Optimised MPI for HPEC applications

- Middleware Libraries and Application Programming Interfaces
- Software Architectures, Reusability, Scalability, and Standards

Systems used for Dataflow applications

- Computing power requirements no evenly spread
- Various transport medium may coexist
- Need for QoS type behaviour
- Performance requirement for I/O between nodes

Requirements

- Need to map process to computing node
- Need to select specific link between process
- Need to implement zero-copy feature

Using MPI in HPEC

- PROs
  - Available on almost every parallel/cluster machine
  - Ensures application code portability
- CONs
  - Made for collective parallel apps, not distributed apps.
  - No choice of communication interface (only know receiver)
  - Does not care about transport medium
  - No control on timeouts
  - Not a communication library
    (no dynamic connection, no select feature)

Zero-copy means memory management

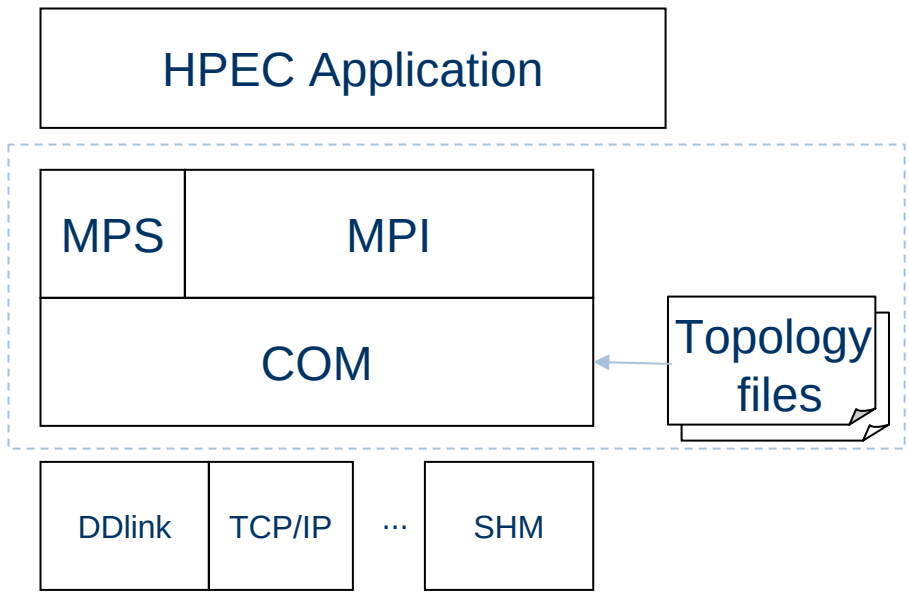
- Same memory buffer used by application and I/O system
- At any given time, buffer must belong to application OR I/O

Zero-copy API

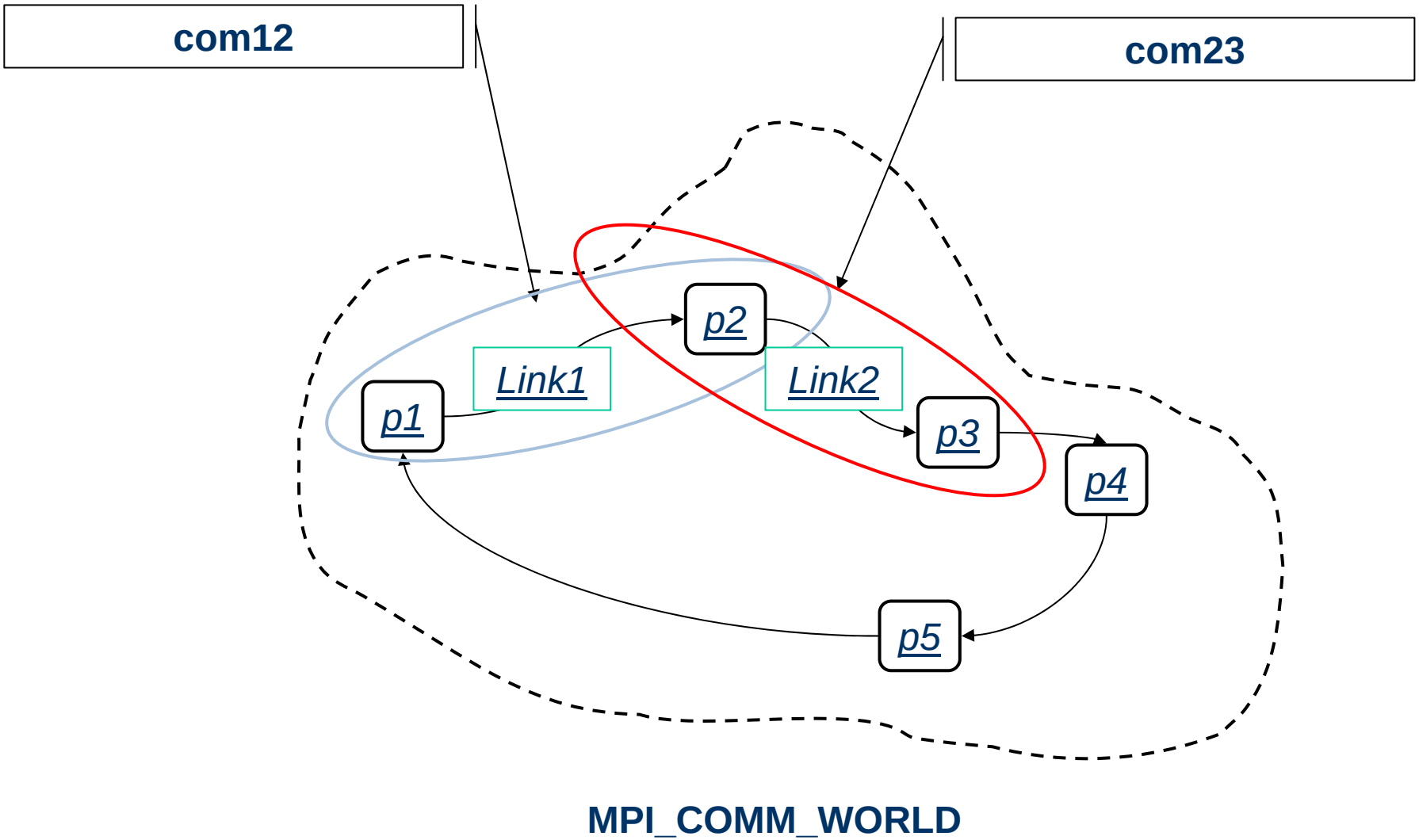
- Buffer Get
  - Data buffer now part of application data
  - Can be used as any private memory
- Buffer Release
  - Data buffer is not to be modified by application any more
  - Can be used by I/O system (likely hardware DMA)

Implementation choice

- MPI Services (MPS) side to side with MPI
  - MPI application source portability
  - Links/Connector relationship
  - Real-Time support
    - Links to select communication channels (~ QoS)
    - Requests timeout support
  - Real zero-copy transfer
    - Buffer Management API (MPS)
  - Heterogeneous machine support
    - Topology files outside application

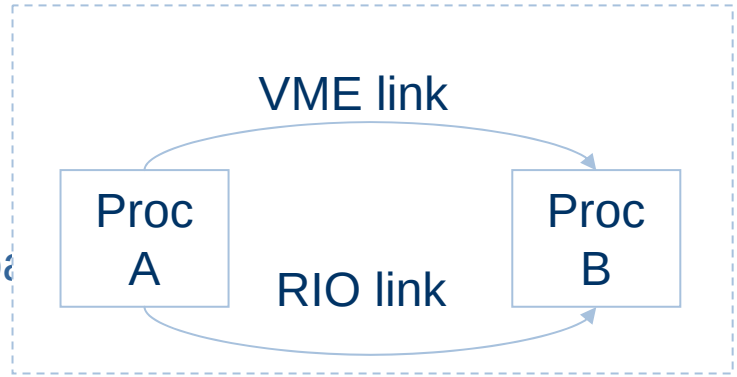


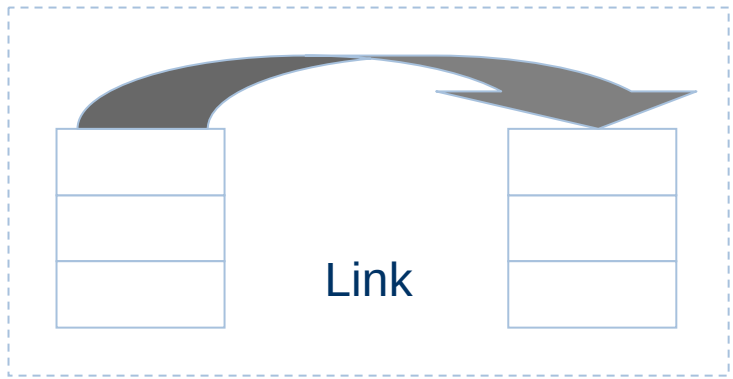
Dedicated MPI Communicator for Zero-copy Link



HPEC System Topology Description

- System topology described outside the application code
- External ASCII files with:
  - Process
    - Process name
    - Process Hardware location (boa
  - Link
    - Link name
    - Medium type (+medium-specific parameters)
    - Buffer size
    - Buffer count





MPS API: processes and links

MPS_Channel_create

(*chan_name, * rendpoint, MPI_Comm *comm, int *lrank, int *rrank) ;

link name ^ ^ | | |

remote end name | | | |

specific communicator for the link *v* | |

my rank in new communicator *v* |

remote end rank in new communicator *v*

MPS_Process_get_name (int rank, char *name) ;

rank in MPI_COMM_WORLD ^ |

my name in link/process file *v*

MPS_Process_get_rank (char *name, int *rank) ;

name in link/process file ^ |

my rank in MPI_COMM_WORLD *v*

MPS_Buf_pool_init

(MPI_Comm com, way, * p_bufsize, * p_bufcount, *p_mps_pool)

MPI communicator ^ ^ | | |

Send or Receive | *v* *v* |

buffer size & count *v*

MPS pool handle

MPS_Buf_get (p_mps_pool, void **p_buffer)

get buffer from pool (may block, or return EEMPTY)

MPS_Buf_release (p_mps_pool, void *buffer)

give buffer to I/O system (compulsory at each use) busy???

MPS_Buf_pool_finalize (p_mps_pool)

free all buffers, all coms must have completed first

MPI/MPS example

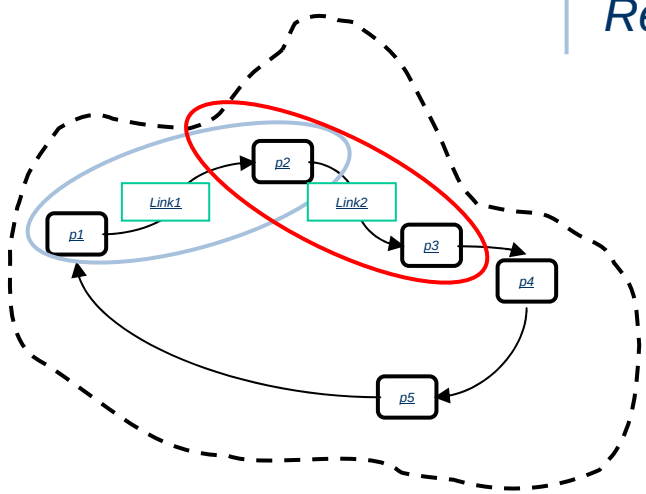
```
MPI_Init(&argc, &argv);

MPS_Channel_create("link1", "proc2", &com, &lrank, &rrank);
MPS_buf_pool_init(com,  (sender) ? MPS_SND : MPS_RCV, &bufsize, &bufcount, &pool);
if (sender) {
        MPS_Buf_get(pool, &buf);
        Fill in with data
        MPI_Isend(buf, size/sizeof(int), MPI_INT, rrank, 99, com, &req);
        MPI_Wait(req, &status);
        MPS_Buf_release(pool, buf);
} else {
        …
}
MPS_Buf_pool_finalize(pool);
MPI_Finalize();
```

Create Dedicated Link
Get Specific connector
Initialize memory pool

Take buffer ownership

Send on connector

Release buffer

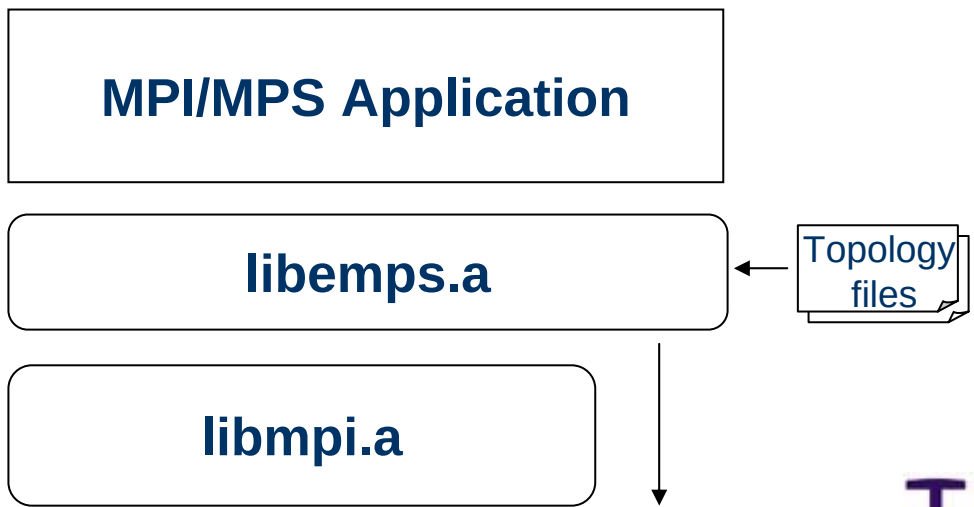


MPI application easily ported to MPI/MPS API

- See example

MPI/MPS application can run on any platform: EMPS

- EMPS is MPS emulation on top of standard MPI com
- Allow to run MPI/MPS code unmodified
  - Includes buffer and link management



Based on MICH ?? Version etc…

Software

- IA32 Red Hat, PowerPC LynxOS 4.0

HW Targets

- PC, Thales multiprocessor VME boards

Multi-protocol support in COM layer

- DDlink : Direct Deposit zero copy layer
  - Fibre Channel RDMA, Shared Memory, VME 2eSST, RapidIO
- Standard unix/posix I/O
  - Shared Memory, TCP/IP

Finalize process mapping

- MPI_RUN and HPEC compatible process mapping

Towards automatic code generation

- Create MPS / MPI code from HPEC application tools

More support for MPI-aware debug tools

- Like TotalView™

- Thank you
  vincent.chuffart@thalescomputers.fr