# Table of Contents

1. Introduction ........................................................................................................... 3
2. Types of interconnects ............................................................................................ 4
   2.1. Omni-Path ........................................................................................................ 4
       2.1.1. Overview .................................................................................................... 4
       2.1.2. Using Omni-Path ..................................................................................... 4
   2.2. InfiniBand ......................................................................................................... 4
       2.2.1. Overview .................................................................................................... 4
       2.2.2. Setup types ............................................................................................... 4
       2.2.3. Using InfiniBand ....................................................................................... 5
   2.3. Aries ................................................................................................................... 6
   2.4. NVLink .............................................................................................................. 7
       2.4.1. NVLink performance overview ................................................................. 8
       2.4.2. GPUDirect ............................................................................................... 8
       2.4.3. NVSwitch ............................................................................................... 9
   2.5. Tofu ................................................................................................................... 10
   2.6. Ethernet ............................................................................................................ 10
   2.7. Interconnect selection ...................................................................................... 11
3. Common HPC network topologies ......................................................................... 12
   3.1. Fat trees .......................................................................................................... 12
   3.2. Torus ................................................................................................................. 12
   3.3. Hypercube ...................................................................................................... 13
   3.4. Dragonfly ........................................................................................................ 13
4. Production environment ......................................................................................... 14
   4.1. CURIE ............................................................................................................. 14
   4.2. Hazel Hen ....................................................................................................... 14
   4.3. JUWELS .......................................................................................................... 15
   4.4. Marconi ............................................................................................................ 15
   4.5. MareNostrum ................................................................................................. 15
   4.6. PizDaint .......................................................................................................... 16
   4.7. SuperMUC ...................................................................................................... 17
5. Network benchmarking .......................................................................................... 18
   5.1. IMB ................................................................................................................... 18
       5.1.1. Installation ............................................................................................... 18
       5.1.2. Benchmark execution ............................................................................. 18
       5.1.3. Benchmark output .................................................................................. 18
   5.2. Linktest ............................................................................................................ 19
       5.2.1. Installation ............................................................................................... 19
       5.2.2. Benchmark execution ............................................................................. 19
       5.2.3. Benchmark output .................................................................................. 20

Further documentation .................................................................................................. 21
1. Introduction

Having a high-bandwidth and low-latency interconnect usually makes the main difference between computers which are connected via a regular low-bandwidth, high-latency network and a so-called supercomputer or HPC system.

Different interconnection types are available, either from individual vendors for their specific HPC setup or in the form of a separate product, to be used in a variety of different systems. For the user of such a system, the interconnect is often seen as a black box.

A good overview regarding the interconnect distribution within the current generation of HPC systems is provided through the TOP500 list [1]:

**Figure 1. Interconnect system share, TOP500 list November 2018**

This guide will give an overview of the most common types of interconnects in the current generation of HPC systems. It will introduce the key features of each interconnect type and the most common network topologies. The selected interconnect type within the current generation of PRACE Tier-0 systems will be listed and the final section will give some hints concerning network benchmarking.
2. Types of interconnects

2.1. Omni-Path

Omni-Path (OPA, Omni-Path architecture) was officially started by Intel in 2015 and is one of the youngest HPC interconnects. Within the last four years more and more systems have been built using Omni-Path technology. The November 2018 TOP500 list [1] contains 8.6% Omni-Path systems.

2.1.1. Overview

Omni-Path is a switch-based interconnect built on top of three major components: A host fabric interface (HFI), which provides the connectivity between all cluster nodes and the fabric; switches, used to build the network topology; and a fabric manager, to allow central monitoring of all fabric resources.

Like InfiniBand EDR, Omni-Path supports a maximum bandwidth of 100 GBit/s in its initial implementation (200GBit/s is expected in the near future). Fiber optic cables are used between individual network elements.

In contrast to InfiniBand, Omni-Path allows traffic flow control and quality of service handling for individual messages, which allows the interruption of larger transmissions by messages having a higher priority.

2.1.2. Using Omni-Path

Typically Omni-Path is used beneath a MPI implementation like OpenMPI or MVAPICH. OPA itself provides a set of so-called performance scaled messaging libraries (PSM and the newest version PSM2), which handle all requests between MPI and the individual HFI, however the OFED verbs interface can be used as an alternative or fallback solution. It is also possible to use the PSM2 libraries directly within an application. [3]

2.2. InfiniBand

InfiniBand is one of the most common network types within the current generation of large scale HPC systems. The November 2018 TOP500 list [1] shows 27% of all 500 systems used InfiniBand as their major network system.

2.2.1. Overview

InfiniBand is a switch-based serial I/O interconnect. The InfiniBand architecture was defined in 1999 by combining the proposals Next Generation I/O and Future I/O and is standarized by the InfiniBand Trade Association (IBTA). The IBTA is an industry consortium with several members such as Mellanox Technologies, IBM, Intel, Oracle Corporation HP and Cray.

The key feature of the InfiniBand architecture is the enabling of direct application to application communication. For this, InfiniBand allows direct access from the application into the network interface, as well as a direct exchange of data between the virtual buffers of different applications across the network. In both cases the underlying operating system is not involved. [4]

The InfiniBand Architecture creates a channel between two disjoint physical address spaces. Applications can access this channel by using their local endpoint, a so-called queue pair (QP). Each queue pair contains a send queue and a receive queue. Within the established channel, direct RDMA (Remote Direct Memory Access) messages can be used to transport the data between individual memory buffers at each site. While data is transferred, the application is able to perform other tasks in the meantime. The device handling the RDMA requests, implementing the QPs and providing the hardware connections is called the Host Channel Adapter (HCA). [7]

2.2.2. Setup types

InfiniBand setups can be categorized mainly by the link encoding and signaling rate, the number of individual links to be combined within one port, and the type of the wire.
To create a InfiniBand network, either copper or fiber optic wires can be used. Depending on the link encoding, copper wires support a cable length between 20m (SDR) to 3m (HDR). Fiber cables support much longer distances (300m SDR, 150m DDR). Even PCB InfiniBand setups are possible to allow data communication over short distances using the IB architecture.

To increase the bandwidth, multiple IB lanes can be combined within one wire. Typical sizes are 4 lanes (4x) (8 wires in total in each direction) or 12 lanes (12x) (24 wires in total in each direction).

<table>
<thead>
<tr>
<th>Signaling rate</th>
<th>Encoding</th>
<th>Theoretical throughput 1x</th>
<th>Theoretical throughput 4x</th>
<th>Theoretical throughput 12x</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDR</td>
<td>DDR</td>
<td>QDR</td>
<td>FDR</td>
<td>EDR</td>
</tr>
<tr>
<td>2.5 Gbit/s</td>
<td>8/10</td>
<td>2</td>
<td>14.0625</td>
<td>64/66</td>
</tr>
<tr>
<td>5 Gbit/s</td>
<td>8/10</td>
<td>4</td>
<td>13.64</td>
<td>25</td>
</tr>
<tr>
<td>10 Gbit/s</td>
<td>8/10</td>
<td>8</td>
<td>16</td>
<td>100</td>
</tr>
<tr>
<td>14.0625 Gbit/s</td>
<td>64/66</td>
<td>32</td>
<td>54.54</td>
<td>600</td>
</tr>
<tr>
<td>25.78125 Gbit/s</td>
<td>64/66</td>
<td>96</td>
<td>163.64</td>
<td>300</td>
</tr>
<tr>
<td>50 Gbit/s</td>
<td>64/66</td>
<td>163.64</td>
<td>300</td>
<td>600</td>
</tr>
</tbody>
</table>

2.2.3. Using InfiniBand

Due to the structure of InfiniBand, an application has direct access to the message transport service provided by the HCA, no additional operating system calls are needed. For this, the IB architecture also defines a software transport interface using so-called verbs methods to provide the IB functionality to an application, while hiding the internal network handling. An application can create a work request (WR) to a specific queue pair to send or receive messages. The verbs are not a real, usable API, but rather a specification of necessary behaviour. The most popular verbs implementation is provided by the OpenFabrics Alliance (OFA) [2]. All software components by the OFA can be downloaded in a combined package named Open Fabrics Enterprise Distribution, see Figure 2. (OFED). [4]

Within an application, the verbs API of the OFED package is directly usable to send and receive messages within an InfiniBand environment. Good examples of this mechanism are the example programs shipped with the OFED. One example is `ibv_rc_pingpong`, a small end to end InfiniBand communication test. On many InfiniBand systems this test is directly available as a command-line program or can be easily compiled.

Example of the `ibv_rc_pingpong` execution on the JURECA system between two login nodes (jrl07 and jrl01):

```
jrl07> ibv_rc_pingpong
  local address: LID 0x00e7, QPN 0x0001c4, PSN 0x08649e, GID ::
  remote address: LID 0x00b5, QPN 0x0001c2, PSN 0x08ea42, GID ::
  8192000 bytes in 0.01 seconds = 10979.39 Mbit/sec
  1000 iters in 0.01 seconds = 5.97 usec/iter
jjrl01> ibv_rc_pingpong jrl07
  local address: LID 0x00b5, QPN 0x0001c2, PSN 0x08ea42, GID ::
  remote address: LID 0x00e7, QPN 0x0001c4, PSN 0x08649e, GID ::
  8192000 bytes in 0.01 seconds = 11369.88 Mbit/sec
  1000 iters in 0.01 seconds = 5.76 usec/iter
```

The program sends a fixed amount of bytes between two InfiniBand endpoints. The endpoints are addressed by three values: LID, the local identifier, which is unique for each individual port, QPN, the Queue Pair Number.
Best Practice Guide -
Modern Interconnects

to point to the correct queue and a PSN, a Packet Sequence Number which represents each individual package. Using the analogy of a common TCP/IP connection, the LID value is equivalent to the IP, QPN the port and PSN the TCP sequence number. Within the example these values are not passed to the application directly, instead a regular DNS hostname is used. This is possible because the `ibv_rc_pingpong` uses a initial regular socket connection to exchange all InfiniBand information between both nodes. A full explanation of the example program can be found in [7].

Most of the time the verbs API is not used directly. It offers a high degree of control over an HCA device but it is also very complex to implement within a application. Instead, so-called Upper Layer Protocols (ULP) are used. Implementations of most common ULPs are also part of the OFED:

- **SDP - Sockets Direct Protocol**: Allows regular socket communication on top of IB
- **IPoIB - IP over InfiniBand**: Allows tunneling of IP packages through a IB network. This can also be used to address regular Ethernet devices
- **MPI**: Support for MPI function calls

**Figure 2. OFED components overview**

![OFED components overview](image)

2.3. Aries

Aries is a proprietary network solution developed by Cray and is used together with the dragonfly topology. The main component of Aries itself is a device which contains four NICS to connect individual compute nodes using PCI-Express, as well as a 48 port router to connect one Aries device to other devices. This rank-1 connection is established directly by a backplane PCP board. Within a Cray system setup, 16 Aries routers are connected by one backplane to one chassis. Six chassis create one (electrical) group (using copper wires) within a top level all-to-all topology, which connects individual groups using optical connections. The P2P bandwidth within a dragonfly Aries setup is 15GB/s. [9]
2.4. NVLink

When NVIDIA launched the Pascal GP100 GPU in 2016 and associated Tesla cards, one of the consequences of their increased multi-GPU server was that interconnect bandwidth and latency of PCI Express became an issue. NVIDIA’s goals for their platform began outpacing what PCIe could provide in terms of raw bandwidth. As a result, for their compute focused GPUs, NVIDIA introduced a new high-speed interface, NVLink. Focused on the high-performance application space, it provides GPU-to-GPU data transfers at up to 160 Gbytes/s of bidirectional bandwidth, 5x the bandwidth of PCIe Gen 3x16. Figure 4 below shows NVLink connecting eight Tesla P100 in a Hybrid Cube Mesh Topology as used in the DGX1 server like the Deep Learning System at LRZ.

Figure 4. Schematic representation of NVLink connecting eight Pascal P100 GPUs (DGX1)

- Two fully connected quads, connected at corners.
- 160 GB/s per GPU bidirectional to Peers (GPU-to-GPU)
- Load/store access to Peer Memory
- Full atomics to peer GPUs
- High speed copy engines for bulk data copy
- PCIe to/from CPU-to-GPU

2.4.1. NVLink performance overview

NVLink technology addresses this interconnect issue by providing higher bandwidth, more links, and improved scalability for multi-GPU and multi-GPU/CPU system configurations. Figure 5 below shows the performance for various applications, demonstrating very good performance scalability. Taking advantage of these technologies also allows for greater scalability of ultrafast deep learning training applications, like Caffe.

Figure 5. Performance scalability of various applications with eight P100s connected via NVLink. Source: NVIDIA

2.4.2. GPUDirect

**GPUDirect P2P:** Peer-to-Peer communication transfers and allows memory access between GPUs.

- Intra node
- GPUs both master and slave
- Over NVLink or PCIe

**Peer Device Memory Access:** The function is: __host__ cudaError_t cudaDeviceCanAccessPeer (int* canAccessPeer, int device, int peerDevice) where the parameters are:

  - canAccessPeer: Returned access capability
  - device: Device from which allocations on peerDevice are to be directly accessed.
  - peerDevice: Device on which the allocations to be directly accessed by device reside.

Returns in *canAccessPeer a value of 1 if the device is capable of directly accessing memory from peerDevice and 0 otherwise. If direct access of peerDevice from the device is possible, then access may be enabled by calling cudaDeviceEnablePeerAccess().

To enable peer access between GPUs on the code, these need to be added:
• checkCudaErrors(cudaSetDevice(gpuid[0]));
• checkCudaErrors(cudaDeviceEnablePeerAccess(gpuid[1], 0));
• checkCudaErrors(cudaSetDevice(gpuid[1]));
• checkCudaErrors(cudaDeviceEnablePeerAccess(gpuid[0], 0));

GPUDirect RDMA: RDMA enables a direct path for communication between the GPU and a peer device via PCI Express. GPUDirect and RDMA enable direct memory access (DMA) between GPUs and other PCIe devices.

• Intra node
• GPU slave, third party device master
• Over PCIe

GPUDirect Async: GPUDirect Async entirely relates to moving control logic from third-party devices to the GPU.

• GPU slave, third party device master
• Over PCIe

2.4.3. NVSwitch

NVSwitch is a fully connected NVLink and corresponds to an NVLink switch chip with 18 ports of NVLink per switch. Each port supports 25 GB/s in each direction. The first NVLink is a great advance to enable eight GPUs in a single server, and accelerate performance beyond PCIe. But looking to the high performance level of applications and taking deep learning performance to the next level will require an interface fabric that enables more GPUs in a single server, and full-bandwidth connectivity between them. The immediate goal of NVSwitch is to increase the number of GPUs in a cluster, enabling larger GPU server systems with 16 fully-connected GPUs (see schematic shown below) and 24X more inter-GPU bandwidth than 4X InfiniBand ports, hence allowing much greater work to be done on a single server node. Such an architecture can deliver significant performance advantages for both high performance computing and machine learning environments when compared with conventional CPU-based systems. Internally, the processor is an 18 x 18-port, fully-connected crossbar. The crossbar is non-blocking, allowing all ports to communicate with all other ports at the full NVLink bandwidth, and drive simultaneous communication between all eight GPU pairs at an incredible 300 GB/s each.

Figure 6. HGX-2 architecture correspond of two eight-GPU baseboards, each outfitted with six 18-port NVSwitches. Source: NVIDIA

Figure 6 above shows an NVIDIA HGX-2 system comprised of 12 NVSwitches, which are used to fully connect 16 GPUs (two eight-GPU baseboards), each outfitted with six 18-port NVSwitches. Communication between the baseboards is enabled by a 48-NVLink interface. The switch-centric design enables all the GPUs to converse with one another at a speed of 300 GB/second -- about 10 times faster than what would be possible with PCI-Express. And fast enough to make the distributed memory act as a coherent resource, given the appropriate system software.
Not only does the high-speed communication make it possible for the 16 GPUs to treat each others' memory as its own, it also enables them to behave as one large aggregated GPU resource. Since each of the individual GPUs has 32 GB of local high bandwidth memory, an application is able to access 512 GB at a time. And because these are V100 devices, that same application can tap into two petaflops of deep learning (tensor) performance or, for HPC, 125 teraflops of double precision or 250 teraflops of single precision. A handful of extra teraflops are also available from the PCIe-linked CPUs, in either a single-node or dual-node configuration (two or four CPUs, respectively).

Figure 7. Schematic representation of NVSwitch first on-node switch architecture to support 16 fully-connected GPUs in a single server node. Source: NVIDIA

A 16-GPU Cluster (see Figure 7) offers multiple advantages:

• It reduces network traffic hot spots that can occur when two GPU-equipped servers are exchanging data during a neural network training run.

• With an NVSwitch-equipped server like NVIDIA’s DGX-2, those exchanges occur on-node, which also offers significant performance advantages.

• It offers a simpler, single-node programming model that effectively abstracts the underlying topology.

2.5. Tofu

Tofu (Torus fusion) is a network designed by Fujitsu for the K computer at the RIKEN Advanced Institute for Computational Science in Japan. Tofu uses a 6D mesh/torus mixed topology (combining a 3D torus with a 3D mesh structure (2x3x2) on each node of the torus structure). This structure highly supports topology-aware communication algorithms. Currently Tofu2 is used in production and TofuD is planned for the post-K machine. [12]

Tofu2 has a link speed of 100 Gbit/s and each node is connected by ten links (optical and electrical) to other nodes. [11]

2.6. Ethernet

Ethernet is the most common network technology. Beside its usage in nearly every local network setup it is also the most common technology used in HPC systems. In the November 2018 TOP500 list [1] 50% of all systems used a Ethernet-based interconnection for the main network communication. Even more systems use Ethernet for backend or administration purposes.

The ethernet standard supports copper as well as fiber optic wires and can reach up to 400 Gbit/s. The most common HPC setup uses 10GBit/s (23.8% of the November 2018 TOP500 list) for their overall point-to-point communication due to the relatively cheap network components involved.

Compared to Infiniband or Omnipath the Ethernet software stack is much larger due to its variety of use cases, supported protocols and network components. Ethernet uses a hierarchical topology which involves more com-
puting power by the CPU, in contrast to the flat fabric topology of Infiniband where the data is directly moved by the network card using RDMA requests, reducing the CPU involvement.

Ethernet uses a fixed naming scheme (e.g. 10BASE-T) to determine the connection type:

- 10, 100, 1000, 10G, ...: Bandwidth (MBit/s/GBit/s)
- BASE, BROAD, PASS: Signal band
- -T, -S, -L, -B, ...: Cable and transfer type: T = twisted pair cable, S = short wavelength (fiber), L = long wavelength (fiber), B = bidirectional fiber
- X.R: Line code (normally Manchester code)
- 1, 2, 4, 10: Number of lanes

2.7. Interconnect selection

Some systems offer multiple ways to communicate data between different network elements (e.g. within an InfiniBand environment data can be transferred directly through an IB network connection or through TCP using IPoIB). Typically the interface selection job is handled by the MPI library, but can also be controlled by the user:

_OpenMPI_: `--mca btl controls the point-to-point byte transfer layer, e.g. mpirun ... --mca btl sm,openib,self` can be used to select the shared memory, the InfiniBand and the loopback interface.

:IntelMPI: The environment variable `I_MPI_FABRICS` can be used (`I_MPI_FABRICS=<fabric>|<intra-node fabric>:<inter-nodes fabric>`). Valid fabrics are [8]:

- _shm_: Shared-memory
- _dapl_: DAPL-capable network fabrics, such as InfiniBand, iWarp, Dolphin, and XPMEM (through DAPL)
- _tcp_: TCP/IP-capable network fabrics, such as Ethernet and InfiniBand (through IPoIB)
- _tmi_: Network fabrics with tag matching capabilities through the Tag Matching Interface (TMI), such as Intel(R) True Scale Fabric and Myrinet
- _ofa_: Network fabric, such as InfiniBand (through OpenFabrics Enterprise Distribution (OFED) verbs) provided by the Open Fabrics Alliance (OFA)
- _ofi_: OFI (OpenFabrics Interfaces)-capable network fabric including Intel(R) True Scale Fabric, and TCP (through OFI API)
3. Common HPC network topologies

3.1. Fat trees

Fat trees are used to provide a low latency network between all involved nodes. It allows for a scalable hardware setup without the need for many ports per node or many individual connections. Locality is only a minor aspect within a fat tree setup, as most node-to-node connections provide a similar latency and a similar or equal bandwidth.

All computing nodes are located on the leaves of the tree structure and switches are located on the inner nodes.

The idea of the fat tree is, instead of using the same wire "thickness" for all connections, to have "thicker" wires closer to the top of the tree and provide a higher bandwidth over the corresponding switches. In an ideal fat tree the aggregated bandwidth of all connections from a lower tree level must be combined in the connection to the next higher tree level. This setup is not always achieved, in such a situation the tree is called a pruned fat tree.

Depending on the number of tree levels, the highest latency of a node to node communication can be calculated. Typical fat tree setups uses two or three levels for the tree.

Figure 8. Example of a full fat tree

Figure 8 shows an example of a two level full fat tree using two switches on the core and four on the edge level. Between the edge and the core level each connection represents 4 times the bandwidth of a single link.

3.2. Torus

A torus network allows a multi-dimensional connection setup between individual nodes. In such a setup each node normally has a limited number of directly connected neighbours. Other nodes, which are not direct neighbours, are reachable by moving the data from node to node. Normally each node has the same number of neighbour nodes, so in a list of nodes, the last node is connected to the first node, which forms the torus structure.

Depending on the number of neighbours and individual connections the torus forms a one, two, three or even more dimensional structure. Figure 9 shows the example wiring of such torus layouts. Often structures are combined, e.g. by having a group of nodes in a three dimensional torus, and each group itself is connected to other groups in a three dimensional torus as well.

Due to its structure, a torus allows for very fast neighbour-to-neighbour communication. In addition, there can be different paths for moving data between two different nodes. The latency is normally quite low and can be distinguished by the maximum number of hops needed. As there is no direct central element, the number of individual wires can be very high. In an N-dimensional torus, each node has 2N individual connections.
3.3. Hypercube

The hypercube and torus setup are very similar and are often used within the same network setup on different levels. The main difference between a pure hypercube and a torus are the number of connections of the border nodes. Within the torus setup each network node has the same number of neighbour nodes, while in the hypercube only the inner nodes have the same number of neighbours and the border nodes have fewer neighbours. This allows for fewer required network hops in a torus network if border nodes need to communicate. This allows a torus network to be faster than a pure hypercube setup [10].

3.4. Dragonfly

The dragonfly topology is mostly used within Cray systems. It consists of two major components: a number of network groups, where multiple nodes are connected within a common topology like torus or fat tree, and, on top of these groups, a direct one-to-one connection between each group. This setup avoids the need for external top level switches due to the direct one-to-one group connection. Within each group a router controls the connection to the other groups. The top-level one-to-one connections can reduce the number of needed hops in comparison to a full fat tree (for this an optimal package routing is necessary). [9]

Figure 10. Dragonfly network topology, source: wikimedia
4. Production environment

The following sections provide an overview of the current generation of PRACE Tier-0 systems and their specific network setups.

4.1. CURIE

<table>
<thead>
<tr>
<th>Hosting site</th>
<th>CEA_TGCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-setup</td>
<td></td>
</tr>
<tr>
<td>• CPUs: 2x 8-cores <a href="mailto:SandyBridge@2.7GHz">SandyBridge@2.7GHz</a> (AVX)</td>
<td></td>
</tr>
<tr>
<td>• Cores/Node: 16</td>
<td></td>
</tr>
<tr>
<td>• RAM/Node: 64GB</td>
<td></td>
</tr>
<tr>
<td>• RAM/Core: 4GB</td>
<td></td>
</tr>
<tr>
<td>#Nodes</td>
<td>5040</td>
</tr>
<tr>
<td>Type of major communication interconnect</td>
<td>InfiniBand QDR</td>
</tr>
<tr>
<td>Topology type(s)</td>
<td>Full Fat Tree</td>
</tr>
<tr>
<td>Theoretical max. bandwidth per node</td>
<td>25.6 Gbit/s</td>
</tr>
<tr>
<td>Interconnect type for filesystem</td>
<td>InfiniBand QDR</td>
</tr>
<tr>
<td>Theoretical max. bandwidth for major (SCRATCH) parallel filesystem</td>
<td>1200 Gbit/s</td>
</tr>
</tbody>
</table>

Possible available interconnect configuration options SLURM submission option “--switches” for user

4.2. Hazel Hen

<table>
<thead>
<tr>
<th>Hosting site</th>
<th>HLRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-setup</td>
<td></td>
</tr>
<tr>
<td>• Processor: Intel Haswell E5-2680v3 2.5 GHz, 12 Cores, 2 HT/Core</td>
<td></td>
</tr>
<tr>
<td>• Memory per Compute Node: 128GB DDR4</td>
<td></td>
</tr>
<tr>
<td>#Nodes</td>
<td>7712 (dual socket)</td>
</tr>
<tr>
<td>Type of major communication interconnect</td>
<td>Cray Aries network</td>
</tr>
<tr>
<td>Topology type(s)</td>
<td>Dragonfly, Cray XC40 has the complex tree like architecture and topology between multiple levels. It consists of the following buildings blocks:</td>
</tr>
<tr>
<td>• Aries Rank-0 Network: blade (4 compute nodes)</td>
<td></td>
</tr>
<tr>
<td>• Aries Rank-1 Network: chassis (16 blades)</td>
<td></td>
</tr>
<tr>
<td>• Aries Rank-2 Network: group (2 cabinets = 6 chassis) (connection within a group, Passive Electrical Network)</td>
<td></td>
</tr>
<tr>
<td>• Aries Rank-3 Network: system (connection between groups, Active Optical Network)</td>
<td></td>
</tr>
<tr>
<td>Theoretical max. bandwidth per node</td>
<td>Depending on communication distance: ~ 15 GB/s</td>
</tr>
<tr>
<td>Interconnect type for filesystem</td>
<td>InfiniBand QDR</td>
</tr>
<tr>
<td>Theoretical max. bandwidth for major (SCRATCH) parallel filesystem</td>
<td>ws7/8: 640Gbit/s each, ws9: 1600Gbit/s</td>
</tr>
</tbody>
</table>
Possible available interconnect configuration options

### 4.3. JUWELS

<table>
<thead>
<tr>
<th>Hosting site</th>
<th>Jülich Supercomputing Centre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-setup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 2271 standard compute nodes - Dual Intel Xeon Platinum 8168, 96 GB RAM</td>
</tr>
<tr>
<td></td>
<td>• 240 large memory compute nodes - Dual Intel Xeon Platinum 8168, 192 GB RAM</td>
</tr>
<tr>
<td></td>
<td>• 48 accelerated compute nodes - Dual Intel Xeon Gold 6148, 4x Nvidia V100 GPU, 192 GB RAM</td>
</tr>
<tr>
<td>#Nodes</td>
<td>2559</td>
</tr>
<tr>
<td>Type of major communication interconnect</td>
<td>InfiniBand EDR (L1+L2) HDR (L3)</td>
</tr>
<tr>
<td>Topology type(s)</td>
<td>Fat Tree (prune 2:1 on first level)</td>
</tr>
<tr>
<td>Theoretical max. bandwidth per node</td>
<td>100 Gbit/s</td>
</tr>
<tr>
<td>Interconnect type for filesystem</td>
<td>• 40 Gbit/s ethernet</td>
</tr>
<tr>
<td></td>
<td>• InfiniBand EDR</td>
</tr>
<tr>
<td>Theoretical max. bandwidth for major (SCRATCH) 2880 Gbit/s parallel filesystem</td>
<td>None</td>
</tr>
</tbody>
</table>

Possible available interconnect configuration options None
for user

### 4.4. Marconi

<table>
<thead>
<tr>
<th>Hosting site</th>
<th>Cineca</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-setup</td>
<td>• 720 nodes, each node 2x18core Intel Broadwell at 2.3GHz, 128 GB RAM/node</td>
</tr>
<tr>
<td></td>
<td>• 3600 nodes, each node 1x68 core Intel KNL at 1.4 GHz, 96GB (DDR4)+16 GB(HBM)/node.</td>
</tr>
<tr>
<td></td>
<td>• 2304 nodes, 2x24 core Intel Skylake ar 2.10 GHz, 192 GB RAM/node</td>
</tr>
<tr>
<td>#Nodes</td>
<td>6624</td>
</tr>
<tr>
<td>Type of major communication interconnect</td>
<td>Omnipath</td>
</tr>
<tr>
<td>Topology type(s)</td>
<td>Fat Tree</td>
</tr>
<tr>
<td>Theoretical max. bandwidth per node</td>
<td>100 Gbit/s</td>
</tr>
<tr>
<td>Interconnect type for filesystem</td>
<td>Omnipath</td>
</tr>
<tr>
<td>Theoretical max. bandwidth for major (SCRATCH) 640 Gbit/s parallel filesystem</td>
<td>None</td>
</tr>
<tr>
<td>Possible available interconnect configuration options None</td>
<td></td>
</tr>
</tbody>
</table>

for user

### 4.5. MareNostrum

<table>
<thead>
<tr>
<th>Hosting site</th>
<th>Barcelona Supercomputing Center</th>
</tr>
</thead>
</table>

Node-setup

- 3240 standard compute nodes - Dual Intel Xeon Platinum 8160 CPU with 24 cores each @ 2.10GHz for a total of 48 cores per node, 96 GB RAM
- 216 large memory compute nodes - Dual Intel Xeon Platinum 8160 CPU with 24 cores each @ 2.10GHz for a total of 48 cores per node, 384 GB RAM

#Nodes 3456

Type of major communication interconnect Intel Omni-Path

Topology type(s) Full-Fat tree (Non-blocking / CBB 1:1)

Theoretical max. bandwidth per node 100 Gbit/s

Interconnect type for filesystem Ethernet (40 Gbit/s)

Theoretical max. bandwidth for major (SCRATCH) 960 Gbits/s

Possible available configuration options for user None

Figure 11. Network diagram of MareNostrum 4

4.6. PizDaint

Hosting site CSCS

Node-setup

- 5319 nodes, Intel Xeon E5-2690 v3; NVIDIA Tesla P100-PCIE-16GB; 64 GB of RAM
- 1175 nodes, Intel Xeon E5-2695 v4; 64 GB of RAM
- 640 nodes, Intel Xeon E5-2695 v4; 128 GB of RAM

#Nodes 7134

Type of major communication interconnect Aries routing and communications ASIC (proprietary)

Topology type(s) Dragonfly

Theoretical max. bandwidth per node 43.12 Gbit/s

Interconnect type for filesystem Infiniband

Theoretical max. bandwidth for major (SCRATCH)

- Sonexion 3000: 112 GB/s = 896 Gbits/s
- Sonexion 1600: 138 GB/s = 1104 Gbits/s

Possible available interconnect configuration options for user None
## 4.7. SuperMUC

<table>
<thead>
<tr>
<th>Hosting site</th>
<th>LRZ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node-setup</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1: Sandy Bridge-EP Xeon E5-2680 8C, 16 cores per node, RAM 32 GBytes</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 2: Haswell Xeon Processor E5-2697 v3, 28 cores per node, RAM 64 GBytes</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: Intel Skylake EP, 48 cores per node and RAM 96 GBytes</td>
<td></td>
</tr>
<tr>
<td><strong>#Nodes</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1: 9216 distributed in 12 Islands of 512 nodes.</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 2: 3072 distributed in 6 Islands of 512 nodes.</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: 6336</td>
<td></td>
</tr>
<tr>
<td><strong>Type of major communication interconnect</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1: Infiniband FDR10</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 2: Infiniband FDR14</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: Omnipath</td>
<td></td>
</tr>
<tr>
<td><strong>Topology type(s)</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1 &amp; 2: All computing nodes within an individual island are connected via a fully non-blocking Infiniband network. Above the island level, the pruned interconnect enables a bi-directional bi-section bandwidth ratio of 4:1 (intra-island / inter-island).</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: Fat-tree</td>
<td></td>
</tr>
<tr>
<td><strong>Theoretical max. bandwidth per node</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1: 40Gbit/s</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 2: 55Gbit/s</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: TBD</td>
<td></td>
</tr>
<tr>
<td><strong>Interconnect type for filesystem</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1: Infiniband FDR10</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 2: Infiniband FDR14</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: Omnipath</td>
<td></td>
</tr>
<tr>
<td><strong>Theoretical max. bandwidth for major (SCRATCH) parallel filesystem</strong></td>
<td></td>
</tr>
<tr>
<td>• SuperMUC Phase 1 &amp; 2: 1040 Gbit/s for writing and 1200 Gbit/s for reading operations.</td>
<td></td>
</tr>
<tr>
<td>• SuperMUC-NG: At least 4000 Gbit/s</td>
<td></td>
</tr>
</tbody>
</table>

Possible available interconnect configuration options None for user
5. Network benchmarking

5.1. IMB

Intel(R) MPI Benchmarks provides a set of elementary benchmarks that conform to the MPI-1, MPI-2, and MPI-3 standards. In addition to its use as a test for MPI setups, it is particularly useful for testing the general network capabilities of point-to-point communication and even more useful for testing the capabilities of collective operations.

5.1.1. Installation

IMB is available on GitHub at https://github.com/intel/mpi-benchmarks

A full user guide can be found here: https://software.intel.com/en-us/imb-user-guide

By default, Intel officially supports the Intel compiler together with Intel MPI, but other combinations can also be used for benchmarking.

A simple `make` can be used to build all benchmarks.

The benchmark is divided into three main parts using different executables to test different MPI standard operations:

- **MPI-1**
  - IMB-MPI1: MPI-1 functions

- **MPI-2**
  - IMB-EXT: MPI-2 extension tests
  - IMB-IO: MPI-IO operations

- **MPI-3**
  - IMB-NBC: Nonblocking collective (NBC) operations
  - IMB-RMA: One-sided communications benchmarks that measure the Remote Memory Access (RMA) functionality

For a pure network test the IMB-MPI1 executable is the best option.

5.1.2. Benchmark execution

The benchmark can be started within a basic MPI environment. E.g. for SLURM:

```
srun -n24 ./IMB-MPI1
```

By default, different MPI operations are automatically tested using different data sizes on different numbers of processes, e.g. `(-n24 means 2,4,8,16,24 processes will be tested for each run). Multiple iterations are used.

Additional arguments are available and can be found here: https://software.intel.com/en-us/imb-user-guide-command-line-control

E.g. `-npmin 24` can fix the minimum number of processes.

5.1.3. Benchmark output

For each test, a table showing the relevant execution times is created. The following run was performed on one JURECA nodes at Juelich Supercomputing Centre:
# Benchmarking Alltoall
# processes = 24

<table>
<thead>
<tr>
<th>#bytes</th>
<th>#repetitions</th>
<th>t_min[usec]</th>
<th>t_max[usec]</th>
<th>t_avg[usec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000</td>
<td>0.05</td>
<td>0.11</td>
<td>0.08</td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>7.34</td>
<td>8.45</td>
<td>8.04</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>7.34</td>
<td>8.52</td>
<td>8.09</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>7.37</td>
<td>8.54</td>
<td>8.11</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8.03</td>
<td>9.25</td>
<td>8.85</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>7.98</td>
<td>9.66</td>
<td>9.02</td>
</tr>
<tr>
<td>32</td>
<td>1000</td>
<td>7.98</td>
<td>9.87</td>
<td>9.08</td>
</tr>
<tr>
<td>64</td>
<td>1000</td>
<td>8.71</td>
<td>10.47</td>
<td>9.69</td>
</tr>
<tr>
<td>128</td>
<td>1000</td>
<td>9.90</td>
<td>11.59</td>
<td>10.87</td>
</tr>
<tr>
<td>256</td>
<td>1000</td>
<td>11.87</td>
<td>13.82</td>
<td>13.06</td>
</tr>
<tr>
<td>512</td>
<td>1000</td>
<td>17.19</td>
<td>19.49</td>
<td>18.58</td>
</tr>
<tr>
<td>1024</td>
<td>1000</td>
<td>22.46</td>
<td>28.16</td>
<td>26.53</td>
</tr>
<tr>
<td>2048</td>
<td>1000</td>
<td>41.92</td>
<td>51.52</td>
<td>47.69</td>
</tr>
<tr>
<td>4096</td>
<td>1000</td>
<td>76.22</td>
<td>86.77</td>
<td>81.16</td>
</tr>
<tr>
<td>8192</td>
<td>1000</td>
<td>111.76</td>
<td>114.15</td>
<td>113.02</td>
</tr>
<tr>
<td>16384</td>
<td>1000</td>
<td>210.43</td>
<td>213.60</td>
<td>212.12</td>
</tr>
<tr>
<td>32768</td>
<td>1000</td>
<td>506.37</td>
<td>540.70</td>
<td>524.83</td>
</tr>
<tr>
<td>65536</td>
<td>640</td>
<td>825.75</td>
<td>841.05</td>
<td>832.70</td>
</tr>
<tr>
<td>131072</td>
<td>320</td>
<td>1942.28</td>
<td>1976.38</td>
<td>1962.48</td>
</tr>
<tr>
<td>262144</td>
<td>160</td>
<td>3786.24</td>
<td>3831.52</td>
<td>3809.20</td>
</tr>
<tr>
<td>524288</td>
<td>80</td>
<td>7445.17</td>
<td>7549.67</td>
<td>7501.71</td>
</tr>
<tr>
<td>1048576</td>
<td>40</td>
<td>14939.59</td>
<td>15139.40</td>
<td>15031.11</td>
</tr>
<tr>
<td>2097152</td>
<td>20</td>
<td>18800.68</td>
<td>37493.79</td>
<td>33221.16</td>
</tr>
<tr>
<td>4194304</td>
<td>10</td>
<td>61286.47</td>
<td>62799.05</td>
<td>62054.62</td>
</tr>
</tbody>
</table>

5.2. Linktest

The Linktest program is a parallel ping-pong test between all possible MPI connections of a machine. Output of this program is a full communication matrix which shows the bandwidth and message latency between each processor pair and a report including the minimum bandwidth.

5.2.1. Installation

Linktest is available here: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LinkTest/linktest-download_node.html

Linktest uses SIONlib for its data creation process. SIONlib can be downloaded here: http://www.fz-juelich.de/jsc/sionlib

Within the extracted src directory Makefile_LINUX must be adapted by setting SIONLIB_INST to the SIONlib installation directory, and CC, MPICC, and MPIGCC to valid compilers (e.g. all can be set to mpicc).

5.2.2. Benchmark execution

A list of Linktest command-line parameters can be found here: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LinkTest/linktest-using_node.html

For SLURM the following command starts the default setup using 2 nodes and 48 processes:

```
  srun -N2 -n48 ./mpilinktest
```
The run generates a binary SIONlib file `pingpong_results_bin.sion` which can be postprocessed by using the serial `pingponganalysis` tool.

### 5.2.3. Benchmark output

```bash
./pingponganalysis -p pingpong_results_bin.sion
```

creates a postscript report:

**Figure 12. Linktest point to point communication overview**

Within the matrix overview, each point-to-point communication duration is displayed. Here the difference between individual nodes, as well as between individual sockets can easily be displayed (e.g. in the given example in Figure 12 two JURECA nodes with two sockets each are shown).
Further documentation

Websites, forums, webinars


Manuals, papers


