

Évaluation de la consommation énergétique des langages informatiques



Benoît Prieur - CC0

Peu d'études et à la qualité contrastée

- Protocole de comparaison ?
- Publication scientifique à comité de lecture ?
- Récente ?




Energy Efficiency across Programming Languages

How Does Energy, Time, and Memory Relate?

- Protocole très élaboré et défini de façon précise.
- *SLE 2017: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering.*
- Quelques mises à jour 2020 et 2021.
- DOI : <https://doi.org/10.1145/3136014.3136031>
- Papier :
<https://greenlab.di.uminho.pt/wp-content/uploads/2017/09/paperSLE.pdf>
- Site officiel : <https://sites.google.com/view/energy-efficiency-languages>
- <https://github.com/greensoftwarelab/Energy-Languages>
-



Le protocole : choix des langages étudiés

- *Computer Language Benchmarks Game (CLBG)* pour la sélection des langages. Existence exhaustive des problèmes pour chaque langage et de l'algorithme optimisé pour chacun. Les langages sont catégorisés par paradigme :
 - Langage impératif ;
 - Langage fonctionnel ;
 - Langage orienté objet ;
 - Langage de script.
- 

Le protocole : choix des langages étudiés (2)

- Un langage peut être dans plusieurs des catégories précitées.
- Autre manière de les classer : script, compilé et/ou basé sur une machine virtuelle.
- Vingt-sept langages sélectionnés.

Table 2. Languages sorted by paradigm

Paradigm	Languages
Functional	Erlang, F#, Haskell, Lisp, Ocaml, Perl, Racket, Ruby, Rust;
Imperative	Ada, C, C++, F#, Fortran, Go, Ocaml, Pascal, Rust;
Object-Oriented	Ada, C++, C#, Chapel, Dart , F#, Java, JavaScript, Ocaml, Perl, PHP, Python, Racket, Rust, Smalltalk, Swift, TypeScript;
Scripting	Dart, Hack, JavaScript, JRuby, Lua, Perl, PHP, Python, Ruby, TypeScript;



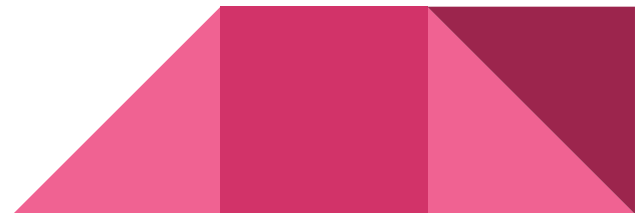
Le protocole : choix des langages selon disponibilité des problèmes/algorithmes optimisés (BCG)

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to tiny integer sequence	12
spectral-norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex-redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k-nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse-complement	Read DNA sequences, write their reverse-complement	fasta output
binary-trees	Allocate, traverse and deallocate many binary trees	21
chameneos-redux	Symmetrical thread rendezvous requests	6M
meteor-contest	Search for solutions to shape packing puzzle	2,098
thread-ring	Switch from thread to thread passing one token	50M

- 28 langages pris en compte dans le CLBG, exclusion de SmallTalk (compilateur propriétaire).
- Exclusion des problèmes non couverts pour certains langages.
- Finalement 13 problèmes.

Le protocole : la mesure

- Le benchmark est effectué sur une machine unique Linux Ubuntu Server 16.10, 16 Go de RAM, processeur Intel Core i5-4460, à 3.20 GHz.
- L'usage de la mémoire est mesuré par une fonction système.
- Tout comme la consommation énergétique, l'usage des fonctions système RAPL.



Le protocole : mesure énergétique

- La mesure énergétique donne un résultat en Joule (J).
- 1 Joule \Leftrightarrow 1 Watt pendant une seconde.
- Recherche Google : environ 1000 Joules.
- 1000 Joules \Leftrightarrow 0,0002 kWh.
- Consommation moyenne mensuelle d'un foyer en France : environ 400 kWh (ordre de grandeur).

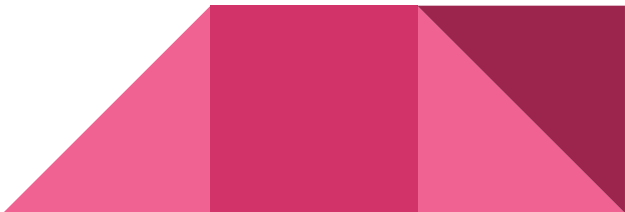


Résultats pour *BinaryTree*

binary-trees				
	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust ↓ ₂	49.07	1263	0.039	180
(c) Fortran ↑ ₁	69.82	2112	0.033	133
(c) Ada ↓ ₁	95.02	2822	0.034	197
(c) Ocaml ↓ ₁ ↑ ₂	100.74	3525	0.029	148
(v) Java ↑ ₁ ↓ ₁₆	111.84	3306	0.034	1120
(v) Lisp ↓ ₃ ↓ ₃	149.55	10570	0.014	373
(v) Racket ↓ ₄ ↓ ₆	155.81	11261	0.014	467
(i) Hack ↑ ₂ ↓ ₉	156.71	4497	0.035	502
(v) C# ↓ ₁ ↓ ₁	189.74	10797	0.018	427
(v) F# ↓ ₃ ↓ ₁	207.13	15637	0.013	432
(c) Pascal ↓ ₃ ↑ ₅	214.64	16079	0.013	256
(c) Chapel ↑ ₅ ↑ ₄	237.29	7265	0.033	335
(v) Erlang ↑ ₅ ↑ ₁	266.14	7327	0.036	433
(c) Haskell ↑ ₂ ↓ ₂	270.15	11582	0.023	494
(i) Dart ↓ ₁ ↑ ₁	290.27	17197	0.017	475
(i) JavaScript ↓ ₂ ↓ ₄	312.14	21349	0.015	916
(i) TypeScript ↓ ₂ ↓ ₂	315.10	21686	0.015	915
(c) Go ↑ ₃ ↑ ₁₃	636.71	16292	0.039	228
(i) Jruby ↑ ₂ ↓ ₃	720.53	19276	0.037	1671
(i) Ruby ↑ ₅	855.12	26634	0.032	482
(i) PHP ↑ ₃	1,397.51	42316	0.033	786
(i) Python ↑ ₁₅	1,793.46	45003	0.040	275
(i) Lua ↓ ₁	2,452.04	209217	0.012	1961
(i) Perl ↑ ₁	3,542.20	96097	0.037	2148
(c) Swift		n.e.		

- Langage C : 39 Joules.
- Javascript : 312 Joules.
- Différentiel d'environ 270 Joules.
- Choix d'écriture d'un micro-service, *C* vs *Javascript*, sollicité 1M/jour.
- Représente 2250 kwh/mois d'économie.

La consommation énergétique comme critère de choix

- Code *BinaryTree* pour Python :
<https://github.com/greensoftwarelab/Energy-Languages/blob/master/Python/binary-trees/binarytrees.py>
 - L'architecture globale obéit à nombreux autres critères : cohérence technique, compétences en interne, coût de développement, infrastructures/hébergement.
 - Approche du choix du langage sur une entité modulaire (comme un micro-service) cela a du sens.
- 

Résultat global (sur les 13 problèmes) et normalisé

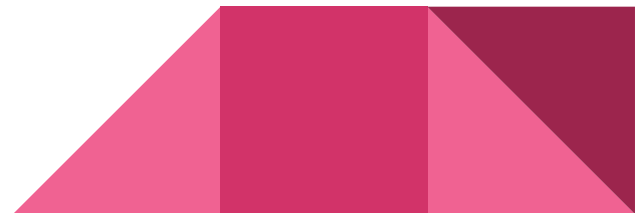
Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

- C#, 3 fois plus énergivore que C.
- Javascript, 4 fois plus.
- Python, 75 fois plus.

Exemple *BinaryTree* avec C & Python

- Langage C : 39 Joules.
- Python : 1800 Joules.
- Différentiel d'environ 1750 Joules.
- Choix d'écriture d'un micro-service, *C vs Python*, sollicité 1M/jour.
- Représente 486 kwh/jour d'économie, 14580 kwh/mois.



Combinaison des trois critères : énergie, mémoire, temps

Table 5. Pareto optimal sets for different combination of objectives.


Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Swift • PHP	Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
F# • Racket • Hack • Python	Lisp • Ocaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
JavaScript • Ruby	Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Dart • TypeScript • Erlang	Swift	TypeScript	Lua • JRuby • Perl
JRuby • Perl	Dart • F#	Erlang • Lua • Perl	
Lua	JavaScript	JRuby	
	Racket		
	TypeScript • Hack		
	PHP		
	Erlang		
	Lua • JRuby		
	Ruby		

Analyse du résultat normalisé

- Les langages compilés sont les moins énergivores.
- La corrélation vitesse/énergie est remise en cause :
 - En comparaison deux à deux, le langage plus rapide n'est pas toujours le moins énergivore.
- Par contre, la corrélation usage mémoire/énergie est observable : un langage économe en mémoire, est en général économe en énergie.



Analyse du résultat normalisé (2)

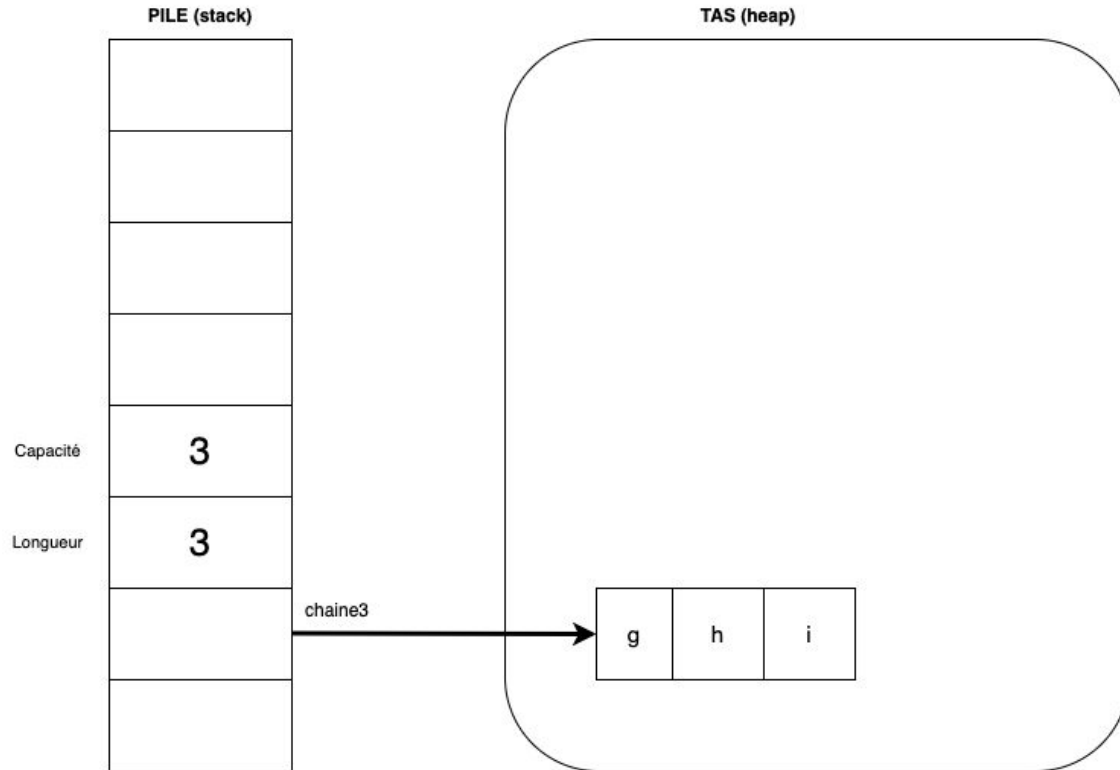
- Quand on considère la jointure énergie/mémoire.
 - Les langages en tête de classements sont largement des langages compilés, ou l'usage de la mémoire est particulièrement laissé à la main du programmeur (C, C++, Rust).
 - Mémoire allouée dans le *heap*, approche “mutable”, utilisation du strict nécessaire en termes de mémoire.
 - L'économie de mémoire peut expliquer la sobriété énergétique ?
- 

Flexibilité Mutable vs Immutable, une explication ?

- Mutable vs Immutable.
- Python, la plupart des types immutables.
- Python : List vs Tuple. Exemple du Set.
- C, C++, Rust : maîtrise de la gestion de la mémoire de manière fine et possiblement économe.



Exemple gestion fine String en Rust



Questions, échanges

Merci :)

