Dynamic Tracing in Userspace

An introductory analysis of possible approaches with focus on DyninstAPI and GDB

Suchakrapani Datt Sharma

May 2, 2013

École Polytechnique de Montréal
Laboratoire DORSAL

Agenda

Introduction

Aim

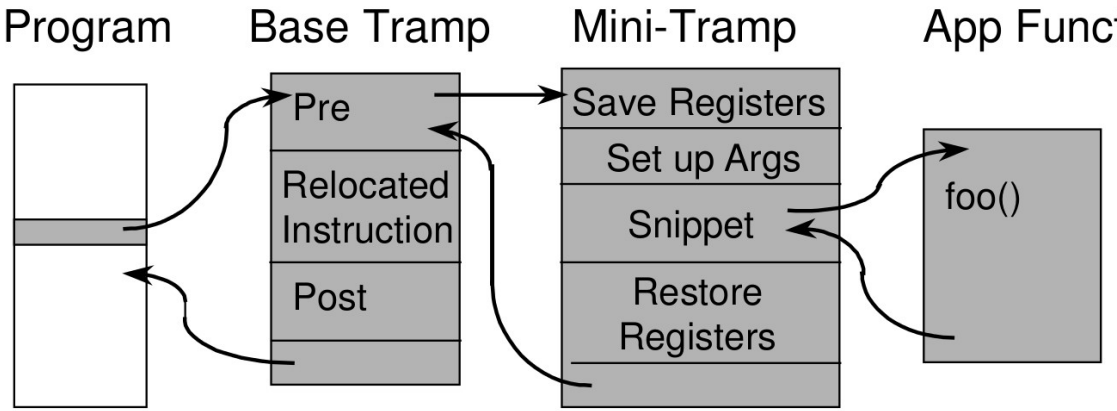
The goal is to investigate tools which can be of use to provide dynamic tracing with UST without compromising performance.

Some tools investigated recently

- Dyninst API for dynamic instrumentation
- GDB's fast tracing infrastructure

Dyninst

A *mutator* program mutates the target program (*mutatee*) by building and inserting code (*snippets*) during runtime

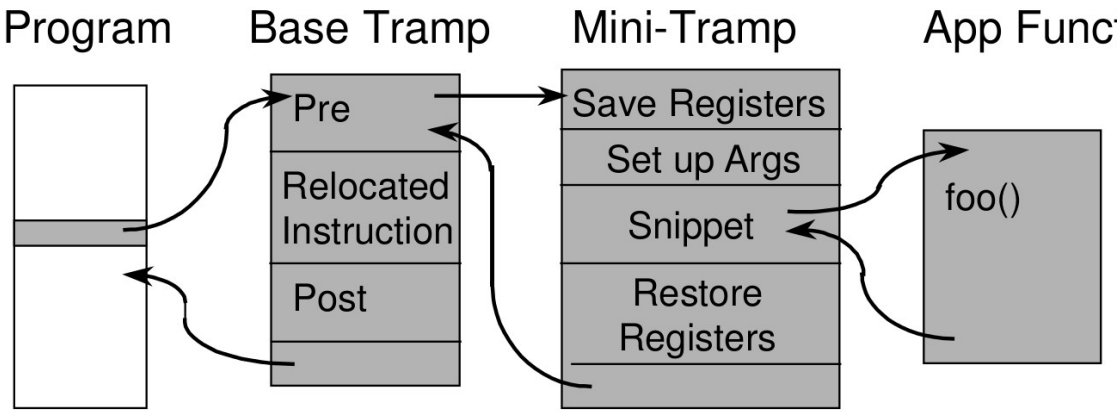


How Dyninst works

- BPatch object : `processCreate()`, `processAttach()`..
- Build snippets : `BPatch_arithExpr`, `BPatch_varExp`..
- Find points : `appImage->findFunction()`, `findPoint()`
- Insert Snippet : `appAddrSpace->insertSnippet()`

Dyninst

A *mutator* program mutates the target program (*mutatee*) by building and inserting code (*snippets*) during runtime

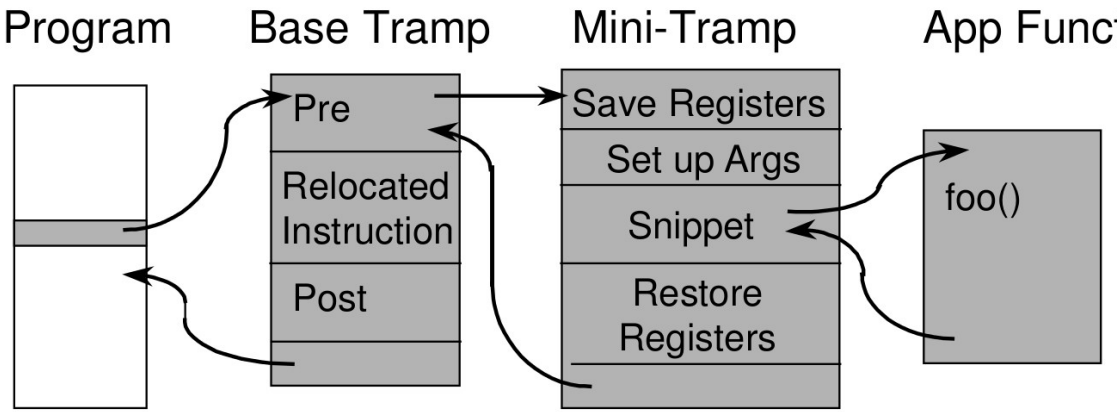


How Dyninst works

- BPatch object : `processCreate()`, `processAttach()`..
- Build snippets : `BPatch_arithExpr`, `BPatch_varExp`..
- Find points : `appImage->findFunction()`, `findPoint()`
- Insert Snippet : `appAddrSpace->insertSnippet()`

Dyninst

A *mutator* program mutates the target program (*mutatee*) by building and inserting code (*snippets*) during runtime

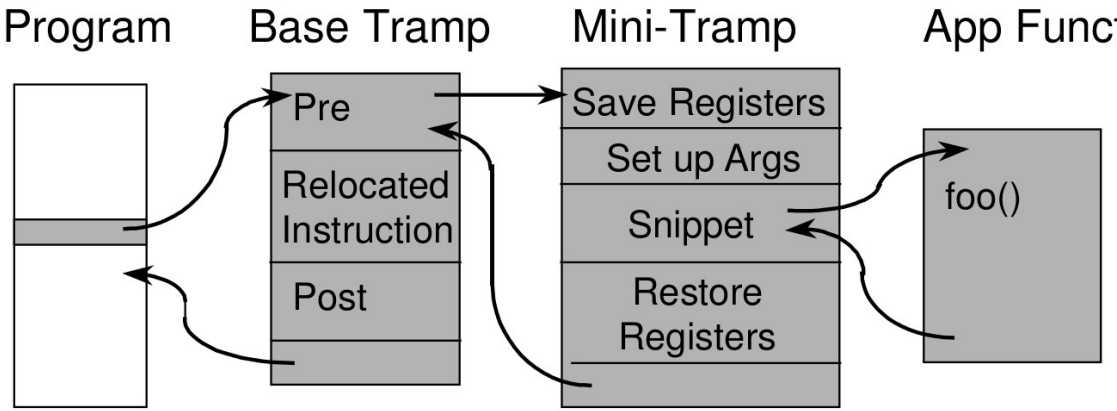


How Dyninst works

- BPatch object : `processCreate()`, `processAttach()`..
- Build snippets : `BPatch_arithExpr`, `BPatch_varExp`..
- Find points : `appImage->findFunction()`, `findPoint()`
- Insert Snippet : `appAddrSpace->insertSnippet()`

Dyninst

A *mutator* program mutates the target program (*mutatee*) by building and inserting code (*snippets*) during runtime

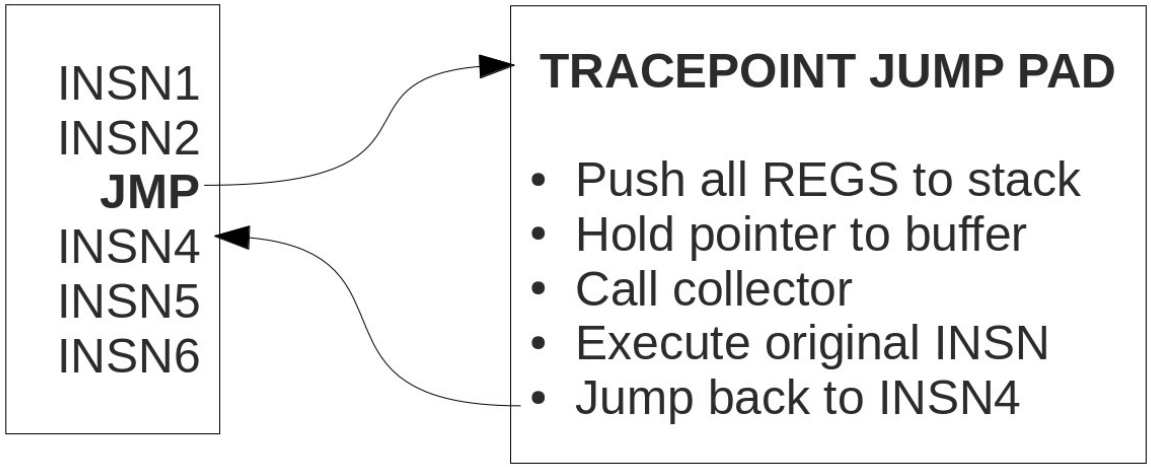


How Dyninst works

- BPatch object : `processCreate()`, `processAttach()`..
- Build snippets : `BPatch_arithExpr`, `BPatch_varExp`..
- Find points : `appImage->findFunction()`, `findPoint()`
- Insert Snippet : `appAddrSpace->insertSnippet()`

GDB's Tracing Infra

Fast Tracepoints : Similar approach by jumping to a small trampoline (*jump pad*). Faster than *normal* trap based tracepoints



How GDB's fast tracepoints work

- Can be placed only at instructions 5 bytes or longer (4 bytes on some targets)
- Needs In-process agent (IPA) library loaded in inferior process

Tests & Results

GDB and Dyninst elementary overhead analysis

Intel Core i5-3317U (1700Mhz x 4), 4GB RAM, Fedora 18 (3.8.1-201.fc18.x86_64)

Dyninst Overhead Analysis

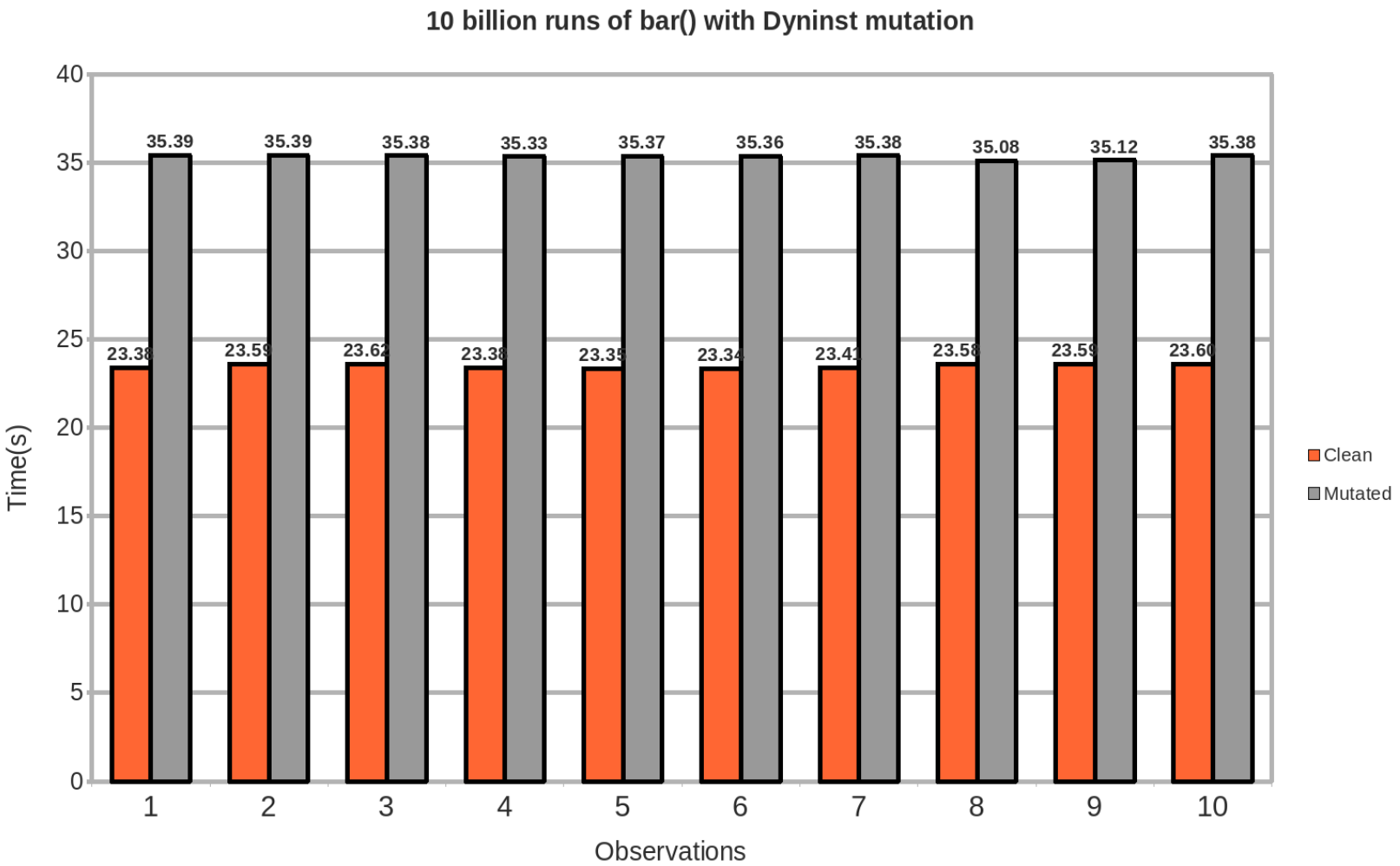
Test Scenario : Consists of a *dummy* program and a *mutator* program mutating it. It instruments the function `bar()` in dummy. The function `bar()` originally just increments a variable by 1.

```
void bar(){ ch=ch+1; }
```

The mutator builds a snippet to initialize another variable with zero and inserts this snippet during *dummy* runtime. Time for 10 billion runs of `bar()` is observed using `clock_gettime()` for a clean vs mutated run

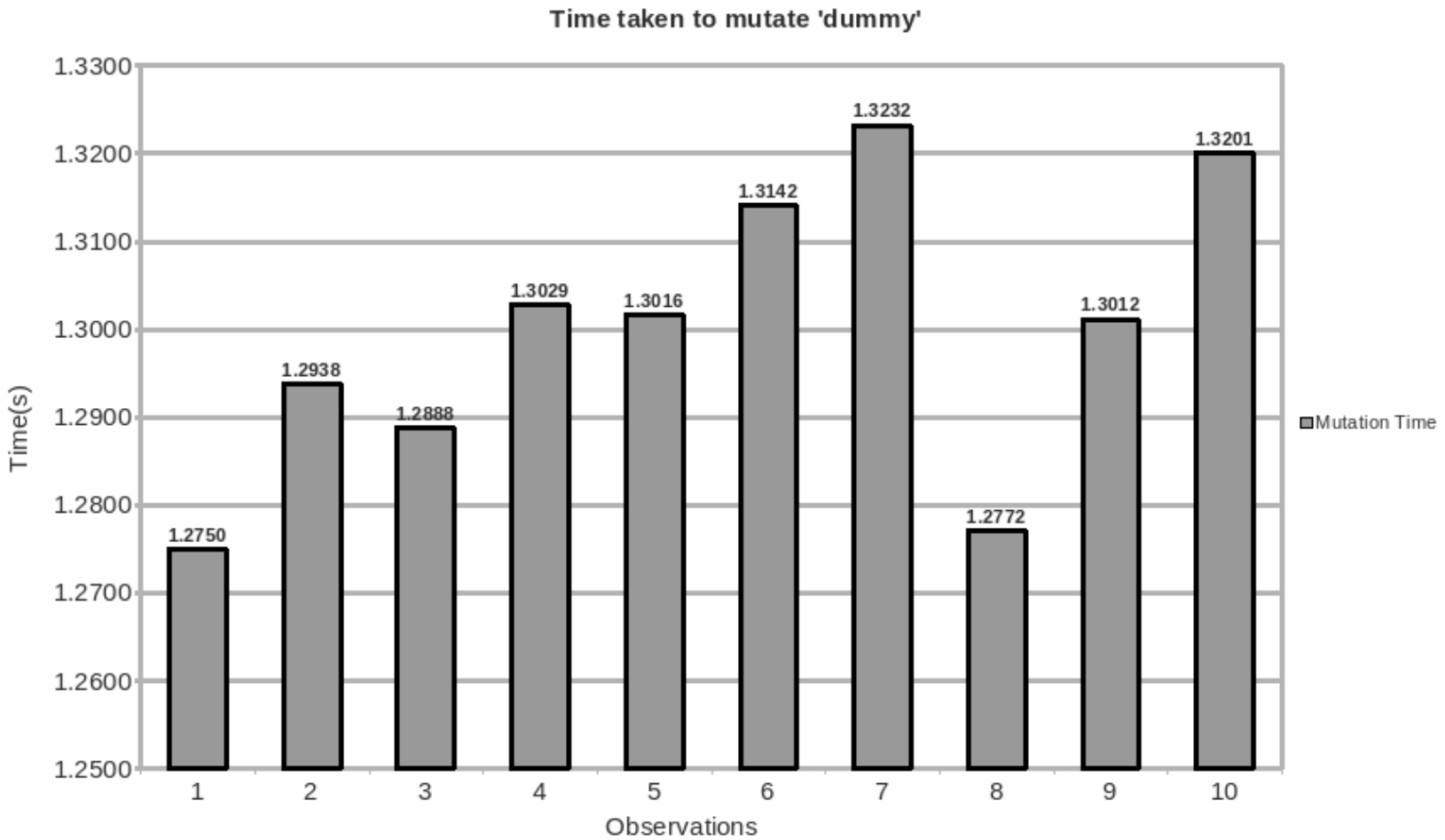
```
a = appAddrSpace->malloc(*appImage->findType("int"));
BPatch_arithExpr initSnippet(BPatch_assign, *a, zero);
appAddrSpace->insertSnippet(initSnippet, *functionEntry);
```

Results



- Mean difference of 11.83s and an overhead of 50.38%

Results



- Mean mutation time : 1.29s

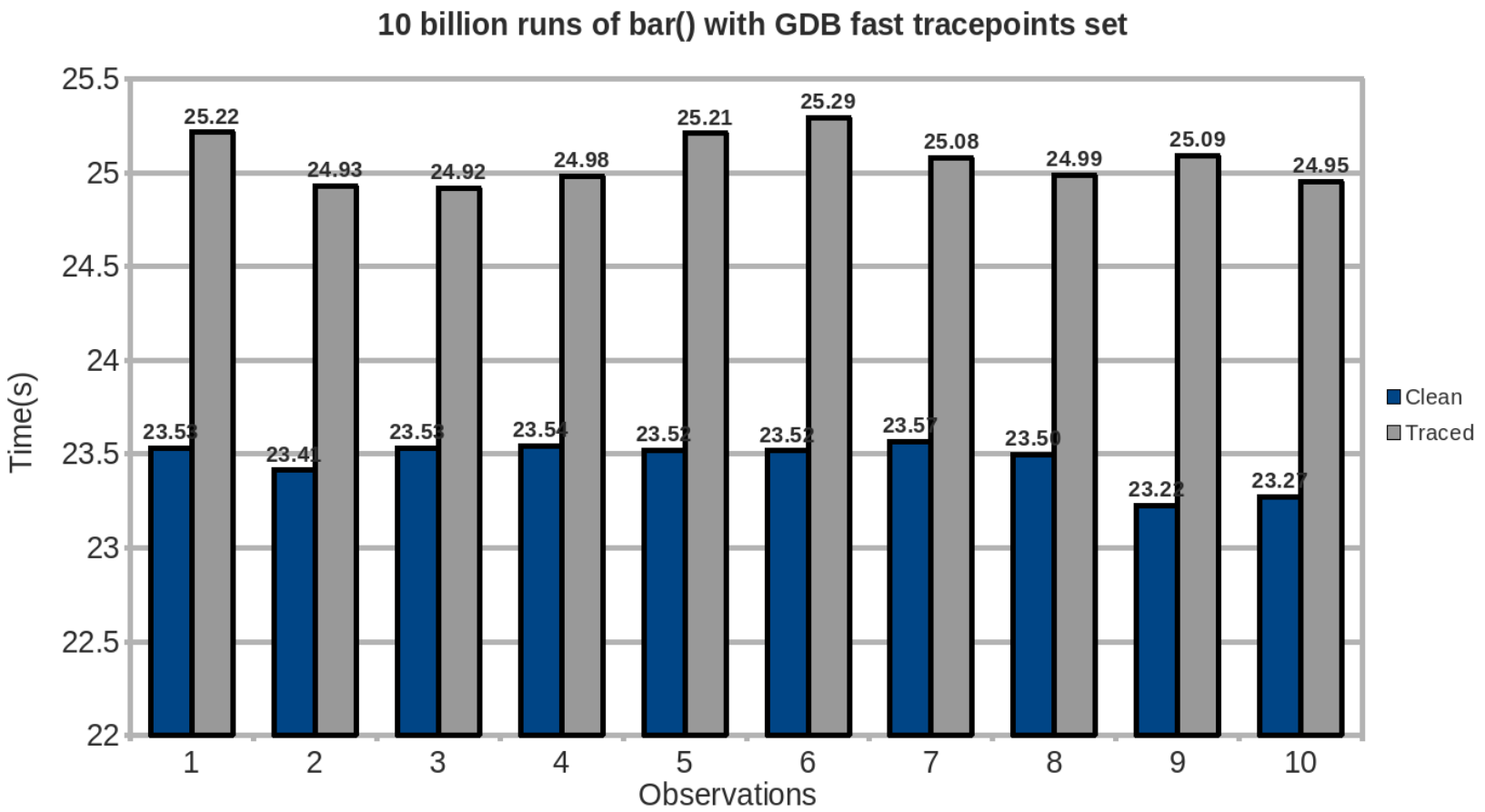
GDB Overhead Analysis

Test Scenario : Consists of a *dummy* program with a function `bar()`. The function `bar()` originally just increments a variable by 1. The *dummy* application is started with GDBserver while pre-loading the IPA library.

```
gdbserver --wrapper env LD_PRELOAD=/usr/lib64/libinproctrace.so
:2222 dummy
```

Through GDB, fast tracepoint is set at beginning of `bar()` and trace experiment is conducted without any 'actions'. Time for 10 billion runs of `bar()` is observed using `clock_gettime()` for a clean vs traced run

Results



- Mean difference of 1.60s and an overhead of 6.83%

What Next?

- Run more rigorous and real life tests on multiple test setups
- Analyze pros and cons for each dynamic instrumentation candidate
- Investigate ways to utilize GDB's dynamic tracing infra for UST
- A glance at other tools - DynamoRIO, Pin

Long Term Vision

The long term goal now is eventually merging the functionality of tracing, profiling and debugging in an integrated tool.

Questions and Suggestions?

suchakrapani.sharma@polymtl.ca

suchakra on #lttng