

## Compilers

Available compilers, including GNU, INTEL and UPC compilers

There are several compilers for different programming languages available on the cluster:

- C/C++
- Fortran 77/90/95/HPF
- Unified Parallel C
- Java

The C/C++ and Fortran compilers are provided by:

Opensource:

- GNU GCC
- Clang/LLVM

Commercial licenses:

- Intel
- PGI

## Intel Compilers

For information about the usage of Intel Compilers and other Intel products, please read the Intel Parallel studio page.

## PGI Compilers

The Portland Group Cluster Development Kit (PGI CDK) is available.

```
$ module load PGI
$ pgcc -v
$ pgc++ -v
$ pgf77 -v
$ pgf90 -v
$ pgf95 -v
$ pghpf -v
```

The PGI CDK also includes tools for debugging and profiling.

PGDBG OpenMP/MPI debugger and PGPROF OpenMP/MPI profiler are available

```
$ module load PGI
$ module load Java
$ pgdbg &
```

```
$ pgprof &
```

For more information, see the PGI page.

## GNU

For compatibility reasons there are still available the original (old 4.4.7-11) versions of GNU compilers as part of the OS. These are accessible in the search path by default.

It is strongly recommended to use the up to date version which comes with the module GCC:

```
$ module load GCC
$ gcc -v
$ g++ -v
$ gfortran -v
```

With the module loaded two environment variables are predefined. One for maximum optimizations on the cluster's architecture, and the other for debugging purposes:

```
$ echo $OPTFLAGS
-O3 -march=native
```

```
$ echo $DEBUGFLAGS
-O0 -g
```

For more information about the possibilities of the compilers, please see the man pages.

## Unified Parallel C

UPC is supported by two compiler/runtime implementations:

- GNU - SMP/multi-threading support only
- Berkley - multi-node support as well as SMP/multi-threading support

## GNU UPC Compiler

To use the GNU UPC compiler and run the compiled binaries use the module `gupc`

```
$ module add gupc
$ gupc -v
$ g++ -v
```

Simple program to test the compiler

```
$ cat count.upc
```

```
/* hello.upc - a simple UPC example */
#include <upc.h>
#include <stdio.h>

int main() {
    if (MYTHREAD == 0) {
        printf("Welcome to GNU UPC!!!n");
    }
    upc_barrier;
    printf(" - Hello from thread %in", MYTHREAD);
    return 0;
}
```

To compile the example use

```
$ gupc -o count.upc.x count.upc
```

To run the example with 5 threads issue

```
$ ./count.upc.x -fupc-threads-5
```

For more informations see the man pages.

## Berkley UPC Compiler

To use the Berkley UPC compiler and runtime environment to run the binaries use the module bupc

```
$ module add BerkeleyUPC/2.16.2-gompi-2015b
$ upcc -version
```

As default UPC network the “smp” is used. This is very quick and easy way for testing/debugging, but limited to one node only.

For production runs, it is recommended to use the native Infiniband implementation of UPC network “ibv”. For testing/debugging using multiple nodes, the “mpi” UPC network is recommended. Please note, that the selection of the network is done at the compile time\*\* and not at runtime (as expected)!

Example UPC code:

```
$ cat hello.upc
```

```
/* hello.upc - a simple UPC example */
#include <upc.h>
#include <stdio.h>

int main() {
```

```

    if (MYTHREAD == 0) {
        printf("Welcome to Berkeley UPC!!!n");
    }
    upc_barrier;
    printf(" - Hello from thread %in", MYTHREAD);
    return 0;
}

```

To compile the example with the “ibv” UPC network use

```
$ upcc -network=ibv -o hello.upc.x hello.upc
```

To run the example with 5 threads issue

```
$ upcrun -n 5 ./hello.upc.x
```

To run the example on two compute nodes using all 48 cores, with 48 threads, issue

```

$ qsub -I -q qprod -A PROJECT_ID -l select=2:ncpus=24
$ module add bupc
$ upcrun -n 48 ./hello.upc.x

```

For more informations see the man pages.

## Java

For information how to use Java (runtime and/or compiler), please read the Java page.

### nVidia CUDA

For information how to work with nVidia CUDA, please read the nVidia CUDA page.