DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr
www.u-bordeaux.fr
www.enseirb-matmeca.fr

Validation partielle des applications parallèles sur supercalculateurs

Présentation INHP@CT

Auteur : Hugo Brunie
Tuteur : Patrick Carribault
Co-encadrante : Emmanuelle Saillard

CEA, DAM, DIF, F-91297 Arpajon, France |

Jeudi 25 juin 2015

Contexte
Calcul Haute Performance (HPC)

- Besoin croissant en puissance de calcul en science ;
- Développement de machines de plus en plus puissantes : supercalculateurs (PetaFLOPS) ;
- Possibilité de faire tourner des codes parallèles.

Problématique

- Codes de calculs de plusieurs centaines de milliers de lignes ;
- Complexes à concevoir, développer, maintenir ;
- Besoin d'outils de débogue :
  - Phase de développement ;
  - Phase de production.

Deux paradigmes du parallélisme

- Mémoire distribuée : envois de messages entre processus (MPI [a]) ;
- Mémoire partagée : thread POSIX ou OpenMP.

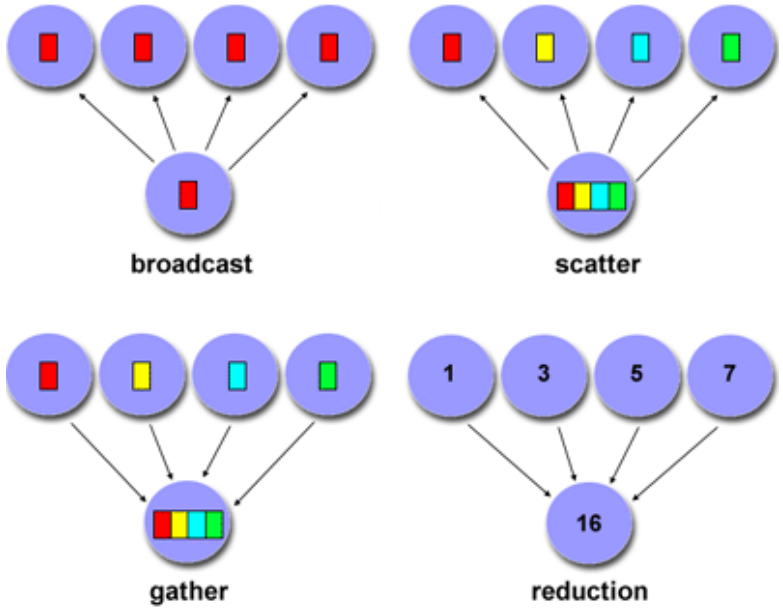
a. Message Passing Interface : standard international de l'envoi de messages

Catégories de bogues

- Manipulation de ressources et débordements mémoire.
- Accès concurrents entre threads ;
- Inadéquations entre les arguments d'un appel MPI ;
- Situations de *deadlock* MPI ;

Cadre du stage

- Calcul Haute Performance ;
- Code scientifiques complexes à déboguer ;
- Parallélisme à mémoire distribuée : MPI ;
- Détection de *deadlock* dus aux collectives MPI.



Définition : collective MPI

Tous les processus d'un même groupe (communicateur) doivent

- Effectuer le même nombre de collectives ;
- Dans le même ordre ;
- Avec des arguments compatibles.

Contexte
Motivation

```
1  void f(MPI_Comm c){
2      MPI_Barrier(c);
3  }
4  int main(int argc, char * argv[]){
5    int rank;
6    MPI_Init(&argc, &argv);
7    MPI_Comm_rank(MPI_COMM_WORLD,
8                  &rank);
9
10   ...
11   if(rank == 0)
12     f(MPI_COMM_WORLD);
13   ...
14   MPI_Finalize()
15   return 0;
16 }
```

Exemple

- Appel à une barrière MPI dans un if ;
- Si nombre de processus $\geq 2 \rightarrow$ *Deadlock* ;
- Processus 0 attend au MPI_Barrier ;
- Autres processus coincés au MPI_Finalize().

But

- Détecter le *deadlock* ;
- Messages d'erreurs indiquant la source du problème ;
- Passage à l'échelle performant.

Existant

- Outil développé au CEA : PARCOACH ;
- Détecte les *deadlocks* dus à des collectives MPI ;
- Analyse chaque fonction indépendemment ;
- Limite : pas d'analyse interprocédurale.

Contribution

- Étendre PARCOACH à l'analyse interprocédurale ;
- État de l'art des analyses utilisées ;
- Conception d'un algorithme adapté ;
- Implémentation et évaluation.

Plan

Plan

PARCOACH

- Plateforme d'aide au débogue de *deadlock* de codes parallèles ;
- Fonctionne sur des codes MPI [a], OpenMP [b] et MPI+OpenMP [c] ;
- Analyse statique du code à la compilation ;
- Instrumentation sélective ;
- Capture à l'exécution des *deadlocks* dus à des collectives MPI.

---

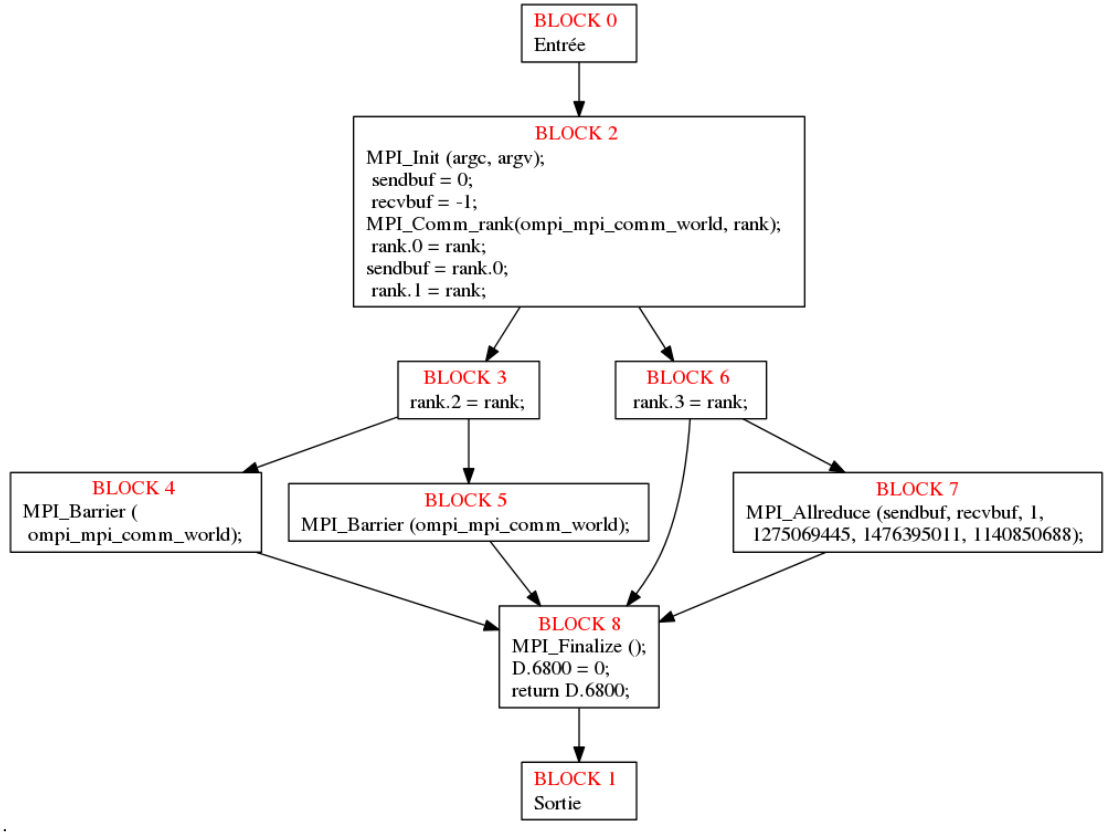
a. Combining static and dynamic validation of MPI collective communications, IJHPCA, 2014

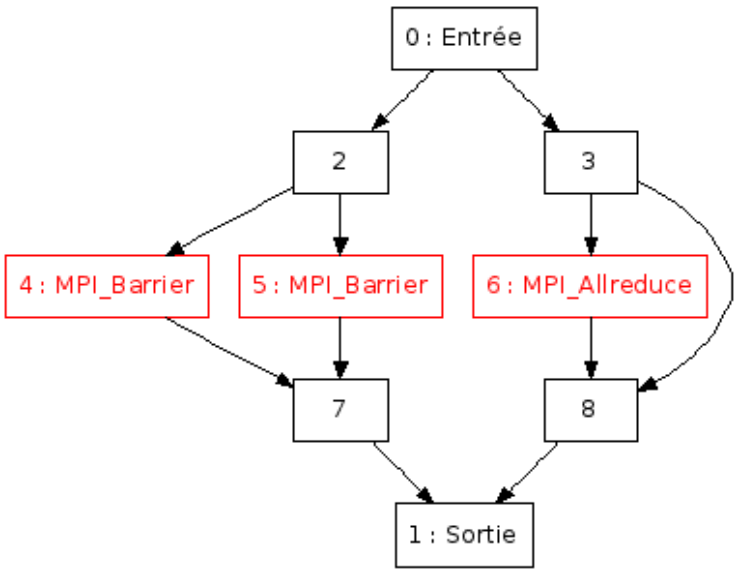
b. Static Validation of Barriers and Worksharing Constructs in OpenMP Applications, IWOMP, 2014

c. Static/Dynamic validation of MPI collective communications in multithreaded context, E. Saillard et al., PPoPP, 2015

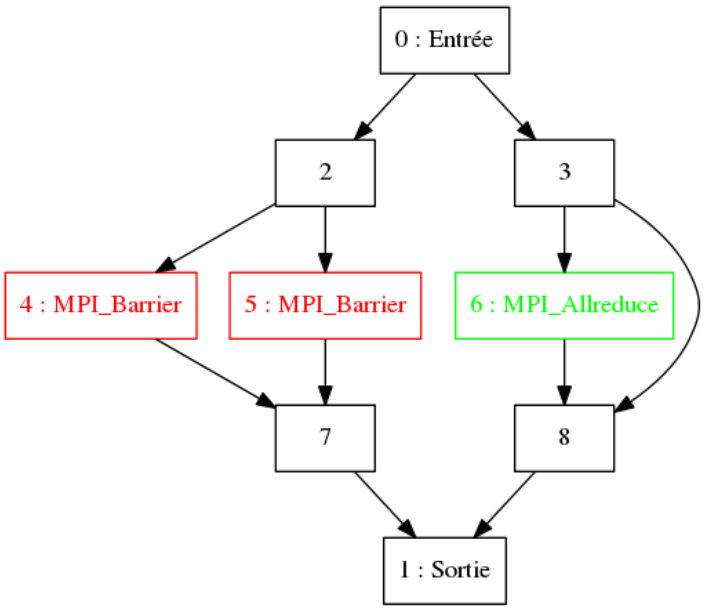
PARCOACH
Graphe de Flot de Contrôle

```
1  #include <mpi.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4
5  int main(int argc, char * argv[])
6  {
7      int rank;
8      MPI_Init(&argc, &argv);
9      int sendbuffer = 0;
10     int recvbuffer = -1;
11     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
12     sendbuffer = rank;
13     if(rank < 10)
14     {
15         if(rank == 0)
16         {
17             MPI_Barrier(MPI_COMM_WORLD);
18         }else
19         {
20             MPI_Barrier(MPI_COMM_WORLD);
21         }
22     }else
23     {
24         if(rank == 0)
25         {
26             MPI_Allreduce([...], MPI_COMM_WORLD);
27         }
28     }
29     MPI_Finalize();
30     return 0;
31 }
```





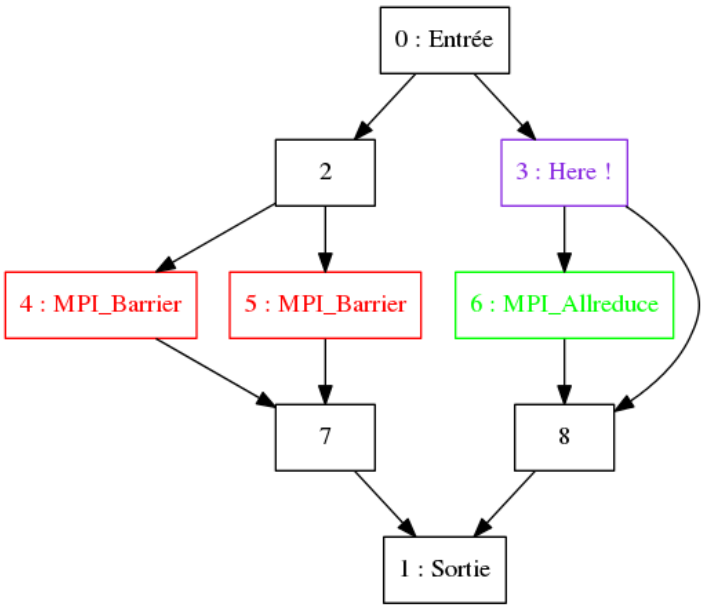
- Modification du CFG pour ne garder que les appels de fonctions ;
- Marquage des noeuds contenant des appels à des collectives ;
- Calcul de l'ordre d'exécution de chaque noeud marqué ;
- Calcul des classes d'équivalence (nom et ordre d'exécution) ;
- Calcul de la PDF (Frontière de Post Dominance) de chaque classe d'équivalence.



- Le CFG.

- $x \gg y$: tous les chemins qui partent de y et vont au nœud de sortie du CFG passent nécessairement par x. Par définition tout nœud se post-domine.

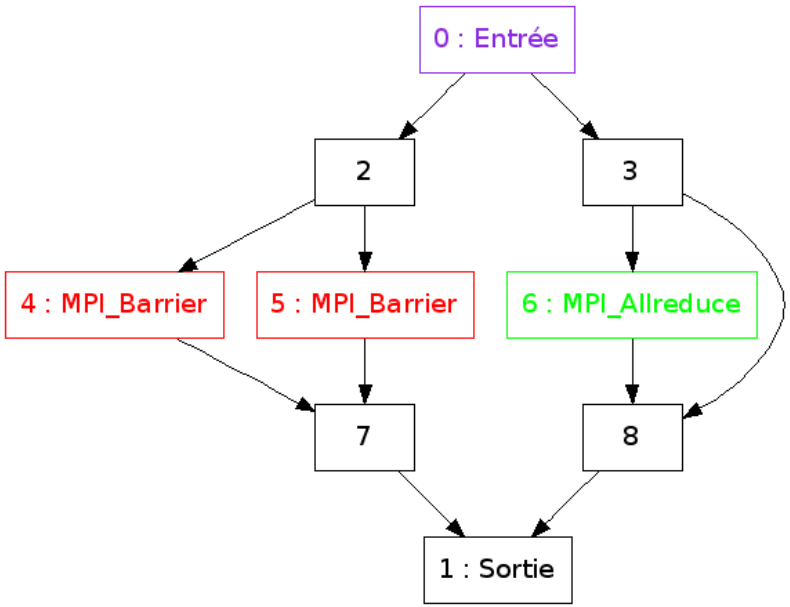
Calcul de la Frontière de Post-Dominance (PDF)



- PDF du nœud 6 $\rightarrow$ nœud 3.

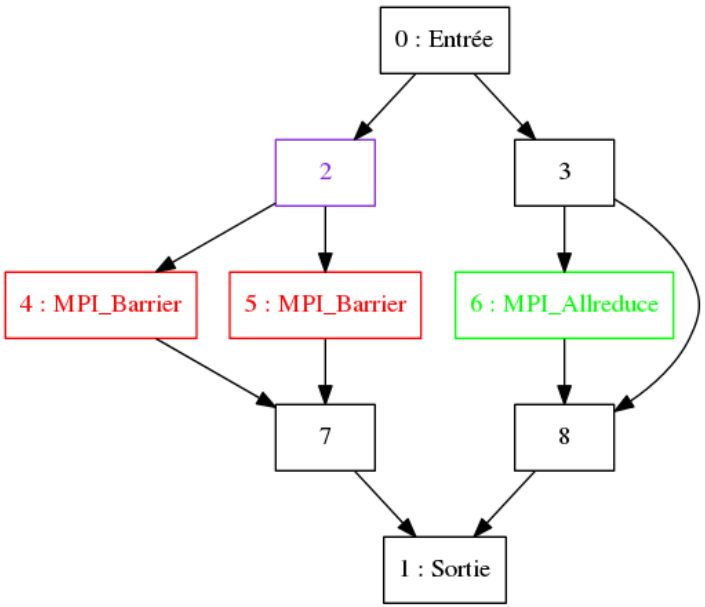
- $x \in PDF(y)$ ssi $\exists$ s $\in SUCC_{CFG}(x)$ | $s \gtrdot y$, and $y \not\gg x$

Calcul de la Frontière de Post-Dominance (PDF)



- PDF du nœud 6 $\rightarrow$ nœud 3 ;
- PDF^+ du nœud 6 $\rightarrow$ nœuds 3 et 0.

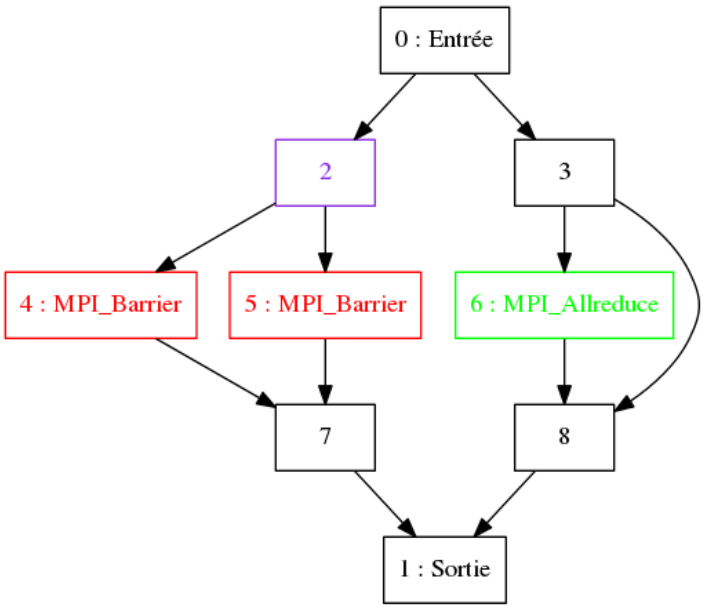
- $x \in PDF(y)$ ssi $\exists$ s $\in SUCC_{CFG}(x)$ | $s \underline{\gg} y$, and $y \not\gg x$
- PDF^+ c'est la fermeture transitive de la PDF.



- PDF du nœud 4 $\rightarrow$ nœud 2.

- $x \in PDF(y)$ ssi $\exists$ s $\in SUCC_{CFG}(x)$ | $s \gtrdot y$, and $y \not\gg x$

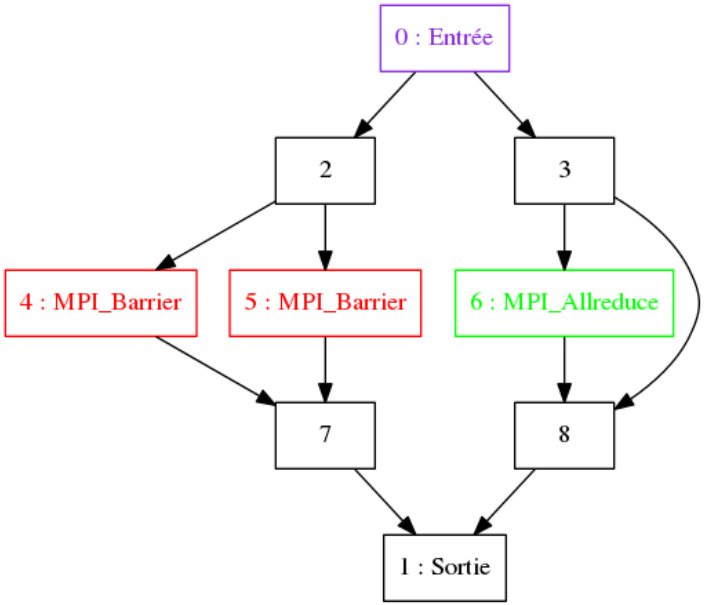
Calcul de la Frontière de Post-Dominance (PDF)



- PDF du nœud 4 $\rightarrow$ nœud 2 ;
- PDF du nœud 5 $\rightarrow$ nœud 2.

- $x \in PDF(y)$ ssi $\exists$ s $\in SUCC_{CFG}(x)$ | $s \gtrdot y$, and $y \not\gg x$

Calcul de la Frontière de Post-Dominance (PDF)



- PDF des nœuds 4 et 5 $\overline{(MPI\_Barrier, 0)}$
  $\rightarrow$ nœud 0.

- $x \in PDF(y)$ ssi $\exists$ s $\in SUCC_{CFG}(x)$ | $s \underline{\gg} y$, and $y \not\gg x$

Faux positif

```
1   MPI_Comm CommA;
2   ...
3   int f()
4   {
5       ...
6       MPI_Barrier(CommA);
7       ...
8   }
9
10  int main(int argc, char * argv[])
11  {
12      ...
13
14      if(...)
15          f();
16      else
17          MPI_Barrier(CommA) ;
18      ...
19      return 0;
20  }
```

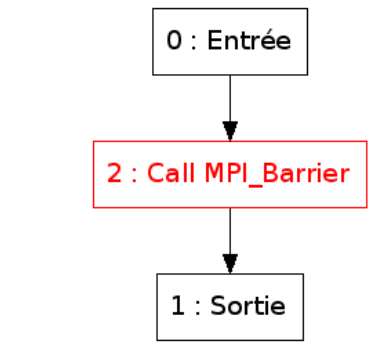


FIGURE – CFG de f

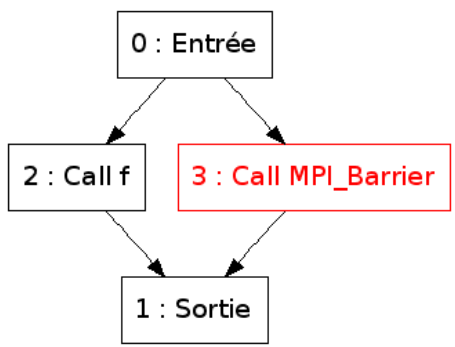


FIGURE – CFG du main

Faux négatif

```
1   MPI_Comm CommA;
2   .
3   int f()
4   {
5       ...
6       MPI_Barrier(CommA);
7       ...
8   }
9
10  int main(int argc, char * argv[])
11  {
12      ...
13
14      if(...)
15          f();
16      else
17          ;
18      ...
19      return 0;
20  }
```

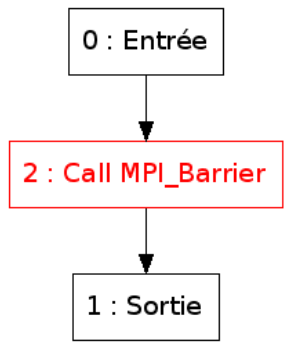


FIGURE – CFG de f

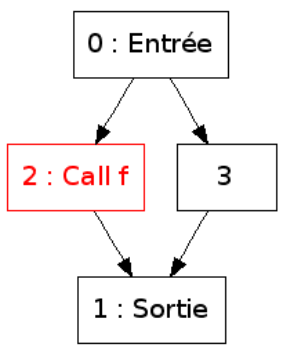


FIGURE – CFG du main

Plan

Contribution
Introduction

Contribution

- Extension de l'analyse intraprocédurale de PARCOACH à une analyse interprocédurale ;
- Etat de l'art et conception d'un algorithme ;
- Implémentation et évaluation.

Définition : analyse interprocédurale

- Besoin de "comprendre" le programme dans son ensemble ;
- Analyse de toutes les fonctions dans leur ensemble.

Exemple d'utilisation dans GCC

- Calcul de la propagation de constante ;
- Optimisation par inlining ;
- Elimination de code mort.

Analyse interprocédurale : première étape [a]

a. *Advanced Compiler Design & Implementation*, Steven S. Muchnick

- Construction du graphe d'appel de fonctions ;
- Programme non récursif : *Direct Acyclic Graph* (DAG) ;
- Difficulté :
  - Programme récursif → Agréger les composantes fortement connexes ;
  - Pointeurs de fonction → Aliasing ;
  - Compilation séparée, bibliothèque dynamique.

Deux parcours

- Parcours dans le sens de l'invocation ;
- Parcours dans le sens inverse de l'invocation ;

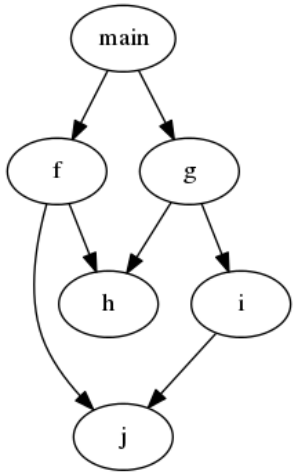


FIGURE – Graphe d'appel de fonction

Inlining et Cloning [a]

a. *Advanced Compiler Design & Implementation* :19 :Steven S. Muchnick

- *Inlining* : Recopier une fonction sur tous les sites d'appels ;
- *Cloning* : Construire une version d'une fonction spécifique à un contexte d'appel particulier ;
- Complexité importante.

Complexité exponentielle

- P_1, P_2 et P_3, 3 procédures ;
- P_1 appelle P_2 2 fois ;
- P_2 appelle P_3 2 fois ;
- *Inlining* : P_3 est recopié 4 fois dans P_1.

Contribution
État de l'art

Analyse du flot de donnée : résumé de fonction

- Construction d'un résumé pour chaque fonction ;
- Utilisation d'une table de résumé ;

Construction des résumés : deux parcours

- Traquer l'information de l'appelante vers l'appelée :
- Problème de la propagation (de constante) ;
- Traquer l'information de l'appelée vers l'appelante :
- Problème de l'effet de bord.

Bottom up [a]

a. Hybrid Top-down and Bottom-up Interprocedural Analysis, Zhang et al.

- Débute avec les fonctions feuilles ;
- Permet de réutiliser l'algorithme intraprocédurale ;
- Pas de dépendance du résumé au contexte : meilleure possibilité de réutilisation ;
- Effet de bord.

Top down

- Débute par les fonctions racines ;
- Dépendance du résumé au contexte d'appel ;
- Intéressant si le résumé dépend des paramètres de la fonction ;
- Propagation de constante.

Principe

- Construire le Graphe d'Appel de Fonctions (GAF) ;
- Parcourir le GAF depuis les feuilles vers la racine ;
- Pour chaque nœud appliquer l'analyse intraprocédurale ;
- Faire remonter la séquence valide d'appels à des collectives MPI.

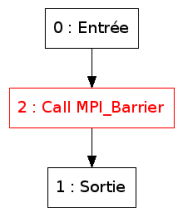


FIGURE – CFG de f

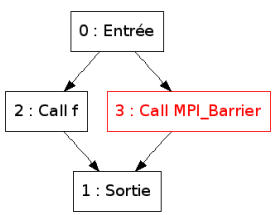


FIGURE – CFG du main

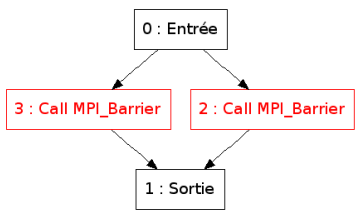


FIGURE – CFG modifé par l'analyse interprocédurale

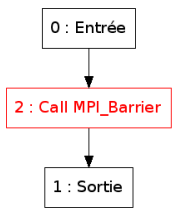


FIGURE – CFG de f

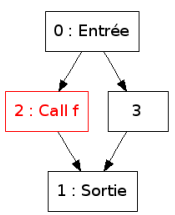


FIGURE – CFG du main

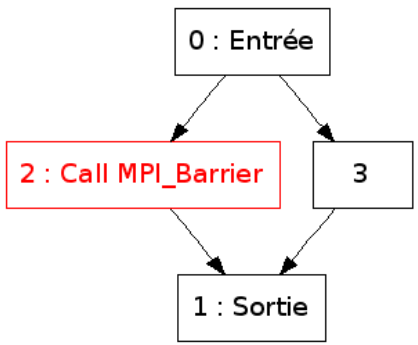


FIGURE – CFG modifié par l'analyse interprocédurale

Principe

- Construire le Graphe d'Appel de Fonctions (GAF) ;
- Parcourir le GAF depuis les feuilles vers la racine ;
- Pour chaque nœud appliquer l'analyse intraprocédurale ;
- Faire remonter un CFG résumé de la fonction appelé ;
- CFG résumé : nœuds des appels aux collectives MPI et des PDF^+.

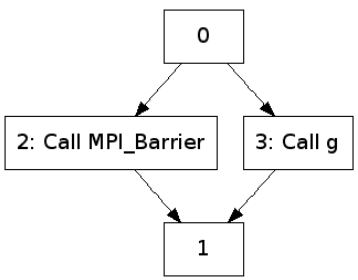


FIGURE – CFG du main

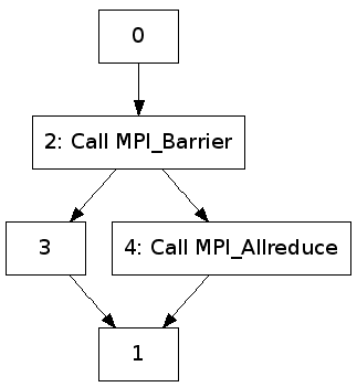


FIGURE – CFG de g

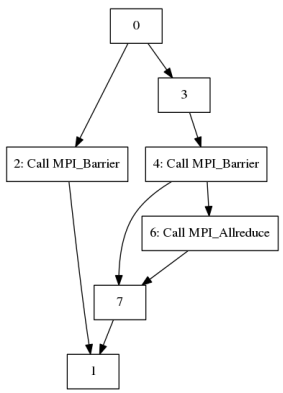


FIGURE – CFG modifié par l'analyse interprocédurale

GCC

- Front end : C,C++, Fortran vers GIMPLE ;
- Middle end : passes de vérification et d'optimisation ;
- Back End : du RTL à l'assembleur.

PARCOACH se situe dans le middle end, c'est un plugin de GCC (4.6.3)

Choix

- Le langage : python ;
- Abandon de la possibilité d'utiliser le Link Time Optimization de GCC ;
- Dump dans un fichier et application de l'algorithme.

NAS FT

- 2000 lignes ;
- 4 appels à des collectives MPI ;
- 25% overheads à la compilation (3secs) ;

```
$> make ft NPROCS=4 CLASS=A
========================================
    =      NAS Parallel Benchmarks 3.3      =
     =      MPI/F77/C                         =
        ========================================

>>>>>>>>>>>>>>>>>> STARTING PARCOACH <<<<<<<<<<<<<<<<<
-------------------------------------------------------

----------------------

>>> MPI VERIFICATION <<<
----------------------

Interproc, filling dump file : ./ft_df.dat
-------------------------------------------------------

>>>>>>>>>>>>>>>>>> CLOSING PARCOACH <<<<<<<<<<<<<<<<<
-------------------------------------------------------
```

FIGURE – Sortie standard : intraprocédurale

```
$>python ./python/main.py sf ./NPB3.3-MPI/FT/ft_df.dat
*************** WARNING PARCOACH: function ft
mpi_reduce lines: 1714-COND STMT line: 161
```

FIGURE – Sortie standard : interprocédurale

HACC IO

- 2000 lignes ;
- 29 appels à des collectives MPI ;
- 30% overheads à la compilation (5secs) ;

```
1  int RestartIO_GLEAN :: Close (void){
2      [...]
3      if (m_mode == WRITE_CHECKPOINT){
4          switch (m_interface){
5              case USE_POSIX:
6                  status = this->__POSIX_Close_Checkpoint();
7                  break;
8              case USE_MPIIO:
9                  status = this->__MPIIO_Close_Checkpoint();
10                 break;
11         }
12     }
13         [...]
```

FIGURE – Extrait de code du test HACC/IO de CORAL : branchement

```
1  int RestartIO_GLEAN :: __POSIX_Close_Checkpoint (void)
2  {
3      [...]
4      MPI_Barrier(m_partitionComm);
5      [...]
6      MPI_Barrier(m_partitionComm);
7      [...]
8  }
9  int RestartIO_GLEAN :: __MPIIO_Close_Checkpoint (void)
10 {
11     [...]
12     MPI_Barrier(m_partitionComm);
13     [...]
14 }
```

FIGURE – Extrait de code du test HACC/IO de CORAL : fonctions appelées

Plan

Conclusion et Perspectives
Validation

Pour conclure

- Besoin de moyen de calcul important (traitement de données CERN) ;
- Codes parallèles complexes (bogues "traditionnels" + bogues dus au parallélisme) ;
- Outils de débogue et d'optimisation (profiling) ;
- PARCOACH propose une approche hybride (statique/dynamique) pour ces outils ;
- Contribution : extension de l'analyse statique ;
- État de l'art sur les analyses interprocédurales ;
- Choix d'une analyse, adaptation et implémentation ;
- Validation à travers des Benchmarks connus.

Perspectives

- Implémentation et validation (ou invalidation) de la deuxième approche ;
- Test de la première approche sur d'autres Benchmarks ;
- Réflexion sur l'apport d'une analyse de Flot de Données ;
- Implémentation et validation d'une telle analyse.

Commissariat à l'énergie atomique et aux énergies alternatives
CEA, DAM, DIF, F-91297 Arpajon, France
T. +33 (0)1 69 26 40 00

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019

Algorithme 1 Analyse intraprocédurale

Données: Graphe de Flot de Contrôle : $G(V, E)$
Résultat: Ensemble d'avertissements : O, séquence valide : S

$O \leftarrow \emptyset$
$S \leftarrow \{\}$
Retirer les arcs revenant (cycle) dans G
Calculer les ordres d'exécution pour chaque noeud contenant une collective : $c(o)$
Pour tout o dans l'ordre croissant d'exécution **faire**
  Pour c dans l'ensemble des noms de collective d'ordre d'exécution o **faire**
    $C_{o,c} \leftarrow \{u \in V | u$ exécute c **et** $o(u) = max_{o \in o(c)}\}$
    Si $PDF^+(C_{o,c}) \neq \emptyset$ **Alors**
      $O \leftarrow O \cup (c, PDF^+(C_{r,c}))$;
    Sinon
      $ajouter(S, c)$;
    Fin Si
  Fin Pour
Fin Pour
retourne (O, S)

Algorithme 2 Analyse interprocédurale : première approche

Données: $\mathcal{G} = \bigcup_{f\ fonction\ du\ programme} CFG(f)$, GAF
Résultat: warning set : O

```
Seq ← {}
O ← ∅
Pour tout n ∈ GAF dans le sens inverse d'appel de fonction faire
    n.SeqValide ← {}
    O' ← ∅
    Si n.Possède_fils() Alors
        Pour tout Fils f de n faire
            n.remplace_appel_CFG(f.SeqValide)
        Fin Pour
    Fin Si
    (O', SeqValide) ← Analyse_IntraProcédural(n)
    n.SeqValide ← SeqValide
    O ← O ∪ O'
Fin Pour
```

Conclusion et Perspectives
Validation

Algorithme 3 Analyse interprocédurale

Données: $\mathcal{G} = \bigcup_{f\ fonction\ du\ programme} CFG(f)$, GAF
Résultat: warning set : O

$Seq \leftarrow \{\}$
$O \leftarrow \emptyset$
Pour tout $n \in GAF$ dans le sens inverse d'appel de fonction **faire**
  n.SeqValide $\leftarrow \{\}$
  n.SeqNonValide $\leftarrow \{\}$
  $O' \leftarrow \emptyset$
  Si n.Possède_fils() **Alors**
    Pour tout Fils f de n **faire**
      n.remplace_appel_CFG(f.SeqValide, f.SeqNonValide)
    Fin Pour
  Fin Si
  (O', SeqValide, $\mathcal{PDF}^+$) $\leftarrow$ Analyse_IntraProcédural(n)
  n.vs $\leftarrow$ SeqValide
  n.pdfp $\leftarrow \mathcal{PDF}^+$
  $O \leftarrow O \cup O'$
  n.Seq $\leftarrow Seq$
Fin Pour