Compiling code and using MPI

scitas.epfl.ch

October 9, 2019

Welcome

What you will learn

How to compile and launch MPI codes on the SCITAS clusters along with a bit of the "why"

What you will not learn

How to write parallel code and optimise it - there are other courses for that!

Compilation

From code to binary

Compilation is the process by which code (C, C++, Fortran etc) is transformed into a binary that can be run on a CPU.

CPUs are not all the same

- CPUs have different features and instruction sets
- The same code will need to be recompiled for different architectures

What is MPI?

What is MPI?

- Message Passing Interface
- Open standard - now at version 3.1
  → Check this website: http://mpi-forum.org
- De facto standard for distributed memory parallelisation
- Multiple implementations - MVAPICH2, MPICH, IntelMPI ...
- Scales to very large systems

Shared vs Distributed Memory

- Shared - all tasks see all the memory (e.g. OpenMP)
- Distributed - tasks only see a small part of the overall memory

Clusters are distributed memory systems so MPI is well suited.

MPI Terminology

Words that you are going to hear

- Rank - how MPI tasks are organised
- Rank 0 to N - the "worker" tasks
- Hybrid - a code that combines shared memory parallelisation with MPI

Pure MPI codes generally run one rank per core.

Compilers - Intel vs GCC

GNU Compiler Collection

- The industry standard and available everywhere
- Quick to support new C++ language features
- Fortran support used to be poor

Intel Composer

- Claims to produce faster code on Intel CPUs
- Better Fortran support
- Generally much stricter by default with bad code!

MPI - Intel vs MVAPICH2 vs OpenMPI

Why are there different flavours?

There are multiple MPI flavours that comply with the specification and each claims to have some advantage over the other. Some are vendor specific and others are open source

The main contenders

- Intel MPI - commercial MPI with support
- MVAPICH2 - developed by Ohio uni for Infiniband
- OpenMPI - Open source and widely used

In SCITAS we support IntelMPI, MVAPICH2 and OpenMPI

We *recommend* IntelMPI or MVAPICH2!

Compiler and MPI choice

First choose your compiler

- GCC or Intel
- This might be a technical or philosophical choice

The associated MPI is then

- GCC with MVAPICH2
- GCC with OpenMPI *if you have a very good reason*
- Intel with IntelMPI

This is a SCITAS restriction to prevent chaos - nothing technically stops one from mixing!

Both work well and have good performance.

Linking

Let someone else do the hard work

For nearly everything that you want to do there's already a library function.

How to use libraries

Linking is the mechanism by which you can use libraries with your code.

- static - put everything in your executable
- dymanic - keep the libraries outside and load them as needed

Dynamic by default

There are very few reasons to statically link code.

What is linked?

ldd is your friend

```
ldd mycode.x

libmpifort.so.12 => /ssoft/intelmpi/5.1.1/RH6/all/x86_E5v2/impi/5.1.1.109/lib64/libmpifort.so.12
libmpi.so.12 => /ssoft/intelmpi/5.1.1/RH6/all/x86_E5v2/impi/5.1.1.109/lib64/libmpi.so.12
libdl.so.2 => /lib64/libdl.so.2
librt.so.1 => /lib64/librt.so.1
libpthread.so.0 => /lib64/libpthread.so.0
libm.so.6 => /lib64/libm.so.6
libgcc_s.so.1 => /lib64/libgcc_s.so.1
libc.so.6 => /lib64/libc.so.6
```

The dark art of mangling

Mangling and decoration

Mechanism to allow multiple functions with the same name but as there is no standard ABI things can get tricky

C/C++

- GCC - `_ZN5NOMAD10Eval_PointD2Ev`
- Intel - `_ZN5NOMAD10Eval_PointD2Ev`

Result: C/C++ libraries are compatible between GCC and Intel

Fortran

- GCC - `__h5f_MOD_h5fget_access_plist_f`
- Intel - `h5f_mp_h5fget_access_plist_f_`

Result: Fortran libraries are not compatible between GCC and Intel!

Get the codes

```
$ git clone https://c4science.ch/diffusion/SCUSINGMPI/using-
$ cd using-mpi
```

or copy the exercises from scratch:

```
$ mkdir using-mpi
$ cd using-mpi
$ cp -R /scratch/examples/compiling-and-using-mpi/ex* .
```

Example 1 - Build sequential 'Hello World'

Compile the source files

```
gcc -c output.c
gcc -c hello.c
```

Link

```
gcc -o hello output.o hello.o
```

Run

```
./hello
Hello World!
```

Compilation - the general case

To compile and link we need

- The libraries to link against
- Where to find these libraries
- Where to find their header files
- Your source code
- A nice name for the executable

-l -L and -I

```
gcc -L path_to_libraries -l libraries -I
path_to_header_filer -o name_of_executable mycode.c
```

Sequential 'Hello World' with shared libraries

In case you were wondering...

```
$ gcc -fPIC -c output.c
$ gcc -shared -o liboutput.so output.o
$ pwd
/home/scitas/using-mpi/ex1
$ gcc hello.c -L `pwd` -loutput -I `pwd` -o hi
$ export LD_LIBRARY_PATH=`pwd`:$LD_LIBRARY_PATH
$ ./hi
Hello World!
```

Now try running `ldd` for the executable

Making code run faster

Compiling is hard work..

By default a compiler will not optimise your code!

```
float matest(float a, float b, float c) {
  a = a*b + c;
  return a;
}
```

For the details see:

https://scitas-data.epfl.ch/confluence/display/DOC/Compiling+codes+on+different+systems

No optimisation

icc mycode.c

```
matest(float, float, float):
 push   rbp
 mov    rbp,rsp
 movss  DWORD PTR [rbp-0x4],xmm0
 movss  DWORD PTR [rbp-0x8],xmm1
 movss  DWORD PTR [rbp-0xc],xmm2
 movss  xmm0,DWORD PTR [rbp-0x4]
 mulss  xmm0,DWORD PTR [rbp-0x8]
 addss  xmm0,DWORD PTR [rbp-0xc]
 movss  DWORD PTR [rbp-0x4],xmm0
 mov    eax,DWORD PTR [rbp-0x4]
 mov    DWORD PTR [rbp-0x10],eax
 movss  xmm0,DWORD PTR [rbp-0x10]
 pop    rbp
 ret
```

With optimisation

icc -O3 -xAXV2 mycode.c

```
matest(float, float, float):
 vfmadd132ss xmm0,xmm2,xmm1
 ret
```

Optimisation levels

O1

Enables optimizations for speed and disables some optimizations that increase code size and affect speed

O2

Enables optimizations for speed. This is the generally recommended optimization level. Vectorization is enabled at O2 and higher levels.

O3

Performs O2 optimizations and enables more aggressive loop transformations such as Fusion, Block-Unroll-and-Jam, and collapsing IF statements.

Modules

How software is organised on the clusters

Modules is utility that allows multiple, often incompatible, tools and libraries to exist on a cluster. We use LMod which is an extension of the classical modules tool.

Load modules to see more modules

- module avail
- module load <compiler>

  Ex: `module load intel`
- module avail
- module load <MPI>

  Ex: `module load intel-mpi`
- module avail

Note that there is an associated BLAS library (MKL or OpenBLAS)

More Modules

Commands

- module purge
- module load gcc
- module load mvapich2
- module load hdf5

  → `Or simply: module load gcc mvapich2 hdf5`

- module list
- module help hdf5
- module show hdf5

LMod features

One compiler at a time

- module purge
- module load gcc
- module load hdf5
- module list
- module load intel

Only one module flavour can be loaded at the same time

slmodules

How we manage software

One "release" per year

- slmodules -r deprecated
- slmodules
- slmodules -s foo

By default you see the architecture (`$SYS_TYPE`) of the system you are connected to.

`Future` becomes `stable` and `stable` becomes `deprecated` in July.

MPICC and friends

mpicc / mpiicc / mpicxx / mpif77 / mpif90 / mpiifort

These are wrappers to the underlying compiler that add the correct options to link with the MPI libraries

- mpicc - C wrapper
- mpiicc - Intel C wrapper
- mpiifort - Intel Fortran Compiler

Check the MPI flavour documentation for more details

mpicc mycode.c

To use the wrappers simply type:

- module load mympiflavour/version
- mpicc hello.c -o hi

Example 2 - Build // MPI-based 'Hello World'

Load modules

```
module load intel intel-mpi
```

Compile-link

```
mpiicc -g -o hello_mpi hello_mpi.c
```

Run two tasks on two different nodes

```
srun -N2 -n2 -partition=debug ./hello_mpi
Hello world:  I am task rank 1, running on node ’b292’
Hello world:  I am task rank 2, running on node ’b293’
```

Configure and Make

The traditional way to build packages

- `./configure -help`
- `./configure -prefix=X -option=Y`
- `make`
- `make install`

cmake

cmake is a better way to do things!

- `cmake -DCMAKE_INSTALL_PREFIX:PATH=X -DOption=Y <sources>`
- `make`
- `make install`

If you're starting a project from scratch then we recommend using cmake rather than configure. There's also a graphic interface called `ccmake`.

MPI and the Batch System

Telling SLURM what we need

We would like 64 processes over 4 nodes

```
#SBATCH - -nodes 4
#SBATCH -ntasks-per-node 16
#SBATCH -cpus-per-task 1
#SBATCH -mem 32000
```

Remember that the memory is per node!

Alternative formulations

We would like 64 processes

```
#SBATCH -ntasks 64
#SBATCH -cpus-per-task 1
#SBATCH -mem 32000
```

SLURM will find the space for 64 tasks on as few nodes as possible

We would like 16 processes each one needing 4 cores

```
#SBATCH -ntasks 16
#SBATCH -cpus-per-task 4
#SBATCH -mem 32000
```

SLURM will allocate 64 cores in total

Note: SLURM does not set `OMP_NUM_THREADS` for OpenMP!

srun and mpirun

Launching a MPI job

Now that we have a MPI code we need some way of correctly launching it across multiple nodes

- `srun` - SLURM's built in job launcher
- `mpirun` - "traditional" job launcher

To use this we type

```
srun mycode.x
```

With the directives on the previous slide this will launch 64 processes on 4 nodes

Multiple srun instances on one node

For code that doesn't scale...

```
#SBATCH -nodes 1
#SBATCH -ntasks 16
#SBATCH -cpus-per-task 1
#SBATCH -mem 32000

srun -mem=16000 -n 8 mytask1 &
srun -mem=16000 -n 8 mytask2 &
wait
```

Note: the `-multi-prog` *option for srun can provide a more elegant solution!*

For more details, check our documentation on this page:

https://scitasadm.epfl.ch/confluence/display/DOC/Running+multiple+tasks+on+one+node

Intel mpirun

Using IntelMPI and mpirun

On our clusters IntelMPI is configured to work with srun by default. If you want to use mpirun then do as follows:

- `unset I_MPI_PMI_LIBRARY`
- `export SLURM_CPU_BIND=none`
- `mpirun ./mycode.x`

We don't advise doing this and strongly recommend using srun! Please note that, behind the scenes, mpirun still uses SLURM.

CPU affinity

Kesako?

CPU affinity is the name for the mechanism by which a process is bound to a specific CPU (core) or a set of cores.

Pourquoi?

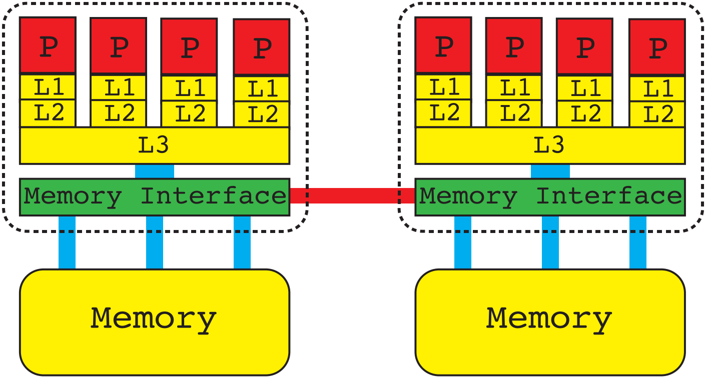
If a mask is not set the OS might place the task on different cores every 100ms or so. For performance this can be a very bad thing to do.

We can also optimise placement of ranks with respect to the underlying hardware.

ccNUMA

Cache Coherent Non Uniform Memory Architecture

This is what compute nodes with more than one processor look like...



CPU bitmasks

11000000

When talking about affinity we use the term "mask" or "bit mask" which is a convenient way of representing which cores are part of a CPU set.

If we have an 8 core system then the following masks mean:

- 10000000 - core 8
- 01000000 - core 7
- 00100000 - core 6
- 11110000 - cores 5 to 8
- 00001111 - cores 1 to 4

CPU bitmasks

11110000 is f0

These numbers can be conveniently written in hexadecimal so if we query the system regarding CPU masks we will see something like:

pid 8092's current affinity mask: 1c0
pid 8097's current affinity mask: 1c0000

In binary this would translate to

pid 8092's current affinity mask: 000000000000000111000000
pid 8097's current affinity mask: 000111000000000000000000

Binding with srun

Examples

```
srun -N 1 -n 4 -c 1 -cpu_bind=verbose,rank ./hi 1
cpu_bind=RANK - b370, task 0 :  mask 0x1 set
cpu_bind=RANK - b370, task 1 :  mask 0x2 set
cpu_bind=RANK - b370, task 2 :  mask 0x4 set
cpu_bind=RANK - b370, task 3 :  mask 0x8 set

srun -N 1 -n 4 -c 4 -cpu_bind=verbose,sockets ./hi 1
cpu_bind=MASK - b370, task 1 :  mask 0xff00 set
cpu_bind=MASK - b370, task 2 :  mask 0xff set
cpu_bind=MASK - b370, task 0 :  mask 0xff set
cpu_bind=MASK - b370, task 3 :  mask 0xff00 set
```

Common errors

Compiled on a different machine

```
Please verify that both the operating system and the
processor support Intel MOVBE, FMA, BMI, LZCNT and
AVX2 instructions.
```

LD_LIBRARY_PATH not correctly set

```
./run.x:  error while loading shared libraries:
libmkl_intel_lp64.so:  cannot open shared object file:
No such file or directory
```

Don't forget the srun

./mympicode.x instead of srun mympicode.x

```
Fatal error in MPI_Init:  Other MPI error, error
stack:
.MPIR_Init_thread(514):
.MPID_Init(320).......:  channel initialization failed
.MPID_Init(716).......:  PMI_Get_id returned 14
```

If things don't work

Try interactively

Errors are much more visible this way

- `salloc -N 2 -n 32 -t 01:00:00 -partition debug`
- `srun mycode.x < inp.in`

Check what's going on with htop and ps

- `ssh b123`
- `htop`
- `ps auxf`

If things still don't work

Crashes or won't start

- Reference input files
- GDB
- TotalView Debugger

Crashes after a while

Memory Leak?

- Valgrind
- MemoryScape (TotalView)

Some useful tricks

compilers

- icc -xAVX -axCORE-AVX2
- icc -mkl mycode.c
- mpiicc -show mycode.c

MKL link line advisor

https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

SCITAS documentation

http://scitas.epfl.ch/documentation/compiling-code-different-systems

Going further

SCITAS offers courses in

- MPI, an introduction to parallel programming
- MPI, advanced parallel programming
- Introduction to profiling and software optimisation
- Computing on GPUs

Exercise - Build Octopus

Download package

http://octopus-code.org

Hints

- load modules:

```
intel intel-mpi intel-mkl fftw gsl
```

- build first libxc
- some configure options to use for // octopus:

```
-enable-openmp -enable-mpi

-disable-zdotc-test

-with-blas="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_core -lmkl_intel_thread \

-lpthread -lm"

-with-fftw-prefix="${FFTW_ROOT}"
```