FROM RESEARCH TO INDUSTRY

cea

www.cea.fr

MPC: AN HPC OPEN SOURCE ECOSYSTEM

Multi-Processor Computing

Marc PERACHE
CEA, DAM, DIF, F-91297 Arpajon, France

NOVEMBER 2016

Context

Activity overview

- Runtime system and software stack for HPC
- Available software
  - MPC framework
  - MALP (performance analysis tool)
  - JCHRONOSS (job scheduler for test suite on production machines)
  - Wi4MPI (MPI abstraction)
- Website for team work: http://hpcframework.com
- Paratools is our main partner on all these activities

MPC framework

- Unified parallel runtime for clusters of NUMA machines
  - Idea: one process per node, compute units exploited by user-level threads
- Integration with other HPC components
  - Parallel memory allocator, compilers, debuggers, topology tool…
- Research and production ready MPI and OpenMP implementation
- Homepage: http://mpc.hpcframework.com
- CeCILL-C license

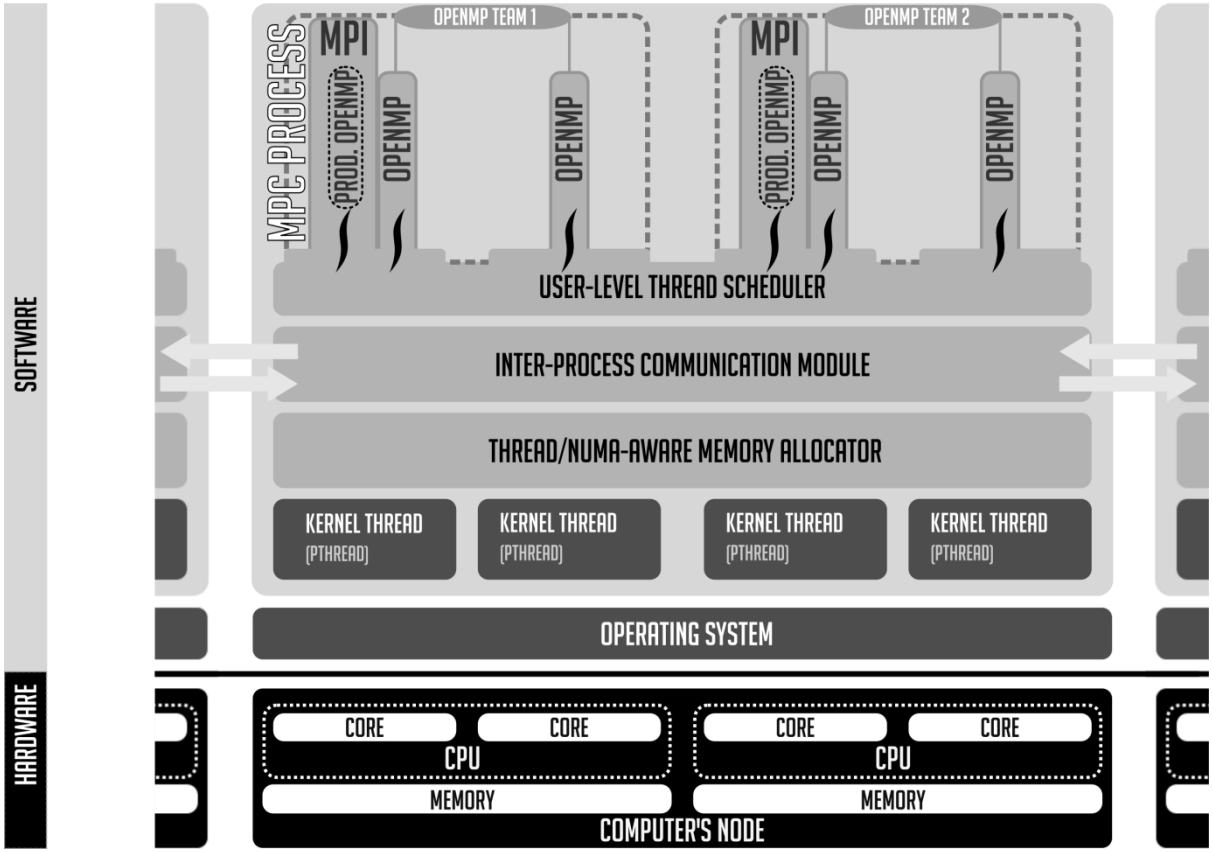
MPC Execution Model

MPI/OpenMP integration

- Automatic MPI task placement on the node
- Automatic OpenMP thread placement
  - Topology inheritance

Example

- Node with 2 CPUs
- 2 cores per CPU
- 2 MPI tasks per node
- Default: 2 OpenMP threads per team



MPC - UNIFIED USER-LEVEL THREAD SCHEDULER

User-Level Thread Scheduler

Why user level threads ?

- Easier development
  - compare to kernel development
- Optimizations:
  - Keep only useful HPC functionalities
  - Less or no system calls (no system calls if no signal support)
- Portable:
  - OS independent
- Drawbacks:
  - Hard to debug: need specific debugger support (collaboration with Allinea for user-level thread support)
  - Architecture dependencies:
    - Optimized ASM context/switch, spinlocks, …
    - Topology detection and binding (easier thanks to HWLOC)

MPC User-Level Thread Scheduler

MPC User-Level thread scheduler features

- MxN thread scheduler
  - M user-level threads > N kernel threads
- Optimizations
  - ASM context switches
  - Link with MPC memory allocator to ensure data locality
- Topological binding:
  - Static a priori distribution (MPI scatter/OpenMP compact)
  - On demand migration (link with memory allocator)
- MPI optimized scheduler
  - Internal dedicated task engine for message progression
- Optimized busy waiting policy
  - Use the thread scheduler to dynamically adapt busy waiting policy according to node workload
  - Busy waiting delegation to thread scheduler (e.g. “smart” mwait/futex)
- Modular approach in order to evaluate new scheduling policies
- No preemption

MPC - MPI IMPLEMENTATION

MPC – MPI: Purpose and Supported Features

Goals

- Smooth integration with multithreaded model
- Low memory footprint
- Deal with unbalanced workload
- Modular architecture

Supported Features

- Fully MPI 1.3 compliant (3.1 soon)
- Handle up to MPI_THREAD_MULTIPLE level (max level)
- MPI I/O
- Non-blocking collectives
- Neighborhood collectives

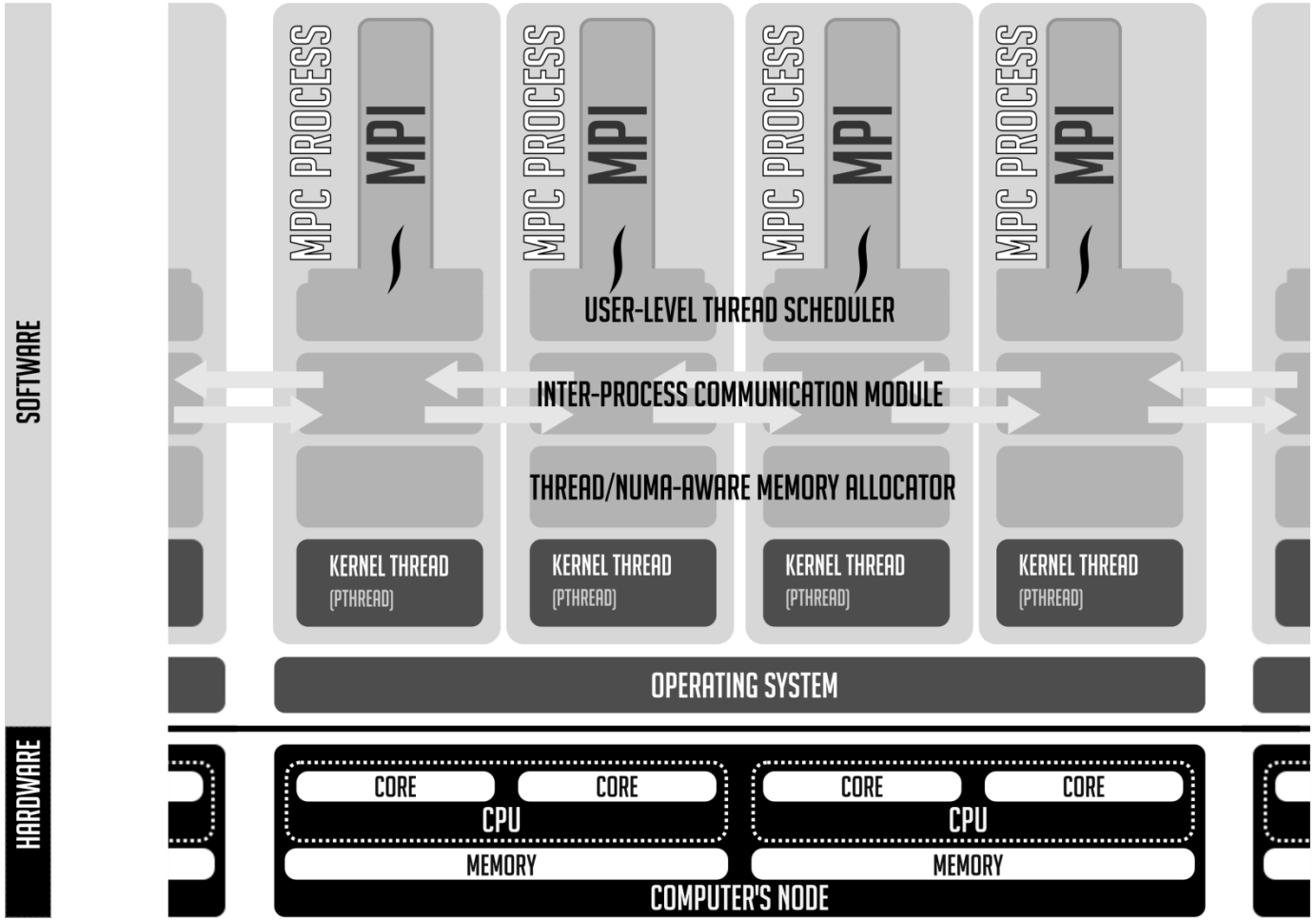
Inter-node communications

- TCP, InfiniBand & Portals4

Tested up to 80,000 cores with various HPC codes

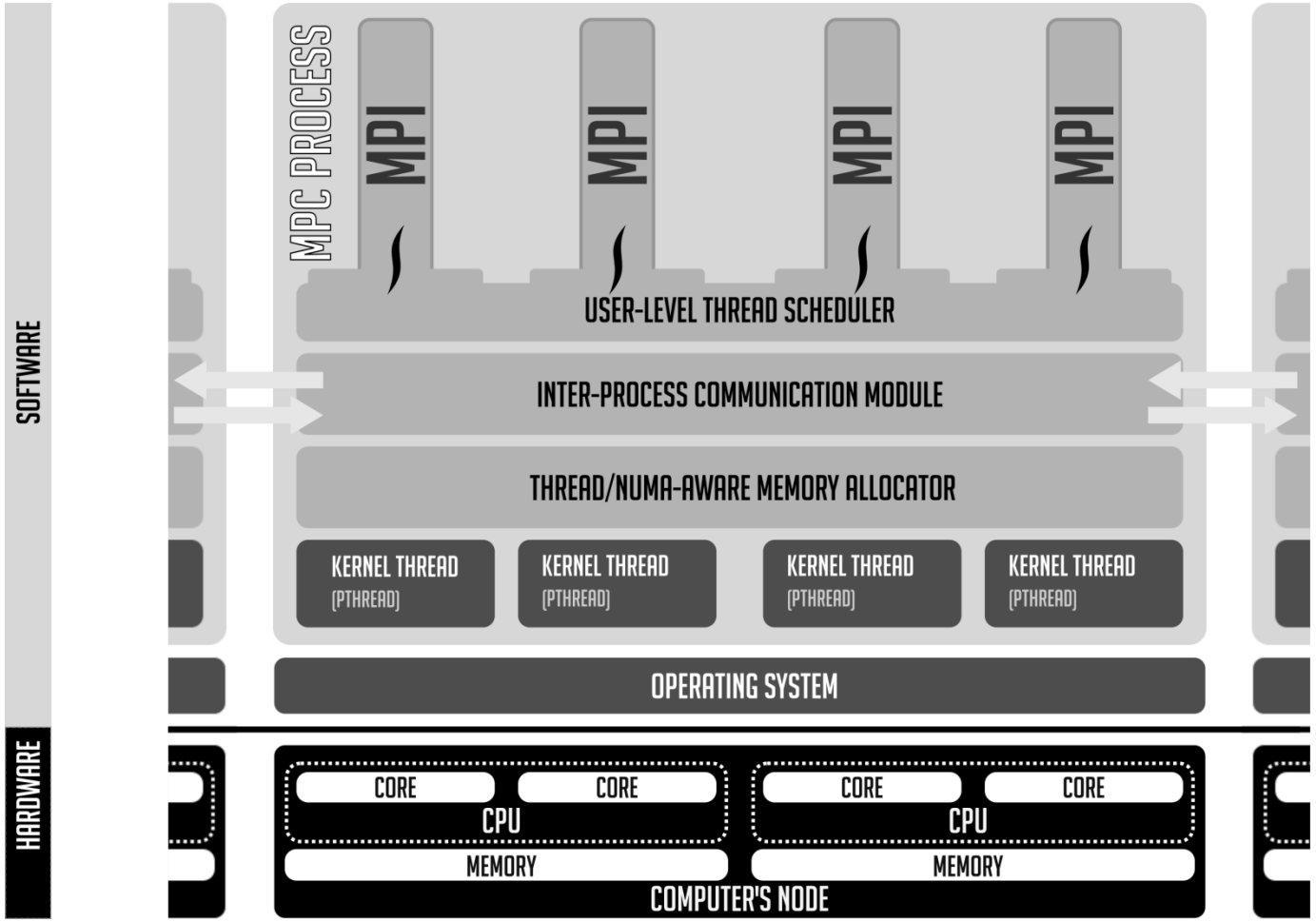
MPC-MPI Process-Based Execution Model

Application with 4 MPI tasks in 4 processes



MPC-MPI Thread-Based Execution Model

Application with 4 MPI tasks in 1 process



User-Level Thread Scheduler: MPI

Thread-based MPI

- Process virtualization
  - Each MPI process is a user-level thread

Benefits

- No busy waiting:
  - Efficient oversubscribing (more MPI processes than cores)
  - Share resources with other programming models
- Efficient message progression at node level:
  - Use “Idle” cycle to perform message progression
  - Collaborative Polling: shared message progression engine for all MPI Processes within a node

Drawbacks

- Not compatible with classical MPI usage
  - Issue with global variables
    - Solution with Automatic Privatization

Member of the MPI-Forum

Status of MPI-3.1 Implementations

	MPICH	MVAPICH	Open MPI	Cray MPI	Tianhe MPI	Intel MPI	IBM BG/Q MPI [1]	IBM PE MPICH [2]	IBM Platform	SGI MPI	Fujitsu MPI	MS MPI	MPC	NEC MPI
NBC	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	(*)	✔	✔
Nbrhood collectives	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✘	✔	✔
RMA	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✘	Q2'17	✔
Shared memory	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✔	*	✔
Tools Interface	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	*	Q4'16	✔
Comm-creat group	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✘	*	✔
F08 Bindings	✔	✔	✔	✔	✔	✘	✔	✘	✘	✔	✘	✘	Q2'16	✔
New Datatypes	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✔	✔	✔
Large Counts	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✔	Q2'16	✔
Matched Probe	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✔	Q2'16	✔
NBC I/O	✔	Q3'16	✔	✔	✘	✘	✘	✘	✘	✔	✘	✘	Q4'16	✔

Release dates are estimates and are subject to change at any time.
"✘" indicates no publicly announced plan to implement/support that feature.
Platform-specific restrictions might apply to the supported features

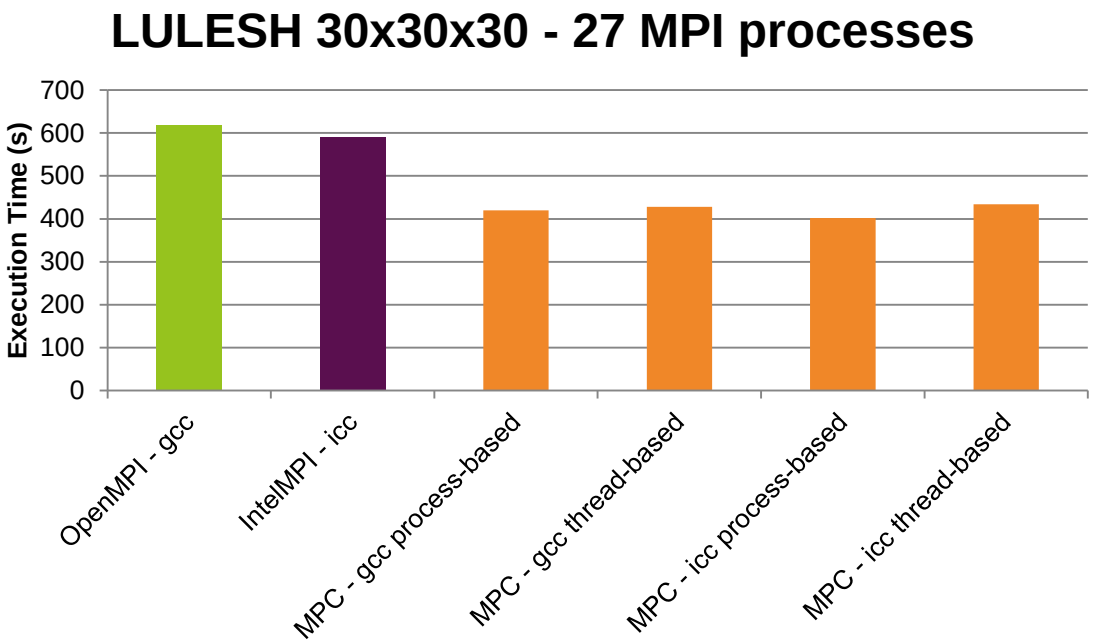
[1] Open Source but unsupported [2] No MPI_T variables exposed * Under development (*) Partly done

MPI EXPERIMENT

Architecture: Intel KNL

Application: LULESH MPI

- Evaluation of multiple MPI implementations (OpenMPI, IntelMPI, MPC with either GNU or Intel compiler)



- Early 15 nodes 1,000 processes EDR results MPC 46.14s vs OpenMPI 60.23s

MPC - OPENMP IMPLEMENTATION

RUNTIME STACKING: OPENMP LAYER

Impact of runtime stacking on OpenMP Layer

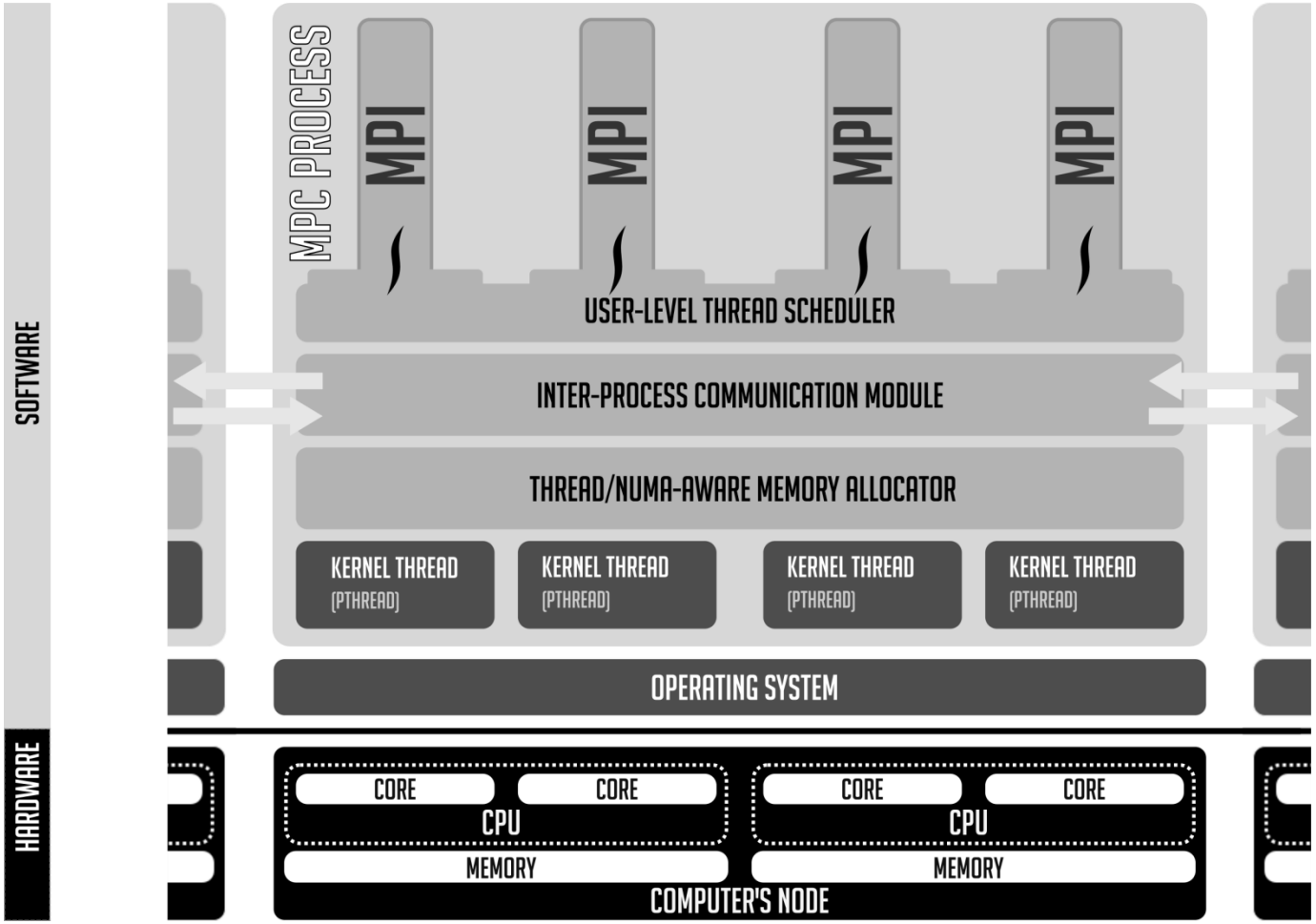
- Strong performance in fine-grain and coarse-grain approaches
  - Fine-grain
    - Optimization of launching/stopping a parallel region
    - Optimization of performing loop scheduling
  - Coarse-grain
    - Optimization of synchronization constructs (barrier, single, nowait…)
- Thread placement according to available cores (job manager + MPI runtime)
  - Oversubscribing → need to avoid busy waiting

OpenMP runtime design and implementation

- Goal: design of OpenMP runtime fully integrated into MPI runtime dealing with Granularity and Placement
- Implementation in MPC unified with optimized MPI layer

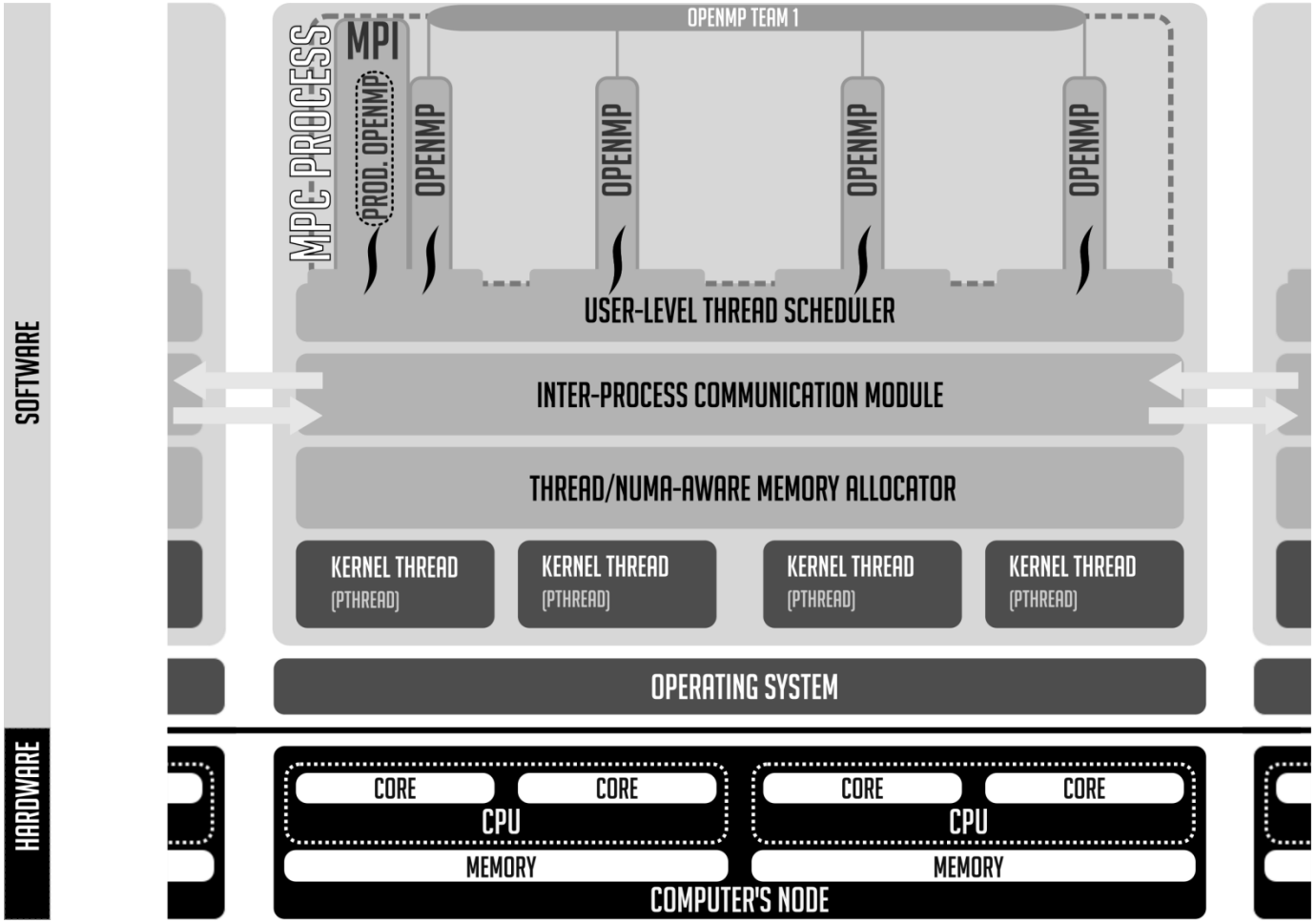
MPC Execution Model: Full MPI

4 MPI tasks in one process



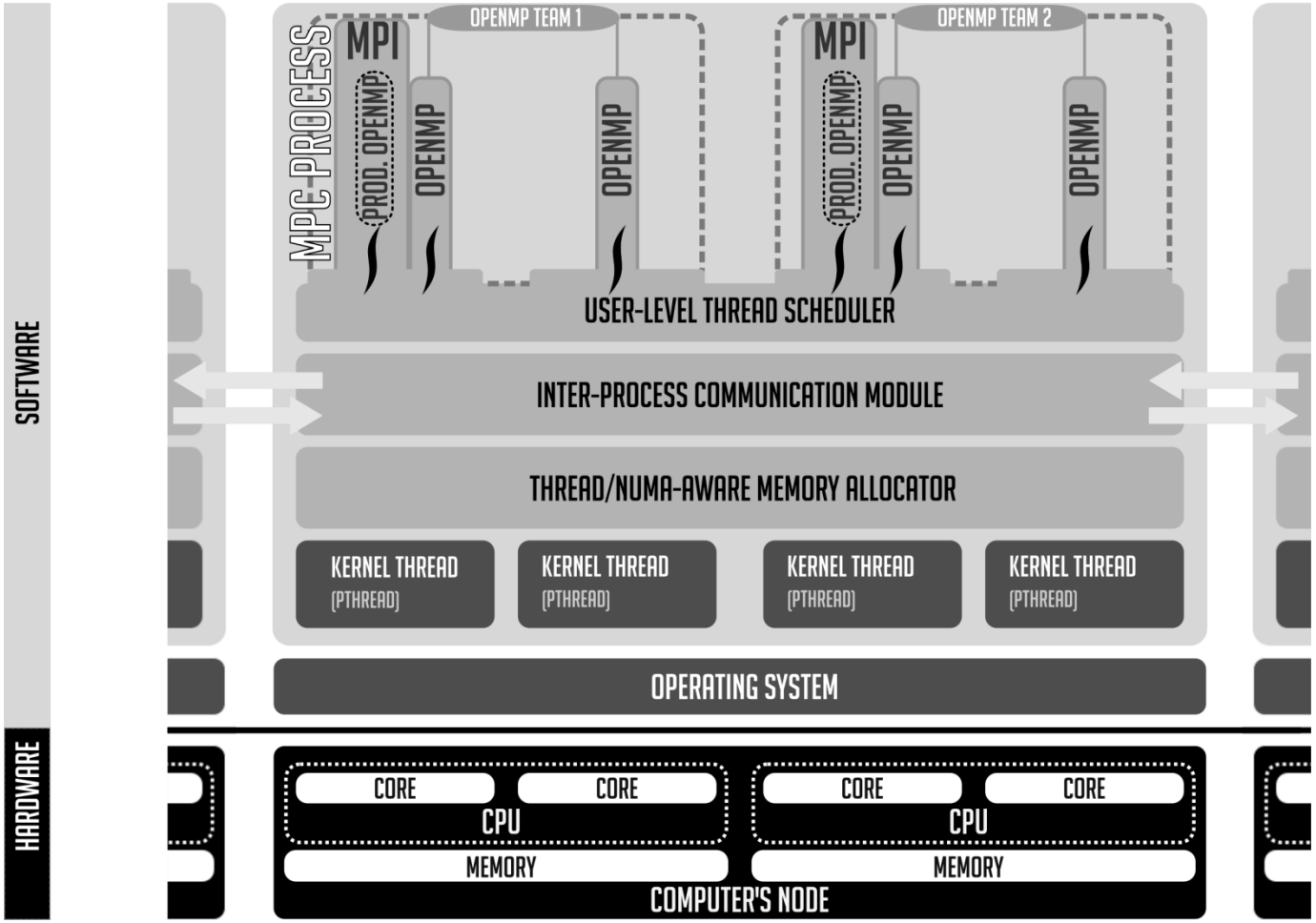
MPC Hybrid Execution Model: MPI+OpenMP (1)

1 MPI task + 4 OpenMP threads in one process



MPC Hybrid Execution Model: MPI+OpenMP (2)

2 MPI tasks + 4 OpenMP threads in one process



User-Level Thread Scheduler: OpenMP

OpenMP threads are User-Level thread

- Smart binding thanks to information sharing with MPI runtime through thread scheduler

Benefits

- Avoid busy waiting in multi-programming model context
  - Useful for fast thread wakeup/sleep (entering/leaving parallel regions, …)
- Data locality (link scheduler ⇔ memory allocator)

Drawbacks

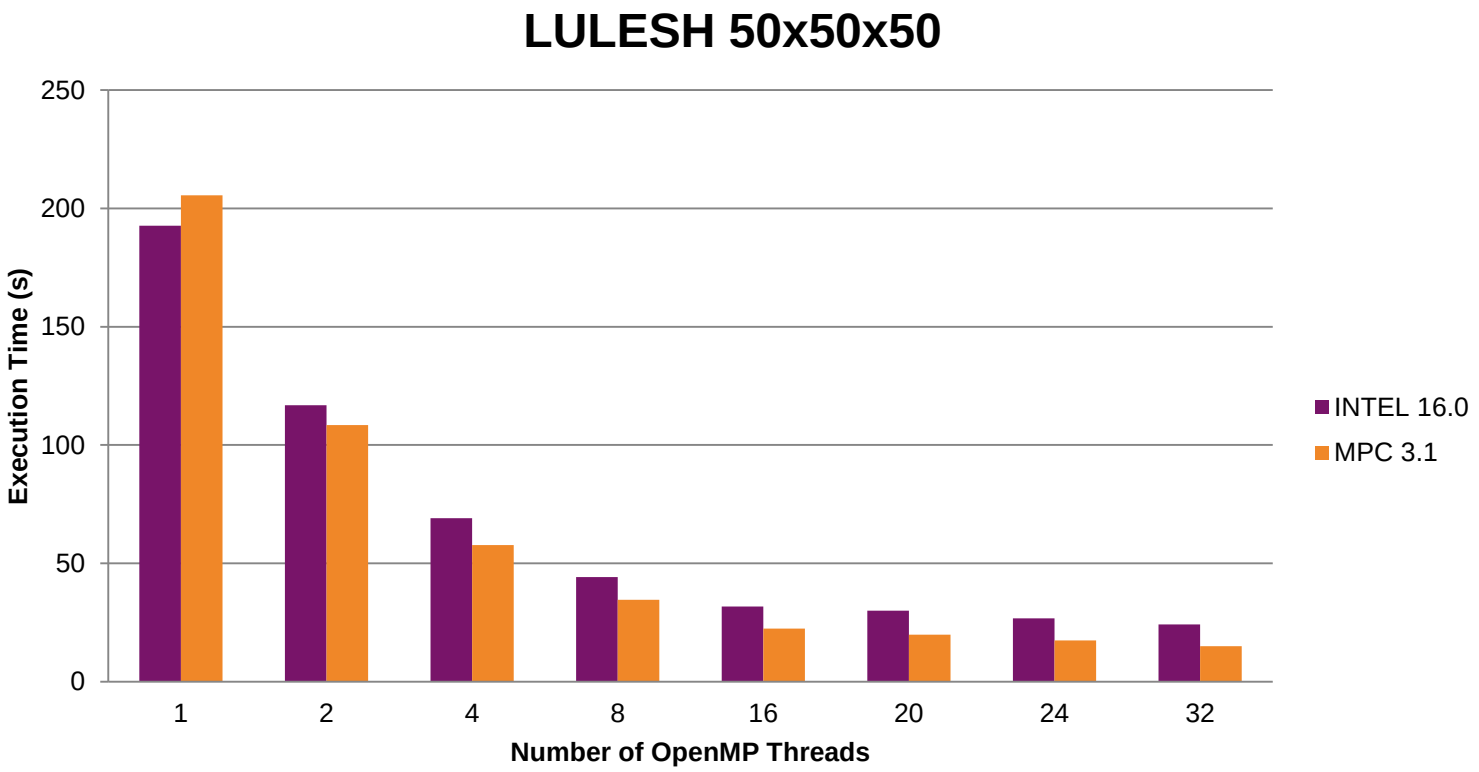
- Unable to use standard TLS for #pragma threadprivate
  - ➢ Solution with extended-TLS

OPENMP EXPERIMENT

Architecture: dual-socket 16-core Haswell

Application: LULESH OpenMP compiled with Intel 16

- Evaluation of Intel OpenMP runtime vs. MPC OpenMP runtime (same compiler)



MPC – EXTENDED TLS AND AUTOMATIC PRIVATIZATION

Extended TLS library: ExTLS

Use threads (e.g. MPI processes) instead of OS processes

- Convert standard OS process to MPC thread
- Encapsulate OS process within threads

Difficulties

- How to handle global variables
- Unable to use standard TLS for #pragma threadprivate
- Deal with non-thread safe libraries
  - User libraries: HDF5, …
  - System libraries: getopt, …
  - Compiler libraries: libgfortran, …

➔Automatic privatization thanks to compiler support

➔Provide patched version if not compatible yet with automatic privatization

➔Provide non-thread safe system libraries

Extended TLS Application: Automatic Privatization

Solution: **Automatic privatization**

- Automatically convert any MPI code for thread-based MPI compliance
- Duplicate each global variable

Design & Implementation

- Completely transparent to the user
- When parsing or creating a new global variable: flag it as MPI thread-local
  - #pragma threadprivate : flag it as OpenMP thread-local
- Generate runtime calls to access such variables (extension of TLS mechanism)
  - Linker optimization for reduce overhead of global variable access

Compiler support

- New option to GCC C/C++/Fortran compiler (`-fmpc-privatize`)
  - Patched GCC provided with MPC (4.8.5, on going on GCC 4.9.x and 5.x)
- ICC support automatic privatization with same flag (`-fmpc-privatize`)
  - ICC 15.0.2 and later
- On-going work for PGI compiler support

Official Support of MPC in Intel Compiler Man Page

Official support of MPC since Intel Compiler v15.02

- Extracted from the icc/icpc/ifort man page

```
> man icc
...
Feature: Privatization of static data for the MPC unified parallel runtime Requirement:
Appropriate elements of the MultiProcessor Computing (MPC) framework For more information,
see http://mpc.sourceforge.net/
...
-fmpc-privatize (L*X only) / -fno-mpc-privatize (L*X only)
Enables or disables privatization of all static data for the MultiProcessor Computing
environment (MPC) unified parallel runtime.
Architecture Restriction: Only available on Intel(R) 64 architecture
Arguments: None
Default: -fno-mpc-privatize
The privatization of all static data for the MPC unified parallel runtime is disabled.
Description:
This option enables or disables privatization of all static data for the MultiProcessor
Computing environment (MPC) unified parallel runtime.
Option -fmpc-privatize causes calls to extended thread-local-storage (TLS) resolution, run-
time routines that are not supported on standard Linux* OS distributions.
This option requires installation of another product. For more information, see Feature
Requirements.
```

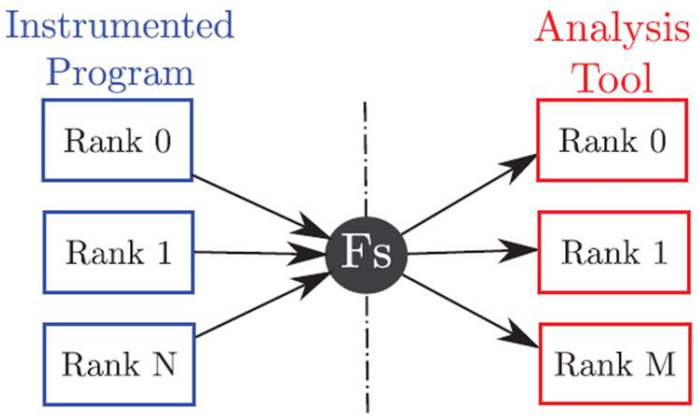
Ecosystem Survey

MALP: MULTI-APPLICATION ON-LINE PROFILING

PROFILING APPLICATIONS AT SCALE

Approaches

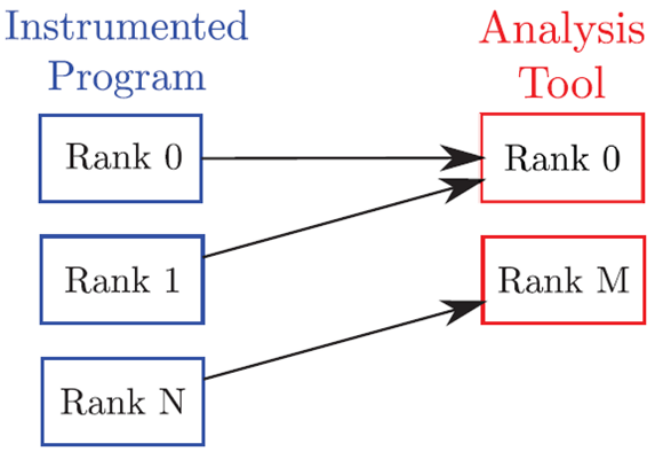
- In-place
  - Analysis uses applications cores
  - Simple analysis only
- Traced based
  - Rely on file system
  - Post mortem analysis
- On-line
  - Combine the two previous approaches
  - In-place analysis to reduce the amount of data on file system
  - Dedicated resources



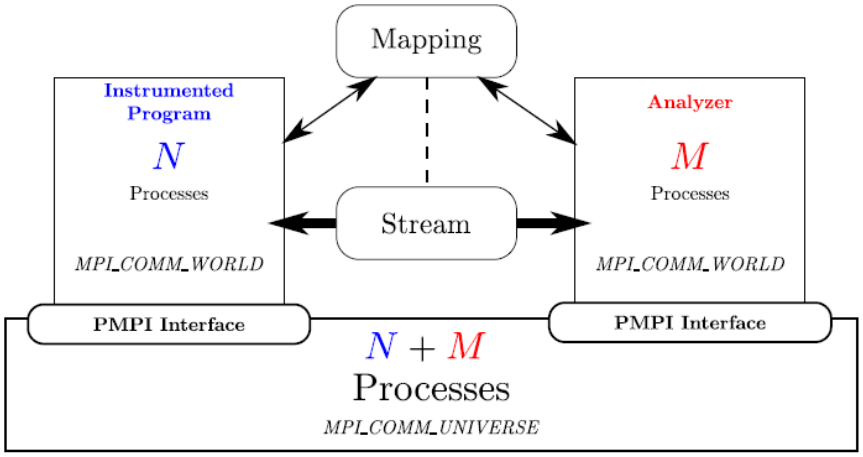
Trace based

Our proposition: the MALP tool

- On-line analysis
- Use high performance networks
- Handle simultaneously multiple applications

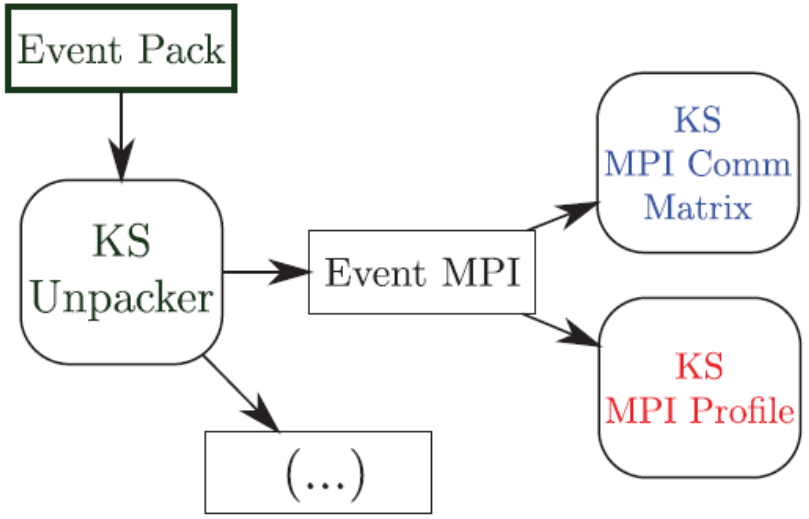


MULTI-APPLICATION ONLINE PROFILING



- Engine
  - Use dedicated cores
    - Trade-off performances vs analysis cost
    - Limited impact on application execution
  - Use high speed interconnect

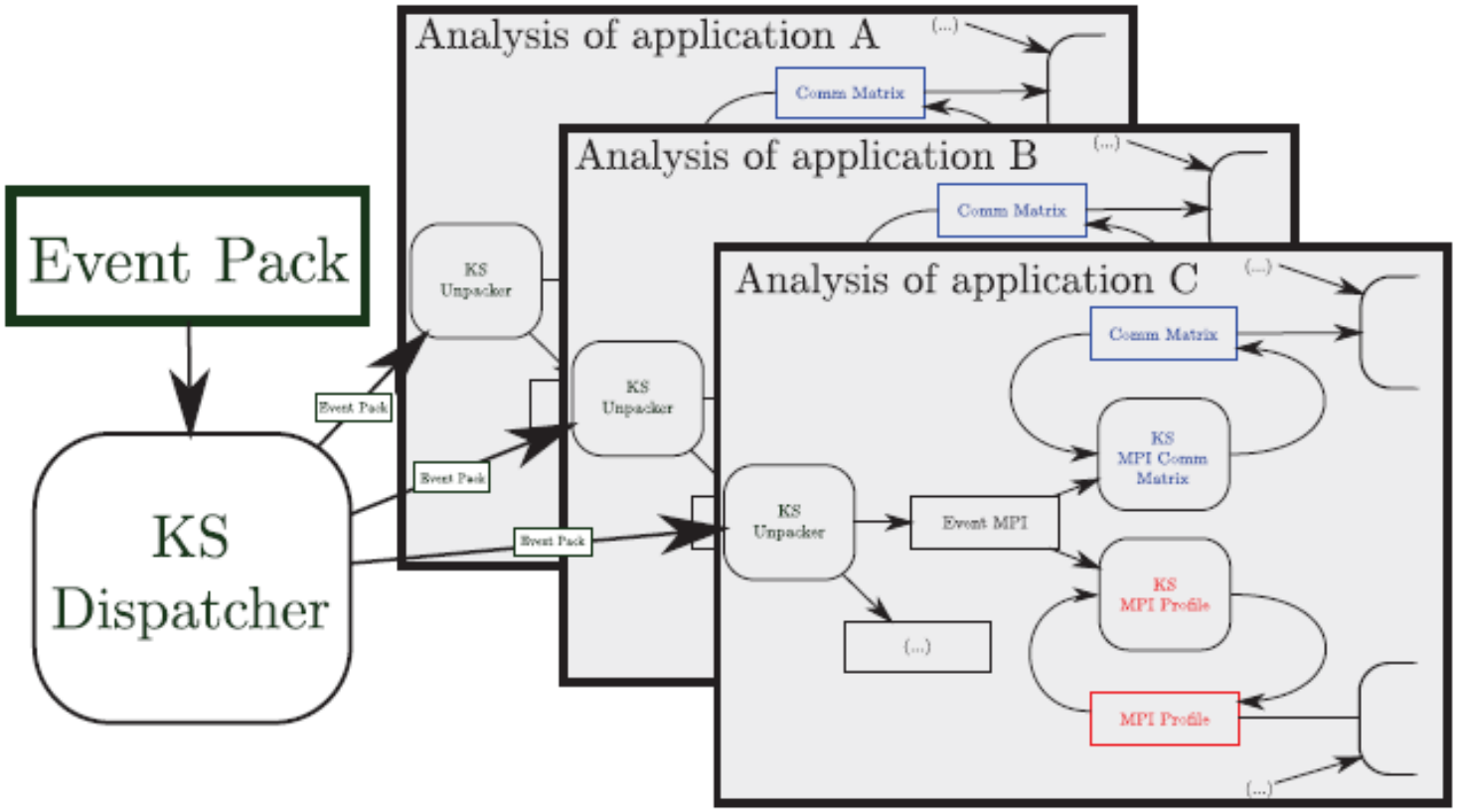
- Analysis
  - Based on blackboards
  - Analysis engine
    - Multiple analysis on events
    - Parallel analysis
  - Events:
    - MPI
    - POSIX
    - Function calls



MULTI-APPLICATION ONLINE PROFILING

Multiple application feature

- Profile multiple application at scale
- Share analysis resources

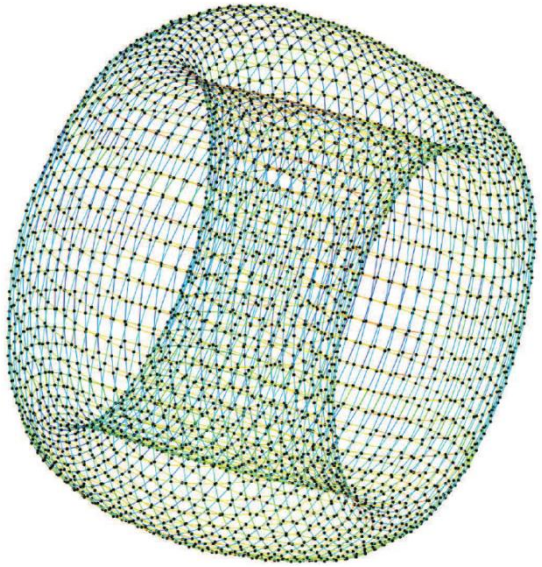


ANALYSIS EXAMPLE

EulerMHD profil

MPI Operation	Hits	Total Time	Avg. time	%	Total Size	Avg. Size
MPI_Wait	2566619136	1 d 12 h 35 m 1 s	51.31 us	13.10 %	-	-
MPI_Allreduce	39534592	1 d 10 h 22 m 57 s	3.131 ms	12.31 %	302.09 MB	8 B
MPI_Isend	1283309568	40 min 15 s	1.883 us	0.24 %	20.35 TB	17.03 KB
MPI_Irecv	1283309568	7 min 49 s	365.6 ns	0.046 %	20.35 TB	17.03 KB
MPI_Type_size	2606161920	2 min 32 s	58.15 ns	0.015 %	-	-
MPI_Reduce	8192	717.7 ms	87.61 us	7.13e-5 %	146.19 MB	18.27 KB
MPI_Comm_rank	24576	18.98 ms	771.8 ns	1.88e-8 %	-	-
MPI_Comm_size	16384	3.731 ms	227.4 ns	3.71e-7 %	-	-

8192 cores on Curie



Multi-Application On-Line Profiling

What is MALP ?

- MALP stands for Multi-Application on-Line Profiling it is an online performance tracing tool aiming at overcoming common file-system limitations by relying on runtime coupling between running applications.

What can I do with MALP ?

- With MALP you can generate compact views of your parallel application behavior. Thanks to its web interface and interactive visualizations, you are able to better understand its MPI behavior at scale thanks to the online data-management approach.

Roadmap

- Compatibility with Allinea MAP/Performance report plugins
- New plugins: memory allocation, OpenMP, …
- Direct access to the underlying network (remove MPI dependency)

Status

- OpenSource Cecill-C
- http://malp.hpcframework.com/

JCHONOSS: Job CHecker under Resources cONstraints with Optimized and Scalable Scheduler

Validation in HPC context

Continuous integration

- Mandatory to validate huge codes or libraries (MPC)
- Require big enough machines (high speed networks, NUMA topologies, …)

Validation on production machines

- Heavy loaded machines
- Need to use batch manager
- How to deal with short tests
  - Time spent waiting for an allocation >> test execution time
- Tests properties:
  - Variable execution times
  - Variable number of nodes/cores

Our approach JCHRONOSS

- Job scheduler
- Interfaces with:
  - Batch manager (SLURM, …)
  - GUIs thanks to JUNIT format
  - XML files to describe the tests
- Integrated checkpoint/restart

Master/Worker

- Master process
  - Distribute tests sets
  - Compute scheduling according to test execution time (if available)
  - Call first step allocation (e.g. salloc)
- Worker processes
  - Execute tests (e.g. srun)
  - Gather results
  - Generate intermediate JUNIT files

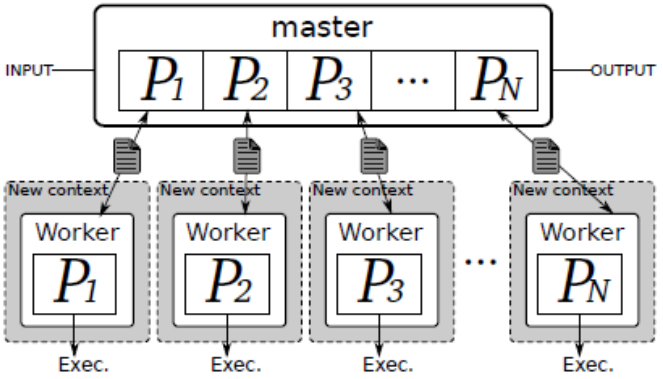
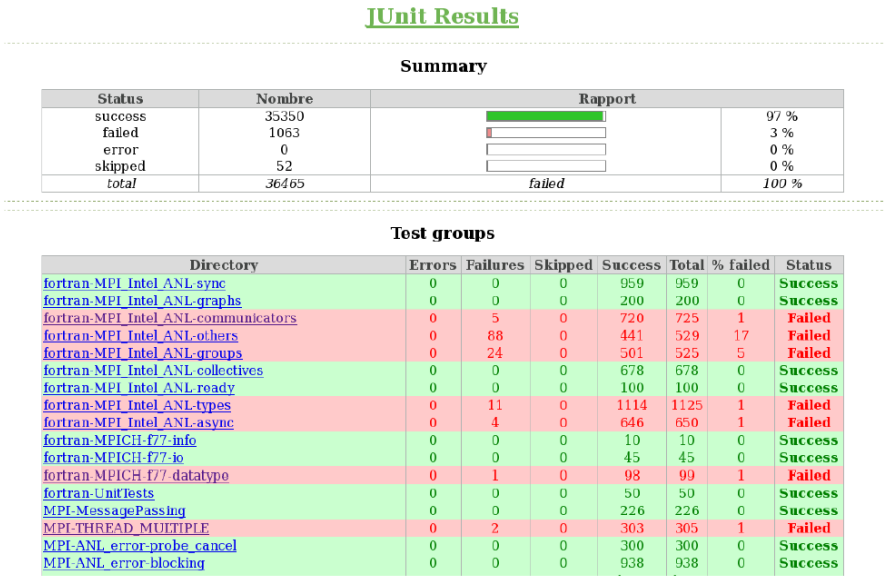


Fig. 1. Master/Worker Architecture Overview

JCHRONOSS Embedded Viewer

Webview

- Visualize intermediate results during test-suite execution
- Portable
  - HTML/XML files
  - Only require a web browser



JUnit Results

Summary

Status	Nombre	Rapport	
success	35350		97 %
failed	1063		3 %
error	0		0 %
skipped	52		0 %
total	*36465*	*failed*	*100 %*

Test groups

Directory	Errors	Failures	Skipped	Success	Total	% failed	Status
fortran-MPI_Intel_ANL-sync	0	0	0	959	959	0	Success
fortran-MPI_Intel_ANL-graphs	0	0	0	200	200	0	Success
fortran-MPI_Intel_ANL-communicators	0	5	0	720	725	1	Failed
fortran-MPI_Intel_ANL-others	0	88	0	441	529	17	Failed
fortran-MPI_Intel_ANL-groups	0	24	0	501	525	5	Failed
fortran-MPI_Intel_ANL-collectives	0	0	0	678	678	0	Success
fortran-MPI_Intel_ANL-ready	0	0	0	100	100	0	Success
fortran-MPI_Intel_ANL-types	0	11	0	1114	1125	1	Failed
fortran-MPI_Intel_ANL-async	0	4	0	646	650	1	Failed
fortran-MPICH-f77-info	0	0	0	10	10	0	Success
fortran-MPICH-f77-io	0	0	0	45	45	0	Success
fortran-MPICH-f77-datatype	0	1	0	98	99	1	Failed
fortran-UnitTests	0	0	0	50	50	0	Success
MPI-MessagePassing	0	0	0	226	226	0	Success
MPI-THREAD_MULTIPLE	0	2	0	303	305	1	Failed
MPI-ANL_error-probe_cancel	0	0	0	300	300	0	Success
MPI-ANL_error-blocking	0	0	0	938	938	0	Success

JCHRONOSS

What's JCHRONOSS

- JCHRONOSS is a tool helping you to maintain a high level of quality in your project by making your validation step easier and faster. Initially suited for High Performance Computing test suites, JCHRONOSS is able to distribute thousands of parallel jobs in parallel environments and over large supercomputers.
- Designed with a high level of abstraction, JCHRONOSS can work with any kind of batch managers or architectures. Some extra features embedded with JCHRONOSS like scheduling visualization and standalone result reviewing, makes JCHRONOSS a complete and powerful tool for most of validation suites.

Roadmap

- Better integration with Jenkins
- Real time interactions with test suite

Status

- OpenSource Cecill-C
- http://jchronoss.hpcframework.com/

Wi4MPI: Wrapper interface For MPI

WI4MPI: CONTEXT

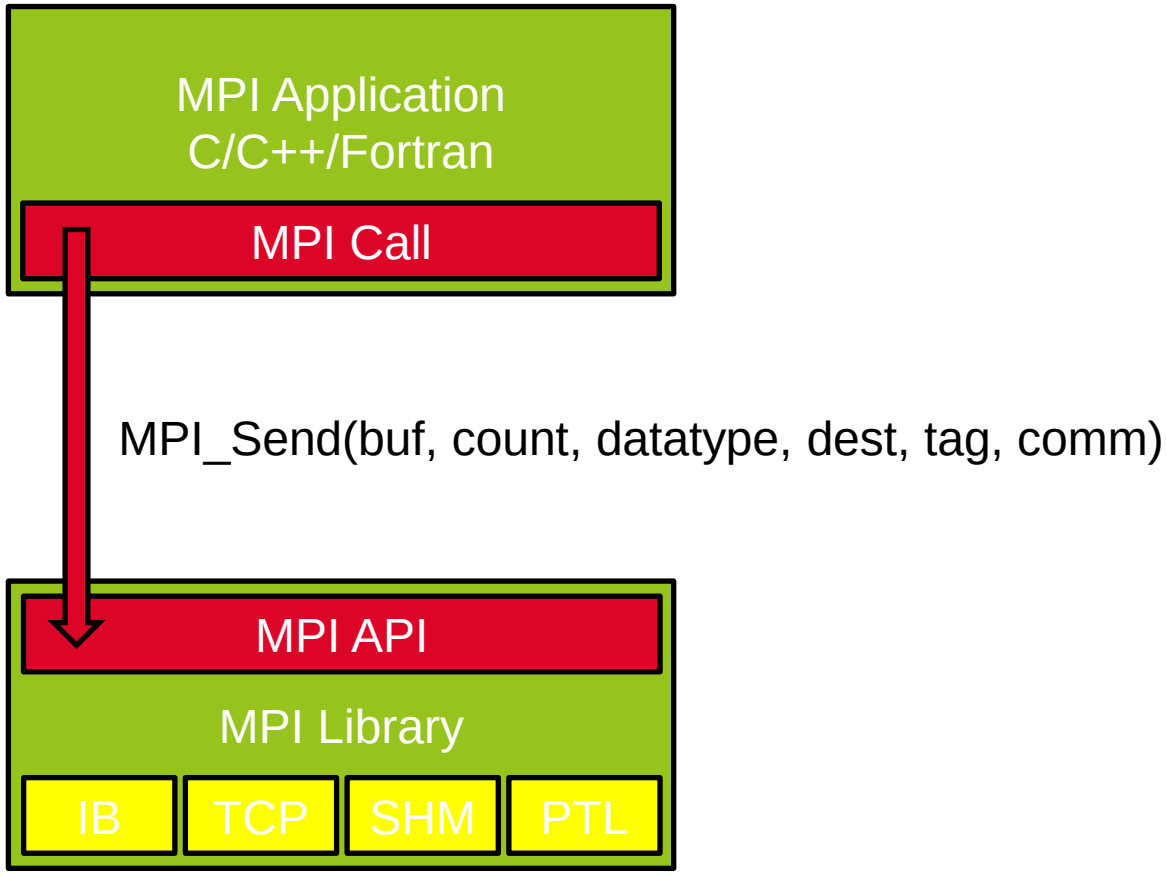
The MPI ABI is not standardized

- Many MPI implementations:
  - MPICH (IntelMPI, MVAPICH, …)
  - OpenMPI (BullxMPI, …)
  - Not binary compatible
- Within the same implementation no binary compatibility
  - OpenMPI 1.8 ABI != OpenMPI 2.0 ABI
- Need to recompile the whole software stack to change the MPI implementation or version

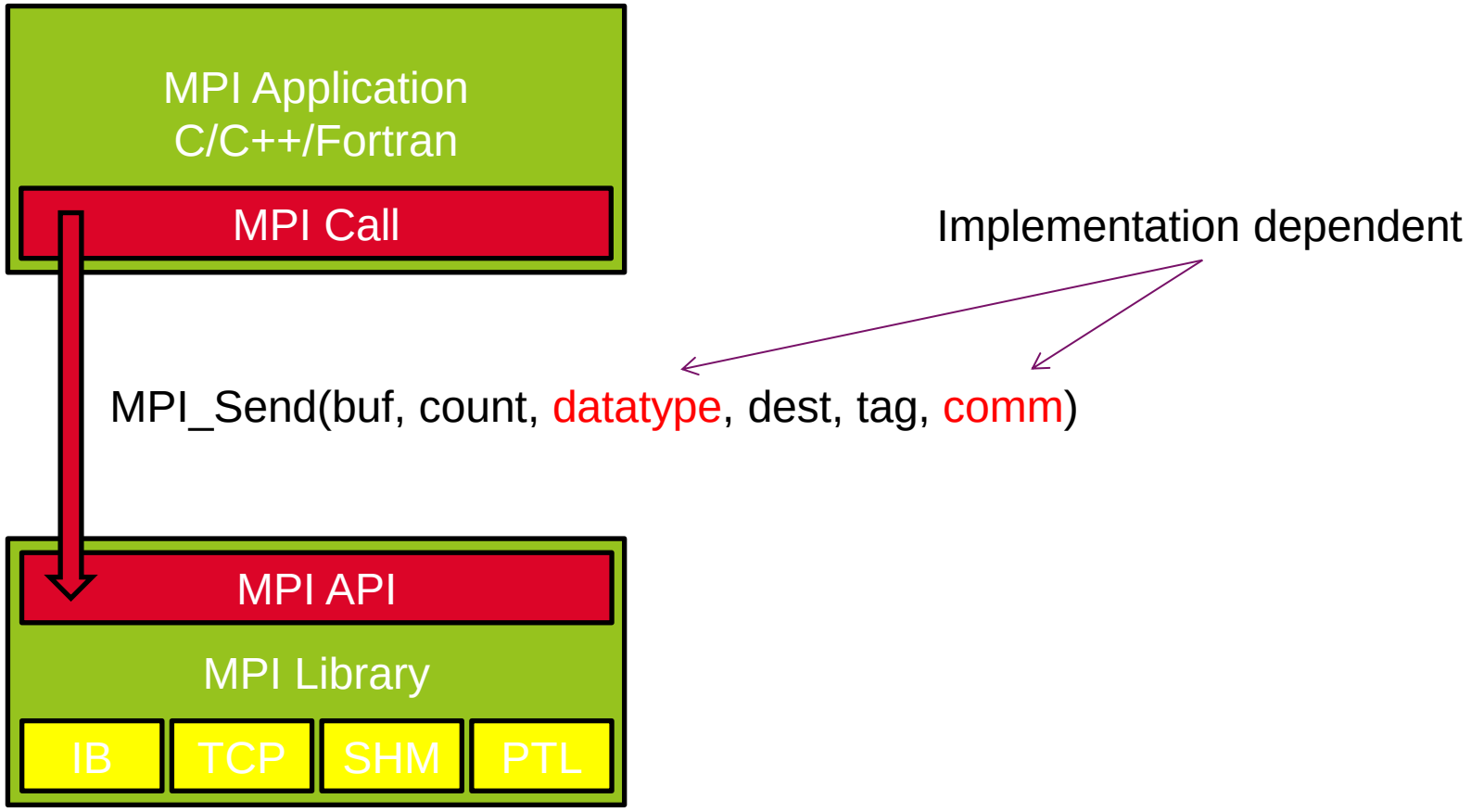
Wi4MPI allows to avoid recompilation phase

- Allow to abstract the underlying MPI implementation
  - Wi4MPI is now the MPI implementation at application level
- Dynamically change the MPI implementation
- Ease debugging phases
- Ease MPI implementation performances evaluation

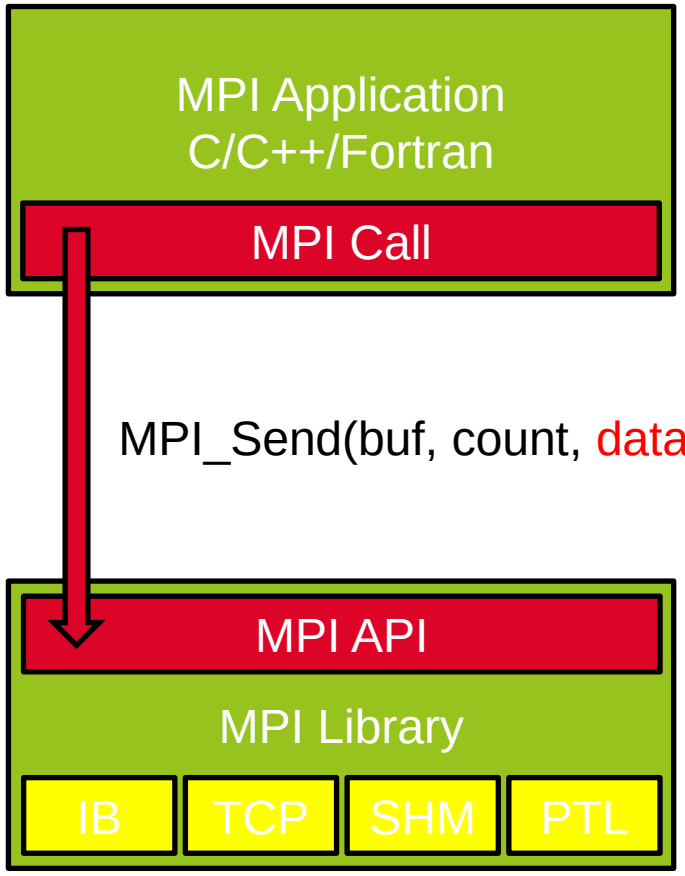
STANDARD MPI APPROACH



STANDARD MPI APPROACH



STANDARD MPI APPROACH



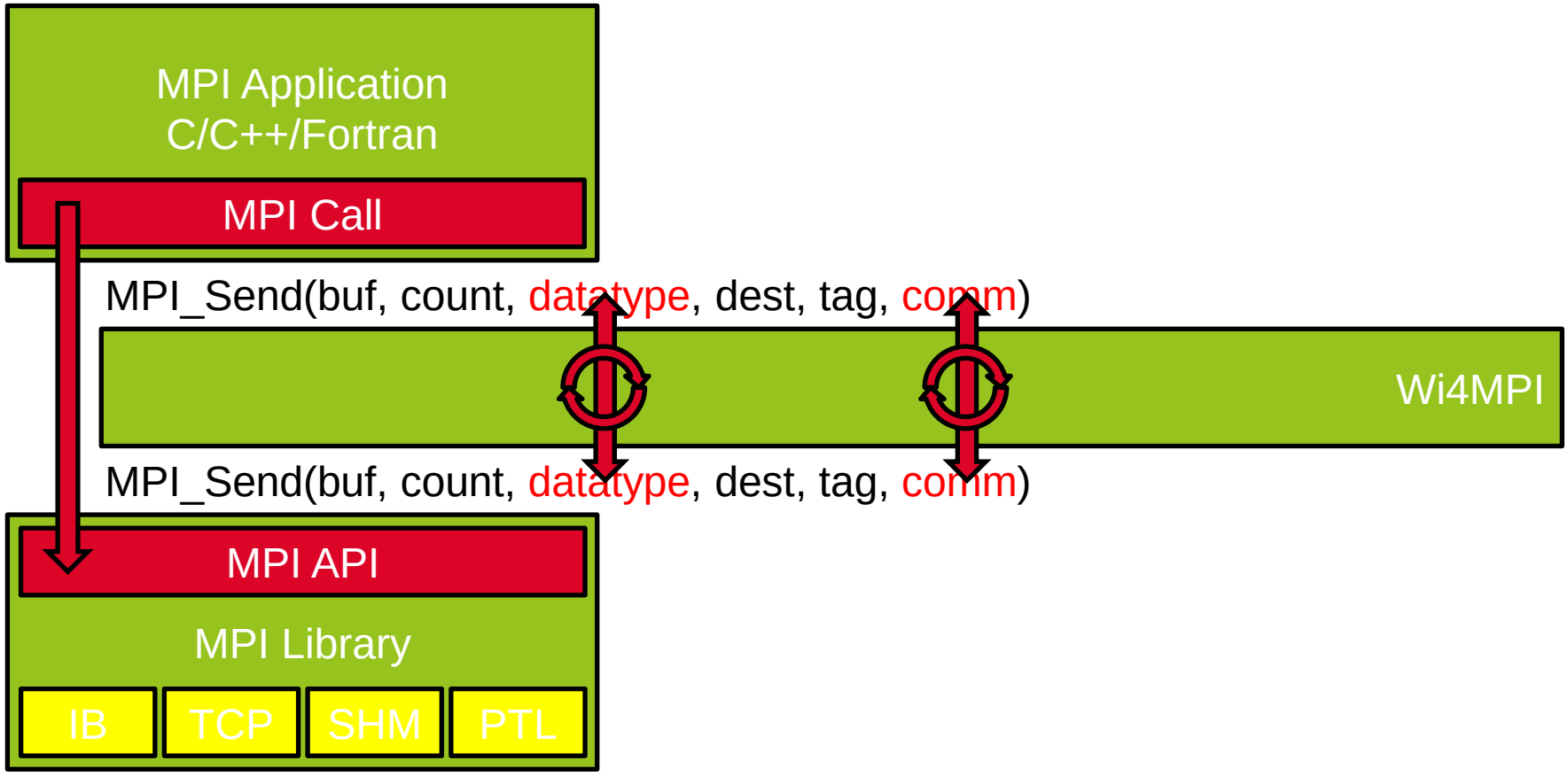
Implementation dependent

MPI_Send(buf, count, datatype, dest, tag, comm)

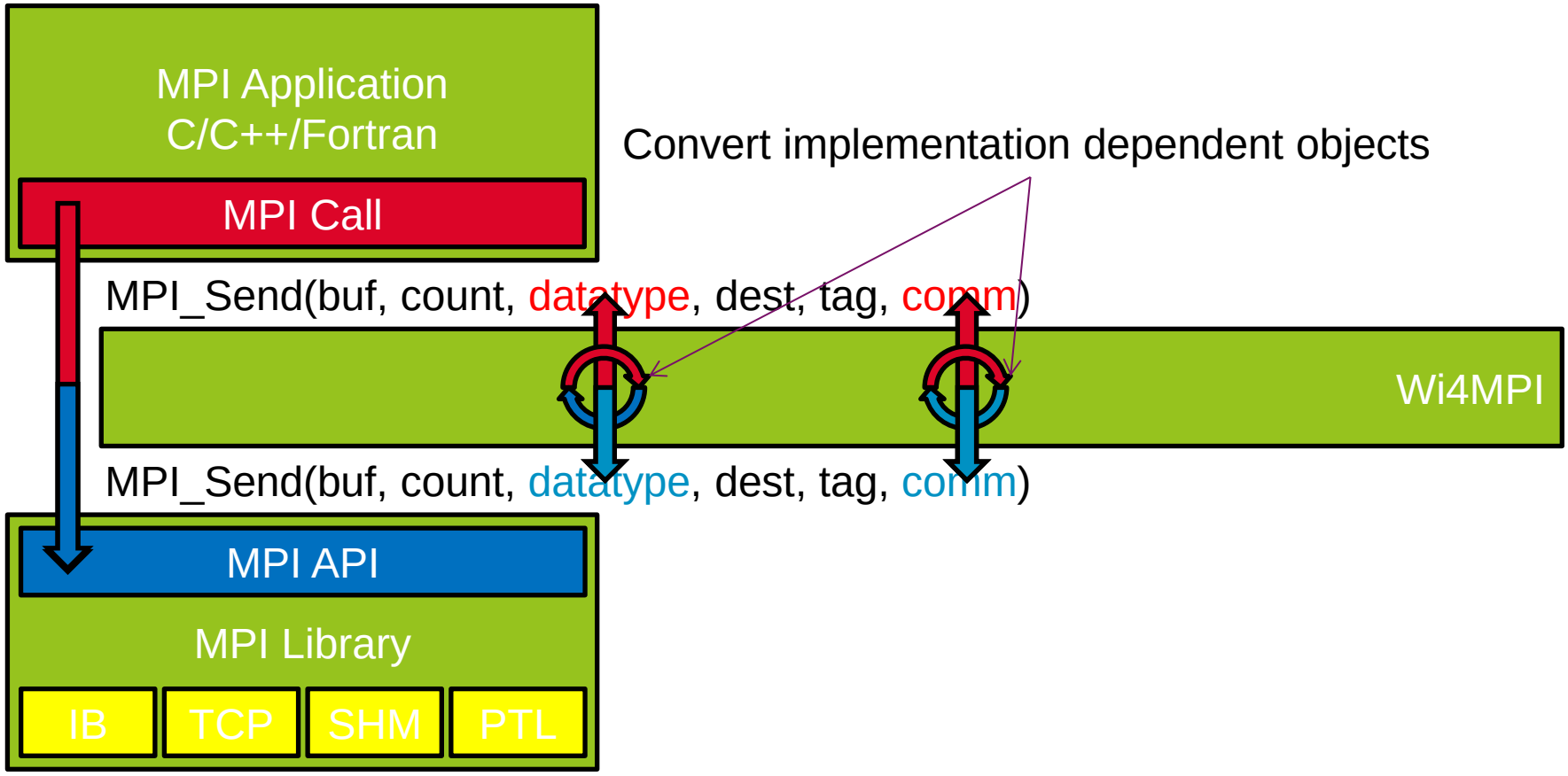
Example: MPI_COMM_WORLD

- MPICH implementation:
  - C type: int
  - Value: 0x44000000
- OpenMPI implementation:
  - C type: ompi_communicator_t *
  - Value: &ompi_mpi_comm_world

WI4MPI APPROACH



WI4MPI: REDMPI-BLUEMPI TRANSLATION



WI4MPI: STATUS

Features

- With Wi4MPI, the software stack is MPI-independent
- Avoid software recompilation to change the MPI implementation
- Choose MPI implementation at execution time (OpenMPI, IntelMPI, …)
- C/C++ and Fortran support
- MPI 1.3 compliant + MPI-IO
- Validated conversions:
  - OpenMPI 1.8.8 ➔ IntelMPI 5.1.3.181
  - IntelMPI 5.1.3.181➔ OpenMPI 1.8.8
- Validated on real production codes

Roadmap

- Process mode MPC support (OpenMPI ⇔ MPC, IntelMPI ⇔ MPC) Q4 16
- In production on CEA clusters
- Full MPI 3.1 support Q4 16

Status

- OpenSource, Cecill-C License (fully LGPL compatible)
- https://github.com/cea-hpc/wi4mpi

Conclusion & Future Work

Conclusion

Programming models

- Provide widely spread standards: MPI 1.3+ (almost MPI 3.1), OpenMP 3.1, Pthread, TBB
- Available at http://mpc.hpcframework.com (version 3.1)
- Optimized for manycore and NUMA architectures

Runtime optimization

- Provide unified runtime for MPI + X applications
- New mechanism to mix thread-based programming models: Extended TLS
  - Compiler option to duplicate global variables: `-fmpc-privatize` (gcc, icc)

Support

- Architecture: x86, x86_64, MIC
- Network: TCP, Infiniband and Portals4 with multi-rail
- Resource manager: Slurm & Hydra

Tools

- Debugger support (Allinea DDT), Profiling
- Compiler support (Intel, GCC)
- MALP (collaboration with Paratools, compatible with Allinea plugins)
- WI4MPI (collaboration with Bull/Atos)
- JCHRONOSS (collaboration with Paratools)

Future Work

MPI

- Full support of MPI 3.1
- Portals 4 API and Atos/Bull BXI optimized support
- Extra-threads scheduling

OpenMP

- Support for parts of OpenMP 4.1 and OpenMP 5.0 (e.g., OMPT interface)
- Compatibility with PGI ABI & GOMP ABI

Hybrid scheduling

- New compiler support for automatic privatization: PGI
- In-depth study of runtime stacking
- Specialized-thread scheduling

From petascale to exascale

- Language evaluation (PGAS, One sided)
- Broader hardware support (Power, ARM)
- Better hardware support (Intel KNL, GPU)

Commissariat à l’énergie atomique et aux énergies alternatives
CEA, DAM, DIF, F-91297 Arpajon, France
T. +33 (0)1 69 26 40 00

Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019