Hardening an L4 Microkernel Against Soft Errors

by Aspect-Oriented Programming and Whole-Program Analysis

Christoph Borchert and Olaf Spinczyk

http://ess.cs.tu-dortmund.de/
Memory Errors are Commonplace!

- **DRAM fault rate:** $10^{-8}$ FIT/bit [1,2]
  - FIT: expected failures per $10^9$ hours
  - Scales with Moore's Law

- **Example:** “Jaguar” supercomputer at Oak Ridge, Tennessee
  - 300 terabytes → “one failure approximately every six hours” [2]

---


Considering Integrity of OS Kernel Data

• Kernels are …
  – … **small** (1 % RAM)
  – … **essential** for *all* application programs
  – … **exposed** to memory faults *all the OS uptime*

• Memory faults should be mitigated there!
  – Need for software-based error correction
  – **Problem**: Manual implementation in C/C++ → tedious, error-prone
Programming Language Support

- Memory-error correction as **generic module**
  - "Pluggable" into various kernel data structures (**C/C++ structs/objects**)
- **AspectC++** compiler support
  - Aspect-Oriented Programming (AOP)
Outline

- Motivation and Idea
- **Generic Object Protection with AspectC++**
- Whole-Program Optimization
- Evaluation
Idea: Generic Object Protection (GOP)

- Extend kernel objects by error-correcting code

- **Check** that code *before* ...
  - **Invocation** of a member function
  - **Field access** (within non-member function)

... and update it *afterwards*

- When **leaving** the object's scope:
  - Update the code, and check on return
GOP (1/3): Class Extension

```cpp
aspect GOP {
    pointcut critical() = "Cpu" || "Timeout_q";

    advice critical() : slice class {
        HammingCode<JoinPoint> code;
    }

    advice construction(critical()) : after() {
        tjp->target()->code.update();
    }
}
...
```
GOP (1/3): Class Extension

```cpp
aspect GOP {
    pointcut critical() = "Cpu" || "Timeout_q";

    advice critical() : slice class {
        HammingCode<JoinPoint> code;
    };

    advice construction(critical()) : after() {
        tjp->target()->code.update();
    }

    ...
}
```
GOP (1/3): Class Extension

aspect GOP {
    pointcut critical() = "Cpu" || "Timeout_q";

    advice critical() : slice class {
        HammingCode<JoinPoint> code;
    }

    advice construction(critical()) : after() {
        tjp->target()->code.update();
    }

    ...
}
GOP (1/3): Class Extension

```java
aspect GOP {
    pointcut critical() = "Cpu" || "Timeout_q";

    advice critical() : slice class {
        HammingCode<JoinPoint> code;
    }

    advice construction(critical()) : after() {
        tjp->target()->code.update();
    }
    ...
}
```

Interface to a compile-time introspection API
**GOP (1/3): Class Extension**

```java
aspect GOP {
  pointcut critical() = "Cpu" || "Timeout_q";
  advice critical() : slice class {
    HammingCode<JoinPoint> code;
  }
  advice construction(critical()) : after() {
    tjp->target()->code.update();
  }
}
```

- **BASECLASSES**
  - number of base classes of the described class
  - `BaseClass<I>::Type` type of the $I^{th}$ base class
  - `BaseClass<I>::prot, BaseClass<I>::spec` Protection level (AC::PROT_NONE /PRIVATE /PROTECTED /PUBLIC) and additional specifiers (AC::SPEC_NONE /VIRTUAL) of the $I^{th}$ base class

- **MEMBERS**
  - number of attributes of the target class
  - `Member<I>::Type, Member<I>::ReferredType` type of the $I^{th}$ member sub-object of the described class
  - `Member<I>::prot, Member<I>::spec` Protection level (see BaseClass<I>::prot) and additional member specifiers (AC::SPEC_NONE /STATIC /MUTABLE)

- **FUNCTIONS**
  - number of member functions
  - `Function<I>::prot, Function<I>::spec` Protection level (see BaseClass<I>::prot) and additional member function specifiers (AC::SPEC_NONE /STATIC /VIRTUAL)

- **FUNCTIONS**
  - number of member functions
  - `Function<I>::prot, Function<I>::spec` Protection level (see BaseClass<I>::prot) and additional member function specifiers (AC::SPEC_NONE /STATIC /VIRTUAL)
GOP (1/3): Class Extension

```cpp
aspect GOP {
  pointcut critical() = "Cpu" || "Timeout_q";

  advice critical() : slice class {
    HammingCode<JoinPoint> code;
  };

  advice construction(critical()) : after() {
    tjp->target()->code.update();
  }

  ...
}
```

Matches constructor execution
GOP (1/3): Class Extension

```
aspect GOP {
    pointcut critical() = "Cpu" || "Timeout_q";

    advice critical() : slice class {
        HammingCode<JoinPoint> code;
    };

    advice construction(critical()) : after() {
        tjp->target()->code.update();
    }
...
GOP (1/3): Class Extension

```java
aspect GOP {
    pointcut critical() = "Cpu" || "Timeout_q";

    advice critical() : slice class {
        HammingCode<JoinPoint> code;
    }

    advice construction(critical()) : after() {
        tjp->target()->code.update();
    }

    ...
}
```

Yields a pointer to the particular object
GOP (2/3): Advice for Object Access

pointcut check() = call(member(critical())) ||
                 get(member(critical())) ||
                 set(member(critical()));

pointcut update() = /* only call and set */

advice check() : before () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.check();
    }
}

advice update() : after () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.update();
    }
}
GOP (2/3): Advice for Object Access

```java
class A {
    void check() {
        if (tjp->that() != tjp->target()) {
            tjp->target()->code.check();
        }
    }
}

class B {
    void update() {
        if (tjp->that() != tjp->target()) {
            tjp->target()->code.update();
        }
    }
}
```

```
pointcut check() = call(member(critical())) || get(member(critical())) || set(member(critical()));

pointcut update() = /* only call and set */

advice check() :
    before() {
        if (tjp->that() != tjp->target()) {
            tjp->target()->code.check();
        }
    }

advice update() :
    after() {
        if (tjp->that() != tjp->target()) {
            tjp->target()->code.update();
        }
    }
```
Matches every invocation of a member function

```java
pointcut check() = call(member(critical())) ||
                    get(member(critical())) ||
                    set(member(critical()));

pointcut update() = /* only call and set */

advice check() : before () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.check();
    }
}

advice update() : after () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.update();
    }
}
pointcut check() = call(member(critical())) || get(member(critical())) || set(member(critical()));

pointcut update() = /* only call and set */

advice check() : before () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.check();
    }
}

advice update() : after () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.update();
    }
}
GOP (2/3): Advice for Object Access

```java
pointcut check() = call(member(critical())) ||
                get(member(critical())) ||
                set(member(critical()));

pointcut update() = /* only call */

advice check() : before () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.check();
    }
}

advice update() : after () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.update();
    }
}
```

Before call/get/set events ...

... invoke check()
pointcut check() = call(member(critical())) ||
get(member(critical())) ||
set(member(critical()));
pointcut update() = /* only call and set */

advice check() : before () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.check();
    }
}

advice update() : after () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.update();
    }
}
**GOP (2/3): Advice for Object Access**

```java
pointcut check() = call(member(critical())) ||
    get(member(critical())) ||
    set(member(critical()));

pointcut update() = /* only call and set */

advice check() : before () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.check();
    }
}

advice update() : after () {
    if (tjp->that() != tjp->target()) {
        tjp->target()->code.update();
    }
}
```

Don't check when the caller and callee are identical ("recursion")
GOP (3/3): Leaving an Object's Scope

```java
pointcut
leave() = call("% ...::%(...)") && within(member(critical()));

advice leave() : before() {
if (tjp->that() != tjp->target()) {
tjp->that()->code.update();
}
}

advice leave() : after() {
if (tjp->that() != tjp->target()) {
tjp->that()->code.check();
}
}
```
More GOP Features

• Protection of
  – Virtual-function pointers (vptr)
  – Static data members

• Choice of Hamming code or CRC32 (SSE4 instructions)

• Optimizations for read-only (const) functions

• Inheritance and polymorphism

• Non-blocking synchronization
Outline

• Motivation and Idea
• Generic Object Protection with AspectC++
• Whole-Program Optimization
• Evaluation
Problem: There are Unneeded Checks!

- Short-running functions
  - e.g., inline getters and setters

- Call sequences on the same object
Problem: There are Unneeded Checks!

- Short-running functions
  - e.g., inline getters and setters

- Call sequences on the same object

Idea: Optimize-out unneeded checks at compile time!
Whole-Program Analysis/Optimization

1. Static analysis

2. Optimization

3. Hardened OS Kernel

#include

Project Repository (XML)

Point-cuts

XQuery

GOP Aspect

Aspect
Outline

• Motivation and Idea
• Generic Object Protection with AspectC++
• Whole-Program Optimization
• Evaluation
Case Study: L4/Fiasco.OC\textsuperscript{1} \(\mu\)-kernel

- Real-time kernel for x86/x64/ARM, open source (C++)

- 7 benchmark programs (shipped with L4/Fiasco.OC)
  - Testing the \(\mu\)-kernel essentials
    - Thread scheduling
    - Inter-process communication (ipc)
    - Interrupt requests (irq)
    - Shared-memory management
    - Access control

- 4 kernel variants
  - Baseline
  - VPtr (virtual-function pointers)
  - GOP (all data members + vptr)
  - GOP-S (static optimization)

Hardening 26 classes

\textsuperscript{1} http://os.inf.tu-dresden.de/fiasco/
Assessment of Fault Tolerance

- **Fault model:** Single-bit errors in memory
  - Uniformly distributed over the kernel address space

- **Fault injection:** One *random* bit flips in one benchmark run
  - 100,000 runs per kernel variant and benchmark program
  - Extrapolate the counted number of failed program runs

- **Fault injection tool:** FAIL*, a modified Bochs x86 emulator
  - Trace-based optimizations only injecting faults into live memory
Fault Injection: Failed Program Runs

- clntsrv
- ipc
- map_irq
- shared_ds
- streammap
- uirq
- utcb-ipc

Experiment result:
- SDC
- Timeout
- CPU Exception
Fault Injection: Failed Program Runs

- **Total reduction**
  - VPtr: -12 %
  - GOP: -59 %
  - GOP-S: -60 %
Overhead: Dynamic CPU Instructions

- Total overhead
  - VPtr: 1.01x
  - GOP: 3.5x
  - GOP-S: 2.3x
Overhead: Dynamic CPU Instructions

This is only kernel time!

Runtime overhead <1% (kernel + application)

- VPtr: 1.01x
- GOP: 3.5x
- GOP-S: 2.3x
Summary and Future Work

- **Generic Object Protection** prevents 60% of kernel failures
  - Only 26 classes protected, yet

- Whole-program analysis *improves* fault tolerance
  - Dynamic instruction overhead: 3.5x → 2.3x

- Embed whole-program analysis into the AspectC++ language
  - Query the call graph ("is function x reachable from here?")
  - Advice for call sequences via regular expressions (call?, call*, call+)