Projet PPN

« TASKIFICATION »

Découpage et Planification

Avant-Propos

- Encadrement du projet :
    - **Jean-Baptiste Besnard : jbbesnard@paratools.fr (Principal)**
    - **Julien Adam adamj@paratools.fr (Secondaire)**

- **Serveur Discord** à disposition pour un travail collaboratif étudiant<-> encadrant ainsi qu'entre étudiants. Disponible à l'adresse **https://discord.gg/VvXtjQ** *(nécessite un compte Discord)*

- Réunion Visio-conférence via la plateforme **Zoom** via l'adresse **https://zoom.us/j/4502383189** pour un échange plus approfondi. N'hésitez pas à demander un rendez-vous !

- Le sujet est vaste, et il y a beaucoup de choses à faire. Vos encadrants seront là pour vous guider, sachez les solliciter au bon moment ! Les moyens sont mis en place pour que vous puissiez les contacter facilement. Il n'y a pas de pièges ni de mauvaises questions.

Avant-Propos

- C’est un sujet compliqué qui touche au programme de **deuxième année !** Il est normal que cela vous paraisse au début assez difficile.

- Nous sommes disponibles pour vous aider il faudra nous contacter pour les questions

- N’attendez pas la fin pour commencer car le sujet demande un peu de recul

- Nous sommes conscients de la difficulté, votre capacité à chercher des solutions au problème est prise en compte !

Introduction

- Le but est la mise en place d'un plugin dans le compilateur CLANG (LLVM) pour identifier les appels de fonctions candidats à la « taskification » (fonction pures)

- Pour ce faire, il faudra mettre en place la structure de base d'un plugin clang

- Définir ce qui distingue les fonctions candidates à identifier

- Implémenter ce support dans le plugin

- Le valider sur un cas de parallélisme producteur / consommateur.

Méthodologie

- **Tous les travaux se feront sous GIT**
- **Le code est en C++, avec un Makefile**
- **Bien que le plugin soit en C++ les programmes cibles seront considérés être en C**

Le Compilateur en Général

- https://fr.wikipedia.org/wiki/Compilateur

- Livre de référence (pour aller plus loin): Compilateurs : principes, techniques et outils, Sethi et Ullman (a.k.a « DragonBook »)

Faire un Plugin LLVM

- https://clang.llvm.org/docs/ClangPlugins.html

- Faire un plugin:
  https://www.youtube.com/watch?v=pdxlmM477KY (voir sous-titres au besoin)
  https://youtu.be/E6i8jmiy8MY (vous pouvez mettre sous-titres en Fr/Eng)

- Connaissances plus générales autours de CLANG:

  - https://www.youtube.com/playlist?list=PLaQBLbMIApyJYNTyRtBFm-gE_-_eapZTt

La Notion de Fonction Pure

- https://www.youtube.com/watch?v=dZ41D6LDSBg
- https://fr.wikipedia.org/wiki/Fonction_pure

Les Étapes

1. **Définir ce qu'est une fonction pure, en particulier lister les cas qui ne sont pas des fonctions pures;**

2. **Se familiariser avec les plugins LLVM en se basant sur des exemples et en mettant en place une première version de plugin (source et makefile) à partir des ressources sur Internet;**

3. **Implémentation dans le plugin (par adaptation de l'exemple) de la détection des fonctions pures (par détection des fonction qui ne le sont pas) on parcourra alors l'AST;**

4. **Si la progression le permet, ouverture sur MPI avec la mise en place de messages RPC pour un parallélisme distribué de ces fonctions identifiées (la partie MPI sera fournie).**

1 - La Fonction Pure

- **Comprendre ce qu'est une fonction pure;**
- **Expliquer pourquoi elle a des avantages pour la parallélisation des codes;**
- **Lister les cas qui rendent une fonction non-pure EN C. Pour chacun de ces cas faire un petit exemple de code qui sera à conserver (pour ensuite valider l'outil)**

2 - Exemple de Plugin LLVM

En suivant un tutoriel, la documentation, ou en partant d'un exemple existant il faudra:

- **Mettre en place un « squelette » de plugin**
- **Expliquer le fonctionnement global d'un plugin**
- **Présenter comment l'analyse de détection des fonction pure sera implémentée**

3 - Implémentation du détecteur de fonctions Pure

En poursuivant avec l'exemple il faudra:

- **Implémenter une analyse qui liste les fonction pures**
- **Si possible, permettre de lister sur la sortie du programme la raison de leur « non-pureté »**
- **Valider sur des exemple de programme correspondant aux cas identifiés**

4 - Validation sur un Exemple

Le détecteur de fonction pure sera utilisé pour paralléliser un code simple en MPI en utilisant des RPC (ou active messages). Les propriétés des fonctions pure permettons leur execution distante en parallèle.

- **La partie MPI vous sera fournie**
- **L’exemple de code « de base » en MPI sera fourni**
- **Avec l’aide de l’analyse une architecture basée sur les RPCs sera proposée (paramètres et valeur de retour)**
- **On comparera enfin l’efficacité des deux approches**