



# CIDM - Compilation

## Avancée

(CA 2023-2024)

## TD3

### Reconnaissance des appels MPI et manipulation du CFG

antoine.capra@eviden.com  
van-man.nguyen@eviden.com  
patrick.carribault@cea.fr

## I Environnement

Comme pour les séances précédentes, il est nécessaire de charger l'environnement du compilateur GCC 12.2.0. en utilisant la commande suivante :

```
Loading Binutils 2.37 environment
Loading GCC 12.2.0 environment
Loading MPI environment
$
```

De plus, les données relatives à ce TD sont disponibles dans le répertoire suivant :

`/home/patrick.carribault/LOCAL/TDs/TD3/`

## II Détection d'appels MPI

Le but de cette partie est de reconnaître les appels MPI, notamment ceux spécifiés dans le fichier de définitions, et de modifier le CFG. Vous pouvez partir des plugins du TP précédent pour réaliser les questions suivantes.

**Q.1:** Dans le répertoire *CODE*, le fichier **plugin\_TP3\_1.cpp** reprend la correction du TD2 (avec l'ajout d'un **enum** et d'un tableau global incluant le fichier **MPI\_collectives.def** qui sera utile dans les questions suivantes). En vous aidant de ce fichier, écrire une fonction prenant en argument une instruction (i.e., un *statement gimple* **gimple \* stmt**) et affichant le nom de la fonction appelée si celle-ci est une fonction MPI (fonctions préfixées par "MPI\_"). Un fichier **test2.c** vous est fourni pour tester votre plugin.

**Q.2:** Dans le répertoire *CODE*, un fichier **MPI\_collectives.def** vous est fourni. Ce fichier définit les fonctions MPI intéressantes dans le cadre de ce TD. Le code présent dans le fichier **plugin\_TP3\_1.cpp** permet de construire un *enum* avec les identifiants des collectives ainsi qu'un tableau contenant les noms de chaque fonction MPI présente dans le fichier de définition.

Servez-vous de ce tableau de nom de fonctions et de la fonction écrite à la question précédente pour afficher uniquement le nom des fonctions MPI appelées présentes dans le fichier de définition.

**Q.3:** Modifiez la fonction pour que celle-ci renvoie la valeur d'*enum* correspondant à la fonction MPI du fichier de définition reconnue par l'analyse de la question précédente.

### III Stockage des informations MPI

**Q.4:** Les structures `basic_block` manipulées permettent de stocker des informations spécifiques à la passe courante. Trouvez le champ de la structure `basic_block` qui permet de stocker des données temporaires.

**Q.5:** Utilisez ce champ pour stocker l'identifiant de la fonction MPI reconnue pour chaque bloc de base. Cela fonctionne-t-il? Créez une fonction permettant de remettre à zéro les champs `aux` de chaque bloc de base à la fin de la passe. Cela fonctionne-t-il? Qu'en déduire?

### IV Affichage des informations MPI

**Q.6:** Modifiez l'écriture du fichier graphviz pour introduire, dans le label d'un bloc, le nom de chaque fonction MPI reconnue dans ce bloc. Que remarquez-vous sur le fichier *test2.c*?

**Q.7:** Écrire une fonction prenant en argument une fonction (`function * fun`) et vérifiant s'il existe au moins un bloc contenant au moins deux appels MPI reconnus.

### V Modification du CFG

**Q.8:** Pour simplifier l'analyse que nous allons réaliser dans les prochains TPs, nous souhaitons séparer les blocs de base contenant plusieurs appels MPI reconnus. La fonction *split\_bloc* permet de séparer un bloc en deux "nouveaux" blocs. Trouvez la définition de cette fonction. Utilisez cette fonction pour séparer un bloc contenant plusieurs appels MPI en différents blocs contenant chacun un seul appel de fonction MPI. Vous pouvez produire un nouveau fichier graphviz pour vérifier que l'opération de *split* s'est bien déroulée.