Impact des langages de programmation sur la performance énergétique des applications de calcul scientifique : un challenge ?

Cyrille Bonamy, *Laurent Bourgès*, *Laurent Lefèvre*

OSUG

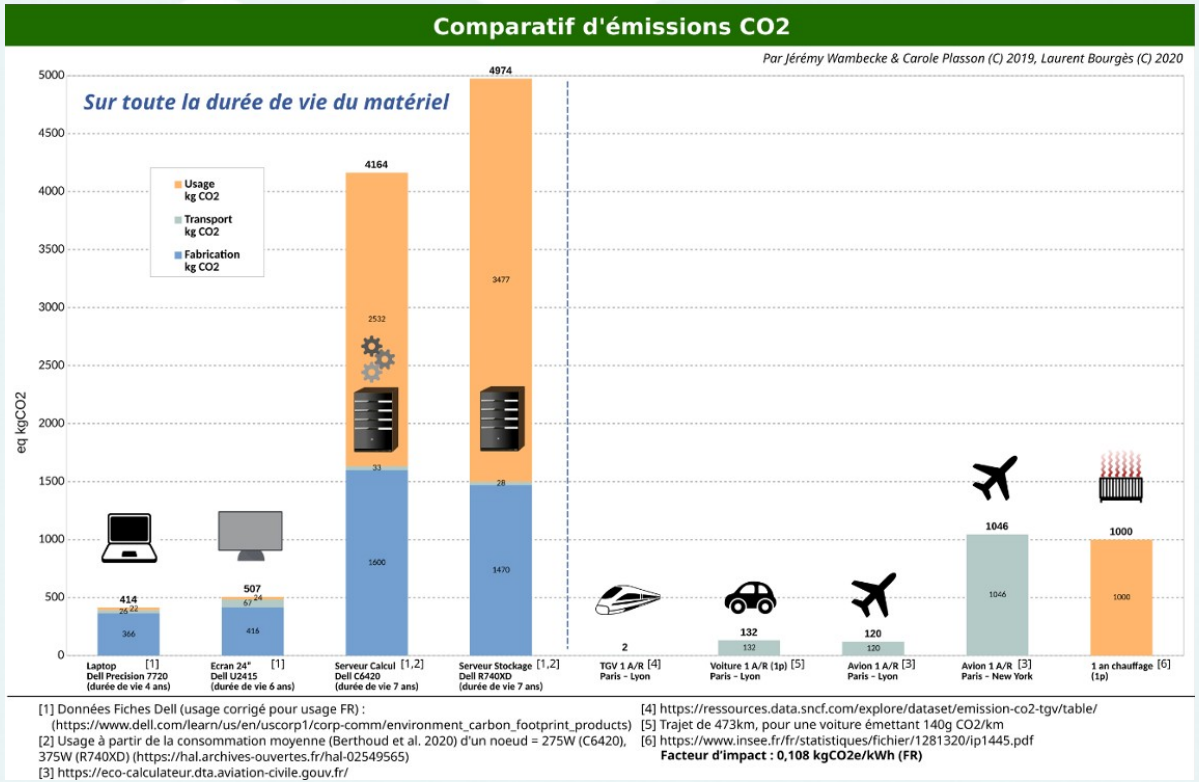
Le GDS EcoInfo

- Groupement de Service : des ingénieurs, chercheurs et étudiants à votre service !
- Agir pour réduire les impacts (négatifs) écologiques et sociétaux des TIC
- Quelques exemples de service
  - Audit Datacentre (CoC)
  - MatInfo
  - Ecodiag
  - Sensibilisation
  - Formation (ANF...)
  - Veille technologique
- Recherche

EcoInfo
Groupement de service

http://ecoinfo.cnrs.fr

Impacts du numérique



https://inhabitat.com

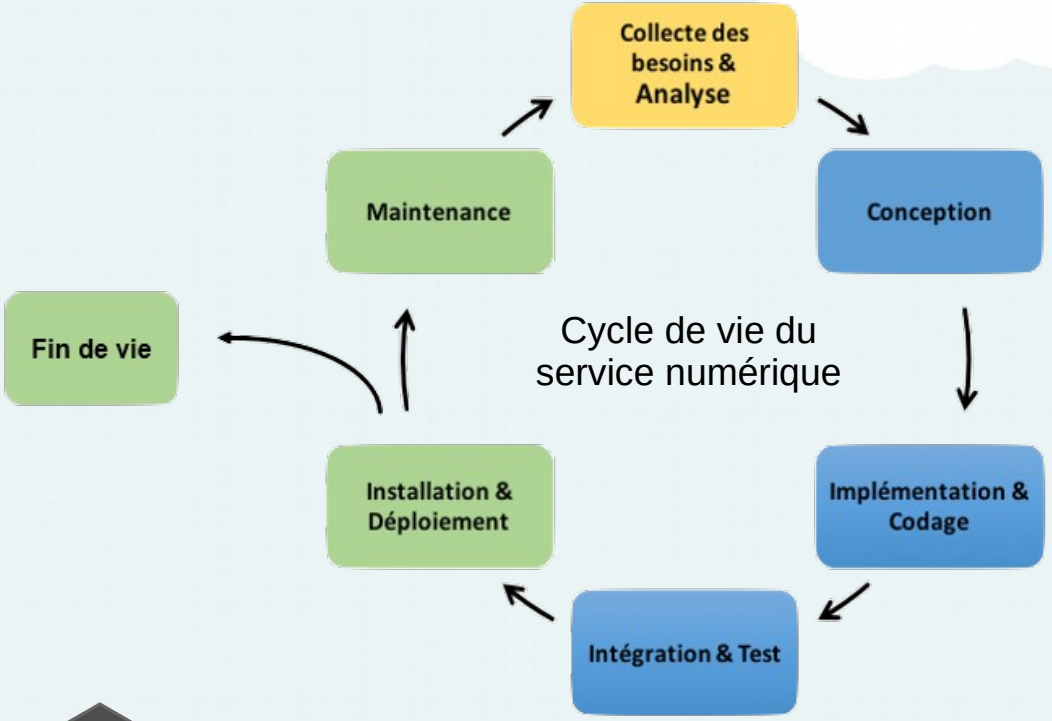
https://journalintegration.com/

- Impacts du matériel et des matériaux associés, impacts sociaux...
- Pas uniquement l'usage, mais tout au long du cycle de vie !
- Globalement autour de 4% des émissions mondiales de Gaz à Effet de Serre (GES)

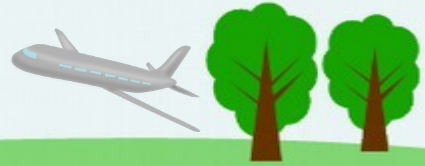
Impact du logiciel

Pourquoi ?

- Source d’obsolescence
- Impact direct sur le besoin matériel
- Impacts indirects
- Consommation énergétique reliée au coût de la phase d’usage



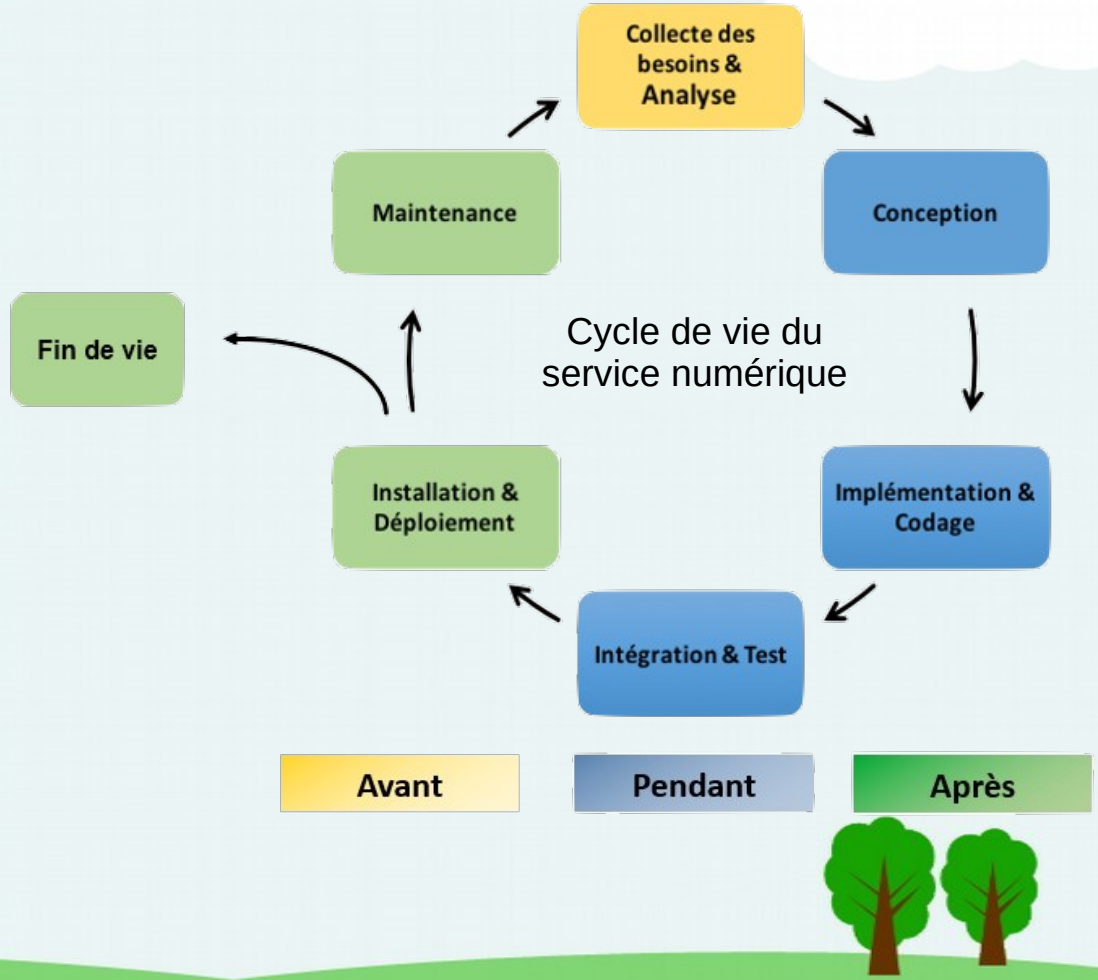
- Ordre de grandeur : 200 000 hCPU  = 1 tonne CO_2 = 1 A-R NY-Paris



Impact du logiciel

Cette étude est un focus sur :

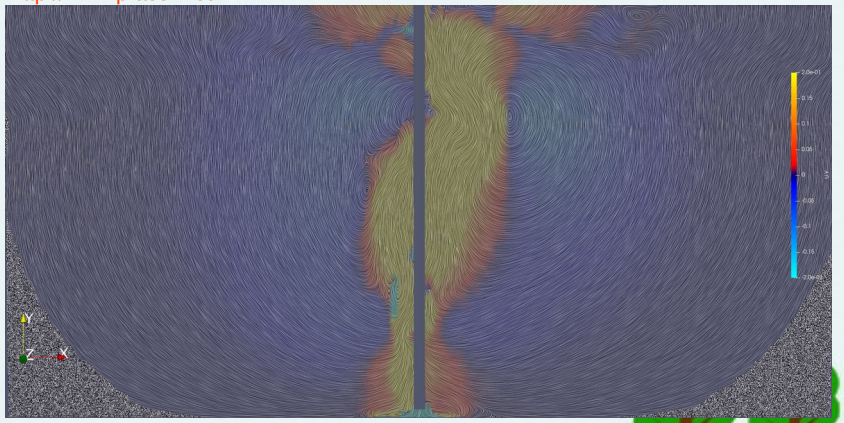
- Le choix du langage,
- À la frontière des étapes «avant» et «pendant» le développement



Contexte «logiciel» de l'étude

- Calcul scientifique = brique de base (simulations, IA, *big data*, etc.)
- Thématique qui nécessite de très grosses infrastructures (top500)

http://www.prace-ri.eu



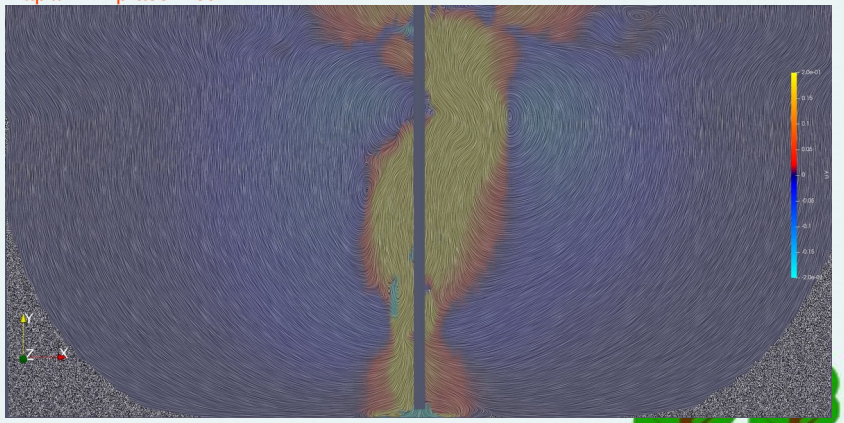
Bonamy, 2021 (OpenFoam, méthaniseur)

Contexte «logiciel» de l'étude

- Calcul scientifique = brique de base (simulations, IA, *big data*, etc.)
- Thématique qui nécessite de très grosses infrastructures (top500)

Impact du langage sur la performance énergétique ?

http://www.prace-ri.eu



Bonamy, 2021 (OpenFoam, méthaniseur)

Objectifs de l’étude

- Impact du langage sur la performance énergétique d’un calcul
- Cas réel : un algorithme de mécanique des fluides
- Inversion de matrice tridiagonale

  ⟹ Algorithme «TDMA» (récursif)

https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

For $i = 2, 3, \ldots, n$, do

$$w = \frac{a_i}{b_{i-1}},$$
$$b_i := b_i - w c_{i-1},$$
$$d_i := d_i - w d_{i-1},$$

followed by the back substitution

$$x_n = \frac{d_n}{b_n},$$
$$x_i = \frac{d_i - c_i x_{i+1}}{b_i} \quad \text{for } i = n-1, n-2, \ldots, 1.$$

```
Sub TriDiagonal_Matrix_Algorithm(N%,
                A#(), B#(), C#(), D#(), X#())
  Dim i%, W#
  For i = 2 To N
    W = A(i) / B(i - 1)
    B(i) = B(i) - W * C(i - 1)
    D(i) = D(i) - W * D(i - 1)
  Next i
  X(N) = D(N) / B(N)
  For i = N - 1 To 1 Step -1
    X(i) = ( D(i) - C(i) * X(i + 1) ) / B(i)
  Next i
End Sub
```

Mais pourquoi ce choix ?

- Cas réel, CPU intensif
- Un existant minimaliste

Intérêts

- Portage et écriture de A à Z
- Maîtrise du code

Choix des langages

Langage	Caractéristiques	Environnement
C	compilé, bas niveau	Gcc 10.2.1
Fortran	compilé, bas niveau	GNU Fortran 10.2.1
Java	JIT, GC, bas/haut niveau	OpenJDK 11.0.13
Julia	JIT, GC, syntaxe math, haut niveau	Julia 1.7.1
Python	interprété (GIL), pas de tableaux Numba : JIT, GC Transonic/Pythran : AOT, GC	Python 3.9.2 Numba 0.54.1 Transonic 0.4.12
Rust	compilé, bas niveau	Rustc 1.57.0
Go	AOT, GC	Go 1.17.5

Méthode de l'étude

- Une quantité de travail fixée a priori
- Nombre de cellules et paramètres fixés
- *Fair benchmark :* https://gricad-gitlab.univ-grenoble-alpes.fr/ecoinfo/ecolang
- *Mono-thread* et *Multi-thread*
- Mesures par wattmètre

Infrastructures considérées

Type	Age	CPU	coeurs	Gflops	Pmin(W)	Pmax (W)
Intel Xeon	2019	Xeon 5218	2x16	2300	160	500
AMD EPYC	2021	EPYC 7642	1x48	1800	307	451
ARM X2 99xx	2020	ThunderX2	2x32	1100	255	425
AMD Opteron	2006	Opteron 250	2x1	9.6	193	262
HP ZBook 15	2017	Intel i7-6820	1x4	173	10	62
RPi4	2020	A72,ARMv8	1x4	12	2	5.1

Focus sur Python

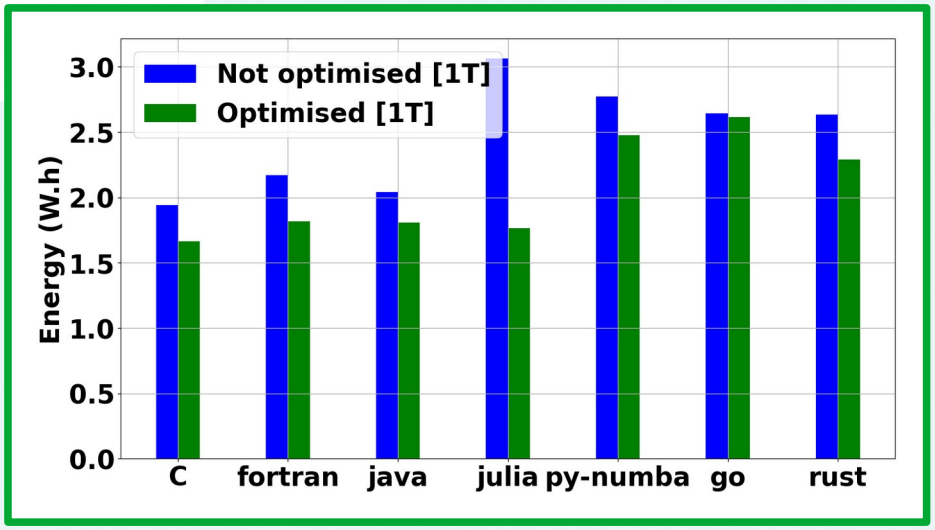
Énergie (W.h), Temps d'exécution (s)	Python Numpy	Python Numba @jit	Python Transonic/ Pythran
Mono-thread [1T]	92.1 W.h, 1 260 s	2.77 W.h, 38 s	2.57 W.h, 35 s
Multi-thread [MT]	6.4 W.h, 54.1 s	0.27 W.h, 2.09 s	0.24 W.h, 1.84 s
Speedup	23.3	18.2	19

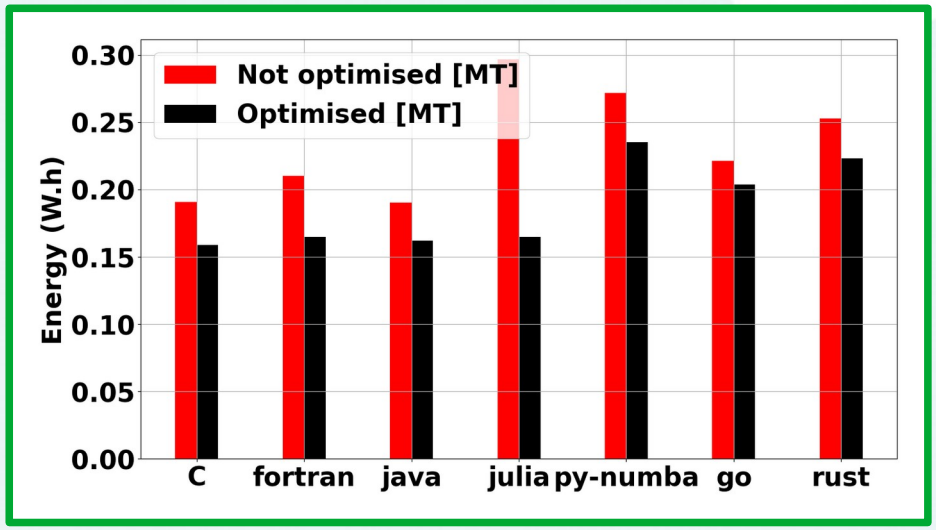
Tableau comparatif Python en usage Numpy, Numba, et Transonic/Pythran sur la machine Xeon 5218, Ncores = 32

$$Speedup = \frac{temps\ \ d'exécution\ \ sur\ \ 1\ \ cœur}{temps\ \ d'exécution\ \ sur\ \ N\ \ cœurs}$$

- Nécessité des accélérateurs (compilateurs) ou des bibliothèques compilées
- Python sans accélérateurs reste pertinent pour bon nombre d'applications (pré et post-processing notamment)

Impact de l’optimisation

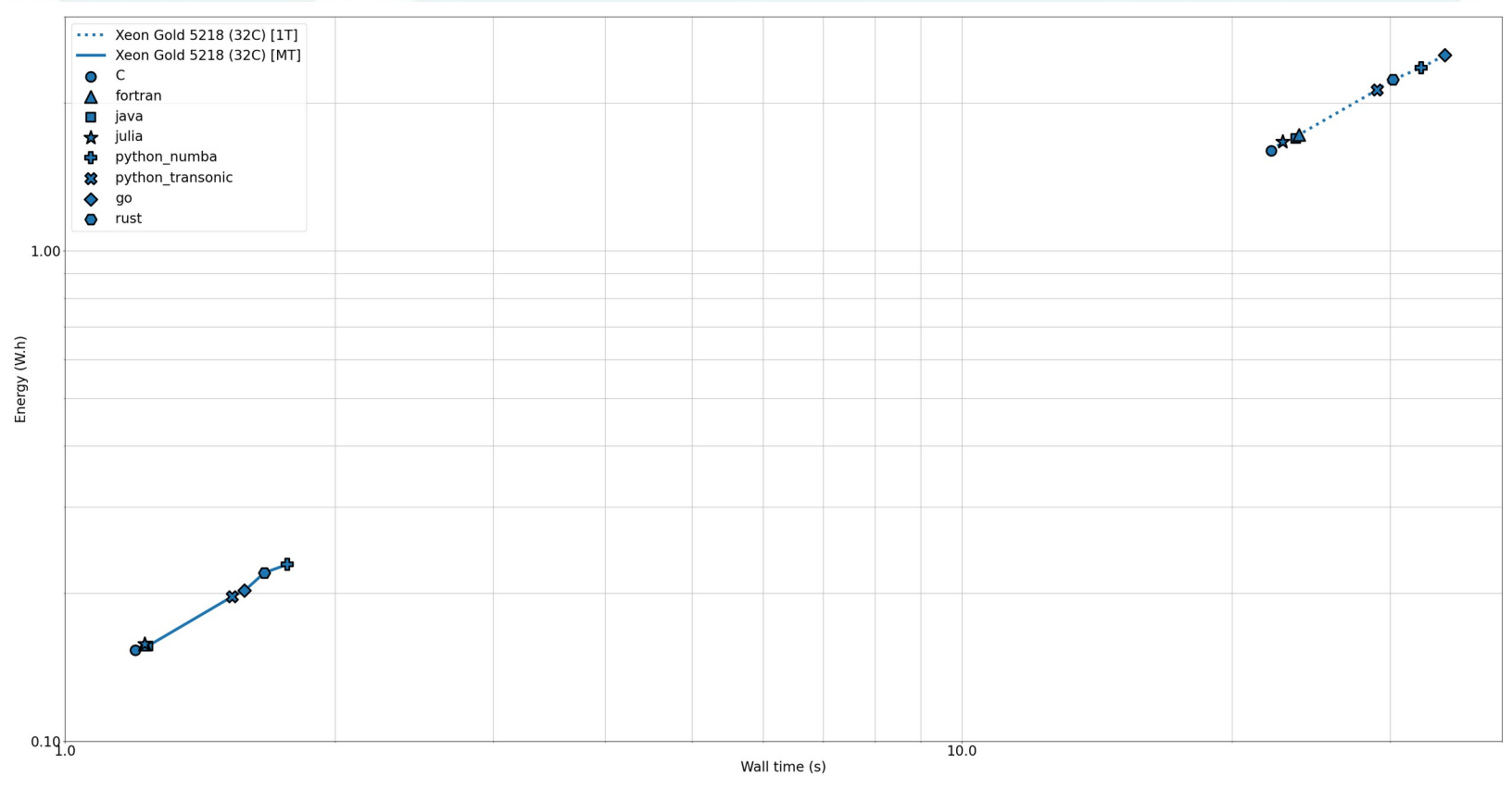




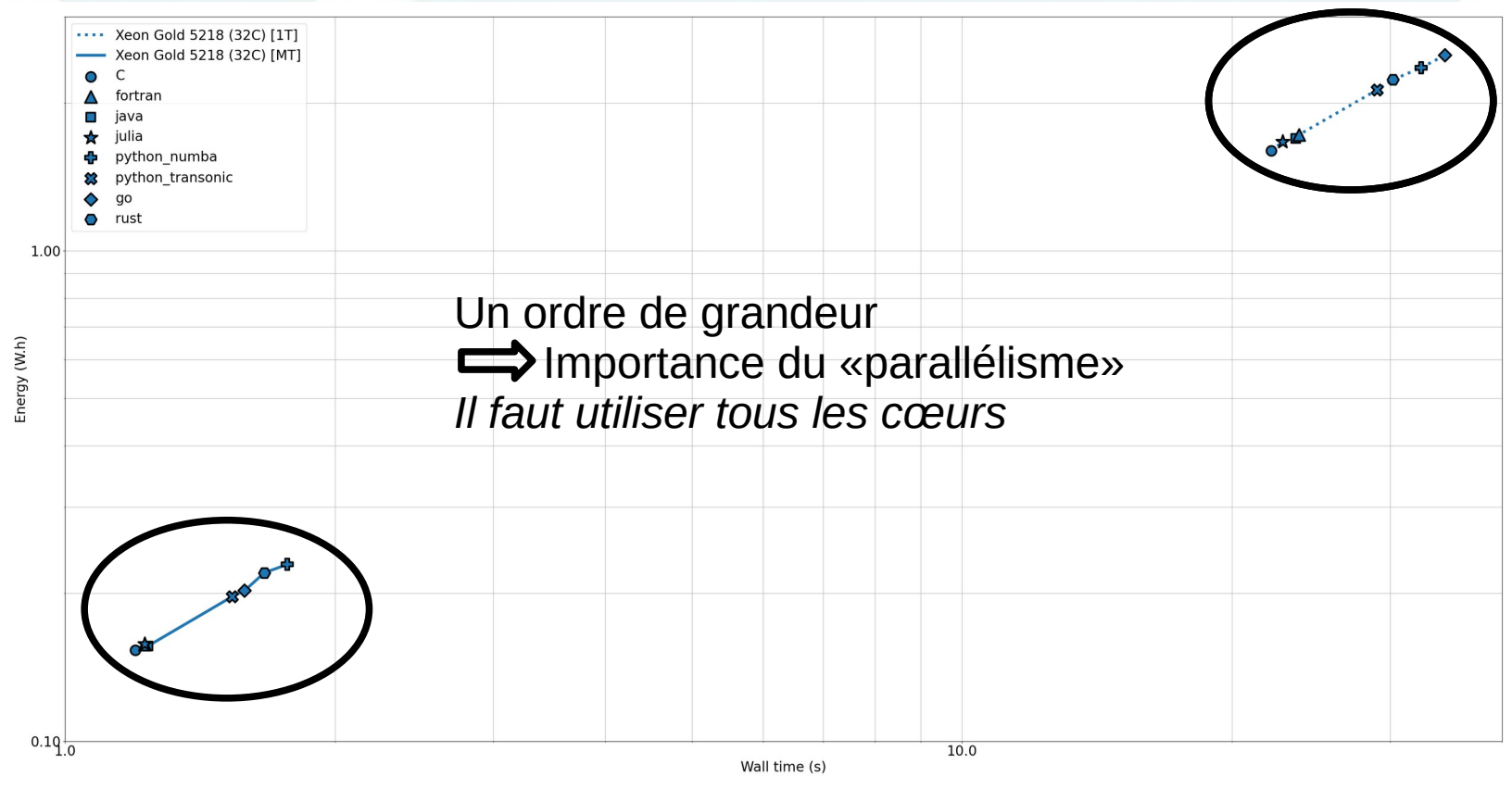
Comparaison énergétique pour la machine Xeon 5218 (32 cœurs) en utilisation mono-thread *(à gauche), et* multi-thread *(à droite)*

- «Optimisation naïve» intéressante (spécialement pour Julia)

Xeon 5218 (1T et MT)



Xeon 5218 (1T et MT)

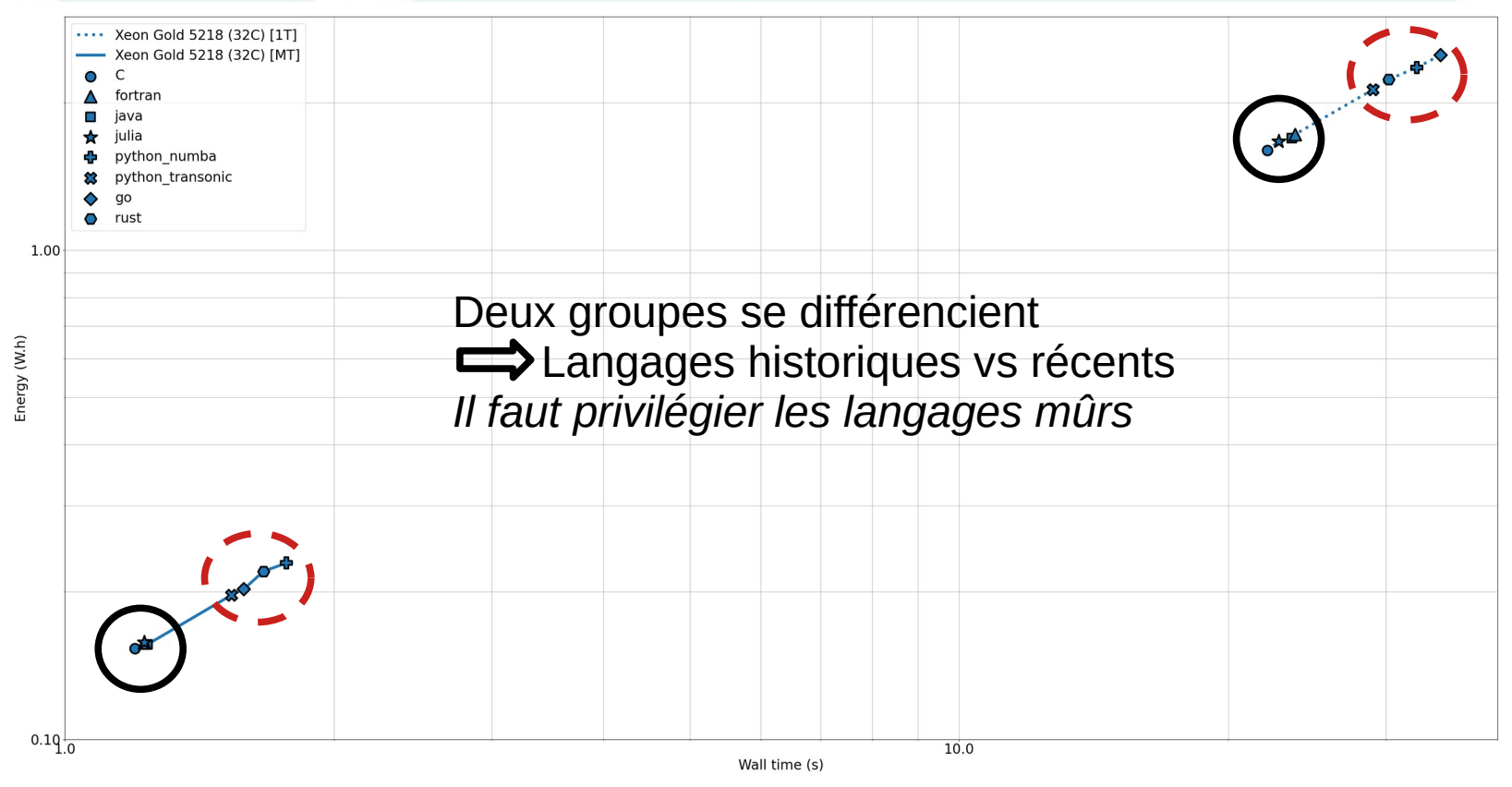


Un ordre de grandeur

⇨ Importance du «parallélisme»

Il faut utiliser tous les cœurs

Xeon 5218 (1T et MT)

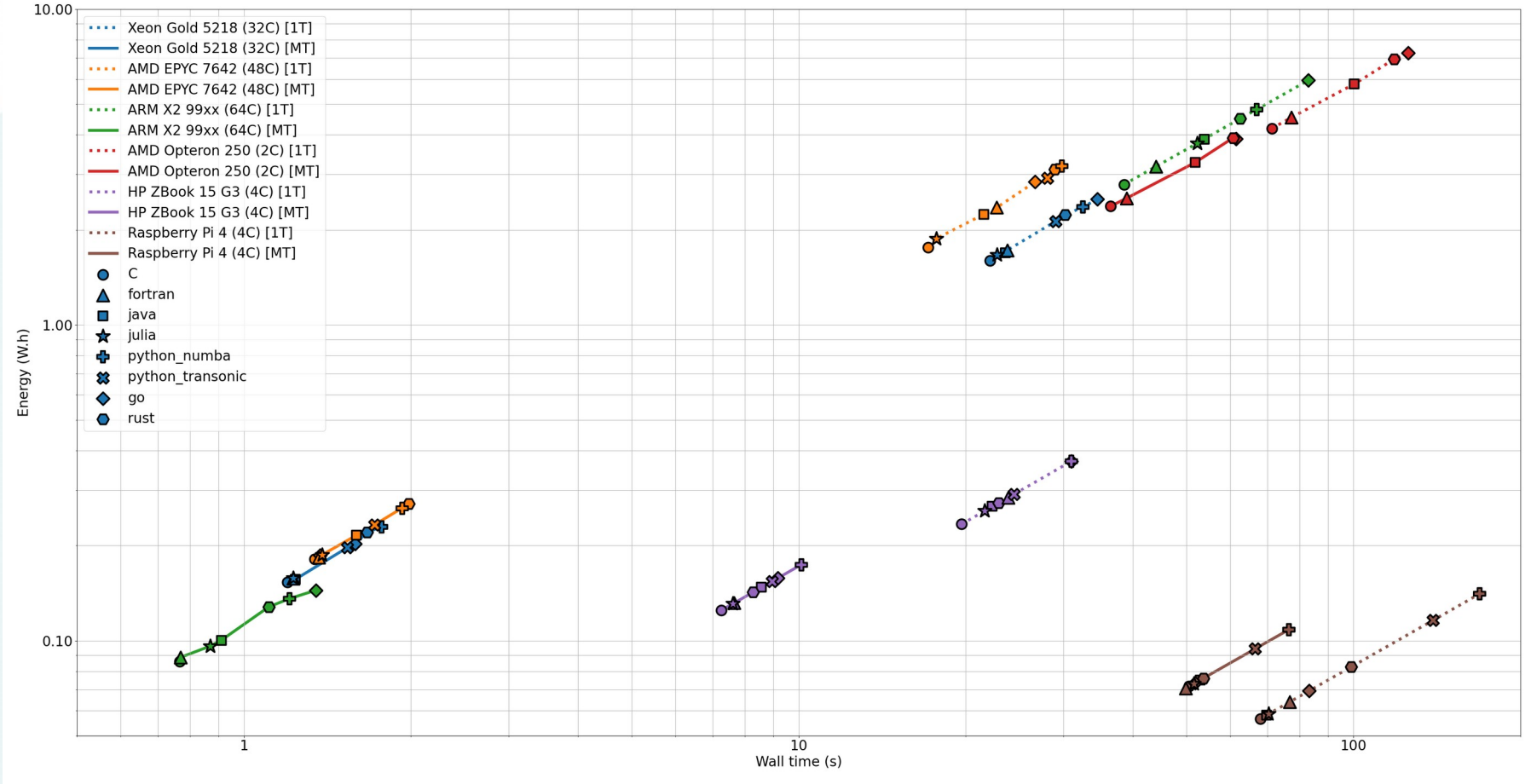


Deux groupes se différencient

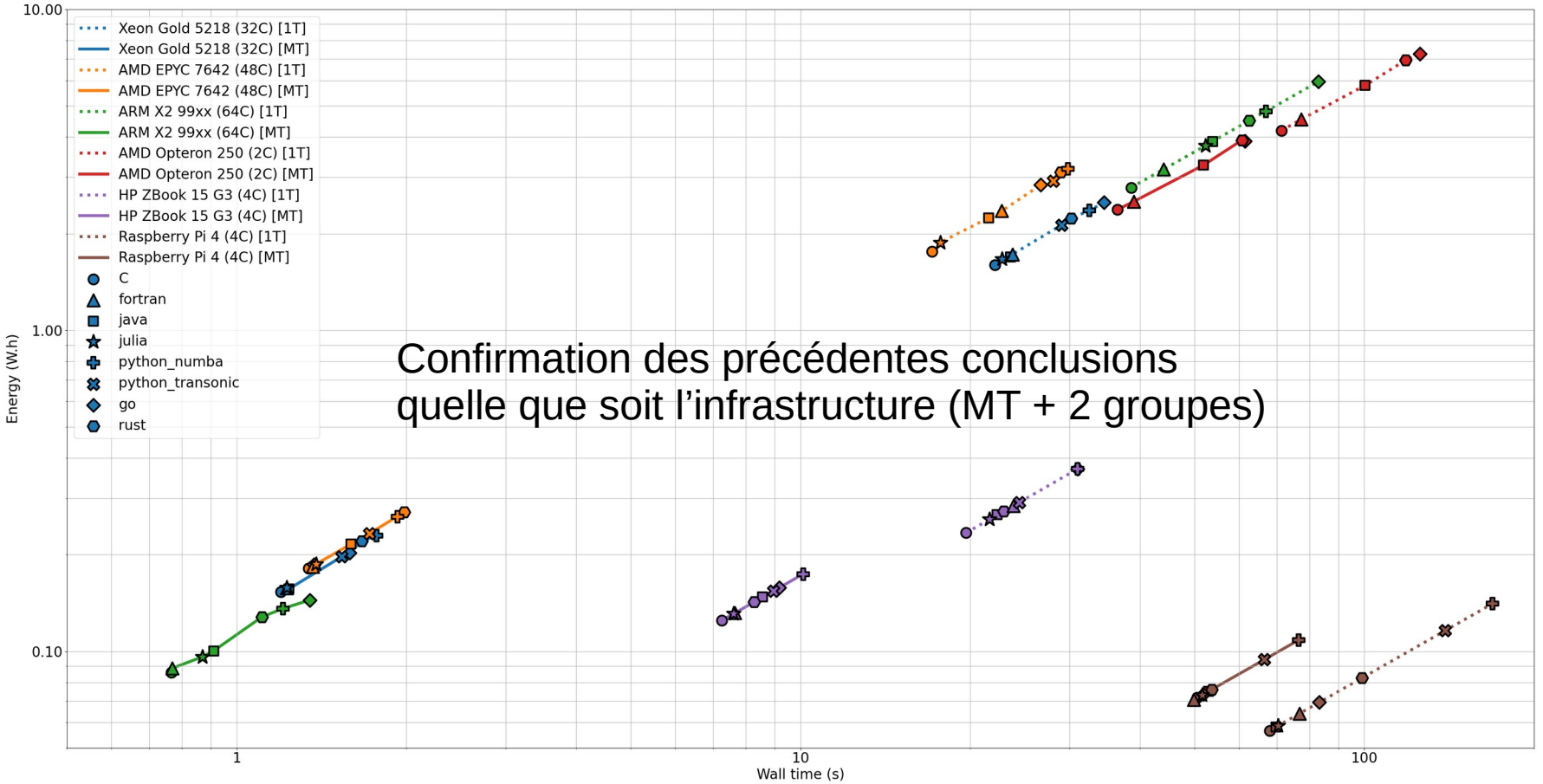
⇨ Langages historiques vs récents

Il faut privilégier les langages mûrs

Et les infrastructures ?

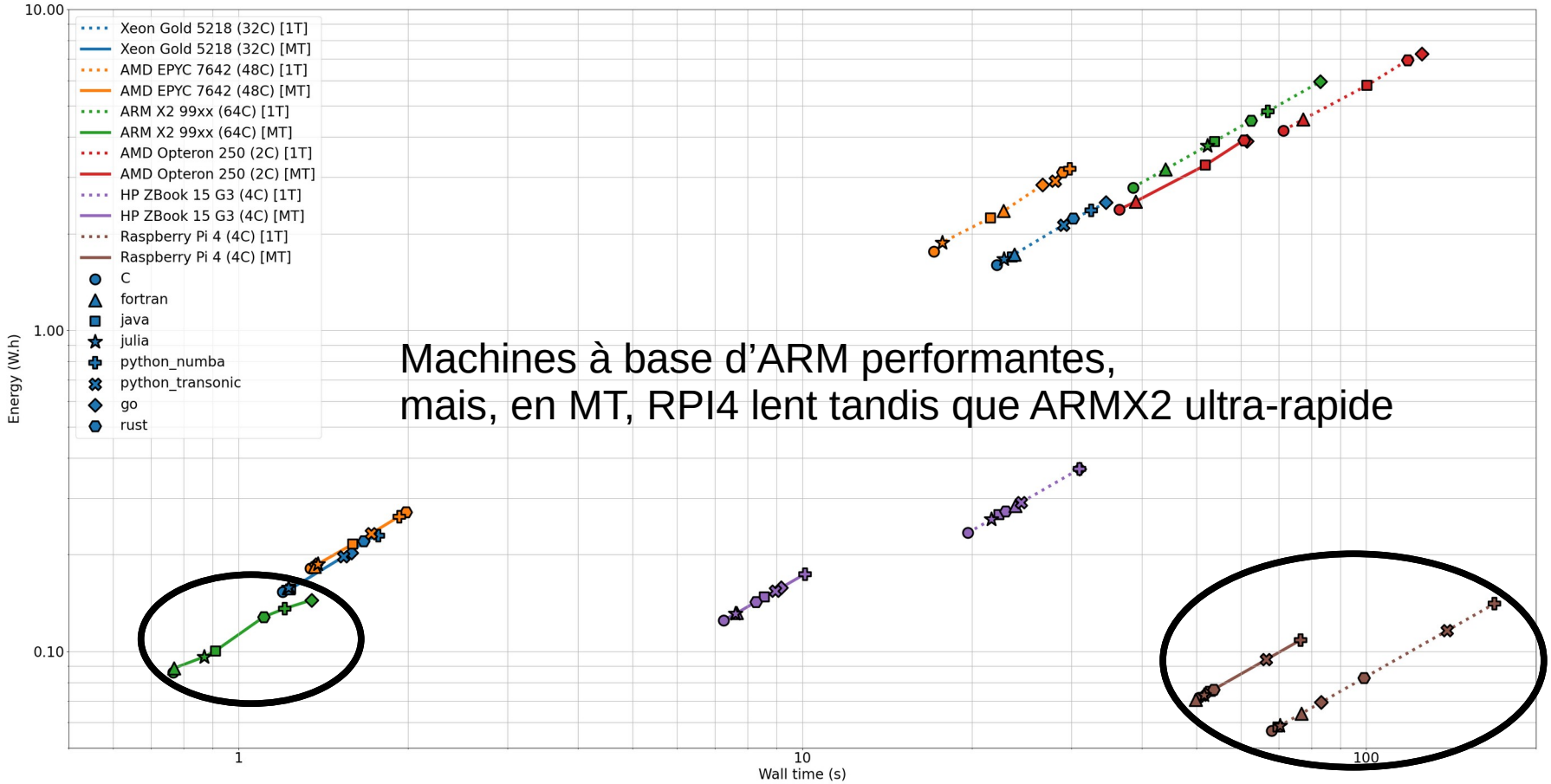


Et les infrastructures ?



Confirmation des précédentes conclusions
quelle que soit l'infrastructure (MT + 2 groupes)

Et les infrastructures ?



Pour conclure,

quelques leçons apprises au travers de cette étude

- Le portage rigoureux et l'optimisation de code prend du temps, mais les gains peuvent être conséquents
- Il est possible de générer du code Python performant, si on utilise des bibliothèques permettant la compilation (ou pré-compilées)
- La meilleure optimisation consiste à revoir le code dans son approche algorithmique ou/et de choisir la précision la plus adaptée au problème à résoudre

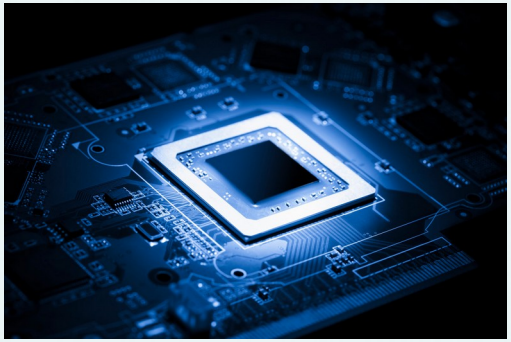
main1D_cb_numba_opt.py

```
26  ## Tri Diagonal Matrix Algorithm(a.k.a Thomas algorithm) solver
27  @jit(nopython=True, cache=True, fastmath=False)
28  def TDMAsolver(a, b, c, d, xc, tmp):
29      """
30      TDMA solver, a b c d can be NumPy array type or Python list type.
31      refer to http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm
32      """
33      nf = len(b)    # number of equations
34
35      # use outputs for temporary arrays:
36      bc = xc
37      dc = tmp
38
39      bc[:] = b[:]  # copy the array
40      dc[:] = d[:]  # copy the array
41
42      for it in range(1, nf):
43          mc = a[it - 1] / bc[it - 1]
44          bc[it] = bc[it] - mc * c[it - 1]
45          dc[it] = dc[it] - mc * dc[it - 1]
46
47      # xc = bc
48      xc[-1] = dc[-1] / bc[-1]
49
50      for il in range(nf - 2, -1, -1):
51          xc[il] = (dc[il] - c[il] * xc[il + 1]) / bc[il]
52
53  @jit(nopython=True, cache=True, fastmath=False)
54  def RMS_E(X1, X2):
55      rms = math.sqrt(((X1 - X2) ** 2).sum() / len(X1))
```

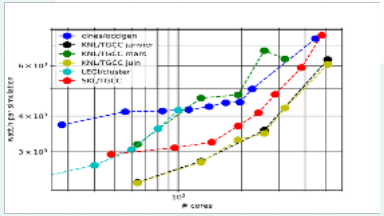
Pour conclure,

quelques leçons apprises au travers de cette étude

- Tous les cœurs d'une machine doivent être utilisés
- Les machines à base d'ARM sont réellement efficaces quel que soit le langage considéré, et sans «portage» comme le demanderait l'utilisation de GPU
- Deux groupes de langages se distinguent en termes d'efficacité : en tête, C, Fortran, Java et Julia, et légèrement derrière Python, Go et Rust
- Pour développer, l'utilisation d'un portable classique (voire d'un Raspberry Pi4) peut être bien plus économe, et celà tout particulièrement si on travaille sur l'algorithme séquentiel



Pour aller plus loin



- **JRES**, 2019 :
  Bonamy, C., Lefèvre, L. & Moreau, G. 2019.
  Calcul haute performance et efficacité énergétique : focus sur OpenFOAM

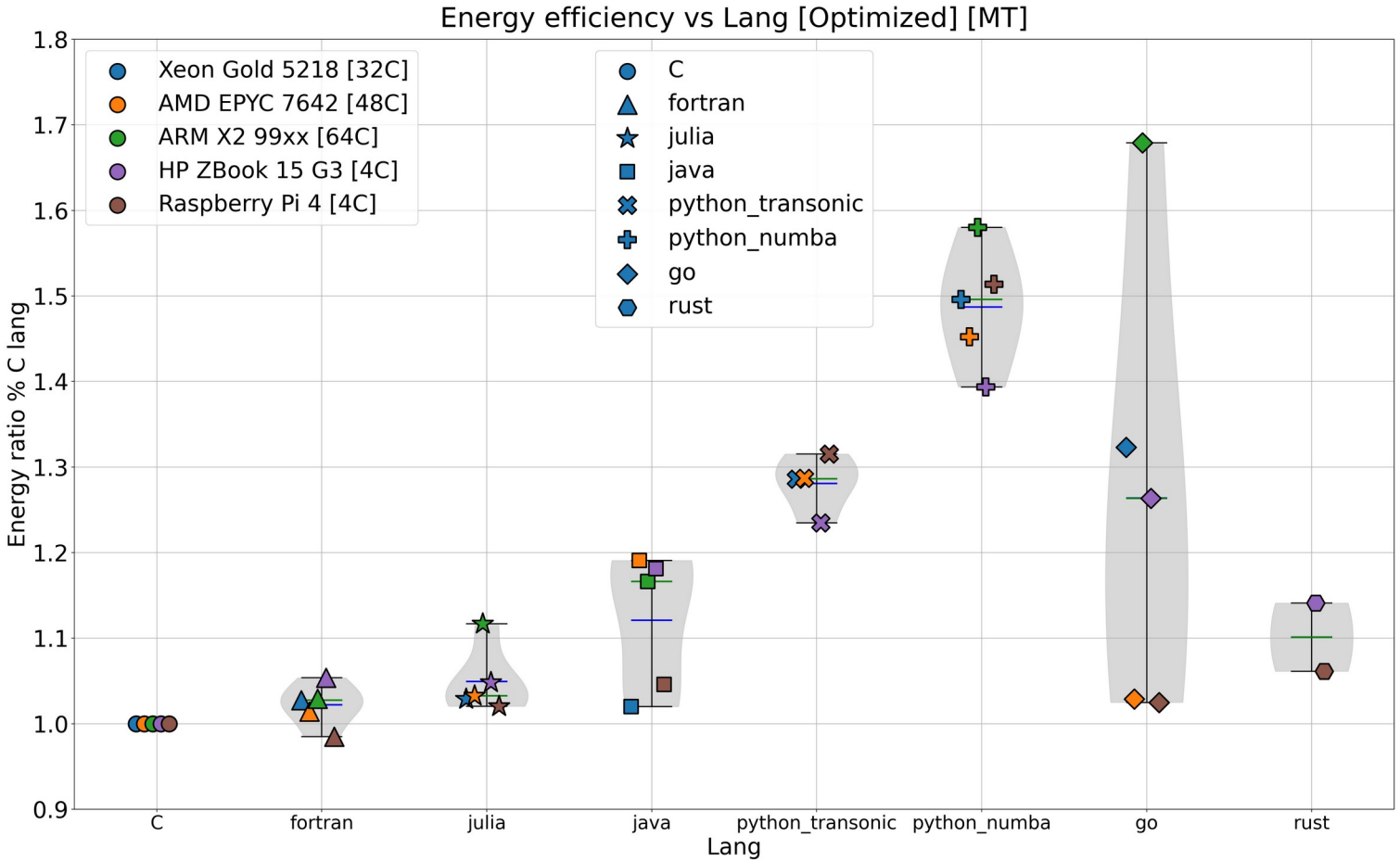
- Guide **EcoInfo V4**, 2022 :
  Bonamy, C., Boudinet, C., Bourgès, L., Dassas, K., Lefevre, L., & Vivat, F
  Je code : les bonnes pratiques en éco-conception de service numérique

- **GRICAD**, 2020 :
  Berthoud, F., Bzeznik, B., Gibelin, N., Laurens, M., Bonamy, C. & al.
  Estimation de l'empreinte carbone d'une heure.coeur de calcul

MERCI POUR VOTRE ATTENTION

OSUG

Bonus


Energy efficiency vs Lang [Optimized] [MT]
Xeon Gold 5218 [32C]
AMD EPYC 7642 [48C]
ARM X2 99xx [64C]
HP ZBook 15 G3 [4C]
Raspberry Pi 4 [4C]
C
fortran
julia
java
python_transonic
python_numba
go
rust
Energy ratio % C lang
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
C
fortran
julia
java
python_transonic
python_numba
go
rust
Lang