S. Di Girolamo, P. Jolivet, K. D. Underwood, T. Hoefler

Exploiting Offload Enabled Network Interfaces

How to program QsNet?

How to offload in Portals 4?

How to offload in libfabric?

Lossy Networks
Ethernet

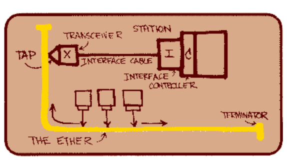
1980's

Lossless N
RDM

200

vice Programming
Offload

2020's



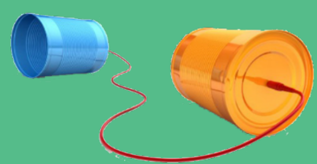
Mellanox TECHNOLOGIES

intel

portals

OPENFABRICS ALLIANCE

Communications
(non-blocking)



Computations

Dependencies

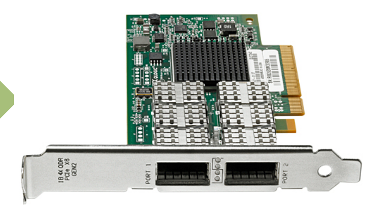
recv → comp → send

EXPRESS

```
L0: recv a from P1;
L1: b = compute f(buff, a);
L2: send b to P1;
L0 and CPU-> L1
L1 -> L2
```

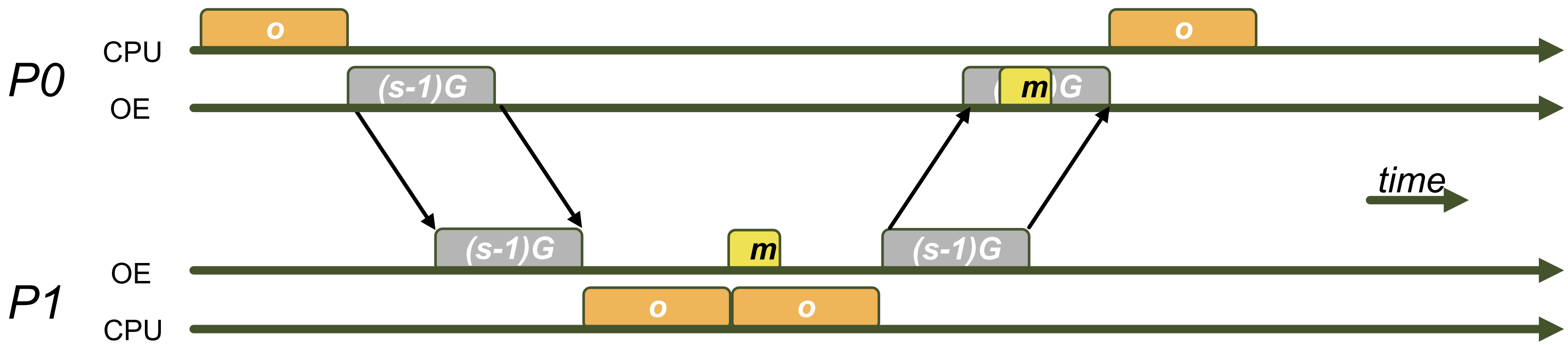
CPU

OFFLOAD



Offload Engine

Performance Model




```
P0{
  L0: recv m1 from P1;
  L1: send m2 to P1;
}
```

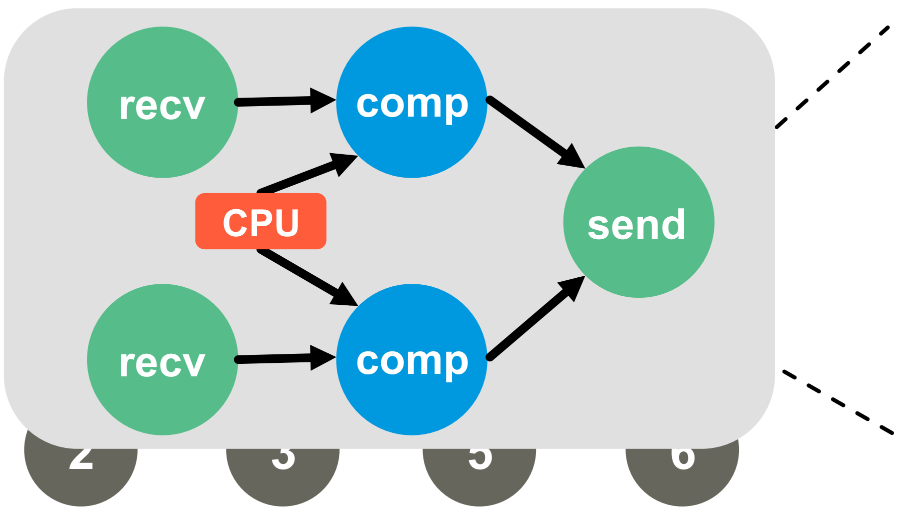
```
P1{
  L0: recv m1 from P1;
  L1: send m2 to P1;
  L0 -> L1
}
```

[1] A. Alexandrov et al. "LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation.“, Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures. ACM, 1995.

Offloading Collectives

A collective operation is fully offloaded if:

1. No synchronization is required in order to start the collective operation
2. Once a collective operation is started, no further CPU intervention is required in order to progress or complete it.




```
L0: recv msg1 from 5;
L1: recv msg2 from 6;
L3: res = compute f(res, msg1);
L4: res = compute f(res, msg2);
L5: send res to 0;
L1 and CPU -> L3
L2 and CPU -> L4
L3 and L4 -> L5
```

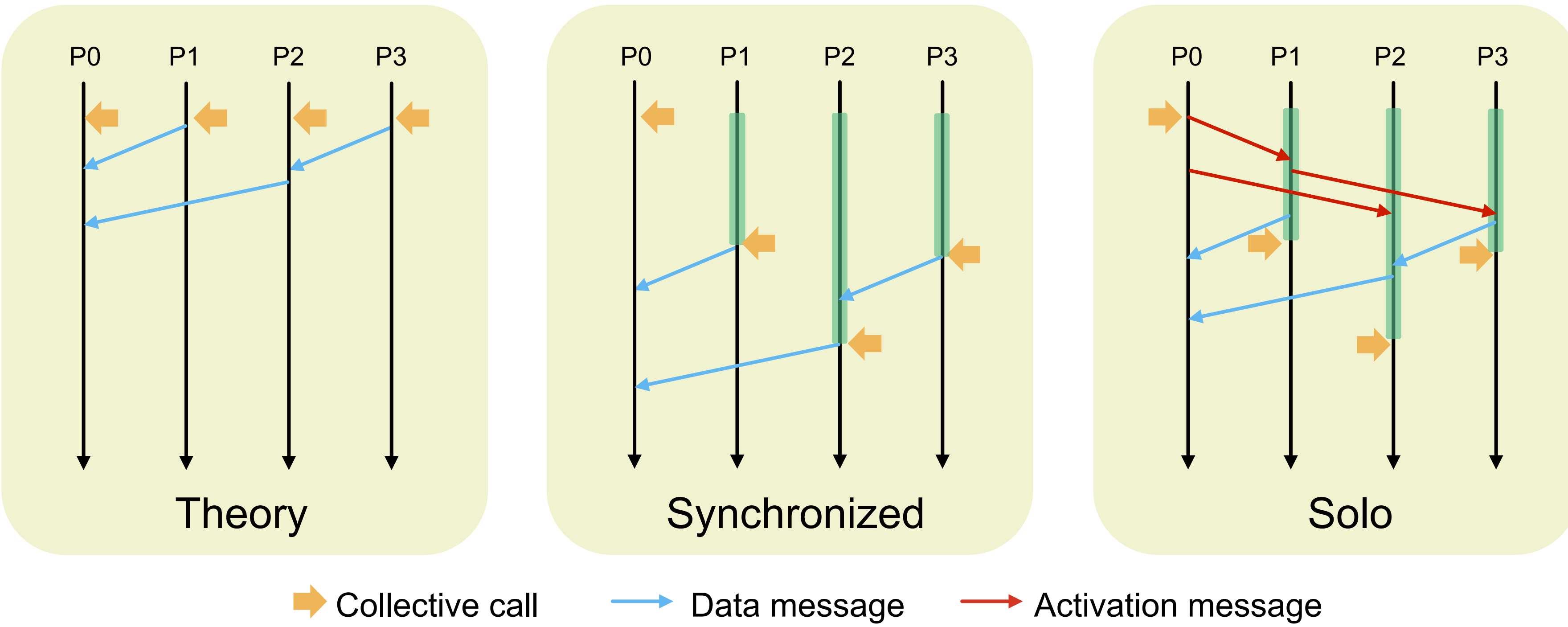
Definition. *A <u>schedule</u> is a local dependency graph describing a partial ordered set of operations.*

Definition. *A <u>collective communication</u> involving n nodes can be modeled as a set of schedules $S = S_1$, ..., S_n where each node i participates in the collective executing its own schedule S_1*

> *Asynchronous algorithms, with their ability to tolerate memory latency, form an important class of algorithms for modern computer architectures.*

Edmond Chow et al., “Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs”, High Performance Computing. Springer International Publishing, 2015.

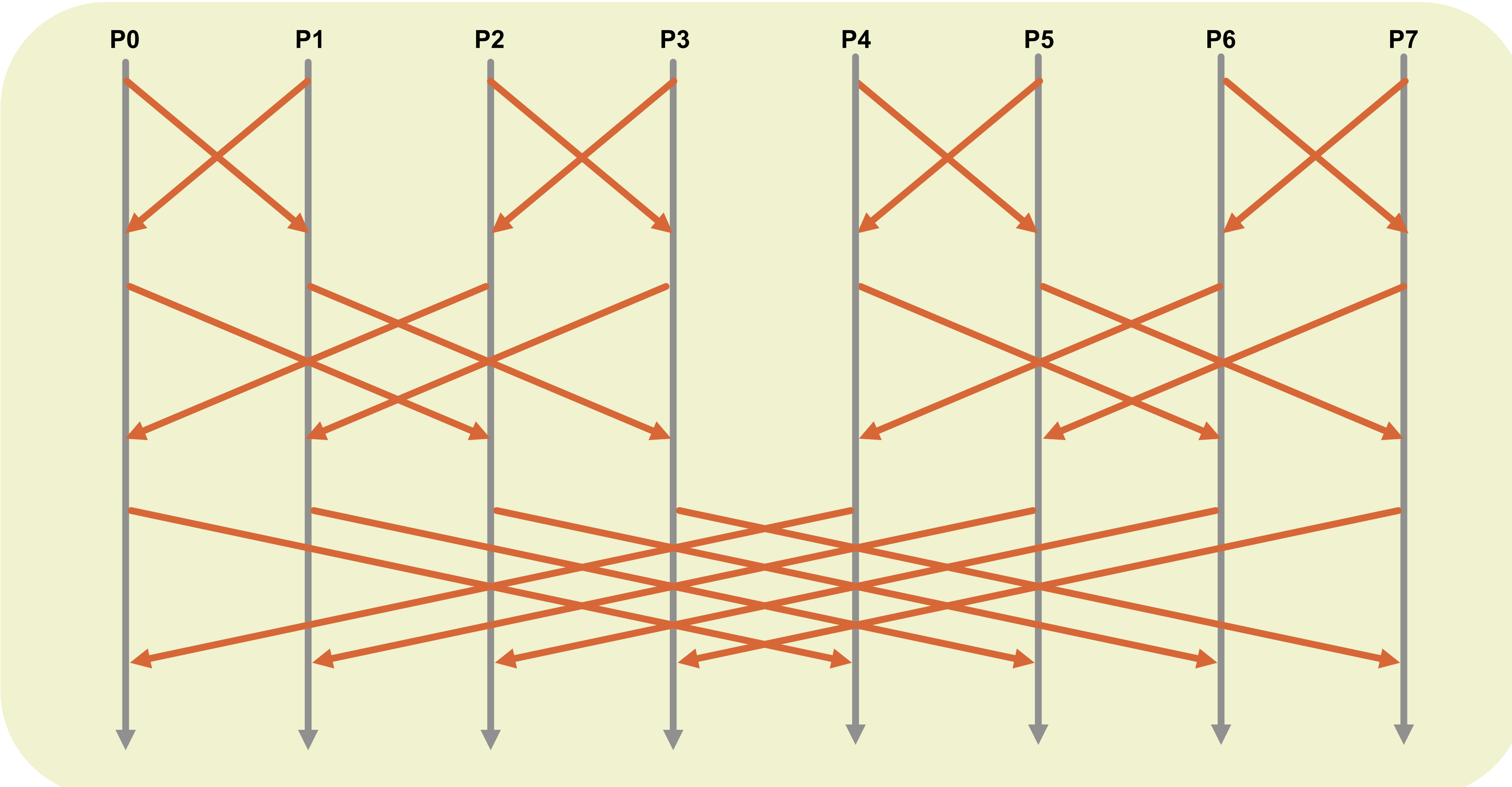
Solo Collectives



- **Synchronized collectives lead to the synchronization of the participating nodes**
- **A solo collective starts its execution as soon as one node (the initiator) starts its own schedule**

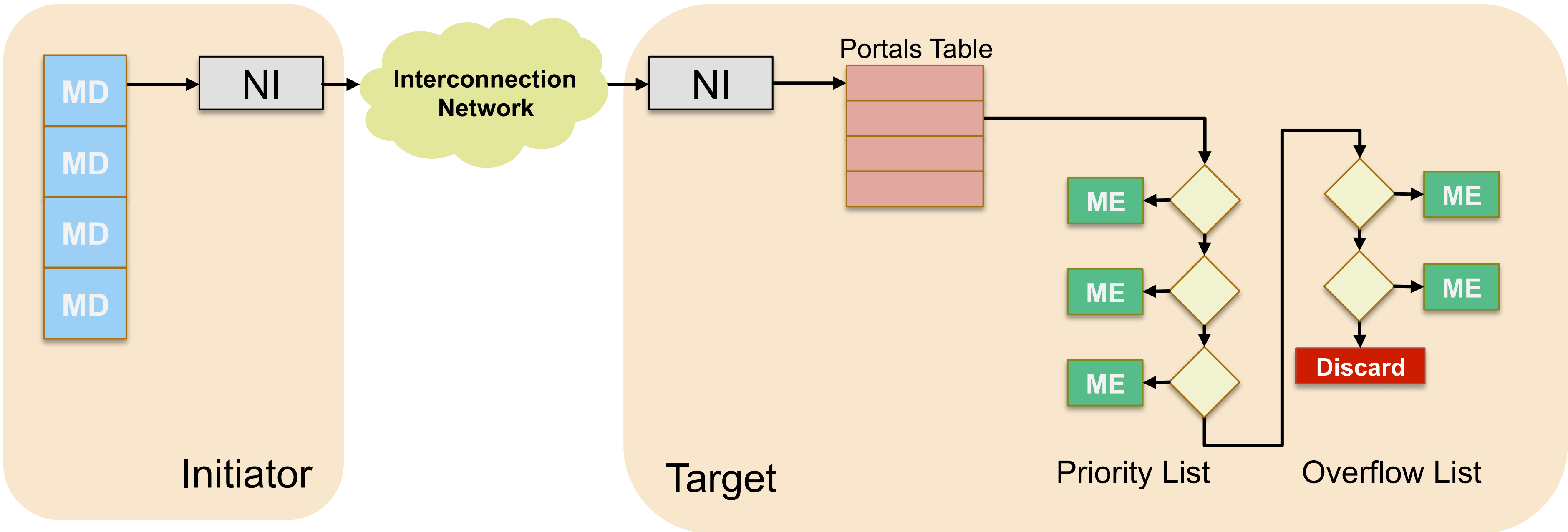
Solo Collectives: Activation

- **Root-Activation: the initiator is always the root of the collective**
- **Non-Root-Activation: the initiator can be any participating node**



A Case Study: Portals 4

- **Based on the one-sided communication model**
- **Matching/Non-Matching semantics can be adopted**

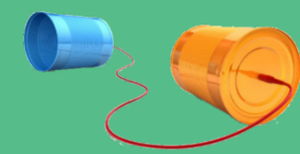


[2] "The Portal 4.0.2 Network Programming Interface"

A Case Study: Portals 4

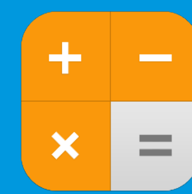
Communication primitives

- Put/Get operations are natively supported by Portals 4
- One-sided + matching semantic



Atomic operations

- Operands are the data specified by the MD at the initiator and by the ME at the target
- Available operators: *min, max, sum, prod, swap, and, or, …*



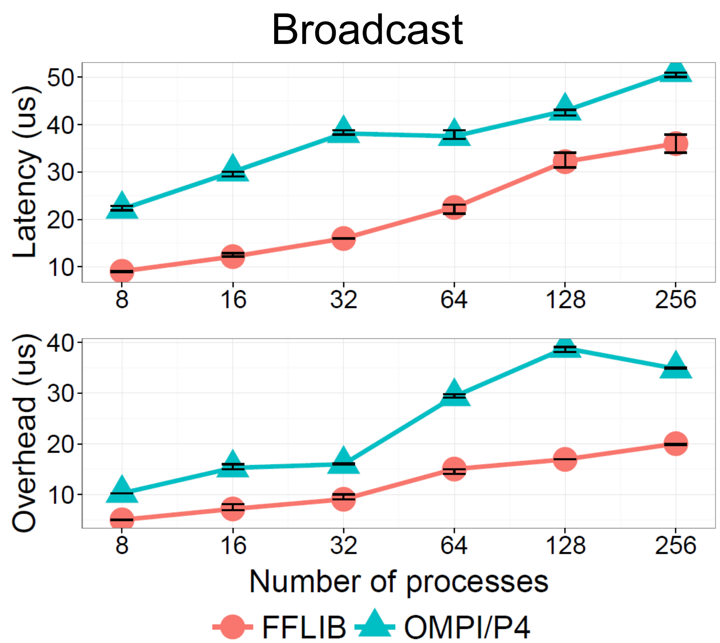
Counters

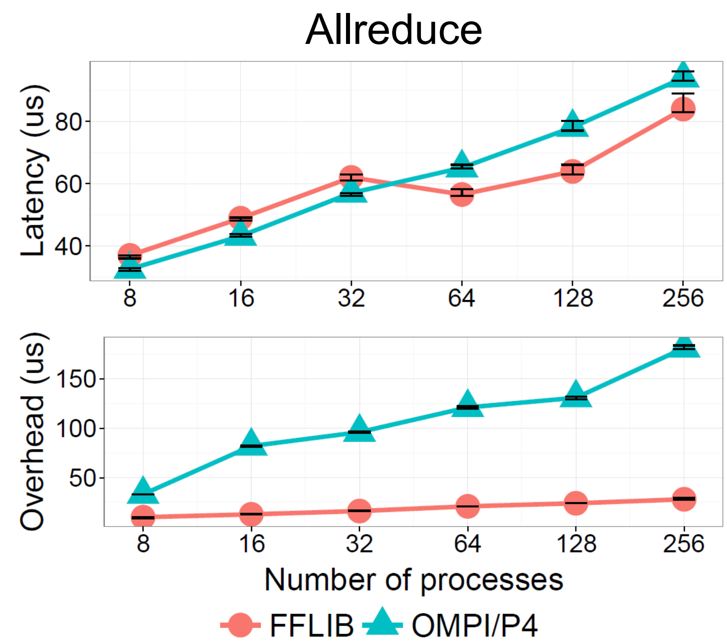
- Associated with MDs or MEs
- Count specific events (e.g., operation completion)

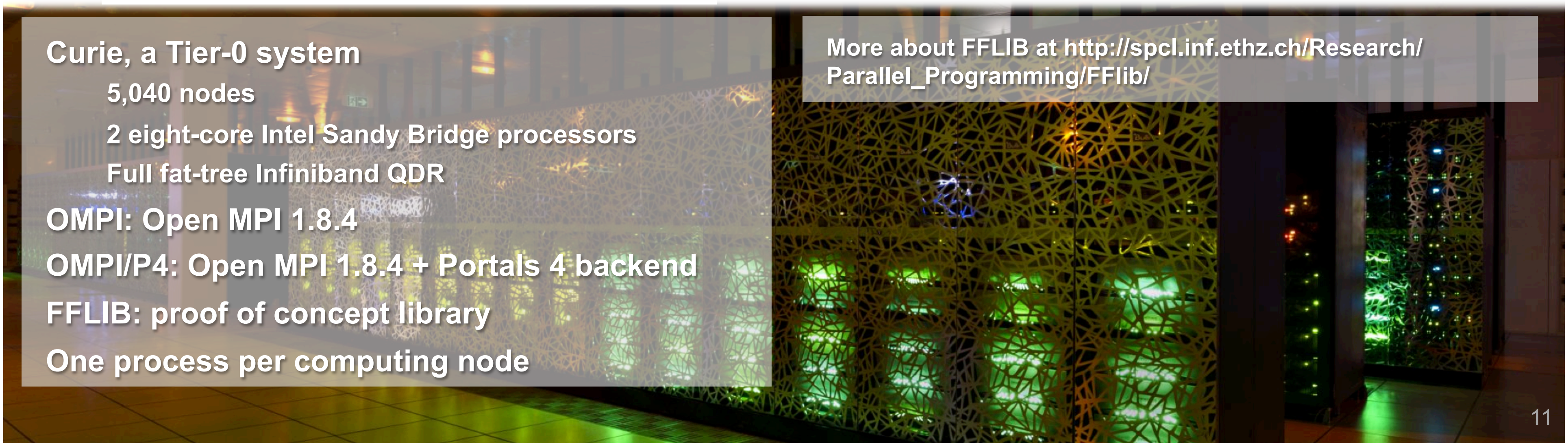
Triggered operations

- Put/Get/Atomic associated with a counter
- Executed when the associated counter reaches the specified threshold

Experimental results



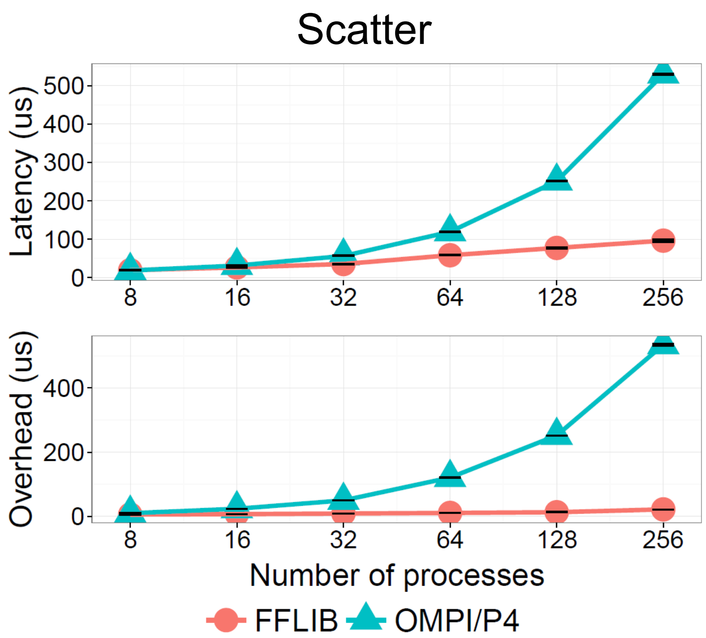


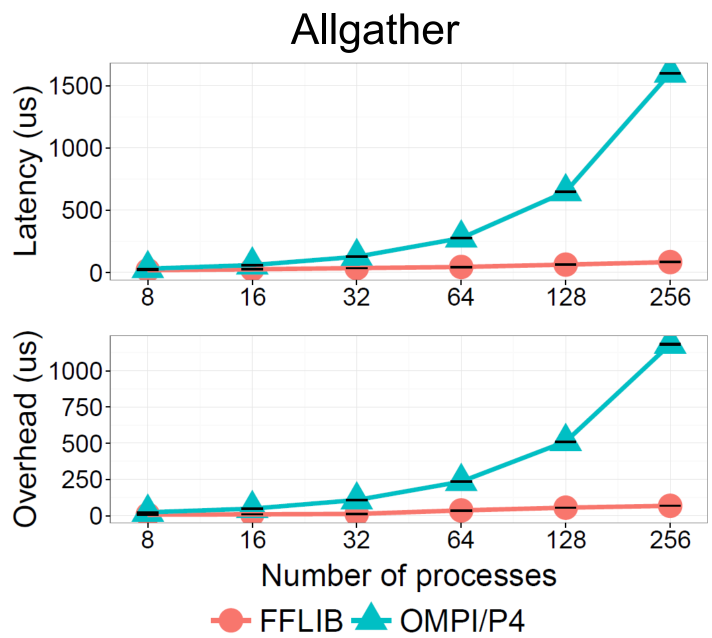


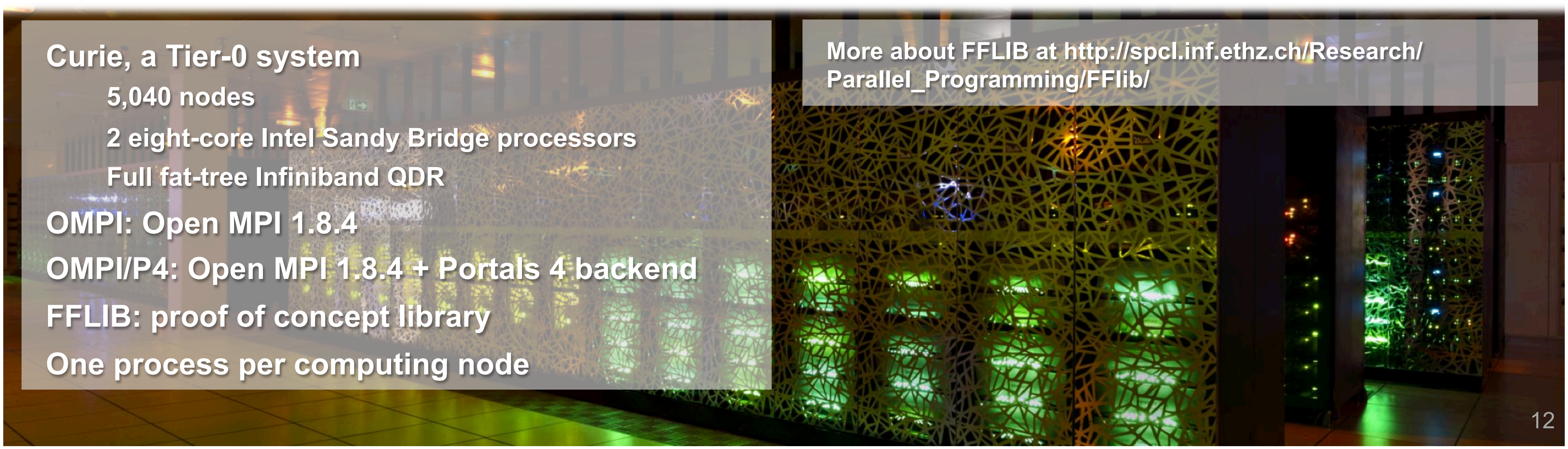
- Curie, a Tier-0 system
  - 5,040 nodes
  - 2 eight-core Intel Sandy Bridge processors
  - Full fat-tree Infiniband QDR
- OMPI: Open MPI 1.8.4
- OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
- FFLIB: proof of concept library
- One process per computing node

More about FFLIB at http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/

Experimental results





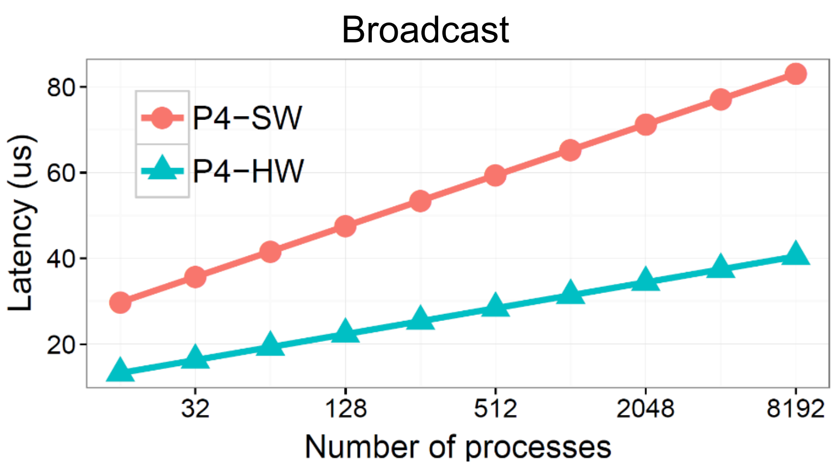


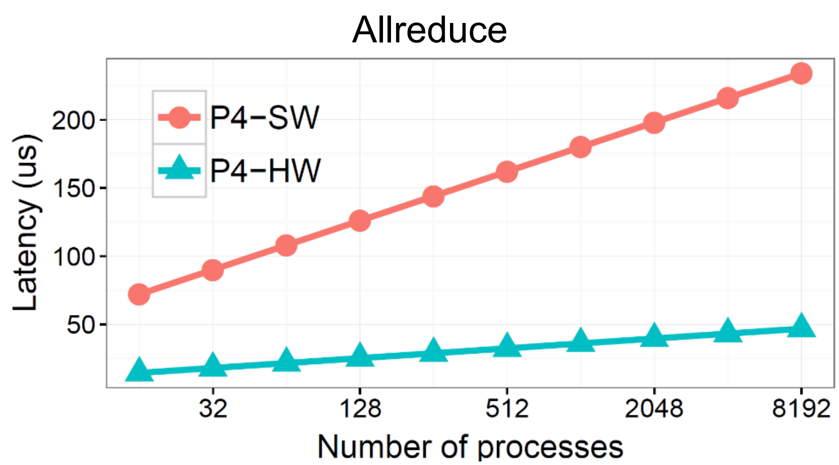
- Curie, a Tier-0 system
  - 5,040 nodes
  - 2 eight-core Intel Sandy Bridge processors
  - Full fat-tree Infiniband QDR
- OMPI: Open MPI 1.8.4
- OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
- FFLIB: proof of concept library
- One process per computing node

More about FFLIB at http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/

Simulations

- **Why?** To study offloaded collectives at large scale
- **How?** Extending the LogGOPSim to simulate Portals 4 functionalities

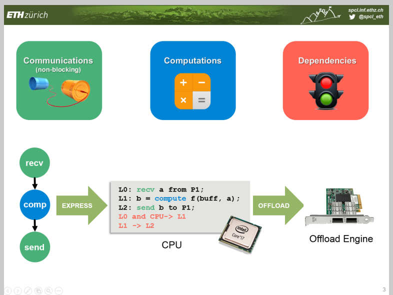


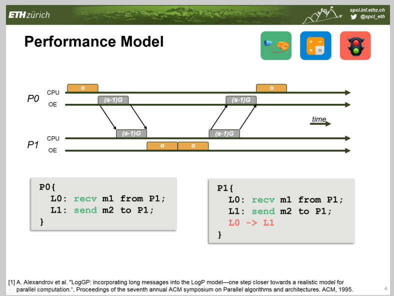


	L	*o*	*g*	*G*	*m*
P4-SW	$5\mu s$	$6\mu s$	$6\mu s$	$0.4ns$	$0.9ns$
P4-HW	$2.7\mu s$	$1.2\mu s$	$0.5\mu s$	$0.4ns$	$0.3ns$ [4]

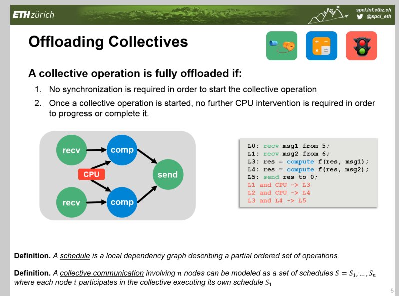
[3] T. Hoefler, T. Schneider, A. Lumsdaine: "LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model", In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (HPDC '10). ACM, 2010.
[4] Underwood et al., "Enabling Flexible Collective Communication Offload with Triggered Operations", *IEEE 19th Annual Symposium on High Performance Interconnects* (HOTI '11). IEEE, 2011.

Abstract Machine Model

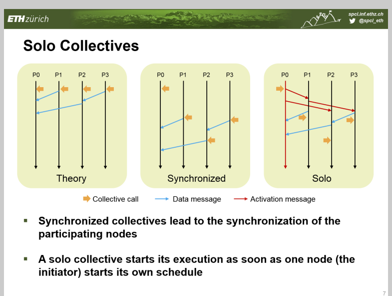


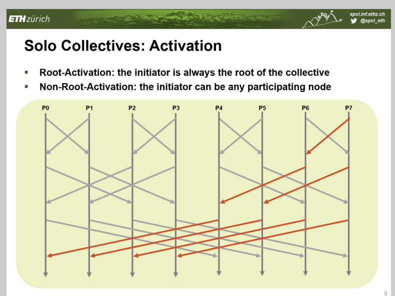


Offloading Collectives

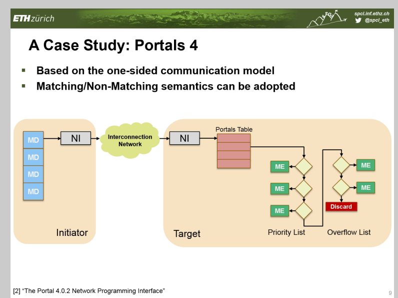


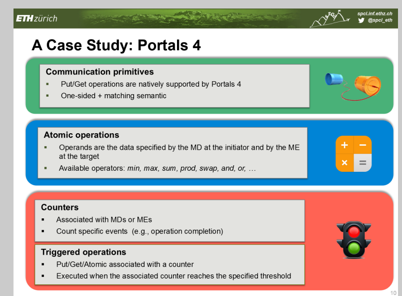
Solo Collectives



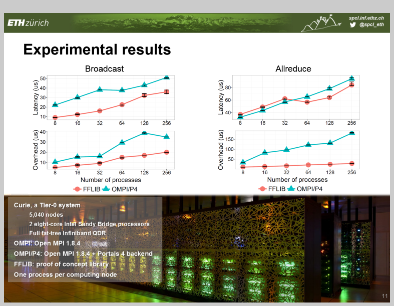


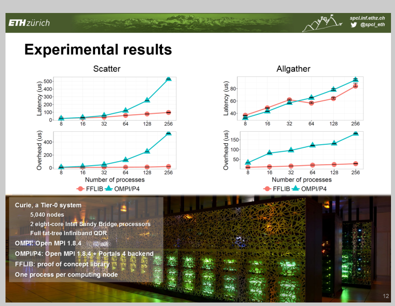
Mapping to Portals 4

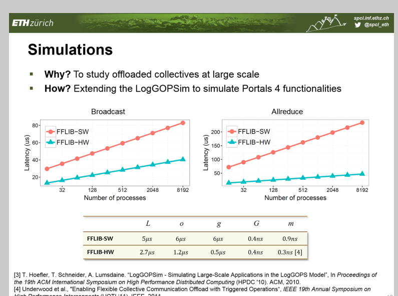




Results



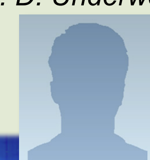




Co-Authors

P. Jolivet

K. D. Underwood



T. Hoefler

Backup slides

Multi-Version Scheduling

- **Enables the multiple asynchronous execution of the same collective**
  - It allows the pre-posting of k versions of the same schedule
  - Each version can have its own buffers
  - Each version can be activated by a different node

- **Implemented as FIFO queue of schedules**
  - Only one scheduled enabled at each time: S_i
  - When S_i is activated, the next in the queue S_{i-1} must be enabled

$$\bigvee_{op_k \in I_i} op_k \rightarrow op_j \quad \forall op_j \in I_{i-1}$$

Independent operations of schedule S_{i-1}

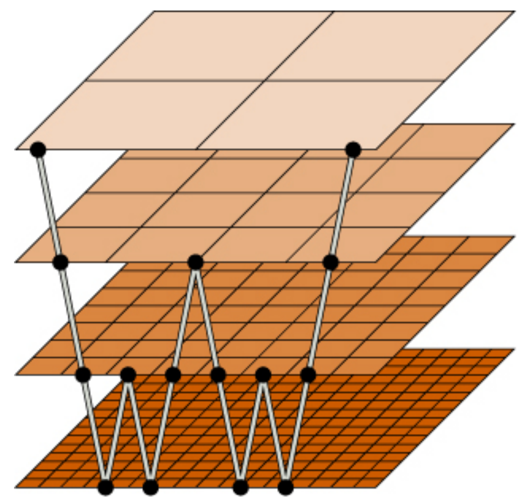
Independent operations of schedule S_i

Use Case: Multigrid

- Multilevel preconditioners are a dominant paradigm for large-scale partial differential equation simulations
  - Theoretically optimal
  - High communication and synchronization overheads
- Two-grid hierarchy
  - Only one process perform the coarsening

```
P0::
  gather()
  coarse_work()
  scatter()
```

```
Pi, i>0::
  work()
  gather()
  scatter()
```



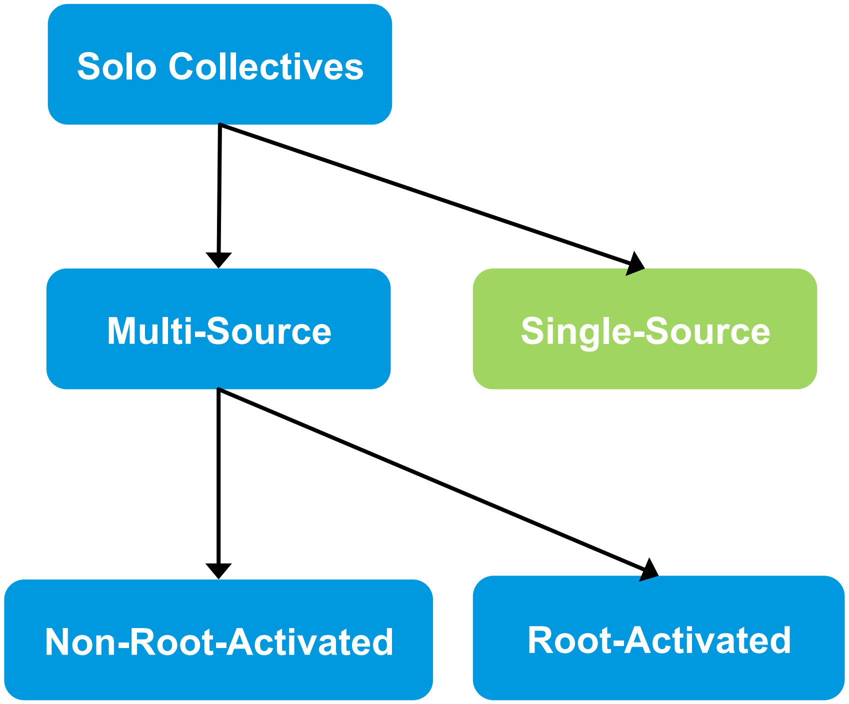
- Simple benchmark implementing the communication patter
  - The introduction of solo-collective led to a 1.5x improvement in the completion time
  - A full benchmark would require a study of the convergence rate for such fully asynchronized approach

Solo Collectives

- **Collective communications lead to the pseudo-synchronization of the participating nodes**
  - Each node starts its own schedule at time t_i
  - The collective communication will terminate at a time $t_s \geq max_i(t_i)$

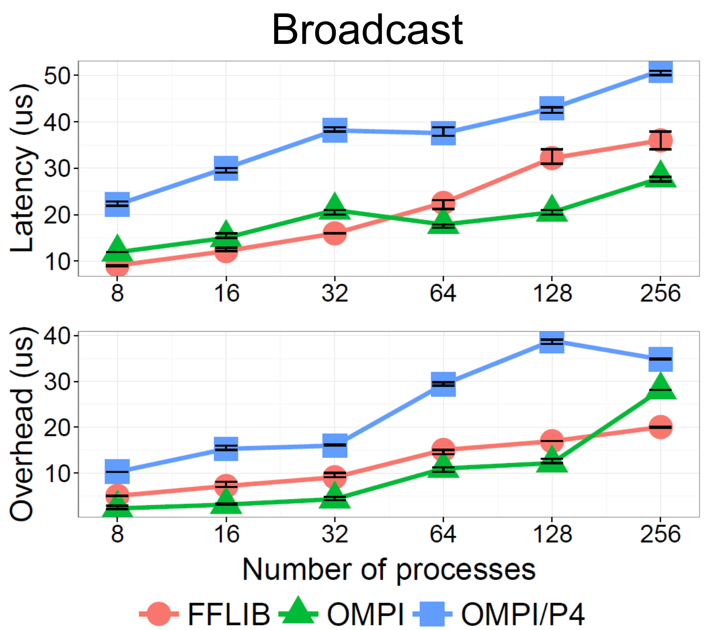
- **A solo collective starts its execution as soon as one node, the initiator, starts its own schedule**
  - The schedule of other nodes is asynchronously activated
  - The initiator starts its schedule at time t_{init}
  - The collective communication will terminate at a time $t_a \geq t_{init} + \epsilon$
  - The term ϵ models the activation time: $\epsilon \leq \max(\epsilon_i)$

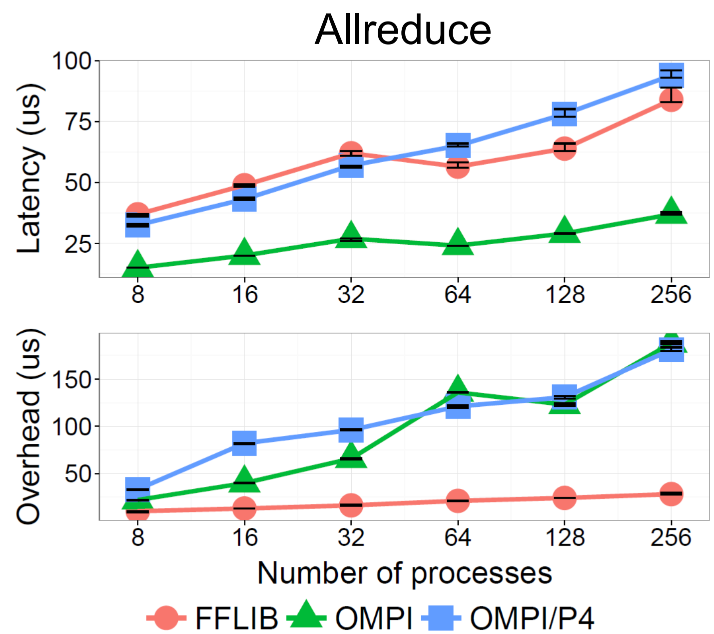
Solo Collectives: activation

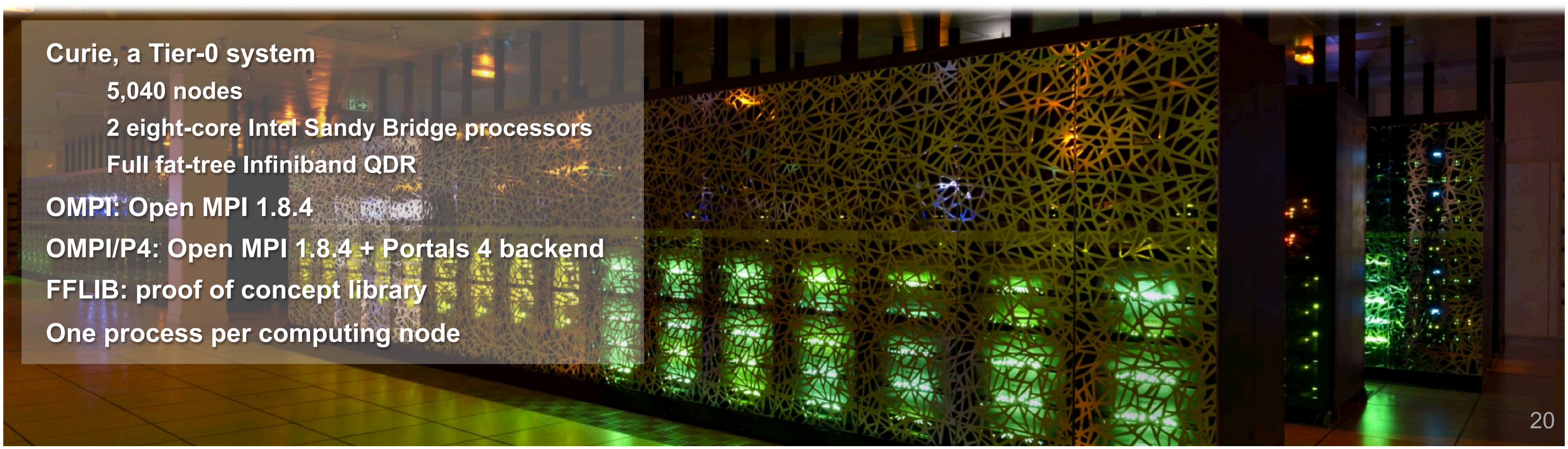


- One active node
- No activation cost
- e.g., broadcast, scatter

Experimental results

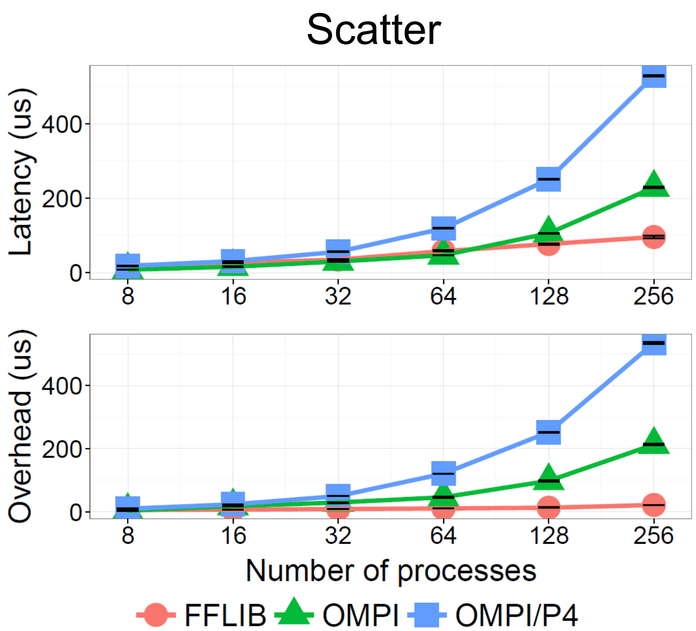


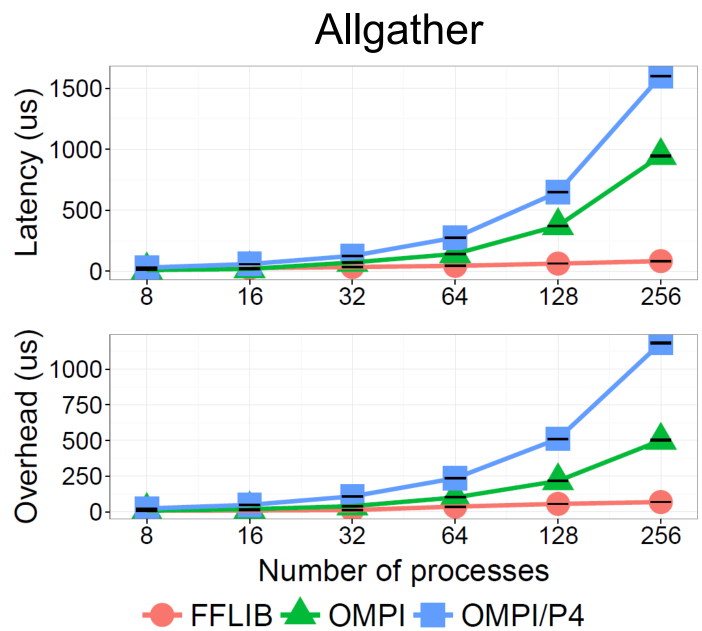




- Curie, a Tier-0 system
  - 5,040 nodes
  - 2 eight-core Intel Sandy Bridge processors
  - Full fat-tree Infiniband QDR
- OMPI: Open MPI 1.8.4
- OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
- FFLIB: proof of concept library
- One process per computing node

Experimental results

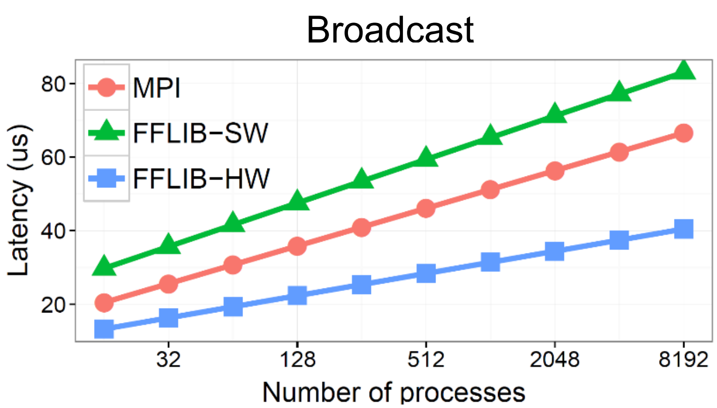


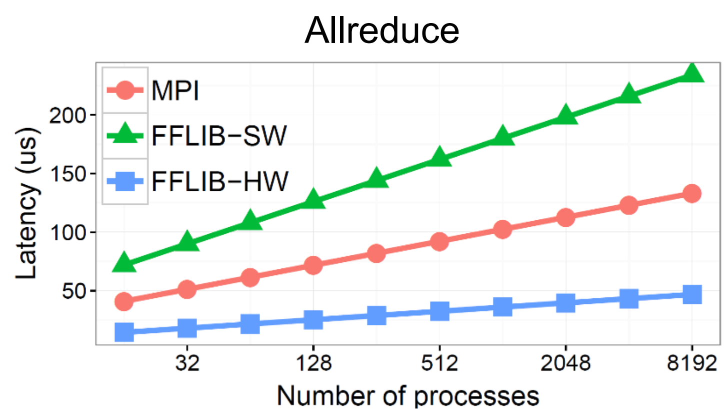


- Curie, a Tier-0 system
  - 5,040 nodes
  - 2 eight-core Intel Sandy Bridge processors
  - Full fat-tree Infiniband QDR
- OMPI: Open MPI 1.8.4
- OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
- FFLIB: proof of concept library
- One process per computing node

Simulations

- **Why?** To study offloaded collectives at large scale
- **How?** Extending the LogGOPSim to simulate Portals 4 functionalities





	L	o	g	G	m
OMPI	$2.7\mu s$	$1.2\mu s$	$0.5\mu s$	$0.4ns$	-
OMPI/P4	$5\mu s$	$6\mu s$	$6\mu s$	$0.4ns$	$0.9\mu s$

- FFLIB-HW uses m=**0.3**μs, discussed in [3] to model the incoming message processing time

[2] T. Hoefler, T. Schneider, A. Lumsdaine. “LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model”
[3] Underwood et al., "Enabling Flexible Collective Communication Offload with Triggered Operations"

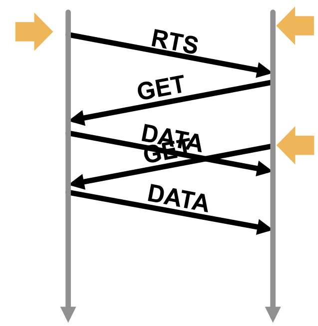
Point-to-Point Protocols

Eager protocol

- Expected messages: *priority list*
- Unexpected messages: *overflow list*

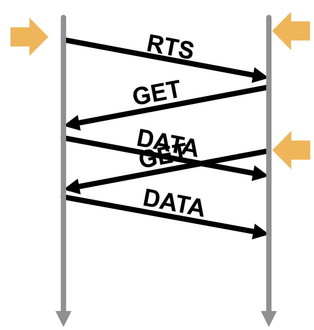
Rendezvous protocol

- No shadow buffers are required
- Synchronization happens among OEs




```
ptl_md_t rts;
ptl_me_t data;
PtlMEAppend(data, NONE);
PtlPut(rts);
```

Sender




```
ptl_md_t data;
ptl_me_t rts;
PtlMEAppend(rts, ct_rts);
PtlTriggeredGet(data, ct_rts, 1);
```

Receiver

Offloading Point-To-Point Protocols

- **P2P communications are building blocks of our abstract model**
  - They can be implemented according with different protocols (i.e., eager, rendezvous)

    Can this protocols be fully offloaded to the OE (e.g., Portals 4-compliant)?

Eager

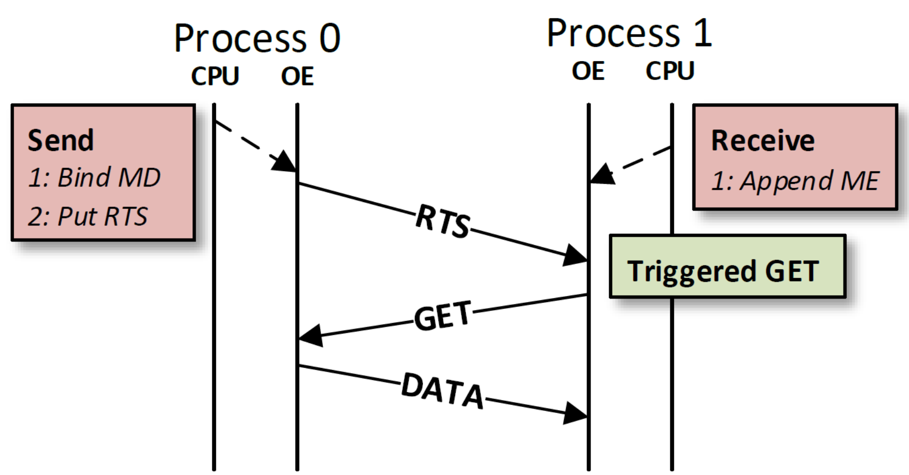
Expected: the message is directly received in the user-provided buffer.

Unexpected: the message is received in a temporary buffer. It will copied in the user-provided one when the receive will be posted.

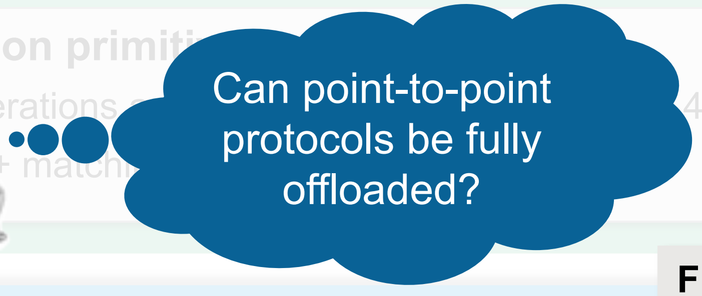
Portals 4 priority and overflow list can be used for a straightforward implementation of this protocol.

Rendezvous

Only the Ready-To-Send (RTS) control message can be unexpectedly received.



A Case Study: Portals 4



Fully offloading:
No synchronization & No CPU intervention

Atomic operations

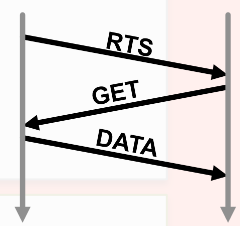
- Operands are the data specified by the MD at the initiator and by the ME at the target

Eager protocol

- Expected messages: *priority list*
- Unexpected messages: *overflow list*

Rendezvous protocol

- No shadow buffers are required
- Synchronization happens among OEs



Counters

- Associated with MDs or MEs
- Count specific events (e.g., operation completion)

Triggered operations

- Put/Get/Atomic associated with a counter
- Executed when the associated counter reaches the specified threshold