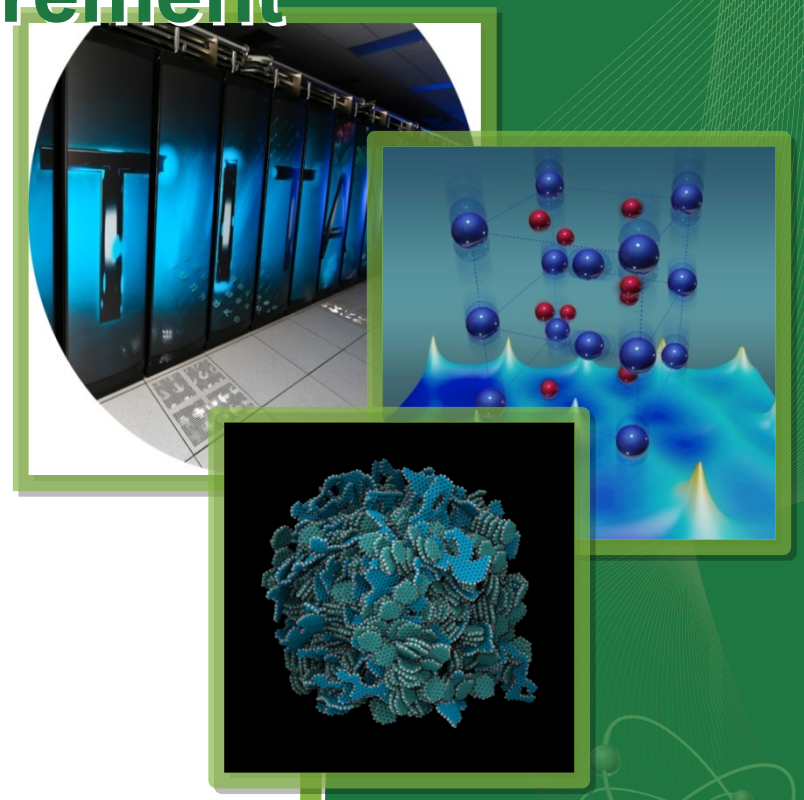
Generic Library Interception for Improved Performance Measurement and Insight



Ronny Brendel, Bert Wesarg, Ronny Tschüter, Matthias Weber, Thomas Ilsche, Sebastian Oeste

6th Workshop on Extreme-Scale Programming Tools (ESPT), Denver, USA, 12th November 2017

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Motivation & Related Work

- Libraries are increasingly important
  - Automatic compiler instrumentation does not provide instrumentation for external code
  - Sampling does not measure exact time and call counts

- Fixed wrappers are commonplace in performance analysis tools
- Aside from performance analysis, library interception is desirable for:
  - Providing wrappers for other languages (SWIG/CLIF)
  - Performance tuning (Spectrum MPI)
  - and e.g. correctness checking (MUST)

Methodology

- A wrapper library has to look exactly the same as the original one
  - Exact same symbol names (function names + signatures)
- And has to forward the calls
  - Needs to know all the return and argument types so that it can forward them
- C/C++ makes this very hard

Methodology

```
int printf(const char *format, ...);
int function_pointer_arg(int (* f)(double));
int function_pointer_without_star_arg(int (f)(double));


typedef int (*fn_t)(double);
fn_t return_function_pointer_typedef(void);      1
int(*)(double) return_function_pointer(void);    2
int (*return_function_pointer(void)) (double);   3
```

Methodology

```
int printf(const char *format, ...);
int function_pointer_arg(int (* f)(double));
int function_pointer_without_star_arg(int (f)(double));

typedef int (*fn_t)(double);
fn_t return_function_pointer_typedef(void);      1
int(*)(double) return_function_pointer(void);    2
int (*return_function_pointer(void)) (double);   3
```

Methodology

```
typedef int array_type[2];
int typedef_array_return(void)[2];
array_type typedef_array_return(void);
array_type* typedef_array_return(void);  1
int[2]* array_return_1(void);            2
int array_return_2(void)[2]*;            3
```

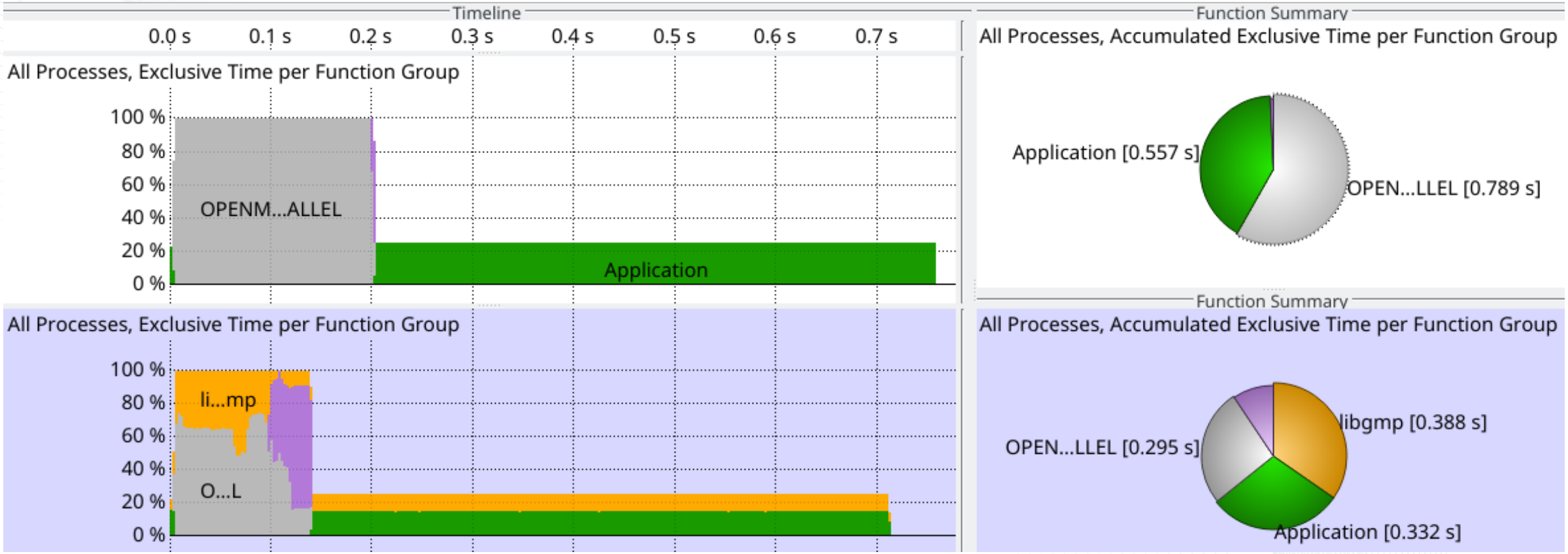
Methodology

```
typedef int array_type[2];
int typedef_array_return(void)[2];
array_type typedef_array_return(void);
array_type* typedef_array_return(void);  1
int[2]* array_return_1(void);            2
int array_return_2(void)[2]*;            3
```

Methodology

- The symbol corresponding to the function has to exist
- Other caveats
  - Macros in headers
  - Compile flags, compiler wrappers
  - Static versus dynamic library linking and wrapping
  - Multiple headers, multiple libraries
- All these considerations led to us conceiving a **library wrapper creation and usage workflow**

Workflow (Live)



Case Studies

- Gromacs (FFTW)
- PERMON (PETSc)

- Examples:
  - Qt Widgets & Qt Gui
  - GNU GMP
  - math.h

Conclusions

- Usable C/C++ library wrapping
- Provides insight into
  - Library Usage
  - Closed-source libraries
  - Interaction between libraries

Future Work

- Support staff friendliness
  - Automation
  - Reuse
  - Versioning
- Score-P for unmodified binaries
- Record function arguments
- Independence from Score-P

Thank you

- http://automaton2000.com/espt.pdf
- https://github.com/score-p/scorep_libwrap_examples

This research used resources of the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.
This work is supported in part by the German Research Foundation (DFG) within the CRC 912 - HAEC.