Through the Steps of Programmable Refactoring of a Large Scientific Code

Michele MARTONE

Leibniz Supercomputing Centre
Garching bei München, Germany

Università degli studi di Verona
November 25, 2019
The GADGET simulation code

- Cosmological large-scale structure formation (galaxies and clusters)
- Highly scalable ($O(100k)$ Xeon cores on SuperMUC@LRZ)
- Several teams and versions ($>100$ kLoC each)

---

Performance pilot study at LRZ

- L. Iapichino, V. Karakasis, F. Baruffa, N. Hammer
- spanned over one year
- focused on 1kLoC extract
- identified changes meant for whole GADGET

```
svn:29684
```

```
gkernel
```

```
git:0
```

```
git:77
```

```
copy
```

```
extract
```

michele DOT martone AT lrz DOT de
Speedup requires data layout change

main header:

```c
struct particle {
    double Mass, ...
    #if defined(BLACK_HOLES)...
    double Hsml, ...
    ...
};
```

```c
struct particle_soa_t {
    double *Mass, ...
    #if defined(BLACK_HOLES)...
    double *Hsml, ...
    ...
};
```

init source file:

```c
// Array of Structures:
struct particle *P;
// allocate one global array:
P = mymalloc(...)
```

```c
// Structure of Arrays
struct particle_soa_t P_SoA;
// allocate one global array
// for each field:
P_SoA.Mass = mymalloc(...)
#if defined(BLACK_HOLES)...
P_SoA.Hsml = mymalloc(...)
...```

favour auto-vectorization in *.c:

```c
... // may not vectorize
P[i].Mass + P[i]...
```

```c
... // vectorizes better
P_SoA.Mass[i] + P_SoA...
```

---


michele DOT martone AT lrz DOT de
Speedup suggests major code change steps

1. each type, $10 \approx 100$ fields, `#ifdefs`:

```c
1 struct particle {
2   double Mass, ...
3   #if defined(BLACK_HOLES)...
4   double Hsml, ...
5 ...
6};
```

```c
1 struct particle_soa_t {
2   double *Mass, ...
3   #if defined(BLACK_HOLES)...
4   double *Hsml, ...
5 ...
6};
```

2. (almost) each field an allocation:

```c
1 // Array of Structures:
2 struct particle *P;
3 // allocate one global array:
4 P = mymalloc(...
5
6 // Structure of Arrays
7 struct particle_soa_t P_SoA;
8 // allocate one global array
9 // for each field:
10 P_SoA.Mass = mymalloc(...
11 P_SoA.Hsml = mymalloc(...
12 ...
```

3. $\gg 10KLoC$ change in `*.c`!

```c
1 ...
2 // may not vectorize
3 P[i].Mass + P[i]...
```

```c
1 ...
2 // vectorizes better
3 P_SoA.Mass[i] + P_SoA...
```
10KLoC diff = ⚠️

- by hand? ❌❌❌
- custom script?
- and ...all-at-once?

Diagram:
- $r_x$ → $HPC\text{ expert's branch}$ → $r_z$ (merge)
- $r_y$ → $r_m$ (merge)
- $r_w$ → branch → domain scientist's branch
A C code matching and transformation engine

► A project from Inria (France)

---

2 https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches
3 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/coccinelle
A C code matching and transformation engine

- A project from Inria (France)
- originally to
  - update Linux kernel drivers
  - smash bugs (hence the name)

---

2 https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches
3 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/coccinelle
A C code matching and transformation engine

▶ A project from Inria (France)
▶ originally to
  🐞 update Linux kernel drivers
  ⚒ smash bugs
  (hence the name)
▶ seemingly underutilized in other contexts

---

2 https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches
3 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/coccinelle
Patch what and how?

```
allvars.cocci

match/patch specification rules

spatch

diff patch

allvars.diff

C source

allvars.h
```
Transformations *encoded in semantic patches* 🛠

- Produces usable patched (e.g. refactored) code
- No requirement to commit patched code
Step 1: modify data structures into SoA
Match main data structure

```c
@prtcl_str@

identifier id = {particle_data, sph_particle_data};

field list fs;

identifier I;

declaration d;

type ST;

@@

(@ (struct id { fs } *I;
    &
    ST { fs } *I;
    &
    d
    ) )
```

- pattern language to *match* C
- *metavariables* `id`, `I`, ... to match C language elements
Create new identifier 🔄

```python
@script:python new_prtcl_str_id@

id <<= prtcl_str.id;
id1;
@@
coccinelle.id1="%s_soa_t"%(id)
```

- reuse identifier `id` from rule `prtcl_str`
- new string using Python
Clone main data structure

@insert_new_prtcl_str depends on prtcl_str@

identifier new_prtcl_str_id.id.id1;
field list prtcl_str.fs;
type T;
@@

extern int maxThreads;
++struct id1 { fs };

- pattern language to patch C
- reminiscent of GNU patch
Filter out unwanted fields

```c
@match_anon_union_in_struct@
identifier id;
identifier J;
field list[n] fs;
identifier new_prtcl_str_id.id1;
@@

struct id1 { fs
  union { ... } J;
  ...
};

@rm_union_from_struct depends on match_anon_union_in_struct@
field list[match_anon_union_in_struct.n] fs;
identifier new_prtcl_str_id.id1;
field fld;
@@

struct id1 { fs
  - fld
  ...
};
```
Whitelisted types to pointers

```c
@make_ptr@
definition new_prtcl_str_id.id1;
definition M;
typedef MyDouble;
typedef MyFloat;
typedef MyLongDouble;
typedef MyDoublePos;
typedef MyBigFloat;
type MT = { double, float, MyDouble, MyFloat, MyLongDouble, MyDoublePos, MyBigFloat };

struct id1 { ... - MT M; + MT *M; ... }
```
Remove non-pointer fields ✗

```c
@del_non_ptr@
identifier new_prtcl_str_id.id1;
identifier J;
type T;
type P != {T*};
@

struct id1 { ...
 - P J;
   ...
};
```

► for each field J of type P which is not a pointer to another type...
Step 2: allocation functions
Populate allocation functions

```c
@per_type_soa_alloc@

identifier new_prtcl_str_id.id1;
identifier prtcl_str_mmbrs.M;
type prtcl_str_mmbrs.MT;
symbol P;
identifier insert_per_type_soa_functions.soa_alloc_fid;
fresh identifier si = "#ifdef "##id1##"##M##" //";
identifier str_from_id.str;

@@

void soa_alloc_fid(...) {
    ...
    ++si;
    ++ P->M = (MT*) mymalloc(str, sizeof(*(P->M)) * N);
    ++ if(!P->M) soa_abort(/*"allocating "*/ str);
    ++##endif
}
```
Step 3:
AoS → SoA
Match old structure defs

```c
@ostr@
identifier id = {particle_data, sph_particle_data};
type ST;
identifier P;
@@

( 
   struct id { ... } *P;
   &
   ST { ... } *P;
)
```

\[\text{\textsuperscript{4}}\text{Subset of rule prtcl\_str from p. 10.}\]
Match fields of new types⁵

@nt@

identifier pps.id1;
identifier I;
type T;
@@

struct id1 {
    ...
    T I;
    ...
};

⁵Relies on rule pps, duplicate of new_prtcl_str_id, p. 11.
Mold new identifiers

```python
@script:python pid@
id1 << pps.id1;
P << ostr.P;
S;
@@
coccinelle.S="%s_soa"%(P)
```

---

Same mechanism as new prtcl_str_id at p. 11.
Patch many expressions ($\gg 10$KLoC diff)

```plaintext
@soa_access@

identifier ostr.P;  
identifier pid.S;  
identifier nt.I;  
extpression E;

@@

- P[E].I  
+ S.I[E]
```

∀ identifier P previously matched in rule ostr

∀ identifier S previously matched in rule pid

∀ identifier I previously matched in rule nt

∀ expression E matching rule soa_access

substitute P[E].I with S.I[E]
GADGET can evolve further

- its semantic patches...
  - ...can be stored
  - ...and applied at anytime
  - ...ease performance experiments
  - ...serve also as a Coccinelle test

---

Thanks to Dr. Julia Lawall (Inria) for successful collaboration!

---

7 https://github.com/coccinelle/coccinelle/commit/ad5a94
One-day training: 24.03.2020 at LRZ

Introduction to Semantic Patching of C programs with Coccinelle

Register online

https://www.lrz.de/services/compute/courses/2020-03-24_hspc2w19/