Copy of Compiling your own software
Dalmata (NYU Abu Dhabi)

Transferring data to/from the clusters
Transferring data to/from Prince cluster using Globus

Submitting jobs with sbatch

Available software
Building Software Tutorials
Programming for Biologists

Research Gallery
FAQs
NYU HPC Cheat Sheet
Wiki Map
HPC Policies
Compiling Code for Mercer

We recommend using the Intel compiler suite (module load intel), and, for MPI programs, OpenMPI (module load openmpi/intel).

The GNU compiler suite is also available (module load gcc) and over time other compiler versions and MPI implementations will be installed. There will be instances in which an alternative compiler or MPI library performs better or avoids a bug in the recommended compilers, but in the long run the functionality and performance of compilers and MPI libraries is very similar, so for easiest maintenance, only use the alternatives if you really need to.

Mercer is a heterogeneous system comprised of Ivy Bridge, Sandy Bridge and Westmere nodes, as well as some older Nehalem nodes. For the best performance, you should compile your code optimizing for Ivy Bridge, with AVX vector instructions for faster floating point computations. However, Westmere CPUs cannot run such code. The Intel compiler can compile for two architectures in the same object files and executables, and we recommend that you use this feature. To do so, use the following flags to icc, icpc and ifort:

```
-xSSE4.2 -axAVX
```

This will compile targeting Westmere (slightly older CPU cores) as a default, with an alternate path optimized for the newer Ivy Bridge cores of Mercer if they are available when the executable is run.

If you need GNU compilers, we recommend disabling AVX instructions to ensure your executable can run on any compute nodes. To do this add the flag:

```
-mno-avx
```

Both SSE and AVX vectorize loops, meaning that several iterations of the loop are executed concurrently. This changes the order of operations, which can change the results. Programs using summations over loops are especially vulnerable to this.

Many algorithms and problems are numerically sensitive, and vectorization or certain aggressive optimizations can be enough to trigger numerical instability. Debugging numerical instability is complex - Intel has some interesting information about it here) but if you suspect your model is experiencing it you should disable vectorization, use -fp-model precise or -fp-model strict and reduce optimization flags from -O3 to -O2 or -O0, and use this as a baseline to test higher levels of optimization.

If you are building software to use for many or long-running jobs, testing for performance with a few MPI libraries and compile options can be very valuable since a specific model might run noticeably faster under one MPI implementation than another.