Invariance degree and composition of commands for loop peeling

Optimization implemented on a toy parser

Thomas Rubiano

LIPN
Université Paris 13

From an idea of L. Kristiansen
supervised by J. Y. Moyen
in collaboration with T. Seiller

From ICC community ?

- Complexity analysis into two parts : termination and data size
- Analysis of termination using “size-change graphs”
- data size analysis around loops : “*mwp*-bounds” acceptable growth rates ?

Interesting techniques to trace and gather dependencies between variables… What if we try to do so for optimizations ?

Motivations

- Learn about variables dependencies around loops
- Learn about loop optimizations, especially loop-invariant detection and hoisting
- Provide another point of view and maybe a new optimization : “*Quasi-invariant block code motion*”
- In a way to Assist Programmers
- Automate “*obvious*” optimizations
- Seems to not be implemented in compilers. . . (not in LLVM, maybe in GCC. . .)

Basic Loop transformations

- Loop unrolling
- Loop unswitching
- Loop interchange
- Loop fusion
- Loop fission
- Loop skewing

Basics with invariance detection in loop

`a = b OP c` is a loop-invariant computation if each variable is :

- constant, or
- has all definitions outside the loop, or
- has exactly one definition, and that is a loop-invariant
- Search until there is no more invariant...

```
int x=rand()%100;
while(i<100){
    y=x+x; //1
    use(y);
    i=i+1;
}
```

Quasi-Invariants

- A quasi-invariant is a variable which does not change after a certain number of loop execution.
- A degree of invariance is the number of time we need to compute the loop until the variable is stable
- It could be very long for a human. . .

```
while(i<100){
    z=y*y; //2
    use(z);
    y=x+x; //1
    use(y);
    i=i+1;
}
```

Definition

Definition

Let $\texttt{C}$ *be a command or a chunk .*
A Data Flow Graph or DFG is a bipartite graph which bounds variables regarding to $\texttt{C}$ *with labeled-arc set* $\mathcal{A}$*.*

- $\mathrm{In}(\texttt{C}) = \{x | \exists y \; x \xrightarrow{\texttt{C}} y\}$
- $\mathrm{Out}(\texttt{C}) = \{y | \exists x \; x \xrightarrow{\texttt{C}} y\}$
- $\mathcal{A} \subseteq \mathrm{In}(\texttt{C}) \times \{\emptyset, 0, 1\} \times \mathrm{Out}(\texttt{C})$

Types of relations

$$\texttt{C} := [x_0 = x_0 + 1;$$

$$x_0 \xrightarrow[1]{\text{dependence}} y_0$$

$$x_1 = x_1;$$

$$x_1 \overset{\text{propagation}}{\underset{0}{\dashrightarrow}} y_1$$

$$x_2 = 0;]$$

$$x_2 \quad \overset{\text{reinitialization}}{\underset{\emptyset}{}} \quad y_2$$

FIGURE – Types of dependence

Matrix

Definition

This Data Flow Graph can be represented as a matrix $N \times N$ with $N = |var(\mathrm{C})|$, we will note C the corresponding matrix to C.

$$C = \begin{bmatrix} 1 & \emptyset & \emptyset \\ \emptyset & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset \end{bmatrix}$$

$$x_0 \xrightarrow[1]{\text{dependence}} y_0$$

$$x_1 \overset{\text{propagation}}{\underset{0}{\dashrightarrow}} y_1$$

$$x_2 \quad \overset{\text{reinitialization}}{\underset{\emptyset}{\quad}} \quad y_2$$

FIGURE – Matrix of dependence

Chunks

- Command Composition
- See one block as one command
- Hoist an entire block (could be a loop !)

Multipath and Composition

Definition

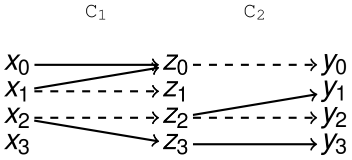
A sequence of commands noted $[\mathtt{C}_1; \mathtt{C}_2; \ldots]$ *can be viewed as a concatenated Data Flow Graph or Multipath.*

Multipath and Composition example

Example of the following sequence :

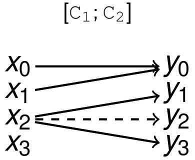
$\mathtt{C}_1 := [x_0 = x_0 + x_1; x_3 = x_2 + 2];$

$\mathtt{C}_2 := [x_1 = x_2; x_3 = x_3 * 2];$



$$C_1 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & 0 & \emptyset & \emptyset \\ \emptyset & \emptyset & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix} C_2 = \begin{bmatrix} 0 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \end{bmatrix}$$

(a) Multipaths



$$M_{\mathtt{C}_1;\mathtt{C}_2} = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

(b) Composition

Matrix Algebra

The Matrix representing a *DFG* is composed of elements in $\mathcal{E} = \{\emptyset, 0, 1\}$. The elements in $\mathcal{E}$ are ordered as follows : $\emptyset < 0 < 1$. And we can introduce two operations noted $\oplus$ and $\otimes$ defined as below :

$\oplus_{max}$	$\emptyset$	0	1
$\emptyset$	$\emptyset$	0	1
0	0	0	1
1	1	1	1

$\otimes_{+}$	$\emptyset$	0	1
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
0	$\emptyset$	0	1
1	$\emptyset$	1	1

$\oplus$ could be seen as a *max* and $\otimes$ as a $+$ if we consider $\emptyset$ as $-\infty$.
Then the composition of matrices is computed as :
$C_{i,j} = \bigoplus_k (A_{i,k} \otimes B_{k,j})$ we can write $C = A \bullet B$.

Condition

$\mathtt{C} := \mathtt{if}\ \mathtt{E}\ \mathtt{then}\ \mathtt{C}_1;$

- all variables defining $\mathtt{E}$ will create a dependence with all modified variables in $\mathtt{C}_1$.
- E is the vector of all variables present with a 1 if $E_i \in Var(\mathtt{E})$

$C_{i,j} = \bigoplus_k (E_i \oplus C_{1_{k,j}})$ also noted $(C)^{\mathtt{E}}$

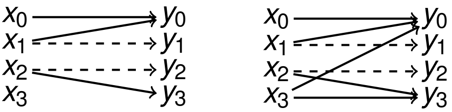
Condition example

Example of the following sequence : $\texttt{C} := \texttt{if E then C}_1$; with

$\texttt{E} := x_3 \geq 0$

$\texttt{C}_1 := [x_0 = x_0 + x_1; x_3 = x_2 + 2];$

$\texttt{E}$ $\qquad$ $\texttt{C}_1$ $\qquad$ $[\texttt{if E then C}_1]$



$$E = \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ 1 \end{bmatrix} \qquad C_1 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & 0 & \emptyset & \emptyset \\ \emptyset & \emptyset & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix} \qquad (C_1)^{\mathtt{E}} = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & 0 & \emptyset & \emptyset \\ \emptyset & \emptyset & 0 & 1 \\ 1 & \emptyset & \emptyset & 1 \end{bmatrix}$$

FIGURE – Condition

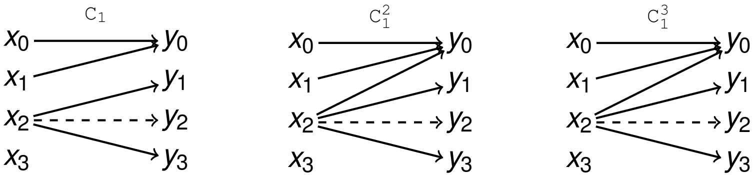
Loop `while`

Let `C` be a command such as : `C` := `while E do C`$_1$;.

- first occurrence of `C`$_1$ will influence the second one and so on.
- we consider the number of iteration undecidable then we treat all the cases with a max (or sum) we write $\mathcal{C}^k = \sum_{n=1}^{k} C^n$
- the number of relations is finite (the number of variables is finite) and the sum rises strictly, then it exists a fix point after a certain k such as $\mathcal{C}^{k+1} = \mathcal{C}^k$. Let's define $\mathcal{C}^* = Fix(\mathcal{C}^k)$.
- Furthermore, `E` influences each occurrence of `C`$_1$ (as previously). We represent E as a vector as previously, then the composition should be expressed as : $C_{i,j} = \bigoplus_k (E_i \oplus (\mathcal{C}_1^*)_{k,j})$ or we can simplify the notation as $C = (\mathcal{C}_1^*)^{\mathrm{E}}$

Loop `while` example

Example of the following sequence ($\mathtt{C}_1$ is the composition presented previously) : $\mathtt{C} := \mathtt{while}\ \mathtt{E}\ \mathtt{do}\ \mathtt{C}_1$; with $\mathtt{E} := x_3 \geq 0$
$\mathtt{C}_1 := [x_0 = x_0 + x_1; x_3 = x_2 + 2; x_1 = x_2; x_3 = x_3 * 2];$

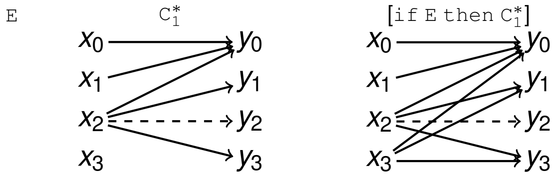


$$C_1 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix} \quad C_1^2 = \mathcal{C}_1^2 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ 1 & 1 & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix} \quad C_1^3 = \mathcal{C}_1^* = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ 1 & 1 & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

FIGURE – Finding fix point of dependence (simple example)

Loop `while` example

Then we just need to add the condition correction.



$$E = \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \\ 1 \end{bmatrix} \qquad \mathcal{C}_1^* = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ 1 & 1 & 0 & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix} \qquad (\mathcal{C}_1^*)^{\mathtt{E}} = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ 1 & 1 & 0 & 1 \\ 1 & 1 & \emptyset & 1 \end{bmatrix}$$

FIGURE – Condition of the loop (simple example)

Independences of chunks

Definition

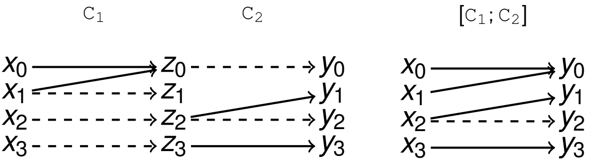
Independence of commands (or chunks of commands). If $\mathrm{Out}(\mathtt{C}_1) \cap \mathrm{In}(\mathtt{C}_2) = \emptyset$ *then* $\mathtt{C}_1$ *is independent to* $\mathtt{C}_2$*.*

Independences of chunks example

Example of the following sequence :

$\mathtt{C}_1 := [x_0 = x_0 + x_1;$

$\mathtt{C}_2 := [x_1 = x_2; x_3 = x_3 * 2];$

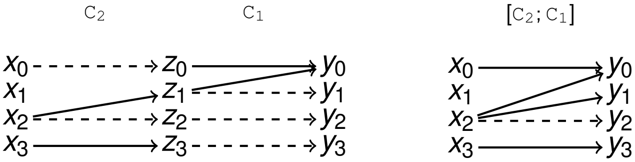


$$C_1 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & 0 & \emptyset & \emptyset \\ \emptyset & \emptyset & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 0 \end{bmatrix} \quad C_2 = \begin{bmatrix} 0 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \end{bmatrix} \quad C_1 \bullet C_2 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \end{bmatrix}$$

FIGURE – Composition of independent chunks of commands

Independences of chunks example

In this example, `C1` is independent of `C2` but the inverse is not true.



$$C_2 = \begin{bmatrix} 0 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \end{bmatrix} \quad C_1 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ 1 & 0 & \emptyset & \emptyset \\ \emptyset & \emptyset & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 0 \end{bmatrix} \quad C_2 \bullet C_1 = \begin{bmatrix} 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ 1 & 1 & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \end{bmatrix}$$

FIGURE – Composition of dependent chunks of commands

Here $C_1 \bullet C_2 \neq C_2 \bullet C_1$.

Self-Independence

Definition

If $\mathtt{C}_1$ *is independent to itself, we say* $\mathtt{C}_1$ *is self-independent*

Lemma

Specialization for `while` *: If* $\mathtt{C}_1$ *is self-independent then*

$$\mathtt{while}\ \mathtt{E}\ \mathtt{do}\ \mathtt{C}_1 \equiv \mathtt{if}\ \mathtt{E}\ \mathtt{then}\ \mathtt{C}_1$$

Mutual independence of chunks

Definition

If $\mathtt{C}_2$ *independent of* $\mathtt{C}_1$ *and* $\mathtt{C}_1$ *independent of* $\mathtt{C}_2$ *then* $\mathtt{C}_2$ *and* $\mathtt{C}_1$ *are mutually independents :*

$$\mathtt{C}_1 \asymp \mathtt{C}_2$$

Example of the following sequence :
$\mathtt{C}_1 := [x_0 = x_0 + x_1;$
$\mathtt{C}_2 := [x_3 = x_2 + x_3 * 2];$

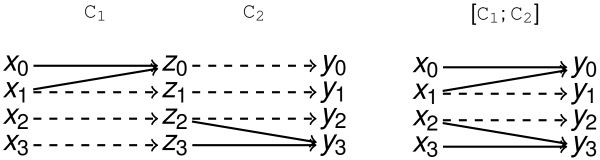


FIGURE – Composition of mutually independent chunks of commands

In this example, $\mathtt{C}_1 \asymp \mathtt{C}_2$ but $[\mathtt{C}_1; \mathtt{C}_2]$ is not self-independent.

Moving Independent Chunks

Lemma

Swapping commands (or chunks of commands) : If $\mathtt{C}_1 \asymp \mathtt{C}_2$ *then*

$$\mathtt{C}_1 ; \mathtt{C}_2 \equiv \mathtt{C}_2 ; \mathtt{C}_1$$

Lemma

Moving mutual independent commands out of `while` *: If* $\mathtt{C}_1 \asymp \mathtt{C}_2$ *and* $\mathtt{C}_1 \asymp \mathtt{C}_1$ *then*

$$\mathtt{while\ E\ do}\ [\mathtt{C}_1 ; \mathtt{C}_2] \equiv [\mathtt{if\ E\ then}\ \mathtt{C}_1 ; \mathtt{while\ E\ do}\ \mathtt{C}_2]$$

Peeling loop ! A new concept ?

- Peeling = removing instructions out of the loop while unrolling
- Suppose we need more "pre-headers"
- Is it always semantically correct ?
- What are the conditions ?

Peeling example

```
while(i<100){
    z=y*y; //2
    use(z);
    y=x+x; //1
    use(y);
    i=i+1;
}
```

```
if (i < 100) //1
{
  z = y * y;
  use(z);
  y = x + x;
  use(y);
  i=i+1;
}
if (i < 100) //2
{
  z = y * y;
  use(z);
  use(y);
  i=i+1;
}
while (i < 100)
{
  use(z);
  use(y);
  i=i+1;
}
```

Figure out the invariance degree

- Statically easy !
- Using dependence graph
- and dominance graph

Let suppose we have computed the list of dependencies for all commands. How to compute the degree of one command ?

1. Initialize every degrees to 0
2. Initialize the current command degree `cd` to ∞
3. **IF** there is no dependencies for the current chunk return 1
4. **ELSE** for each dependencies compute the degree `dd` of the command
   1. **IF** `cd <= dd` and the current command dominates this dependence **THEN** `cd = dd + 1`
      ELSE `cd = dd`

Hoisting and Renaming

- Hoist : create a `if` statement for each degree before the loop and insert every commands of the loop which has a higher or equal degree than the current
- We have to rename variables which are modified by the removed command and appear before it.

Hoisting and Renaming example

```
while(x<100){
    b=b+1; //2
    use(b);
    x=x+1;
    y=0; //1
    while(y<100){ //1
        b=a+y;
        c=b+a;
        y=y*y;
        y=y+1;
    }
    use(b);
}
```

```
if (x < 100) //1
{
  B = b + 1;
  use(B);
  x = x + 1;
  y = 0;
  while (y < 100)
  {
    b = a + y;
    c = b + a;
    y = y * y;
    y = y + 1;
  }
  use(b);
}
if (x < 100) //2
{
  B = b + 1;
  use(B);
  x = x + 1;
  use(b);
}
while (x < 100)
{
  use(B);
  x = x + 1;
  use(b);
}
```

A toy for testing

- to validate, we implemented on a toy parser in python
- around 400 lines
- tested on several examples
- if you have some in mind ?

Last example

```
srand(time(NULL));
int n=rand()%100;
int j=0;
while(j<100){
    fact=1;
    i=1;
    while (i<n) {
        fact=fact*i;
        i=i+1;
    }
    j=j+1;
    use(fact);
}
```

```
srand(time(NULL));
int n = rand() % 100;
int j = 0;
if (j < 100)
{
  fact = 1;
  i = 1;
  while (i < n)
  {
    fact = fact * i;
    i = i + 1;
  }
  j = j + 1;
  use(fact);
}
while (j < 100)
{
  j = j + 1;
  use(fact);
}
```

Revelations !

- I discovered a paper few days ago. . .
  "A Loop Optimization Technique Based on Quasi-Invariance" by Litong Song, Yoshihiko Futamura, Robert Glück, Zhenjiang Hu - 2000
- Why did I miss it ?
- But maybe we still have a new concept ? Chunks or Compositions

Questions !

- Which level ? Is it better to do it at the IR level ?
- Do you think it's relevant to do it in real compilers ?