Performance analysis is easy
... but obtaining the traces is not

C. Niethammer, A. Shamakina,
S. Walter, J. Gracia
HLRS, U Stuttgart

POP Project: performance analysis as a service

- CoE POP – Performance Optimisation and Productivity
- EU-funded
- Partners: **BSC**, HLRS, JSC, NAG, RWTH-Aachen, Teratec

- Service activities:

  ? Performance Audit:

  identify performance issues

  ! Performance Plan:

  root cause analysis and recommendations

  ✓ Proof-of-Concept:

  prototype code changes

  show effect of proposed optimisations

Performance analysis is simple

Musubi performance assessment report

Document Information

Reference Number	PO rsion 0.3
Author	Stephan Walter (HLRS)
Contributor(s)	José Gracia (HLRS)
Date	21.07.2016
Application	
Service Level	Performance Audit
Keywords	Load Balance, MPI, Strong Scaling, Performance variability

Notices: *The research leading to these results has received funding from the European UnionâĂŹs Horizon 2020 research and innovation programme under grant agreement No n° 676553.*

POP Ref.No. CONTENTS

Contents

From these basic measurements we can derive the following performance metrics.

ranks	12	192
Parallel Efficiency	93%	82%
Load Balance	94%	86%
Communication Efficiency	99%	95%
Computation Scalability (strong scaling)	100%	96%
Instructions Scalability	100%	94%
IPC Scalability [3]	100%	106%
Global Efficiency	92%	78%

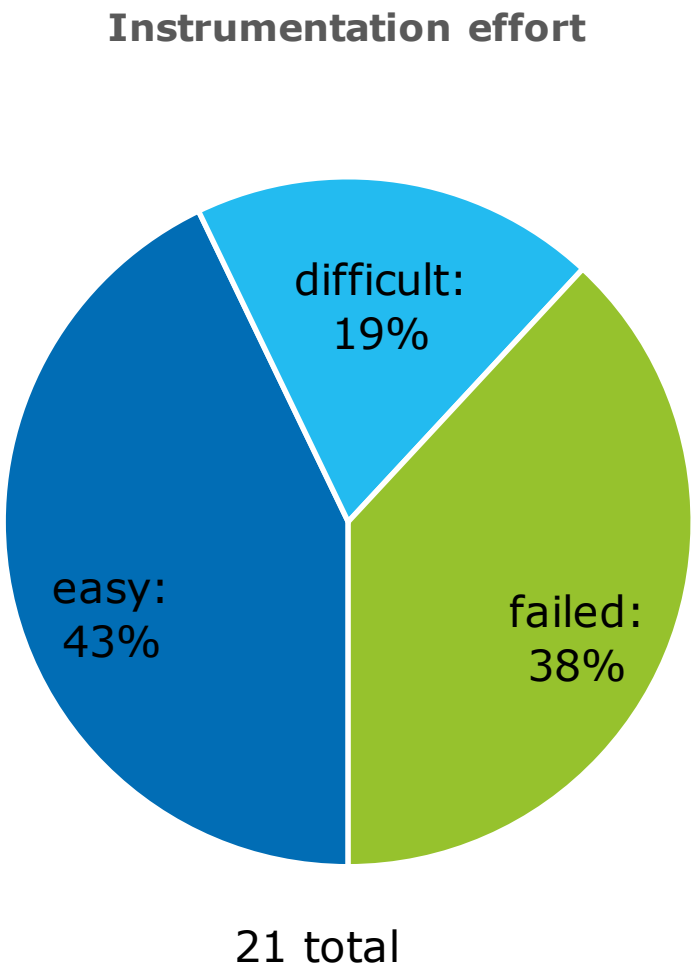
Table 2: Efficiency metrics for the *FoA*.

Obtaining traces is non-trivial in many cases

- HLRS: 21 tracing cases
  (code, tool, target)

- *target* specified by customer,
  e.g.: MPI, MPI+OpenMP, IO

- classified *instrumentation effort* to obtain full traces for target:
  - easy: just using official documentation
  - *difficult*: required sources beyond docu, e.g. developers, other specialist
  - *failed: impossible to obtain full traces as planned*



Closer look at instrumentation effort

	easy/difficult/failed [%]	total#
▪ By language:		
Fortran	58 / 17 / 25	12
C++	0 / 33 / 67	3
mixed (Fortran & C/C++)	33 / 17 / 50	6
▪ By tool:		
VI-HPS	29 / 21 / 50	14
others (Cray, perf, …)	71 / 14 / 14	7
▪ By customer:		
HLRS user	62 / 13 / 25	8
others	31 / 23 / 46	13

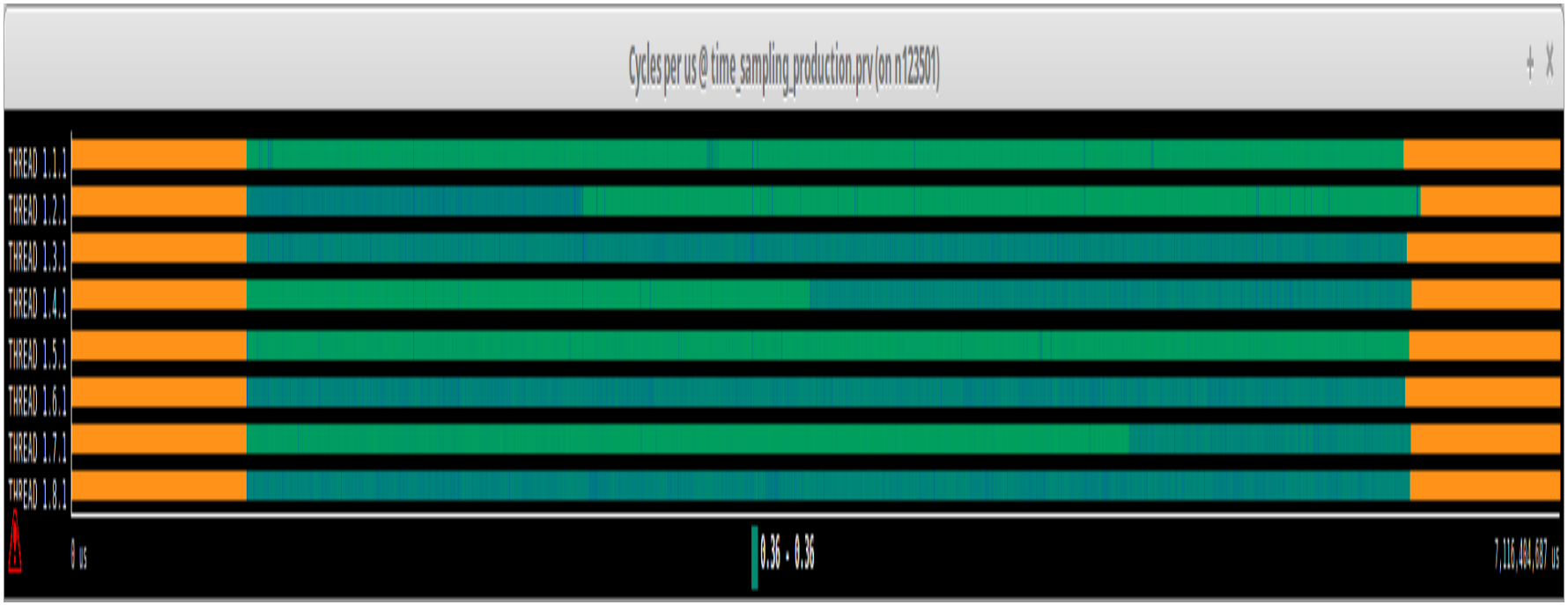
POP customers seem to have more "complex" codes than HLRS users

Some instrumentation issues

- large instrumentation overhead for C++ -> trace useless
  setting up filtering for C++ codes is non-trivial
- collecting data from IO, in particular Fortran
- code spawns new process which is invisible to tracing
- unreliable time-stamps in traces
- uncommon language or programming model constructs
- incomplete documentation

Misleading failed instrumentation

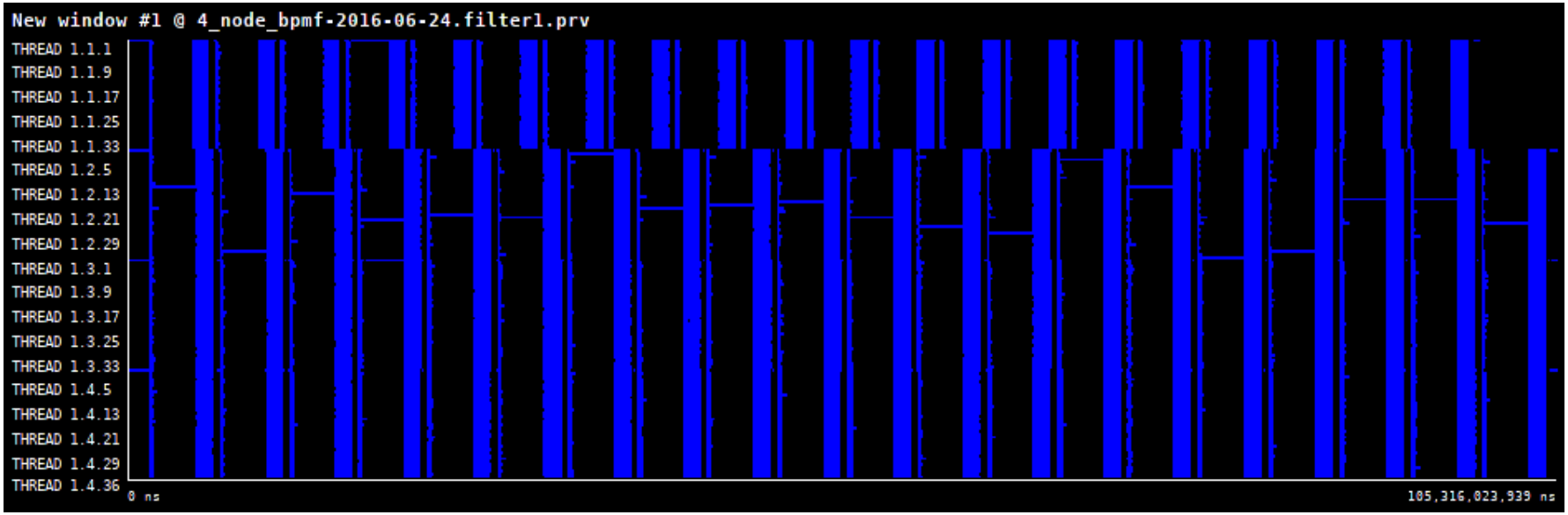
Looks like really low IPC …



but is a spawned Fortran process which remains invisible to tracing infrastructure

Misleading failed traces

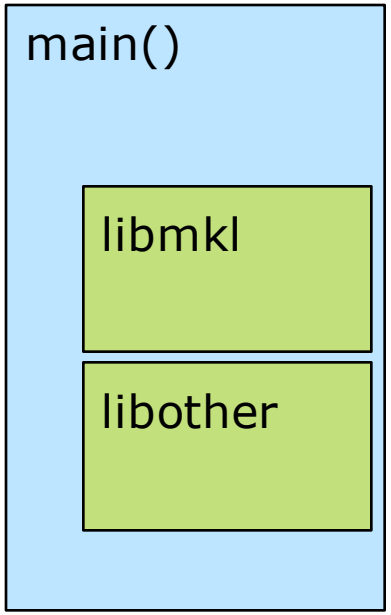
Looks like load-imbalance(?), one node does different stuff …



but is unreliable time-stamps due to clock skew.

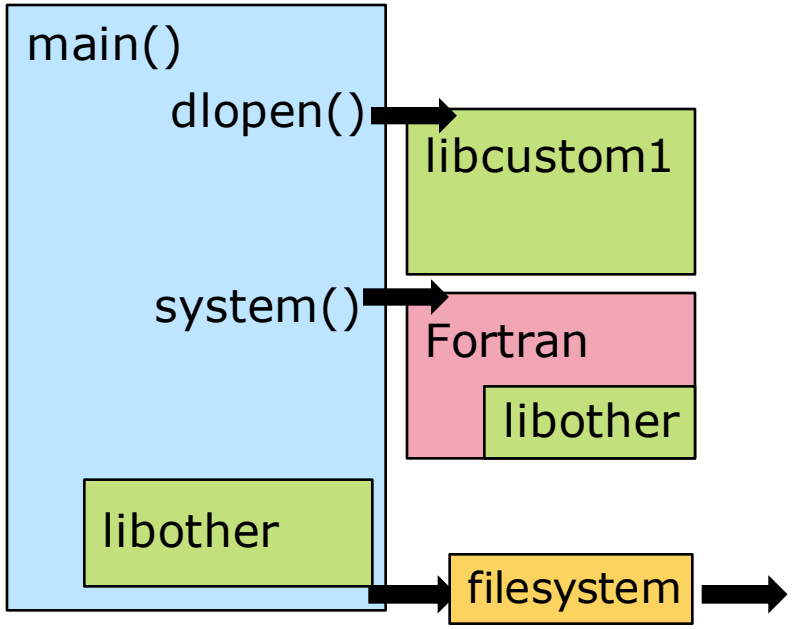
Are POP customer codes different?

typical HLRS user code?



monolithic binary

typical POP code?



multiple binaries, workflow, etc

Are "typical" HPC codes only a non-representative subgroup of all codes running on clusters?

Discussion

- Analysis of POP applications is relatively simple
  - serialisation of communication
  - algorithmic load-imbalance
- Instrumenting POP applications is relatively difficult
- Larger variation of HPC codes outside of large centres? Multi-binary, workflows?
- As a community, put less effort in analysis capabilities? And more in instrumentation framework?

Thank You