COST AWARE TRACING

Amir Haghshenas
Michel Dagenais
Naser Ezzati-Jivan
Summer 2022

CONTENT TABLE

- Previously on cost aware tracing
- Definition of cost
- How to calculate the cost
- How to control the cost
- Future work

WHAT IS COST AWARE TRACING

- Tracing can generate large amount of data in short period of time
- Cost aware tracing is to adjust tracing based on a defined budget
- It can be most helpful for devices with limited resources (IoT devices)
- Only a limited amount of overhead on the application is acceptable (budget)

WHAT IS COST

Cost function	Description	Purpose
Time	Program Execution delay added by tracing	Real time applications have very limited time budget
Memory	Capacity to store all the generated data	Limited memory applications cannot store all the data
Detection delay	Time between execution and recording the event	Some events should be captured as fast as possible
Concurrent Execution	Effect of tracing on concurrent behavior of multi-threaded applications	Program execution is not the same on different threads

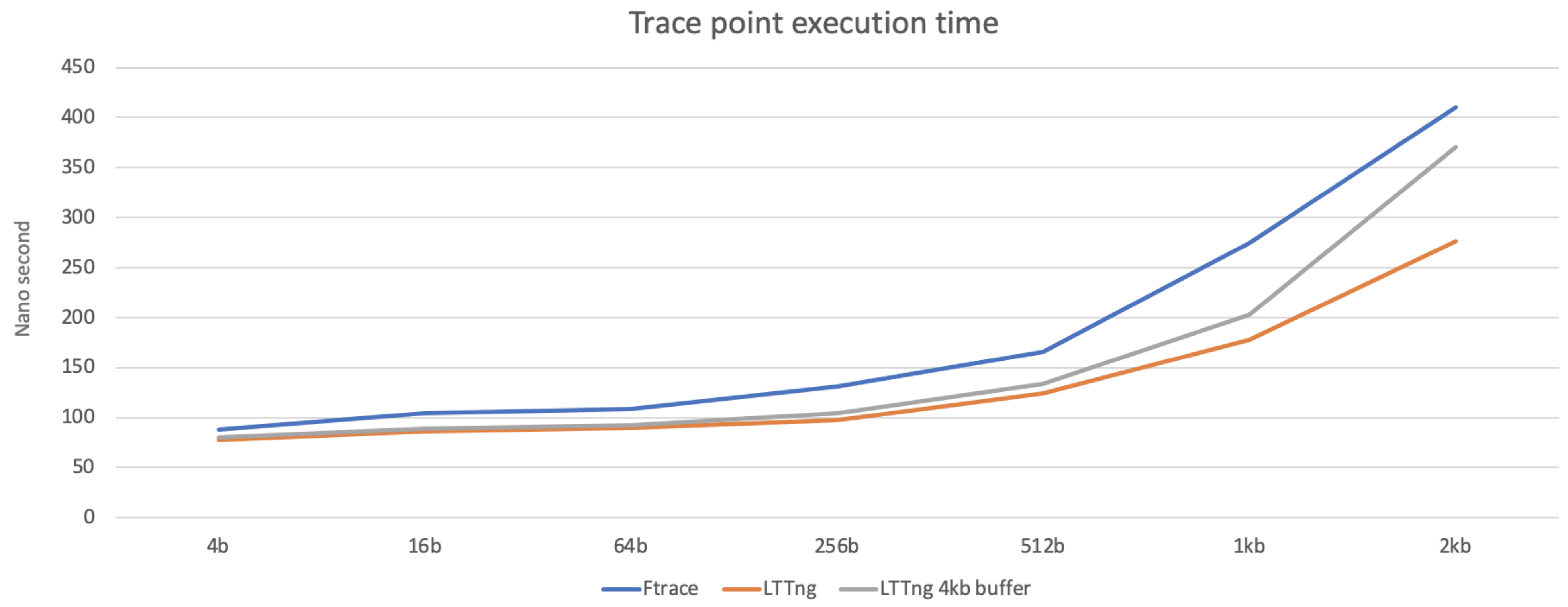
CALCULATION OF COST

FIRST STOP, FTRACE

- A kernel benchmark was developed to analyze the overhead of a trace point with different payloads.
- Step 1: developed a series of trace points in Ftrace with various payloads (from 4b to 2kb)
- Step 2: developed a simple kernel module to call the trace points in a loop and calculated the overhead caused by each trace point when tracing by Ftrace.

NEXT STOP, LTTNG

- Adding LTTng layer to Ftrace trace points.
- Step 1: added trace point definitions for existing Ftrace trace points in LTTng.
- Step 2: provided the required probe in LTTng modules for the trace points.
- Step 3: tested the effect of different payloads under various conditions.


Trace point execution time
Nano second
450
400
350
300
250
200
150
100
50
0
4b
16b
64b
256b
512b
1kb
2kb
Ftrace
LTTng
LTTng 4kb buffer

RESULTS

COST CALCULATION IN USER SPACE

- A benchmark was developed to calculate trace points with similar payloads as the kernel for user space in C.
- Function tracing a simple C application and calculating the overhead of each trace point at the start and end of each function.
- Results require a bit of adjustment to be presented.

COST CONTROL

- There are two entities required for solving such a problem
  - Cost function vs objective function
- Cost functions related to time:
  - Contribution of each trace point (each function trace point) to the execution time
- Objective function related to time:
  - NO specific objective (very first step)
  - Variation in the number of times each trace point is called
  - Detecting a trend in the number of times they are called.
  - Detecting trend compared to normal behavior
  - Possibility of considering user intention

PROTOTYPING THE SOLUTION

A simple prototype is developed to analyze the overhead caused by tracing

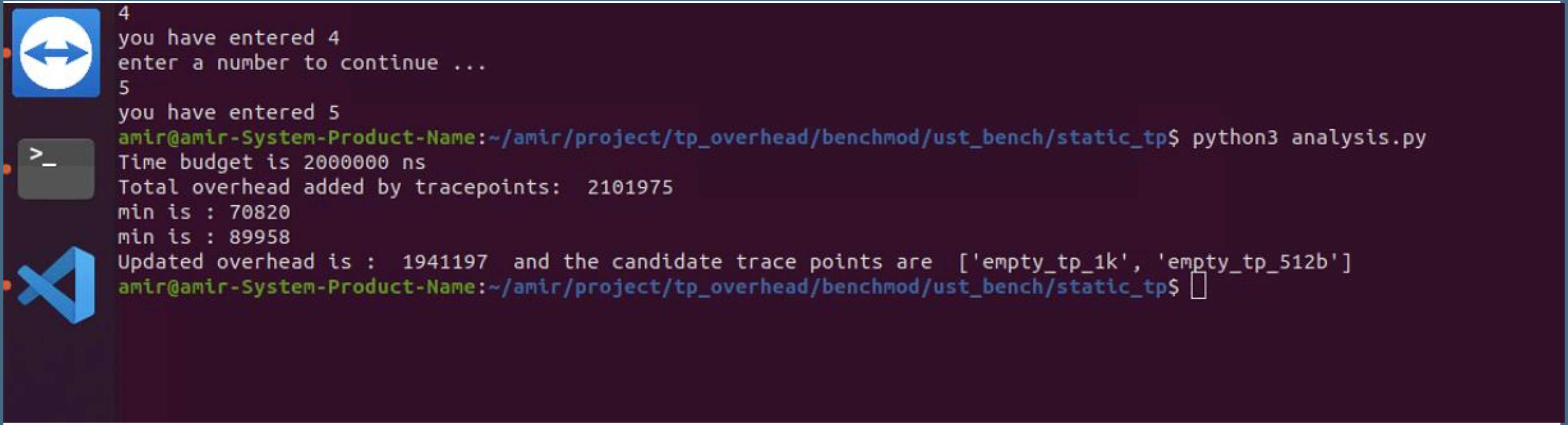
The required data is gathered from the user space benchmark

Information about the trace point are collected

An analyzer is developed to solve the optimization problem

Goal is to just automatically suggest candidates to be disable for the next round of tracing to satisfy the time budget

Select the minimum cost until the time budget is satisfied.



```
4
you have entered 4
enter a number to continue ...
5
you have entered 5
amir@amir-System-Product-Name:~/amir/project/tp_overhead/benchmod/ust_bench/static_tp$ python3 analysis.py
Time budget is 2000000 ns
Total overhead added by tracepoints:  2101975
min is : 70820
min is : 89958
Updated overhead is :  1941197  and the candidate trace points are  ['empty_tp_1k', 'empty_tp_512b']
amir@amir-System-Product-Name:~/amir/project/tp_overhead/benchmod/ust_bench/static_tp$ 
```

RESULT

NEXT STEP

Using LTTng rotation, modify tracing for the next rotation

Adding objective functions such as variation in the number of calls or detecting a trend in the number of calls

More complex algorithms to learn the normal behavior of the system and decide accordingly

Including user intention and adjust tracing based on them.

REFERENCES

- Gebai, M., & Dagenais, M. R. (2018). Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead. *ACM Computing Surveys (CSUR)*, *51*(2), 1-33.
- Orton, I., & Mycroft, A. (2021, September). Tracing and its observer effect on concurrency. In *Proceedings of the 18th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes* (pp. 88-96).
- S. Fischmeister and P. Lam. Time-AwareInstrumentation of Embedded Software.IEEETransactions on Industrial Informatics, 6(4):652–663,Nov 2010.
- H. Kashif, P. Arafa, and S. Fischmeister. INSTEP: AStatic Instrumentation Framework for PreservingExtra-Functional Properties. InIEEE 19thInternational Conference on Embedded and Real-TimeComputing Systems and Applications, RTCSA'13,pages 257–266, Aug 2013.
- P. Arafa, H. Kashif, and S. Fischmeister. DIME:Time-aware Dynamic Binary Instrumentation UsingRate-based Resource Allocation. InProceedings of theEleventh ACM International Conference on EmbeddedSoftware, EMSOFT'13, pages 25:1–25:10. ACM, 2013

THANK YOU

amir.haghshenas@polymtl.ca