Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Accelerator Aware MPI Micro-benchmarking using CUDA, OpenACC and OpenCL

Sadaf Alam, Ugo Varetto[a]

[a]*Swiss National Supercomputing Centre, Lugano, Switzerland*

Abstract

Recently MPI implementations have been extended to support accelerator devices, Intel Many Integrated Core (MIC) and nVidia GPU. This has been accomplished by changes to different levels of the software stacks and MPI implementations. In order to evaluate performance and scalability of accelerator aware MPI libraries, we developed portable micro-benchmarks to indentify factors that influence efficincies of primitive MPI point-to-point and collective operations. These benchmarks have been implemented in OpenACC, CUDA and OpenCL. On the Intel MIC platform, existing MPI benchmarks can be executed with appropriate mapping onto the MIC and CPU cores. Our results demonstrate that the MPI operations are highly sensitive to the memory and I/O bus configurations on the node. The current implemetation of MIC on-node communication interface exhibit additional limitations on the placement of the card and data transfers over the memory bus.

1. Introduction to Accelerator Aware MPI and Micro-benchmarks

Intel Many Integrated Core (MIC) and AMD and Nvidia Graphical Processing Unit (GPU) accelerator devices offer a significantly larger fraction of computational capabilities for PRACE prototypes with heterogeneous compute nodes [1,2,3]. As a result, code developers tend to port and migrate the most demanding part of their scientific simulations to these accelerator devices. Within a cluster environment and specifically on massively parallel processing (MPP) tier-0 systems, there are hundreds or thousands of heterogeneous compute nodes and an efficient MPI communication interface from accelerator devices is essential in order to sustain node and parallel efficiencies. An accelerator aware MPI library therefore provides an efficient mechanism to transfer data between the discrete memories of the host and accelerator devices over a high speed interconnect [4].

The micro-benchmark development was focused on extending a widely used MPI benchmark, the OSU MPI benchmarks [5], for homogeneous multi-core devices to the PRACE target platforms. Different versions of point-to-point and collective benchmarks have been ported to CUDA (GPU programming language for Nvidia devices) [6], OpenACC (a portable, directives based standard) [7] and OpenCL (a portable language extension) [8]. We also experimented with MPI derived data types on GPU devices in order to evaluate both performance and productivity using GPU aware MPI benchmarks. This benchmark development effort was undertaken to evaluate PRACE prototypes. In addition to the development of micro-benchmarks for accelerator devices, particularly GPU accelerators, different modes of execution have been explored to evaluate MPI performance on a MIC platform [9]. Existing MPI benchmarks for CPU devices can be configured and executed on systems with only MIC accelerators.

The PRACE prototypes and the target systems that are considered for this study include clusters at CSC (Finland), CINECA (Italy), PSNC (Poland) and CSCS (Switzerland). These systems contain either the GPU or MIC accelerator devices or both. Details of the target platforms are provided in the subsequent sections.

- The goals of the project
  - Evaluate instances of accelerator aware MPI and underlying technologies on PRACE prototype
  - Evaluate different node and network configurations using micro-benchmarks
  - Compare performance across all three prototypes using a common set of benchmarks
- Work done in the project, including
  - Develop accelerator aware MPI micro-benchmarks for point-to-point and collective MPI operations
  - Develop portable versions to target multiple platforms
  - Introduce device bindings to benchmarks for high fidelity measurements
  - Compare performance of CPU and accelerator versions of benchmarks
  - Identify bottlenecks, specifically for tier-0 scale systems
- Results obtained
  - A high fraction of peak performance can be obtained for bandwidth sensitive benchmarks using platform tuned optimization settings
  - Latencies could be 10x slower compared to the point-to-point CPU communication
  - Two MIC devices on a single node have limitations on transfers over the memory bus
  - Bandwidth sensitive benchmarks can saturate on-node I/O bandwidth
  - PCIe gen 2 limitation poses a serious bottleneck for bandwidth scaling
- Conclusions and summary
  - Accelerator aware MPI can benefit applications as users can rely on efficient, portable implementation across multiple platforms rather than tuning device, host and network transfers on individual platforms
  - Availability of portable languages and languages standard across multiple devices is work in progress. As a result, OpenACC and OpenCL benchmarks can currently be executed on a subset of platforms
  - Benchmarks that have been developed as part of PRACE work package effort can be used to evaluate optimal node and network configurations of tier-0 systems as these benchmarks can assist in exploring on and off node bottlenecks and comparative analysis of CPU and accelerator network performance
  - Support for MPI data types is work in progress and could improve users productivity and portability across different system configurations

2. Implementation of Benchmarks

The OSU MPI micro-benchmark package consists of a number of point-to-point and collective benchmarks for latency and bandwidth measurements for a range of payload sizes. Two benchmarks, MPI latency (osu_latency) and MPI bandwidth (osu_bw) were extended to CUDA when the first version of accelerator aware MVAPICH2 library was released. In collaboration with the OSU team, we developed collective benchmarks, OpenACC extensions and OpenCL versions of CUDA benchmarks. The benchmarks have been developed using a very simple concept of initializing MPI data on the device memory, as shown in Figure 1. Two modes of data transfer are shown in the figure. In the first mode, depicted with the path in red, data resides in the accelerator memory and is then transferred to the interconnect via the host CPU memory on the sender side. This memory can be pinned and shared between the two I/O devices namely the network interface card (NIC) and the GPU for efficient transfers. The same procedure is repeated on the receiver side.

Instead of users explicitly copying over data between the host and the device, data residing on the device memory can be used as send and receive buffers. Intermediate steps are performed by the accelerator aware MPI implementation. Hence in Figure 1, from the conceptual point of view, data can go directly from the GDDR memory of the device to the network buffers and vice versa hence eliminating extra transfers between the host and

the device. This approach has a potential for not only yielding higher, portable performance but it could also simplify parallel algorithm design on heterogeneous platforms. There are benefits to performance efficiency and portability as the code developer can rely on an efficient MPI implementation for accelerators rather than embedding tuning strategies for different target platform configurations.

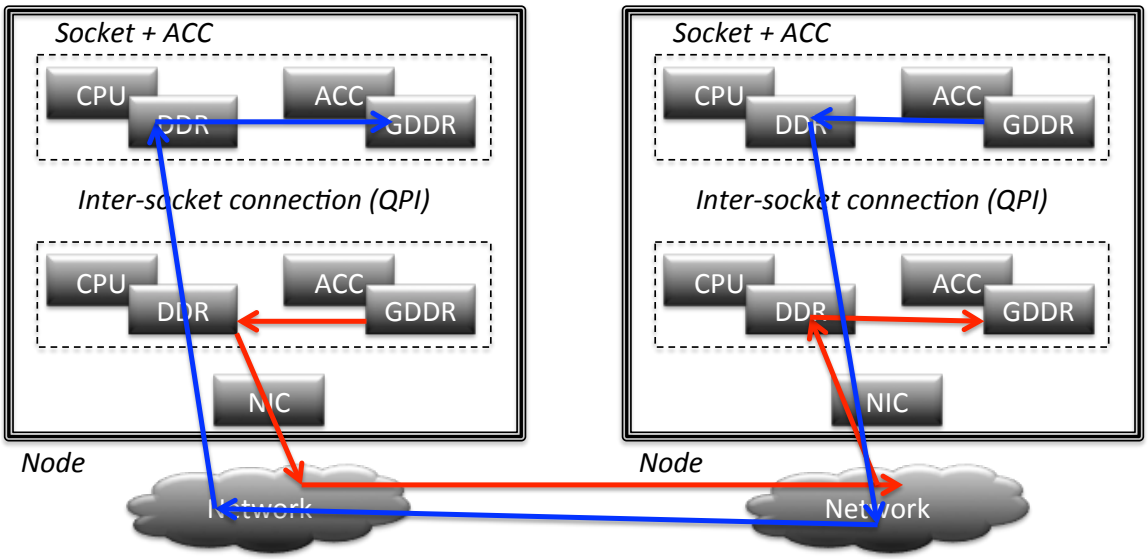

Figure 1: MPI workflow without an accelerator aware MPI implementation, where the programmer explicitly implements data transfers between the CPU memory (DDR) and the device memory (GDDR). Data transfer path shown with the red line has fewer transfers than the path that is shown with the blue line, which may involve additional bus transfers over the memory bus.

We tested the GPU-aware implementations of OpenMPI and MVAPICH. For linear buffers it is faster to pass the GPU memory pointer to MPI send/receive functions instead of performing a GPU-host copy and passing the pointer to host memory to the MPI implementation. For more complex layouts like 3D halo regions of regular grids it is in general faster to perform a manual packing/unpacking to/from a linear buffer on the GPU and then calling the send/recv functions than using MPI data types to describe the layout.

MPI vector data types seem to be optimized only for certain power of two sizes.

MPI Send-receive

```
device_pointer_* = pointer to device (GPU) memory
host_pointer_*   = pointer to host (CPU) memory
```

Wihtout MPI-aware implementation:

```
// SEND
copy from device to host
cudaMemcpy(host_pointer_target, device_pointer_source, size, cudaMemcpyDeviceToHost);
//call mpi
MPI_Send(host_pointer_target,...);
...
// RECEIVE
MPI_Recv(host_pointer_source, ...);
//copy from host to device
cudaMempy(device_pointer_target, host_pointer_source, size, cudaMemcpyHostToDevice);
```

With MPI-aware implementation:

```
SEND
MPI_send(device_pointer_source,...)

RECEIVE
MPI_recv(device_pointer_target,...)
```

MPI data types

MPI data types are a way of specifying a complex data layout with a type definition that includes all the information (size and offsets) required to copy data from/to memory.

MPI data types work the same way for host and device.

1) allocate memory on the GPU

```
void* block;
cudaMalloc(&block, size);
```

2) define the data type e.g. a sub-region of a linear buffer

```
MPI_Datatype blocktype1, finaltype;
MPI_Type_create_subarray(..., &blocktype1); //define buffer
MPI_Type_commit(&blocktype1);
MPI_Type_create_hindexed(..., blocktype1, &finaltype); //specify sub-region
MPI_Type_commit(&finaltype);
```

3) send data passing the pointer to either cpu or gpu memory to MPI

```
MPI_Send(block, 1, finaltype,...);
```

Online resources:

Minimal MPI datatype example:
https://github.com/ugovaretto/cuda-mpi-scratch/blob/master/mpi-complex-types.cpp

Full 2D grid halo exchange with (reusable) C++ code
https://github.com/ugovaretto/cuda-mpi-scratch/tree/master/stencil2d

An example of CUDA extended version of the MPI_Allreduce (a collective communication operation) benchmark is shown below that shows how memory is allocated to the GPU device instead of the host:

```
#ifdef _ENABLE_CUDA_
    cuerr = cudaMalloc((void**) &sendbuf, max_msg_size * sizeof(float));
    cuerr = cudaMalloc((void**) &recvbuf, max_msg_size * sizeof(float));
    cudaMemset(sendbuf, 1, max_msg_size);
    cudaMemset(recvbuf, 0, max_msg_size);
#endif

…

t_start = MPI_Wtime();
MPI_Allreduce(sendbuf, recvbuf, size, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD );
t_stop=MPI_Wtime();
```

Likewise, for OpenACC extensions, the data is allocated on the device.

```
#ifdef OPENACC
        sendbuf = (float *) acc_malloc(sizeof(float) * max_msg_size);
        recvbuf = (float *) acc_malloc(sizeof(float) * max_msg_size);
#pragma acc parallel loop deviceptr(sendbuf,recvbuf)
            for(i = 0; i < max_msg_size; i++) {
                sendbuf[i] = 1.0f;
                recvbuf[i] = 0.0f;
            }
#endif
```

Similar implementation strategy is applied to the OpenCL version.

```
       // Create memory buffers on the device
cl_mem  s_mem  =  clCreateBuffer(context,  CL_MEM_WRITE_ONLY  |  CL_MEM_ALLOC_HOST_PTR,
MYBUFSIZE, NULL, &ret);
cl_mem  r_mem  =  clCreateBuffer(context,  CL_MEM_WRITE_ONLY  |  CL_MEM_ALLOC_HOST_PTR,
MYBUFSIZE, NULL, &ret);

   // pinned memory (blocked call)
   s_buf1 = (char *) clEnqueueMapBuffer(command_queue, s_mem, CL_TRUE,
CL_MAP_WRITE, 0, MYBUFSIZE, 0, NULL, NULL, &ret);

r_buf1 = (char *) clEnqueueMapBuffer(command_queue, r_mem, CL_TRUE,
CL_MAP_WRITE, 0, MYBUFSIZE, 0, NULL, NULL, &ret);
```

On the MIC system, a MIC specific executable needs to be generated (using `-mmic`) for the OSU micro-benchmarks. We also generated CPU benchmarks for debugging MPI performance issues on the MIC device itself, between the MIC and the host device and between multiple MIC devices on a single node and across multiple nodes.

Unlike MPI and GPU aware MPI execution models, the execution model on the MIC system requires careful mapping of the MPI tasks, on the host and on the MIC device itself. On the CSC prototype, different environment variables are introduced to control the mapping of MPI tasks on different nodes and MPI devices via the SLURM resource management and job scheduling system. In the absence of such interface, users are responsible for explicit mapping of tasks and for ensuring that the tasks are placed as desired.

On the CSC prototype, the SLURM environment variable called MIC_PPN defines the number of MPI tasks per

node. If a user chooses to have a different number of MPI tasks, this has to be done manually. The number of host MPI tasks can be defined using (-n) with mpi_run while tasks-per-node defines the number of MPI tasks per node. The SLURM command for executing two MPI tasks on two MIC nodes and two MPI tasks on two CPU nodes each will be:

```
$ MIC_PPN=1 srun -n 2 --tasks-per-node 1 mpirun_mic –m mic_exe –c cpu_exe
```

However, for executing two MPI tasks on a single node between two MIC devices, each with one MPI task, users may need to manually map the tasks on a single node:

```
$ mpi_run –n num_tasks –host cpu0-mic0 ./mic_exe: -n num_tasks –host cpu0-mic1 ./mic_exe
```

The naming convention for CPU and MIC devices therefore is critical for running MPI benchmarks successfully onto a system with one or more MIC devices per node. The above command could become complex once MPI collective experiments are performed on multiple MIC devices and hosts.

3. Target Systems: PRACE Prototypes and Clusters with Accelerator Devices

The development of accelerator aware MPI micro-benchmarks has been targeted to PRACE prototype systems with accelerator devices. These include:

- Advanced Scalable Hybrid Architecture (Scalable Hybrid) from CSC
- European Many Integrated Core Architecture (EURORA) from CINECA
- Assessment of Hybrid CPU/GPU Architecture (Fusion) from PSNC

Only key characteristic features that are necessary to understand why a different porting and mapping approach is necessary for these three systems have been highlighted in this whitepaper. Figure 2 shows three type of nodes designs that are representative of the node architecture of these platforms. A CPU could be any x86_64 multicore system such as an Intel Sandy Bridge or AMD Opteron. An ACC could be a GPU or MIC accelerator device from any vendor: Intel, Nvidia or AMD. Two types of memory technologies could be present. DDR memories could be DDR3 for CPU or GDDR5 for accelerator devices or could be a unified memory. Although the three prototypes exhibit distinct network characteristics, the scope of this white paper is limited only to accelerator aware MPI implementation, which were not available on the PSNC prototype at the time of writing this report. However, the current portable benchmarks can be executed on the PSNC system as soon as an extended version of MPI or an OpenACC compiler for AMD devices is available.

Node design A shows a node with dual-socket CPU and dual accelerator devices. The two CPU sockets are connected with a memory interface, for example, QPI for Intel Sandy Bridge processors, resulting in non-uniform memory architecture (NUMA). There are two accelerator devices, which are connected to each CPU, hence their distances from the network devices are different. The EURORA system exhibits node design A. CSC phase I system is similar but has a single accelerator device. The system contains two Intel MIC (Knights Corner) devices, each with more than 50 cores. The system has DDR3 memory for the CPU and a higher bandwidth GDDR5 memory for the MIC devices.

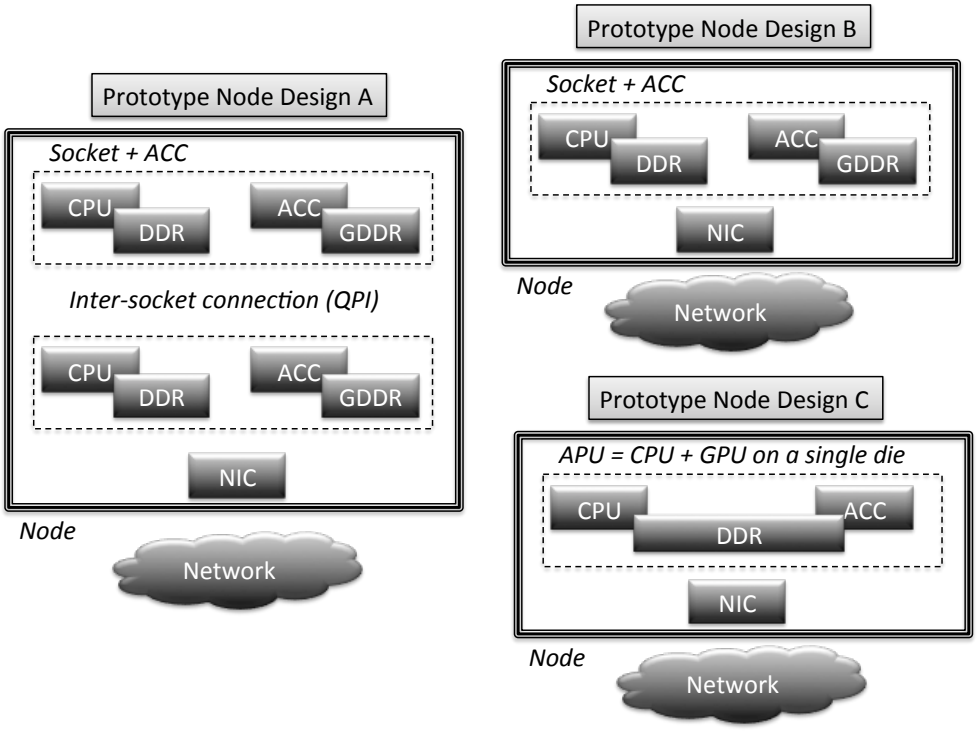

Figure 2: Representation of three types of nodes that are designs that are being developed for the PRACE accelerator based prototype systems

Node design B is similar to node design A except only one CPU socket and accelerator device is present on each processing node. This Scalable Hybrid prototype is expected to contain 50% of nodes with Intel KNC cards and 50% with Nvidia Kepler K20 devices.

Node design C represents a node of the PSNC Hybrid CPU/GPU prototype. This design has a few unique features. Firstly, there is a single memory that can be accessed by both the CPU and the accelerator device. The CPU is an AMD Opteron and GPU is an AMD Radeon device. The system contains a unified northbridge unit that manages and schedules memory access from CPU and GPU devices.

In addition to the PRACE prototype systems, a number of experiments are performed on an in-kind system at CSCS. This system is composed of node design A nodes, which can also be configured to exhibit characteristics of node design B. The system was composed of two Intel Sandy Bridge CPUs and either two Intel KNC 5110p or two Nvidia K20 accelerator devices per node.

4. Results

In this report we provide two sets of results that could be used to benchmark an accelerator based prototype system with and without an accelerator aware MPI library. The first set of experiments are designed as a regression for the system to establish whether on and off-node MPI performance features comply with the underlying system characteristics. Table 1 lists results of MPI point-to-point experiments that are performed using 8 bytes and 1 Mbytes message sizes for latency and bandwidth sensitive benchmarking respectively. Inter-node host-to-host (between two CPU devices over the FDR interconnect) establishes the upper bound for bandwidth and lower bound for latencies. The overhead in latencies for 8 bytes messages are introduced by additional hops between the device and the host transfers, which cannot be pipelined as is the case for bandwidth sensitive payload sizes. As a result, on the GPU based systems, inter and intra node, device-to-device bandwidth has a small slowdown with respect to host-to-host transfers. The MIC based systems exhibit lower latencies as compared to GPU based systems, both for inter and intra node transfers. There is a bottleneck on the inter-node host-to-device transfers, especially on systems with two MIC cards that are connected to two separate sockets on a node. This may be attributed to transfer of data through a memory bus (QPI) to I/O interfaces (PCIe).

Execution model	Xeon Phi 8B latency (usec)	Kepler 8B latency (usec)	Xeon Phi 1MB BW (GB/s)	Kepler 1MB BW (GB/s)
Inter-node Host-Host	1.09	1.09	6.4	6.4
Intra-node Host-Device	4.1	9.11	5.1	4.95
Inter-node Host-Device	7.9	16.5	0.4	5.7
Intra-node Device-Device	3.4	15.8	2.2	5.8
Inter-node Device-Device	9.2	18.08	0.3	5.7

In order to stress the on-node and off-node communication interfaces, we experiment with latency sensitive collective communication (MPI_Allreduce) and bandwidth sensitive (MPI_Alltoall) benchmarks within the OSU benchmark suite. These benchmarks have been ported to both CUDA and OpenACC. Results are shown in Figure 3 and Figure 4 for MPI_Allreduce and MPI_Alltoall benchmarks respectively for 8 MPI tasks, each mapped onto a single CPU core and a single GPU. The results are collected on systems with InfiniBand FDR and QDR interconnects and with two generations of GPU devices: M2090 (Fermi) and K20 (Kepler). Base results are collected on multi-core CPUs to measure the relative slowdown. As shown in both figures, there is less than 2x differences in performance using two different networks, InfiniBand QDR and InfiniBand FDR, and two generations of GPU devices, Fermi M2090 and Kepler K20 for MPI_Allreduce (for message sizes of 4 KB or less) and MPI_Alltoall. The slowdown with respect to the CPU implementation could be up to 8x. For some message sizes, it could be up to 15x. Hence, further tuning of MPI_Allreduce and MPI_Alltoall operations are needed depending on the node configuration as the on-node contention of resources can contribute to the slowdown for MPI communications that take place from device memories.

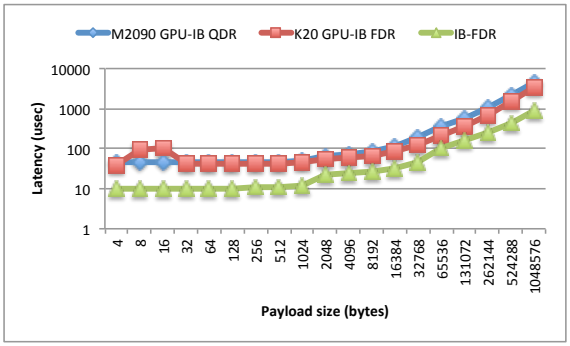

Figure 3: MPI_Allreduce latency measurements on platforms with InfiniBand QDR with M2090 GPU devices, with K20 GPU and InfiniBand FDR and without accelerators on an InfiniBand FDR clusters

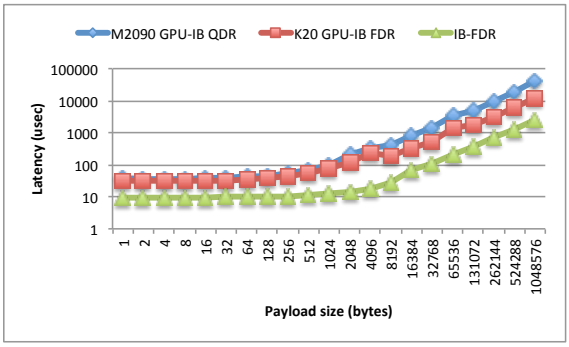

Figure 4: MPI_Alltoall latency measurements on platforms with InfiniBand QDR with M2090 GPU devices, with K20 GPU and InfiniBand FDR and without accelerators on an InfiniBand FDR clusters

5. Summary and Suggestions for Future Work

The implementation of micro-benchmarks for accelerator aware MPI libraries, enabled us to identify the following:

- For bandwidth sensitive payload sizes, accelerator aware MPI benchmarks could achieve close to theoretical bandwidth of the I/O bus connecting the devices and the network interface cards (for example, PCIe bandwidth and number of lanes)
- Latency sensitive benchmarks demonstrate a significant slowdown since the data transfers cannot be pipelined to hide data transfer latencies between CPU & GPU memories. Technologies like GPUDirect-RDMA have a

potential to improve performance by addressing the issue of extra memory transfers between the CPU and GPU devices

- The libraries are code and performance portable across multiple platforms and can exploit node configuration and software optimization, for example, peer-to-peer and inter-node device copies on GPUs.
- In some instances, users may need to explore some tunable parameters such as the chunk sizes to obtain optimal performance for selected message sizes. Different MPI library implementations provide some environmental variable to control chunk sizes.
- Currently, there is a bug with node configurations with two MIC cards that are connected through the I/O interfaces of two separate sockets on a node as shown in the results included in this paper. This may have serious implications for symmetric execution modes where MPI tasks are spread over host and MIC devices
- Collective benchmarks have a higher overhead with respect to the host or CPU only runtimes as demonstrated in the experiments. Further investigation is needed in order to improve efficiencies.
- As accelerator aware MPI is an emerging technology, it is not well integrated into the job management and execution systems across different clusters and job schedulers. Additional work is needed to provide an efficient and portable interface

As the PRACE prototypes continue to evolve, there is a need for comparative evaluation, which can only be performed using OpenACC and OpenCL versions of the micro-benchmarks. At the time of writing the report, OpenACC is not available on the Intel MIC system and implementation of OpenCL aware MPI libraries are not present on any platform. We therefore recommend benchmarking the fully assembled versions of the prototype hardware with the micro-benchmarks that have been developed as part of this study using both portable and platform specific implementation of the benchmarks.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU's 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-283493. The work was achieved using the PRACE Research Infrastructure resources at CSC, PSNC and CSCS.

References

1. EURORA: http://www.hpc.cineca.it/hardware/eurora
2. CSC prototype: https://confluence.csc.fi/display/HPCproto/HPC+Prototypes
3. PSNC: http://www.man.poznan.pl/online/en
4. Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K. Panda. 2011. MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters. Comput. Sci. 26, 3-4 (June 2011), 257-266.
5. OSU benchmarks: http://mvapich.cse.ohio-state.edu/benchmarks/
6. CUDA: https://developer.nvidia.com/category/zone/cuda-zone
7. OpenACC: http://www.openacc-standard.org
8. OpenCL: http://www.khronos.org/opencl/
9. Intel MIC: http://software.intel.com/en-us/mic-developer
10. OpenMPI: http://www.open-mpi.org/
11. MVAPICH2: http://mvapich.cse.ohio-state.edu/overview/mvapich2/
12. A. J. Peña and S. Alam. “Evaluation of inter- and intra-node data transfer efficiencies between GPU devices and their impact on scalable applications”, in The 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013), pp. 144-151, Delft, The Netherlands, May. 2013.
13. SLURM: http://slurm.schedmd.com/