Large-Scale Code Restructuring and Tooling with Coccinelle

Michele MARTONE

Leibniz Supercomputing Centre (Garching bei München, Germany)

45TH VI-HPS Tuning Workshop
LRZ, June 10, 2024

The Linux Kernel

- runs everywhere
- probably on your phone now
- mostly C language code

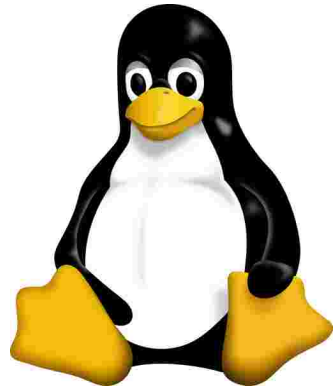


Figure: Linux mascot Tux by Larry Ewing

Bug: !x&y[1]

```
commit e6bafba5b4765a5a252f1b8d31cbf6d2459da337
Author: Al Viro <viro@ZenIV.linux.org.uk>
Date:   Wed Feb 13 04:03:25 2008 +0000

    wmi: (!x & y) strikes again

    Signed-off-by: Al Viro <viro@zeniv.linux.org.uk>
    Acked-by: Carlos Corbacho <carlos@strangeworlds.co.uk>
    Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

diff --git a/drivers/acpi/wmi.c b/drivers/acpi/wmi.c
index 457ed3d3f51c..efacc9f8bfe3 100644
--- a/drivers/acpi/wmi.c
+++ b/drivers/acpi/wmi.c
@@ -247,7 +247,7 @@ u32 method_id, const struct acpi_buffer *in, struct acpi_buffer *out)
        block = &wblock->gblock;
        handle = wblock->handle;

-       if (!block->flags & ACPI_WMI_METHOD)
+       if (!(block->flags & ACPI_WMI_METHOD))
                return AE_BAD_DATA;

        if (block->instance_count < instance)
```

[1] See https://coccinelle.gitlabpages.inria.fr/website/papers/fosdem10.pdf

Bug: possible NULL dereference[2]

```
commit 3c8a9c63d5fd738c261bd0ceece04d9c8357ca13
Author: Mariusz Kozlowski <m.kozlowski@tuxland.pl>
Date:   Sun Jul 5 19:48:35 2009 +0000

    tun/tap: Fix crashes if open() /dev/net/tun and then poll() it.

...
diff --git a/drivers/net/tun.c b/drivers/net/tun.c
index b393536012fb..027f7aba26af 100644
--- a/drivers/net/tun.c
+++ b/drivers/net/tun.c
@@ -486,12 +486,14 @@ static unsigned int tun_chr_poll(struct file *file, poll_table * wait)
 {
        struct tun_file *tfile = file->private_data;
        struct tun_struct *tun = __tun_get(tfile);
-       struct sock *sk = tun->sk;
+       struct sock *sk;
        unsigned int mask = 0;

        if (!tun)
                return POLLERR;

+       sk = tun->sk;
+
        DBG(KERN_INFO "%s: tun_chr_poll\n", tun->dev->name);

        poll_wait(file, &tun->socket.wait, wait);
```

[2] See https://coccinelle.gitlabpages.inria.fr/website/papers/fosdem10.pdf

Coccinelle, a tool *to identify* and *to eliminate* bugs

Rules with patch to fix the !x&y bug[3]

```
@@
expression E;
constant C;
@@

- !E & C
+ !(E & C)
```

[3] See https://coccinelle.gitlabpages.inria.fr/website/papers/fosdem10.pdf

Rules to detect possible NULL dereference bugs (sketch)[4]

```
@@
expression x;
identifier fld;
@@
* x->fld
  ...
* x == NULL
```

- *isomorphisms* cause `x == NULL` to also match eg `!x`.

[4] See https://coccinelle.gitlabpages.inria.fr/website/papers/fosdem10.pdf

Rules to detect possible NULL dereference bugs[5]

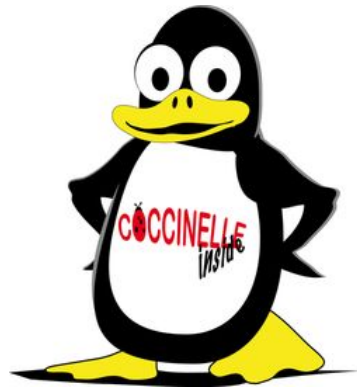
```
@@
type T;
identifier i,fld;
expression E;
statement S;
@@
- T i = E->fld;
+ T i;
  ... when != E
      when != i
  if (E == NULL) S
+ i = E->fld;
```

- fewer false positives than previous rule

[5] See https://coccinelle.gitlabpages.inria.fr/website/papers/fosdem10.pdf

What is Coccinelle?

- regexps (`sed`, `awk`, `perl`, `python`...)?
- a *source-to-source translator*
- *semantic patching*
- safeguards Linux from bugs
- used for updating device drivers
- main author: Julia LAWALL (Inria)



Another use case

Suppose you have a

- large C simulation code base ($>$ 100kLoC)
- developed for many years, by many people

Lots of loop accesses

E.g.

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

What could be most obvious optimization (for CPUs)?

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

What could be most obvious optimization (for CPUs)?

Hint: S[i] are structures

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

Change AoS into Structure-of-Arrays (SoA)

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

$\Rightarrow$

```
for(i = 0; i < N; ++i)
  {
    SoA.Metals[i] = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = SoA.Metals[i] * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

AoS to SoA in a Nutshell

```
@@
identifier q =~ "prsr";
identifier s = {P};
identifier i = {i,k};
fresh identifier n = s ## "_" ## q;
@@
- s[i].q
+ n[i]
```

basicaos2soa1.cocci

```
@@ -1,9 +1,9 @@
 struct p_t {
   double prsr, vel;
 };

 p_t P[3];
 int main() {
         int i,j,k;
-        P[i].prsr++;
+        P_prsr[i]++;
 }
```

basicaos2soa1.diff

Current focus of Coccinelle

Managing complexity

- ease code annotation
- GPU *language extensions* (CUDA, HIP, ...)
- modern C++ features (mdspan, SYCL, ...)
- advanced `#pragma` manipulation: OpenMP, ...
- enforcing C++ coding guidelines

Ease code annotation

```
@@ @@
  #include <omp.h>
+ #include <caliper/cali.h>

@@ @@
+ CALI_MARK_BEGIN(__func__);
+ CALI_MARK_BEGIN("parallel");
  #pragma omp parallel
  {
    ...
  }
+ CALI_MARK_END("parallel");
+ CALI_MARK_END(__func__);
```

caliper1.cocci

```
@@ -1,8 +1,13 @@
 #include <omp.h>
+#include <caliper/cali.h>
 int f() {
-  #pragma omp parallel
+  CALI_MARK_BEGIN(__func__);
+  CALI_MARK_BEGIN("parallel");
+  #pragma omp parallel
   {
     // expect loops here
   }
+  CALI_MARK_END("parallel");
+  CALI_MARK_END(__func__);
 }
 int main() { }
```

caliper1.diff

CUDA GPUs extensions

```
#spatch --smpl-spacing
@@
attribute name __device__;
@@
+ __device__
  int f(void) {
+   const int gthid =
+     blockIdx.x * blockDim.x
+                + threadIdx.x;
    ...
 }

@@
identifier d,b;
@@
  int d;
  int b;
- f();
+ f<<<d,(d+b-1)/b>>>();
```

```
@@ -1,10 +1,13 @@
-int f(void)
+__device__ int f(void)
 {
+  const int gthid =
+      blockIdx.x * blockDim.x
+                 + threadIdx.x;
   { /* loops */ }
 }
 int main(void)
 {
   int ds;
   int bs;
-  f();
+  f<<<ds,(ds+bs-1)/bs>>>();
 }
```

cuda.diff

C++23 multi-index operators

```
# spatch --c++=23
@tomultiindex@
symbol a,i,j;
expression k;
@@
- a[i][j][k]
+ a[i, j, k]

@@
symbol b;
@@
- b[...]
+ b[0]
```

mdspan1.cocci

```
@@ -1,8 +1,8 @@
 int main()
 {
    int a[1][1][1];
    int b[1][1][1];
    int i=0,j=0,k=0;
-   a[i][j][k]++;
-   b[i][j][k]++;
+   a[i, j, k]++;
+   b[0][j][k]++;
 }
```

mdspan1.diff

Decluttering with OpenMP 5.1 pragmas

```
@@
identifier c, f, l;
expression b, k;
type T;
@@

+ const T j = k;
+ #pragma omp unroll partial (j)
  for (T c=0; c
-                 + k - 1
                          < l ; ... )
 {
-        f(b+0);
-        f(b+1);
-        f(b+2);
-        f(b+3);
+        f(b);
 }
```

deunroll1.cocci

```
@@ -1,12 +1,11 @@
 int f(int i){}
 int main()
 {
         const int n = 13;
-        for (int i=0;i+4-1<n;i++)
+        const int j = 4;
+#pragma omp unroll partial (j)
+        for (int i=0;i<n;i++)
         {
-                f(i+0);
-                f(i+1);
-                f(i+2);
-                f(i+3);
+                f(i);
         }
 }
```

deunroll1.diff

C++ Core Guideline F.17 (Bjarne Stroustrup, Herb Sutter)

```
#spatch --c++
@r1@
type T;
identifier f;
parameter list pl;
@@
T f(pl) { ... }

@r2@
typedef A, B;
type heavy_type={A,B};
type r1.T;
identifier r1.f;
symbol i;
@@
+// Note: heavy copy!
  T f (
  ..., heavy_type i, ...
  ) { ... }
```

cppcg_f17.cocci

```
@@ -1,13 +1,16 @@
 #include <array>
 struct A { std::array<int,999> a; }; // heavy
 struct B { std::array<int,999> a; }; // heavy
 struct C { std::array<int, 16> a; }; // light
 void if1(int i) {}
 void if2(int &arg) {}
 void if3(const int &arg) {}
+// Note: heavy copy!
 void af1(A i) {}
+// Note: heavy copy!
 void bf1(B i) {}
+// Note: heavy copy!
 void bf2(const B i) {}
 void bf3(const B & i) {}
 void cf1(C i) {}
 int main() { }
```

cppcg_f17.diff

For "in" parameters, pass cheaply-copied types by value and others by reference to const

No Raw Loops (Sean Parent)

```
#spatch --c++=17
@@ @@
  #include <iostream>
+ #include <algorithm>
+ #include <functional>

@@
type T;
constant k;
identifier elem,result,arrid;
@@
-  bool result = false;
   ...
-  for ( T &elem : arrid )
-  if ( \( elem == k \| k == elem \) )
-    {
-      ...
-      result = true;
-      break;
-    }
+ const bool result =
+   (find(begin(arrid),end(arrid),k) !=
+    end(arrid));
```

norawloops.cocci

```
@@ -1,20 +1,15 @@
 #include <vector>
 #include <iostream>
+#include <algorithm>
+#include <functional>
 int main()
 {
   using namespace std;
   vector v = {1,2,3};
-  bool has_zero = false;

   v[2] = 0;

-  for ( int & a : v )
-    if (0 == a)
-    {
-      cout << "doing things\n";
-
-      has_zero = true;
-      break;
-    }
+  const bool has_zero =
+    (find(begin(v), end(v), 0) != v.end());
   cout << has_zero << endl;
 }
```

norawloops.diff

Current state

- C language covered well
- working on
  - covering C++
  - easing further use cases

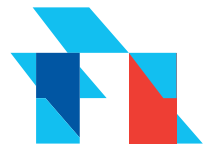
Some possible uses

- writing tests / enforcing properties
- throwaway performance experiments
- clean up code
- introducing tooling

Acknowledgements

LRZ-Inria collaboration made possible by

- SiVeGCS Project
- BayFrance travel grant



Resources

- Coccinelle's GitHub page
  https://github.com/coccinelle/coccinelle
- (also RUST version in preparation, see website)
- 247-page training from 2019
  https://www.lrz.de/services/compute/courses/x_lecturenotes/hspc1w19.pdf
- Misc Coccinelle things
  https://martone.userweb.mwn.de/