The potential of MPI shared-memory model for supporting hybrid communication scheme

Huan Zhou

High Performance Computing Center Stuttgart (HLRS), University of Stuttgart, Germany

WSSP / Stuttgart, Germany / 10.10.2018

Outline

Background

Use case 1 – hybrid MPI RMA (DART-MPI)

Use case 2 – hybrid MPI collectives

Discussion and Summary

Multi-/many-core processor design in current clusters

- Decouple the intra-node parallelism and inter-node parallelism
- Parallel applications should be implemented with locality-awareness
  - MPI dominates the HPC applications
  - Encourage hybrid MPI communication scheme
- Hazel Hen@HLRS: Cray XC40 (used as experimental platform)

  - Dual twelvecore Intel Haswell processors comprise one compute node
    - 24 cores in one compute node
    - The two processors are connected with intel QPI
  - The nodes are connected via the Cray Aries network

MPI communication – message passing, RMA and collective

Message passing

- Two-sided communication
- Routines:
  - Blocking: MPI_Send and MPI_Recv
  - Non-blocking: MPI_Isend, MPI_Irecv and MPI_Wait/Waitall

RMA

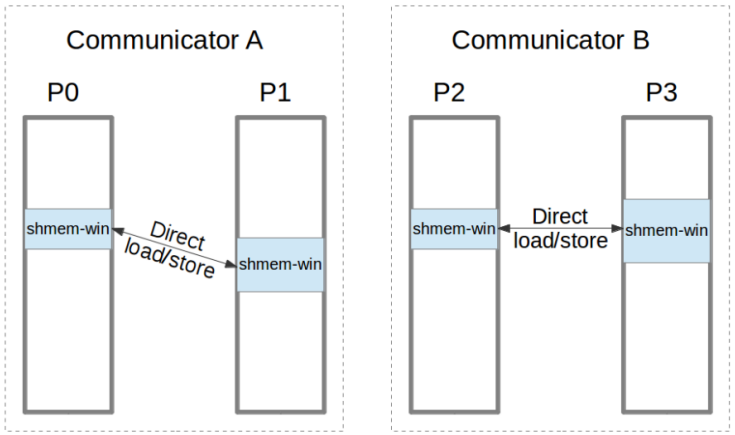
- One-sided communication
- Routines
  - Window: A known memory region permissible for RMA operations
  - Non-blocking: MPI_Put and MPI_Get and other synchronization calls (e.g., MPI_Win_flush, MPI_Wait...)

Collective communication

- Involves multiple senders or receivers
- Routines:
  - MPI_Bcast, MPI_Gather, MPI_Reduce, MPI_Allreduce, MPI_Allgather, ...

MPI shared-memory window

- Shared-memory window accesses are supported:
  - MPI_Comm_split_type: identify sub-communicators on which the shared-memory region can be created
  - Collectively call MPI_Win_allocate_shared: generate a shared-memory window object (*shmem-win*) on each of the above sub-communicators
  - MPI_Win_shared_query: Obtain the base pointer to the local shared-memory region of any other target in the same sub-communicator



⇒ The exposure of MPI shared-memory window enables shared-memory programing in MPI

Outline

Background

Use case 1 – hybrid MPI RMA (DART-MPI)

Use case 2 – hybrid MPI collectives

Discussion and Summary

DART – the runtime system of DASH (a C++ library realizing PGAS model)

DART-MPI

- An MPI implementation of DART

Team and group

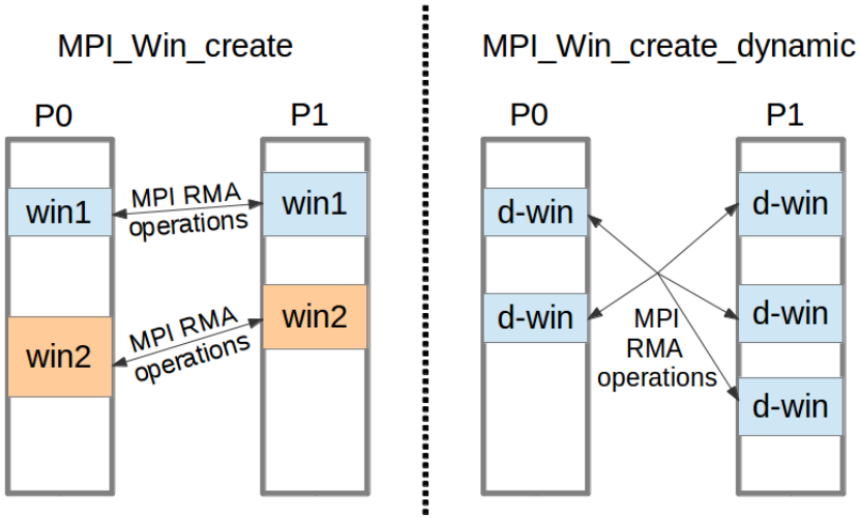
- Based on MPI communicator and group

Global memory

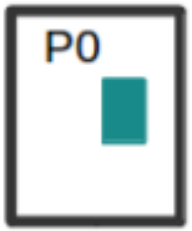
- Based on a nested window structure
  - Leads to a two-tier global memory model
  - Supports for locality-aware RMA communication scheme
- Collective global memory
  - Allocated collectively and distributed across a team
  - Only visible to the processes in the given team

MPI RMA window

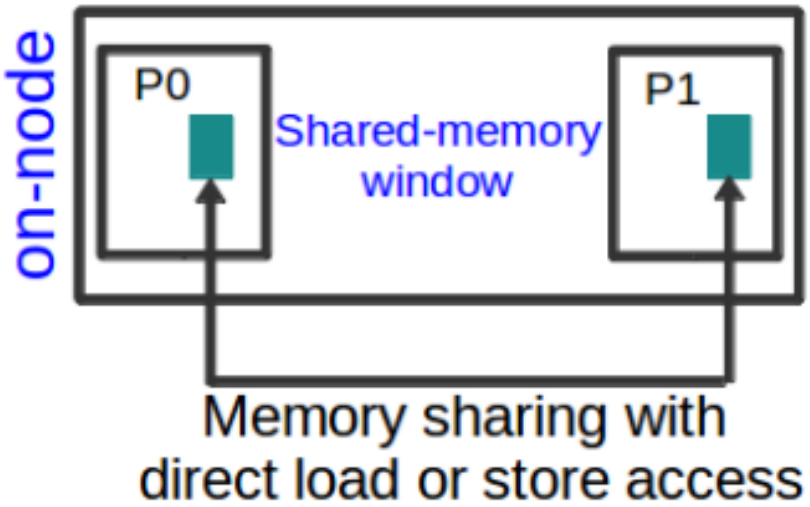
- Two ways to make a known memory region available for RMA operations
  - MPI-created window: collectively call MPI_Win_create: return a window object(*win*) spanning the existing memory
  - MPI dynamically-created window:
    - Collectively call MPI_Win_create_dynamic: return a window object(*d-win*) without any memory associated
    - Locally call MPI_Win_attach or MPI_Win_detach



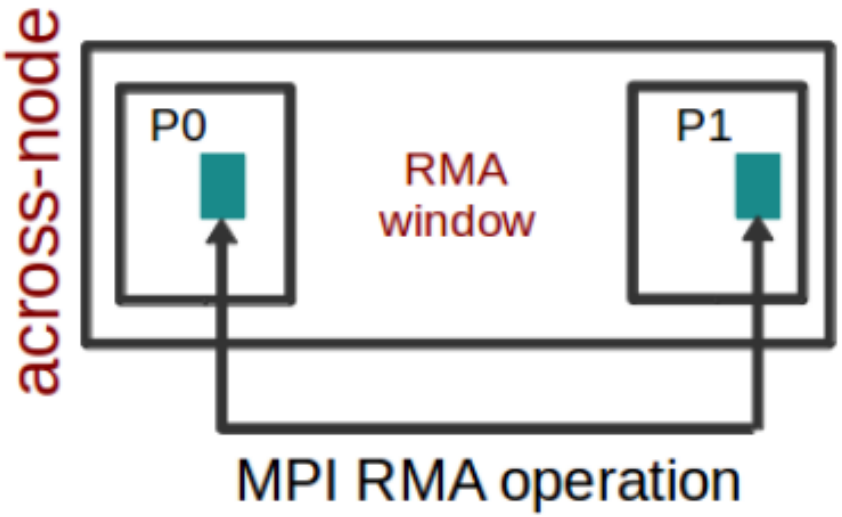
DART-MPI two-tier global memory model



DART-MPI two-tier global memory model

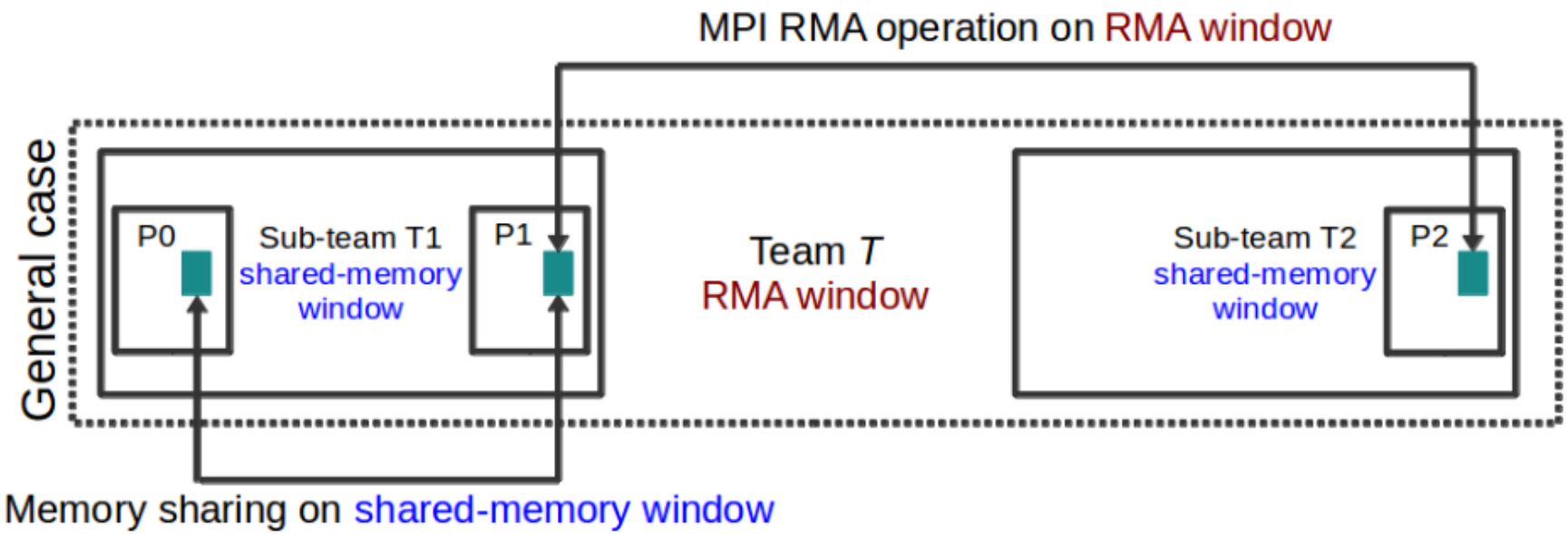


DART-MPI two-tier global memory model



DART-MPI two-tier global memory model

Take global memory on team T for example

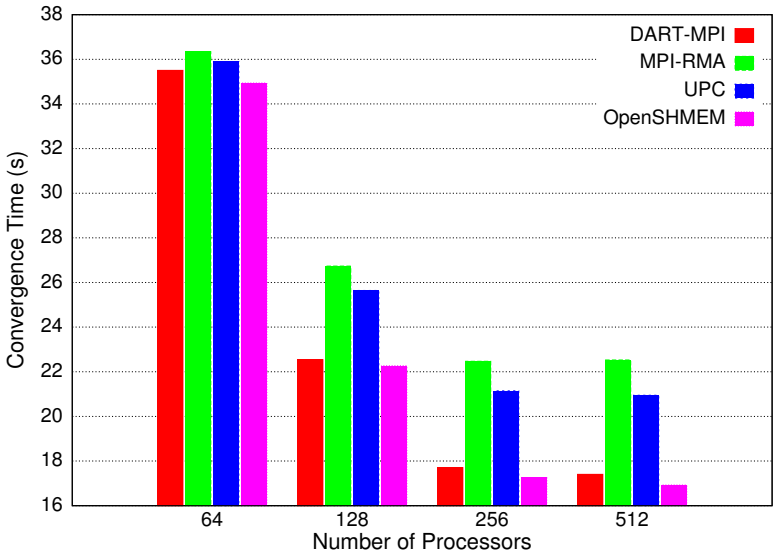


Experimental evaluation – put

- 2D stencil benchmark
  - Grid is decomposed by rows
  - The boundary elements are transmitted by using blocking put operations
- Comparison targets (Cray compiler)
  - MPI; UPC (use the compiler flag *-h upc*); OpenSHMEM

Experimental evaluation – put

- 2D stencil benchmark
  - Grid is decomposed by rows
  - The boundary elements are transmitted by using blocking put operations
- The grid with 1024X1024 elements is distributed across nodes
  - Increase process counts from 64 to 512
- DART-MPI RMA scales well, as in the case with OpenSHMEM
- DART-MPI RMA performs and scales better then MPI RMA
  - Attributed to the locality-aware implementation of DART-MPI RMA
- Comparison targets (Cray compiler)
  - MPI; UPC (use the compiler flag *-h upc*); OpenSHMEM



Outline

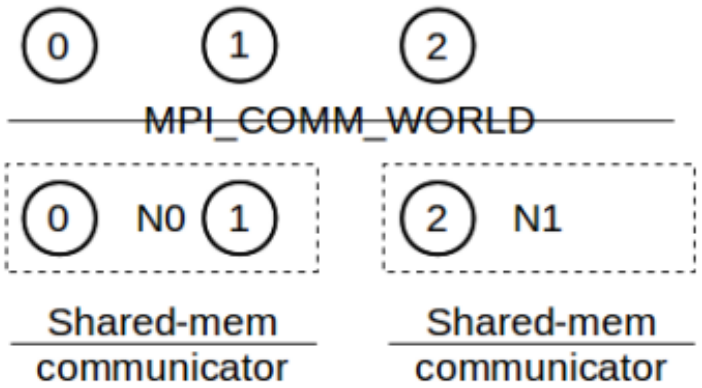
Background

Use case 1 – hybrid MPI RMA (DART-MPI)

Use case 2 – hybrid MPI collectives

Discussion and Summary

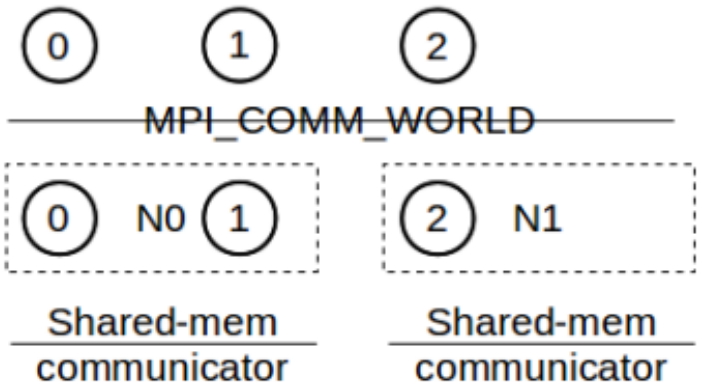
Shared-memory and node communicator



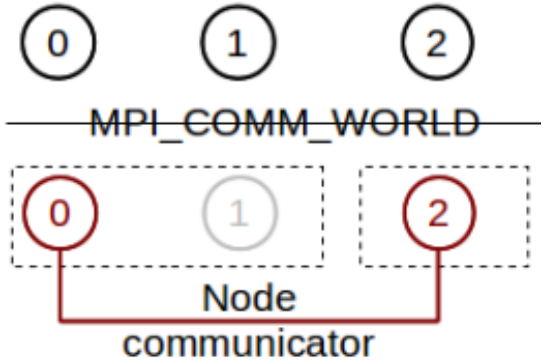
Creation of two shared-memory communicators on two nodes

- Processes in the same shared-memory communicators
  - Share one memory buffer
  - Locally accessed with immediate load/store operations

Shared-memory and node communicator



Creation of two shared-memory communicators on two nodes

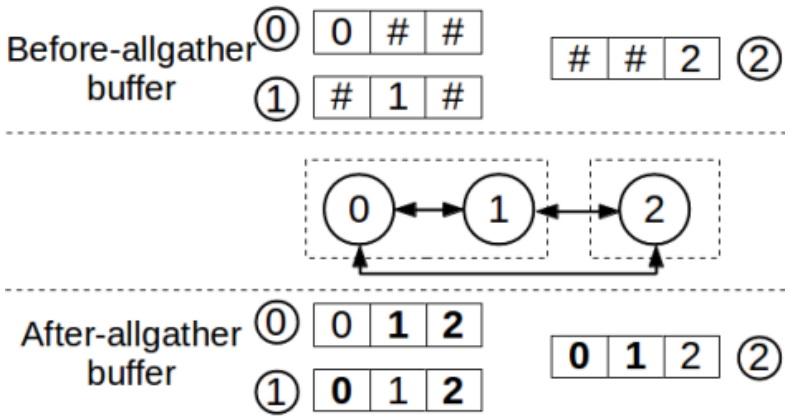


Creation of node communicator

- Processes in the same shared-memory communicators
  - Share one memory buffer
  - Locally accessed with immediate load/store operations

- A representative (the first logical process) within each node is chosen
  - Take the responsibility for the across-node data exchanges
  - Eliminating the unnecessary on-node memory copy operations

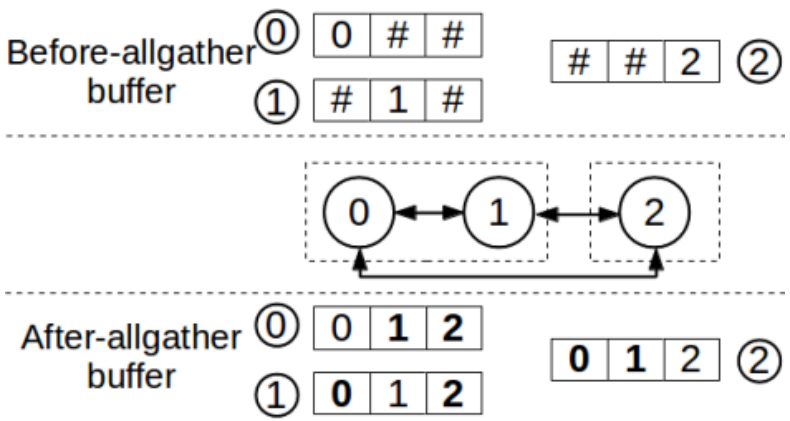
Native and hybrid allgather communication operation



Native allgather implementation

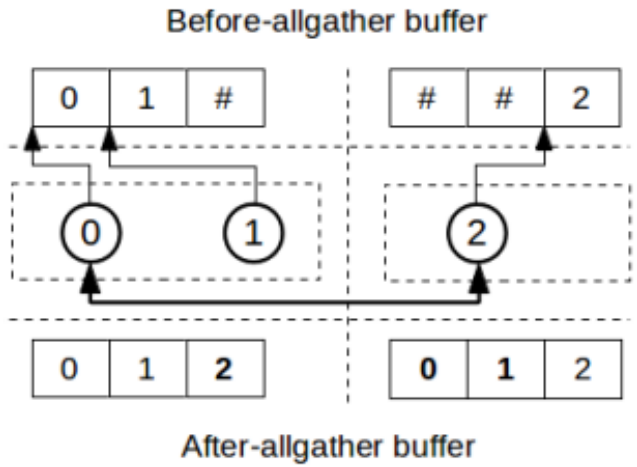
- Each process owns a region of memory to explicitly receive data from other (on-node or across-node) processes

Native and hybrid allgather communication operation



Native allgather implementation

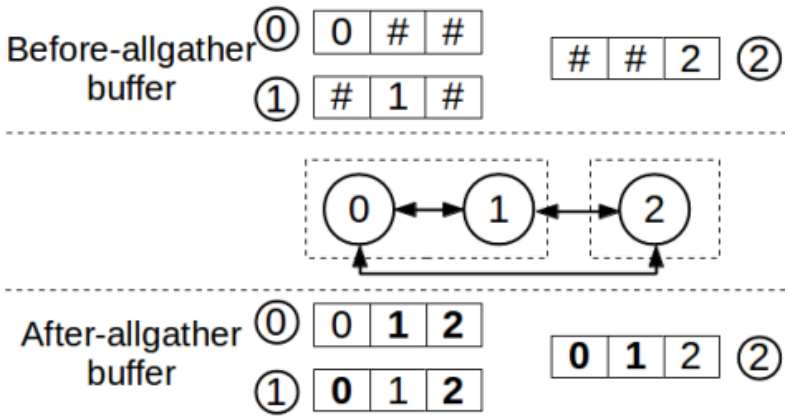
- Each process owns a region of memory to explicitly receive data from other (on-node or across-node) processes



Hybrid allgather implementation

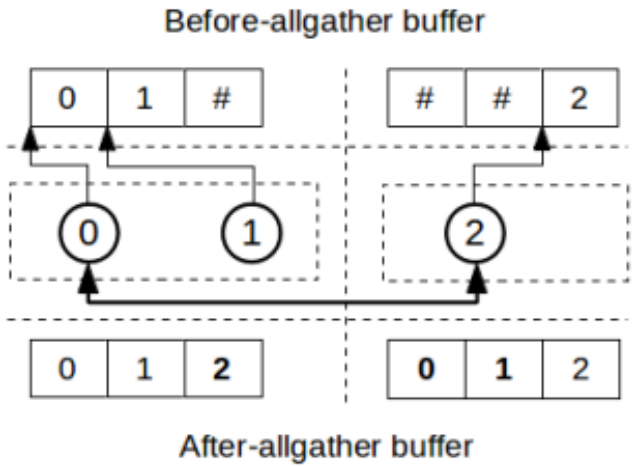
- Processes 0 and 1 share data and thus no communication between them
- Processes 0 and 2 comprise the node communicator and explicitly exchange data by performing allgather operation

Native and hybrid allgather communication operation



Native allgather implementation

- Each process owns a region of memory to explicitly receive data from other (on-node or across-node) processes



Hybrid allgather implementation

- Processes 0 and 1 share data and thus no communication between them
- Processes 0 and 2 comprise the node communicator and explicitly exchange data by performing allgather operation

Note that, this hybrid method applies to the situation that each process only alter the portion of data, to which it locally points

Experimental evaluation – allgather

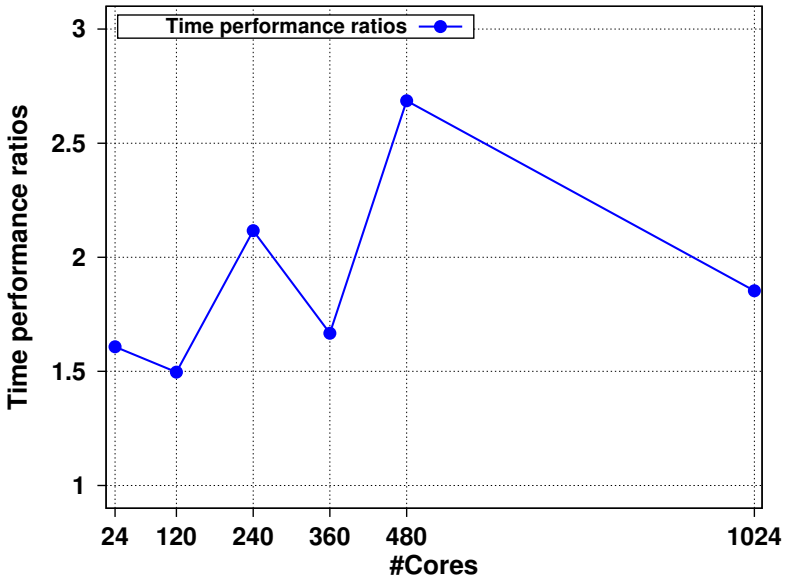
BPMF application

- Provided by Imec
- C++ implementation of Bayesian Probabilistic Matrix Factorization
- The number of iterations to be sampled is set to be 20
  - Three MPI allgather calls are performed in each iteration

Experimental evaluation – allgather

BPMF application

- Provided by Imec
- C++ implementation of Bayesian Probabilistic Matrix Factorization
- Time performance ratios between the native and hybrid allgather communication, for BPMF application
  - The number of MPI processes is increased from 24 to 1024
  - The ratios are always above 1
    - Hybrid allgather performs better than the native one in this scenario
  - Hybrid allgather leads to better BPMF application performance
- The number of iterations to be sampled is set to be 20
  - Three MPI allgather calls are performed in each iteration



Outline

Discussion: synchronization consideration

MPI_Barrier on shared-memory communicator

- A straightforward but heavy-weight choice
- Example (hybrid allgather):

```
Each process updates the data it points to;
MPI_Barrier(shared-memory communicator); //all sent data buffers are clear
The representatives explicitly perform data exchanges;
MPI_Barrier(shared-memory communicator); //all receive data buffers are ready
```

Pair of intra-node MPI_Send and MPI_Recv containing empty message

- A relatively light-weight choice
- Example:

```
P0: Write data to the "receiveBuf" pointed by P1;
    MPI_Send (NULL, 0, 1); // notify P1, exist as a handshake
P1: MPI_Recv (NULL, 0, 0); // continue till the "receiveBuf" is ready
    Use "receiveBuf";
```

Summary

- MPI shared-memory model
  - Facilitates data locality-aware programming
    - Hybrid RMA – DART-MPI
    - Hybrid collectives – allgather
    - The above two use cases show that the usage of MPI shared-memory model can bring performance benefit
- Extra synchronization overhead
- Further investigation on the application of MPI shared-memory model and synchronization approaches

Thank You for Your Attention!

zhou@hlrs.de