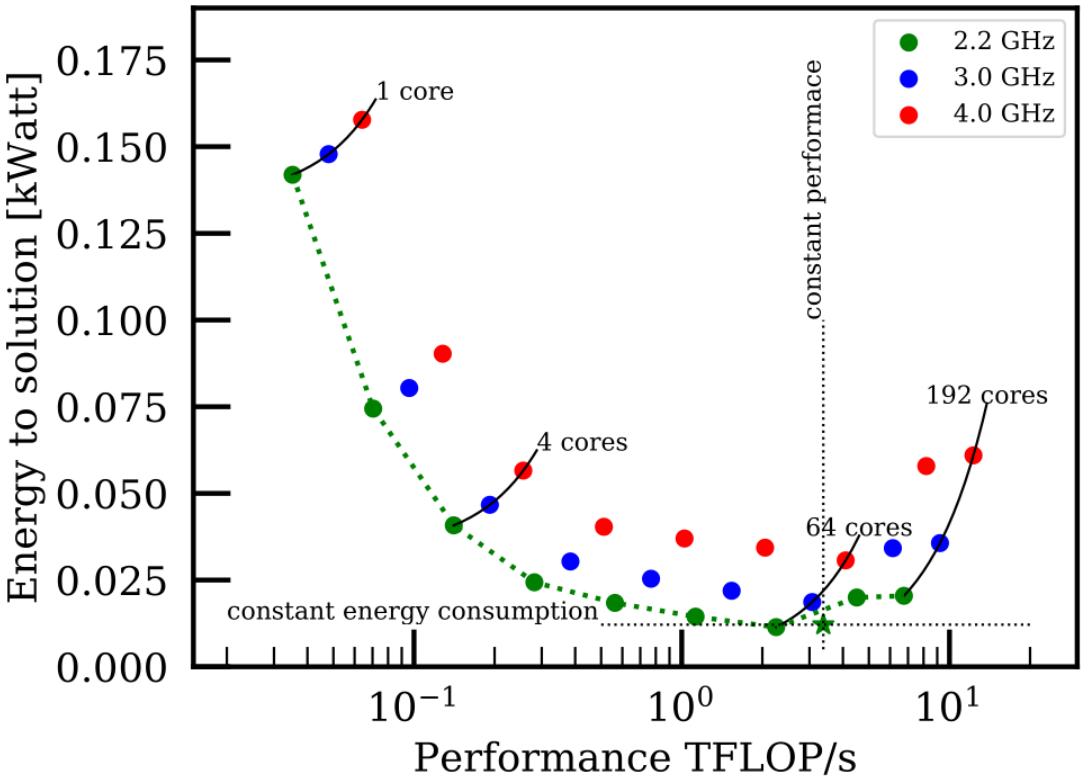
L'interprétation faite par les auteurs est la suivante : aujourd'hui, les processeurs d'ordinateurs ont pour la plupart plusieurs cœurs. Passer de 1 à 4 cœurs ne change rien à la consommation d'énergie du processeur, mais permet de l'utiliser moins longtemps. Cela reste vrai pour 8, 16 et 32 cœurs (certains processeurs très puissants peuvent atteindre ce nombre de cœurs). Cependant, pour effectuer plus de 32 calculs à la fois, on utilise non pas des processeurs, mais des cartes graphiques, des composants spécialement conçus pour disposer de centaines de cœurs. Ces composants sont bien plus gourmands en énergie pour un même

calcul. En conséquence, au-delà de 64 cœurs, la consommation d'énergie pour une même tache augmente.

Implémenter une fonctionnalité peut être fait avec plusieurs algorithmes différents. Un algorithme est une suite d'instructions à réaliser sur des données pour obtenir un résultat. Il peut en exister plusieurs pour une même tache, plus ou moins simple à coder et consommateurs d'énergie. Pour une même tache, certains algorithmes nécessitent plus de calculs et donc plus d'énergie et plus de temps. Dans l'article "Green Algorithms: Quantifying the Carbon Footprint of Computation"[Green algorithms], les auteurs montre comment ils ont mis au point une méthode pour mesurer les émissions de CO2 d'un algorithme. Cette méthode est intégrée dans un outil en ligne, http://calculator.green-algorithms.org/.

Leur méthode prend en comptes de nombreux facteurs :

- La durée des calculs
- Le type de cœurs (soit des cœurs de processeur, soit des cœurs de cartes graphiques)
- Le nombre de cœurs
- Le modèle de la carte graphique ou du processeur.
- Espace mémoire disponible
- La plateforme sur laquelle l'algorithme est utilisé, à savoir soit un datacenter, soit un ordinateur personnel. Dans les deux cas, on cherche à connaître la quantité de CO2 émise pour produire l'électricité utilisée par cette plateforme, donc dans le cas du datacenter, on demande à l'utilisateur quelle entreprise met en location son datacenter, et dans le cas de l'ordinateur personnel, on demande dans quel pays l'utilisateur se situe. Cela permet d'obtenir des informations sur le mix énergétique utilisé pour alimenter la plateforme.
- Le Primary Scale Factor, une estimation du nombre de fois que l'algorithme va être utilisé. Par exemple, certains algorithmes dans le domaine de l'intelligence artificiels sont voués à êtres utilisés de manière répétée des milliers de fois, là où d'autres algorithmes remplissent la tâche pour laquelle ils sont conçus en une seule exécution.


Energy to solution [kWatt]
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
10^{-1}
10^{0}
10^{1}
Performance TFLOP/s
2.2 GHz
3.0 GHz
4.0 GHz
1 core
4 cores
64 cores
192 cores
constant performace
constant energy consumption

Cet outil permet de rapidement et simplement choisir entre deux algorithmes pour une même tâche. Une manière encore plus simple est de facilement comparé leur “complexité”. La complexité est un indicateur de performance des algorithmes qui correspond à la relation entre la taille des données d’entrée et le nombre d’opérations a effectué pour obtenir le résultat. Par exemple, un algorithme avec une complexité de $O(n^2)$ va faire n^2 opérations pour obtenir un résultat avec n données d’entrée. Si la taille des données augmente, le nombre d’opérations augmente aussi avec une croissance très rapide qui correspond à la fonction carrée. Choisir un algorithme avec une faible complexité permet de réduire le nombre d’opérations et donc la consommation énergétique pour effectuer une tâche.

Il est cependant important de prendre en compte les émissions d’ordre 3 (l’effet rebond). En effet, l’utilisation de très bons algorithmes, avec de très bonnes performances, peut engendrer non pas une réduction de la consommation énergétique, mais une augmentation de l’utilisation de la fonctionnalité implémentée. En conséquence, on observe une augmentation de la consommation énergétique.

3.2 Développement :

Lors du développement de logiciel, des outils peuvent assister le programmeur à réduire la consommation énergétique. C’est un tel outil qui est décri dans l’article “SPELLing Out Energy Leaks Aiding Developers Locate Energy Inefficient Code”[SPELL]. Dans cet article, les auteurs décrivent un outil appelé SPELL qui permet d’identifier les endroits dans le code d’un logiciel qui consomment de l’énergie.

Les développeurs écrivent des tests pour vérifier que leur logiciel fonctionne comme attendu. Ainsi, on écrit généralement du code qui vérifie que le reste du code fonctionne bien. C’est une manière puissante d’éviter les bugs et, grâce à SPELL, d’identifier et neutraliser les parties du code qui consomment de l’énergie. Ainsi, SPELL mesure l’énergie consommée par l’ordinateur lorsque chaque test est effectué (grâce aux données RAPL notamment). Grâce à cela, il est capable d’identifier quel test a consommé le plus d’énergie. En comparant le bouts de code exécuté dans chaque test, il est capable de trouver les bouts de codes qui se trouvent dans les tests les plus énergivores, et de les afficher en rouge au développeur. De cette manière, le développeur peut visuellement identifier les lignes énergivores et agir pour optimiser son code.

D'après l'étude, cet outil permet au développeur d'être plus efficace pour optimiser la consommation énergétique du logiciel. En utilisant l'outil, des développeurs cobayes ont pu réduire la consommation énergétique de leur logiciel de 43 % en moyennes.

D'autres outils existent et sont recommandés dans la plaquette d'Ecoinfo[Ecoinfo] (Sonarqube, codacy, gcov, gprof, perf, valgrind).

Ces outils sont utiles pour réduire la consommation énergétique du logiciel pendant son développement. La plaquette d'Ecoinfo[Ecoinfo] recommande aussi des bonnes pratiques pour réduire la consommation énergétique du développement lui-même. Ces pratiques évitent de surconsommer de l'énergie en développant le logiciel :

- Utiliser un gestionnaire de versions, mais en faisant attention à ne pas le surcharger de donner et notamment en évitant d'y stocker ni les binaires des applications, ni les fichiers issus de compilation, ni les fichiers de sorties.
- L'usage de l'intégration continue doit être frugale, avec des images dockers légères. Elle ne doit être activée que sur certaines branches pour éviter des calculs superflues à chaque petit changement dans le code. Il faut surveiller la consommation de l'intégration continue, qui peut être utilisée par des dizaines voir des centaines de personnes chaque jour.
- Privilégié les forges mutualisées aux ordinateurs personnels pour les calculs intensifs.

4 Après

La plaquette d'Ecoinfo[Ecoinfo] recommande des bonnes pratiques lors du déploiement de l'application, puis lors de sa maintenance. La plupart consistent en des améliorations matérielles, des choix d'infrastructure (quel datacenter choisir, quel label certifie la bonne utilisation de l'énergie dans ces datacenter…) qui ne nous intéressent pas dans ce document.

Cependant, certaines bonnes pratiques logicielle peuvent permettre de réduire la consommation énergétique.

4.1 Virtualisation

Pour être exécuté correctement et de manière sécurisée, un logiciel dans un datacenter a besoin d'être isolé. La méthode par défaut est d'allouer un ordinateur par logiciel. Cependant, de nombreux logiciels ont besoin de très peu de performance, et ce serait un gaspillage de ressource de les mettre seuls sur un ordinateur surdimensionné. C'est pourquoi on a inventé la virtualisation, qui permet de simuler un ordinateur et son système d'exploitation dans un autre ordinateur, et donc de mettre plusieurs ordinateurs virtuels isolés et sécurisés (en théorie) sur le même ordinateur physique. De cette manière, on évite d'allouer plus de ressources que nécessaire, et on peut même adapter l'allocation dynamiquement. Par exemple, si un site web a ponctuellement besoin de beaucoup plus de ressource qu'en moyenne, au lieu de lui allouer plus de ressources que ce dont il a besoin en permanence, on peut lui en allouer très peu la plupart du temps, et soudain lui en allouer beaucoup plus lors d'un pique d'utilisation. La virtualisation permet donc de limiter les besoins en infrastructure des datacenters, et donc d'éviter des émissions de gaz à effets de serres pour la production de ces infrastructures, puis pour leur alimentation.

4.2 Supervision

De nombreux outils de supervisions permettent de mesurer la manière dont se comporte notre logiciel. Il est donc important de mettre en place ces outils, et d'exploiter les résultats, pour faire de l'amélioration continue.

Il est important, dans le cas de virtualisation, d'éteindre les machines virtuelles non utilisées. De nombreux outils comme Kubernetes permettent de créer des infrastructures horizontales qui peuvent rapidement s'adapter à la demande et éviter des coûts énergétiques non nécessaires lorsque la demande est faible.

4.3 Distribution

Pour une bonne diffusion de mon logiciel, il est important de le déposer à un endroit facilement accessible. Des mises à jour doivent être réalisée régulièrement, mais avec une fréquence faible, pour éviter trop de transfert de données. Pour cela, on peut agréger plusieurs mises à jour en une seule. La règle d'or en matière de mise à jour est la rétrocompatibilité : le support des vieux appareils prolonge leur durée de vie et évite leur renouvellement, or, on a déjà vu que la grande

majorité des émissions sont due à la production et non pas l'utilisation des appareils.

5 Conclusion

Le secteur du logiciel peut être très impactants énergétiquement, notamment dans le cas de calculs intenses (calculs scientifiques, datacenters, IA...). De plus, le logiciel à une grande influence sur la durée de vie des appareils numériques, or leur production est très polluante. Il existe donc des mesures que l'on peut prendre à différentes étapes de la création d'un logiciel pour limiter son impact sur l'environnement.

Le numérique est une opportunité de réduire notre impact sur l'environnement sans baisser, voir en améliorant, notre niveau de vie. Ainsi, les visioconférences, les e-mails ou encore les cours à distances sont autant de moyens de diminuer les flux physiques qui sont la cause de nombreux impacts environnementaux.

Cependant, la surutilisation du numérique est, elle aussi, polluante. Il tient à nous de rationaliser son usage. Ainsi, de nombreuses consommations d'énergies ne sont pas venues remplacer des flux physiques, mais se sont ajoutés à eux : les réseaux sociaux, pensés pour être addictifs, font augmenter l'impact du numérique sans remplacer des flux physiques. Un usage sobre du numérique consisterait à limiter les usages superflus, pour ne conserver que ceux qui permettent de réellement réduire les impacts sur l'environnement.

Bibliographie

SPELL: Rui Pereirab,1, Tiago Car¸c˜aob, Marco Coutob, J´acome Cunhac, Jo˜ao Paulo Fernandesd, Jo˜ao Saraivab, SPELLing Out Energy Leaks Aiding Developers Locate Energy Inefficient Code,
Android Keyboard: Rui Rua and al., Greenspecting Android Virtual Keyboard,
Green algorithms: Loïc Lannelongue, Jason Grealey, Michael Inouye, Green Algorithms: Quantifying the Carbon Footprint of Computation,
Ecoinfo: Cyrille Bonamy, Cédric Boudinet, Laurent Bourgès, Karin Dassas, Laurent Lefèvre, Francis Vivat, Je code : les bonnes pratiques en éco-conception de service numérique à destination des développeurs de logiciels, 2020
Astrophysics: Simon Portegies Zwart, The Ecological Impact of High-performance Computing in Astrophysics,