GRENOBLE-INP ENSIMAG

INGÉNIERIE DES SYSTÈMES D'INFORMATION

2A

GRENOBLE-INP ENSIMAG

INFORMATION SYSTEMS ENGINEERING

2A

ÉCOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET DE MATHÉMATIQUES APPLIQUÉES DE GRENOBLE

PROJET GL

DOCUMENTATION D'ANALYSE DES IMPACTS ENERGETIQUES

Réalisé par :

DIAKITE Alpha Ousmane
DREUILLE Nathan
FLISS Yassine
KOSSI Yao David
OUDRHIRI IDRISSI Safwane

CLIENT :

FREIRE MARCO

Année académique : 2025–2026

Table des matières

1 Introduction – Enjeux écologiques dans le projet GL

La question de l'impact environnemental du numérique est devenue un sujet important ces dernières années. Si l'attention se porte souvent sur les infrastructures matérielles, les logiciels ont eux aussi un impact énergétique réel, lié à leur conception et à leur usage. Chaque exécution consomme des ressources, et à grande échelle, ces coûts peuvent devenir significatifs.

Les projets de génie logiciel, et en particulier ceux portant sur des compilateurs, sont directement concernés par ces enjeux. Un compilateur est un outil destiné à être utilisé de manière intensive. Des compilateurs largement répandus, comme Javac, GCC ou Clang utilisés pour Java ou pour le langage C/C++, sont exécutés un très grand nombre de fois chaque jour, aussi bien par des entreprises qui produisent enormement de code mais aussi par de nombreux particuliers. À cette échelle, un surcoût énergétique faible pour une compilation isolée peut devenir important lorsqu'il est multiplié par un grand nombre d'occurrences.

L'impact énergétique d'un compilateur ne se limite pas à son utilisation finale. Il est également présent tout au long de son cycle de production, en particulier lors des phases de développement et de validation. Dans le cadre du projet GL, ces phases se traduisent par de nombreuses compilations successives et par l'exécution répétée de suites de tests automatisés. Certaines pratiques courantes, comme lancer systématiquement l'ensemble des tests alors qu'une simple compilation aurait suffi, peuvent ainsi conduire à une consommation énergétique inutile lorsqu'elles sont répétées fréquemment.

L'objectif de ce document est d'analyser l'impact énergétique du compilateur Deca développé dans le cadre du projet GL. Cette analyse repose sur des mesures indirectes et des observations expérimentales, et vise à mieux comprendre où se situent les principaux postes de consommation énergétique, aussi bien lors de la compilation que lors de l'exécution des programmes générés.

2 Protocole utilisé

Afin de mener une étude sur l'impact énergétique de notre compilateur, nous avons choisi de relever différentes mesures sur l'ensemble des tests effectués depuis le début du projet GL (en prenant bien sur uniquement les tests deca valides sinon il est un peu plus compliqué de générer un fichier assembleur...).

Cette approche nous permet d'obtenir à la fois une estimation des ressources nécessaires à la compilation de programmes avec notre compilateur `decac`, ainsi qu'une vision plus globale du coût énergétique associé à l'exécution complète de la chaîne de validation. En particulier, elle permet de mieux appréhender ce que représente, en termes de res-

sources, l'exécution répétée de commandes telles que `mvn test`, largement utilisées tout au long du projet.

Pour chaque test Deca analysé, nous avons décidé de relever quatre types de données :

- **Le temps CPU**, qui donne une indication du temps pendant lequel le processeur est effectivement sollicité pour la compilation d'un programme.
- **Le temps total du processus de compilation**, qui correspond au temps réel nécessaire pour compiler un programme, en tenant compte de l'ensemble des ressources mobilisées sur la machine.
- **Le nombre d'instructions exécutées par le processeur virtuel** `ima`, qui constitue une métrique centrale pour la suite de l'étude. Cette donnée sera notamment utilisée pour mettre en perspective l'énergie nécessaire à l'exécution en la rapportant à des ordres de grandeur plus parlants, issus de situations de la vie quotidienne.
- **L'utilisation de la mémoire**, enfin, qui fera l'objet d'une analyse spécifique. Comme nous le verrons par la suite, cette métrique met en évidence des résultats intéressants qui n'étaient pas nécessairement attendus avant l'analyse.

L'ensemble de ces mesures constitue la base du protocole expérimental utilisé dans cette étude et servira de support à l'analyse et à l'interprétation des résultats présentées dans les sections suivantes.

3 Résultats et interprétations

3.1 Efficacité du processus : Comparaison CPU Time vs User Time

Pour évaluer l'efficacité énergétique de notre compilateur, nous avons d'abord comparé le temps passé à exécuter le code du programme (*User Time*) au temps total consommé par le processeur (*CPU Time*).

Le *CPU Time* inclut non seulement le temps de calcul, mais aussi le temps passé par le système d'exploitation pour gérer le programme (appels systèmes, gestion de la mémoire, entrées/sorties). Dans un scénario idéal d'efficacité énergétique, le temps système (l'écart entre les deux mesures) doit être nul.

La Figure 1 présente ces mesures pour l'ensemble de la base de tests. La ligne rouge en pointillés représente l'idéal théorique ($y = x$).

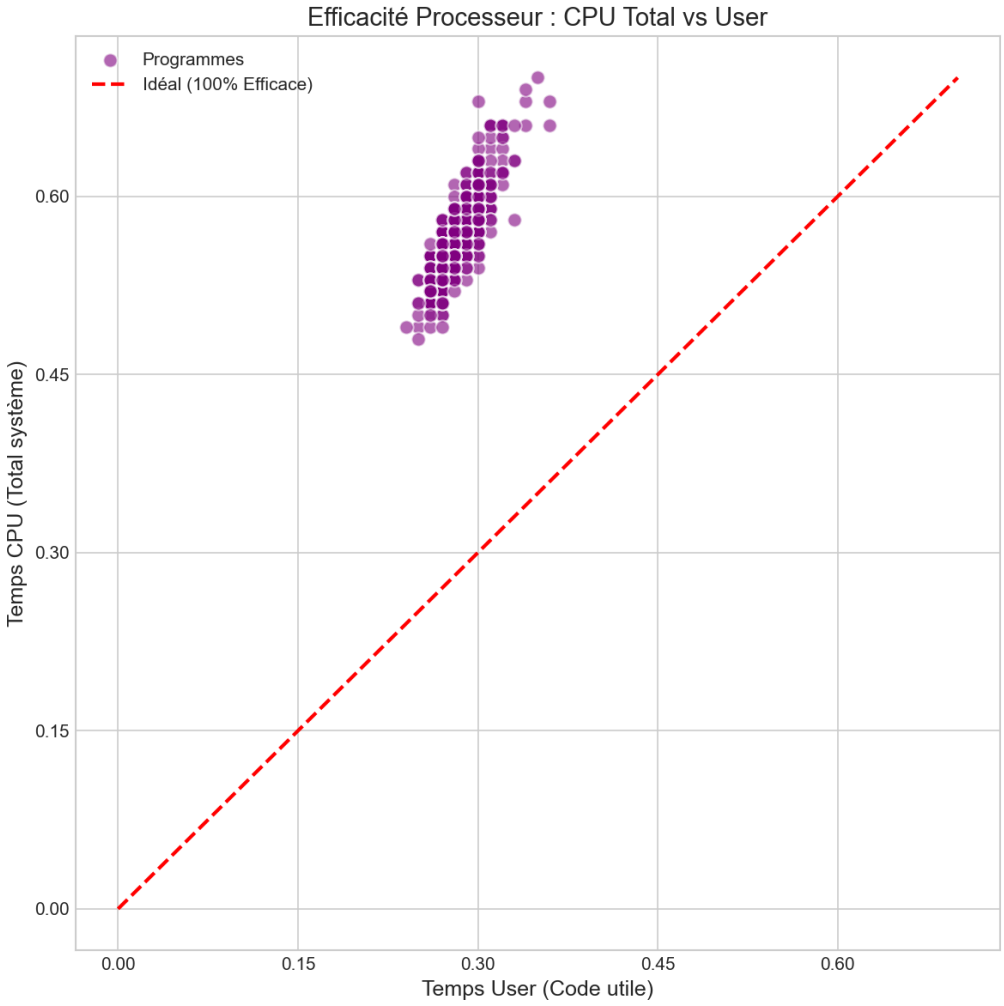

FIGURE 1 – Corrélation entre le Temps CPU Total et le Temps Utilisateur. La proximité des points avec la diagonale témoigne de l'efficacité du code.

Interprétation et Bilan

L'analyse du graphique révèle une forte corrélation linéaire : la quasi-totalité à proximité , de la diagonale.

Cela indique que le surcoût lié au système d'exploitation est négligeable. Concrètement, cela signifie que lorsque le compilateur `decac` tourne, l'énergie consommée par le processeur est convertie en travail utile (analyse syntaxique, génération de code) et n'est pas gaspillée dans des attentes ou des appels systèmes inefficaces. C'est un indicateur que le programme est autonome et ne surcharge pas le noyau du système d'exploitation.

3.2 Analyse croisée : Indépendance entre empreinte mémoire et temps d'exécution

Un point crucial de l'efficacité d'un langage compilé est sa capacité à gérer de larges volumes de données sans que le coût de gestion de cette mémoire (allocation/désallocation) ne pénalise la vitesse d'exécution.

La Figure 2 illustre la relation entre l'espace mémoire occupé (axe vertical) et le temps utilisateur nécessaire à l'exécution (axe horizontal).

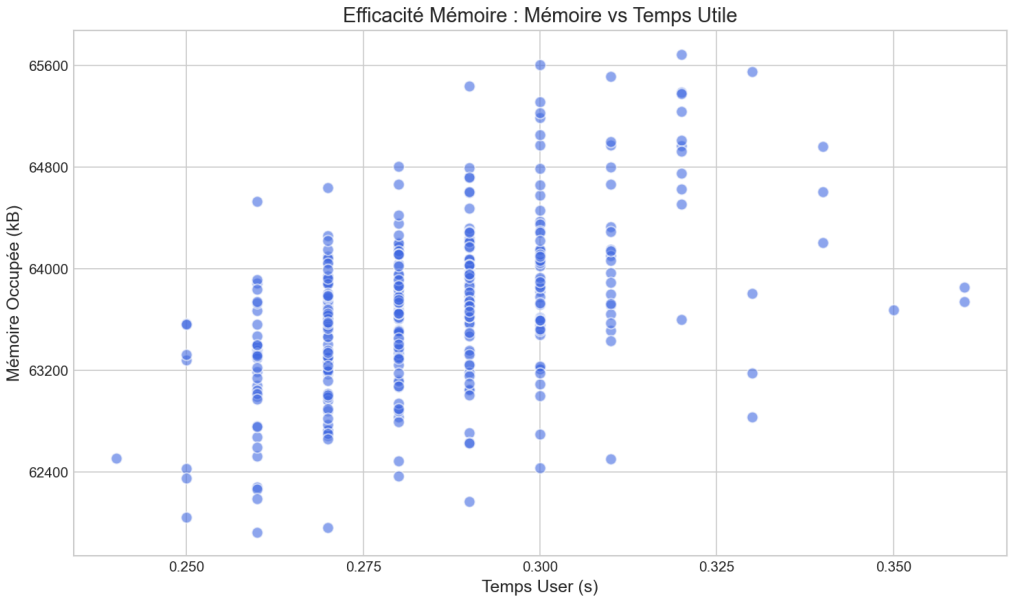

FIGURE 2 – Relation Mémoire occupée vs Temps d'exécution. La dispersion des points indique une absence de corrélation forte entre la taille des données et la durée du calcul.

Interprétation et Bilan

L'analyse de ce nuage de points met en évidence qu'il n'existe pas de corrélation linéaire forte entre la consommation mémoire et le temps d'exécution.

Contrairement à ce que l'on pourrait penser, les programmes consommant davantage de mémoire (points situés haut sur l'axe Y) ne sont pas systématiquement plus lents (ils ne sont pas déportés vers la droite sur l'axe X).

D'un point de vue énergétique, cette "non-influence" est un indicateur positif de la qualité du code généré par `decac`. Elle suggère que :

1. Le coût unitaire des allocations mémoire (le temps CPU nécessaire pour réserver de l'espace dans la pile ou le tas) est suffisamment faible pour être négligeable face au temps de calcul algorithmique.

2. L'augmentation de la taille des données traitées n'entraîne pas de surcharge exponentielle du processeur.

3.3 Distribution des Instructions

L'impact écologique d'un compilateur ne se mesure pas uniquement à sa propre consommation, mais surtout à l'efficacité du code qu'il produit. Un code assembleur trop long pour rien forcera le processeur à effectuer des cycles d'horloge inutiles à chaque exécution du programme final, multipliant ainsi le gaspillage énergétique.

La Figure 3 ci-dessous montre la répartition du nombre d'instructions générées pour l'ensemble des tests de la suite de validation.

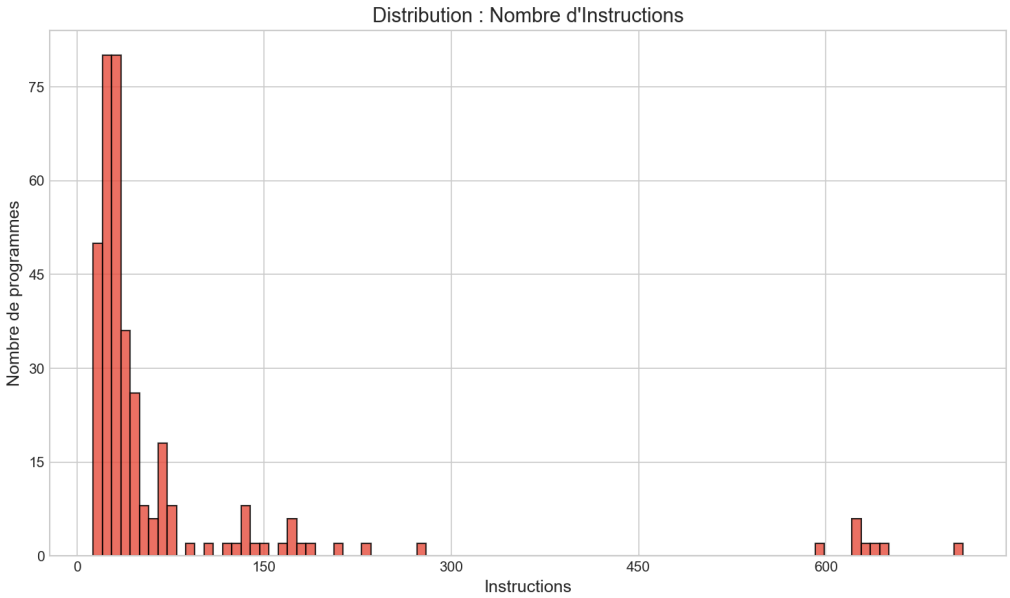

FIGURE 3 – Distribution du nombre d'instructions générées.

Interprétation et Bilan

L'analyse de cet histogramme nous permet de qualifier la "densité" du code produit par `decac`.

— **Concentration vers les valeurs faibles :** Nous observons un pic dominant sur la gauche du graphique. Cela indique que pour la majorité des cas, le compilateur génère un nombre d'instructions assez faible (nos tests ne sont pas non plus très complexes). C'est un indicateur que le compilateur ne génère pas de code inutile .

— **Analyse queue de distribution :** L'étalement vers la droite représente les programmes complexes générant un grand nombre d'instructions. D'un point de vue

énergétique, ces cas sont les plus coûteux. Cette queue est fine, confirmant que les explosions de complexité sont rares.

3.4 Analyse des distributions temporelles (CPU vs User)

Puisque nous avons établi une forte corrélation entre le temps processeur total et le temps utilisateur, il est pertinent d'analyser leurs distributions conjointement.

Ces histogrammes représentent la durée d'exécution des tests en secondes. L'objectif ici est de vérifier si le compilateur est capable de traiter la majorité des fichiers sources rapidement, ou si une proportion importante des tests entraîne des temps de calcul longs.

La Figure 4 ci-dessous met en regard la distribution du Temps CPU Total (à gauche) et celle du Temps Utilisateur (à droite).

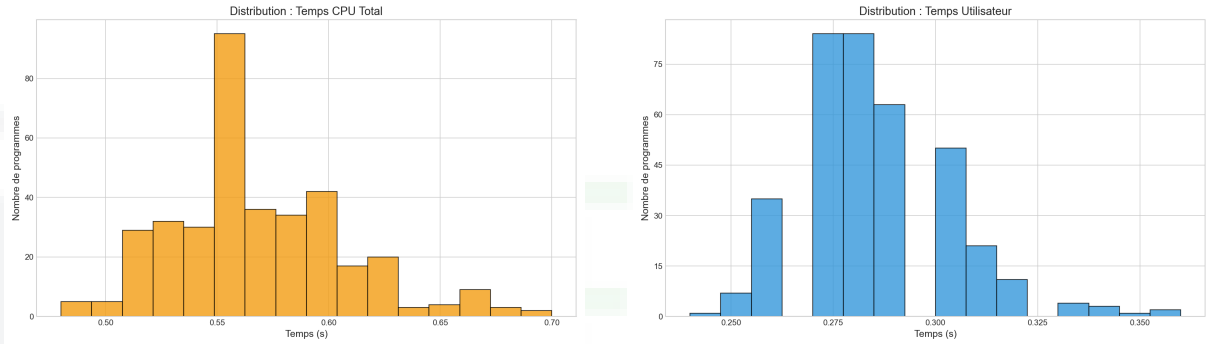

FIGURE 4 – Comparaison des distributions temporelles (en secondes).

Interprétation et Bilan

L'analyse de ces durées d'exécution appelle deux observations majeures sur l'efficacité de notre solution :

— **Similitude des profils :** Les deux histogrammes sont visuellement très proches. Cela confirme que le temps passé par le système d'exploitation (pour gérer les fichiers ou la mémoire) reste marginal par rapport au temps de calcul pur, et ce, quelle que soit la durée du test. Le compilateur est donc "autonome" et ne subit pas de ralentissements externes significatifs.

— **Prédominance des exécutions courtes :** Nous observons une distribution très marquée vers la gauche (proche de 0 seconde). Cela signifie que la grande majorité des programmes de test sont compilés quasi-instantanément.

Conclusion : Cette concentration des résultats vers des temps très courts est un indicateur de performance positif. En libérant le processeur le plus vite possible, le compilateur minimise son empreinte sur les ressources partagées de la machine, garantissant une disponibilité maximale pour les autres processus.

4 Estimation énergétique à partir du nombre d'instructions

Afin de donner un ordre de grandeur de l'impact énergétique lié à l'exécution des programmes générés, nous avons cherché à estimer l'énergie associée au nombre d'instructions exécutées par le processeur virtuel `ima`. Il est important de noter que `ima` étant un processeur virtuel, il n'est pas possible d'associer directement une consommation électrique réelle à l'exécution de ses instructions. L'objectif de cette estimation n'est donc pas d'obtenir une valeur précise, mais de fournir un ordre de grandeur permettant de mieux interpréter les résultats obtenus.

L'énergie totale consommée lors de l'exécution peut être approximée par la relation suivante :

$$E = N \times E_{instr}$$

où N correspond au nombre d'instructions exécutées et E_{instr} à l'énergie moyenne consommée par instruction. D'après les ordres de grandeur généralement admis pour des processeurs modernes, l'énergie consommée par instruction se situe dans une plage allant de quelques nanojoules. Dans le cadre de cette étude, nous avons retenu une valeur moyenne (le choix de cette approximation est détaillé dans la documentation en fin de rapport) :

$$E_{instr} = 5\,nJ$$

Cette hypothèse permet d'estimer l'énergie associée à l'exécution d'un programme en fonction du nombre d'instructions `ima` exécutées. Sur l'ensemble de notre batterie de tests, nous avons mesuré un total de **13 377 instructions IMA**. En appliquant l'approximation précédente, cela correspond à une énergie de l'ordre de quelques dizaines de microjoules, ce qui reste très faible à l'échelle d'une exécution isolée.

Cependant, ce résultat doit être interprété avec prudence. Si un tel coût est négligeable pour une exécution unique, il peut devenir significatif lorsque l'on considère un usage à grande échelle. Dans un contexte industriel, un compilateur est susceptible d'être exécuté des milliers, voire des millions de fois par jour, notamment dans des chaînes d'intégration continue ou des environnements de développement intensifs.

Enfin, il est important de souligner que cette estimation ne prend en compte que l'énergie associée à l'exécution des instructions `ima`. Elle ne couvre pas l'ensemble de l'énergie réellement consommée par l'ordinateur lors de l'utilisation du compilateur, comme le rendu graphique, l'activité du système d'exploitation ou encore le fonctionnement des périphériques et des autres processus s'exécutant en parallèle. Les valeurs obtenues doivent donc être vues comme une estimation partielle, destinée à fournir un ordre de grandeur

et non une mesure exhaustive de la consommation énergétique globale.

À titre de comparaison, une énergie de l'ordre de quelques dizaines de microjoules correspond par exemple à l'alimentation d'une LED pendant seulement quelques millisecondes, ou à quelques microsecondes d'activité d'un processeur moderne. Ce coût est donc négligeable à l'échelle d'une exécution isolée, mais peut devenir significatif lorsqu'il est multiplié par un très grand nombre d'exécutions dans un contexte d'utilisation intensive.

5 Utilisation des assistants IA et Analyse d'Impact

Dans le cadre du projet, nous avons intégré l'utilisation d'assistants basés sur les grands modèles de langage (LLM). Cette démarche a été documentée afin d'en évaluer non seulement l'apport technique, mais aussi le coût environnemental, dans une optique de transparence et de sobriété numérique.

5.1 Cas d'usage et méthodologie

L'IA a été utilisée comme un outil de support ciblé sur des tâches spécifiques, et non comme un substitut au travail de conception :

- **Compréhension et Débogage :** Aide à l'interprétation des spécifications du polycopié et analyse des erreurs de compilation complexes (ex : mécanismes de réduction de l'argument pour les fonction TRIGO).
- **Validation et Tests :**
  - Génération de squelettes de tests `JUnit` pour augmenter la couverture de code.
  - Pour la partie génération de code, création de petits programmes et calcul des valeurs attendues , notamment pour vérifier les résultats des opérations flottantes.
- **Productivité et Outils annexes :**
  - Auto-complétion de code (patterns répétitifs).
  - Aide à la rédaction LaTeX pour la structuration du rapport.
  - Génération de UML pour les diagrammes d'architecture.
- **Recherche documentaire (Extension Trigo) :** Suggestion de bibliographie et résumé des méthodes d'approximation (Minimax, Taylor, Coddy-Whaite) pour l'implémentation des fonctions trigonométriques.

5.2 Bilan énergétique : La dualité de l'IA

L'intégration de l'IA dans un projet visant le numérique responsable soulève un paradoxe qu'il est nécessaire d'analyser.

Le coût caché de l'inférence

Contrairement à une recherche par mot-clé classique, chaque requête adressée à un LLM entraîne une consommation énergétique significative côté serveur (inférence sur GPU). Des études estiment qu'une requête générative peut consommer 10 à 50 fois plus qu'une requête web standard. Nous avons donc, en partie, "externalisé" une dépense énergétique vers les data centers.

Le gain en efficacité locale

Cependant, ce coût "investi" doit être mis en balance avec l'économie réalisée localement :

1. **Réduction du temps de développement :** En résolvant des bugs bloquants plus vite et en automatisant l'écriture de tests répétitifs, nous avons réduit la durée d'activation de nos postes de travail.
2. **Optimisation du code final :** L'IA a permis d'identifier rapidement des structures de code inefficaces. L'énergie dépensée pour une requête ponctuelle est "rentabilisée" si elle permet de produire un compilateur qui, lui, sera exécuté des milliers de fois de manière plus optimale.

Conclusion : L'impact de l'IA sur ce projet est un compromis. Pour limiter son empreinte, nous avons adopté une pratique de "prompting responsable" : privilégier des requêtes précises et groupées plutôt qu'une conversation exploratoire longue, afin de maximiser le rapport *utilité / coût énergétique*.

6 Conclusion

L'étude menée met en évidence la difficulté, avec les outils dont nous disposons dans le cadre du projet GL, de réaliser une analyse extrêmement précise et rigoureuse de l'impact énergétique d'un projet de génie logiciel tel qu'un compilateur. L'absence de mesures directes de consommation électrique et l'utilisation de métriques indirectes imposent en effet un certain nombre d'hypothèses et d'approximations, qui limitent les résultats.

Néanmoins, au-delà des chiffres et des estimations proposées, ce travail met en lumière un point essentiel : la consommation énergétique associée à un compilateur est bien réelle.

Même si elle reste faible à l'échelle d'une exécution isolée, elle devient significative lorsque l'on considère un usage répété.

Cette étude souligne ainsi l'importance de prendre en compte l'impact énergétique comme un paramètre à part entière dans la réalisation de projets de génie logiciel. Sans chercher nécessairement une optimisation systématique, le simple fait d'avoir conscience de ces enjeux permet d'adopter des pratiques plus sobres, notamment lors des phases de développement, de compilation et de validation.

7 Références

@articleShankar2022EPI, title=Trends in Energy Estimates for Computing in AI/Machine ..., author=Shankar, S., year=2022, note=arXiv preprint, url=https ://arxiv.org/pdf/2210.17331

@miscEnergyPerInstr64bit, title=Energy Per Instruction for different classes of instructions, howpublished=ResearchGate, note=Mesures d'EPI pour processeur 64-bits, url=https ://www.researchgate.net/figure/Energy-Per-Instruction-for-different-classes-of-instructions_fig1_349392055