Refactoring for Performance with Semantic Patching: Case Study with Recipes

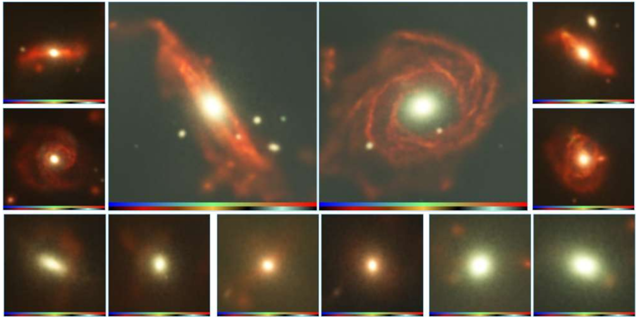
Michele MARTONE* and Julia LAWALL′

Leibniz Supercomputing Centre (Garching bei München, Germany)*
Inria, Paris, France′

C3PO'21 Workshop at ISC Frankfurt, Germany
July 2, 2021

The GADGET simulation code[1]

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



Galaxies simulated with Gadget – courtesy http://magneticum.org

[1] We work with a derivative of the one described by V. Springel, "The cosmological simulation code GADGET-2", MNRAS, vol. 364, pp. 1105–1134, 2005

Performance pilot study at LRZ

- L. Iapichino, V. Karakasis, F. Baruffa, N. Hammer[2]
- spanned over one year
- focused on 1kLoC extract
- identified changes meant for whole GADGET

[2] Presented at HPCS'17, 10.1109/HPCS.2017.64.

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...
```

Speedup *suggests* major code change steps

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

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

⇒

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

2. (almost) each field an allocation:

```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(...
...
```

3. ≫10KLoC change in *.c!

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

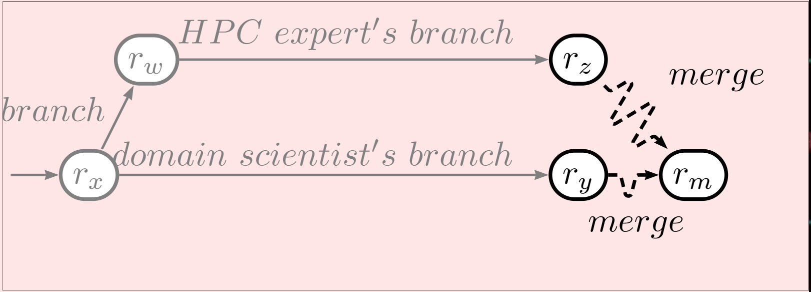
⇒

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

```
(float)fac_friction * (P_soa.Vel[n][2] - B
```

≫10KLoC *diff*=⚠

- ✍ ~~*by hand*~~ ? ⧗⧗⧗⧗
- 🐞 custom script?
- 💣 and ...all-at-once?



```
eastDouble dz = NGB_PERIODIC_LONG_Z(P_soa.Pos[p][2] -
```

Tool choice criteria

1. *timeliness*: How to quickly change so many lines of code?
2. *correctness*: How to avoid introducing mistakes? Note the aggravation of having numerous build-time *code paths* implied by the `#ifdefs`.
3. *flexibility*: Can we enact only a *partial* SoA translation, possibly on demand?
4. *continuity*: Can we develop in AoS, transforming only before *build and run*?
5. *acceptance*: How to have the community *accept* the proposed solution?

A C code matching and transformation engine ⚙

- A project from Inria (France)

[3] https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches

[4] 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[3]
  - **smash bugs** (hence the name)[4]

[3] https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches

[4] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/coccinelle

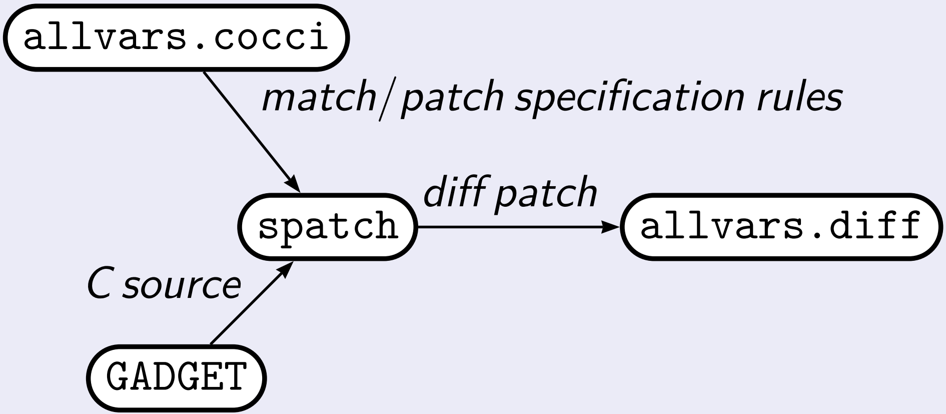
A C code matching and transformation engine

- A project from Inria (France)
- originally to
  - update Linux kernel drivers[3]
  - **smash bugs**
    (hence the name)[4]
- seemingly **underutilized** in other contexts

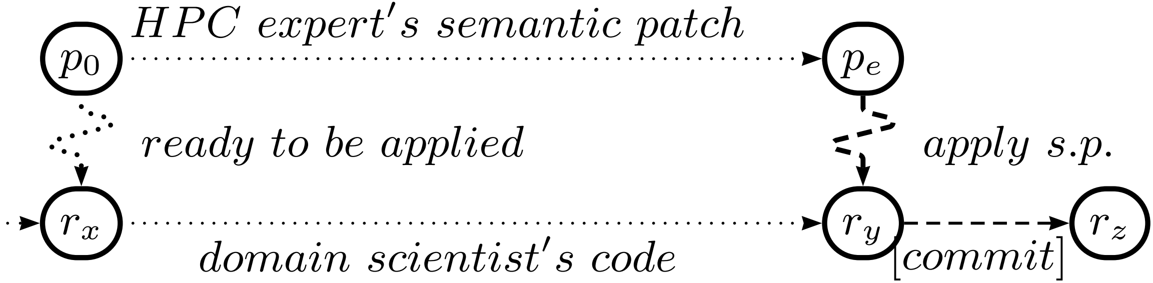
[3] https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches

[4] https://://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/coccinelle

Patch what and how ?



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

```
@prtcl_str@
identifier id = {particle_data,sph_particle_data};
field list fs;
identifier I;
@@

struct id { fs } *I;
```

- pattern language to *match* C
- *metavariables* `id`, `I`, ... to match C language elements

Create new identifier

```
@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.id1;
field list prtcl_str.fs;
@@

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

- pattern language to *patch* C
- reminiscent of `GNU patch`

Whitelisted types to pointers ♻

```
@make_ptr@
identifier new_prtcl_str_id.id1;
identifier 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 ✖

```
@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

```
@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 "##"HAVE__"##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[5]

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

struct id { ... } *P;
```

[5] Subset of rule `prtcl_str` from p. 12.

Match fields of new types[6]

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

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

[6] Relies on rule `pps`, duplicate of `new_prtcl_str_id`, p. 13.

Create new identifiers[7]

```
@script:python pid@
id1 << pps.id1;
P << ostr.P;
S;
@@

coccinelle.S="%s_soa"%(P)
```

[7] Same mechanism as `new_prtcl_str_id` at p. 13.

Patch many expressions (≫10KLoC diff)

```
@soa_access@
identifier ostr.P;
identifier pid.S;
identifier nt.I;
expression 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]`

Lessons learned

Easier semantic patching if ..

- Adopt *coding guidelines*
- Follow emerging practices in *research software engineering*

Outcome ✔

- 🚀 GADGET can evolve further
- its semantic patches...
  - ...can be stored 💾
  - ...and applied at anytime ▶
  - ...ease performance experiments ⚗
  - ...serve also as a Coccinelle test[8] 🔄

[8] https://github.com/coccinelle/coccinelle/commit/ad5a94