Dynamic Tracing in Userspace

Dyninst, Kaji and the way ahead..

Suchakrapani Datt Sharma

Dec 11, 2013

École Polytechnique de Montréal

Laboratoire DORSAL

Agenda

Recap

- Questions raised

Investigations

- How Dyninst + UST performs
- A separate dynamic tracing lib – Kaji
- Analysis of Dyninst and Kaji

What Next

- Further investigations
- New features

Recap

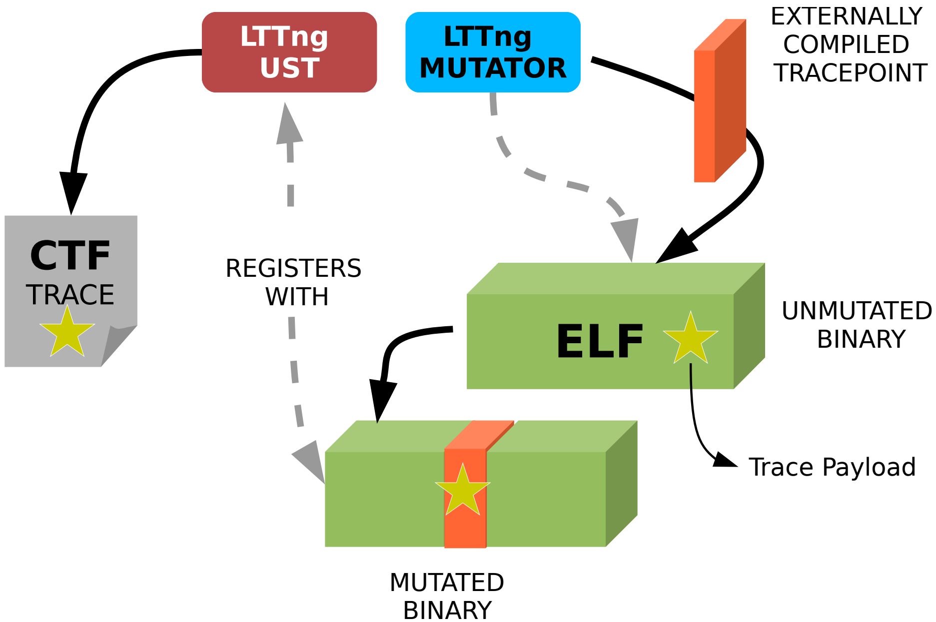
The goal was to investigate tools which can be of use to provide dynamic tracing with UST without compromising performance

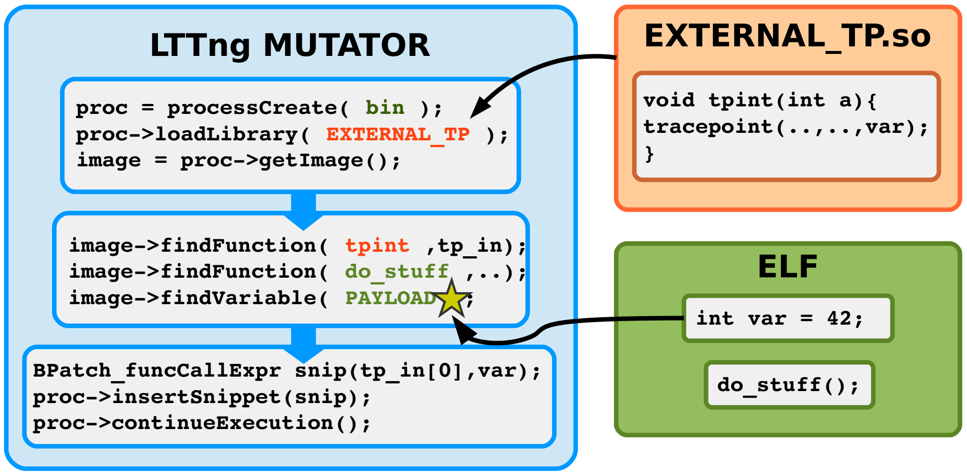
Questions raised :

- How well would Dyninst perform?
- What does it actually do?
- Is GDB's infrastructure better than Dyninst?
- Are there new ways to leverage the current tools?

Investigations

Dyninst + UST

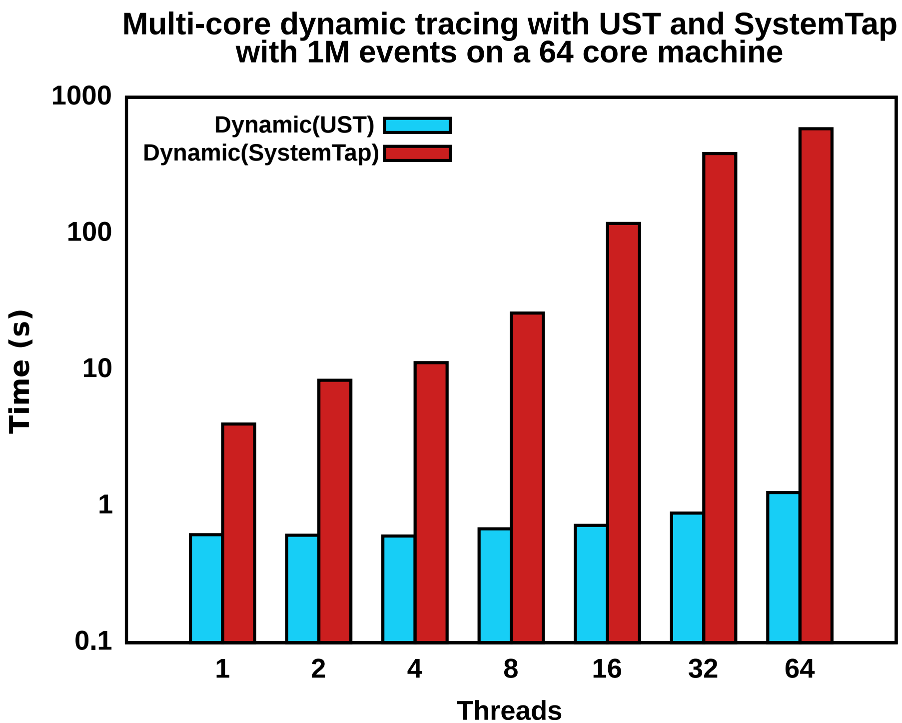


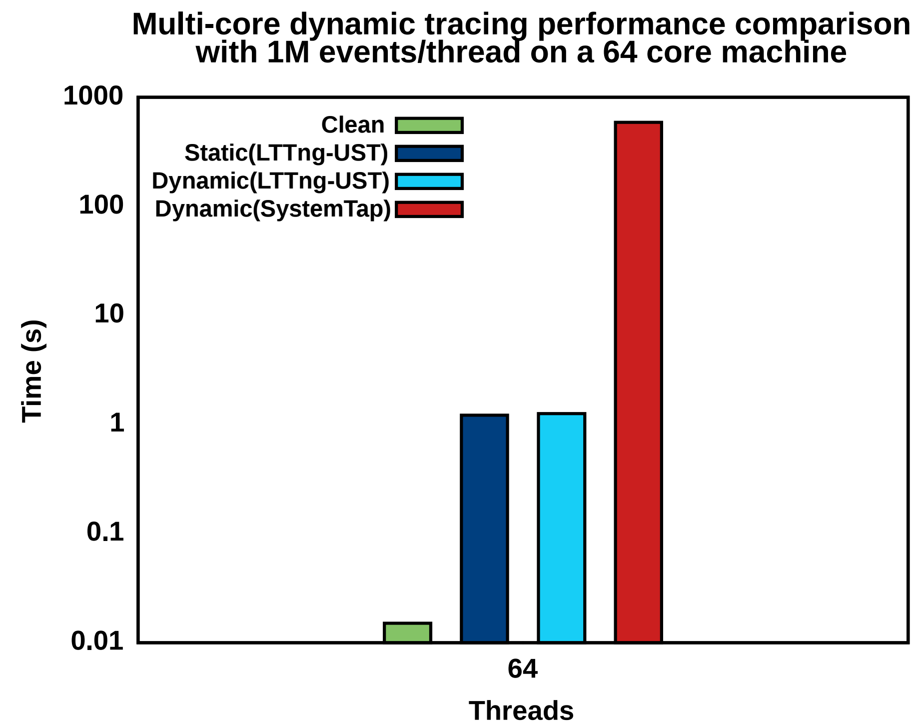


Target binary is first started and then mutated in process

Investigations

Dyninst + UST





Dyninst+UST provides similar overhead as compared to static tracing. Good scalabilitywhen tuned with right options (disable recursive trampoline check and disable FPR save)

Investigations

Kaji

- A new lightweight library for dynamic tracing in development
- We used GDB's jump-pad based approach – very minimal
- At a very nascent stage – more like a proof of concept for now

Zifei's repo : https://github.com/5kg/kaji

My repo : https://github.com/tuxology/kaji

Investigations

Dyninst and Kaji Analysis

```
#### Original ###
4009e8 <+0>:      push   %rbp
4009e9 <+1>:      mov    %rsp,%rbp
4009ec <+4>:      movl   $0x2a,-0x4(%rbp)
4009f3 <+11>:     pop    %rbp
4009f4 <+12>:     retq
```

Target function was dynamically instrumented with a tracepoint call and observed

```
#### Dyninst's Modification ###
4009e8 <+0>:      jmpq   0x10000
4009ed            rex.RB cld
4009ef <+7>:      sub    (%rax),%al
4009f1 <+9>:      add    %al,(%rax)
4009f3 <+11>:     pop    %rbp
4009f4 <+12>:     retq
```

Whole block replaced

```
#### Kaji's Modification ###
                  push   %rbp
                  mov    %rsp,%rbp
400aa4 <+4>:      jmpq   0x100000
400aa9 <+9>:      add    %al,(%rax)
400aab <+11>:     pop    %rbp
400aac <+12>:     retq
```

Jump at instruction > 5byte

Investigations

Dyninst' s Jump

Out of line execution

```
0x10000:    push   %rbp
0x10001:    mov    %rsp,%rbp
0x10004:    movl   $0x2a,-0x4(%rbp)
0x1000b:    pop    %rbp
```

Trampoline start
Grow stack

```
0x1000c:    lea    -0xa8(%rsp),%rsp
            mov    %rax,0x20(%rsp)
0x10019:    lea    0xa8(%rsp),%rax
```

Do some tricks

```
            and    $0xffffffffffffffe0,%rsp
0x10025:    mov    %rax,(%rsp)
0x10029:    mov    -0x88(%rax),%rax
```

Push regs

```
0x10030:    push   %rax
0x10031:    push   %rbx
0x10032:    push   %rcx
0x10033:    push   %rdx
0x10034:    push   %rsp
0x10035:    push   %r12
0x10037:    push   %r13
0x10039:    push   %r14
0x1003b:    push   %r15
```

Grow stack

```
0x1003d:    lea    -0x18(%rsp),%rsp
0x10042:    movabs $0x601064,%rax
0x1004c:    mov    (%rax),%edi
0x1004e:    movabs $0x0,%rax
            movabs $0x7f448928fa06,%rbx
0x10062:    callq  *%rbx
0x10064:    lea    0x18(%rsp),%rsp
```

var = 43

tpint() from
tracepoint lib

Shrink stack

Pop regs

```
0x10069:    pop    %r15
            pop    %r14
0x1006d:    pop    %r13
0x1006f:    pop    %r12
0x10071:    pop    %rsp
0x10072:    pop    %rdx
0x10073:    pop    %rcx
0x10074:    pop    %rbx
0x10075:    pop    %rax
0x10076:    mov    (%rsp),%rsp
0x1007a:    retq
```

Restore
orignial rsp

Investigations

Kaji' s Jump

Push regs

```
0x100000:    push   %rax
0x100001:    push   %r8
0x100003:    push   %r9
0x100005:    push   %rcx
0x100006:    push   %rdx
0x100007:    push   %rsi
0x100008:    push   %rsp
0x100009:    push   %r12
0x10000b:    push   %r13
0x10000d:    push   %r14
0x10000f:    push   %r15
0x100011:    movabs $0x7f65432cb472,%rax
0x10001b:    callq  *%rax
0x10001d:    pop    %r15
0x10001f:    pop    %r14
0x100021:    pop    %r13
0x100023:    pop    %r12
0x100025:    pop    %rsp
0x100026:    pop    %rsi
```

kaji_int_probe()

Pop regs

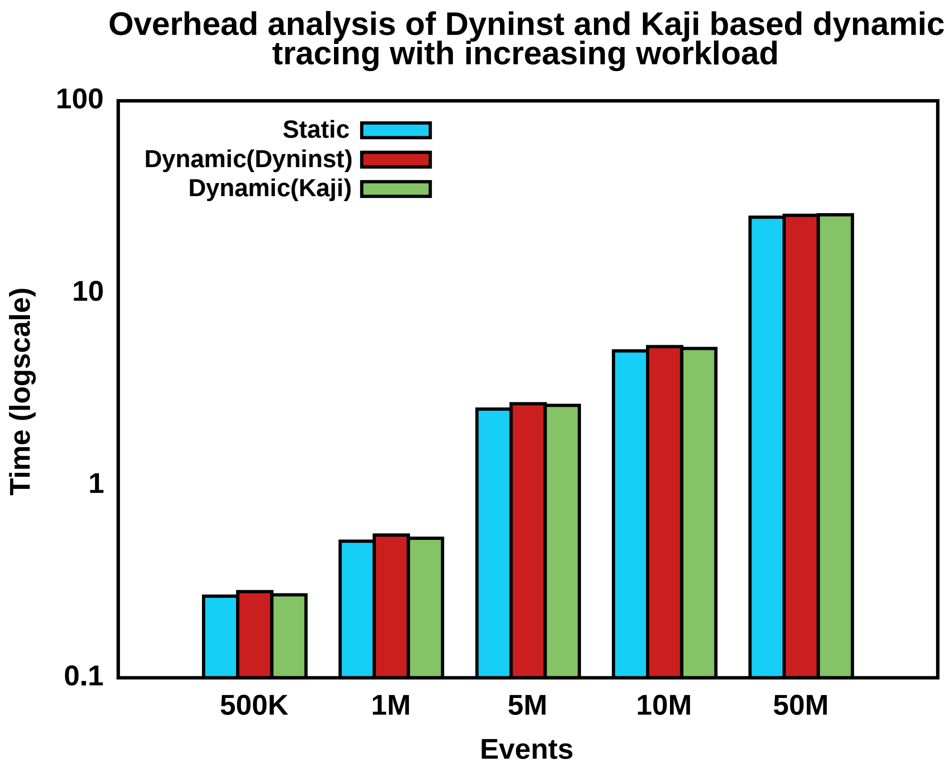
```
0x100027:    pop    %rdx
0x100028:    pop    %rcx
0x100029:    pop    %r9
0x10002b:    pop    %r8
0x10002d:    pop    %rax
0x10002e:    movl   $0x2a,-0x4(%rbp)
0x100035:    jmpq   0x400aab <do_stuff+11>
```

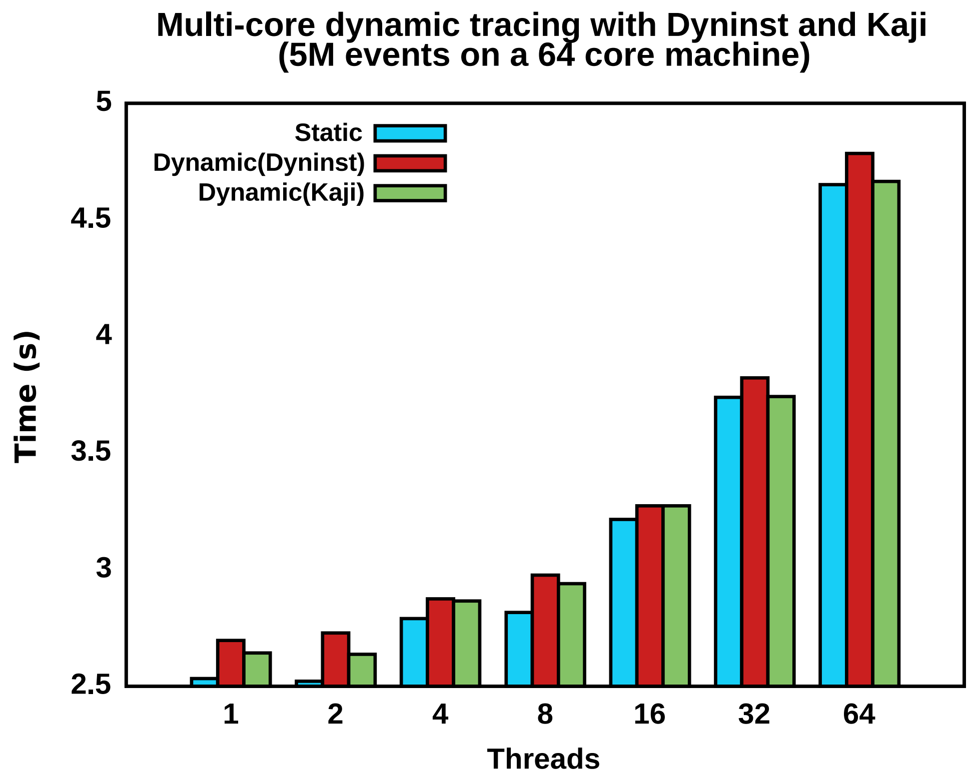
Execute displaced instructions

Go back from Jump-pad

Investigations

Kaji/Dyninst + UST (Overhead & Scalability)



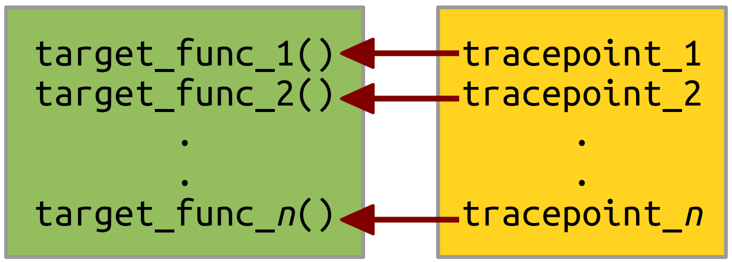


As expected, the similarity in both approaches translates to similar performance.

But hold on...

Investigations

Kaji/Dyninst + UST (Startup)



Test Binary Tracepoint Lib

Measure $\mathbf{T}_{reg}$ + $\mathbf{T}_{instr}$
with *n* varying from 1 to 5000

(for Kaji, n is restricted to 1 as its not mature enough to handle multiple tracepoints for now)

Dyninst

n	T_{instr} (s)	T_{reg} (s)
1	2.63	0.03
10	2.65	0.03
100	2.99	0.04
1000	6.68	0.05
5000	**35.03**	0.11

Kaji

n	T_{instr} (s)	T_{reg} (s)
1	0.002	0.012

Even for *n*=1, instrumentation cost for Kaji is way less (**0.002s** compared to **2.63s** for Dyninst) as we can have fine grained control of instrumentation time unlike Dyninst.

What Next?

More analysis!

One does not simply... stop analyzing stuff!

- Real-life benchmarks
  - PostgreSQL, MariaDB, Kenrel build – Mimic multiple static tracepoints – but build and instrument them dynamically
- Isolate startup time for multiple scenarios with a real life benchmarks

Possible features

- On-the-fly dynamic tracepoints
  - Generate dynamic tracepoints based on user inputs – scripts, switches
  - *Zifei's* early implementation (expand the macro strategy) - http://ur1.ca/g5w27
- Fixed type dynamic tracepoints
  - Common tracepoints based on types – regs, ints, floats, strings
  - Easy access, no need to generate separate tracepoints

What Next?

Further investigation

- Use of bytecode interpreters and JIT in tracing infrastructure
  - Can be useful for various features – LTTng already has bytecode interpretation for implementing filters
  - Ktap uses bytecode based dynamic tracing for kernel
- Can this lead to a purely userspace based bytecode tracing design?
- Seccomp-bpf – syscall filtering using BPF for sandboxing.
  - Chrome is already using that.
  - A step further – libseccomp has provision to output BPF code
- JIT for BPF improves the performance further. Should we aim for a similar approach?

Questions?

suchakrapani.sharma@polymtl.ca

suchakra *on #lttng and #fedora-india*