Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing

Compiling and running code on CIMS machines

We have different versions of GNU and Intel compilers installed on CIMS machines. To compile a serial code such as those in the `lecture1` repository, you can use

```
g++ -std=c++11 01-compute.cpp -o compute
./compute
```

If you prefer using an Intel compiler, can use using Intel's `icpc` compiler:

```
icpc -std=c++11 01-compute.cpp -o compute
```

To get faster code, you can use optimization flags, e.g.:

```
icpc -std=c++11 -O2 01-compute.cpp -o compute
```

Alternatives to `-O2` are `-O1` (less aggressive optimization) or `-O3` (more aggressive optimization). You can get a newer Intel compiler by loading the corresponding module:

```
icpc --version
module load intel-2018
icpc --version
```

Compiling distributed memory code requires that MPI (Message Passing Interface) is available, this can be done using the following command:

```
module purge
module load mpi/openmpi-x86_64 gcc-8.1
```

Then, MPI code can be compiled as:

```
mpic++ 06-mpi-simple.cpp -o mpi-simple
```

Then you can run the program on, say, 3 processor cores in parallel:

```
mpirun -np 3 ./mpi-simple
```

The output should look something like that:

```
I am process 1 of 3
I am process 2 of 3
I am process 0 of 3
```

You can ask for more cores than your computer has, and MPI will run several processes on a single physical compute core.
Finally, lets try to run the shared memory version of our program. The newer versions of the GNU compilers (as well as Intel compilers) support OpenMP, i.e., the framework that allows running several processes with a shared memory. You have to explicity link against the OpenMP libraries:

```
g++ -std=c++11 -fopenmp 04-omp-inner.cpp -o omp-inner
```

To run the shared memory version of our program, you can use:

```
./omp-inner -n 100000000 -repeat 10
```

Here, the system will decide how many threads to use. On my desktop, it uses 8 threads. You can also choose how many threads you want the system to use for your shared memory parallel computation:

```
OMP_NUM_THREADS=16 ./omp-inner -n 100000000 -repeat 10
```

Running MPI across several CIMS machines

To run accross several CIMS machines, possibly including our compute servers in the basement[1], you first need to first ensure password-free ssh access, for instance following the steps described here[2]. Since your home directly is the same on all CIMS machines, this just amounts to copying your public key into the authorized_keys file (which is in the same directory). I usually just do a manual copy-and-paste, adding to the end of that file. You also need to add the line "module load mpi/openmpi-x86_64" to your " /.bashrc" file so that MPI is loaded by default. Then, the commands

```
mpic++ -std=c++11 -O3 -fopenmp 07-mpi-inner.cpp -o mpi-inner
mpirun -np 20 --oversubscribe -H crunchy3,crunchy5 ./mpi-inner -n 100000000 -repeat 10
```

will compile and run the mpi-inner program on overall 20 cores of `crunchy3` and `crunchy5`. You can also create a file that contains the names of all the hostnames you want to be using, and pass this file to `mpirun` via `-f FILENAME` instead of listing the hosts on the command line.

[1]http://cims.nyu.edu/webapps/content/systems/resources/computeservers

[2]https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2