Intel and OpenMP Cheat Sheet

Erik Schnetter
(Dated: May 20-22, 2013, IHPC 2013, Iowa City)

Using the Intel compilers on Helium

By default, the Intel compilers are not easily accessible on Helium. To make them available, use the command

```
module load intel
```

after logging in to Helium. If you forget this, you will receive the error

```
ifort: command not found
```

when trying to compile.

By default, the Intel compilers use a medium level of optimisation. To create even more efficient code, compile your code with this command:

```
ifort -g -fast -o PROGRAM SOURCE.f90
```

where PROGRAM and SOURCE obviously need to be adapted.

Profiling: Finding out where the program spends its time

Compilers support *profiling*, which adds additional code to the executable that measures how much time is spent in which routine, and how often these routines are called. For the Intel compiler, use the option `-p` for this (other compilers use other options; e.g. the GNU compilers use `-pg`). That is, use this command:

```
ifort -g -fast -p -o PROGRAM SOURCE.f90
```

Then run the program to completion (do not interrupt it). Choose parameters such that the program runs for some time (tens of seconds), but not for too long (to save time).

A file `gmon.out` will then have been generated that contains the profiling information. This file is not human-readable. Use the command

```
gprof PROGRAM | less
```

to see the profiling result.

The top of the profiling result will show a flat profile, describing how much time is spent in each routine. (There may be some internal routines listed that are provided by the compiler and not written by you.)

The second, larger part of the profiling result describes which routine called by what other routine how many times. These numbers are more difficult to interpret, but luckily an explanation of this part is also appended.

You can use profiling to find out which routines are worthy of parallelisation.

Using OpenMP with the Intel compilers

By default, OpenMP directives are ignored by the compiler, and your program code will be treated as a serial program. To enable OpenMP, use the option `-openmp`. (Different compilers use different options; e.g. the GNU compilers use `-fopenmp` instead.) That is, use this command:

```
ifort -g -fast -openmp -o PROGRAM SOURCE.f90
```

When running an OpenMP-parallelised program, you should choose how many OpenMP threads to use. If you don't choose this explicitly, a system-dependent default will be used, probably trying to start one thread on each core of the system. This may or may not be a good idea. To choose the number of OpenMP threads, start your program like this:

```
env OMP_NUM_THREADS=12 ./PROGRAM
```

where you obviously need to adapt both the number 12 and the name of your program.

Do not run OpenMP programs on the head node; use `qlogin` to run them on a compute node. The compute nodes don't have emacs installed – keep another session open where you edit and compile.

To achieve maximum performance, you also need to ensure that the operating system does not move the OpenMP threads between different *NUMA nodes*. On Helium, each node contains two NUMA nodes, each with its own set of cores and main memory (6 cores and 12 GByte each). There is a performance penalty if a core accesses memory of another NUMA node. Thus, if all threads fit onto a single NUMA node (i.e. if you are using 6 or fewer OpenMP threads on Helium), you should explicitly *bind* the process to one of the NUMA nodes:

```
env OMP_NUM_THREADS=6 /Users/G1305_ESCHNE/hwloc/bin/hwloc-bind node:0 -- ./PROGRAM
```

where you obviously need to adapt both the number 6 and the name of your program.