# M2 - Compilation Avancée

(COA 2016-2017)

# TD7

Transformation de codes & bibliothèque externe

julien.jaeger@cea.fr
patrick.carribault@cea.fr

## I Prise de trace basique

Ce TD reprend les grandes lignes du début du projet en proposant l'implémentation d'un instrumenteur dynamique en transformant le code pendant la compilation.

**Q.1:** Adapter le code de la fin du TD5 pour créer et garder un bitmap pour les PDFs itérées de chaque ensemble de Basic blocks contenant la même collective MPI.

**Q.2:** Nous allons maintenant créer un nouveau noeud GIMPLE que nous allons insérer avant chaque appel MPI problématique pour afficher, lors de l'exécution du programme, le nom et la ligne de l'appel.

- La fonction **gimple_build_call** crée un noeud GIMPLE d'appel de fonction. Elle prend en argument la déclaration de la fonction à créer, et les arguments de la fonction, sous représentation intern d'arbre. Voir le fichier *gimple.h*.
- La fonction **printf** que nous allons utiliser est déjà prédéfini dans la représentation interne de GCC. Sa représentation se trouve dans le tableau `built_in_decls`, à la cellule pointée par `BUILT_IN_PRINTF`.
- Il nous reste à construire les arguments de la fonction. La séquence de code ci-dessous permet de construire la représentation de chaîne de caractères, à utiliser comme argument pour la construction d'un nouveau noeud gimple.
- Une fois le nouveau noeud GIMPLE créé, il faut l'insérer dans le CFG. Voir le fichier *gimple-iterator.h*.

Pensez que vous pouvez observer la sortie graphviz d'un *printf* pour comprendre la hiérarchie des objets gimple et opérandes.

```
{
  tree mystring_tree      = fix_string_type( build_string (strlen(<mystring> + 1, <mystring>) ) );
  tree mystring_type      = build_pointer_type( TREE_TYPE (TREE_TYPE (mystring_tree) );
  tree mystring_args_tree = build1( ADDR_EXPR, mystring_type, mystring_tree);
}
```

**Q.3:** Dans le répertoir code, un fichier **checking_collectives.c** est fourni. Il contient une fonction permettant de vérifier à l'exécution, avant l'appel d'une collective MPI, si celle-ci va générer un deadlock. Dans ce cas, la fonction affiche un message d'erreur avant de quitter le programme proprement.

Au lieu d'insérer un appel à printf, nous allons insérer un appel à cette fonction avant chaque appel MPI pouvant poser problème. Construire les arguments de cette fonction, puis construire la déclaration de cette fonction en gimple pour pouvoir l'insérer de la même façon que la fonction printf dans la question précédente.

(Voir les fonctions *build_fn_decl* et *build_function_type_list* pour la construction de l'appel de fonction, et la fonction *build_int_cst* pour la construction d'un argument de type **int**