A multithreaded communication engine for multicore architectures

F. Trahay, E. Brunet, A. Denis, R. Namyst

Runtime Team

University of Bordeaux I – INRIA – LaBRI

Hardware evolution

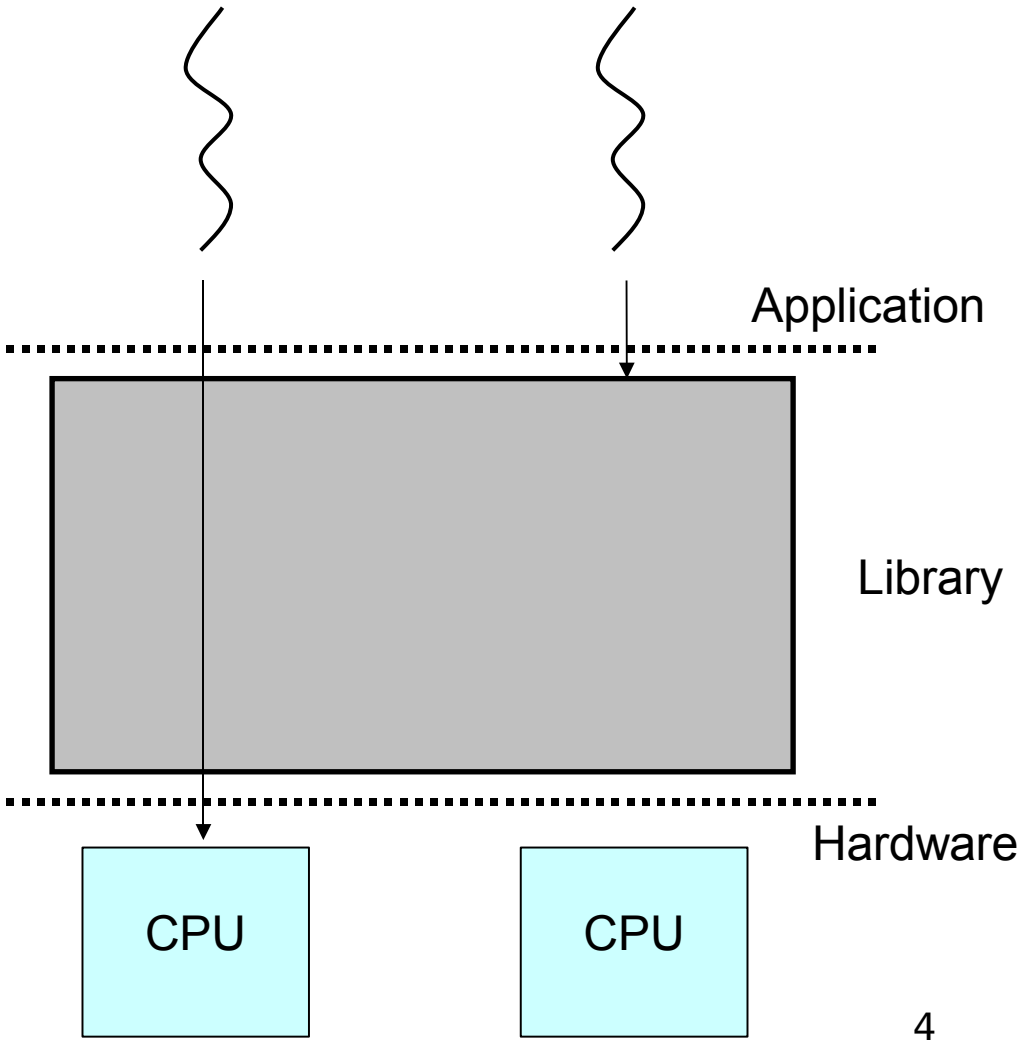
- Cluster's evolution
  - 5 years ago : < 4 cores per node
  - Today : > 8 cores per node
  - Tomorrow : tens / hundreds of cores per node
- Consequences
  - Memory bus contention
  - I/O contention
    - Many CPUs share a few NICs
- How to accelerate communication processing ?
- How to multiplex communication flows ?

Existing approaches

- "Pure-MPI" approaches
  - One MPI process per core
    - Intra-node communication through shared memory
    - Inter-node communication through the network
  - Scalability issues
    - Resources limitations (memory, load balancing...)
    - Hardware-level contention issues
- Hybrid approach (MPI + threads)
  - One MPI process per node
  - Global view of the node at the runtime level
    - Easier load balancing
    - Communication flows optimizations
    - Runtime-level congestion avoidance

Today's MPI implementations

- Support of multithreaded applications
- Do not take advantage of multithreading
  - Communication processing performed sequentially
  - Simultaneous accesses ensured through a library-wide scope mutex
  - (Almost) no load balancing

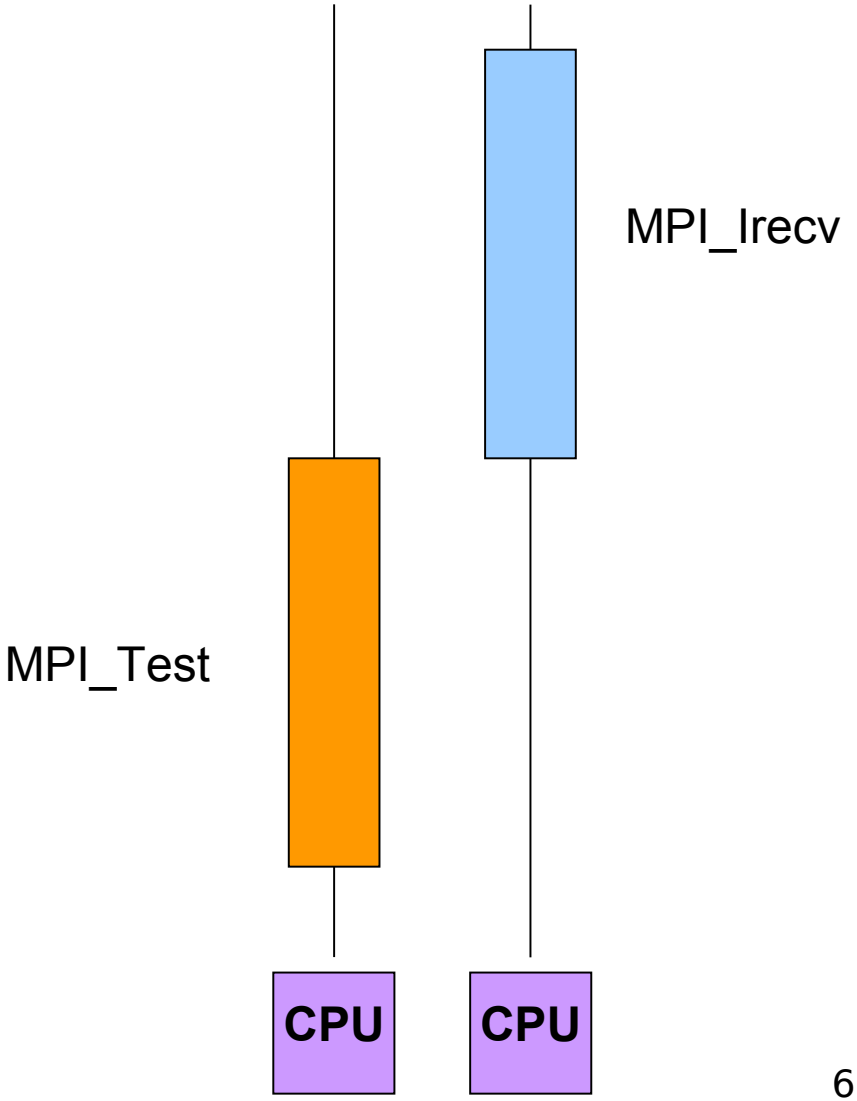


Challenges

- Thread safety
  - Allowing simultaneous access to the library
- Background progression of communication
  - Making rendezvous progress in background
- Parallel processing
  - Using several cores to process communication

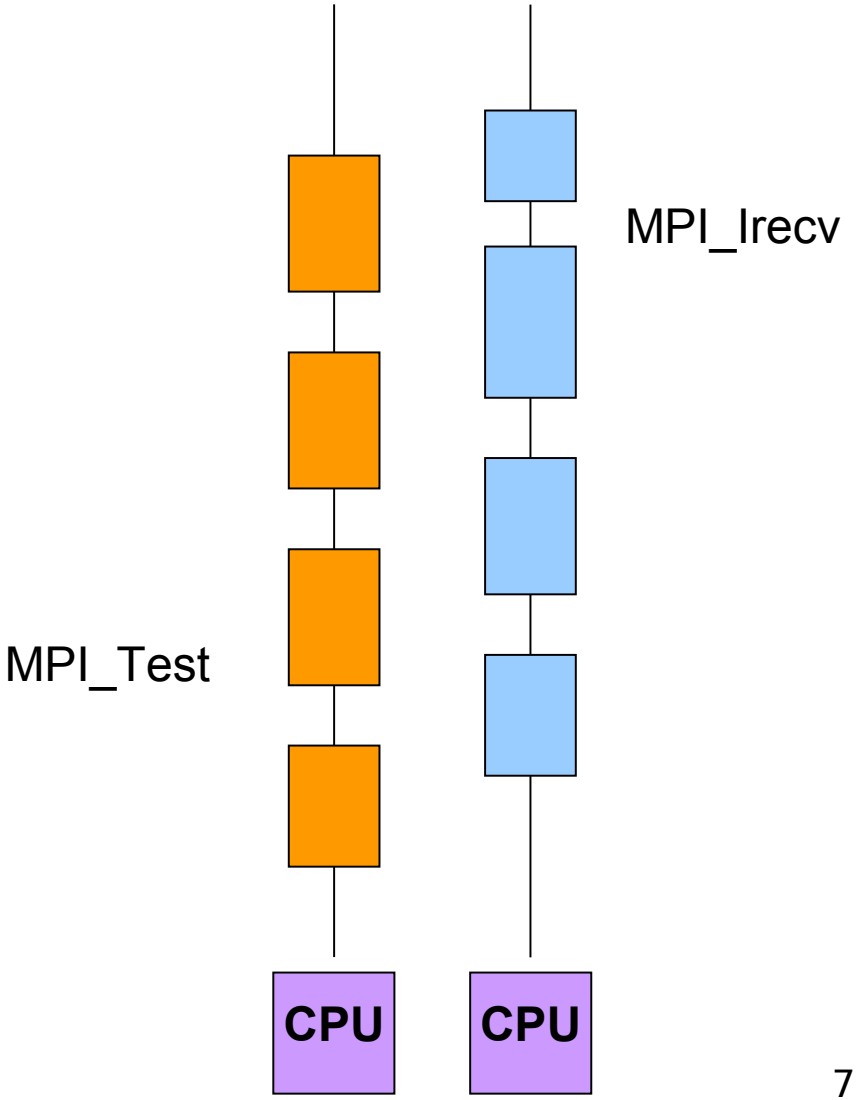
Ensuring thread-safety

- Easiest way : Coarse grain parallelism
  - Library-wide mutex
  - Avoids simultaneous access to the library



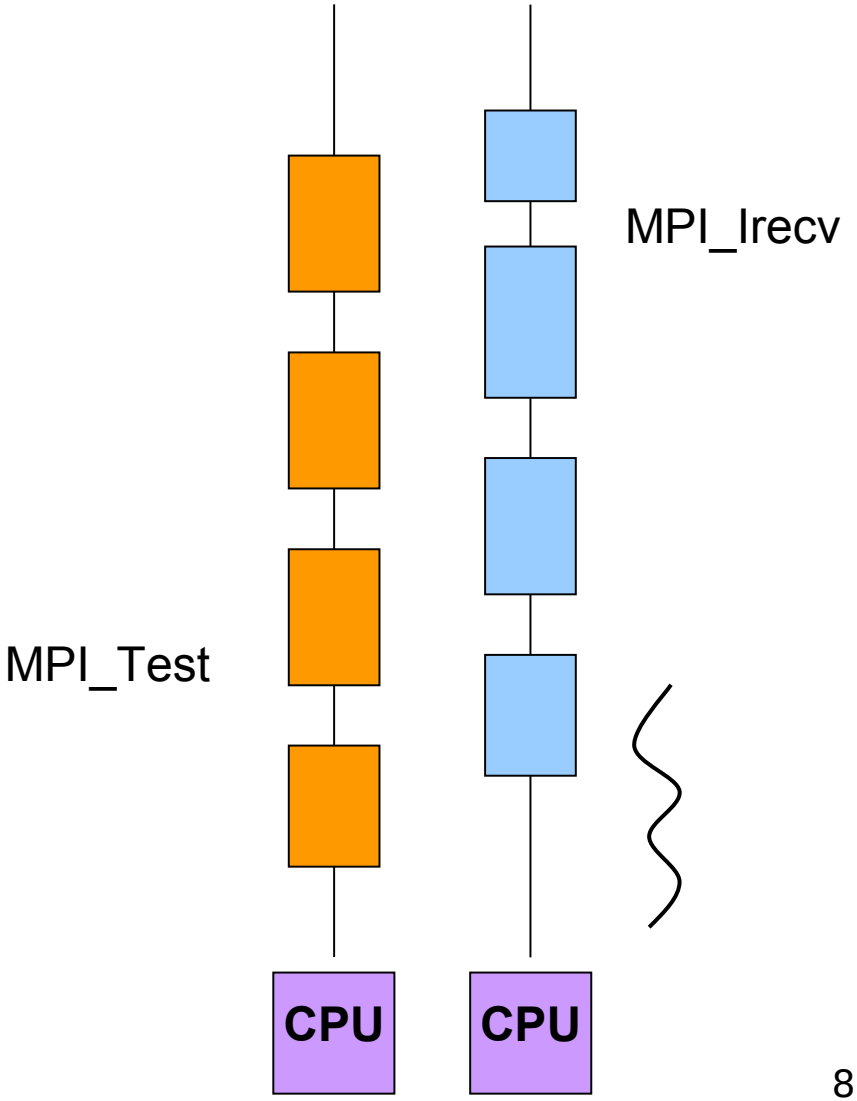
Ensuring thread-safety

- Easiest way : coarse grain parallelism
  - Library-wide mutex
  - Avoids simultaneous access to the library
- More efficient : fine grain parallelism
  - Action-wide mutexes
  - Ensure local thread-safety
  - Allow simultaneous access to the library



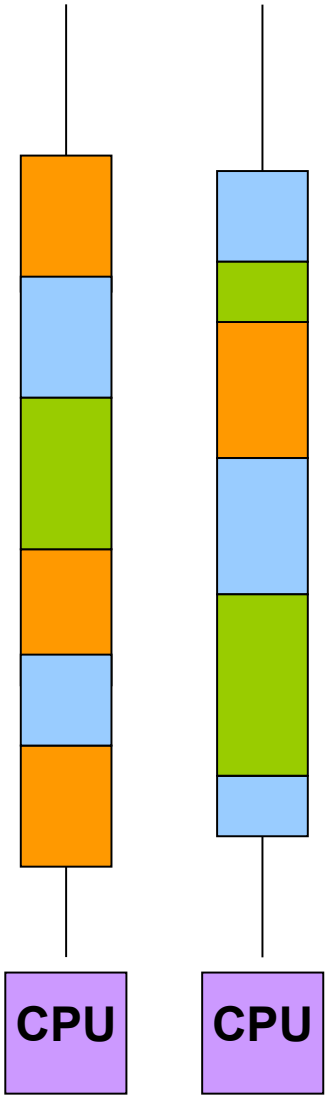
Background processing

- Adding a progression thread per NIC
  - Rendezvous handshake progression (Myrinet's MX, OpenMPI, etc.)
  - Priority problem on overloaded systems
- Scheduler-centric approach
  - Idle cores able to poll and make progress
  - No priority problem **[EuroPVM/MPI 07]**



Parallel processing of communication flows

- Communication processing seen as a sequence of operations
  - Operations may be performed on different cores
  - Load balancing of processing
    - Idle cores 'help' working cores
  - Offloading asynchronous operations

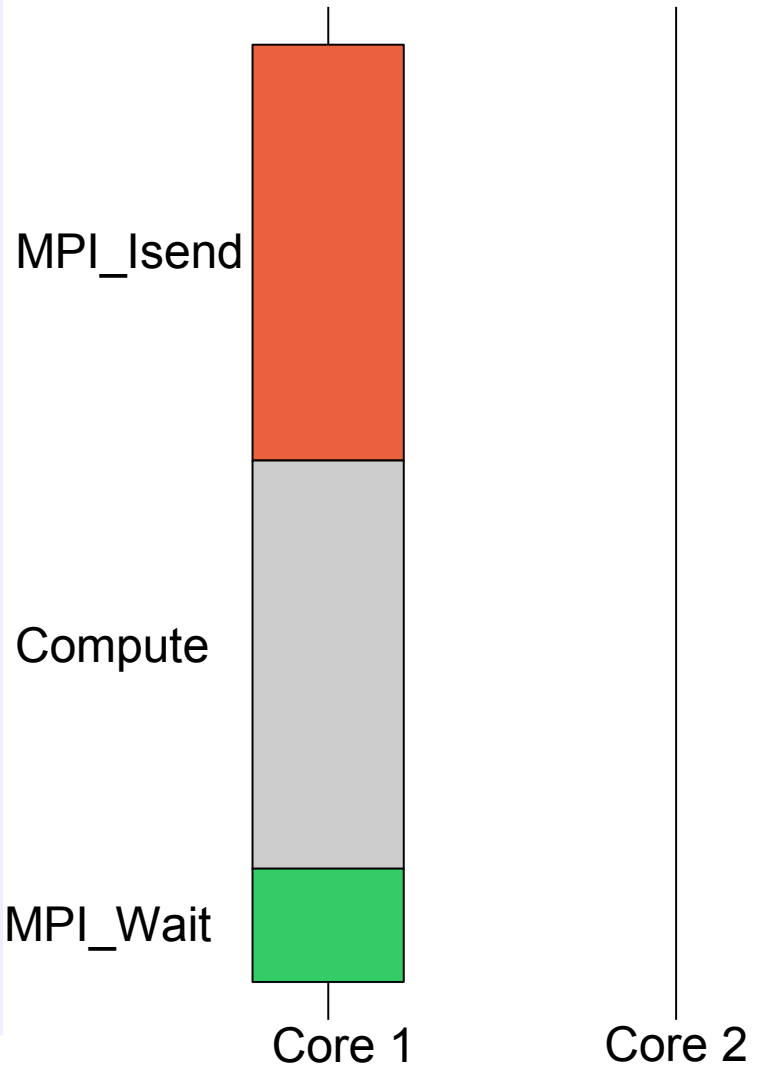


Implementation overview

- A progression thread runs on each core
- Idle cores make communication progress in the background
- Split and spread strategy
  - Offloading message submissions
  - Making rendezvous progress

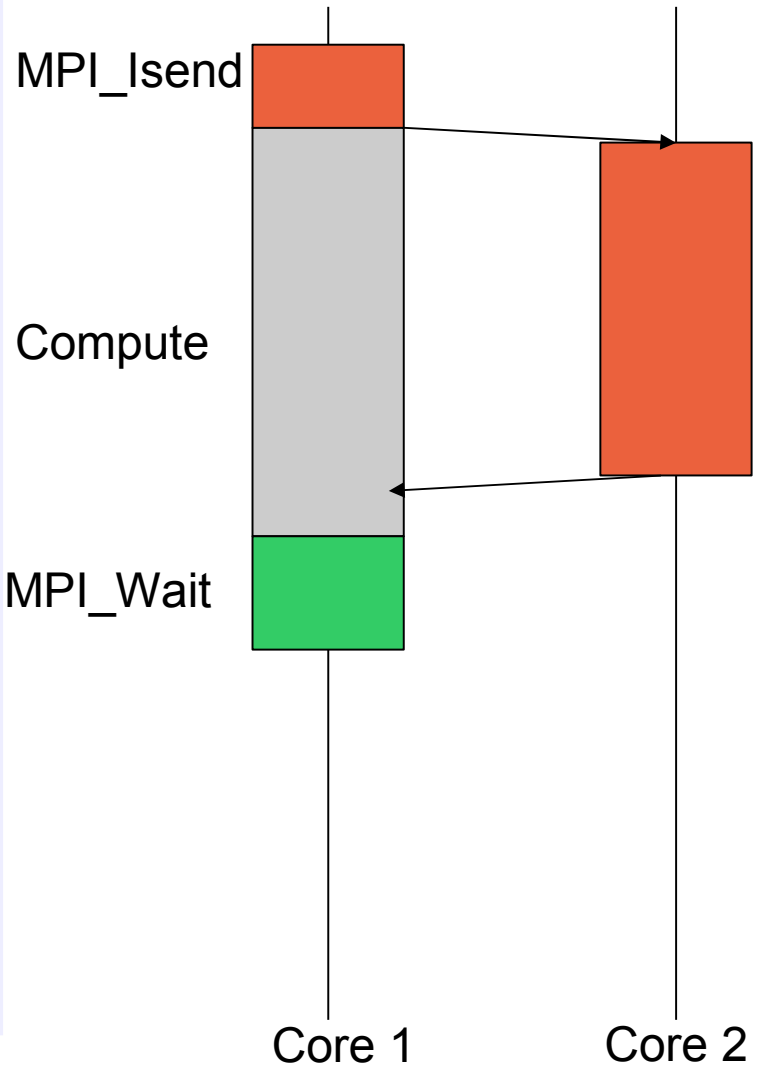
Offloading small messages

- Sending small messages consumes CPU
  - Memcopy required
  - May monopolyze a CPU for dozens of µs
- Even a MPI_Isend can be split
  - Split the non-blocking send
    a) Registering the request
    b) Submitting the request to the NIC
  - Spread the operations on cores



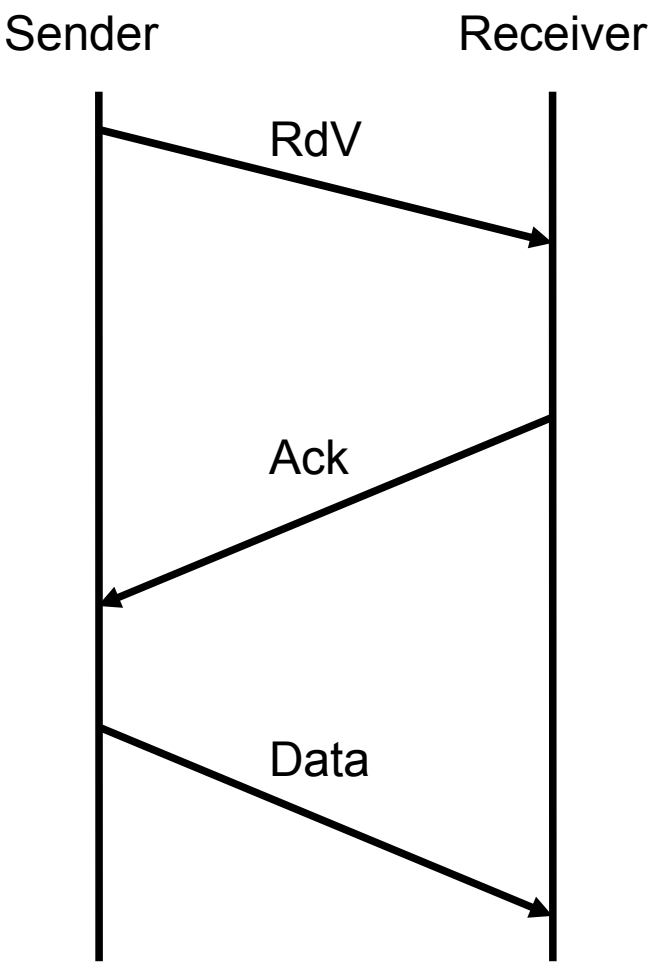
Offloading small messages

- Sending small messages consumes CPU
  - Memcopy required
  - May monopolyze a CPU for dozen of µs
- Even a MPI_Isend can be split
  - Split the non-blocking send
    a) Registering the request
    b) Submitting the request to the NIC
  - Spread the operations on cores



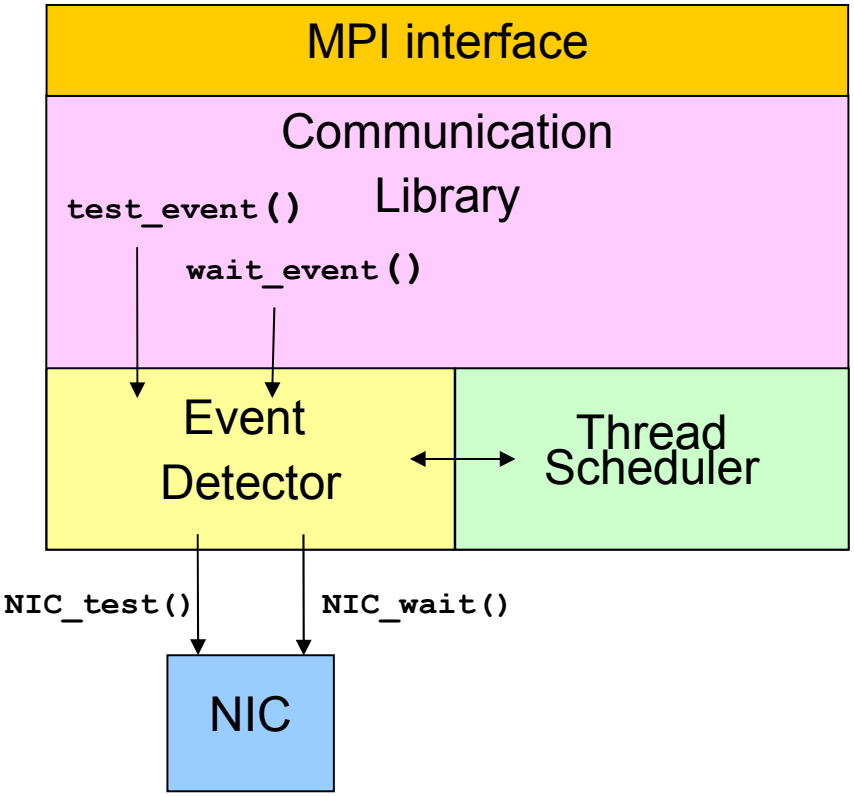
Rendez-vous handshakes progression

- *Rendez-vous* required for large messages
  - Need a high level of reactivity
- Treatments performed on any core
  - Registering the request
  - Sending/receiving the Rendez-vous request
  - Answering/waiting for the acknowledgement
  - Sending/receiving the data



Implementation

- Model implemented in the PM2 software suite
  - NewMadeleine communication library
    - Complex optimizations on communication flows
    - Mad-MPI : light MPI implementation
  - Marcel user-level thread library
  - PIOMan event detector
    - Provides an event detection service
    - Tightly works with Marcel and NewMadeleine

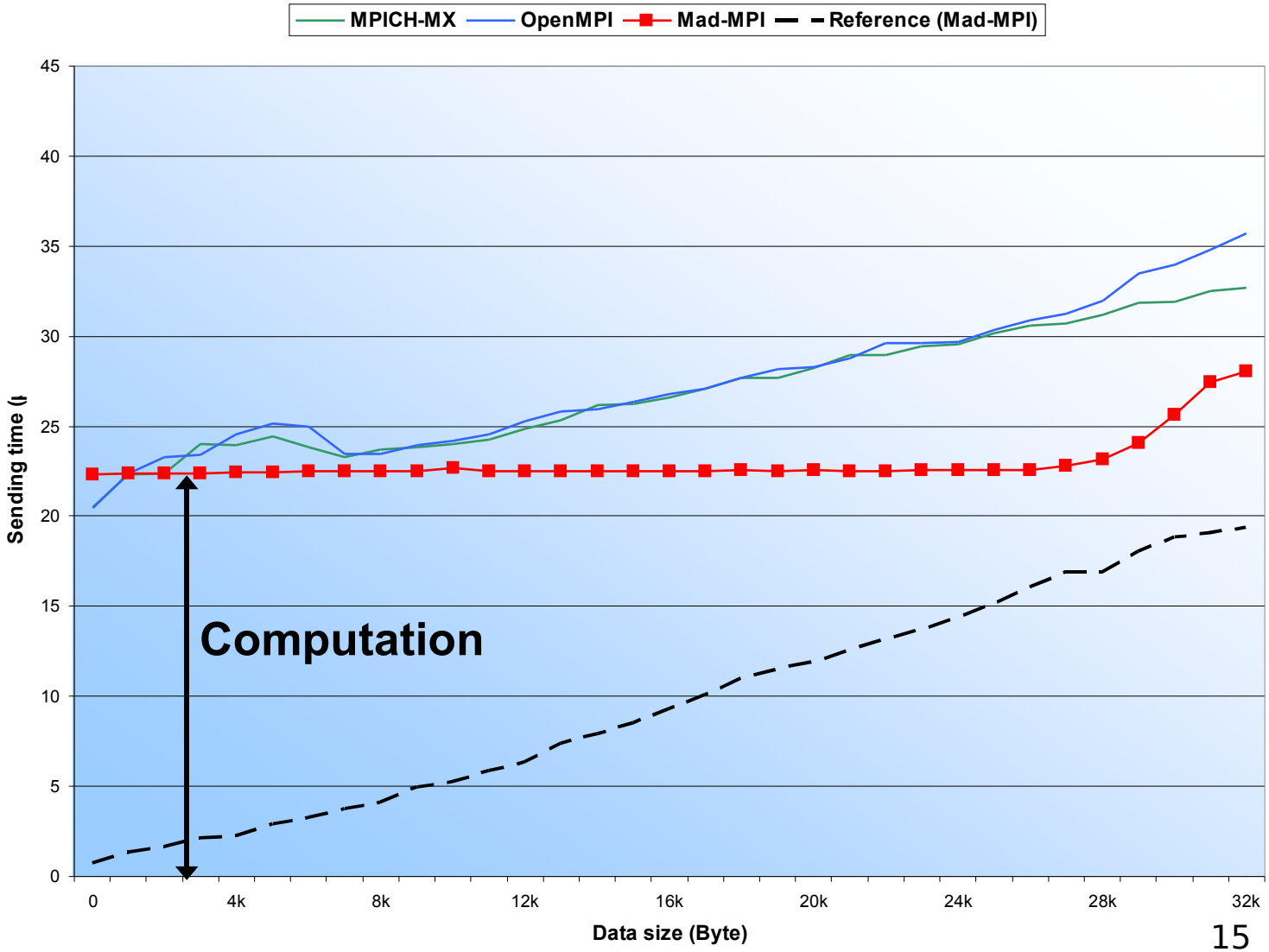


Offloading small messages submission

```
t1=MPI_Wtime();
MPI_Isend();
Compute();
MPI_Wait();
t2=MPI_Wtime();
```

Dual Quad-core Xeons
Myri-10G NICs

MPICH-MX 1.2.7
OpenMPI 1.2.5
Multithreaded Mad-MPI

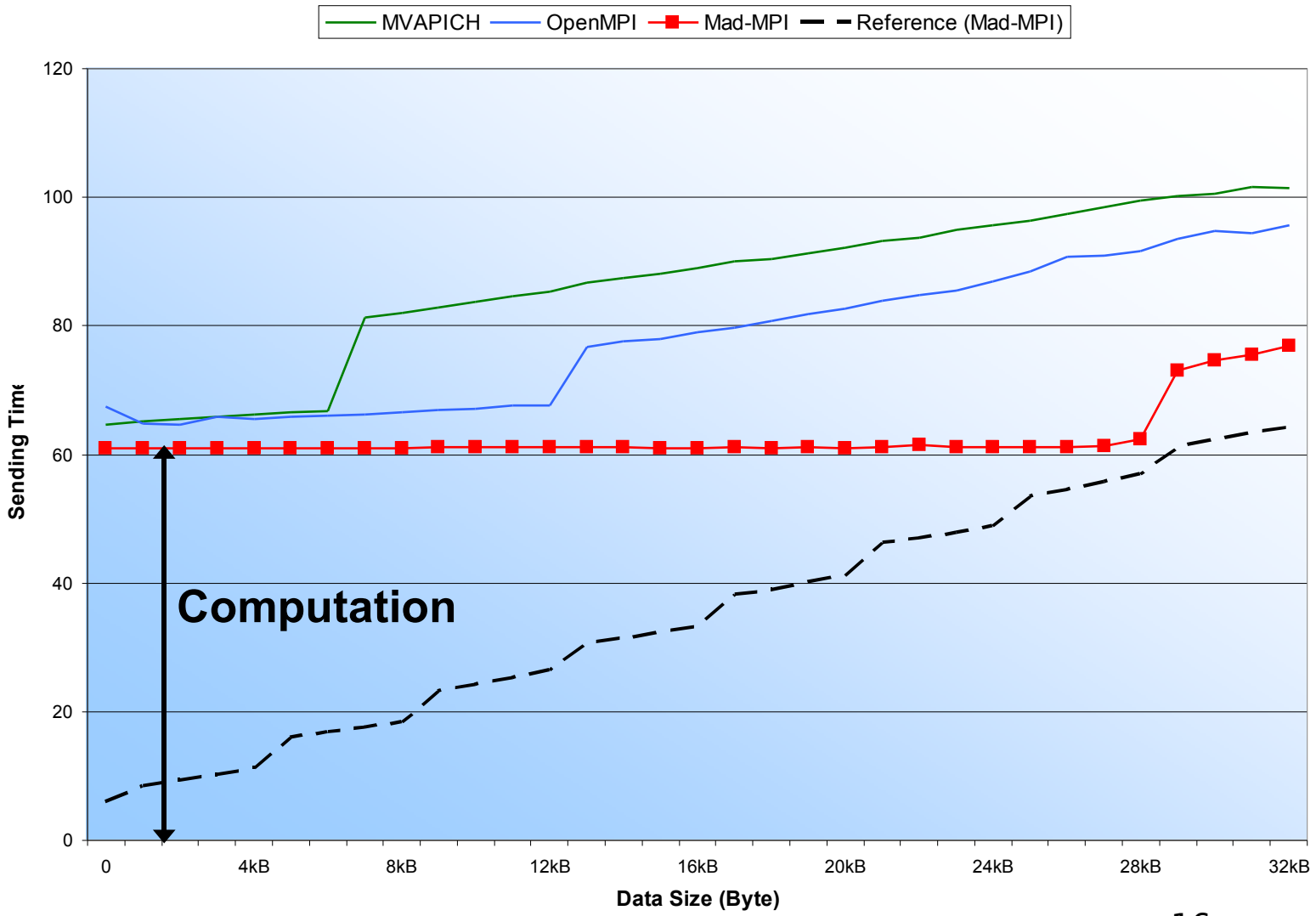


Offloading small messages submission

```
t1=MPI_Wtime();
MPI_Isend();
Compute();
MPI_Wait();
t2=MPI_Wtime();
```

Dual Dual-core Xeons
IB NICs

MVAPICH2 1.0.2
OpenMPI 1.2.5
Multithreaded Mad-MPI

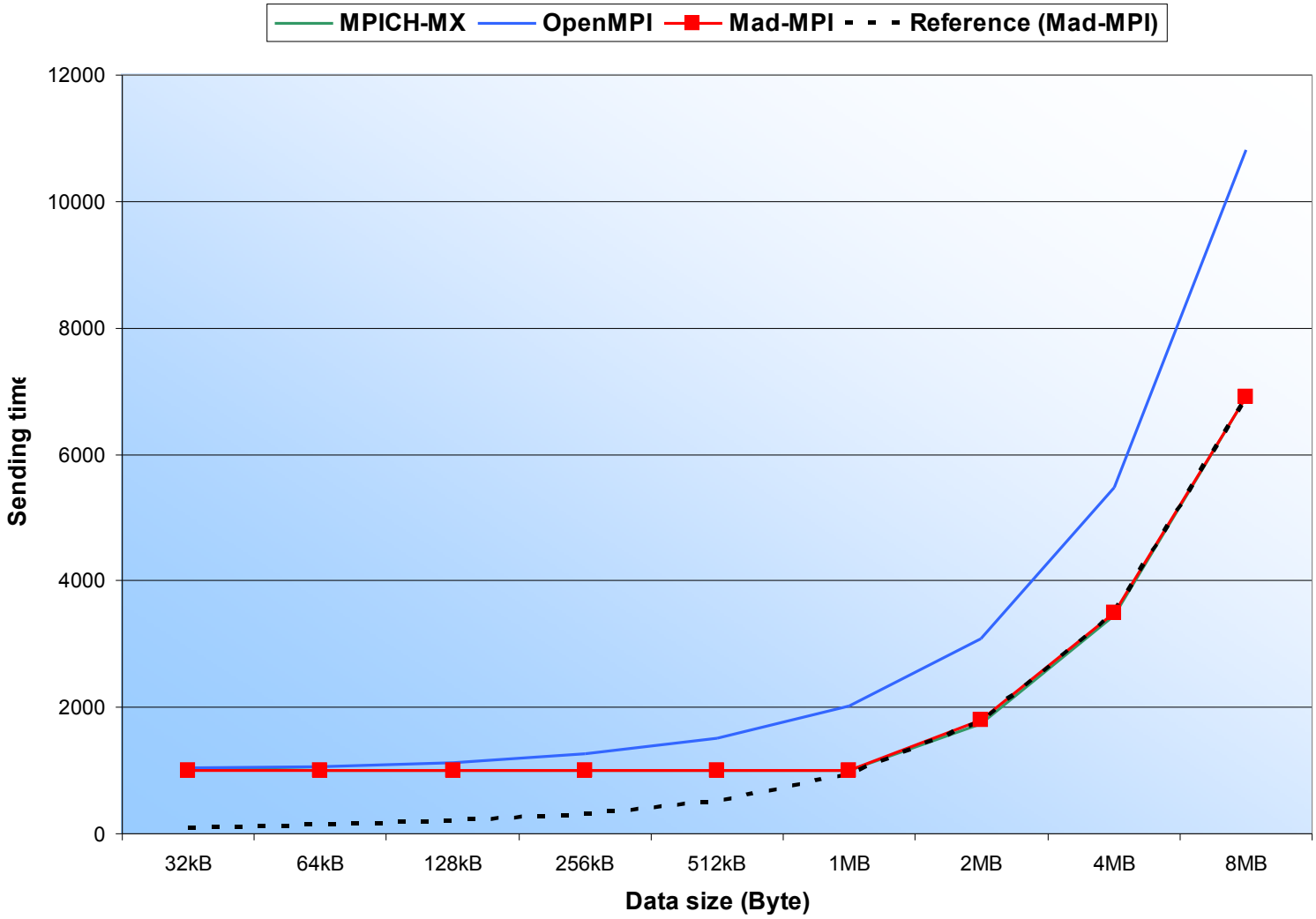


Rendezvous handshake progression

```
t1=MPI_Wtime();
MPI_Isend();
Compute();
MPI_Wait();
t2=MPI_Wtime();
```

Dual Quad-core Xeons
Myri-10G NICs

MPICH-MX 1.2.7
OpenMPI 1.2.5
Multithreaded Mad-MPI

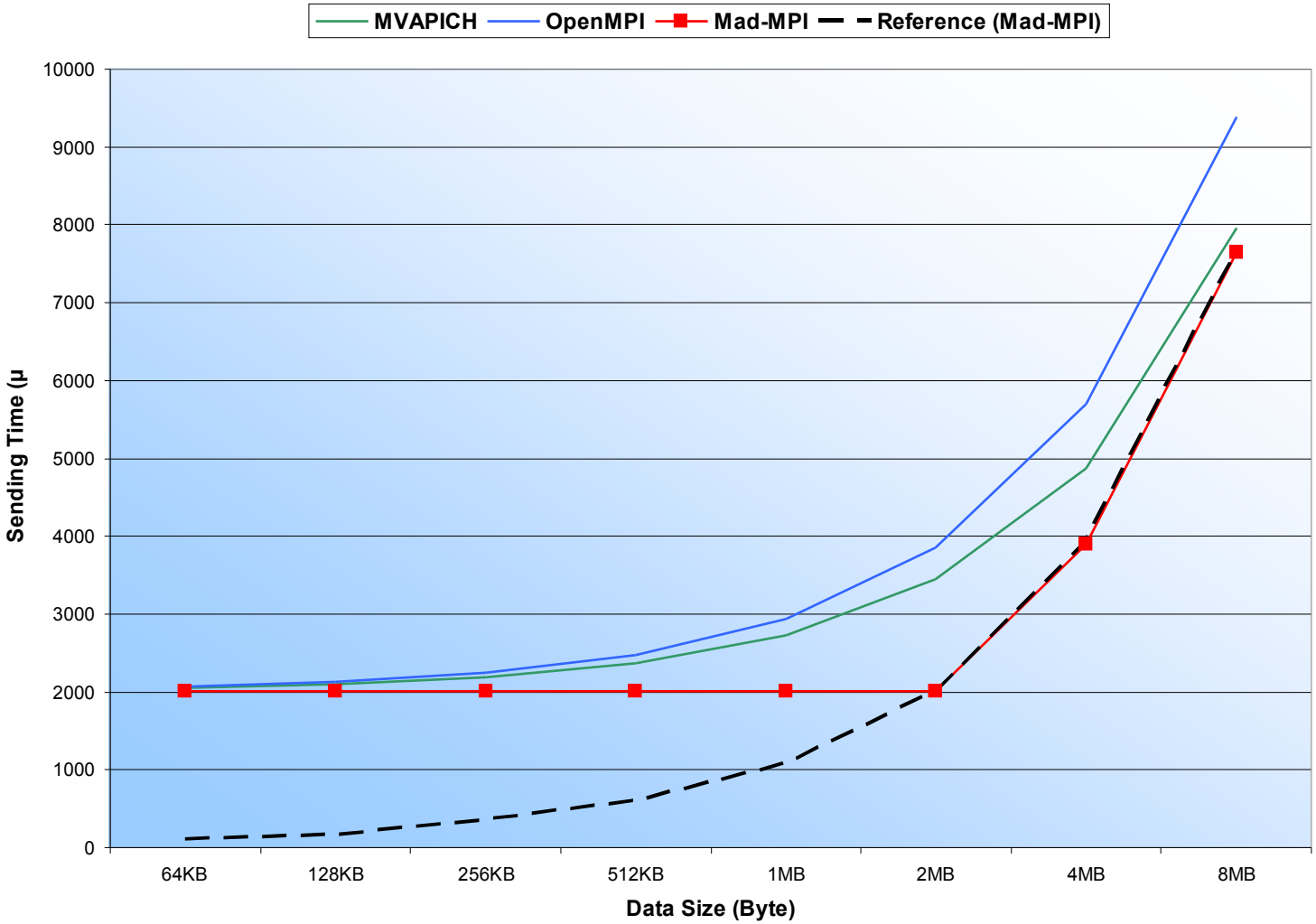


Rendezvous handshake progression

```
t1=MPI_Wtime();
MPI_Isend();
Compute();
MPI_Wait();
t2=MPI_Wtime();
```

Dual Dual-core Xeons
IB NICs

MVAPICH2 1.0.2
OpenMPI 1.2.5
Multithreaded Mad-MPI



Conclusion

- Hardware evolution requires a software evolution
  - Taking advantage of multithreading instead of suffering from it
- Event-driven parallel communication engine
  - Load-balancing of communication processing
  - Efficient resource multiplexing
  - « Real » asynchronous operations
- Implementation evaluated
  - Full overlap of communication and computation

Future works

- Copy offloading strategies
  - Give the choice to the application
- Offloading of packet scheduling
  - Computing more complex optimizations
- Integration in MPICH2-NewMadeleine
  - Benchmarking on « real-life » applications

http://runtime.futurs.inria.fr/pioman/