

Support de MPI/OpenMP et de la vectorisation dans Verificarlo

Master Calcul Haute Performance et Simulation

Hery ANDRIANANTENAINA

Ali LAKBAL

Nicolas BOUTON

Encadrant: Eric PETIT

Année 2020-2021

Compilateur de base pour verificarlo

- CLANG
- LLVM

Domaine d'utilisation de verificarlo

Verificarlo permet par instrumentation des opérations flottantes, de pouvoir déboguer les erreurs, dû à la précision machine.

Vectorisation dans le calcul scientifique

Jeux d'instruction

- 128 = sse
- 256 = avx
- 512 = avx512

Compilation

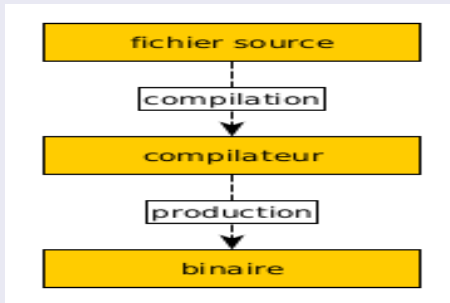


Figure – Fonctionnement de base d'un compilateur

Compilation pour verifcarlo

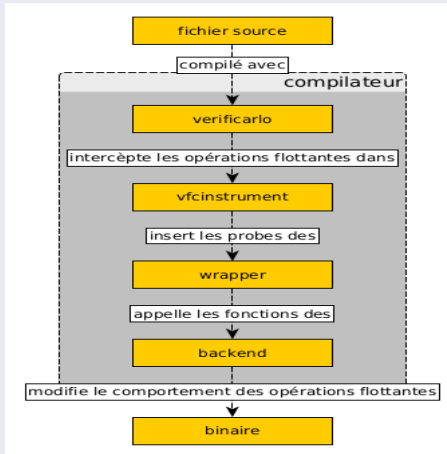


Figure – Fonctionnement de verifcarlo

Définition de certains termes techniques

- probes : Les probes sont des fonctions implémenté dans vfcwrapper qui est linker avec le programme par la partie compilation de verifcarlo.
- backend : Dans le cadre de verifcarlo, c'est la/les librairie(s) dynamique(s) qui seront appelées par le wrapper dans les probes. Dans le cadre d'un compilateur c'est la dernière phase qui descend de la représentation intermédiaire vers le binaires
- wrapper : Ce sont des fonctions qui enveloppent l'appel à d'autres fonctions.
- link : Il s'agit de la phase de compilation qui consiste à aller chercher toute les librairies externes appelé par l'application pour les liées au programme utilisateur afin de résoudre les références non défini.
- sérialisation : Dans le contexte de l'utilisation de vecteur il s'agit d'exécuter en séquence les éléments du vecteur.

Notion indispensable pour le parallélisme

- Système à mémoire partagée
 - SMP
 - NUMA
- Système à mémoire distribuée
- Thread ou flot d'exécution
- Processus parallèle

Présentation d'open MPI

- Installation : source :
<https://www.open-mpi.org/software/ompi/v4.1/>
- Configuration : `./configure --prefix='/chemin/bin'`
- Compilation : **make**
- Installation : **sudo make install**
- Préparation de l'environnement :

Présentation d'open MPI

Description de communication dans Open MPI

- l'environnement d'exécution
- les communication point à point
- les communication collectives
- les groupes de processus
- les topologies de processus

Compilation d'un programme parallèle avec verifcarlo

CC=OMPI_CC=verifcarlo mpicc

Bibliography

- <https://www.open-mpi.org>
- <https://fr.wikibooks.org>

Introduction

- **Compilateurs** : *Clang* et *gcc*
- **probleme** : *le support de gcc était éphémère dû à une dépendance avec fortran qui vas être enlevé dans le futur*
- **solutions** : *supporter les types vectoriels de **clang***
- **test** : *configurer **verificarlo** avec **clang** pour le C et C++ avec la commande suivante :*

```
./configure --without-flang CC=clang CXX=clang++
```


Tests

- *Suivre le fonctionnement de test que Verificarlo a commencé à implémenter.*
- *Les tests sont principalement écrits en bash, avec un code de test écrit en c et un code python*


Tests

- *Donc en général nous avons effectué des tests sur les opérations arithmétique vectorielles avec les jeux d'instruction sse, avx et avx512, et s'assurer du bon fonctionnement*
- *Nous avons effectué trois sous tests pour s'assurer du bon fonctionnement.*

sous test 1 / : le bon resultat des opérations vectorielles

- *Dans ce cas nous avons testé sur les différents backends, les différentes operations, avec les vecteurs de taille differente sur les précisions qu'on a choisit (float et double); et on a deduit que les résultats retournés sont vrai*

exemple



`../resources/bon_resultat.png`

sous test 2 / : l'appel des probes vectorielles

- *Nous avons généré le fichier intermédiaire pendant la compilation avec la commande : **-save-temps***
- *une fois le fichier généré , on remarque que on a effectivement fait appel à notre probe vectorielle*

exemple

```
../resources/appe1_des_fonction.png
```

sous test3 / :Utilisation des jeux d'instructions vectorielles

- *Dans verifcarle, les instructions vectorielles pour les opérations arithmetiques sont présentées par la concaténation de :
operation,vectoriel,précision*
- *elle s'utilise sur les registres **xmm,ymm,zmm** associés respectivement au jeux d'instructions **sse,avx,avx512***

exemple

```
../resources/ajout_instructions.pn
```

support des vecteurs 512/256 bits

- les vecteurs 256 et 512 bits sont déjà inclus et supportés

Ajout des probes vectorielles

- *les probes vectorielles sont implémentées avec la version scalaire.*
- *Appel des fonction vectorielles des backends dans :
`src/vfcwrapper/main.c`*

Ajout des fonctions vectorielles dans l'interface

- *ajouter les fonctions vectorielles dans l'interface qui se trouve dans le fichier `src/common/interflop.h`*
- *mettre la taille des vecteur en paramètre dans les fonctions vectorielles*
- *Comme nous passons la taille en argument, il faudra tester la taille pour permettre à clang d'effectuer une opération vectorielle en changeant le type de notre tableau dans le bon type vectorielles de clang.*
- *changer le type vectorielles en son pointeur sur sa précision.*
- *Deplacer la définitions des types vectorielles dans le fichier `src/common/inteflop.h` .*

Changements aux niveaux des backends

Fonctions vectorielles en mode scalaire

Tous les backends

Fonctions vectorielles en mode vectoriel

- ieee
- vprec

Changements aux niveaux du backend vprec

Fonctionnement du backend

- norme IEEE754
- fonction de debug

Opérande constantes

- avertissement de clang sur les types des paramètres de fonction
- ajout d'un pragma pour retirer l'avertissement

Fonctionnement du backend

- nombres fini et infini
- nombres normaux et dénormaux

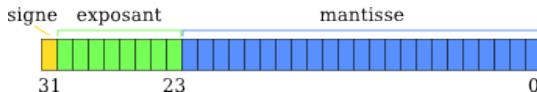


Figure – Représentation d'un nombre flottant simple précision

Ajout dans verifcarlo

Compilation des **wrappers** et des **backends** avec le drapeau
-march=native

Types vectorielles

Vecteur de 4 double précision

Jeu d'instruction disponible

SSE

Clang

Utilise 4 addition vectoriel SSE

Verificarlo

- Backend : vectorisé comme pour clang
- Problème : vecteur passé par registre entre les modules

Cours en relation

Architecture Parallèle