MPI

Setting up MPI Environment

The Salomon cluster provides several implementations of the MPI library:

MPI Library	Thread support
Intel MPI 4.1	Full thread support up to MPI_THREAD_MULTIPLE
Intel MPI 5.0	Full thread support up to MPI_THREAD_MULTIPLE
OpenMPI 1.8.6	Full thread support up to MPI_THREAD_MULTIPLE, MPI-3.0 support
SGI MPT 2.12	

MPI libraries are activated via the environment modules.

Look up section modulefiles/mpi in module avail

```
$ module avail
------------------------------ /apps/modules/mpi -------------------------------
impi/4.1.1.036-iccifort-2013.5.192
impi/4.1.1.036-iccifort-2013.5.192-GCC-4.8.3
impi/5.0.3.048-iccifort-2015.3.187
impi/5.0.3.048-iccifort-2015.3.187-GNU-5.1.0-2.25
MPT/2.12
OpenMPI/1.8.6-GNU-5.1.0-2.25
```

There are default compilers associated with any particular MPI implementation. The defaults may be changed, the MPI libraries may be used in conjunction with any compiler. The defaults are selected via the modules in following way

Module	MPI	Compiler suite
impi-5.0.3.048-iccifort-	Intel MPI 5.0.3	2015.3.187
OpenMP-1.8.6-GNU-5.1.0-2	OpenMPI 1.8.6	.25

Examples:

```
$ module load gompi/2015b
```

In this example, we activate the latest OpenMPI with latest GNU compilers (OpenMPI 1.8.6 and GCC 5.1). Please see more information about toolchains in section Environment and Modules .

To use OpenMPI with the intel compiler suite, use

```
$ module load iompi/2015.03
```

In this example, the openmpi 1.8.6 using intel compilers is activated. It's used

"iompi" toolchain.

Compiling MPI Programs

After setting up your MPI environment, compile your program using one of the mpi wrappers

```
$ mpicc -v
$ mpif77 -v
$ mpif90 -v
```

When using Intel MPI, use the following MPI wrappers:

```
$ mpicc
$ mpiifort
```

Wrappers mpif90, mpif77 that are provided by Intel MPI are designed for gcc and gfortran. You might be able to compile MPI code by them even with Intel compilers, but you might run into problems (for example, native MIC compilation with -mmic does not work with mpif90).

Example program:

```
// helloworld_mpi.c
#include <stdio.h>

#include<mpi.h>

int main(int argc, char **argv) {

int len;
int rank, size;
char node[MPI_MAX_PROCESSOR_NAME];

// Initiate MPI
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

// Get hostame and print
MPI_Get_processor_name(node,&len);
printf("Hello world! from rank %d of %d on host %sn",rank,size,node);

// Finalize and exit
MPI_Finalize();

return 0;
}
```

Compile the above example with

```
$ mpicc helloworld_mpi.c -o helloworld_mpi.x
```

Running MPI Programs

The MPI program executable must be compatible with the loaded MPI module. Always compile and execute using the very same MPI module.

It is strongly discouraged to mix mpi implementations. Linking an application with one MPI implementation and running mpirun/mpiexec form other implementation may result in unexpected errors.

The MPI program executable must be available within the same path on all nodes. This is automatically fulfilled on the /home and /scratch filesystem. You need to preload the executable, if running on the local scratch /lscratch filesystem.

Ways to run MPI programs

Optimal way to run an MPI program depends on its memory requirements, memory access pattern and communication pattern.

Consider these ways to run an MPI program: 1. One MPI process per node, 24 threads per process 2. Two MPI processes per node, 12 threads per process 3. 24 MPI processes per node, 1 thread per process.

One MPI** process per node, using 24 threads, is most useful for memory demanding applications, that make good use of processor cache memory and are not memory bound. This is also a preferred way for communication intensive applications as one process per node enjoys full bandwidth access to the network interface.

Two MPI** processes per node, using 12 threads each, bound to processor socket is most useful for memory bandwidth bound applications such as BLAS1 or FFT, with scalable memory demand. However, note that the two processes will share access to the network interface. The 12 threads and socket binding should ensure maximum memory access bandwidth and minimize communication, migration and numa effect overheads.

Important! Bind every OpenMP thread to a core!

In the previous two cases with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You want to avoid this by setting the KMP_AFFINITY or GOMP_CPU_AFFINITY environment variables.

24 MPI processes per node, using 1 thread each bound to processor core is most suitable for highly scalable applications with low communication demand.

Running OpenMPI

The **OpenMPI 1.8.6** is based on OpenMPI. Read more on how to run OpenMPI based MPI.

The Intel MPI may run on theIntel Xeon Phi accelerators as well. Read more on how to run Intel MPI on accelerators.