Tool support for HPC performance optimization and productivity services

Dr. Joachim Jenke (jenke@itc.rwth-aachen.de)

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Why am I here today?

- Developing HPC tools since 2010
  - Score-P (performance: tracing)
  - MUST (correctness: MPI runtime error detection)
  - Archer (correctness: OpenMP-aware data race detection)
  - OTF-CPT (performance: on-the-fly critical path analysis)

- Contributing to OpenMP standard and MPI specification
  - OMPT + OMPD
  - MPI continuations
  - MPI handle debugging interface

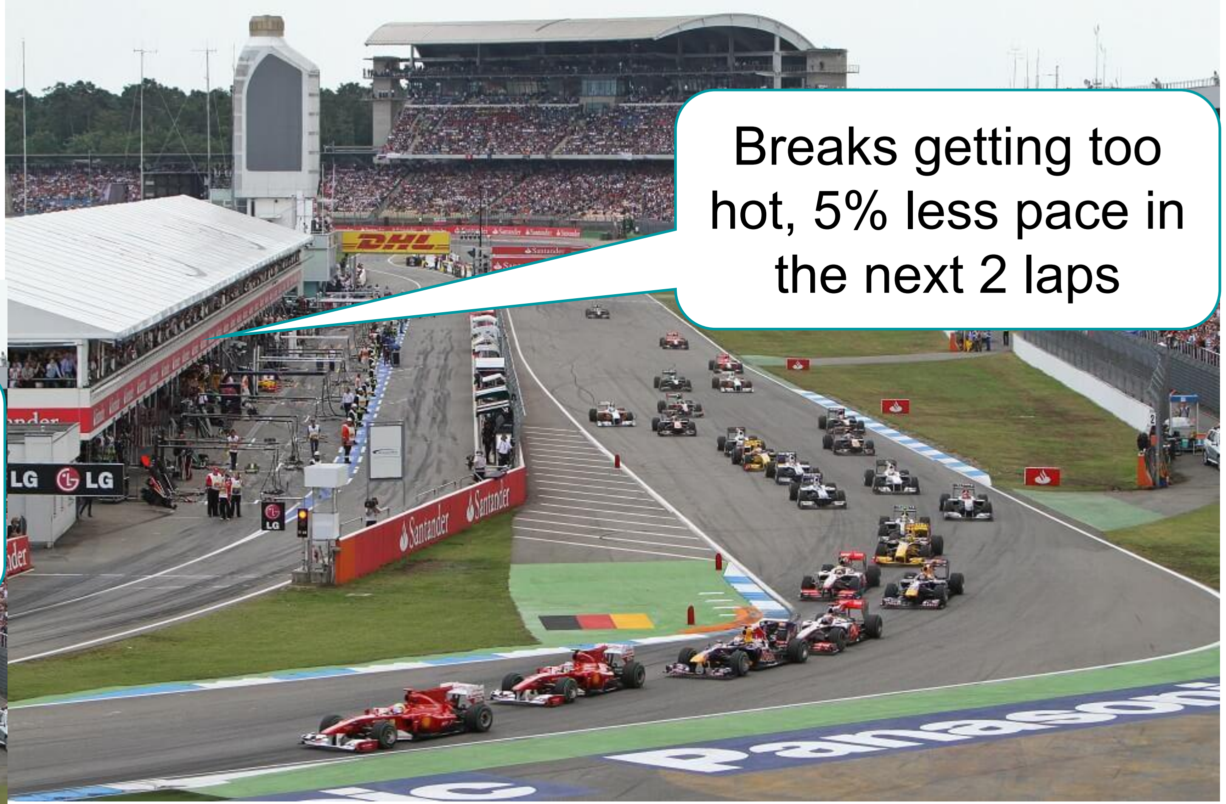
Standardization Work

Standardization work

- 👍 Interaction with great community
- 👍 Involves quite some travelling (needs funding)
- 👍 Great chance for networking
- 👎 Process of getting a feature into a standard exceeds the duration of a typical PhD

Tools interfaces in OpenMP (‘14 - ‘18 and ongoing)

- OMPT: 1st person view
- The tool executes as part of the application
- E.g.: Performance / runtime correctness tools



- OMPD: 3rd person view
- The tool executes in a separate process
- E.g.: Debuggers

MPI handle debugging interface

Use case:

Execution stalls in `MPI_Wait(&request, &status);`

- ➢ What kind of request? Where does it come from? → `MPI_Irecv` in `foobar.c:42`
- ➢ Who is the expected sender?
  - – Which source? How does it translate to a process in the debugger?
- ➢ What is the tag?
- ➢ Are there any pending messages from this source? Possibly a tag mismatch?

Segfault in `MPI_Recv(buffer, count, vtype, source, 23, MPI_COMM_WORLD, &status);`

- ➢ What memory would be written by this recv considering the type information?

OpenMP + MPI Tools Work

Motivation: Undefined Behavior: What could go wrong?

- UB allows compilers any behavior
  - ➢ Possible optimization: assume absence of UB
  - ➢ Unexpected results
  - ➢ Avoid UB in any case!

```
void contains_null_check(int *P) {
  int dead = *P;
  if (P == 0)
    return;
  *P = 4;
}
```

clang 17:

```
contains_null_check(int*):
        test    rdi, rdi                # P == 0
        je      .LBB0_2                 # skip
        mov     dword ptr [rdi], 4      # *P = 4
.LBB0_2:
        ret                             # return
```

gcc 13:

```
contains_null_check(int*):
        mov     dword ptr [rdi], 4      # *P = 4
        ret                             # return
```

OMPT tool: Archer

- OpenMP-aware **data race** detection (identifying UB)
- Based on ThreadSanitizer in LLVM / GNU compilers

- Shipped with LLVM since 10.0

- Early adopter tool for new OpenMP / OMPT functionality
  - E.g.: detached tasks, free-agent tasks

- Recently implemented features (in context of ECP SOLLVE):
  - DR analysis for SIMD instructions (TSan)
  - Task-centric analysis (Archer runtime)
  - Improved analysis for reductions (OpenMP codegen, TSan)
  - Evaluated Archer use with flang

Tool configuration	FN	TN	TP	FP
LLVM 17 release	36	110	73	2
thread-centric	22	**112**	87	**0**
task-centric	**14**	**112**	**95**	**0**

- Intel Inspector is discontinued → Archer now available with `icx`

Data race detected in NEST Simulator

```
==================
WARNING: ThreadSanitizer: data race (pid=111865)
  Write of size 1 at 0x7b1000056a70 by main thread:
    #0 Token::datum() const nest-simulator/sli/token.h:362:15
    #1 double getValue<double>(Token const&) nest-simulator/sli/tokenutils.cc:77:53
    #2 bool updateValue<double, double>(lockPTRDatum<Dictionary, &SLIInterpreter::Dictionarytype> const&, Name, double&) nest-simulator/sli/dictutils.h:253:11
    #3 nest::Connection<nest::TargetIdentifierIndex>::set_status(lockPTRDatum<Dictionary, &SLIInterpreter::Dictionarytype> const&, nest::ConnectorModel&) 
nest-simulator/nestkernel/connection.h:364:8
    #4 nest::static_synapse<nest::TargetIdentifierIndex>::set_status(lockPTRDatum<Dictionary, &SLIInterpreter::Dictionarytype> const&, nest::ConnectorModel&) 
nest-simulator/models/static_synapse.h:199:19


  Previous write of size 1 at 0x7b1000056a70 by thread T1:
    #0 Token::datum() const nest-simulator/sli/token.h:362:15
    #1 double getValue<double>(Token const&) nest-simulator/sli/tokenutils.cc:77:53
    #2 bool updateValue<double, double>(lockPTRDatum<Dictionary, &SLIInterpreter::Dictionarytype> const&, Name, double&) nest-simulator/sli/dictutils.h:253:11
    #3 nest::Connection<nest::TargetIdentifierIndex>::set_status(lockPTRDatum<Dictionary, &SLIInterpreter::Dictionarytype> const&, nest::ConnectorModel&) 
nest-simulator/nestkernel/connection.h:364:8
    #4 nest::static_synapse<nest::TargetIdentifierIndex>::set_status(lockPTRDatum<Dictionary, &SLIInterpreter::Dictionarytype> const&, nest::ConnectorModel&) 
nest-simulator/models/static_synapse.h:199:19
```

```
nest-simulator/sli/token.h
359|  Datum* datum() const {
362|    accessed_ = true;
363|    return p;
364|  }
```

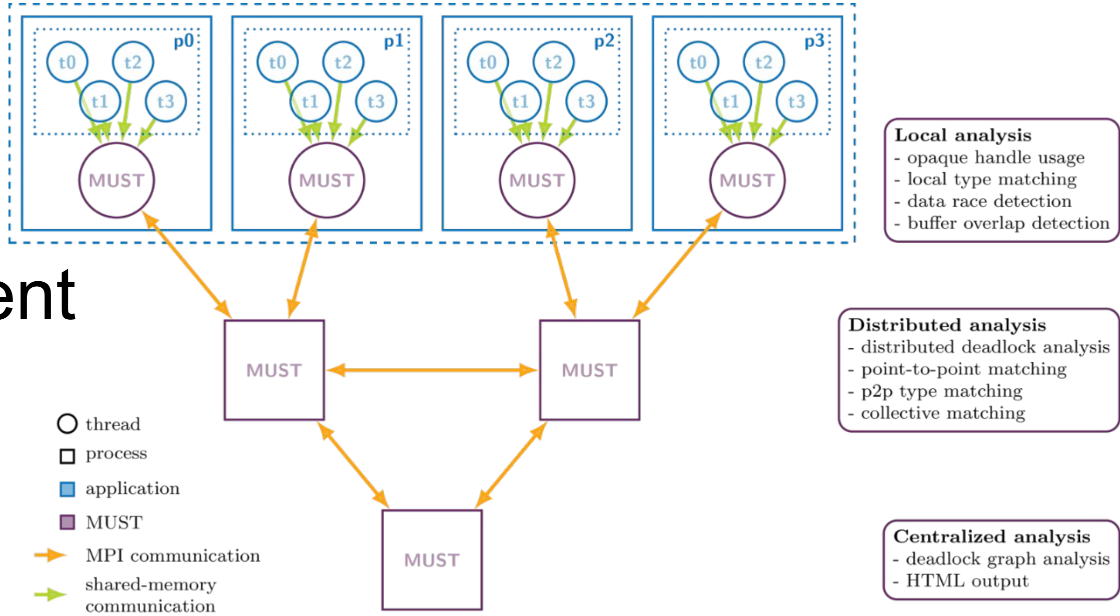
```
nest-simulator/sli/token.h
162|  mutable bool accessed_;
```

MUST

more at: https://git-ce.rwth-aachen.de/hpc-public/must-tutorial

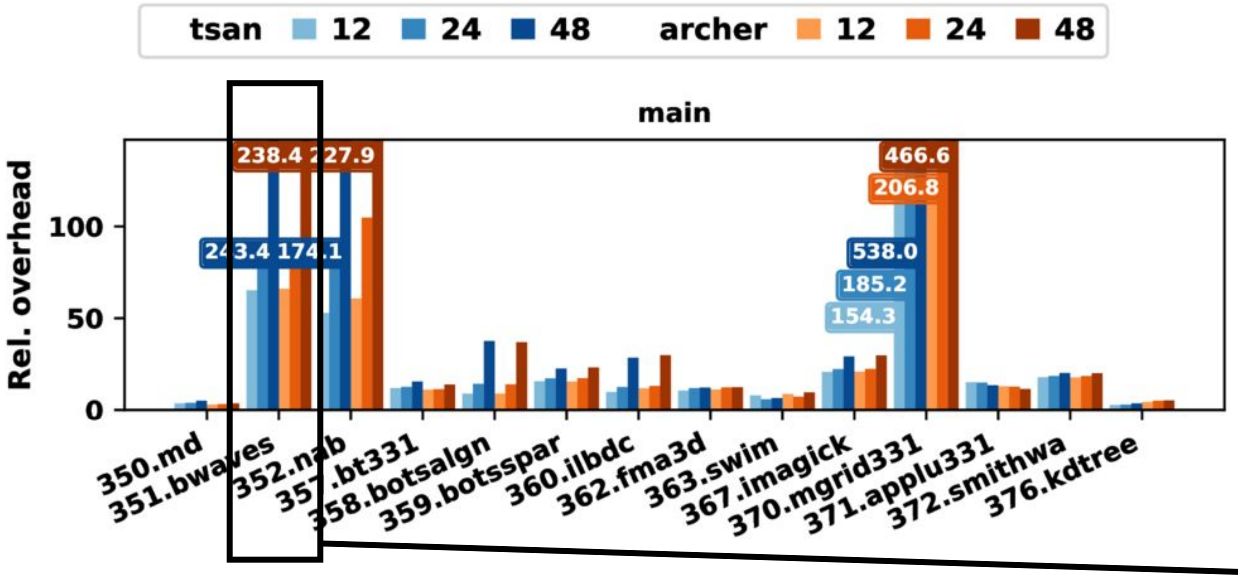
- Runtime correctness analysis for MPI applications
- Correctness’23: Data race analysis for hybrid MPI + OpenMP tasking
- Analysis for hybrid applications is still a construction site
  - Making all analyses thread-safe
  - Update and integrate hybrid DL-analysis
  - For MPI-thread-multiple, DL-analysis reports false positives

- CI is important, also for tool development
  - Running 4500 tests for each commit
  - Covering different MPI/compiler setups

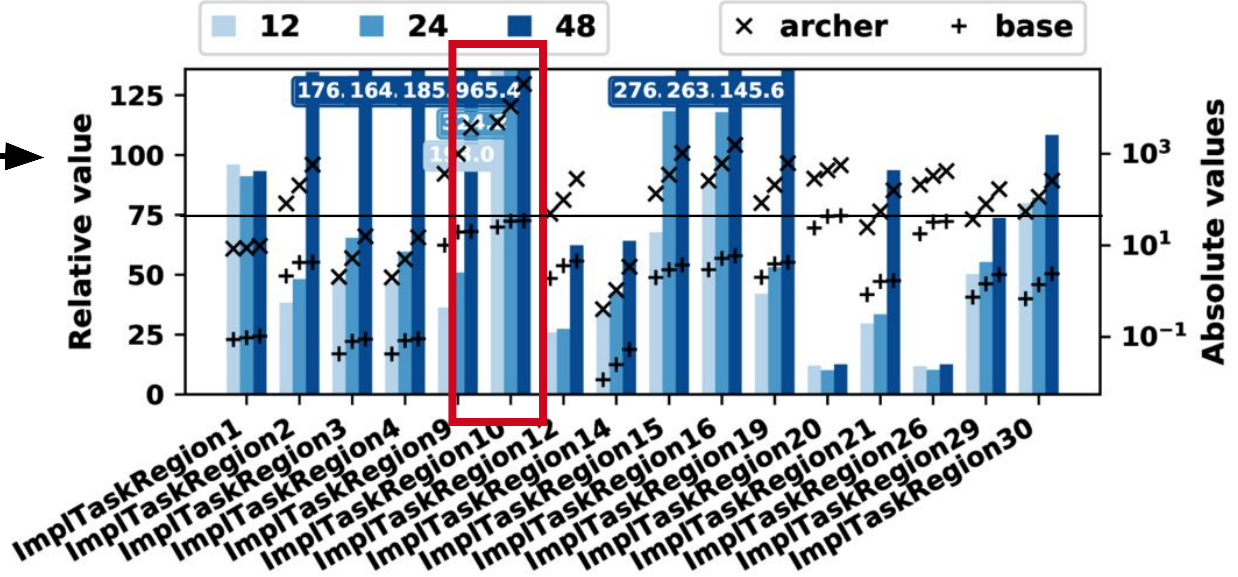


Differential performance analysis of dynamic data race detection

CORRECTNESS@SC'21
Protools@SC'21



- Break down runtime overhead to OpenMP tasks
  - Implicit tasks represent the threads within a parallel region
- Implicit task region 10 (`shell_lam.fppized.f:231`)
  - highest runtime overhead
  - highest execution time
  - significant base execution time



- Among all apps, 351.bwaves, 352.nab, and 370.mgrid331 show the highest runtime overhead

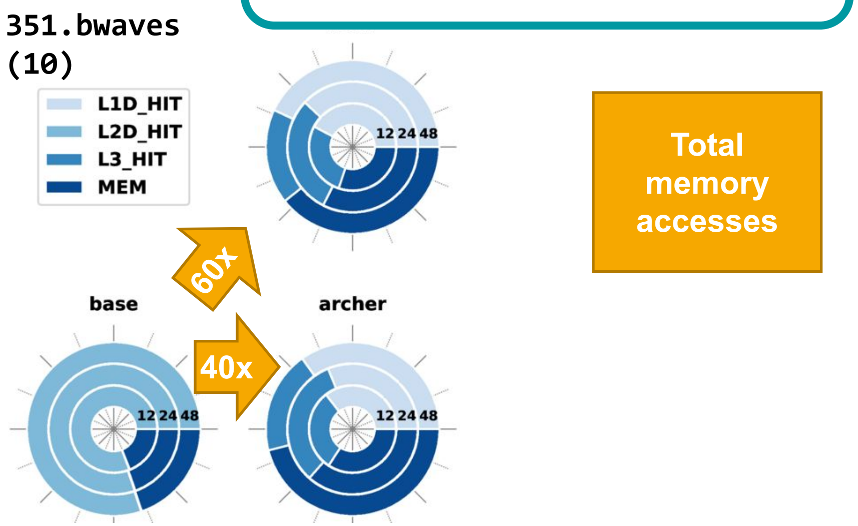
- Focus further analysis on these 3 apps

OMPT tools: light-weight OpenMP performance analysis tools

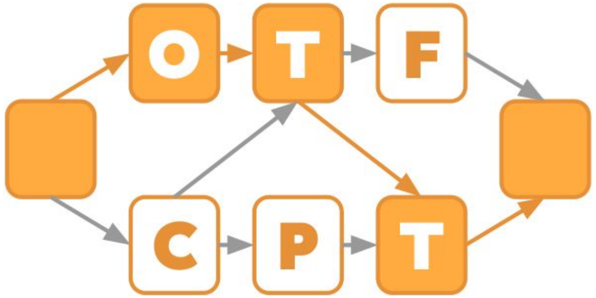
Protools@SC’21
Euro-Par’22, EuroMPI’23

OMPT profiler

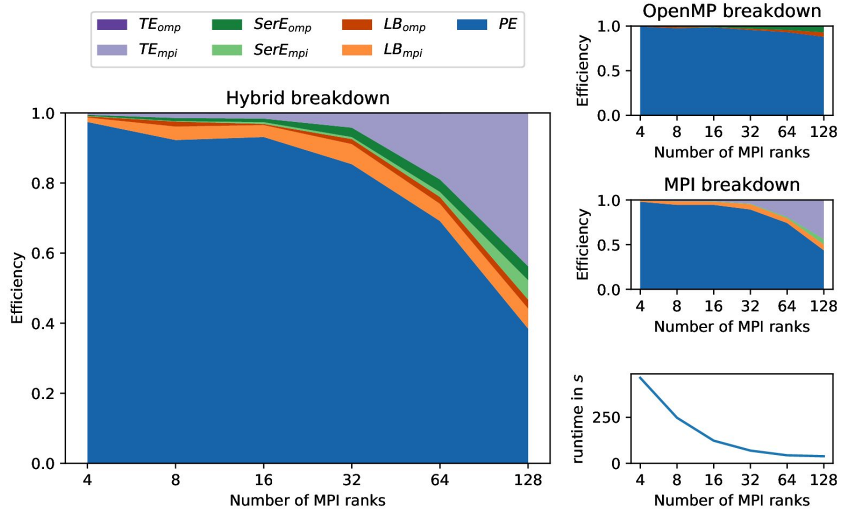
- Callstack/flat profiling based on OpenMP regions
- User regions based on `omp_control_tool`
- Integration of PAPI counters



Critical path tool



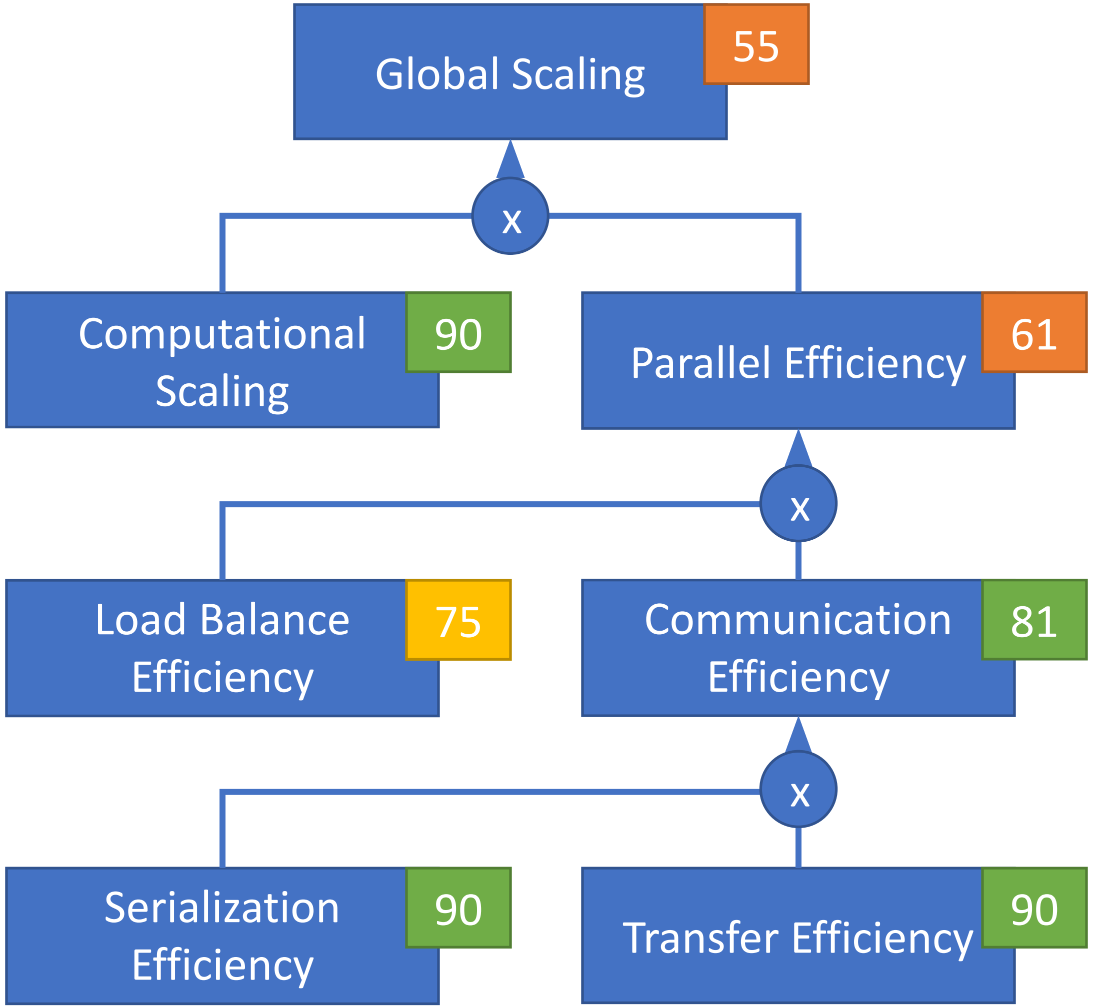
- Tracking critical path at runtime
- Hybrid PMPI + OMPT instrumentation
- Calculates Hybrid Model Factors on-the-fly
- Usecase for EuroMPI’23 paper on properly tracking requests



Performance Model Factors (as used in POP)

Performance model factors

- Hierarchy of metrics developed at BSC
- Highlight issues in the parallel structure of an application
- Parallel Efficiency breaks down into
  - Load balance
  - Serialization
  - Transfer
- Computational Scaling captures impact of scaling to node-level performance



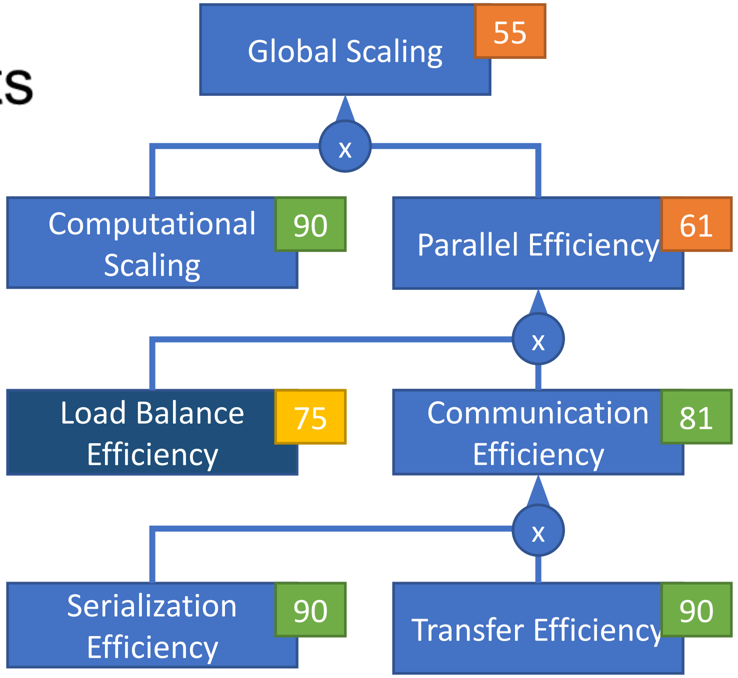
Performance model factors

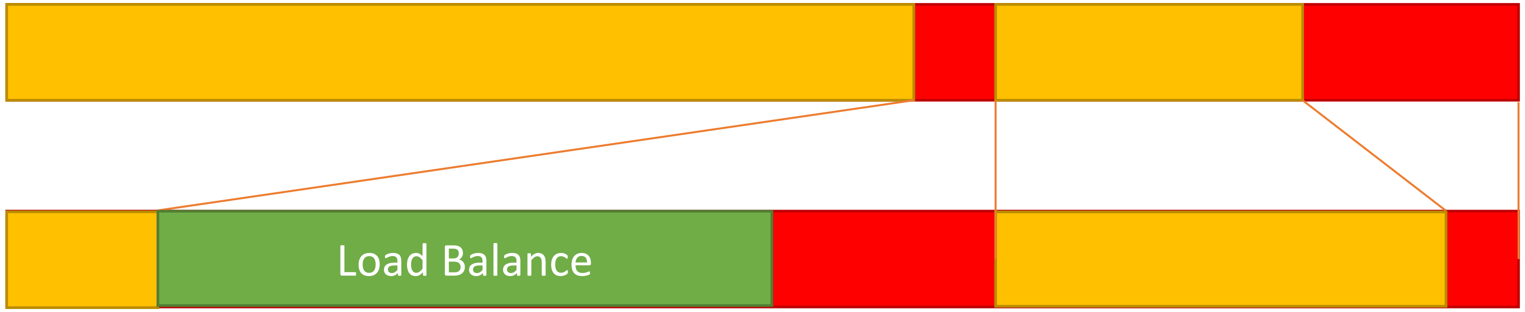
- Hierarchy of metrics developed at BSC
- Highlight issues in the parallel structure of an application
- Parallel Efficiency breaks down into
  - Load balance
  - Serialization
  - Transfer
- Computational Scaling captures impact of scaling to node-level performance

Threads per Process	1	2	4	8	12
Global Efficiency	0.94	0.64	0.40	0.19	0.13
↪ Parallel Efficiency	0.94	0.76	0.59	0.44	0.39
↪ Process Level Efficiency	0.94	0.93	0.91	0.88	0.94
↪ Load balance	0.97	0.97	0.95	0.94	0.98
↪ MPI Communication Efficiency	0.97	0.96	0.96	0.94	0.96
↪ MPI Transfer Efficiency	1.00	1.00	1.00	1.00	1.00
↪ MPI Serialisation Efficiency	0.97	0.96	0.96	0.94	0.96
↪ Thread Level Efficiency	1.00	0.83	0.68	0.56	0.45
↪ OpenMP Region Efficiency	1.00	0.98	0.98	0.97	0.92
↪ Serial Region Efficiency	1.00	0.85	0.70	0.59	0.52
↪ Computational Scaling	1.00	0.84	0.67	0.44	0.34
↪ Instruction Scaling	1.00	0.97	0.94	0.90	0.86
↪ IPC Scaling	1.00	0.96	0.88	0.74	0.67

Load balance

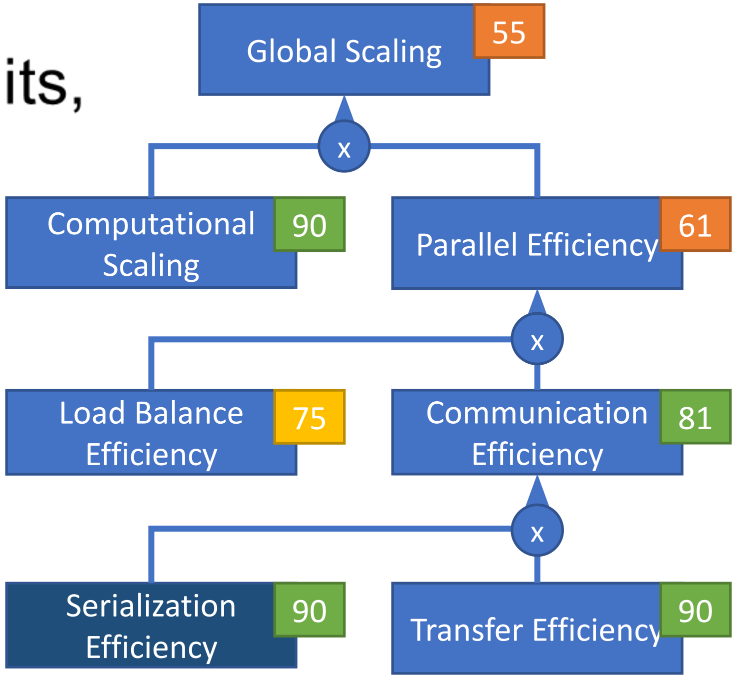
- Reflects global imbalance of work between execution units
- $LB = \frac{avg(useful\ time)}{\max(useful\ time)}$
- *Useful time*: execution time outside parallel runtimes

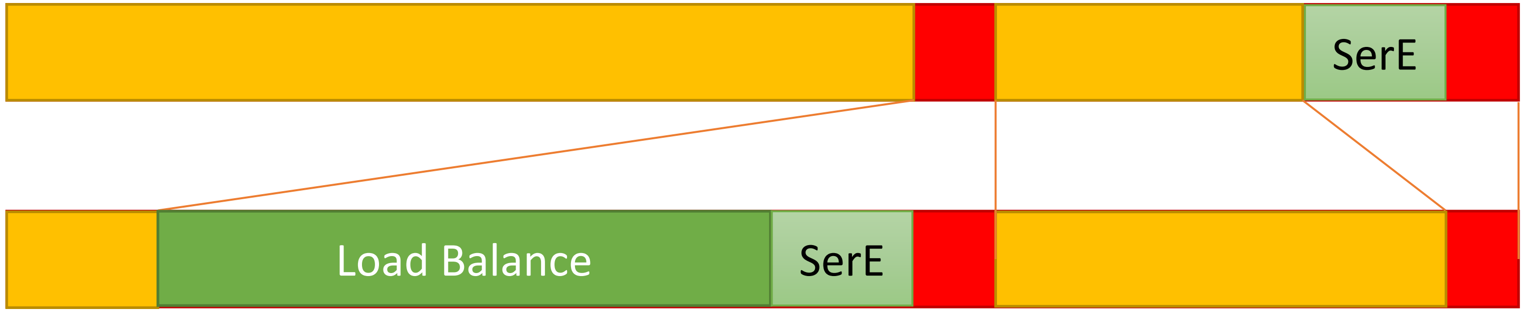




Serialization efficiency

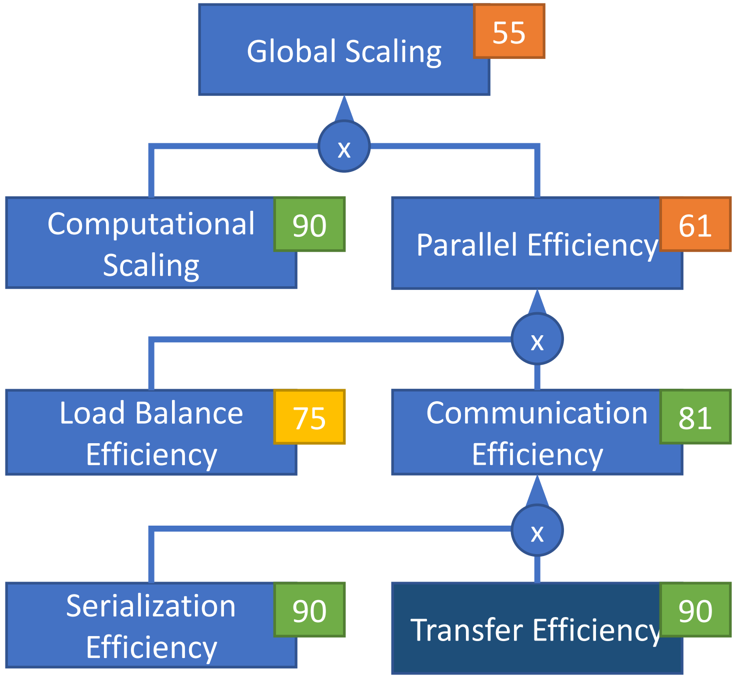
- Reflects moving imbalance of work between execution units, resp., alternating dependencies
- $SerE = \frac{max(useful\, time)}{ideal\, runtime}$
- *Ideal runtime*: execution time on an ideal machine with 0 communication cost (inf. BW / 0 lat)

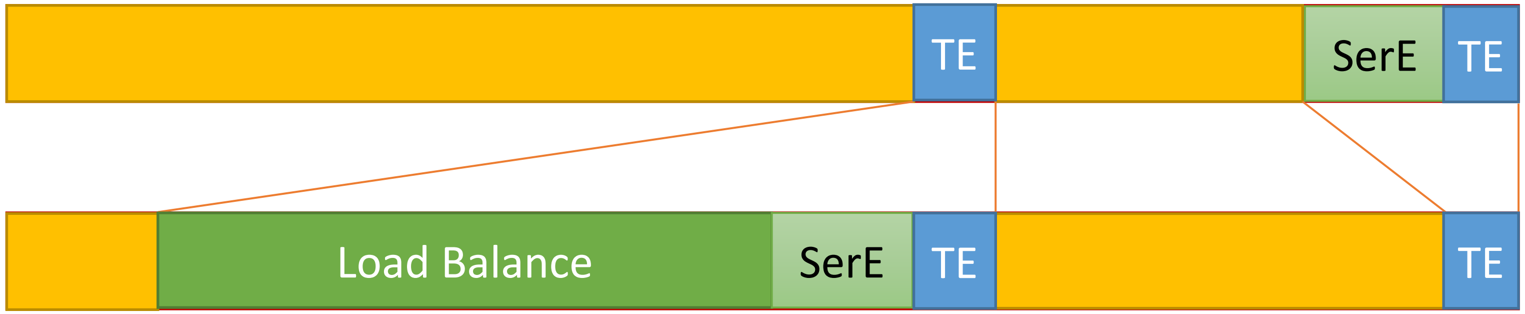




Transfer efficiency

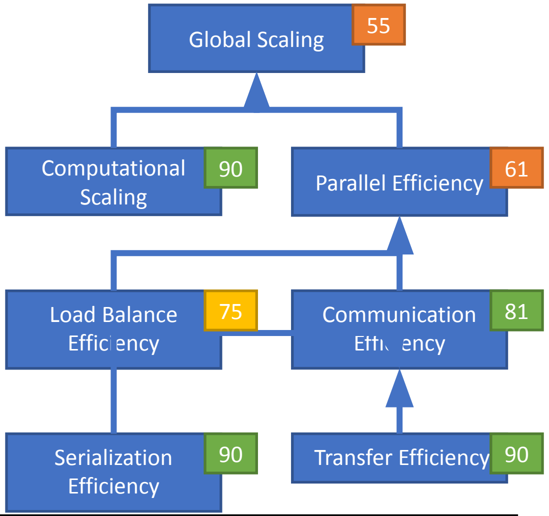
- Cost of transfer / communication / synchronization
- $TE = \frac{ideal\, runtime}{real\, runtime}$
- *Real runtime*: observed execution time





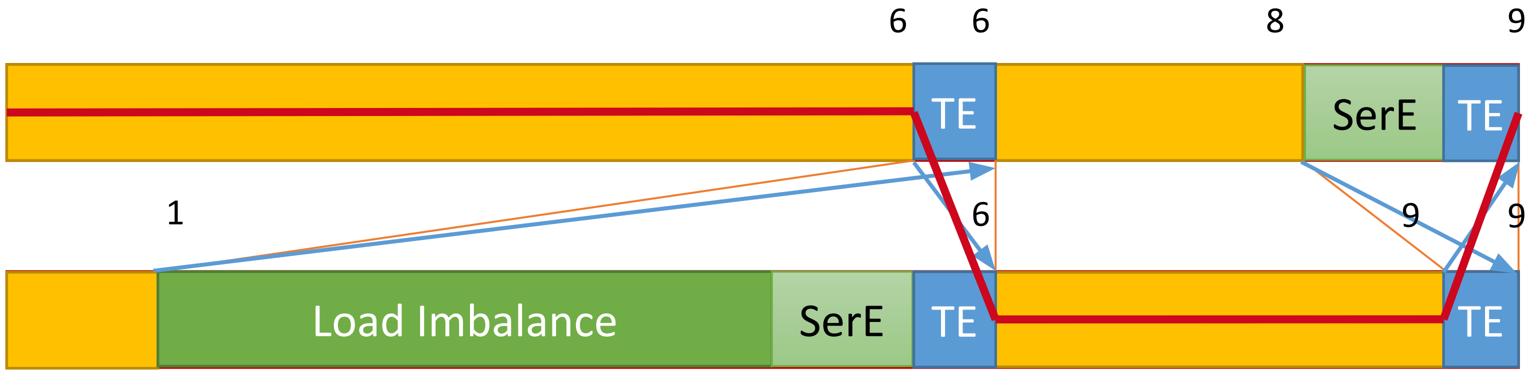
Which metrics to measure?

- *Useful time*: execution time outside parallel runtimes
  - Track execution time on each thread excluding time inside MPI / OpenMP runtimes
- *Real runtime*: observed execution time
  - Track wall clock time from start to end.
- *Ideal runtime*: execution time on an ideal machine with 0 communication cost (inf. BW / 0 lat)
  - Track *useful time* on **critical path** assumes 0 communication cost



O-T-F critical path analysis for hybrid model factors

- Forward-only analysis
  - we only need the metrics of the critical path, but not the concrete path
- Treat time metrics as Lamport clock and implement the necessary propagation of this clock (MPI communication, OpenMP synchronization)
- Relevant metrics: useful computation, time outside the OpenMP runtime
- Relevant critical paths: global, process-local, thread-local
   Formulation of MPI-specific and OpenMP-specific model factors in the paper

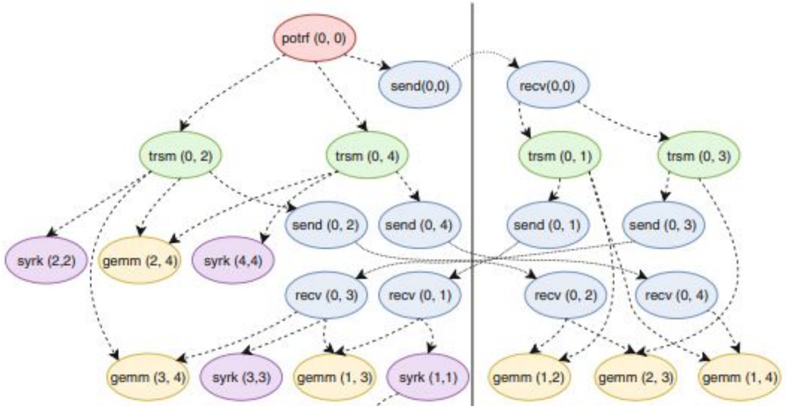


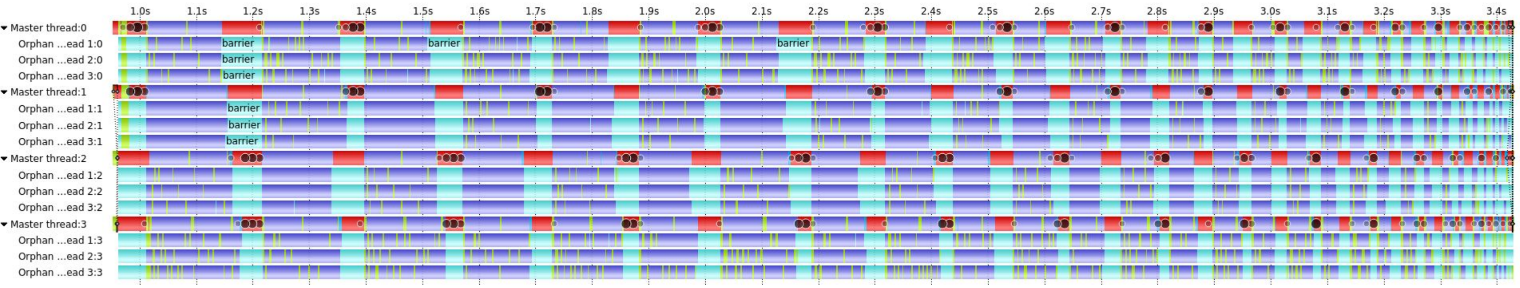
MPI Continuations (MPI-Detach @ EuroMPI'20): Non-blocking Distributed Block Cholesky Factorization

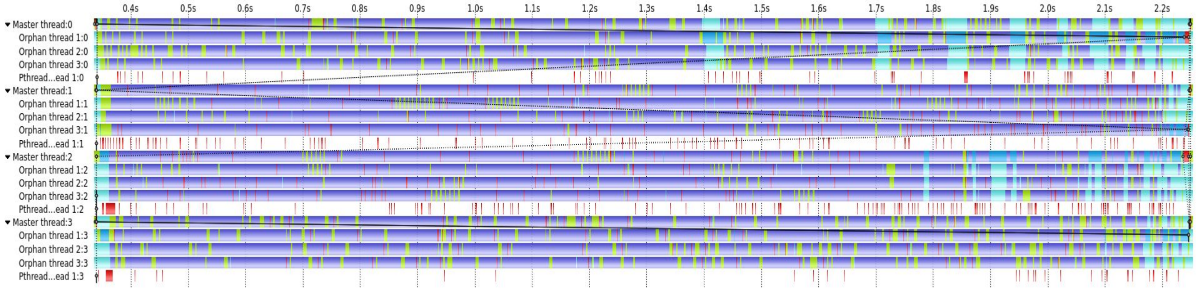
Truly asynchronous MPI:

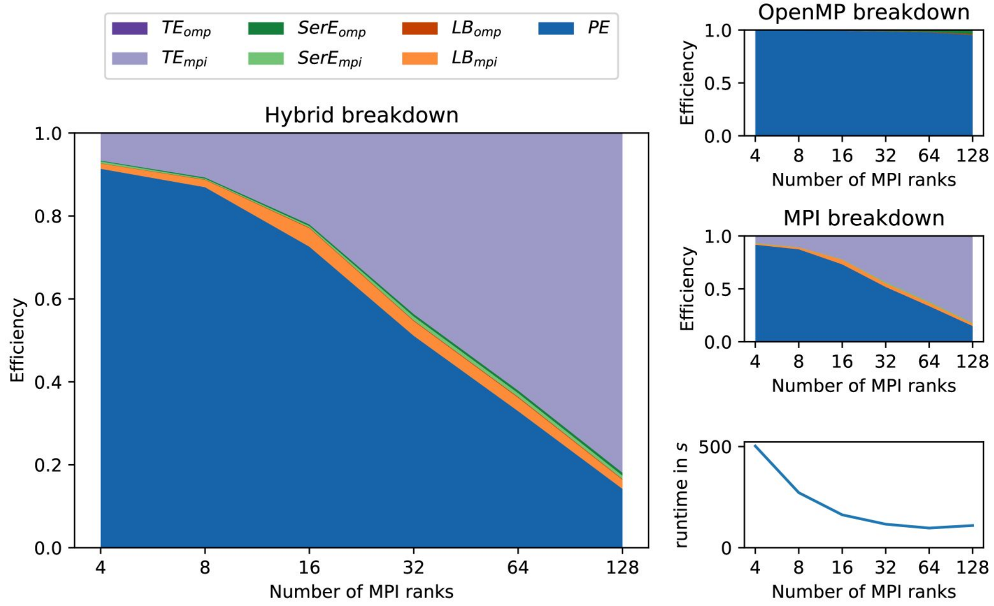
Register a callback for completion of non-blocking communication

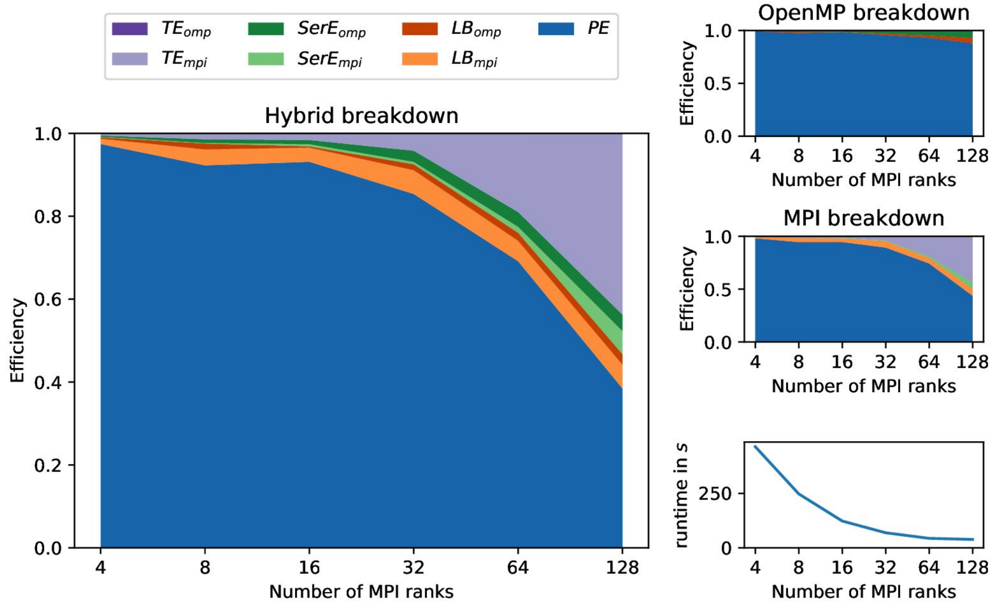
→ Release dependencies











Score-P limitation: Tracing OpenMP + std::thread

Workaround:

- Score-P in pthread mode
- OMPT tool that marks OpenMP regions as user-defined regions

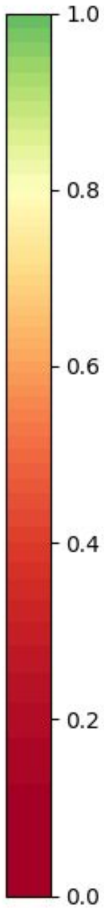
SLDG vs. NuFI – Hybrid Model Factors

SLDG (24^3 x 32^3 DOF)

	96	192	384	768	1536
Parallel Efficiency	72.4	65.0	54.2	48.8	48.3
Load Balance	80.2	74.0	69.9	65.9	65.0
Communication Efficiency	90.3	87.7	77.6	74.2	74.3
Serialisation Efficiency	91.3	88.5	79.6	75.7	75.8
Transfer Efficiency	99.0	99.1	97.5	98.0	98.0
MPI Parallel Efficiency	100.0	96.1	91.2	87.1	84.4
MPI Load Balance	100.0	99.9	99.6	97.9	96.5
MPI Communication Efficiency	100.0	96.2	91.6	89.0	87.5
MPI Serialisation Efficiency	100.0	97.0	93.4	90.6	89.0
MPI Transfer Efficiency	100.0	99.1	98.1	98.2	98.3
OMP Parallel Efficiency	72.4	67.6	59.5	56.0	57.2
OMP Load Balance	80.2	74.1	70.2	67.2	67.3
OMP Communication Efficiency	90.3	91.2	84.7	83.3	85.0
OMP Serialisation Efficiency	91.3	91.2	85.3	83.5	85.2
OMP Transfer Efficiency	99.0	100.0	99.3	99.8	99.7

NuFI (64^3 x 64^3 DOF)

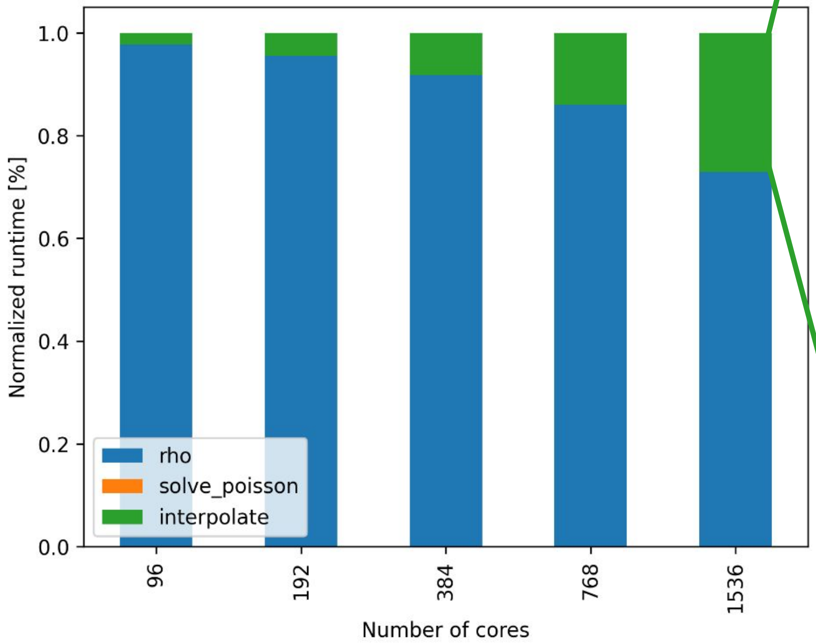
	96	192	384	768	1536
Parallel Efficiency	95.7	94.1	90.0	83.3	71.0
Load Balance	97.2	95.0	91.4	84.9	73.7
Communication Efficiency	98.5	99.0	98.5	98.1	96.3
Serialisation Efficiency	98.5	99.1	98.6	98.5	98.1
Transfer Efficiency	100.0	100.0	100.0	99.6	98.2
MPI Parallel Efficiency	99.3	99.0	99.0	99.3	98.4
MPI Load Balance	99.3	99.0	99.0	99.0	98.2
MPI Communication Efficiency	100.0	100.0	99.9	100.3	100.3
MPI Serialisation Efficiency	100.0	100.0	99.9	100.3	100.3
MPI Transfer Efficiency	100.0	100.0	100.0	100.0	100.0
OMP Parallel Efficiency	96.4	95.1	91.0	83.9	72.1
OMP Load Balance	97.9	96.0	92.3	85.8	75.0
OMP Communication Efficiency	98.5	99.1	98.6	97.8	96.1
OMP Serialisation Efficiency	98.5	99.1	98.6	98.2	97.8
OMP Transfer Efficiency	100.0	100.0	100.0	99.6	98.2



NuFI – OpenMP Load Balance

- Reasons for low OpenMP Load Balance
  1. Imbalanced workload between threads within OpenMP parallel regions OR
  2. Sequential code parts that are only executed by the main thread (Amdahl‘s law)

- NuFI timestep consists of three parts:
  1. Computation of charge density (rho)
  2. Solving Poisson equation
  3. Interpolation



Callpath profile of NuFI‘s `interpolation` (metric: time)

- 0.4929 lsmr::iteration
  - 1108.0730 mat_t::operator()
  - 48.8086 blas::daxpy
  - 687.7368 lsmr::norm
  - 20.6489 blas::dscal
  - 6.6581 lsmr::reorthogonalise
    - 910.2005 blas::ddot
    - 768.1483 blas::daxpy
    - 678.0131 lsmr::norm
    - 5.9528 blas::dscal
    - 17.9180 blas::dcopy
  - 934.3614 transposed_mat_t::operator()

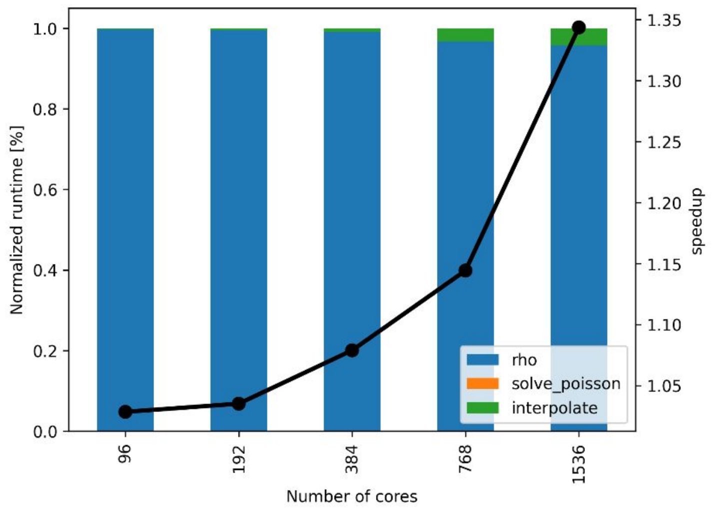
Runtime of NuFI‘s `interpolation` function on rank 0

- - MPI Rank 0
  - 32.4286 Master thread
  - 0.5144 OMP thread 1
  - 0.5144 OMP thread 2
  - 0.5144 OMP thread 3
  - 0.5144 OMP thread 4
  - 0.5139 OMP thread 5
  - 0.5140 OMP thread 6
  - 0.5140 OMP thread 7
  - 0.5144 OMP thread 8
  - 0.5140 OMP thread 9
  - 0.5140 OMP thread 10
  - 0.5140 OMP thread 11

(all MPI ranks show a similiar pattern)

NuFI – Parallelization of sequential code

1. Parallelization of transposed matrix-vector product using OpenMP for-worksharing construct
2. Replacing custom euclidean vector norm function `lsmr::norm()` with `nrm2()` from BLAS
3. Linking NuFI with threaded version of Intel MKL (for multi-threaded BLAS)



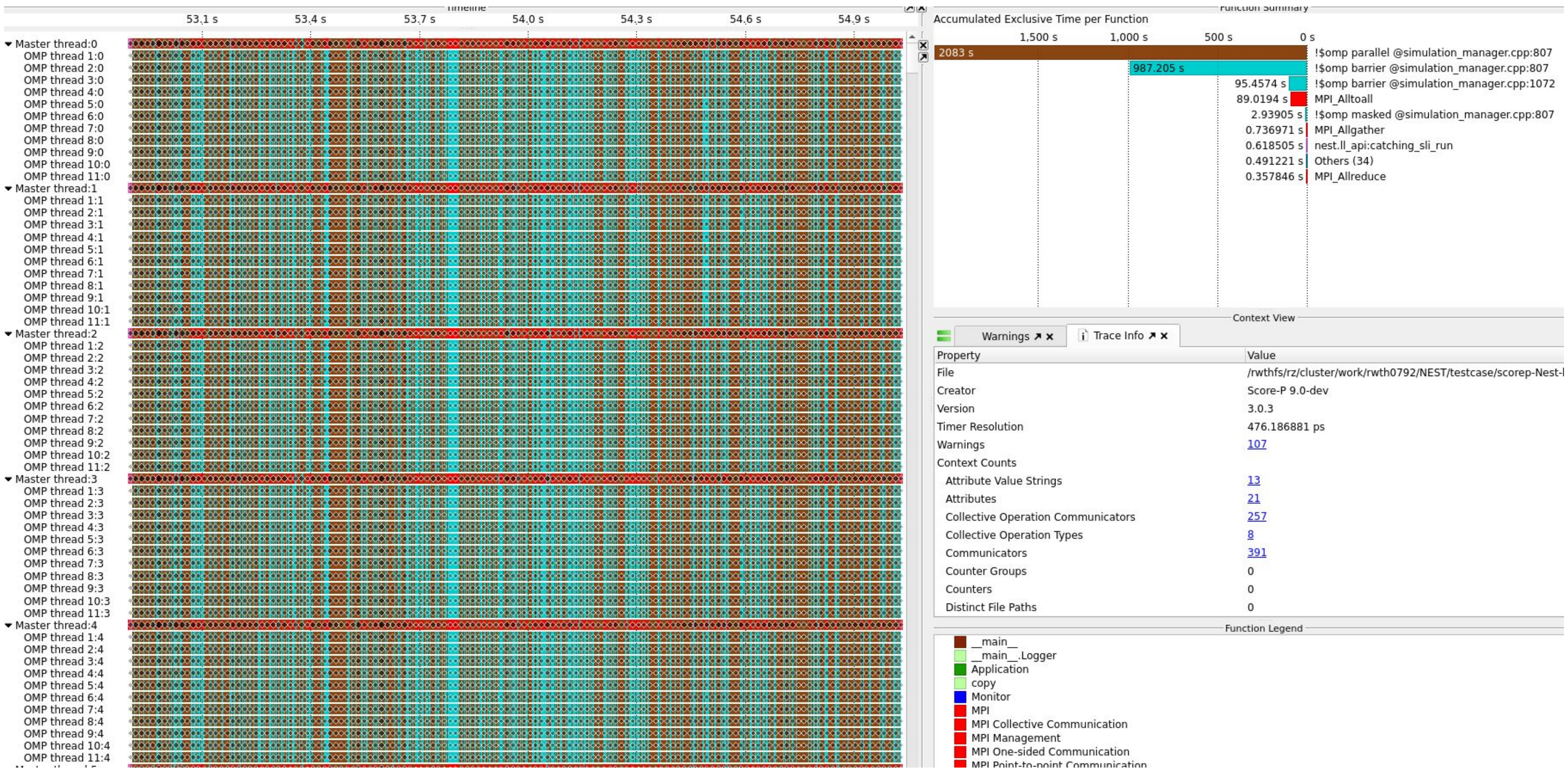
reference

	96	192	384	768	1536
Parallel Efficiency	95.7	94.1	90.0	83.3	71.0
Load Balance	97.2	95.0	91.4	84.9	73.7
Communication Efficiency	98.5	99.0	98.5	98.1	96.3
Serialisation Efficiency	98.5	99.1	98.6	98.5	98.1
Transfer Efficiency	100.0	100.0	100.0	99.6	98.2
MPI Parallel Efficiency	99.3	99.0	99.0	99.3	98.4
MPI Load Balance	99.3	99.0	99.0	99.0	98.2
MPI Communication Efficiency	100.0	100.0	99.9	100.3	100.3
MPI Serialisation Efficiency	100.0	100.0	99.9	100.3	100.3
MPI Transfer Efficiency	100.0	100.0	100.0	100.0	100.0
OMP Parallel Efficiency	96.4	95.1	91.0	83.9	72.1
OMP Load Balance	97.9	96.0	92.3	85.8	75.0
OMP Communication Efficiency	98.5	99.1	98.6	97.8	96.1
OMP Serialisation Efficiency	98.5	99.1	98.6	98.2	97.8
OMP Transfer Efficiency	100.0	100.0	100.0	99.6	98.2

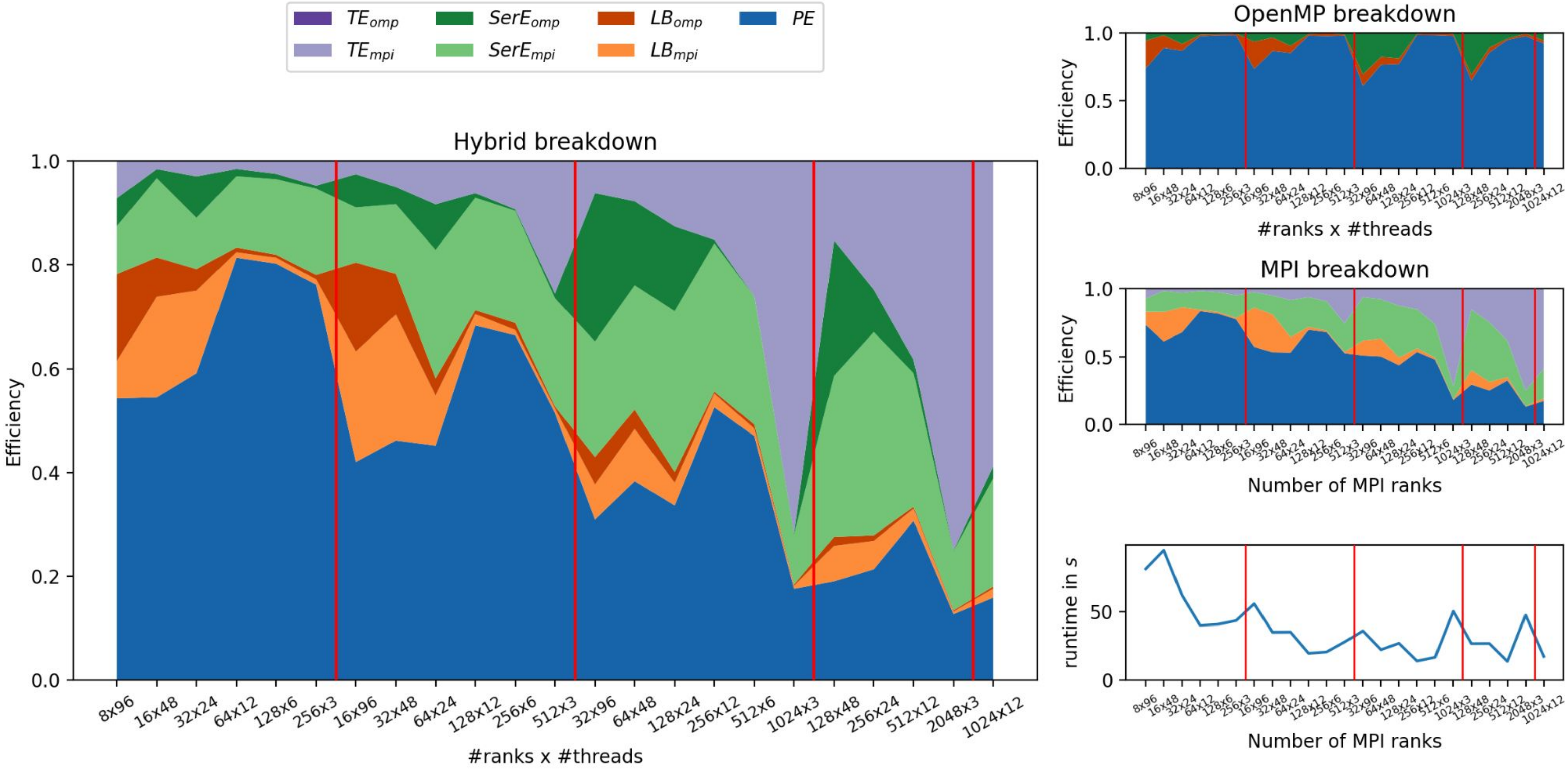
optimized

	96	192	384	768	1536
Parallel Efficiency	98.5	97.5	97.2	95.3	95.0
Load Balance	98.7	97.6	97.4	95.8	95.5
Communication Efficiency	99.8	99.9	99.7	99.5	99.4
Serialisation Efficiency	100.0	100.0	99.8	100.0	100.2
Transfer Efficiency	99.9	99.9	99.9	99.5	99.2
MPI Parallel Efficiency	98.8	98.0	97.7	96.2	96.4
MPI Load Balance	98.8	98.0	97.8	96.2	96.1
MPI Communication Efficiency	100.0	100.0	99.9	100.0	100.3
MPI Serialisation Efficiency	100.0	100.0	99.9	100.0	100.3
MPI Transfer Efficiency	100.0	100.0	100.0	100.0	100.0
OMP Parallel Efficiency	99.6	99.4	99.5	99.0	98.5
OMP Load Balance	99.8	99.6	99.6	99.5	99.4
OMP Communication Efficiency	99.8	99.8	99.8	99.5	99.1
OMP Serialisation Efficiency	100.0	99.9	100.0	100.0	99.9
OMP Transfer Efficiency	99.9	99.9	99.9	99.5	99.2

Nest-Simulator: Trace of 12 threads x 128 procs



Exploring MPI/OpenMP scalability of a hybrid application



Comparing results from multiple tools

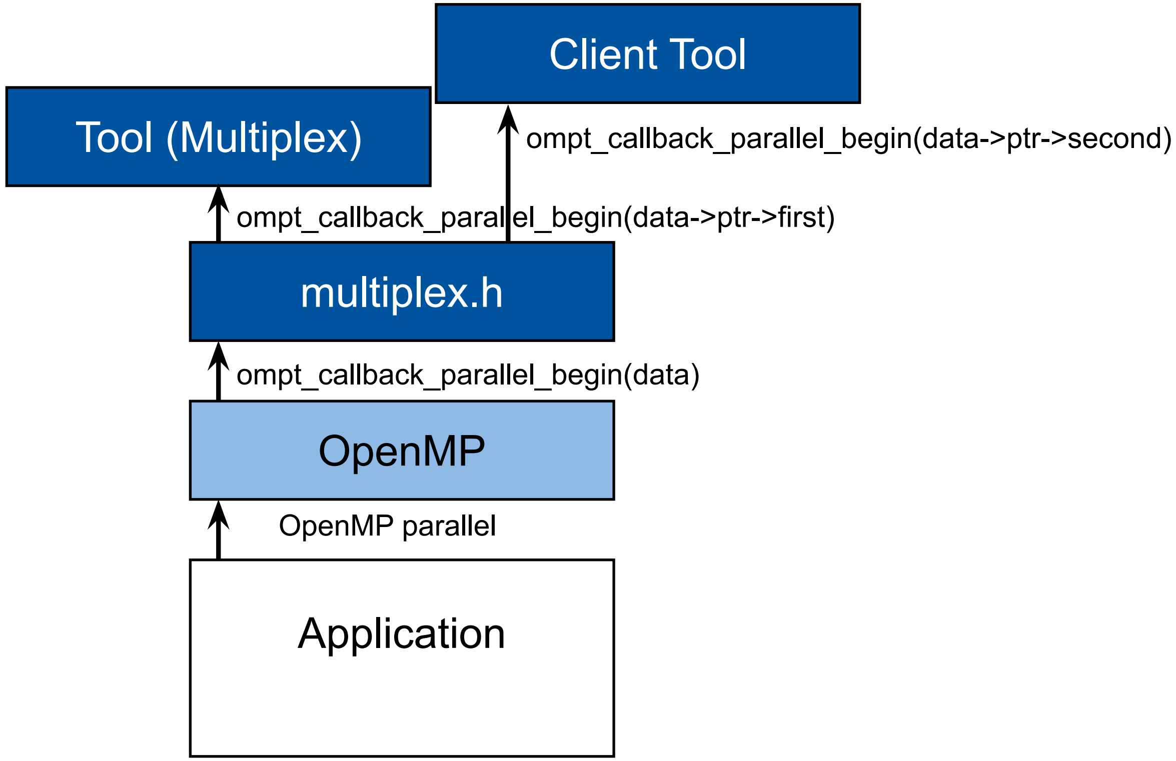
Apple-to-apple comparison

- Running both tools at the same time is crucial for meaningful results
- PnMPI: stack MPI interceptors of different tools
- OMPT-multiplex: chain OMPT tools

Stacking OMPT tools: ompt-multiplex.h

Workshop on Parallel Tools for HPC 2019

- Shipped with LLVM:
  - openmp/tools/multiplex/ompt-multiplex.h
- Tool defines a name for `CLIENT_TOOL_LIBRARIES_VAR` e.g.: `"SCOREP_TOOL_LIBRARIES"` and includes the header
- First tool is loaded with `OMP_TOOL_LIBRARIES` variable, second tool is loaded with `SCOREP_TOOL_LIBRARIES`
- Tool can optimize the allocation of data structures (default: multiplex allocates pair for each tool data)



Comparing results from OTF-CPT and Score-P/Cube

```
----------------POP metrics----------------
Parallel Efficiency:                    0.757081
  Load Balance:                         0.931473
  Communication Efficiency:             0.812778
    Serialisation Efficiency:           0.885207
    Transfer Efficiency:                0.918179
  MPI Parallel Efficiency:              0.779985
    MPI Load Balance:                   0.956048
    MPI Communication Efficiency:       0.815843
      MPI Serialisation Efficiency:     0.885975
      MPI Transfer Efficiency:          0.920842
  OMP Parallel Efficiency:              0.970635
    OMP Load Balance:                   0.974295
    OMP Communication Efficiency:       0.996243
      OMP Serialisation Efficiency:     0.999133
      OMP Transfer Efficiency:          0.997108
```