

## Les passes de GCC

David Odin

Forma3Dev pour CPE-Lyon

2011

- Suppression des commentaires
- Inclusion des **#include**
- Résolution des **#define**

```
#include <stdio.h>

/* La valeur de Pi ! */
#define PI 3.14

float aire(float rayon)
{
    return rayon * rayon * PI;
}
```

```
int printf(const char *, ...);
[ ... ]

float aire(float rayon)
{
    return rayon * rayon * 3.14;
}
```

## PARSING

Lecture et transformation du programme en plus simple.

```
int foo(int a)
{
    int i, c = a;

    for (i = 0; i < 10; i++) {
        if (a > 4)
            c += a - 3 * a * a + 2;
        else
            c += a;
    }
    return c;
}
```

```
int foo(int a)
{
    int i;
    int c = a;

    i = 0;
    goto test_for;
debut_for:
    if (a > 4)
    {
        c = (((a * a) * -3 + a) + 2) + c;
    }
    else
    {
        c = c + a;
    }
    i++;
test_for:
    if (i <= 9) goto debut_for; else goto fin;
fin:
    return c;
}
```

## GIMPLIFICATION

Transformation du programme en code à 3 adresses.

```
int foo(int a)
{
    int i, c = a;

    for (i = 0; i < 10; i++) {
        if (a > 4)
            c += a - 3 * a * a + 2;
        else
            c += a;
    }
    return c;
}
```

```
foo (int a)
{
    int temp1, temp2, temp3, temp4;
    int ret_val, i, c;

    c = a;
    i = 0;
    goto test_for;
debut_for:
    if (a > 4) goto if_true; else goto if_false;
if_true:
    temp1 = a * a;
    temp2 = temp1 * -3;
    temp3 = temp2 + a;
    temp4 = temp3 + 2;
    c = temp4 + c;
    goto if_fin;
if_false:
    c = c + a;
if_fin:
    i = i + 1;
test_for:
    if (i <= 9) goto debut_for; else goto fin;
fin:
    ret_val = c;
    return ret_val;
}
```

## SUPPRESSION DES EXPRESSIONS INUTILES

- Suppression des calculs inutiles et constants.
- Suppression des tests constants.

```
#define TEST 0
[ ... ]
for (i = 0; i < 5; i++) {
  if (TEST) {
    appel_fonction_1 ();
  }
  if (TEST + 1) {
    appel_fonction_2 ();
  }
  i + 5 * i * i;
  10 + 5 * 3 * 8;
}

for (i = 0; i < 5; i++) {
  appel_fonction_2 ();
  i + 5 * i * i;
}
```

## AVERTISSEMENT À PROPOS DES VARIABLES NON-INITIALISÉES

```
int i, a, b;

i = 0;
b = 2;

i = a + 5; // Avertissement !
```

## SUPPRESSION DU CODE MORT

- Suppression des calculs inutiles.

```
int i = 3;

i += 24;
i + 5 * i * i;

int i = 3;

i += 24;
```

## PROPAGATION DES VARIABLES À USAGE UNIQUE

```
{
  int i = 5;
  int a;

  a = 3 * i + i * i - 7;
  appel_fonction(a);
}

{
  int a;

  a = 33;
  appel_fonction(a);
}
```

## SUPPRESSION DU STOCKAGE MORT

- Suppression du premier stockage dans une variable réaffectée.
- Même chose pour un élément d'un tableau.

```
int i;  
int tab[10];  
  
i = 4;  
i = 5;  
tab[5] = 4;  
tab[i] = 28;  
tab[5] = 3;
```

```
int i;  
int tab[10];  
  
i = 5;  
tab[i] = 28;  
tab[5] = 3;
```

## ÉLIMINATION DES RÉCURSIVITÉS TERMINALES

```
int fact(int a)  
{  
    if (a == 0 || a == 1)  
        return 1;  
    return a * fact(a - 1);  
}
```

```
int fact(int a)  
{  
    int mult_acc = a;  
debut:  
    if ((unsigned int)a <= 1)  
        return mult_acc * 1;  
  
    a = a - 1;  
    mult_acc = mult_acc * a;  
    goto debut;  
}
```

## DÉPLACEMENT DES INVARIANTS DE BOUCLE

- Repérage de ce qui ne change pas dans une boucle.
- Déplacement avant la boucle

```
int a, b, i, tab[5];  
[...]  
for (i = 0; i < 5; i++) {  
    tab[i] = a + b;  
}
```

```
int a, b, i, tab[5];  
[...]  
int c = a + b;  
for (i = 0; i < 5; i++) {  
    tab[i] = c;  
}
```

## CRÉATION DE VARIABLES DE BOUCLE CANONIQUES

- Transformation des boucles pour toujours avoir un incrément de 1.

```
for (i = 5; i < 15; i += 2) {  
    [ ... ]  
}
```

```
for (i_boucle = 0; i_boucle < 5; i_boucle++) {  
    i = 5 + i_boucle * 2;  
    [ ... ]  
}
```

## INVERSION BOUCLE/TEST

```
for (i = 0; i < 5; i++) {  
  if (a > 4)  
    tab[i] = a;  
  else  
    tab[i] = 3;  
}
```

```
if (a > 5) {  
  for (i = 0; i < 5; i++) {  
    tab[i] = a;  
  }  
} else {  
  for (i = 0; i < 5; i++) {  
    tab[i] = 3;  
  }  
}
```

## REPLACEMENT/EXÉCUTION DES FONCTIONS INTERNES

```
int a = strlen("Coucou.");  
char text2[10];  
strcpy(text2, "Salut");
```

```
int a = 7;  
char text2[10] = "Salut";
```

## VECTORISATION

- Transforme les boucles simples, en regroupant les opérations 4 par 4.

```
int c = 0;  
for (i = 0; i < 50; i++) {  
  c += i * 3;  
}
```

```
vec4 vec_c = { 0, 0, 0, 0 };  
vec4 vec_i = { 0, 1, 2, 3 };  
vec4 vec_i_inc = { 4, 4, 4, 4 };  
vec3 vec_a = { 3, 3, 3 };  
for (i = 0; i < 12; i++) {  
  vec_c += vec_i * vec_a;  
  vec_i += vec_i_inc;  
}  
c = sum(vec_c);  
for (i = 48; i < 50; i++) {  
  c += i * 3;  
}
```

## ÉLIMINATION DES APPELS TERMINAUX

```
void foo(void)  
{  
  [...]  
}  
  
void bar(void)  
{  
  [...]  
  foo();  
}
```

```
void foo(void)  
{  
  debut_foo:  
  [...]  
}  
  
void bar(void)  
{  
  [...]  
  goto debut_foo;  
}
```

## OPTIMISATION DES BOUCLES IMBRIQUÉES

```
for (i = 0; i < 10; i++) {  
  for (j = 0; j < 100000; j++) {  
    tab[i][j] = tab[i][j] * 3;  
  }  
}
```

```
for (j = 0; j < 100000; j++) {  
  for (i = 0; i < 10; i++) {  
    tab[i][j] = tab[i][j] * 3;  
  }  
}
```

## SUPPRESSION DES BOUCLES VIDES

```
i = 400000;  
for (j = 0; j < 100000; j++) {  
}
```

```
i = 400000;
```

## DÉROULEMENT DES PETITES BOUCLES

```
vec4 vec_c = { 0, 0, 0, 0 };  
vec4 vec_i = { 0, 1, 2, 3 };  
vec4 vec_i_inc = { 4, 4, 4, 4 };  
vec3 vec_a = { 3, 3, 3, 3 };  
for (i = 0; i < 12; i++) {  
  vec_c += vec_i * vec_a;  
  vec_i += vec_i_inc;  
}  
c = sum(vec_c);  
for (i = 48; i < 50; i++) {  
  c += i + 3;  
}
```

```
vec4 vec_c = { 0, 0, 0, 0 };  
vec4 vec_i = { 0, 1, 2, 3 };  
vec4 vec_i_inc = { 4, 4, 4, 4 };  
vec3 vec_a = { 3, 3, 3, 3 };  
vec_c += vec_i * vec_a;  
vec_i += vec_i_inc;  
vec_c += vec_i * vec_a;  
vec_i += vec_i_inc;  
vec_c += vec_i * vec_a;  
vec_i += vec_i_inc;  
vec_c += vec_i * vec_a;  
vec_i += vec_i_inc;  
vec_c += vec_i * vec_a;  
vec_i += vec_i_inc;  
c = sum(vec_c);  
i = 48;  
c += i + 3;  
i = 49;  
c += i + 3;  
}
```

## GÉNÉRATION DU CODE RTL

- Dépendant de l'architecture
- Presque de l'assembleur

## NETTOYAGE DU FLUX DES INSTRUCTIONS

- Suppression des sauts vers l'instruction suivante,
- Suppression des sauts vers des sauts,
- Suppression du code inatteignable.

```
{ [ ... ]  
i = 4 * j;  
goto suivant;  
suivant:  
goto ailleurs;  
j = 3 * k;  
ailleurs:  
return j;  
a = 0;  
}
```

```
{ [ ... ]  
i = 4 * j;  
return j;  
}
```

## SUPPRESSION DES SOUS-EXPRESSIONS COMMUNES

```
c = a - b;  
d = c * a;  
e = a * b;
```

```
c = a - b;  
d = c * a;  
e = c;
```

## OPTIMISATION DES BOUCLES RTL

- Même chose qu'en SSA

## CONVERSION DES TESTS

- Utilisable sur certaines architectures

```
if (a > 9)  
c = 'A' - 10 + a;  
else  
c = '0' + a;
```

```
c = '0' + a + (a > 9) * 7
```

## ALLOCATION FINALE DES REGISTRES

- Remplacement des variables par des registres ou des emplacements sur la pile.
- Optimisation des déplacements de registres.
- Problème NP-complet.
- Très dépendant de plein de choses.

## PASSE FINALE

- Écriture du code assembleur.

## ÉCRITURE DES INFORMATIONS POUR LE DÉBUGGAGE

- Correspondance code généré et fichier/numéro de lignes
- Description des types