

# A Fully-Non-transparent Solution to the Code Location Problem

Hugo Venturini

March 8, 2008

Introduction

Scientific and Industrial Contexts

Principles

Results

Conclusions

# Introduction

## The subject

Development of applications aimed at being embedded onto devices such as cell-phones, PDA, digital camera, etc ...



# Introduction

## The subject

Development of applications aimed at being embedded onto devices such as cell-phones, PDA, digital camera, etc ...



## Dev. Chain Elements

- Analysis
- Design
- Implementation
- Test
- **Debug**
- Release

# Introduction

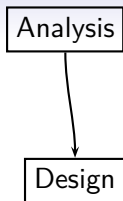
## Motivations

**Analysis**

# Introduction

## Motivations

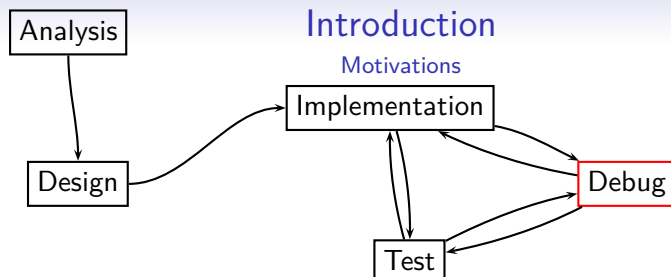
- Analysis: 15%



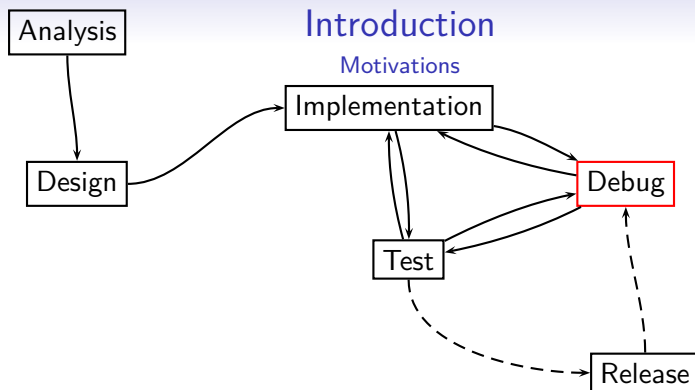
# Introduction

## Motivations

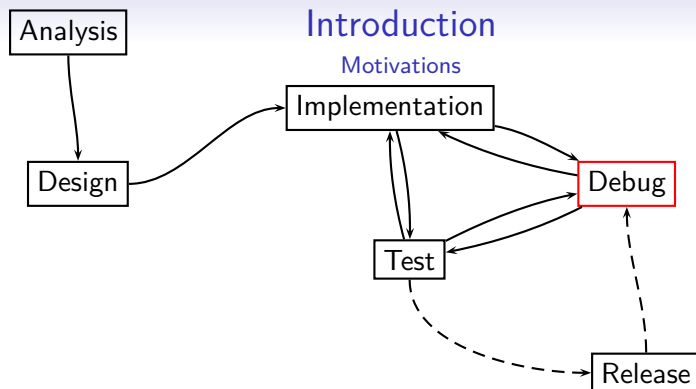
- Analysis: 15%
- Design: 25%



- Analysis: 15%
- Design: 25%
- Implementation and Debug: 15%
- Integration and Test: 45%



- Analysis: 15%
- Design: 25%
- Implementation and Debug: 15%
- Integration and Test: 45%
- Once in a while ... Release



- Analysis: 15%
- Design: 25%
- Implementation and Debug: 15%
- Integration and Test: 45%
- Once in a while ... Release
- ... and what about the support ?? 75%

# Introduction

## Motivations

- Bug Reports  
Bugs found in the released program,
- Real-Life Experimentation  
The real program need be executed,
- Test Onto The Device  
Due to constraints, the unoptimized program may not fit, we need embed the final program,

# Introduction

## Motivations

- Bug Reports  
Bugs found in the released program,
- Real-Life Experimentation  
The real program need be executed,
- Test Onto The Device  
Due to constraints, the unoptimized program may not fit, we need embed the final program,
- ...  
... the optimized program.

# Scientific Context

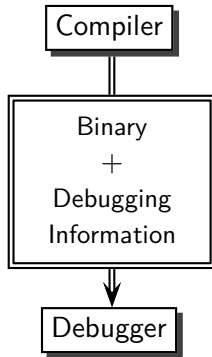
## Problems and other issues

### Data Value Problems

- Residency
- Location
- Value

### Code Location Problem

Irrelevance of the mapping between source code and optimized program.



# Two Main Problems

## Data Value Problem

3 subproblems:

- **Residency Problem**

occurs when a variable's value is not accessible.

*dead code elimination*

- **Data Location Problem**

occurs when a variable is not located in the expected register or at the expected address.

*scalarization, register allocation*

- **Currency Problem**

arises when a variable's value might not be the same at the same point in the source program.

*code hoisting, loop reversal*

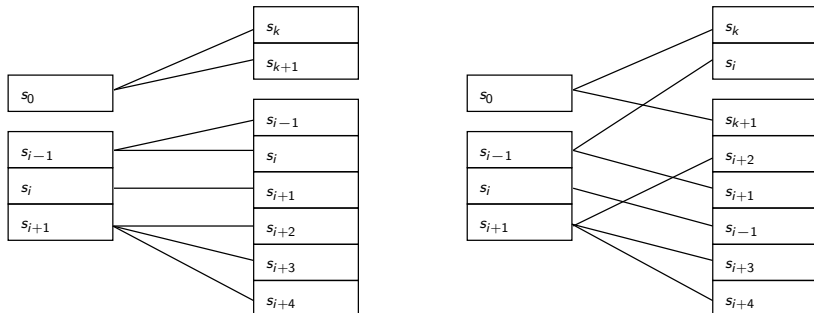
# Two Main Problems

## Code Location Problem

- Code Location Problem**

arises in mapping between locations in the source code and locations in the optimized program.

*code sinking*



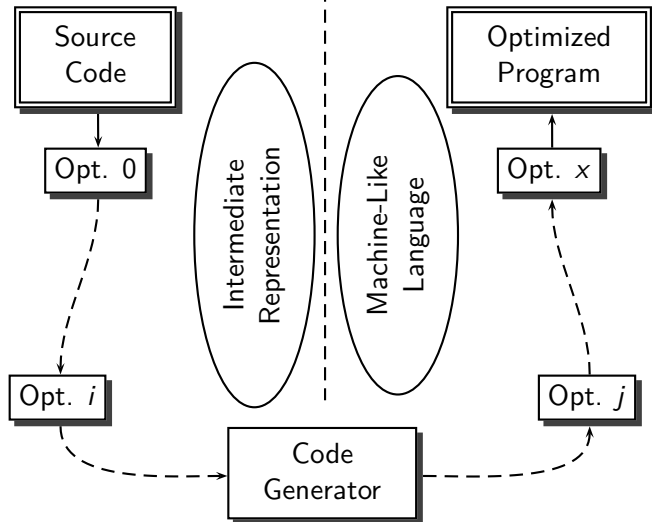
# Scientific Context

## State of the art

### Approaches and issues defined by [Hennessy 82, Zellweger 83]

- Two Approaches
  - Transparent  
Follow the source code [Jaramillo 00, Adl-Tabatabai 96]
  - Non-Transparent  
Follow the execution [CXdb 91, Tice 99]
- Main Issues
  - Data Value Problems [Adl-Tabatabai 96]
  - Code Location Problem [CXdb 91, Tice 99]

## Industrial Context



# Principles

## Compiler

- Debugging Information Structures
- Debugging Information Propagation

## Debugger

- Breakpoint mechanisms
- Command Line Interface (CLI)
- Machine Interface (MI)

## Exemple CSE

Source to optimized code

```
a = b + c / d;  
e = a + b;  
g = f + c / d;
```

## Exemple CSE

Source to optimized code

```
a = b + c / d;  
e = a + b;  
g = f + c / d;
```

```
tmp1 = c / d;  
a = b + tmp1;  
e = a + b;  
tmp2 = c / d;  
g = f + tmp2;
```

## Exemple CSE

Source to optimized code

```
a = b + c / d;
```

```
e = a + b;
```

```
g = f + c / d;
```

```
tmp1 = c / d;
```

```
a = b + tmp1;
```

```
e = a + b;
```

```
tmp2 = c / d;
```

```
g = f + tmp2;
```

```
tmp = c / d;
```

```
a = b + tmp;
```

```
e = a + b;
```

```
g = f + tmp;
```

# Exemple CSE

## idbug listing

```
(idbug) b myTest.c:15
Breakpoint 1 : file myTest.c, line 15.
(idbug) run
Starting program: /tmp/debug/example/myTest.elf
Breakpoint 1, in main () at myTest.c:15
15      int f = 2;
(idbug) stepi
16      a = b + c / d;
18      g = f + c / d;
      Common subexpression stored in a temporary variable.
(idbug) step
16      a = b + c / d;
      Subexpression hoisted.
(idbug) step
17      e = a + b;
(idbug) step
18      g = f + c / d;
      Subexpression hoisted.
(idbug) quit
```

# Conclusions

## The code location problem

- ST-MMDSP+ C Compiler
  - Information propagation
- Extended debug information
  - Data structure
- ST-IDbug debugging framework
  - Breakpoint mechanisms
  - CLI and MI

# Perspectives

- Dwarf-3
- Parallelism
  - Other software-pipelining algorithms .. ILP in general
  - Data parallelism
  - Task parallelism
- Transparent Debugging, two techniques:
  - Hidden breakpoints
  - Slicing
- Compiler Development
  - Refined dumps

# Questions ?

... any question ???

?