CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Extending a Compiler Backend for Complete Memory Error Detection

Norman A. Rink and Jeronimo Castrillon

Technische Universität Dresden

Automotive – Safety & Security 2017

30 May 2017

Stuttgart

TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

DFG

WR | WISSENSCHAFTSRAT

Outline

1. Motivation
2. Error detection, AN encoding
3. The extended compiler backend
4. Evaluation
5. Summary

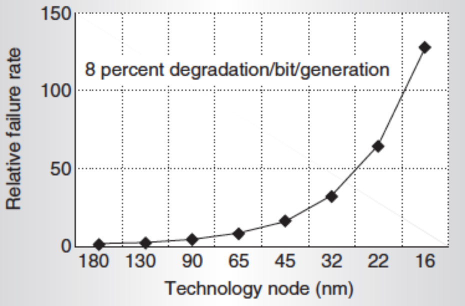
Outline

1. **Motivation**
2. Memory error detection, AN encoding
3. The extended compiler backend
4. Evaluation
5. Summary

Motivation

- Frequency of transient HW faults (aka. *soft errors*) is increasing.
  - Traditional cause of faults: cosmic rays.
  - Vulnerability is increasing due to smaller feature sizes and lower operating voltages.
  - Dark/dim silicon in memory modules:
    - Extended refresh cycles for DRAM.
    - Lower supply voltage for SRAM.

for energy efficiency



S. Borkar, "Designing reliable systems from unreliable components: …," IEEE Micro, vol. 25, no. 6, 2005.

- Memory errors: ECC memory modules have their limitations.
  - Typically SEC-DED codes (*single error correction, double error detection*).
  - Large fractions of memory errors cannot be handled by SEC-DED codes (Hwang et al., ASPLOS 2012).
  - ECC not necessarily extended to the entire memory hierarchy. (Load-store queues?)

Software-implemented error detection has the flexibility to detect also complex error patterns.

Norman Rink, norman.rink@tu-dresden.de

Software-implemented error detection

- Manual incorporation of integrity checks.
  - ✗ Laborious and cumbersome.
  - ✗ Mixes functional and non-functional requirements.
  - ✗ Requires expert knowledge.
  - ✗ Error detection limited to anticipated errors.

- Automated, disciplined approaches.
  - Enable comprehensive error detection.
  - Source-to-source transformation.
  - *Aspects*.
  - Compiler-based approaches:
    - Transformation of machine code.
    - Transformation of intermediate representation (IR).

popular in the late 90s and early 2000s

increasingly popular since the advent of the LLVM framework

gives access to sophisticated program analysis

```
var = a + b;
r = c * var;
```

```
check(a0, a1);
...
var0 = a0 + b0;
var1 = a1 + b1;
check(var0, var1);
...
r0 = c0 * var0;
r1 = c1 * var1;
check(r0, r1);
```


man.rink@

Limitations of software-implemented error detection

```
var = a + b;
r = c * var;
```

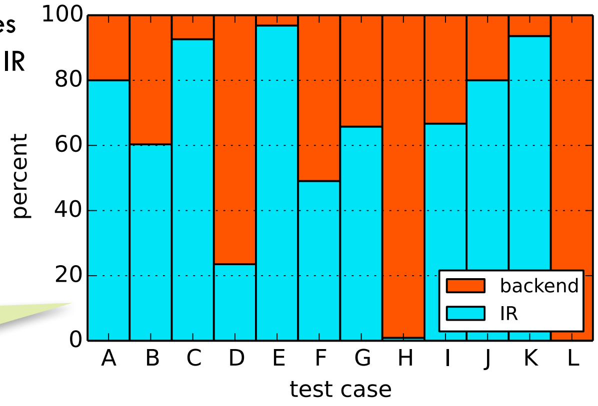
```
check(a0, a1);
...
var0 = a0 + b0;
var1 = a1 + b1;
check(var0, var1);
...
r0 = c0 * var0;
r1 = c1 * var1;
check(r0, r1);
```

- To detect errors in memory …
  - Which variables are kept in memory?
  - When are variables kept in memory?
  - Are there any *hidden* variables that are put into memory?

Ultimately, the compiler knows all this …
… but only very late!

Percentage of dynamic memory accesses (loads) that are present in the program IR or inserted by the compiler backend:

(Twelve test programs, labeled A-L.)



In some cases (H, L) virtually all loads are inserted by the compiler backend!

Norman Rink, norman.rink@tu-dresden.de

Outline

1. Motivation
2. <u>Memory error detection, AN encoding</u>
3. The extended compiler backend
4. Evaluation
5. Summary

CHAIR FOR COMPILER CONSTRUCTION

Memory error detection by DMR

- DMR (dual modular redundancy).
  - In the context of software-implemented error detection: duplication of data.

```
store i64 %0, i64* %p
...
%1 = load i64* %p
```

```
store i64 %0, i64* %p0
store i64 %0, i64* %p1
...
%10 = load i64* %p0
%11 = load i64* %p1

%f0 = icmp eq i64 %10, %r11
br i1 %f0, label continue,
          label recover
```

duplication of data

error detection

- DMR may introduce race conditions in multi-threaded applications.
  - State-of-the-art work usually assumes memory is protected by ECC (in hardware).

AN encoding

- AN encoding:
  - Fix an integer constant A.
  - Encode integer values by multiplying by A: $n_{enc} = n * A$
  - Decode by dividing by A: $n = n_{enc} / A$
  - Check for errors: $n_{enc} \ mod \ A = 0$
- Error-detecting capability varies with the constant A.
  - Generally, multi-bit errors can be detected by suitable A.
  - A = 58659 is known to have good properties; can detect up to 5 bit flips, Hoffmann et al., 2015.
- AN encoding introduces large overheads if used to protect operations: several **10*x*-100*x***.

Norman Rink, norman.rink@tu-dresden.de

Memory error detection by AN encoding (1)

- Detection of multi-bit errors in memory, including caches, load-store queues.

- Apply AN encoding only to values stored to memory → low overhead due to AN encoding.

encode before storing:

```
%01 = mul i64 %00,  A
store i64 %01, i64* %p
```

check and decode after loading:

```
%1 = load i64* %p
%2 = srem i64 %1, A

%f0 = icmp eq i64 %2, 0
br i1 %f0, label continue,
           label recover

%3 = sdiv i64 %2, A
...
```

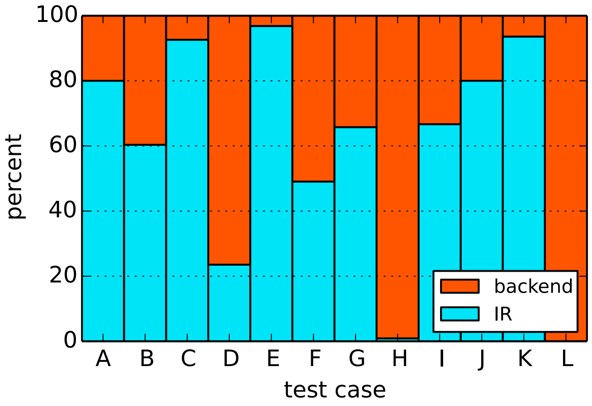
- AN encoding is applied at the LLVM IR level.
  - Common approach in software-implemented fault tolerance schemes.

→ Error detection at the IR level misses memory accesses that are inserted by the compiler backend.

Norman Rink, norman.rink@tu-dresden.de

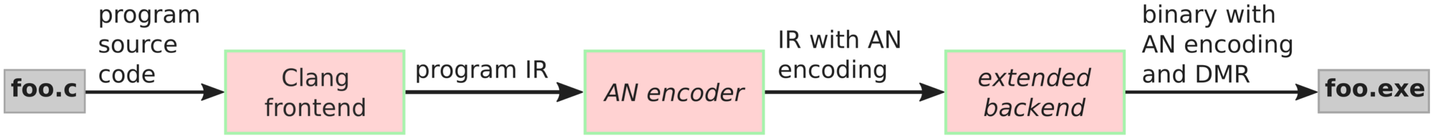
Memory error detection by AN encoding (2)

- Remember this plot:



- Backend for the C programming language inserts memory accesses for:
  - Register spills (*spill*).
  - Callee-saved registers (*csr*).
  - Frame pointer (*fptr*).
  - Return address (*return*).
  - Function arguments (*arg*).
  - Jump tables (*jt*).

implement function calls



Norman Rink, norman.rink@tu-dresden.de

CHAIR FOR COMPILER CONSTRUCTION

Outline

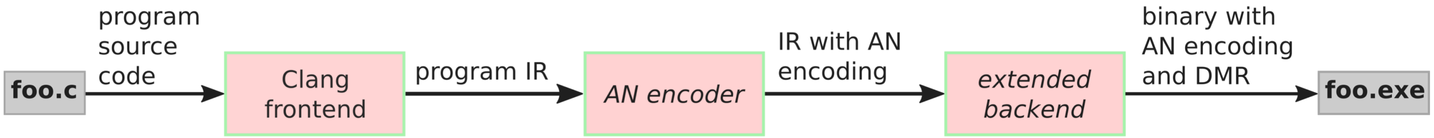
1. Motivation
2. Memory error detection, AN encoding
3. **The extended compiler backend**
4. Evaluation
5. Summary

The extended compiler backend

- Backend for the C programming language inserts memory accesses for:
  - Register spills (*spill*).
  - Callee-saved registers (*csr*).
  - Frame pointer (*fptr*).
  - Return address (*return*).
  - Function arguments (*arg*).
  - Jump tables (*jt*).

- Implement error detection in the compiler backend by DMR:
  - Faster than AN encoding.
  - Keeps function calls efficient.
  - Adds (almost) no register pressure.

- Duplicated store/load:
  - Additional memory accesses are ”cheap”.
  - Memory locations already in the cache.
  - (All) memory accesses are thread-local.



DMR for register spills

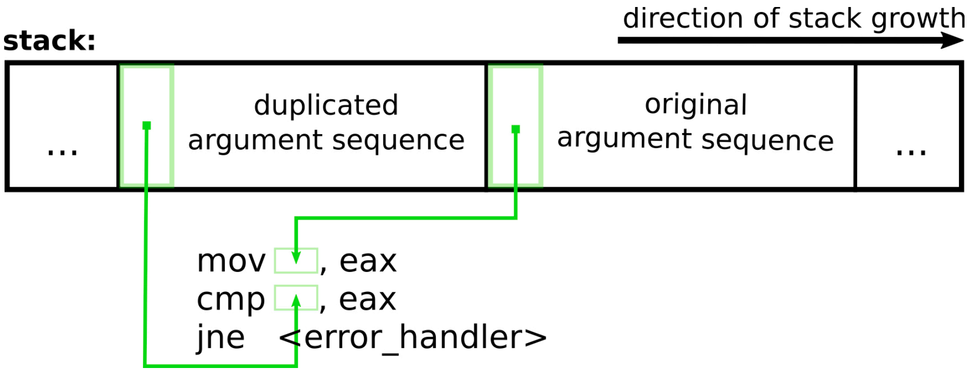
```
mov eax, -0x30(ebp)
...
mov -0x30(ebp), eax
add eax, (esi)
```

```
mov eax, -0x34(ebp)
mov eax, -0x30(ebp)
...
mov -0x30(ebp), eax
cmp -0x34(ebp), eax

jne <error_handler>

add eax, (esi)
```

- Comparison memory/register is specific to x86 – more generally, CISC machines.
- RISC machines? → `cmp mem/reg` might be sensible ISA extension.

Norman Rink, norman.rink@tu-dresden.de

DMR for function arguments



- Requires co-operation between caller and callee (modified calling convention).
- Library calls still work. (Caller can ignore duplicated arguments).
- The number of arguments passed on the stack may be low (depending on the architecture).

Norman Rink, norman.rink@tu-dresden.de

DMR for the return address

caller:

```
0x804a99e: ...
0x804a9a3: call <foo>
0x804a9a8: ...
```

callee ("foo"):

```
...
ret
```

caller:

```
0x804a99e: mov 0x804a9a8, ebx
0x804a9a3: call <foo>
0x804a9a8: ...
```

callee ("foo"):

```
push ebx
...
pop ebx
cmp (esp), ebx

jne <error_handler>

add 0x4, esp
jmp *ebx
```

- Modified calling convention: pass return address in register *ebx*.
- No modification required on, e.g., ARM or MIPS.

Norman Rink, norman.rink@tu-dresden.de

Outline

1. Motivation
2. Memory error detection, AN encoding
3. The extended compiler backend
4. **Evaluation**
5. Summary

Fault injection

- Assumptions:
  - Only a single fault affect program execution.
  - Only single bit flips occurs.

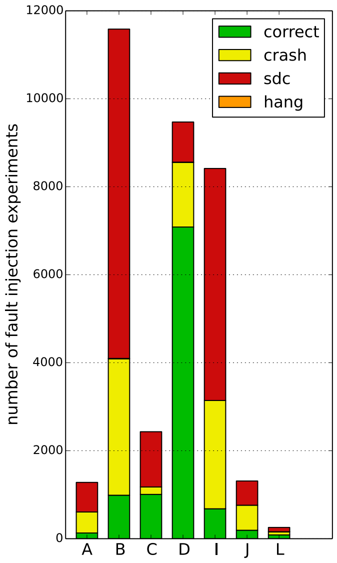
Commonly justified by the rarity of faults.
(**SEU** – single event upset)

- Simulate symptoms of faults by …
  - … flipping a bit in a memory location that is loaded from.
- Perform exhaustive fault injections:
  - Flip a bit in all possible locations in all loads from memory

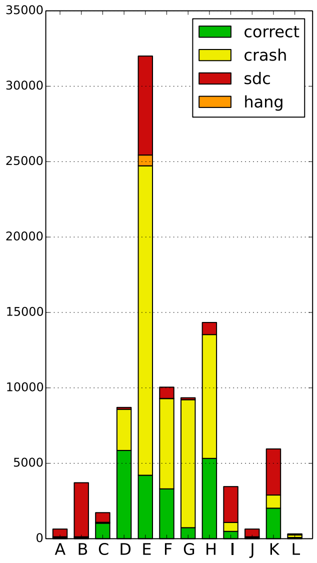
letter	test case
A	array reduction
B	bubblesort
C	CRC-32
D	DES encryption
E	Dijkstra (shortest path)
F	expression evaluation
G	token lexer
H	expression parser
I	matrix multiplication
J	array copy
K	quicksort
L	switch

Full memory error detection

no error detection:

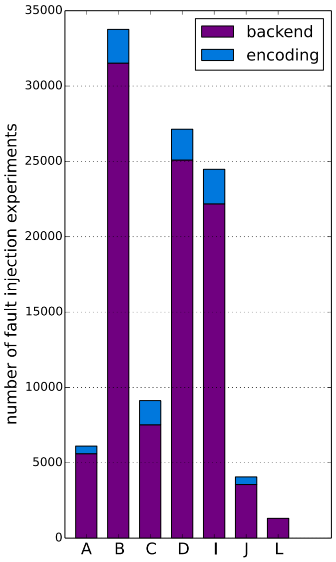


i386 (32bit)

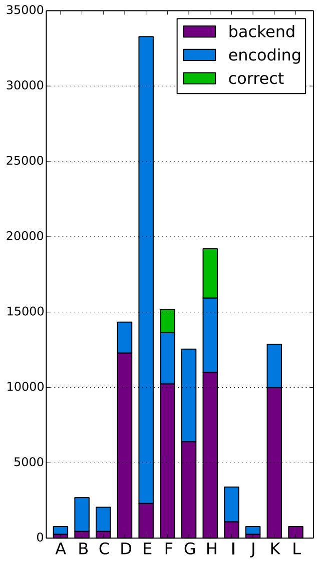


x86_64 (64bit)

AN encoding and DMR in the backend:



i386 (32bit)



x86_64 (64bit)

Norman Rink, norman.rink@tu-dresden.de

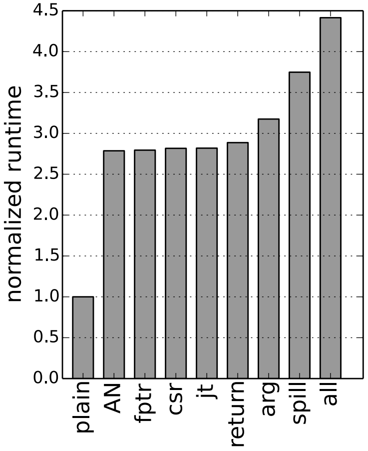
CHAIR FOR COMPILER CONSTRUCTION

Runtime overhead

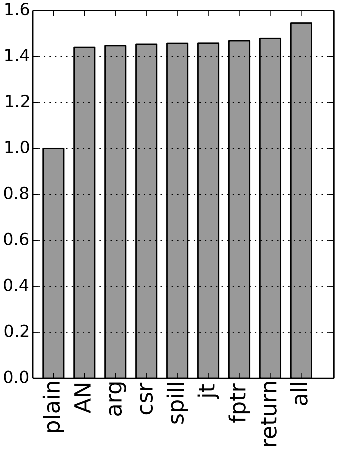
AN encoding dominates the slow down.

Slow down dominated by register spills.

Test programs:

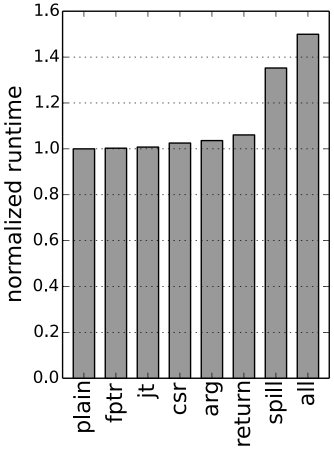


i386 (32bit)

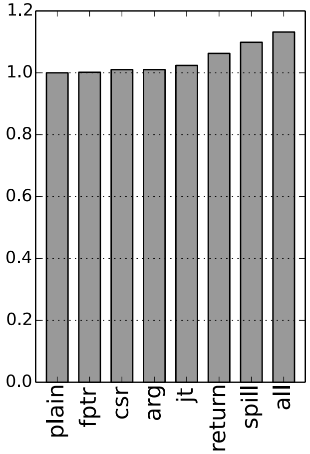


x86_64 (64bit)

Subset of SPEC CINT2006:



i386 (32bit)



x86_64 (64bit)

Outline

1. Motivation
2. Memory error detection, AN encoding
3. The extended compiler backend
4. Evaluation
5. <u>Summary</u>

CHAIR FOR COMPILER CONSTRUCTION

Summary

- Automatic code transformation that introduced memory error detection not comprehensive when applied above the level of machine code.
  - Transformations at the level of source code or IR desirable for productivity.

- Supporting memory error detection with DMR introduced by the compiler backend …
  - … leads to full memory error detection,
  - … incurs a runtime overhead of
    - 1.50 on i386 (SPEC CINT2006),
    - 1.13 on x86_64 (SPEC CINT 2006).

- Absence of vulnerabilities introduced by the compiler backend required for …
  - … (reliable analysis/evaluation of) relaxed fault tolerance schemes.
  - … applications with strict safety and reliability requirements.

- The stack has been found a major weakness.

Norman Rink, norman.rink@tu-dresden.de

CHAIR FOR COMPILER CONSTRUCTION

Extending a Compiler Backend for Complete Memory Error Detection

Norman A. Rink and Jeronimo Castrillon

Technische Universität Dresden

Thank you.