TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Denis Hünich[1], Ronny Tschüter[2], Bert Wesarg[1,3]

[1] Information Services and High Performance Computing (ZIH), CIDS, TU Dresden
[2] Institute of Software Methods for Product Virtualization, German Aerospace Center (DLR)
[3] GWT-TUD GmbH

Exploring Multi-threaded Communication Behavior of a Large-Scale CFD Solver with Vampir

15th International Parallel Tools Workshop 2024, Dresden, 2024 -09-19

CIDS

Outline

Motivation

Distributed event loading

Evaluation

Racy message order

Conclusion & Outlook

Motivation

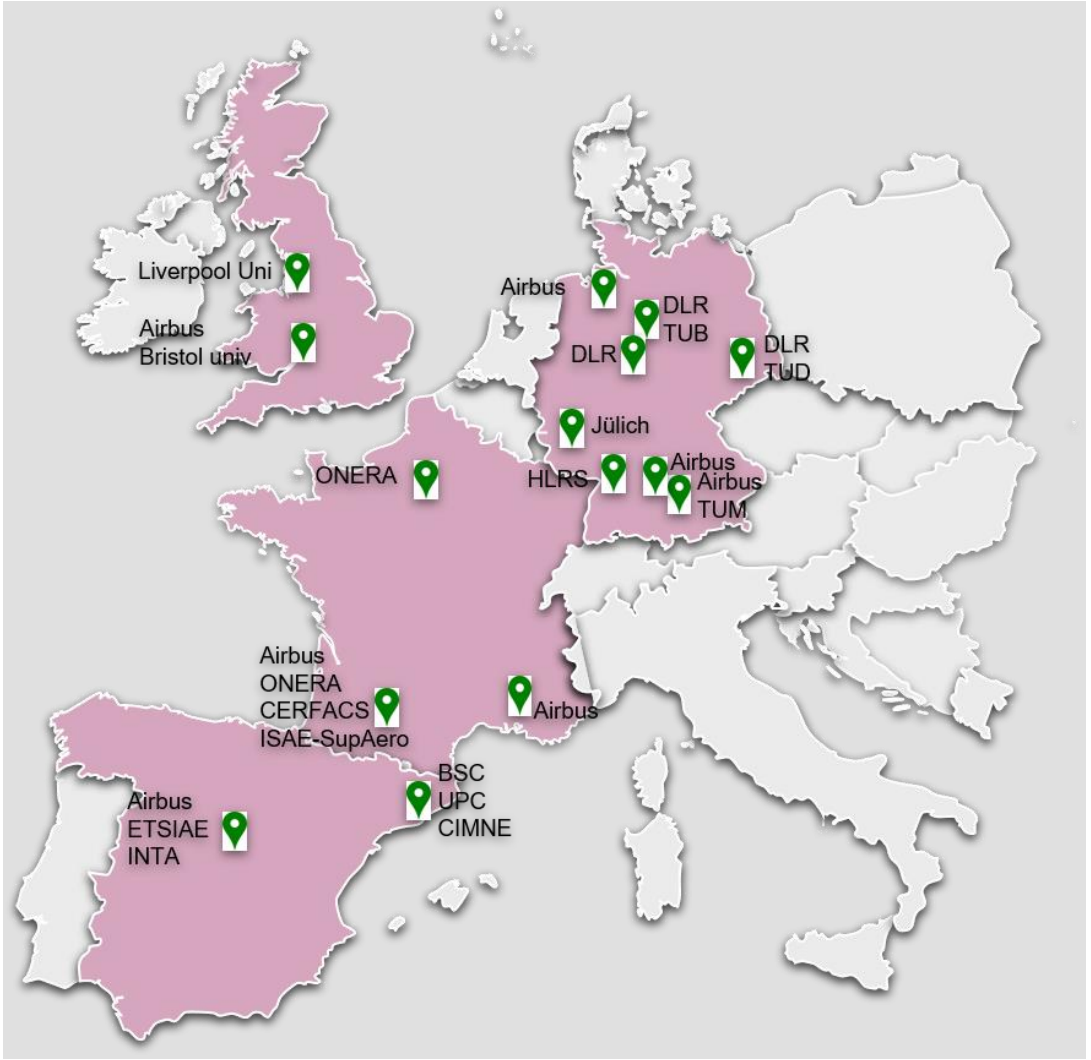
Performance Analysis of CODA

CODA

- Flow solver written by
  - ONERA
  - DLR
  - Airbus
  - Many other European partners

Coding

- User scenarios written in Python, solver core written in C++
- Hybrid parallelization, computation & communication overlap
- Spliss as linear solver



Courtesy: Immo Huismann

Performance Analysis of CODA: Current Status

- MPI provides different level of thread support
- CODA uses MPI + OpenMP threads with `MPI_THREAD_MULTIPLE`
  - Limited support in recent versions of performance analysis tools
- Initial performance analyses restricted to `MPI_THREAD_FUNNELED`
- Valuable first insights into applications performance, however
  - Varying the MPI level of thread support changes the application behavior

R. Tschüter, I. Huismann, B. Wesarg and M. Knespel, "Performance Analysis of the CFD Solver CODA - Harnessing Synergies between Application and Performance Tools Developers," *2022 IEEE/ACM Workshop on Programming and Performance Visualization Tools (ProTools)*

Performance Analysis of CODA: Current Status

- Limitations w.r.t. handling the `MPI_THREAD_MULTIPLE` level of thread support
  - Message matching does not work
  - Excessive increase in time to load trace data
  - Severely restricted insight into communication patterns

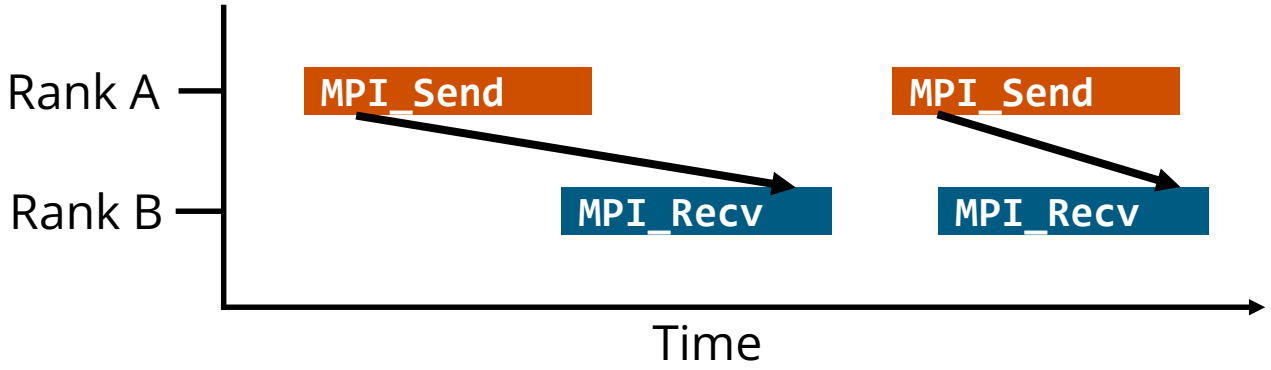
Distributed event loading

MPI point-to-point message matching

Need pairwise event matching to derive upper bound of message duration

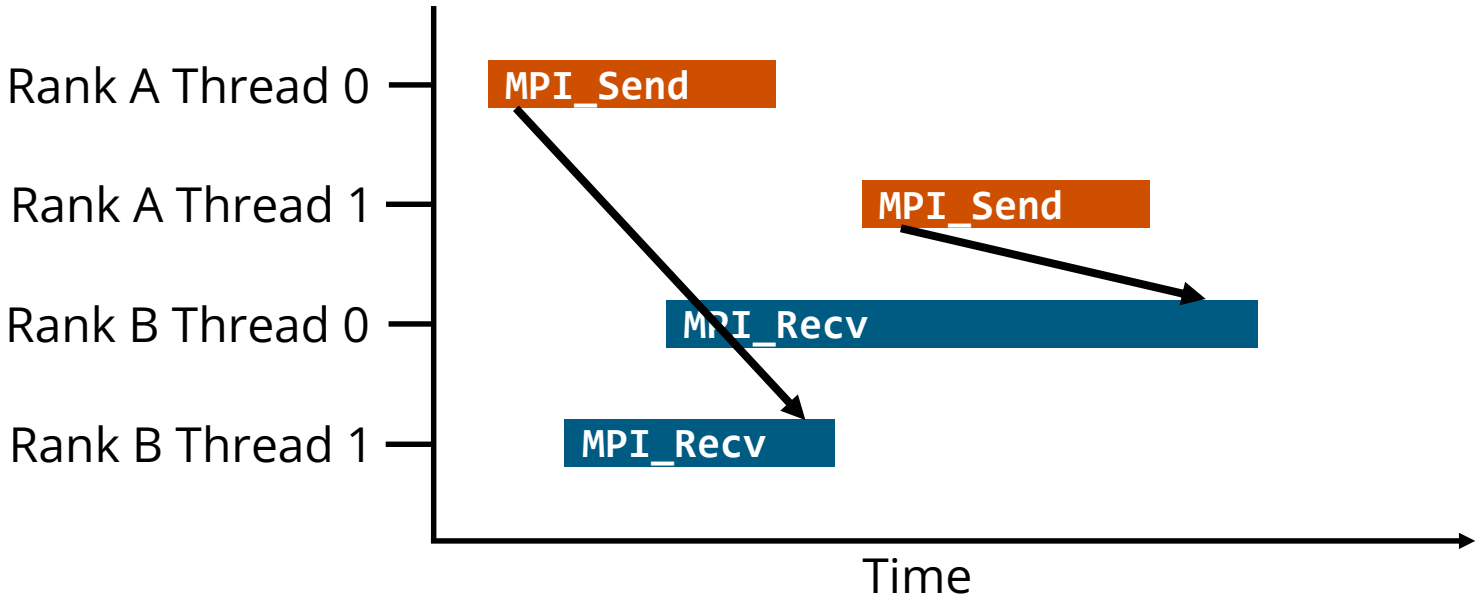
MPI messages consists of a *send* and *receive* event in the same *envelop*

Timestamp of sends and receives determine message matching order



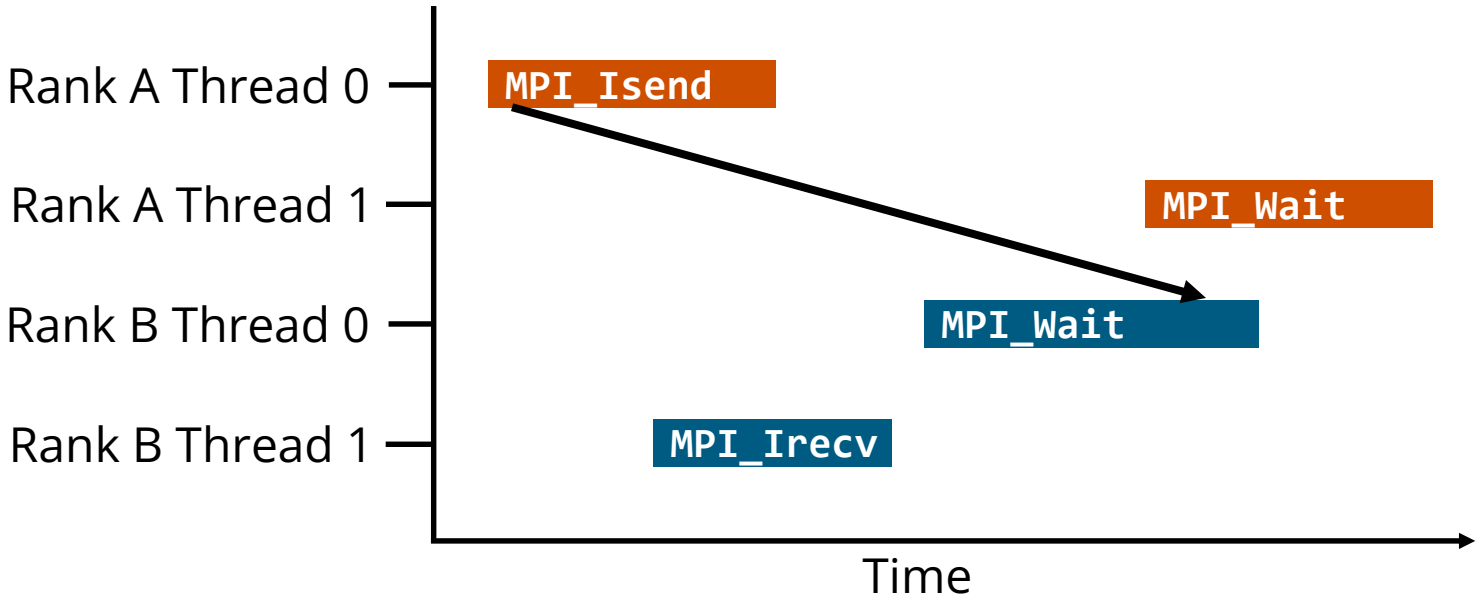
MPI point-to-point message matching
MPI_THREAD_MULTIPLE

Must obey message order across all threads of a rank

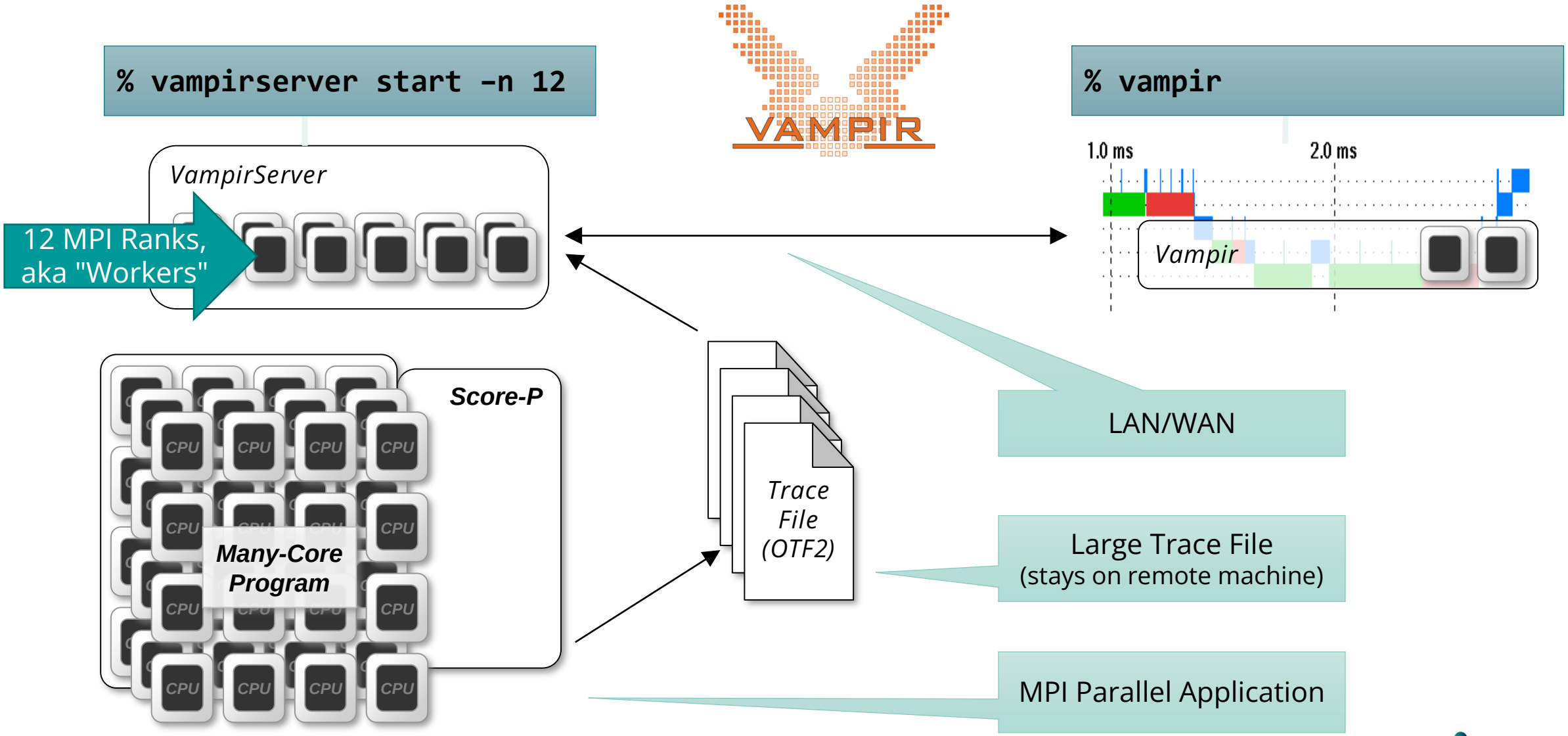


MPI point-to-point message matching

MPI_THREAD_MULTIPLE & non-blocking



Vampir


% vampirserver start -n 12
VAMPIR
% vampir
VampirServer
12 MPI Ranks,
aka "Workers"
1.0 ms
2.0 ms
Vampir
Score-P
CPU
Many-Core
Program
Trace
File
(OTF2)
LAN/WAN
Large Trace File
(stays on remote machine)
MPI Parallel Application

Vampir
Distributed event processing

Storing events for one location (thread, accelerator stream, …) in one worker

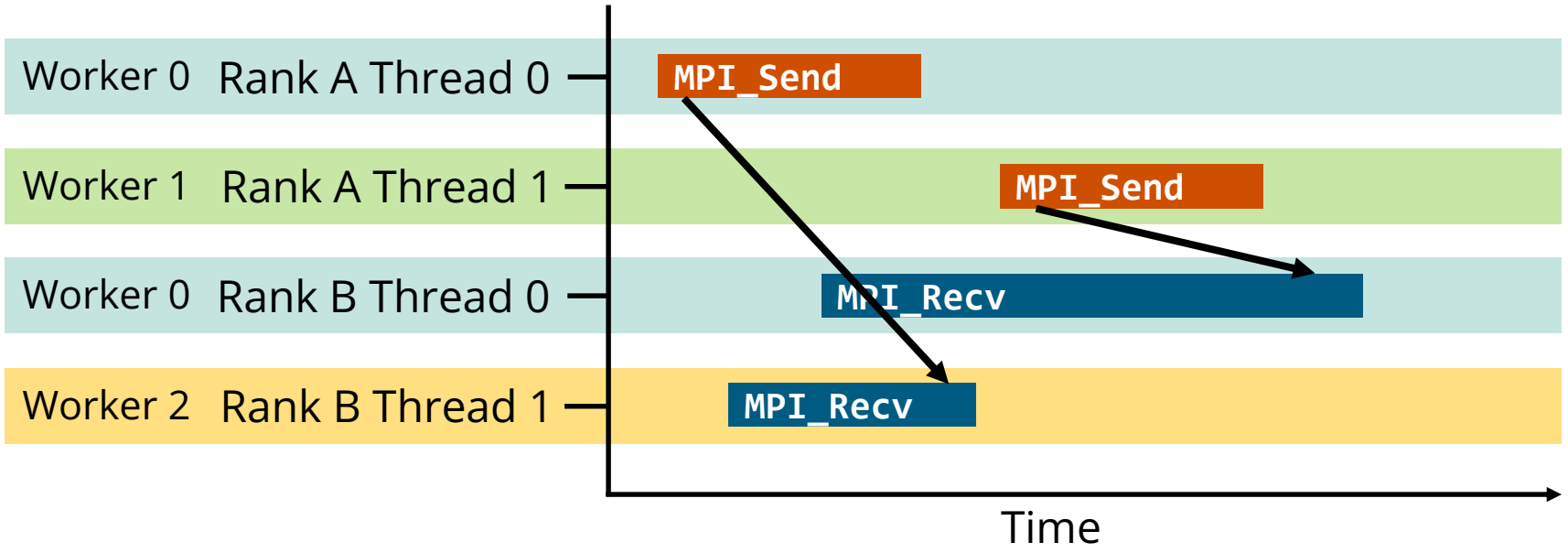
- Open each file at most once

Randomly distribute locations to workers

- Balances memory and compute load on average

Vampir

Distributed event processing and **MPI_THREAD_MULTIPLE**



Distribution destroys event order across threads of one rank

- Need to process all ordered events of a process by one worker

Vampir

Constrains:

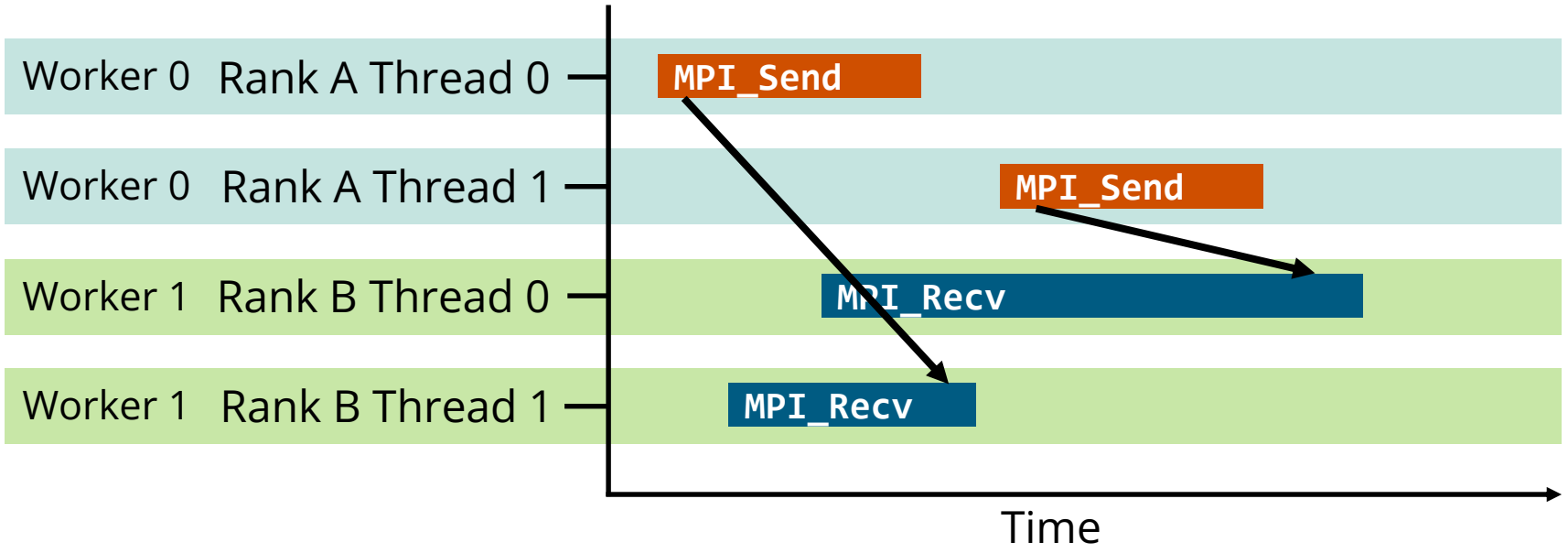
— Do not interfere with the visualization (balance memory and compute)

— Avoid opening event files by multiple workers

Ideas:

1. Keep distribution and send MPI events around
   – Implementing a parallel discrete event simulation
2. Keep distribution, but let workers read all necessary event files
   – Reading the same event file by multiple workers
3. Change distribution at load time only
   – Restores distribution via message across workers

Vampir
Load time distribution



➢Reduces load balance at load time

Evaluation

Overhead Measurement

Measurement environment:

— CIDS/ZIH HPC-Cluster Barnard with 2 x Intel Xeon Platinum 8470 (52 cores) @ 2.00 GHz per node

— Compiler: GCC 13.2

Measurement parameter:

— Vampir versions:
  – Current version of Vampir (master)
  – Vampir with `MPI_THREAD_MULTIPLE` support turned on (*thread_multiple_on*)
  – Vampir with `MPI_THREAD_MULTIPLE` support turned off (*thread_multiple_off*)

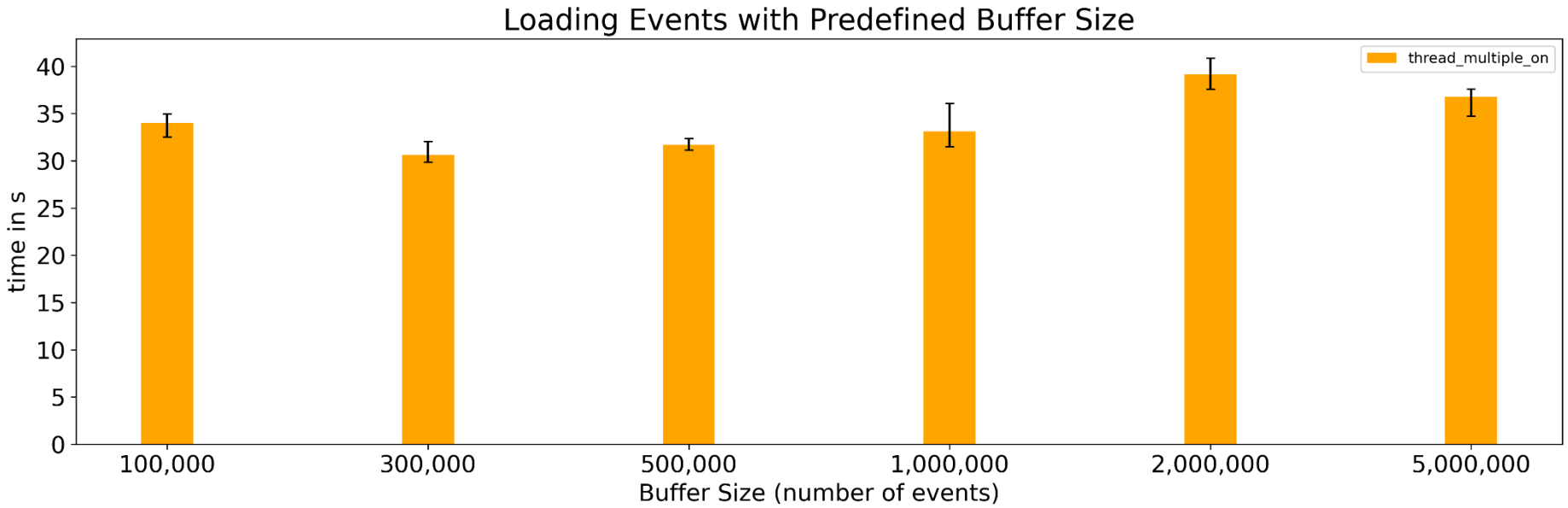
— Sample size is 5

— CODA Trace:
  – Application that used `MPI_THREAD_MULTIPLE`
  – 64 ranks with 4 threads each
  – 2.8 GiB

Overhead Measurement

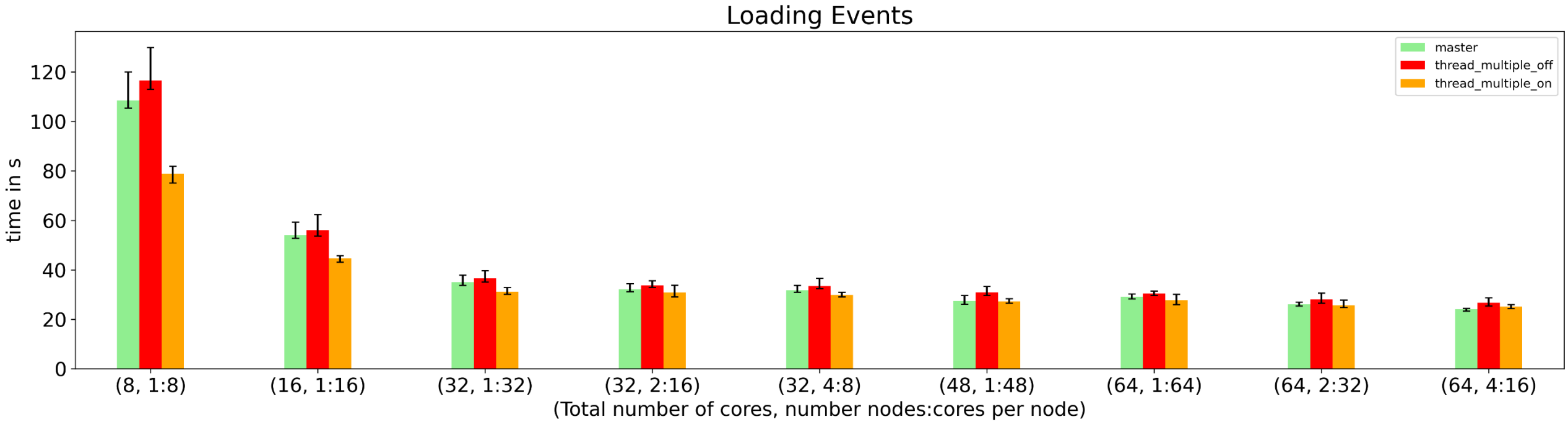
`MPI_THREAD_MULTIPLE` support turned on and 32 MPI ranks on one node

Different buffer sizes to distribute events for balanced worker/location ratio

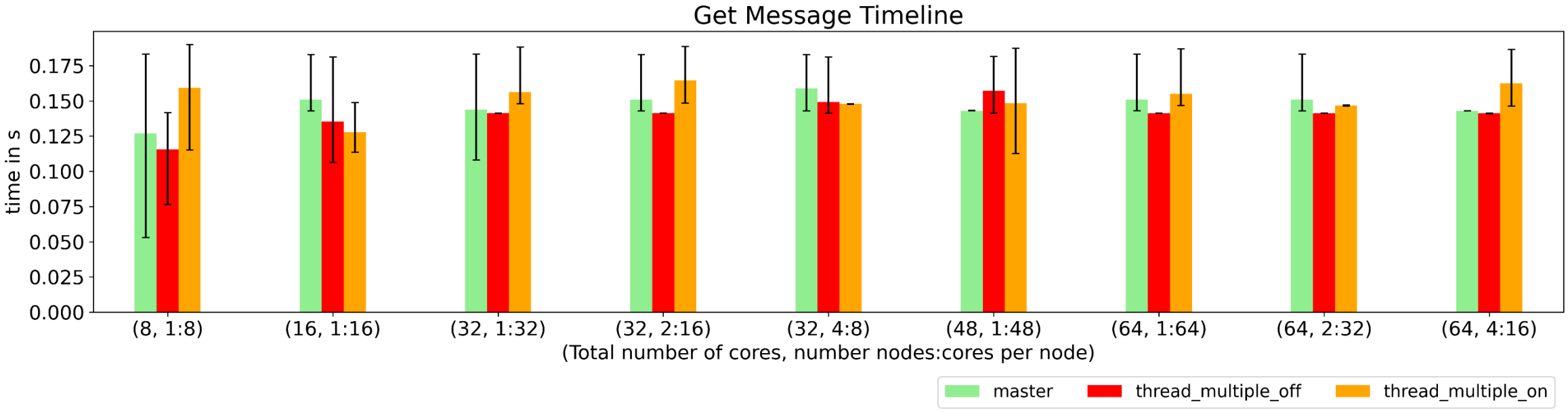


Overhead Measurement

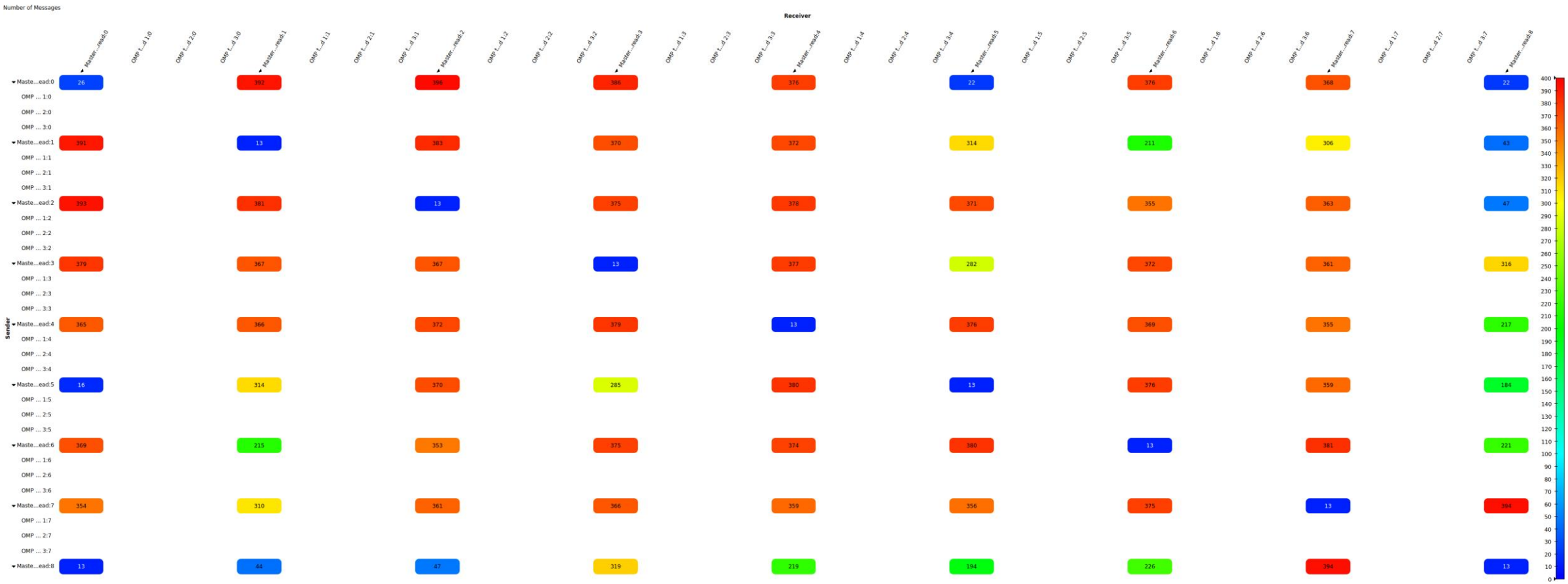
Buffer size of 300,000 used for *thread_multiple_on*



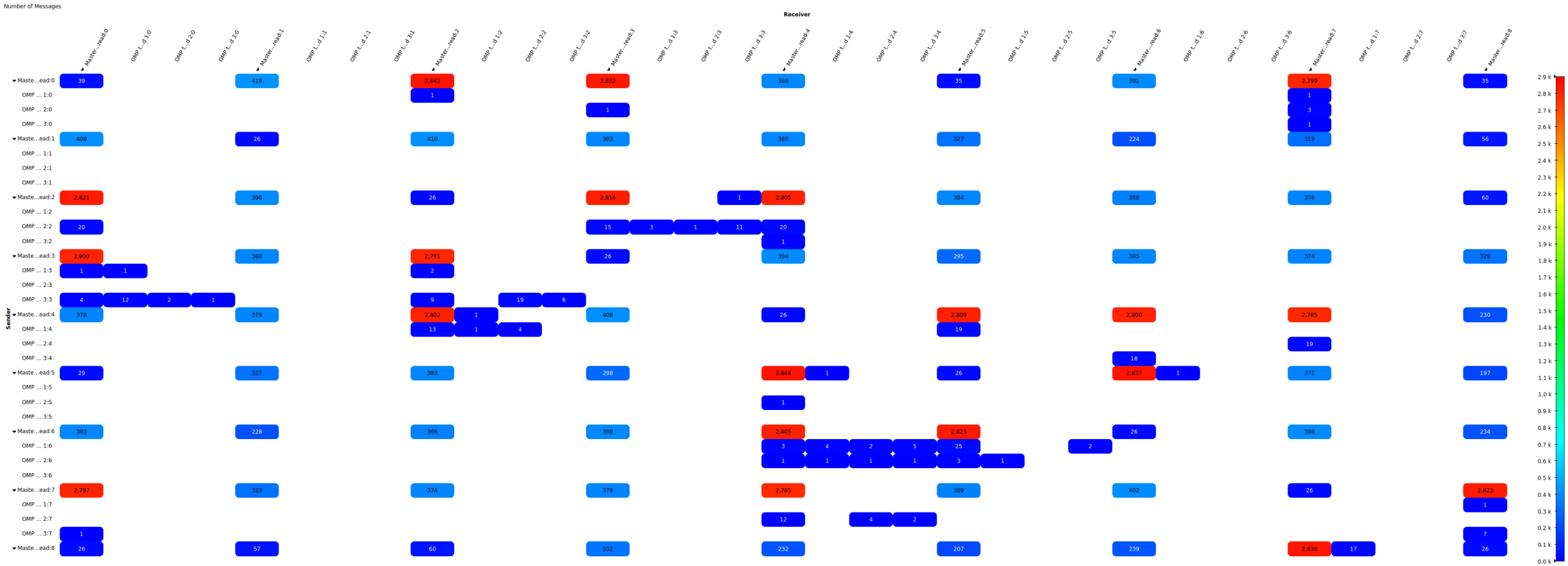
Overhead Measurement



Vampir Communication Matrix (Master)



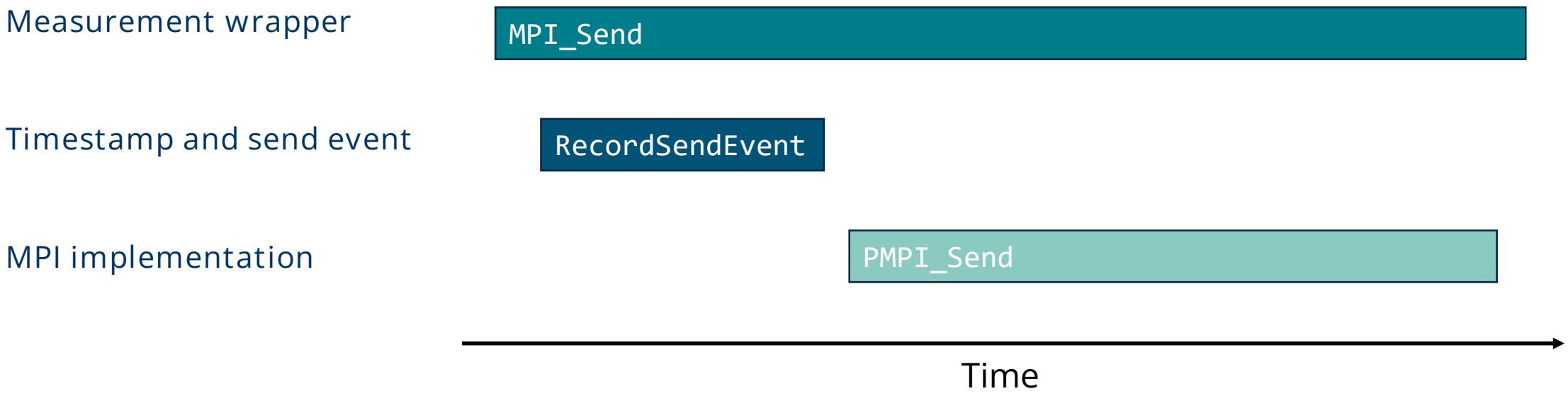
Vampir Communication Matrix (*thread_multiple_on*)



Racy message order

Determining message order

Timestamps of MPI calls determine message order



Level of thread support in

Level of thread support	Ruling	Serialized on MPI call level
`MPI_THREAD_SINGLE`	Only one thread will execute.	Yes
`MPI_THREAD_FUNNELED`	Only the thread that called `MPI_Init_thread` will make MPI calls.	Yes
`MPI_THREAD_SERIALIZED`	Only one thread will make MPI library calls at one time.	Yes
`MPI_THREAD_MULTIPLE`	Multiple threads may call MPI at once with no restrictions.	No

➢Serializations happens inside MPI implementation

Racy message order

Avoid:

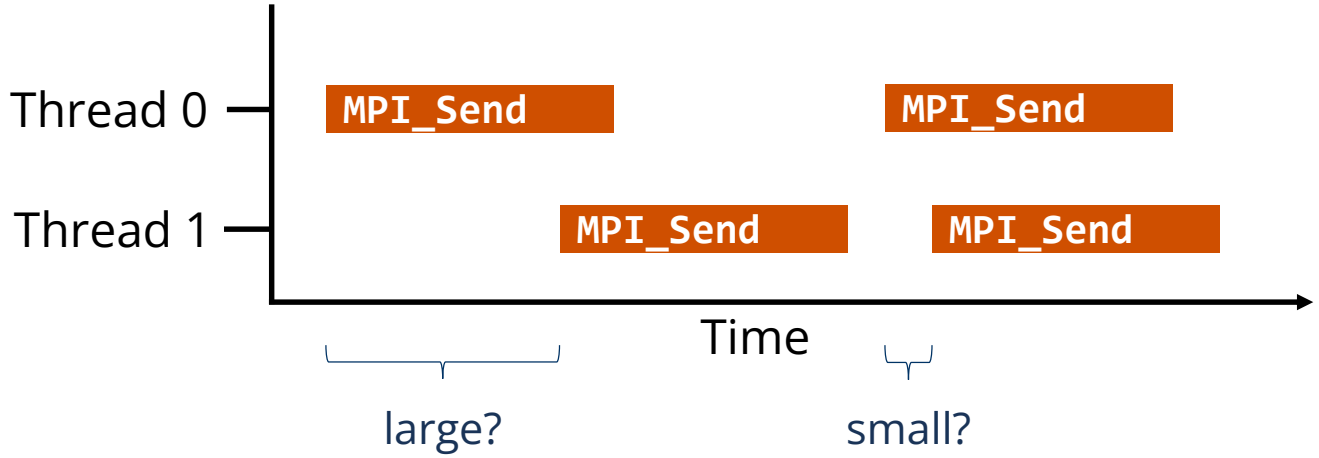
- Force serialization in the measurement
- Change application to use different tags (`MPI_TAG_UB`)
- Needs help of MPI implementation

Detect:

- Are there any two messages with the same envelop on different threads at all?
- Are there any two MPI post calls with the same envelop on different threads which overlap?
- Is the difference between two MPI post calls with the same envelop on different threads "small" enough to permit a race?

Racy message order

Inter Message Duration



Global minimum value presented to user

User needs to interpret value

Conclusion & Outlook

Conclusion & Outlook

Conclusion

Vampir able to visualize applications with any level of thread support

Acceptable overhead when loading

No overhead after loading

Outlook

Overhead only needed if level of thread support exceeds `MPI_THREAD_FUNNELED`

- ➢ Measurement annotates trace with level of thread support

Improve racy message order by checking MPI call overlap

Opens new analysis possibilities for other paradigms