Frequency-based Overhead Compensation in HPC Application Traces

Alef Farah [1] Lucas Mello Schnorr [1 2] Jean-Marc Vincent [2]

[1]GPPD - INF - UFRGS

[2]Univ. Grenoble-Alpes, France

WSPP 2016

Application tracing

- **Performance analysis**
- Logging of significant events
- Unique identifiers (timestamp)
- Chronological order
- **Parallel and distributed applications**

Overhead in traces?

Additional instructions!

- Direct perturbations
  - Logging overhead

Overhead in traces?

Additional instructions!

- Direct perturbations
  - Logging overhead $\rightarrow$ log less

Overhead in traces?

Additional instructions!

- Direct perturbations
  - Logging overhead $\rightarrow$ log less $\rightarrow$ **compensation**

Overhead in traces?

Additional instructions!

- Direct perturbations
  - Logging overhead $\rightarrow$ log less $\rightarrow$ **compensation**
- Indirect perturbations
  - Compilier optimizations

Overhead in traces?

Additional instructions!

- Direct perturbations
  - Logging overhead $\rightarrow$ log less $\rightarrow$ **compensation**
- Indirect perturbations
  - Compilier optimizations $\rightarrow$ binary instrumentation

Overhead in traces?

Additional instructions!

- Direct perturbations
  - Logging overhead → log less → **compensation**
- Indirect perturbations
  - Compilier optimizations → binary instrumentation
  - Cache, CPU optimizations

Compensation and overhead measurement

$$event_a^i = event_a^{i-1} + (event_m^i - event_m^{i-1}) - O$$

- Isolate the logging routine
- Take enough measurements
- Produce an estimator (e.g. mean)

Compensation and overhead measurement

$$event_a^i = event_a^{i-1} + (event_m^i - event_m^{i-1}) - O$$

- Isolate the logging routine
- Take enough measurements
- Produce an estimator (e.g. mean)
- Very fast routines $\rightarrow$ high variability

Frequency

Logging overhead is a function of the logging **frequency**. The difference may be small, the error is **cumulative**. Also

- How high is the variability?
- What can be done about it?

Notes

- Mean frequency for the entire trace
- **Regular applications**
- MPI

Metrics

How to compare with previous methods?

- Total execution time
- Space/time view

Platform

- 2 NUMA nodes
- Intel Xeon E5-2630 (24 PU total)
- 32 GB RAM
- OpenMPI 1.6.5
- **Shared memory**
- Linux 3.16.0-51 (Ubuntu 14.04.1), GCC 4.8

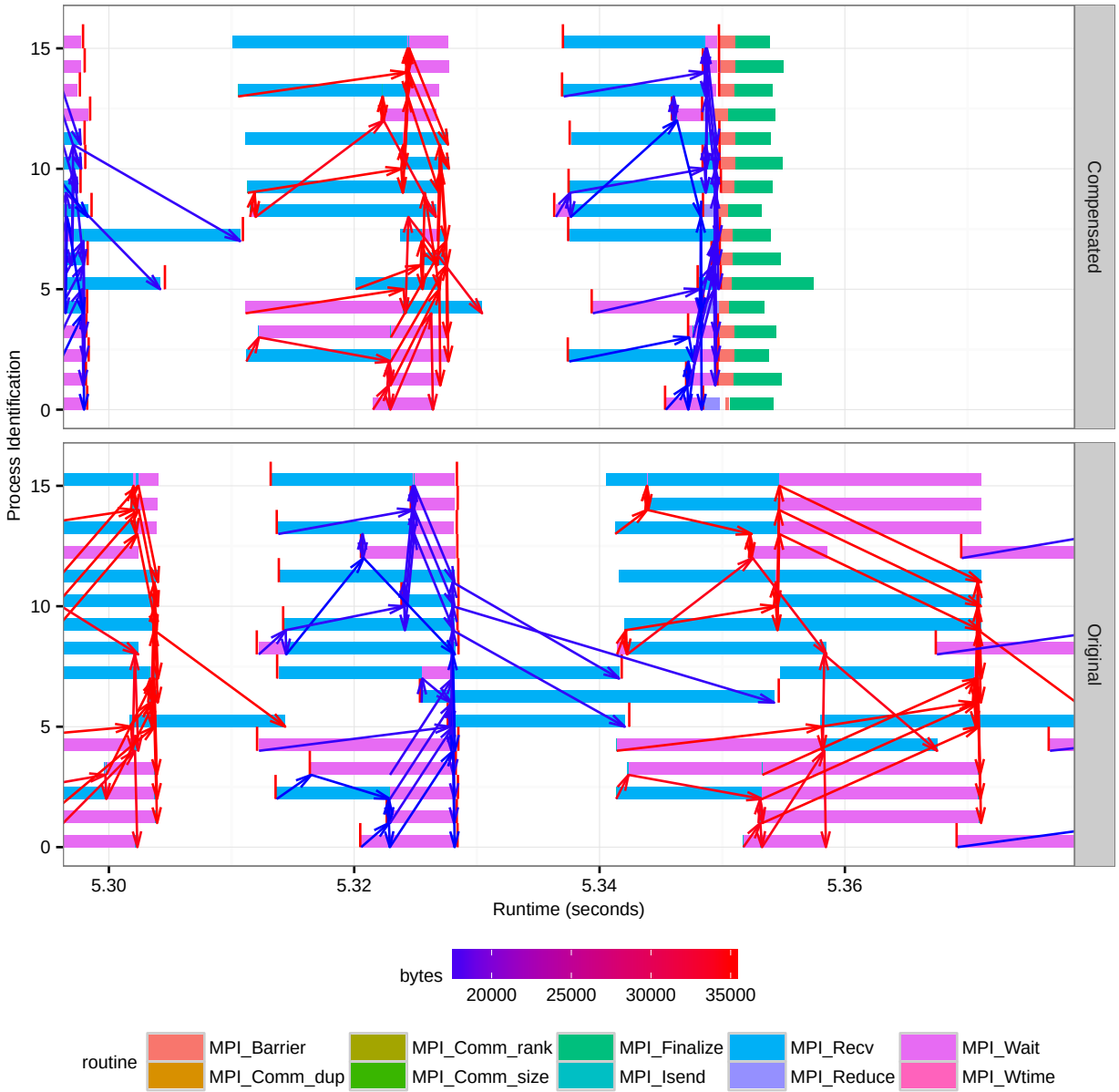
Applications

- OSU Microbenchmarks v5.2 (`osu_multi_lat`)
- Ondes3D v1.1

OSU Microbenchmarks

Execution	Mean	Standard error
UnInstrumented	12.9576502561	0.280464331573357
Instrumented	13.1024921894073	0.176561479255295
Traditional	13.0582891357813	0.176510576519728
Frequency	12.9450804535228	0.176508398007264

Ondes3D


Compensated
Original
Process Identification
0
5
10
15
5.30
5.32
5.34
5.36
Runtime (seconds)
bytes
20000
25000
30000
35000
routine
MPI_Barrier
MPI_Comm_dup
MPI_Comm_rank
MPI_Comm_size
MPI_Finalize
MPI_Isend
MPI_Recv
MPI_Reduce
MPI_Wait
MPI_Wtime

Conclusion

- Execution time is a function of the frequency
- Care should be taken with measurement variability
- Encouraging results using coarse metric
- Inconclusive results using fine grain metric

Future work

- Test traces with higher intrusion
- Tests in a networked environment
- Tests with tools with a higher intrusion
- Search for a fine grain metric
- Investigation with irregular applications

Thank you for the attention!

The results reported in this study were generated in virtue of the agreement between Hewlett Packard Enterprise (HPE) and the Federal University of Rio Grande do Sul (UFRGS), financed by resources in return for the exemption or reduction of the IPI tax, granted by Brazilian Law nº 8248, 1991, and its subsequent updates. This investigation also receives funds from the H2020 program EU and MCTI / RNP-Brazil through HPC4E project with code 689772, the FAPERGS / Inria ExaSE design, universal design CNPq 447311 / 2014-0, and international CNRS / LICIA laboratory.