code_aster HPC
Généralités

Nicolas Sellenet

EDF R&D - Saclay

Sommaire

- Introduction
- Déroulé d’une étude : Points chauds
- Points chauds du calcul
- Ce qui consomme du temps et de la mémoire
- Zoom sur le temps CPU
- Zoom sur la consommation mémoire
- Parallélisme MPI

Introduction

- Prérequis
  - Vous savez déjà réaliser des calculs avec code_aster
  - Vous avez déjà des études robustes et fiables

- Qu’est-ce qu’on entend par performances ?
  - Amélioration de la vitesse de convergence pour une étude non-linéaire
  - Diminution du temps de restitution de l’étude
  - Diminution de la consommation mémoire

Déroulé d'une étude : Points chauds

- Grandes étapes d'une étude
  - Lecture du maillage
  - Affectation du modèle
  - Affectation des chargements
  - Affectation de caractéristiques élémentaires
  - Calcul
  - Post-traitement

- Vous avez la main sur le calcul et le post-traitement
  - Les autres étapes ne peuvent pas espérer de gains

Points chauds du calcul

- Calculs élémentaires et loi de comportement
  - Intégration du comportement
  - Calcul des contributions élémentaires au problème global
- Assemblage matrice et second membre
  - Insertion des contributions élémentaires dans le problème global
- Factorisation matrice
- Inversion matrice
- Post-traitement

Ce qui consomme du temps et de la mémoire

- Consommation de temps CPU
  - Calculs élémentaires et intégration de la loi de comportement
  - Factorisation matrice
  - Inversion matrice (dépend du solveur linéaire)

- Consommation de mémoire
  - Fin de l'assemblage de la matrice : pic de consommation mémoire Aster
  - Factorisation de la matrice : pic de consommation mémoire solveur linéaire (MUMPS)

Zoom sur le temps CPU (1/2)

- Commande `DEBUT`, mot-clé facteur `MESURE_TEMPS`
  - `NIVE_DETAIL = 2`, affiche des détails sur les consommations de temps (en fin de commande)

```
#1      Resolution des systemes lineaires          CPU (USER+SYST/SYST/ELAPS):     0.54     0.03     0.27
#1.1    Numerotation, connectivite de la matrice   CPU (USER+SYST/SYST/ELAPS):     0.02     0.00     0.01
#1.2    Factorisation symbolique                   CPU (USER+SYST/SYST/ELAPS):     0.36     0.02     0.17
#1.3    Factorisation numerique (ou precond.)      CPU (USER+SYST/SYST/ELAPS):     0.13     0.00     0.07
#1.4    Resolution                                 CPU (USER+SYST/SYST/ELAPS):     0.03     0.01     0.01
#2      Calculs elementaires et assemblages        CPU (USER+SYST/SYST/ELAPS):     0.08     0.00     0.04
#2.1    Routine calcul                             CPU (USER+SYST/SYST/ELAPS):     0.06     0.00     0.03
#2.1.1  Routines te00ij                            CPU (USER+SYST/SYST/ELAPS):     0.05     0.00     0.03
#2.2    Assemblages                                CPU (USER+SYST/SYST/ELAPS):     0.02     0.00     0.01
#2.2.1  Assemblage matrices                        CPU (USER+SYST/SYST/ELAPS):     0.02     0.00     0.01
#2.2.2  Assemblage seconds membres                 CPU (USER+SYST/SYST/ELAPS):     0.00     0.00     0.00
```

- Temps “elapse” : temps réellement attendu par l’utilisateur
- Temps Système donne une information intéressante sur
  - Le temps passé à écrire et lire sur le disque
  - Temps système important : peut évoquer une mémoire insuffisante

Zoom sur le temps CPU (2/2)

- Temps système important
  - Se reporter à la fin du fichier .mess
    - `APPELS AU MECANISME DE LIBERATION : 3`

- `INFO = 2` dans les commandes de calcul
  - Chaque commande a sa politique
  - Souvent : affichage de détails concernant le solveur linéaire (ex : MUMPS)

- `STAT_NON_LINE` :
  - Affichage des temps après chaque itérations
  - Affichage global à l'issue du calcul

Zoom sur la consommation mémoire

- Uniquement à la fin d'une commande

```
# Mémoire (Mo) :   541.54 /   535.01 /    38.83 /    33.18 (VmPeak / VmSize / Optimum / Minimum)
# Fin commande No : 0004   user+syst:        0.01s (syst:        0.00s, elaps:        0.00s)
# --------------------------------------------------------------------------------
```

- Valeur importante : VmPeak
  - Valeur maximum atteinte par le processus (exécutable + librairie + allocations + …)
- Optimum
  - Valeur maximum utilisée pour l'allocation d'objet côté code_aster
- Minimum
  - Valeur en dessous de laquelle le calcul ne pourrait pas avoir lieu

Parallélisme MPI (1/2)

- Idée de base pour améliorer les performances
  - Gain dès lors qu'on choisit un nombre de CPU > 1 dans astk
  - Calculs élémentaires et intégration du comportement parallélisés

- Choix basique pour la répartition de la charge : "SOUS_DOMAINE"
  - Il en existe d'autres :
    - "GROUP_ELEM"
    - "SOUS_DOM.OLD"
  - En théorie, c'est "SOUS_DOMAINE" le plus performant

- Déchargement possible du proc 0 dans le cas "SOUS_DOM.OLD"
  - Dans le cas d'un chargement dualisé type AFFE_CHAR_MECA

Parallélisme MPI (2/2)

- Changement du solveur linéaire
  - Par défaut `MULT_FRONT` : solveur séquentiel
  - Choisir un solveur parallèle, par exemple : `MUMPS` ou `PETSC`
  - Cf. exposés dédiés

- `MUMPS` : passe-partout ; `PETSC` : massivement parallèle

- Performances variables
  - Cible maximum : 32 processeurs
  - Cible moyenne : entre 8 et 16 processeurs

Conseils généraux

- Pour améliorer les temps de retour, pensez à :
  - Réduire la taille de vos bases (commande `DETRUIRE`) dans les cas volumineux
  - Utiliser l'`ARCHIVAGE` (dans `STAT_NON_LINE`) pour limiter :
    - 1) les temps de sauvegarde
    - 2) les temps de recopie de la base

- Donnez assez de mémoire à votre calcul
  - Sinon déchargement sur disque