ENVOL 2023 - Impacts des algorithmes et des langages

Laurent Bourgès (CNRS - OSUG), Cyrille Bonamy (CNRS - LEGI) - Laurent Lefèvre (Inria Avalon)

Note: Démarrer une machine dédiée sur la plate-forme Grid'5000 avant l'introduction.

Introduction

Cette séance de travaux pratiques va vous permettre d'évaluer l'efficacité énergétique des algorithmes, de plusieurs langages de programmation et vous montrera l'importance des optimisations et du dimensionnement (scalabilité) des algorithmes.

Le cas d'étude considéré est un code de calcul utilisé lors d'un cours de mécanique des fluides par le Maitre de Conférences Julien Chauchat. Pour les personnes intéressées, le cours associé à ce code est accessible via le fichier FormationVFMecaFlu.pdf.
Dans le cadre de ce TP, le même algorithme a été codé dans les langages Python, Julia, Fortran, C, Java, Go et Rust. Pour un même langage, plusieurs versions du code ont été implémentées pour évaluer certaines optimisations.
Dans un premier temps, nous allons exécuter les diverses versions du code, en faisant attention aux temps de réponse de chaque version. Puis nous vérifierons que le temps de calcul est bien relié à consommation électrique.
Nous allons donc travailler sur la plate-forme Grid'5000 pour mesurer la consommation électrique d'une machine de calcul pendant chaque expérience afin d'évaluer les 2 critères : temps de calcul et consommation énergétique.
Le but est de mettre en lumière certaines bonnes pratiques du développement logiciel.

Note: Tous les fichiers sont disponibles dans un dépôt public (GPL2):
`https://gricad-gitlab.univ-grenoble-alpes.fr/ecoinfo/ecolang`

Le dossier **ecoconception_logicielle** contient des scripts shells et les implémentations du code dans les sous-répertoires cfiles (C), ffiles (Fortran), gofiles (Go), javafiles (Java), jlFiles (Julia), pyfiles (Python) et rustFiles (Rust).

1 Accès machine Grid'5000 - cluster Lyon/nova

Au préalable, vous avez obtenu un accès (login) sur la plateforme Grid'5000 par email et une clé SSH a été générée sur votre poste pour la renseigner dans votre compte Grid'5000.

Quelques liens utiles:

- Monika / Lyon (réservation) : `https://intranet.grid5000.fr/oar/Lyon/monika.cgi`
- Grafana / Lyon (supervision): `https://api.grid5000.fr/stable/sites/lyon/metrics/dashboard/d/kwollect/kwollect-metrics`

Les questions ont un code couleur qui précise si vous travaillez sur la frontale de Grid'5000, si vous travaillez sur votre nœud réservé ou si vous travaillez sur votre machine locale.

1.1 Accès à un noeud de calcul reservé

Question 1
Dans un terminal, connectez vous à une machine d'accès via la commande
`ssh <login>@access.grid5000.fr`
puis connectez vous à la frontale (ou *frontend*) de Lyon via la commande
`ssh lyon`

Dans le cadre de ce TP, tout le cluster `Lyon - nova` a été reservé (20 noeuds).
Vous allez donc travailler au sein de ressources réservées appelées *container job* et repéré par un `JOB_ID` qui vous sera donné au début du TP.

```
JOB_ID: 1614676
```

Question 2
Réserver un nœud du cluster nova pour une durée de 3h avec l'option de monitoring haute fréquence (50Hz) via la commande

```
oarsub -t deploy -t inner=1614676 -l "host=1,walltime=03:00"
-t monitor="wattmetre_power_watt" "sleep infinity" -project
lab-2023-anf-envol
```

L'interface OAR va attribuer un noeud et un nouveau job ID et afficher son identifiant:
`OAR_JOB_ID=1614829`

Question 3
Récupérer le nœud nova réservé via la commande
oarstat -u

Job id	Name	User	Submission Date	S	Queue
1614912		lbourges	2023-11-16 09:19:35	R	default

Question 4
Récupérer le nom du noeud réservé via la commande

```
export OAR_JOB_ID=1614912
oarstat -j $OAR_JOB_ID -f | grep assigned_hostnames
```

L'interface répond :

```
assigned_hostnames = nova-XX.lyon.grid5000.fr
```

Question 5
Déployez l'image de l'OS préparée pour ce TP avec l'outil `kadeploy3` via la commande:

```
kadeploy3 -m nova-XX.lyon.grid5000.fr -e ubuntu2004-tp2-cr17-ecolang-2
-u lbourges
```

Le déploiement de l'image va prendre quelques minutes.
Lorsque le déploiement est terminé, vous pouvez vous connecter au nœud avec la commande:
`ssh root@<nova-XX`

Question 6
Sur le nœud de calcul, les fichiers du TP sont installés dans le répertoire `ecoconception_logicielle`.

```
cd ecoconception_logicielle
```

Lister les fichiers et dossiers avec la commande

```
ls -l
```

```
bench-settings.sh   cpu_list_cpu_cores.sh   jlfiles          run-bench-MT.sh
cfiles              cpu_normal.sh           log_env.sh       run-bench.sh
clean-all.sh        cpu_show_freq.sh        matlab           run-gov-C_opt.sh
clean-bench.sh      env-all.sh              plot.sh          run-gov.sh
cpu_fixed.sh        env-bench.sh            postpro_scripts  run-hpl.sh
cpu_hpc.sh          ffiles                  pyfiles          rustfiles
cpu_hpc_turbo.sh    gofiles                 results          setup-all.sh
cpu_ht_off.sh       hpl                     run-all.sh       setup-bench.sh
cpu_ht_on.sh        javafiles               run-bench-1T.sh
```

Question 7
Pour lister les langages installés et leurs versions, utiliser le script `env-all.sh` :
`./env-all.sh`
Quelle est la distribution linux et sa version ?

```
env-all: start ...
--------------------------------------
Environment on command 'cfiles/1DC' ...
CC version:
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

CCLAGS : -Wall -O3
--------------------------------------
Environment on command 'jlfiles/1Djulia' ...
Julia version:
julia version 1.9.3
--------------------------------------
--------------------------------------
Environment on command 'pyfiles/1Dpython_numpy' ...
Python version:
Python 3.8.10
...
```

1.2 Collecte de la consommation électrique du noeud

Un second terminal sur la frontale sera utilisé pour les transferts de fichiers et récupérer les mesures de consommation électrique de l'interface kwollect.

Question 8
Dans un **second** terminal, connectez vous à une machine d'accès via la commande
`ssh <login>@access.grid5000.fr`
puis connectez vous à la frontale (ou *frontend*) de Lyon via la commande
`ssh lyon`

Afin de faciliter la récupération des données de consommation depuis l'API Kwollect et leur visualisation, nous avons préparé quelques scripts pour ce TP.

Question 9
La première fois, veuillez récupérer et extraire l'archive avec les scripts avec les commandes suivantes:
```
wget http://public.lyon.grid5000.fr/~vostapenco/
tp-consumption-cr17/scripts_front.tar.gz
tar -xvf scripts_front.tar.gz
```

Question 10
Utiliser le script `get_consumption_kwollect.py` pour récupérer les données de consommation du nœud entre deux dates au format CSV :
```
python3 get_consumption_kwollect.py --site lyon --host nova-9
--start 2023-11-16T09:45:00 --end 2023-11-16T09:55:00 --csv
kwollect_TEST-CASE.csv
```
Au début du TP, vous pouvez récupérer la consommation pendant la phase (START) entre l'heure de début du déploiement (kadeploy) et maintenant.

Question 11
Utiliser le script `plot_consumption_kwollect.py` pour créer un graphique de consommation à partir du fichier CSV :
```
python3 plot_consumption_kwollect.py --csv kwollect_TEST-CASE.csv
--plot kwollect_TEST-CASE.jpg
```

Question 12
Utiliser le script `calculate_energy_kwollect.py` pour calculer l'énergie consommée par le nœud à partir du fichier CSV :
```
python3 calculate_energy_kwollect.py --csv kwollect_TEST-CASE.csv
```

```
Wattmeter energy (Joules): 40170.87093419999
Wattmeter energy (Wh): 11.158575259499997
```

Question 13
Récupérer le graphique de consommation dans un **troisième** terminal :
```
scp <login>@access.grid5000.fr:lyon/kwollect_TEST-CASE.jpg .
```

Note: Afin de bien suivre les opérations et la reproductibilité, il est recommandé d'utiliser un journal de bord sous la forme d'un fichier texte pour garder l'historique des expériences avec un identifiant [TEST-CASE-1] les dates de début et de fin ainsi que les conditions de l'expérience.
Dans la suite du TP, vous aurez l'occasion d'utiliser des scripts pour lancer les expériences qui tracent tous les résultats afin de garder l'historique complet des expériences.
Il est recommandé d'utiliser l'identifiant d'expérience dans les noms de fichier de sorties (graphique, données ...) pour plus de clarté.

Veuillez bien garder les terminaux ouverts et utiliser l'historique pour répéter ces enchaînements de commande afin de récupérer la consommation électrique, faire le graphique et calculer l'énergie consommée par l'expérience.

2 Prise en main du code scientifique (Python)

Pour consulter les codes utilisés pendant ce TP, en local ou via l'interface Gitlab (web), il suffit de consulter le dépôt public `ecolang` :
https://gricad-gitlab.univ-grenoble-alpes.fr/ecoinfo/ecolang

Nous allons étudier la version originale du code Python en termes de fonctionnalité, quelques implémentations différentes (optimisées) et la performance des algorithmes.

Dans cette section, les fichiers sont dans le répertoire:
`ecoconception_logicielle/pyfiles/deprecated/`

Question 14
Sur le nœud de calcul, aller dans le répertoire :
```
cd ecoconception_logicielle
ls -l
```

```
-rw-r--r-- 1 1000 1000 4580 Nov 14 14:31 main1D.py
-rw-r--r-- 1 1000 1000 5278 Nov 14 14:31 main1D_cb.py
-rw-r--r-- 1 1000 1000 4330 Nov 14 14:27 main1D_cbv1.py
-rw-r--r-- 1 1000 1000 5368 Nov 14 14:28 main1D_cbv4_plot.py
```

Voici la fiche de résultats pour cette partie:

Langage	programme	Temps (s)	Taille (M)
Python	main1D.py		2048
Python	main1D_cb.py		2048
Python	main1D.py		4096
Python	main1D_cb.py		4096
Python	main1D_cbv1.py		
Python	main1D_cbv4_plot.py		

2.1 Python : main1D.py - version originale

Implementation: `main1D.py`
Algorithme: matriciel (2048 pt)

Question 15
Exécuter la version originale de l'algorithme : :
```
python3 main1D.py
```

```
case : diffusion
M : 2048
dx : 0.0004885197850512946
dt : 0.1
eps: 1e-09
Matrice M x M: 32768.0 kb
k= 0 : RMS U = 0.045408200843564415
k= 1 : RMS U = 0.022800814991688578
k= 2 : RMS U = 0.011474216548153275
k= 3 : RMS U = 0.00577473863179966
k= 4 : RMS U = 0.0029063177386033294
```

```
k= 5 : RMS U = 0.001462695455104649
k= 6 : RMS U = 0.0007361473171634006
k= 7 : RMS U = 0.00037048920247915833
k= 8 : RMS U = 0.00018646029944925454
k= 9 : RMS U = 9.384198802950539e-05
k= 10 : RMS U = 4.722892067053654e-05
k= 11 : RMS U = 2.376943408366081e-05
k= 12 : RMS U = 1.1962712477294349e-05
k= 13 : RMS U = 6.020609889342228e-06
k= 14 : RMS U = 3.0300605246560863e-06
k= 15 : RMS U = 1.524973002460496e-06
k= 16 : RMS U = 7.674903935427645e-07
k= 17 : RMS U = 3.862634675551273e-07
k= 18 : RMS U = 1.94399290516959e-07
k= 19 : RMS U = 9.78375961565514e-08
k= 20 : RMS U = 4.923981920865934e-08
k= 21 : RMS U = 2.4781404322828395e-08
k= 22 : RMS U = 1.2472116265092462e-08
k= 23 : RMS U = 6.276992430093799e-09
k= 24 : RMS U = 3.159007987581329e-09
k= 25 : RMS U = 1.589892257133484e-09
The simulation has converged in 26 iterations
Time in seconds = 5.642541324999911
result samples:
u[ 0 ] = 3.660975555710436e-12
u[ 204 ] = 0.044863155641457694
u[ 408 ] = 0.07979458714416685
u[ 612 ] = 0.1047942945426996
u[ 816 ] = 0.11986227786163131
u[ 1020 ] = 0.12499853711690102
u[ 1224 ] = 0.12020307231402412
u[ 1428 ] = 0.10547588344773287
u[ 1632 ] = 0.08081697050254508
u[ 1836 ] = 0.046226333454115504
u[ 2040 ] = 0.0017039722716841726
```

Le programme donne les conditions initiales, le temps d'exécution et quelques résultats.

✘ **Question 2.1.** *Que fait le programme ?*
Quels sont les résultats ?
Combien de répétitions ? d'itérations ?
Quelle est la durée donnée par le programme ?

✘ **Question 2.2.** *Exécutez à nouveau le script en utilisant la commande time afin de récupérer le temps CPU.*
Quelle est l'unité de mesure ? Est-ce cohérent avec la mesure du programme [Time in seconds] ?

Lancer vmstat ou htop pour observer la charge de la machine:

```
vmstat 1
htop
```

✘ **Question 2.3.** *Quelle est la consommation CPU et de mémoire du programme ?*
Est-ce que les programmes utilisent le multi-threading (1T ou plus) ?

Question 16
Regarder le code original (ou en local):
`vi main1D.py`

✘ **Question 2.4.** *Quelle est la taille du problème ?*
Quel algorithme est utilisé ?
Quelle est la précision des résultats ?

2.2 Python : main1D_cb.py - TDMA

Implementation: `main1D_cb.py`
Algorithme: numpy TDMA (2048 pt)

Question 17
Exécuter la version modifiée par Cyrille Bonamy de l'algorithme : :
`python3 main1D_cb.py`

Question 18
Quelles sont les différences à l'aide de la commande :
`diff main1D.py main1D_cb.py`

✘ **Question 2.5.** *Quelle est la taille du problème ?*
Quel algorithme est utilisé ?
Est-ce que les résultats sont identiques ?
Combien de répétitions ? d'itérations ?
Quelle est la durée donnée par le programme ? Quel est le gain de performance et l'impact sur la consommation de mémoire ?

2.3 Python : scalabilité VO vs TDMA

Question 19
Modifier les programmes pour changer M = 4096 :
`vi main1D.py`
`vi main1D_cb.py`
Puis relancer les codes:
`python3 main1D.py`
`python3 main1D_cb.py`

✘ **Question 2.6.** *Quel est l'impact sur les temps de réponse ?*
Etablir le facteur de scalabilité (2048 vs 4096) pour les 2 versions ?

✍ **Note:** O(N^2) vs O(N): TDMA bien adapté à ce problème précis => les temps de réponses sont excellents !
La taille du problème de 4 million de points pour la suite : c'est l'effet 'Rebond' !

2.4 Python : main1D_cbv1.py - référence

Implementation: `main1D_cbv1.py`
Algorithme: numpy TDMA (4M pt)

Question 20
Exécuter la commande :
```
python3 main1D_cbv1.py
```

✘ **Question 2.7.** *Quelle est la taille du problème ?*
Quelle est la précision des résultats ? Quel est l'impact sur les temps de réponse et de mémoire du programme ?

Note: Numpy est lent, pourquoi ?
itérations directes a[i] dans solver => pas de fonction C optimisée => interpreteur python
=> Idée: comparer les accélérateurs python ensuite et d'autres langages HPC ou plus généralistes

2.5 Python : main1D_cbv4_plot.py - numba

Cette version utilise la librairie numba pour compiler la fonction TDMASolver (JIT):

Question 21
Quelles sont les différences à l'aide de la commande :
```
diff main1D_cbv1.py main1D_cbv4_plot.py
```

Question 22
Exécuter la commande :
```
python3 main1D_cbv4_plot.py
```

✘ **Question 2.8.** *Quel est l'impact sur les temps de réponse ?*
Quel est le gain de performance ?

Note: @jit est magique ! Il est possible d'éxécuter le code python numpy (pur) beaucoup plus vite à l'aide de librairies numba, transonic ... qui compilent le code critique (@jit) à la volée pour optimiser les performances. Jusqu'à quel point est-ce efficace en comparaison avec d'autres langages ?

Enfin, vous pouvez regarder le monitoring Grafana pour observer la consommation eléctrique depuis le début de cette section à https://api.grid5000.fr/stable/sites/lyon/metrics/dashboard/d/kwollect/kwollect-metrics.

✘ **Question 2.9.** *Comment interpréter la consommation du noeud ?*

2.6 Python : visualisation (démo)

Les programmes `main1D.py` et `main1D_cbv4_plot.py` affichent les résultats en fin de calcul à l'aide de la librairie connue matplotlib.

(en raison de l'utilisation des serveurs (X non disponible ?), une démo va être présentée)

Question 23
Exécuter la commande :
`python3 main1D.py`
Puis :
`python3 main1D_cbv4_plot.py`

✘ **Question 2.10.** *Quel est l'impact de la visualisation sur la machine ? (top)*

☞ **Note:** 4s de calcul mais plot très long à traiter car 4M pt (déjà en local mais pire à distance via ssh -X !!)

Il faut utiliser poste utilisateur (plus efficace?) pour la visualisation et des algorithmes adaptés (GPU, heat-map, spatial indexes...)
C'est un métier et une expertise en soi !

☞ **Note:** Attention au coût des traces (printf, stdout, log files) qui peut être non négligeable :

- gestion des niveaux de verbosité
- volume des traces
- synchronisation pénalisante en multi-thread)

3 Etude performance & consommation énergie sur algorithme TDMA

Dans cette section, nous allons utiliser les scripts shell pour s'assurer d'un protocole expérimentale reproductible et détaillé: 10 répétitions du programme pour les statistiques, les scripts shells pour compiler, lancer les expériences, stocker les résultats ...

Le principe des scripts est simple:

- `cpu_...sh` : scripts de gestion du CPU (governor, freq, hyper-threading)
- `setup-bench.sh` : script de compilation du langage donné
- `run-bench-1T.sh` : test du langage donné (single-thread)
- `run-bench-MT.sh` : test du langage donné (multi-thread), use `CPU_CORES` variable

Les implémentations dans les différents langages sont stockées dans les sous-répertoires `cfiles` (C), `ffiles` (Fortran), `gofiles` (Go), `javafiles` (Java), `jlFiles` (Julia), `pyfiles` (Python) et `rustFiles` (Rust).

Voici la fiche de résultats pour cette partie:

Time	Langage	programme	Threads	Temps (s)	Energie (W.h)

Pour commençer, nous allons exécuter le code Python numpy de référence avec le protocole de test:

Question 24
Lancer l'expérience à l'aide de la commande :
```
cd ecoconception_logicielle
./run-bench-1T.sh pyfiles/1Dpython_numpy
```
Et regarder le code :
```
vi pyfiles/1Dpython_numpy
```

✘ **Question 3.1.** *Quels changements sont apportés par le protocole ?*
Les temps de réponses sont-ils cohérents avec les précédentes expériences ?

A l'aide des scripts, il est donc possible de lancer des expériences reproductibles dans les langages supportés (voir sous-dossiers).

3.1 Tests single-thread

Dans cette section, quelques langages vont être testés: python, C, julia ... et vous utiliserez les wattemètres de la plate-forme Grid'5000 (cf partie 1) pour récupérer le profil énergétique (consommation électrique pendant l'expérience), le graphique correspondant et la quantité d'enèrgie consommée (W.h).

Question 25
Lancer l'expérience à l'aide de la commande :
```
cd ecoconception_logicielle
./run-bench-1T.sh cfiles/1DC
./run-bench-1T.sh jlfiles/1Djulia
```
Et regarder le code :
```
vi cfiles/main1D_cb.c
vi jlfiles/main1D_cbv3.jl
```

✘ **Question 3.2.** *Est-ce que les programmes donnent les mêmes résultats numériques ?*
Est-ce plus performant que les programmes Python ? Faites une mesure de la consommation électrique des diverses versions des expériences Python, C et Julia.

Remplir la Fiche de résultats (temps CPU et consommation)
Quelles sont les incertitudes sur l'estimation de l'énergie consommée ? Quels sont les différences en termes de performance et de consommation ?
Observez les profils de consommation, qu'en pensez-vous ?
Quel est la consommation à vide (idle) ?
Quel est l'algorithme le plus efficace énergétiquement parlant ?

3.2 Tests MT (hyper-threading)

Dans cette section, les tests seront réalisés en utilisant plusieurs processus en parallèle pour reproduire un cas idéal de parallélisation idéale (zero over-head) et mesurer des différences importantes sur la consommation (charge de 50% ou 100%)

Question 26
Lancer l'expérience à l'aide de la commande :
```
cd ecoconception_logicielle
./run-bench-MT.sh cfiles/1DC
```

✘ **Question 3.3.** *Combien de threads / coeurs CPU sont utilisés ?*
Faites une mesure de la consommation électrique de cette expériences C (MT).
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?
Sachant que ce protocole (MT) effectue 10 itération par thread, le travail réalisé est multiplié par le nombre de processus (= CPU_CORES).
Quel est l'éfficacité du programme 1T et MT ?

Question 27
Lancer l'expérience sans CPU Hyper-threading à l'aide de la commande :
```
cd ecoconception_logicielle
./cpu_ht_off.sh
./run-bench-MT.sh cfiles/1DC
```

✘ **Question 3.4.** *Combien de threads / coeurs CPU sont utilisés ?*
Faites une mesure de la consommation électrique de cette expériences C (MT).
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?

Le script `./run-bench-MT.sh` utilise la variable `CPU_CORES` pour définir le nombre de processus à lancer.

Question 28
Lancer l'expérience avec 8 coeurs utilisés à l'aide de la commande :
```
cd ecoconception_logicielle
CPU_CORES=8 ./run-bench-MT.sh cfiles/1DC
```

✘ **Question 3.5.** *Faites une mesure de la consommation électrique de cette expérience.*
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?
Quel est l'éfficacité du programme MT dans ce cas ?
Pouvez-vous tracer le profil énergétique en fonction de la charge (1T, 8T, 16T) ?

3.3 Tests MT - Python

Dans cette section, des expériences seront réalisées en utilisant les différentes version du code Python (python numpy, numba ou transonic) pour évaluer les gains des accélarateurs transonic et numba.

Question 29
Lancer l'expérience à l'aide de la commande :
```
cd ecoconception_logicielle
./run-bench-MT.sh pyfiles/1Dpython_numba
```

✘ **Question 3.6.** *Faites une mesure de la consommation électrique de cette expérience.*
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?
Comparer avec l'experience C (MT)

Question 30
Lancer l'expérience à l'aide de la commande :
```
cd ecoconception_logicielle
./run-bench-MT.sh pyfiles/1Dpython_transonic
```

✘ **Question 3.7.** *Faites une mesure de la consommation électrique de cette expérience.*
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?
Comparer avec l'experience C et numba (MT)

Enfin si vous avez le temps (numpy est trés lent):

Question 31
Lancer l'expérience à l'aide de la commande :
```
cd ecoconception_logicielle
./run-bench-MT.sh pyfiles/1Dpython_numpy
```

✘ **Question 3.8.** *Combien d'itérations sont réalisées ? Faites une mesure de la consommation électrique de cette expérience.*
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?

✘ **Question 3.9.** *Quels sont vos conclusions sur la comparaison des versions Python ?*

3.4 Tests MT - Autres langages

Dans cette section, des expériences seront réalisées en utilisant les autres langages disponibles (Julia, Java, Fortran...).

Question 32
Lancer les expériences à l'aide des commandes :
```
cd ecoconception_logicielle
./run-bench-MT.sh jlfiles/1Djulia
./run-bench-MT.sh jlfiles/1Djava
./run-bench-MT.sh jlfiles/1Dfortran
...
```

✘ **Question 3.10.** *Faites une mesure de la consommation électrique de ces expériences.*
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation ?
Comparer les portages ? sont-ils équivalents ?

3.5 Conclusion sur les langages

✘ **Question 3.11.** *Quel bilan tirer de cette étude sur les différents langages en terme de performance (CPU) et de consommation ?*

3.6 Etude performance & consommation énergie sur algorithme TDMA optimisé (Bonus)

Comparer les versions de référence et optimisée (`_opt`) pour quelques langages: python transonic, C et julia ...

✘ **Question 3.12.** *Quel est l'impact des optimisations ? quel ordre de grandeur ? Faites une mesure de la consommation électrique de ces expériences (MT).*
Remplir la Fiche de résultats (temps CPU et consommation)
Quels sont les différences en termes de performance et de consommation par rapport aux codes de référence ?

4 Impact de la volumétrie & Scalabilité du programme

Dans cette section, nous allons faire varier la taille du problème (M) pour évaluer la loi de montée à l'échelle (scalabilité) d'une implémentation de l'algorithme TDMA.

Question 33
Modifier le programme `cfiles/main1D_cb.c` pour changer M aux valeurs suivantes: 1000 * 1000, 100 000 puis 10 000 :
`vi cfiles/main1D_cb.c`
Lancer l'expérience à l'aide de la commande :
`./run-bench-MT.sh cfiles/1DC`

✘ **Question 4.1.** *Lancer le programme pour le temps CPU pour M = 1000 * 1000, 100 000, 10 000, 1000.*
Quel est l'impact sur les résultats ?

✘ **Question 4.2.** *Etablir la loi de scalabilité en fonction de la taille du problème (M) : Time vs M*
Quel est le coût CPU (donc la consommation) pour doubler la taille du problème ? pour multiplier par 10 ? Extrapolation: Quel temps CPU faudrait-il pour 10^9 *points,* 10^12 *points (très grande échelle) ?*

✘ **Question 4.3.** *Questions subsidiaires: quelle est la précision suffisante pour cette simulation ?*
Quelle est la meilleure optimisation entre le choix du langage et adapter les paramètres de la simulation ?
Dans quel cas, faut-il priviligier l'une ou l'autre des approches ?

✍ **Note:** Dans cette algorithme itératif (26 itérations), la convergence est déterminée par delta < eps = 1e-9.
Pour avoir un résultat rapide mais approximatif, il est possible de changer le paramètre eps à 1e-6 voire moins pour calculer moins d'itérations et réduire les impacts.

5 Impact de Docker et Intégration continue (EXTRA)

L'utilisation des technologies de container (Docker) et de l'intégration continue sont très 'à la mode' ...

Tous les codes des TPs EcoConception logicielle sont gérés dans un dépot gitlab sur la forge GRICAD (U.G.A).
Ces codes ont été préparés pour fonctionner sur un serveur Ubuntu 20.04 et Docker a été utilisé pour valider le bon fonctionnement (tests d'intégration) sur ubuntu 18, 20 et 22 et debian 10 et 12.

5.0.1 Impact de l'utilisation de Docker

Docker et la procédure d'installation (image Ubuntu + paquets debian + dépendances julia, python) nécessite le réseau : environ 2GB (download)...

```
docker images

REPOSITORY                                                      TAG
IMAGE ID        CREATED          SIZE
ecoinfo_anf                                                     0.2
173821527559    20 hours ago     2.02GB
ubuntu                                                          18.04
3556258649b2    23 months ago    64.2MB
```

Lancer la construction de l'image docker :

```
cd setup/docker_ubuntu20
make
```

✘ **Question 5.1.** *Quelles sont les étapes de construction de l'image (DockerFile) ?*
Quels sont les impacts (réseau, temps CPU, consommation) ?

5.0.2 Impact de l'intégration continue

Gitlab propose une intégration continue basée sur Docker.

Voir le fichier .gitlab-ci.yml:

```
image: ubuntu:18.04

job_test:
    before_script:
        - apt-get update
        - apt-get upgrade -y
        - apt-get install -y locales
        - localedef -i en_US -c -f UTF-8 -A /usr/share/locale/locale.alias en_US.UTF-8
        - export LANG='en_US.UTF-8'
        - export LANGUAGE='en_US:en'
        - export LC_ALL='en_US.UTF-8'
        - cd $CI_PROJECT_DIR
        - cd ateliers/dev
        - bash setup.sh

    script:
        - cd $CI_PROJECT_DIR
        - cd ateliers/dev
```

```
        - bash run_tests.sh > test.log

    when: manual
#     when: delayed
#       start_in: 30 minutes

    artifacts:
        paths:
          - ateliers/dev/test.log
        when: always
```

Voir les jobs sur gitlab: https://gricad-gitlab.univ-grenoble-alpes.fr/ecoinfo/anf2021/-/jobs/

✘ **Question 5.2.** *Quelle est la fréquence de lancement du job ? (18 minutes 25 seconds)*
Quelle est la taille de l'artefact (paquet produit) ? (17 Mb)
Quelle volumétrie représente 1000 téléchargement ? 10 utilisateurs avec des mises à jour tous les mois sur 1 an ?
Quel impact (réseau, CPU) a l'intégration continue si elle était lancée chaque jour ? à chaque commit (10 par jour) ?