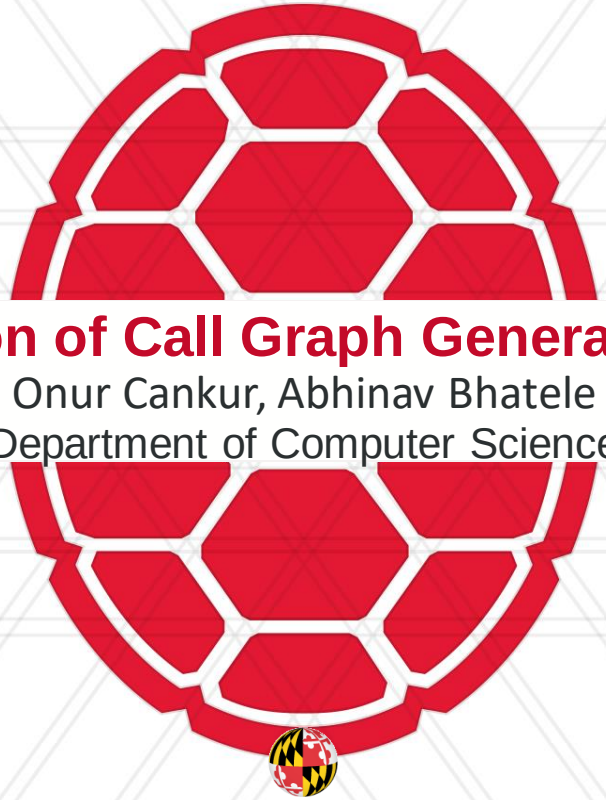


Comparative Evaluation of Call Graph Generation by Profiling Tools

Onur Cankur, Abhinav Bhatele
Department of Computer Science

UNIVERSITY OF MARYLAND

Introduction

- Analyzing the performance of parallel programs is critical but challenging.
- Profiling tools allow for measuring performance data.
- Different profiling methods and capabilities.
- Comparatively evaluate call graph data generation capabilities of several profiling tools.

Overview

Profiling Tools	Proxy Apps	Evaluation Metrics
• Caliper 2.6.0 • HPCToolkit 2021.05.15 • Score-P 7.1 • TAU 2.30.1	• AMG • LULESH • Quicksilver	1. Runtime Overhead 2. Memory Usage 3. Data Size 4. Data Correctness 5. Data Richness

Different Methods for Profiling

- Instrumentation
  - Instrument the source or binary code.
  - Collect performance measurements at each instrumentation point.
  - Can be done by the user or the tool itself.
- Sampling
  - Periodically sample the program, check the PC and collect function call stack.
  - Aggregate the performance measurements of a code block across multiple samples.

Tool	Samp.	Instr.
Caliper	Yes	Yes
HPCToolkit	Yes	No
Score-P	Partially	Yes
TAU	Yes	Yes

Comparison 1: Runtime Overhead

- The execution time of an application should not be perturbed significantly by the profiling tool.
- Measure the time using `MPI_Wtime()`.

$$Relative\ Execution\ Time = \frac{MPI\_Wtime(with\_profiling)}{MPI\_Wtime(without\_profiling)}$$

Comparison 2: Memory Consumption

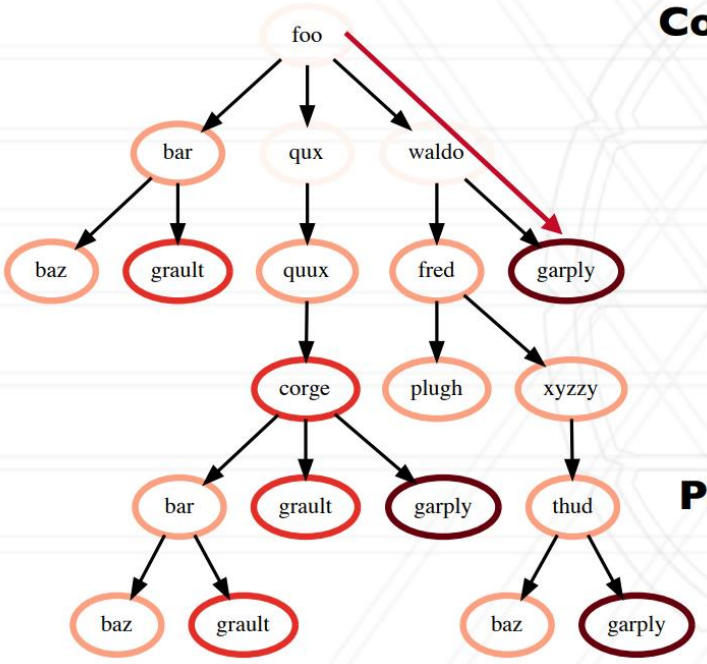
- Performance tools should not consume large amounts of memory.
- Obtain the largest memory usage at any point during program execution using `getrusage()`.

$$AMU = getrusage(with\_profiling) - getrusage(without\_profiling)$$

- AMU: Additional Memory Usage

Information Gathered by Profiling Tools

- CCT: tree of call paths
- Call path: sequence of function invocations
- Refer as “Call Graph”



Calling context tree (CCT)

Contextual information

```
File
Line number
Function name
Callpath
Load module
Process ID
Thread ID
```

Performance Metrics

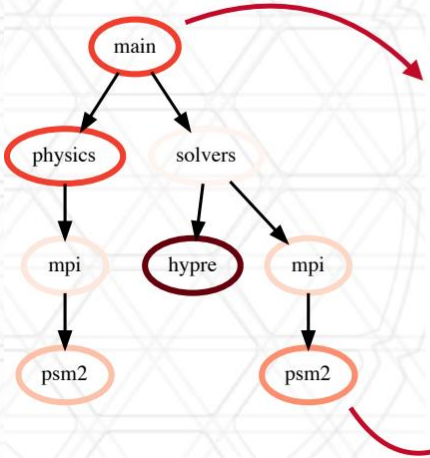
```
Time
Flops
Cache misses
```

Comparison 3: Size of Call Graph Data

- Profiling tools generate significant amount of call graph data.
- Use default settings and collect per-process data.
- Compare which tool generates more data.
- Generating more data might not be a downside.
  - Further analysis on the call graph data is needed.

Post-mortem Analysis

- Hatchet
- Programmatically analyze call graph data



node	name	nid	node	time	time (inc)
main	main	0	main	40.0	200.0
physics	physics	1	physics	40.0	60.0
mpi	mpi	2	mpi	5.0	20.0
psm2	psm2	3	psm2	15.0	15.0
solvers	solvers	4	solvers	0.0	100.0
hypre	hypre	5	hypre	65.0	65.0
mpi	mpi	6	mpi	10.0	35.0
psm2	psm2	7	psm2	25.0	25.0

- Graphframe = graph + dataframe

Comparison 4: Correctness of Call Graph Data

- Assumption: If the data generated by multiple tools is nearly identical, it should be close to the ground truth.
- Default settings are used for each tool.

Comparison 4: Correctness of Call Graph Data

Comparison Metrics	Rationale
Top five slowest call graph nodes	Do tools identify the same slowest nodes?
Call path of the slowest nodes	Can tools provide the correct program structure?
File	Can tools provide correct information about the source code?
Line number	
Hot nodes	Do tools identify the same most time-consuming call path?

Comparison 5: Richness of Call Graph Data

- Richness of call graph profiling data refers to having detailed information in the call graph.
- Default settings are used for each tool.

Comparison 5: Richness of Call Graph Data

Comparison Metrics	Rationale
Max. call path length	Investigate: • If the call graph data provides sufficient information. • If the call graph data has some abnormalities.
Avg. call path length	
Number of nodes	
Number of .so files	
Number of MPI functions	

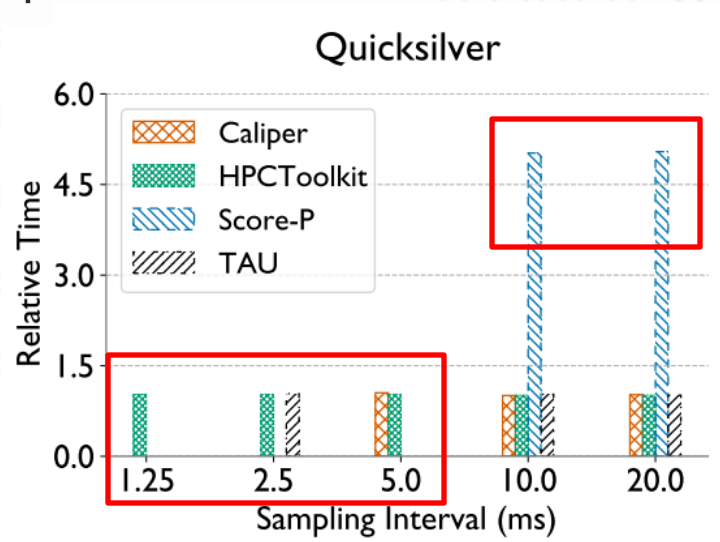
Experimental Setup

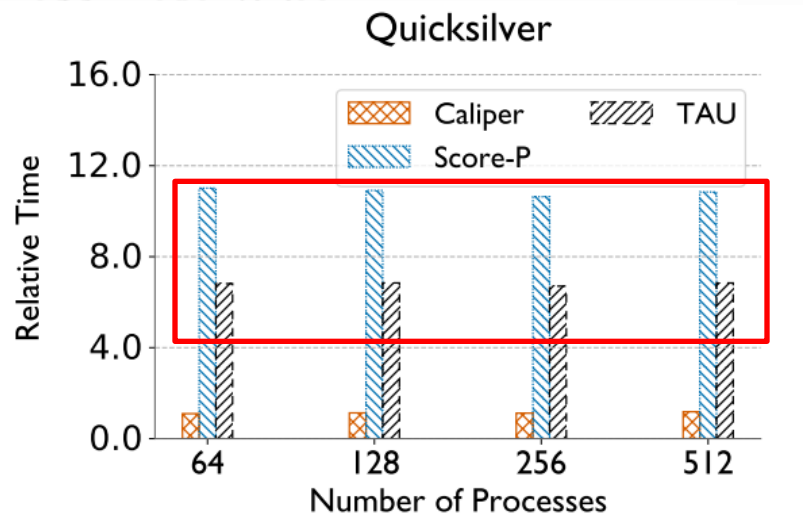
Profiling Tools	Proxy Apps	Evaluation Metrics	Environment
• Caliper 2.6.0 • HPCToolkit 2021.05.15 • Score-P 7.1 • TAU 2.30.1	• AMG • LULESH • Quicksilver	• Runtime Overhead • Memory Usage • Data Size • Data Correctness • Data Richness	• GCC 8.3.1 • Open MPI 3.0.1 • x86_64 architecture

- All applications are written in C/C++ and use only MPI.
- Weak scaling experiments.
- Sampling and instrumentation methods are evaluated separately.

Evaluation 1: Runtime Overhead

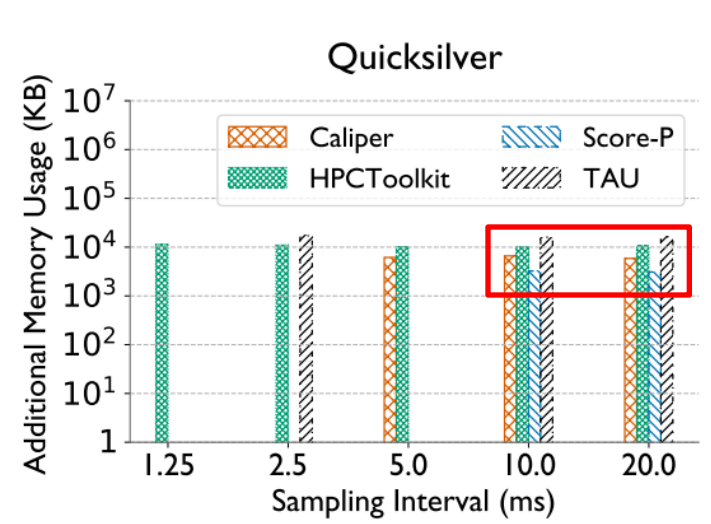
- Varying sampling intervals.
- Each execution used 64 MPI processes.
- Default instrumentation methods.

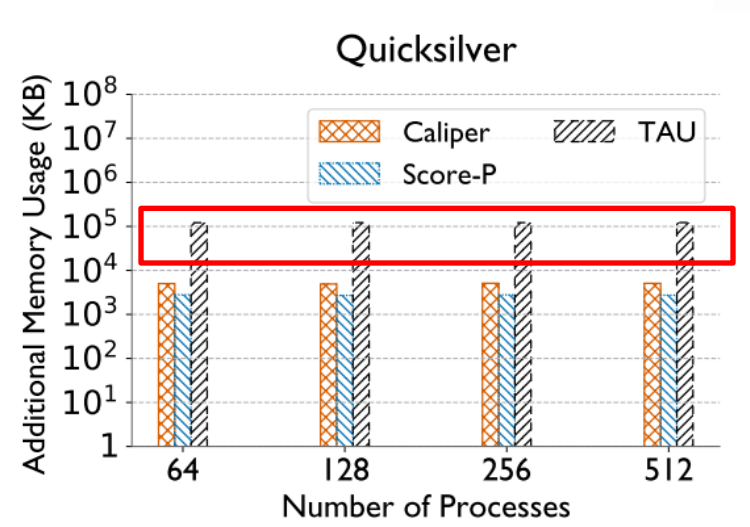




Evaluation 2: Memory Consumption

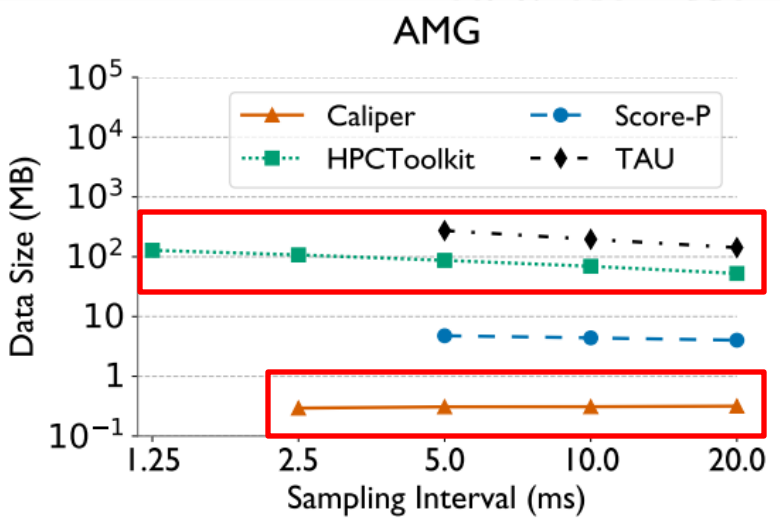
- Varying sampling intervals.
- Each execution used 64 MPI processes.
- Default instrumentation methods.

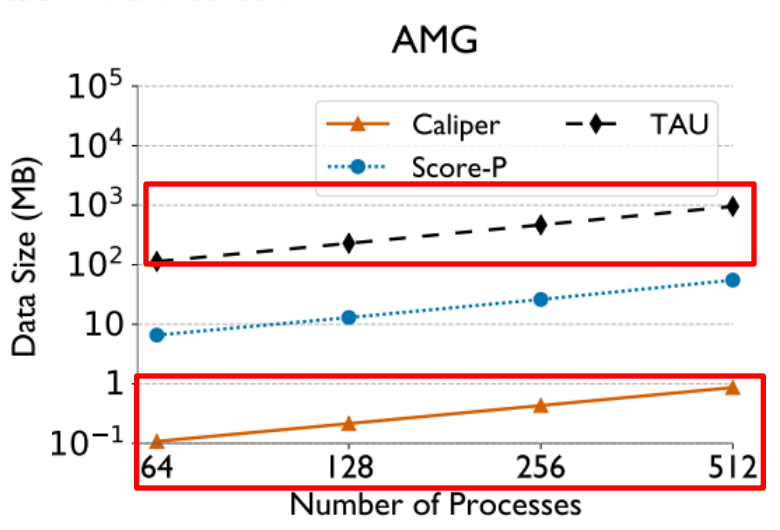




Evaluation 3: Generated Data Size

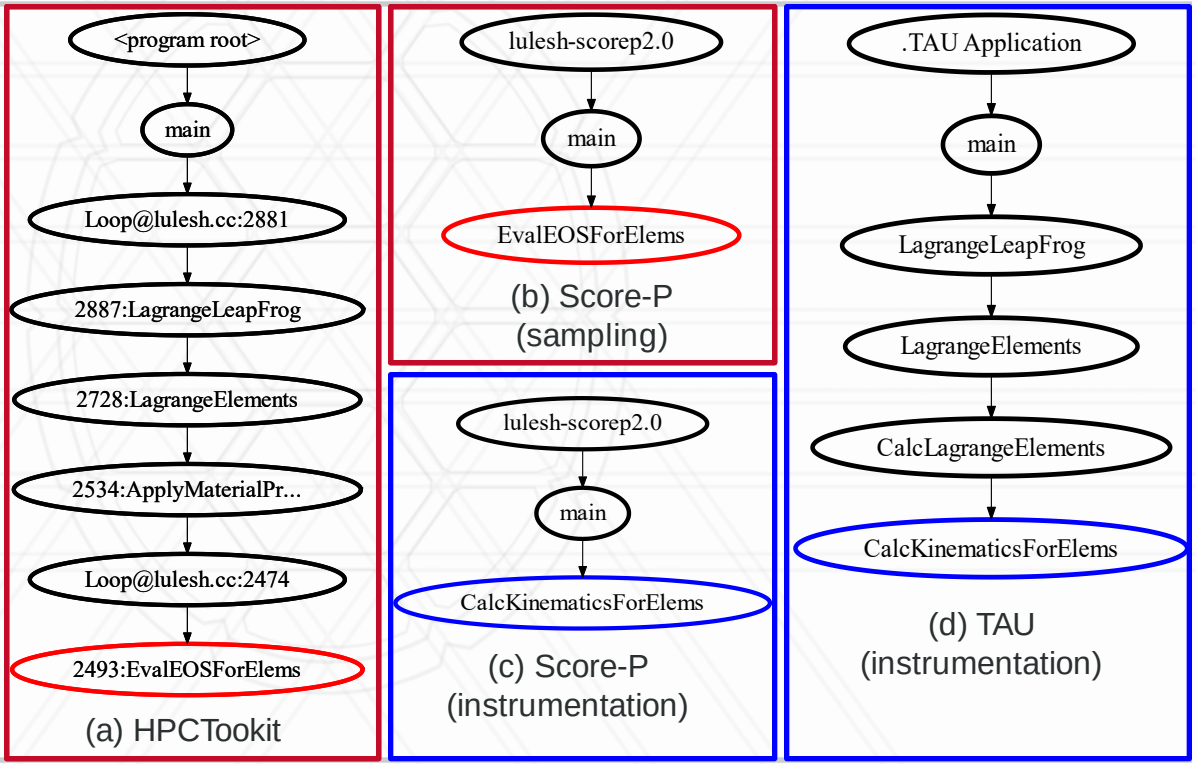
- Varying sampling intervals.
- Each execution used 64 MPI processes.
- Default instrumentation methods.





Evaluation 4: Call Path Correctness

- Call path of the second slowest node obtained by different tools for LULESH.
- The leaf nodes are the second slowest nodes.
- But the tools agree on the slowest node.



Evaluation 5: Call Path Richness

- Fixed sampling interval (20.0 ms)
- Default instrumentation methods.
- Each execution used 64 MPI processes.

App.	Tool	Method	Max depth	Avg depth	No. of nodes (all,unq)	No. of .so files (all,unq)	No. of MPI functions (all,unq)
AMG	Caliper	Sampling	30	9.724	(1414,739)	(363,**36**)	(37,17)
		Instrumentation	3	2.384	(50,22)	0	(38,10)
	HPCToolkit	Sampling	**35**	**13.931**	(12112,**2528**)	(**4616**,25)	(585,66)
	Score-P	Sampling	63	10.859	(1470,199)	0	(668,52)
		Instrumentation	163*	31.428*	(3117,332)	0	(676,51)
	TAU	Sampling	12	8.416	(**13645**,1976)	(2010,20)	(**1036,91**)
		Instrumentation	111*	10.12*	(1956,334)	0	(683,52)

Summary

- The first empirical study on call graph data generation capabilities of profiling tools.
- Evaluated Caliper, HPCToolkit, Score-P, and TAU.
- Considered runtime overhead, memory consumption, and the size and quality of the generated call graph data.

UNIVERSITY OF
MARYLAND

Joshua Harless
3101 Turner Hall, College Park, MD 20742
301.405.3384 / jharless@umd.edu