## Compiling your own software at Prince

<table>
<thead>
<tr>
<th>Quick Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC Home</td>
</tr>
<tr>
<td>Getting an account</td>
</tr>
<tr>
<td>Getting started on Prince</td>
</tr>
<tr>
<td>Prince How-to Articles</td>
</tr>
<tr>
<td>Logging in Windows</td>
</tr>
<tr>
<td>Mac / Linux</td>
</tr>
<tr>
<td>Clusters and Storage Prince (HPC)</td>
</tr>
<tr>
<td>Dumbo (Hadoop)</td>
</tr>
<tr>
<td>Brooklyn (OpenStack)</td>
</tr>
<tr>
<td>Dalmata (NYU Abu Dhabi)</td>
</tr>
</tbody>
</table>
Transferring data to/from the clusters

Transferring data to/from Prince cluster using Globus

Submitting jobs with sbatch

Available software

Licensed Software Available on the HPC Cluster

Building Software

Slurm Tutorial

Tutorials

FAQs

Scratch Area Cleanup

Programming for Biologists

Acknowledgment Statement

Research Gallery

HPC People
Running jobs on the Prince Cluster

Accessing the Prince Cluster

From Windows workstation

From Mac workstation

Software and Environment Module

Job script and resource request

Introduction to job scheduling

Submitting jobs with sbatch

Requesting resources

Using computing nodes interactively

Monitoring batch jobs

Monitoring batch jobs - squeue
Compiling Code for Prince

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

The GNU compiler suite is also available (module spider 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.

For the best performance, you should compile your code optimized with AVX2 vector instructions for faster floating point computations, also to generate multi-threaded code based on the OpenMP* directives. To do so, use the following flags to icc, icpc and ifort:

```
-O3 -fPIC -unroll -ip -axCORE-AVX2 -qopenmp -qopt-report-stdout
-qopt-report-phase=openmp
```

For the link step, you can add a flag “-shared-intel”.

If you need GNU compilers, we recommend enabling AVX and disabling AVX2 instructions to ensure your executable can run on any compute nodes. To do this add the flag:

```
-mavx -mno-avx2
```

AVX2 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.