Paralléliser en Esope dans Cast3M avec la bibliothèque pthread

Mise en place rapide en Esope

- Mettre l'include `-INC CCASSIS` qui contient les infos sur le nombre de threads `nbthrs`
- Déclarer la source d'interfaçage en `EXTERNAL` (ex : `EXTERNAL chole3i`)
- Déclarer un `COMMON` contenant tous les `TABLEAUX` / `SEGMENTS` que doivent se partager les threads
- Ne pas paralléliser en pthread dans les ASSISTANTS : `CALL OOONTH(ITH)`
    - Appeler la `SUBROUTINE` qui fait le travail sans passer par le `COMMON` sinon il y aura des soucis
- Initialisation des threads et verrouillage : `CALL THREADII`
    - **ATTENTION :** Ne remplir le `COMMON` qu'après `CALL THREADII`
- Déterminer le nombre de threads utiles `NBTHR` (Boucles avec moins d'éléments que de threads)
- Boucle `ith` sur les threads de `2` à `NBTHR`
    - Lancement du travail : `CALL THREADID (ith, chole3i)`
- Lancement du thread maître séparément en dernier : `CALL chole3i(1)`
- Boucle `ith` sur les threads de `2` à `NBTHR`
    - Fin du travail : `CALL THREADIF(ith)`
- Stopper les threads : `CALL THREADIS`

Important

- Il n'est pas toujours possible de faire des `WRITE` de `REAL*8` dans les threads…
    - `WRITE` fonctionne dans le thread maitre
    - `PRINT` semble fonctionner dans les threads
- Il ne faut pas faire de **`SEGINI`**, **`SEGACT`**, **`SEGDES`**, **`SEGADJ`**, **`SEGSUP`** dans les threads
    - Créer / Activer tous les SEGMENTS avant l'appel et les laisser actifs durant le travail (On peut utiliser les SEGMENTS actifs sans soucis).
    - Désactiver / Supprimer les SEGMENTS après l'appel.
    - `OOV(1)` : Accessible par tous les threads. Indicateur de l'utilisation d'un élément de SEGMENT
- Il ne faut pas créer de tableaux dynamiques
    - `REAL*8 XTOTO(NN)` ➔ `XTOTO` et `NN` doivent arriver en argument ou en `COMMON`. Un segment contenant `XTOTO` doit être préparé avant.

Exemple de mise en application

```
      SUBROUTINE TOTO
C Préparation et lancement de TUTUi en //
-INC CCASSIS
      EXTERNAL TUTUi
      COMMON/tutuc/NBTHR,IPOIN1,IPOIN2,...

C Initialisation des threads
      ITH = 0
      IF (NBESC.NE.0) CALL OOONTH(ITH)
      IF((NT1.LE.1).OR.(NBTHRS.EQ.1).OR.
      (ITH.GT.0)) THEN
        NBTHR = 1
        BTHRD = .FALSE.
      ELSE
        BTHRD = .TRUE.
        NBTHR = MIN(NT1, NBTHRS)
        CALL THREADII
      ENDIF
      NBTHR = MIN(N1, NBTHRS)
      IF((NBTHR.GT. 1).AND. BTHRD) THEN
C Lancement du travail dans les threads
        DO ith=2,NBTHR
          CALL THREADID(ith, TUTUi)
        ENDDO
        CALL TUTUi(1)
C Fin du travail dans les threads
        DO ith=2,NBTHR
          CALL THREADIF(ith)
        ENDDO
C       En multithread il peut y avoir
C       n'importe quoi dans OOV(1)
        OOV(1) = 0
      ELSE
        CALL TUTUi (1)
      ENDIF
      IF (BTHRD) CALL THREADI
      END
```

```
      SUBROUTINE TUTUi(ITH)
C Source qui peut être lancée en //
      COMMON/tutuc/NBTHR,IPOIN1,IPOIN2,...

C PRIVILEGIER LA CONTINUITE MEMOIRE
      IRES  = MOD(N1,NBTHR)
      IF (IRES .EQ. 0) THEN
        ILON  = N1 / NBTHR
        IDEB  = (ithr -1)* ILON + 1
      ELSE
        IF (ithr .LE. IRES) THEN
          ILON  = (N1 / NBTHR) + 1
          IDEB  = (ithr -1)* ILON + 1
        ELSE
          ILON  = N1 / NBTHR
          IDEB    =  (IRES  *  (ILON+1))  +
      (ithr-IRES-1)* ILON + 1
        ENDIF
      ENDIF
      IFIN  = IDEB + ILON - 1
      DO J1 = IDEB,IFIN
        ...
      ENDDO

C SANS CONTINUITE MEMOIRE :
      DO J2 = ith,N1,NBTHR
        ...
      ENDDO
      END
```

Paralléliser dans Cast3M avec les ASSISTANTS

Sources intéressantes :

- `chkesc.eso` : Vérifie si les opérandes contiennent des `TABLES ESCLAVES` et si une condensation de ces dernières est nécessaire (On y passe seulement si « `OPTI PARA VRAI ;` »).
- `etg.eso` : OPERATEUR permettant de lancer la condensation d'une `TABLE ESCLAVE`
- `funobj.eso` : Effectue la condensation par tournoi d'une `TABLE ESCLAVE` avec un traitement particulier pour les `FLOTTANTS` et les `LOGIQUES`
- `assist.eso` : Prépare la panoplie complète des SEGMENTS (vierges) de `CCASSIS.INC` en attente de l'écriture des résultats par `nouins2.eso`
- `nouins2.eso` : écriture des résultats de chacun des ASSISTANTS dans le SEGMENT `NESRES` ➔ `MESRES` ➔ `esrees(iop)` où `iop` est le numéro de l'assistant
- `acctab.eso` : Attend que l'objet `ESCLAVE` soit disponible avant de le récupérer

Faire un point d'arrêt :

Un point d'arrêt consiste à attendre que tous les ASSISTANTS aient terminés leur travail et écrits le résultat dans les SEGMENTS (`NESRES`, `MESRES`).

Afin de mettre un point d'arrêt sur l'utilisation d'un objet parallélisé (`TABLE` de sous-type `ESCLAVE`), il faut regarder l'état du `LOGIQUE LOREMP` dans le `SEGMENT MESRES` (Voir `CCASSIS.INC`). Tant qu'il n'est pas à `VRAI`, il faut refermer le SEGMENT puis le rouvrir dans une boucle (exemple concret d'utilisation dans `lirabj.eso` et `liresc.eso`).

Remarque

- Un ménage effectué peu après le lancement de l'opération sur les ASSISTANTS peut supprimer un SEGMENT de travail (`MESRES`, `NESRES`), il faut penser à ne pas l'effacer durant le ménage (voir `menag6.eso`)
- D'après P. VERPEAUX, la cohabitation des ASSISTANTS avec les PTHREAD n'est pas un problème pour des opérations unitaires qui utiliseraient les 2 méthodes en même temps (voir `threadid.c` dans `/u2/castem/castem.arc`). En pratique les 2 niveaux de parallélisation utilisés ensembles sont contreproductifs…
- Veiller à ce que les sources qui sont appelées avec les ASSISTANTS ferment bien tous les segments qu'elles ont ouverts sans quoi il faut s'attendre à des `DEADLOCK`.
- Si une source a déjà activé les THREADS (`threadii`) et que cette opération est refaite plus loin, cela provoque un `DEADLOCK` non signalé par une erreur `GEMAT` (on est tout simplement bloqué…)

DEADLOCK

Source qui affiche le message

`ooooddl.eso` (Source ESOPE)

Blocage sans DEADLOCK

Il arrive parfois d'être bloqué sans que la main ne soit rendue par Cast3M et pour autant il n'y a pas de `DEADLOCK` affiché. Dans ce cas, il suffit de récupérer la main sur le processus bloquée dans gdb et de demander au thread où ils sont en ce moment :

```
ps -edf | grep cast
CB215821 25759 25725 0 14:48 pts/5 00:00:00 ./cast
/usr/bin/gdb -p 25759
thread apply all where
```

Comprendre le message affiché

En travaillant avec les ASSISTANTS, il est courant de faire face à des `DEADLOCK`. Cette situation survient lorsque tous les ASSISTANTS sont bloqués. Tant qu'au moins 1 ASSISTANT travaille, on n'est pas considéré en `DEADLOCK`. L'exemple ci-dessous permet de comprendre comment lire le message d'erreur que renvoie Cast3M :

```
0--- GEMAT ERROR --- SEGACT ,
 ---                      0     POSLOG 94 IPILOC
0--- GEMAT ERROR --- SEGACT ,
 ---                      1     NOUINS 243 MESINS
0--- GEMAT ERROR --- SEGACT ,
 ---                      2     NOUINS 243 MESINS
0--- GEMAT ERROR --- SEGACT ,
 ---                      3     NOUINS 243 MESINS
0--- GEMAT ERROR --- SEGACT ,
 ---                      4     NOUINS 243 MESINS
0--- GEMAT ERROR --- SUBROUTINE  : POSLOG
 ---                  INSTRUCTION : 94
 ---                  SEGACT , IPILOC
 ---                2100148     DEADLOCK DETECTE
```

Numéro de l'ASSISTANT (0 = maître)

Nom du SEGMENT non activable sur cet ASSISTANT

Numéro de l'instruction à rechercher dans le FORTRAN .f

SUBROUTINE dans laquelle le DEADLOCK est rencontré

Informations de la dernière instruction avant le DEADLOCK

Dernière action tentée

Numéro du POINTEUR dans la dernière instruction bloqué
`'OPTI' 'SURV' 2100148` **; permettra de suivre ce POINTEUR**

Débusquer l'anomalie

Afin de trouver la source qui a créé les SEGMENTS (`SEGINI`) sans les désactiver (`SEGDES`), voici la procédure à suivre :

- Ouvrir la source traduite en FORTRAN que l'un des ASSISTANTS mentionne : `nouins.f` sur cet exemple
- Rechercher l'instruction `243` qui ne parvient pas à s'exécuter à cause d'un verrouillage sur ce SEGMENT
- Identifier l'instruction ESOPE correspondante (dans `nouins.eso`)
- Afficher le numéro du SEGMENT en question (`PRINT *, 'DANS NOUINS.ESO',MESINS`)
- Compiler, faire l'édition des liens et relancer le cas-test
- Surveiller le SEGMENT en question en ajoutant : `OPTI SURV NUMERO ;`
- Relancer le cas-test

GEMAT ERROR

Obtenir l'arborescence d'appel

- lancer le cas test problématique avec `gdb (castemd cas-test.dgibi)`
- faire `break oooerr_` pour s'arrêter dès que le message GEMAT est affiché (`stop in` sur IBMRS6000)
- faire `where` pour obtenir l'arbre d'appel (`backtrace`)

ARGUMENT(S) ELEMENT(S) DE SEGMENT

Cela signifie que dans l'arborescence d'appel il y a dans un argument un élément de segment. Il suffit de rechercher la SUBROUTINE qui dans ses arguments possèdes des `OOO(…)`.

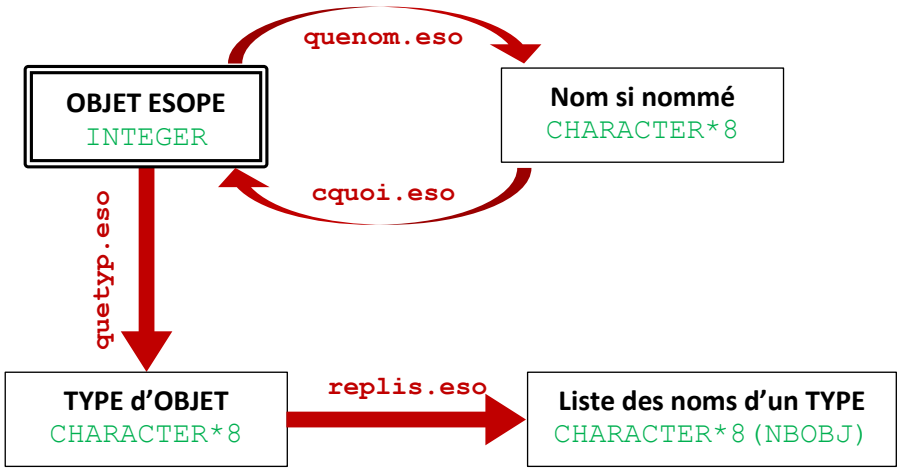
SUBROUTINES utiles pour travailler sur les OBJETS en Esope

`quenom.eso` ➔ Donne le NOM (`CHARACTER*8`) du dernier objet lu (voir `lirobj.eso`)

`cquoi.eso` ➔ Donne le POINTEUR (`INTEGER`) d'un objet dont on connait le NOM (`CHARACTER*8`)

`quetyp.eso` ➔ Donne le TYPE (`CHARACTER*8`) du premier objet dans la PILE sans le retirer

`replis.eso` ➔ Répertorie tous les NOMS d'un TYPE d'objet dans un tableau de `CHARACTER*8`



`rmpgbn.eso` ➔ Remplace dans la pile GIBIANE un objet par un autre. Remplace également selon la valeur d'un `LOGIQUE` l'objet de manière permanente (pour ne pas avoir à refaire systématiquement la fusion des `TABLES ESCLAVES`)

`poscha.eso` ➔ Position d'un `MOT` dans la pile

`poslog.eso` ➔ Position d'un `LOGIQUE` dans la pile de travail

`posree.eso` ➔ Position d'un `FLOTTANT` dans la pile de travail

`trbac.eso` ➔ Permet de renvoyer l'ECHO de la ligne GIBIANE en cours de traitement

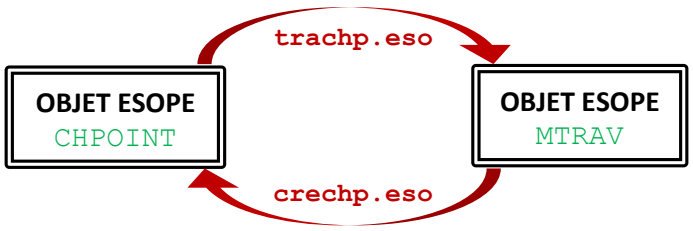
`typfil.eso` ➔ Etablit la correspondance entre un TYPE d'objet et son numéro de PILE

`tassp2.eso` ➔ Permet entre autre de protéger des OBJETS que l'on ne souhaite pas voir supprimés par `MENAGE` **Attention**, il faut s'assurer qu'après un `SAUV/REST` cela fonctionne toujours !

`GIBI.ERREUR` ➔ Contient tous les messages d'erreurs dans toutes les langues (FRAN et ANGL actuellement). Pour l'évoluer sur `semt2` avec `dial20` il faut le renommer `gibi.erreur` en minuscule (C'est historique).

`trachp.eso` ➔ Convertit un `CHPOINT` en SEGMENT de travail `MTRAV`

`crechp.eso` ➔ Convertit un SEGMENT de travail `MTRAV` en `CHPOINT`. Il faut compléter le chapeau après.



SEGDES, SEGACT et SEGSUP d'un SEGMENT quelconque

Pour manipuler un SEGMENT dont le type n'est pas connu : (voir `menag4.eso`) :

- Définir un SEGMENT de travail : `SEGMENT ISEG(0)`
- Affecter ce SEGMENT d'un POINTEUR : ISEG = IPOIN
- Manipuler ce SEGMENT : `SEGSUP,ISEG` ou `SEGACT,ISEG` ou `SEGDES,ISEG`

Créer un nouvel opérateur dans Cast3M

Modifier `pilot.eso`

1ère couche : SUBROUTINE sans ARGUMENTS

Râteau pour les différentes options (un seul CALL LIRMOT de préférence)

Lecture des OBJETS GIBIANES en ESOPE

LIROBJ

REDUAF (si lecture d'un MMODEL et MCHAML sauf pour l'opérateur PROI)

LIRREE

LIRENT

LIRLOG

LIRMOT

Ecriture des OBJETS GIBIANES en ESOPE

ECROBJ

ECRREE

ECRENT

ECRLOG

ECRCHA

2ème couche : SUBROUTINES avec ARGUMENTS

Pour être appelé directement en ESOPE depuis une autre SUBROUTINE

Fait le travail, éventuellement en parallèle

Revoie un code de retour (`IRET`)

`IRET = 1` si l'opération a été réalisée correctement

`IRET = 0` si l'opération a rencontré une erreur

Rédaction d'une NOTICE pour un opérateur / procédure

- 1ère ligne : `$$$$ LE_NOM  NOTICE` (Compter 10 caractères entre le dernier `$` et le `N` de `NOTICE`)
- Dernière ligne : `$$$$`
- Ligne séparatrice indiquant que le texte qui suit est en Français / Anglais :
  ```
  FRAN=======================================================
  ANGL=======================================================
  ```
- Découpage en CHAPITRES et PARTIES des notices pour un affichage plus clair avec INFO (Voir mode.notice) :
  ```
  CHAP{Nom de mon premier CHAPITRE}
  PART{Nom de la 1ère sous-partie de ce CHAPITRE}
  ```
- Si des Opérateurs / Procédures sont en relation, bien penser à les mettre dans la rubrique `Voir aussi`

Séminaire SEMT : Bonnes pratiques en FORTRAN et DEBUG avec Valgrind

- **ATTENTION** : `READ` dans un fichier `ASCII` le comportement diffère suivant la plateforme :
  - IBM Aix : Les caractères `CR` et `LF` sont laissés comme tel, `READ` peut échouer
  - Linux/Windows: Les caractères `CR` et `LF` sont remplacés par un espace `' '`
- Division entière `2x` plus rapide que la division par un flottant
- Pour diviser un tableau par une constante `C1`, il faut faire au préalable `C2=1.D0/C1` dans une variable et multiplier les valeurs du tableau par `C2`
- Puissances flottantes `16x` plus lentes que les puissances entières
  - Remplacer `x**(3.D0/2.D0)` par `SQRT(x**3)` est `4,5x` plus rapide
- Privilégier la continuité de la mémoire dans la lecture des tableaux (`2,5x` plus rapide dans le bon sens)
  - `Tab1(I,J,K)` est continue en mémoire d'abord sur `I`, puis sur `J` et enfin sur `K`
- Options de compilation recommandées en mode RELEASE :
  - `-funroll-loops`
  - `-funroll-all-loops`
- Options de compilation recommandées en mode DEBUG :
  - `-fbacktrace` : Permet de remonter l'arbre d'appel
  - `-Wunused-parameter` : Pour la lisibilité des sources
  - `-Wunderflow`
  - `-fbounds-check` : Vérifie les bornes des tableaux FORTRAN. Ne fonctionne pas avec l'ESOPE (à cause des `OOA(… OOO(…,2**30))`
  - `-O0 -g`
  - `-ftrapv` : Le mode de capture des signaux est verbeux
  - `-ffpe-trap=invalid,zero,overflow,denormal`
  - `-fimplicit-none` : La plupart des codes ne le supporte pas du tout (ce n'est pas dans la spécification de Cast3M)

- Utilisation de VALGRIND :
  - `Memcheck` : Vérification de la mémoire RAM : `--tool=memchek`
  - `Cachegrind` : Vérification de la mémoire CACHE : `--tool=cachegrind`
  - `Massif` : Profiler d'allocation mémoire : `--tool=massif`
  - `Helgrind` : Détecteur d'erreur dans les THREAD : `--tool=helgrind`
  - `DRD` : Détecteur d'erreur dans les THREAD : `--tool=drd`
  - `Iogrind` : Vérifications des Entrées / Sorties
  - `Wine` : Considéré comme illégal par Microsoft (Reverse Engineering sur les librairies)

- VALGRIND avec Cast3M :
  - Mettre `ZERMEM=VRAI` dans la variable d'environnement `ESOPE_PARAM`
  - Utiliser `ooozmr.eso` et non pas `ooozmr.c` (Sinon la mémoire est déclarée non initialisée…)
  - Exécuter VALGRIND sur `./cast`

Analyse des performances du code : PERF

Lancement de l'analyse : `perf record castem Test.dgibi`

Dans un autre terminal : `perf top` donne d'autres infos que top (activité processeur en direct par `SUBROUTINE`)

Post-traitement : `perf report`