Multiverse: Compiler-Assisted Management of Dynamic Variability in Low-Level System Software

Florian Rommel    Christian Dietrich    Michael Rodin    Daniel Lohmann

{rommel, dietrich, lohmann}@sra.uni-hannover.de    michael@rodin.online

Leibniz Universität Hannover, Germany

2019-03-28
void __init sched_init(void) {
    int i, j;
    unsigned long alloc_size = 0, ptr;

    wait_bit_init();

    ifdef CONFIG_FAIR_GROUP_SCHED
    alloc_size += 2 * nr_cpu_ids * sizeof(void **);
    endif

    ifdef CONFIG_RT_GROUP_SCHED
    alloc_size += 2 * nr_cpu_ids * sizeof(void **);
    endif

    if (alloc_size) {
        ptr = (unsigned long)kzalloc(alloc_size, GFP_NOWAIT);

        ifdef CONFIG_FAIR_GROUP_SCHED
        root_task_group.se = (struct sched_entity **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);

        root_task_group.cfs_rq = (struct cfs_rq **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);
        endif

        /* CONFIG_FAIR_GROUP_SCHED */
        ifndef CONFIG_FAIR_GROUP_SCHED
        root_task_group.rt_se = (struct sched_rt_entity **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);
        endif
    }
}
Dynamic Variability in Linux

Example: Operations for Paravirtualized Kernels (PV-Ops)

- Inside paravirtualization: Privileged operations must be replaced by calls to the hypervisor (e.g., enable/disable interrupts)
- Implemented by function pointers → too much overhead
- Run-time binary patching: Replace indirect calls by direct calls
Complex implementation:

PV-Ops: 7 files, ~2000 loc (for x86)

Highly architecture-dependent: Multiple implementations

Highly problem-specific: Multiple implementations

e.g., alternative instructions, SMP alternatives

→ Means for efficient dynamic variability are rarely used
Efficient Dynamic Variability

Problems:
- Default approach: Performance costs (e.g., branches, function pointers)
- Binary patching: Code complexity → maintenance costs

Multiverse
Compiler-Assisted Dynamic Variability via Binary Patching

Language extension to express efficient dynamic variability
- Express binary patching via standard control flow modification (if, ...)
- Generic mechanism for function-level run-time patching
  → Compiler plugin + small run-time library
Example: Linux Lock Elision

Linux Spinlock Implementation (simplified):

CONFIG_SMP set in the build system

```c
void spin_irq_lock(raw_spinlock_t *lock) {
    #ifdef CONFIG_SMP
        irq_disable();
        spin_acquire(&lock);
    #else
        irq_disable();
    #endif
}
```
Dynamic Linux Lock Elision

```c
bool smp;

void spin_irq_lock(...) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}
```
__attribute__((multiverse))
bool smp;

__attribute__((multiverse))
void spin_irq_lock(...) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}

void foo(void) {
    smp = true;
    multiverse_commit();
    // ...
    spin_irq_lock(lock);
}
Multiversed Linux Lock Elision

```c
__attribute__((multiverse))
bool smp;

__attribute__((multiverse))
void spin_irq_lock(...) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}

void foo(void) {
    smp = true;
    multiverse_commit();
    // ...
    spin_irq_lock(lock);
}
```
Multiversed Linux Lock Elision

```c
__attribute__((multiverse))
bool smp;

__attribute__((multiverse))
void spin_irq_lock(...) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}

void foo(void) {
    smp = false;
    multiverse_commit();
    // ...
    spin_irq_lock(lock);
}
```
Ahead-of-Time Variant Generation

Source Code

__attribute__((multiverse))
bool smp;

void spin_irq_lock(...)
{
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}

void foo()
{
    multiverse_commit();
    spin_irq_lock();
    //...
}

Code Segment

spin_irq_lock.smp=1:
    cli
    call spin_acquire
    ret

spin_irq_lock.smp=0:
    cli
    ret

spin_irq_lock:
    cmp <smp>, 0
    je .else
    cli
    call spin_acquire
    ret
    .else:
    cli
    ret
Run-Time Patching

Initially Loaded Binary

```c
foo:
  ...
  call multiverse_commit
  ...
  call spin_irq_lock
  ...
  ret

spin_irq_lock.smp=1:
  cli
  call spin_acquire
  ret

spin_irq_lock.smp=0:
  cli
  ret

spin_irq_lock:
  cmp <smp>, 0
  je .else
  cli
  call spin_acquire
  ret

  .else:
    cli
    ret
```

Multiverse Descriptors
Run-Time Patching

Patched (smp == 1)

```
spin_irq_lock.smp=1:
  cli
  call spin_acquire
  ret

spin_irq_lock.smp=0:
  cli
  ret

spin_irq_lock:
  jmp spin_irq_lock.smp=1
  je .else
  cli
  call spin_acquire
  ret
 .else:
    cli
    ret
```

```
foo:
  ...
  call multiverse_commit
  ...
  call spin_irq_lock.smp=1
  ...
  ret
```

Multiverse Descriptors
Run-Time Patching

Patched (smp == 0)

foo:
  ...
call multiverse_commit
  ...
cli
  ...ret

Callsite Inlining!

spin_irq_lock.smp=1:
  cli
  call spin_acquire
  ret

spin_irq_lock.smp=0:
  cli
  ret

spin_irq_lock:
  jmp spin_irq_lock.smp=0
  je .else
  cli
  call spin_acquire
  ret
  .else:
  cli
  ret

Multiverse Descriptors
Evaluation – Kernel Microbenchmarks

Linux Kernel – Lock Elision
(spinlock_irq_enable/disable)

Mean Run-Time (cycles)

SMP | Uniprocessor

Linux Kernel – PV-Ops
(enable/disable interrupts)

Mean Run-Time (cycles)

Native | XEN (guest)

Lock Elision in Kernel 4.16:
→ 1161 spin-lock call sites
→ +40 KiB size (zipped total: 10 MiB)
Evaluation – Userspace Microbenchmarks

musl libc – Lock Elision
(single threaded)

(multi threaded)

Mean Run-Time (cycles)

<table>
<thead>
<tr>
<th>Function</th>
<th>no multiverse (default)</th>
<th>multiverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>random()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>malloc(0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>malloc(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fputc(‘a’)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

random(), malloc(0), malloc(1), fputc(‘a’)

2019-03-28
Multiverse
Evaluation

- GNU Grep
  → Optimized for more than 30 years
  *Multiversed a conditional branch in the inner loop (recognition of multi-byte characters on/off)*
  → End-to-end measurement:  \(-2.73\%\) run-time
  → Only 50 changed lines to use multiverse

- Multiverse – Numbers
  
<table>
<thead>
<tr>
<th>GCC Plugin</th>
<th>Run-Time Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code: &lt;1200 + compatibility headers: ~1300</td>
<td>Lines of Code: &lt; 850</td>
</tr>
<tr>
<td>GCC version: 6.3 and higher</td>
<td>Compiled: 6.5 KiB</td>
</tr>
<tr>
<td></td>
<td>Architectures: IA-32, AMD64 ARM [to come]</td>
</tr>
</tbody>
</table>
Multiverse
Compiler-Assisted Dynamic Variability via Binary Patching

- Language extension for easy-to-use, efficient dynamic variability
  GCC-Plugin: Generates specialized function variants
  Run-Time Library: Function-level binary patching

- Evaluation
  - Consolidation of current patching mechanisms (PV-Ops)
  - Introduction of new dynamic variation points (Lock Elision, Grep)

- Try it: https://github.com/luhsra/multiverse