Tracepoint Factory

Automated LTTng Instrumentation & Babeltrace2 Plugin Generation from header files

Thomas Applencourt, Brice Videau | {apl,bvideau}@anl.gov

September 25, 2025

Argonne: Aurora

Our group at Argonne

- “Exascale” Machine
- 10k node, 6 Intel GPUs per nodes
- Developed a Tracing for “Heterogeneous APIs” (OpenCL, Cuda Runtime, Cunda Driver, MPI...) `git@github.com:argonne-lcf/THAPI.git`
- Everything in user space, rely heavily on EfficiOS tools (lttng and babeltrace)

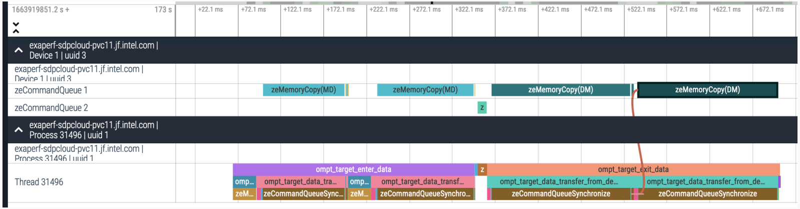
THAPI Example: Summary

```
BACKEND_MPI | 1 Hostnames | 12 Processes | 12 Threads |
       Name |     Time | Time(%) |     Calls  | Average |      Min |      Max |
zeSynronize |    2.61h |  74.75% |  217672947 | 43.10us |    141ns | 209.35ms |
[...]
MPI_Waitall | 21.86min |  61.57% |   87900923 | 14.92us |    115ns | 104.12ms |
MPI_Testall |  9.94min |  28.00% |   87420744 |  6.82us |    454ns | 811.00us |
[...]
      Total |    3.49h | 100.00% | 1515216024 |
```

THAPI Example: Trace (Hip on top of Level Zero)

```
13:36:02.387547645 - x4204c4s2b0n0 - vpid: 146726, vtid: 146726
- lttng_ust_hip:hipMemset_entry: { dst: 0xff00fffffc4f0000, value: 0, sizeBytes: 12392 }
13:36:02.387550815 - x4204c4s2b0n0 - vpid: 146726, vtid: 146726
- lttng_ust_ze:zeCommandListAppendMemoryFill_entry:
{ hCommandList: 0x0000000004f2da68, ptr: 0xff00fffffc4f0000, pattern: 0x00007fff829294df,
  pattern_size: 1, size: 12392, hSignalEvent: 0x000000001e672818, numWaitEvents: 2,
  phWaitEvents: 0x000000001e673d00,
  pattern_vals: "\x00", phWaitEvents_vals: [ 0x000000001e670658, 0x000000001ed15bd8  ] }
13:36:02.387558470 - x4204c4s2b0n0 - vpid: 146726, vtid: 146726
- lttng_ust_ze:zeCommandListAppendMemoryFill_exit: { zeResult: ZE_RESULT_SUCCESS }
- lttng_ust_hip:hipMemset_exit: { hipError_t: hipSuccess }
[...]
```

THAPI Example: Perfetto output



- We usually generate protobuf greater than 2GB... so excited about perfetto presentation tomorrow :) !
- And thanks for Perfetto people fixing our bugs are few years back when we opened on github

Challenge

- Small Team
- Currently 6 providers, total of 14737 Tracepoints and growing
- 10ich Trace Analysis Plugin

Only one solution: Automation[1]!

Take the official header of API and generate:

- The LTTng for tracepoint
- The Babeltrace plugins for analysis infrastructure

We rely heavily on EfficiOS software stack!

[1] Or I guess we should vibe-code everything

H2Yaml

How to generation so many tracepoint? From C Header to YAML

- `git@github.com:TApplencourt/h2yaml.git`
- Will generate a YAML file, then from there it's trivial to generate LTTng trace-points[2]

[2] Exercise left to the reader, or just talk to Olivier

Example of nice C found in header

```c
// Forward Declaration
typedef struct signal_s signal_t;
// Function Pointer with named arguments
typedef void (*SignalHandler)(int signum);
// Attribute in function call
void foo(const signal_t signum, SignalHandler handler);
// Declaration, anonymous enum (don't do that...)
struct signal_s {
  enum { S0 } signum;
};
```

We will generate

```
functions:
- name: foo
  params:
  - name: signum
    type:
      const: true
      kind: custom_type
      name: signal_t
  - name: handler
    type:
      kind: custom_type
      name: SignalHandler
  type:
    kind: void
```

```
enums:
- members:
  - name: S0
    val: 0

structs:
- members:
  - name: signum
    type:
      kind: enum
  name: signal_s
```

```
typedefs:
- name: signal_t
  type:
    kind: struct
    name: signal_s
- name: SignalHandler
  type:
    kind: pointer
    type:
      kind: function
      params:
      - name: signum
        type:
          kind: int
      type:
        kind: void
```

Technology

- Using clang, python binding (first proof of concept by EfficiOS Oliver, thanks!)
- Need to deal with lots of idiosyncrasy (anonymous built-in doesn't really work, getting function pointer argument name is atrociously tedious, forward-declared structs point to final node, typedef of struct are parsed twice, ...)
- But obviously clang can parse everything! So at least their is that...
- h2yaml 100% unit tested, generated same output as our legacy ruby parser. Can parse all the header we give them now

Example of libclang Madness

```
def is_anonymous2(self):
  match self.kind:
    case clang.cindex.CursorKind.PARM_DECL:
# `is_anonymous()` returns True for `double a` in `void (*a5)(double a, int);`.
# We no longer use `not spelling` trick to due to a libclang quirk:
# In `(*a6)(a6_t)`, the spelling of `a6_t` will be `a6_t` instead of None.
      return not self.get_usr()
    case clang.cindex.CursorKind.FIELD_DECL:
# - Unnamed structs have "anonymous ..." in `spelling`
# - Named structs within unions: `is_anonymous()` returns True.
# - Unnamed bitfields: `is_anonymous()` returns False, but `spelling` is empty.
       return not self.spelling or "(anonymous at" in self.spelling
    case clang.cindex.CursorKind.ENUM_DECL:
# In `struct S2 { enum { H0 } a; }` where `is_anonymous()` returns False
# Fortunately, Clang uses `@EA@` and `@Ea@` in the USR for anonymous enums.
# (Though I never saw `@Ea@`...)
       return self.is_anonymous() or is_in_usr(["@EA@", "@Ea@"])
[...]
    case _:
      return self.is_anonymous()
```

Sweet dream

- Will be nice if multiple tool can agree on the same format
- Will ease the maintenance burden, and avoid duplication of effort
- I know I'm dreaming… But maybe can serve as starting point for your own project.

Metababel

The lesser discussed topic: Analysis of tracez

- We need analyse/transform our lttng trace
- Generate Summary, Timeline...
- We use babeltrace2: we need babeltrace2 plugin
- Python are too slow[3] so using C plugin
- But writing Babeltrace C plugin is a little tedious

[3] Babeltrace2 is too slow too but this is another topic

Cons of Babeltrace2

- Babeltrace2 API is powerfull aka not usable for “common” user.
- Need to understand the CTF hierarchy (common context, environment, stream class) and how to unpack message, push message, handle multiple port, create new class, ...
- One of our typical plugin is 152066 lines of (generated) babeltrace2 C API

Introducing: Metababel!

- Take a YAML (more or less the one generated by h2yaml), generate all the plugin infrastructure.
- Then one can register callback to particular "event/message" and push new message downstream, without knowing anything about babeltrace2 internal.

Example Metababel YAML format

```
:stream_classes:
- :name: ze
  :event_common_context_field_class:
    :type: structure
    :members:
    - :name: vpid
      :field_class:
        :type: integer_signed
        :field_value_range: 32
        :cast_type: int
    - :name: vtid
      :field_class:
        :type: integer_signed
        :field_value_range: 32
        :cast_type: int
```

```
:event_classes:
- :name: GetPlatformIDs
  :payload_field_class:
    :type: structure
    :members:
      - :name: num_entries
        :field_class:
          :type: integer_unsigned
      - :name: platforms
        :field_class:
          :type: integer_unsigned
      - :name: num_platform
        :field_class:
          :type: integer_unsigned
```

Example: Callback

The only code user need to wrote

```c

#include <metababel/metababel.h>
#include <stdio.h>

static void btx_getplatformids_callbacks(
    void *btx_handle,
    void *usr_data,
    const char* name,
    int vpid, int vtid, uint64_t num_entries, uint64_t platforms, uint64_t num_platform) {

   printf("Received btx_getplatformids_callbacks message\n");
}

void btx_register_usr_callbacks(void *btx_handle) {
btx_register_callbacks_GetPlatformIDs(btx_handle,&btx_getplatformids_callbacks);
}
```

Example: Compilation

```
ruby -I../lib ../bin/metababel --component-type SINK \
                               --upstream fake_api.yaml \
                               -o btx_sink
# Wrote your callbacks
...
# then link everything together
gcc -g -o btx_sink.so btx_sink/*.c btx_sink/metababel/*.c
       -I btx_sink/ -I ../include/ ${CFLAGS}
babeltrace2 --plugin-path=.
            --component=source.metababel_source.btx
            --component=sink.metababel_sink.btx
```

More metabel info

- This is how we handle all our plugins.
- We have the luxury to know statically all our messages types.
- If it's also your case, please give it a shot[4]

Future work

- Improve performance[5]

[4] I'm pretty proud of this project. It served us well!

[5] No more error checking?

Conclusion

Conclusion

- Small Team, need a lot of automation
- h2YAML to generate a yaml from header (so can be used to generate tracepoint and babeltrace plugin)
- Metababel, abstract away babeltrace for ease and speed of trace analysis