- **Compute the theoretical peak FLOPs for the processor type available on coc-ice. Consider both a "CPU-only" configuration as well as a "GPU-only" configuration. (10 points)**

performance in GFlops = (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node) found here.

CPU-only config -

On coc-ice, the max number of cores per node is 24.

Because we know that coc-ice has up to 48 CPU per job, and the max number of cores per node is 24, we can assume there are 2 nodes allocated.

2.7 GHz is the max CPU speed in GHz for any given CPU on coc-ice, which equates to 2.7 billion cycles per second.

In the research performed, we were unable to find any number of IPC above 6 for the benchmark, so we will take 6 to find the maximum flops.

performance in GFlops = (2.7) x (2) x (6) x (24) = 777.6 Gigaflops

GPU-only config -

"Arm Blog - Flipping the FLOPS - how ARM measures GPU compute performance": We multiply the number of FLOPS per cycle by the number of arithmetic pipelines per core, then the number of cores, then by the frequency. That gives you a number of FLOPS.

We found that the Quadro Pro RTX6000 was the fastest GPU available on coc-ice with 1770 MHz being the speed. Number of arithmetic pipelines per core and number of cores will again multiply to be at most 48. Again, we will assume 6 is the max frequency/flops per cycle.

performance in GFlops = (1.77) x (48) x (6) = 509.76 Gigaflops

- **Build and install a CPU-only version of the HPL benchmark using your choice of linear algebra library and MPI library. Which linear algebra library did you choose and why? Which MPI library did you choose and why? (20 points)**

One such library that is often used for HPC and specifically for the HPL benchmark is the Basic Linear Algebra Subroutines (BLAS) library. We chose it because BLAS provides a set of low-level routines for performing basic linear algebra operations such as matrix-vector multiplications, vector-vector operations, and matrix-matrix multiplications. Many high-performance libraries, such as LAPACK, build on top of BLAS to provide higher-level linear algebra functions.

MPI was already installed, so we didn't need to worry about that.

- **Run the HPL benchmark on a single node using a fixed problem size (N) and by varying the number of cores from 2 to 20, doubling the cores at each trial. Which parameters did you need to change? Plot the GFLOPs number for each run vs. no. of cores. (30 points)**

See below for the command we ran for 2 nodes, and we will only change that parameter in subsequent trials.

```
[blaenger3@login-coc-ice-1 ~]$
[blaenger3@login-coc-ice-1 ~]$
[blaenger3@login-coc-ice-1 ~]$ qsub -I -q coc-ice -l nodes=2:ppn=2,pmem=8gb,walltime=2:00:00
qsub: waiting for job 481906.sched-coc-ice.pace.gatech.edu to start
qsub: job 481906.sched-coc-ice.pace.gatech.edu ready

----------------------------------------
Begin PBS Prologue Wed Feb  8 23:51:25 EST 2023
Job ID:     481906.sched-coc-ice.pace.gatech.edu
User ID:    blaenger3
Job name:   STDIN
Queue:      coc-ice
End PBS Prologue Wed Feb  8 23:51:25 EST 2023
----------------------------------------
[blaenger3@atl1-1-02-003-19-l ~]$
```

We started at 2 cores, and doubled each time to give us this table:

Number of Cores	2	5	10	20
Gflops	1.1512e-01	1.3271e-01	1.2971e-01	1.2830e-01

Each time when setting up the new job, we changed the number of nodes as a parameter. We kept all other parameters constant.

Then we did cd hpl, cd bin, cd Linux_Intel64. Then we did the command mpirun -np 4 ./xhpl. We were unsure what we should use as the result of the test, so we just took the last Gflops displayed before the test says Finished. See screenshot below.

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2R4          35     4     4     1               0.00             1.2971e-01
HPL_pdgesv() start time Thu Feb  9 00:01:53 2023

HPL_pdgesv() end time   Thu Feb  9 00:01:53 2023

--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV-
Max aggregated wall time rfact . . . :               0.00
+ Max aggregated wall time pfact . . :               0.00
+ Max aggregated wall time mxswp . . :               0.00
Max aggregated wall time update  . . :               0.00
+ Max aggregated wall time laswp . . :               0.00
Max aggregated wall time up tr sv  . :               0.00
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   1.57008646e-02 ...... PASSED
================================================================================

Finished    864 tests with the following results:
            864 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
[blaenger3@atl1-1-02-003-19-l Linux_Intel64]$
```

- **Run the HPL benchmark using all 20 cores [@Richie we thought there were up to 24 cores] and tune the HPL.dat file to achieve the best GFLOPs number. Which parameters did you tune? What were your results using the unoptimized parameter(s) vs. the optimized parameter(s)? (40 points)**

The results using the unoptimized parameters were the same as from the table above, 1.2830e-01 Gflops. With the optimized parameters we got as nearly 3 times the Gflops as shown below:

```
Column=000000004 Fraction= 0.0% Gflops=5.983e+02
```

We used [this website] to tune these parameters in the HPL.dat file:

N, NBs, # of process grids, Ps, Qs, # of panel fact, # of recursive stopping criterion, # of recursive panel fact, DEPTHs

Now we are doing vim hpl.dat and we copy and pasted the new .dat file [see below for modified .dat]

We ran the same mpi command we have been running but ran it with np - 40 because np is = p x q.

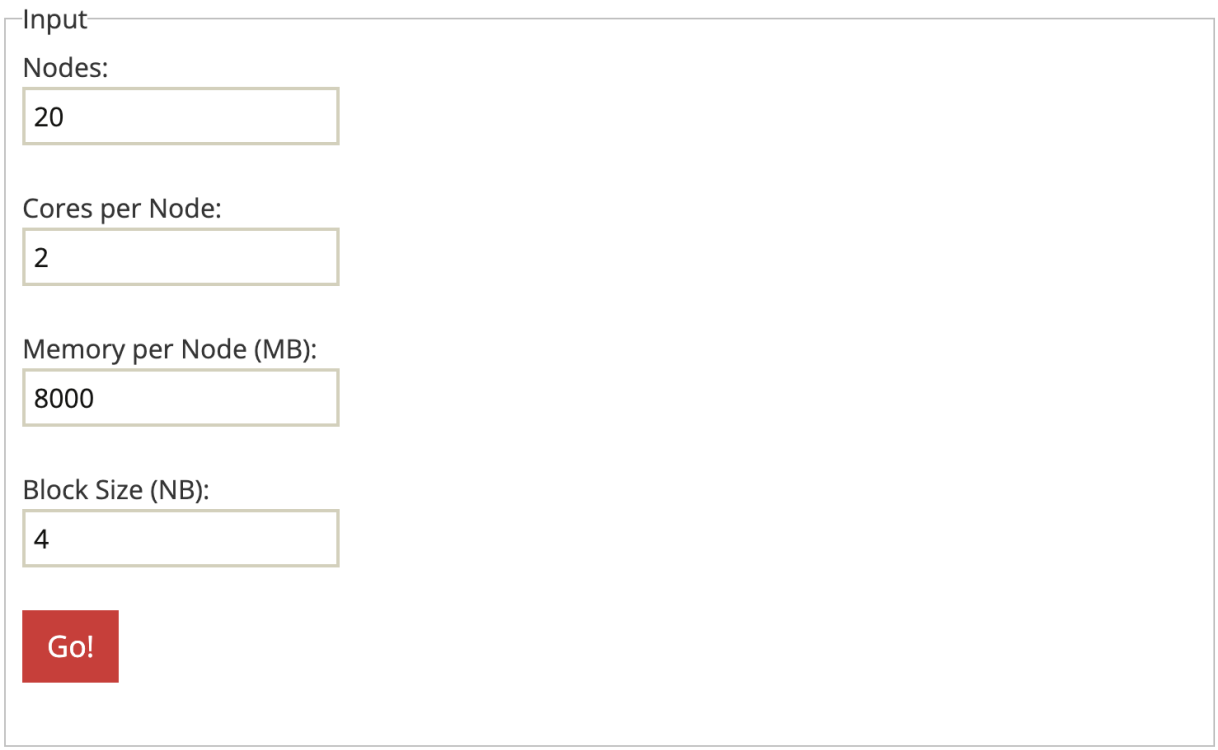
Modified.dat file:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
129520         Ns
1            # of NBs
4           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
5            Ps
8            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Command ran:

```
[blaenger3@login-coc-ice-2 ~]$ qsub -I -q coc-ice -l nodes=20:ppn=2,pmem=8gb,walltime=2:00:00
```

Website input:



- **Run HPL for 2, 4, and 8 nodes. Was the GFLOPs number exactly twice that of your single-node performance? Why or why not? (10 points)**

```
[blaenger3@atl1-1-01-002-5 Linux_Intel64]$ qsub -I -q coc-ice -l nodes=1:ppn=2,pmem=8gb,walltime=2:00:00
```

```
[blaenger3@atl1-1-01-002-5 Linux_Intel64]$ qsub -I -q coc-ice -l nodes=2:ppn=2,pmem=8gb,walltime=2:00:00
```

```
[blaenger3@atl1-1-01-002-5 Linux_Intel64]$ qsub -I -q coc-ice -l nodes=4:ppn=2,pmem=8gb,walltime=2:00:00
```

```
[blaenger3@atl1-1-01-002-5 Linux_Intel64]$ qsub -I -q coc-ice -l nodes=8:ppn=2,pmem=8gb,walltime=2:00:00
```

No it was not. This was most likely due to the fact that the performance of a system depends on various factors including the architecture of the processors, the memory subsystem, the interconnects between the cores, and the software stack.

When you double the number of cores, you also increase the demand on the memory system, as each core needs to access memory to perform its computations. If the memory system can't keep up with the increased demand, it becomes a bottleneck, limiting the overall performance of the system.

- **(Bonus) Run HPL using GPUs, as many as you can. What fraction of peak GPU speed are you able to achieve? Can you explain why? (10 points)**

```
[blaenger3@login-coc-ice-2 Linux_Intel64]$
[blaenger3@login-coc-ice-2 Linux_Intel64]$ qsub -I -q coc-ice-gpu -l nodes=1:ppn=6,gpus=4:teslav100,walltime=2:00:00
qsub: waiting for job 481931.sched-coc-ice.pace.gatech.edu to start
qsub: job 481931.sched-coc-ice.pace.gatech.edu ready

----------------------------------------
Begin PBS Prologue Thu Feb  9 01:04:53 EST 2023
Job ID:     481931.sched-coc-ice.pace.gatech.edu
User ID:    blaenger3
Job name:   STDIN
Queue:      coc-ice-gpu
End PBS Prologue Thu Feb  9 01:04:53 EST 2023
----------------------------------------
[blaenger3@atl1-1-01-002-5 ~]$
```

For this portion we modified our request to request something from coc-ice-gpu. We requested Teslav100s which we think is the fastest GPU available to us.

We then ran the same mpi command we have been running this whole time for the benchmark. We were able to achieve 0.179 GFlops. (see photo below)
From part 1 we calculated the max GFlops to be 509.76 Gigaflops. So we achieved 0.179/509.76 = .035114563% of the max GPU speed. We think that we may have messed up something in the .dat file or with the -np parameter, but we are not 100% sure. Or we messed up something in the peak GPU speed. We did try though!

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2R4          35     4     1     4               0.00             1.7960e-01
HPL_pdgesv() start time Thu Feb  9 01:05:13 2023
```

Deliverables:

- You should also include a description of your cluster and how you created the cluster.

Here is a screenshot of one of the many jobs we ran:

```
[blaenger3@atl1-1-02-003-19-l hpl]$
[blaenger3@atl1-1-02-003-19-l hpl]$ qstat -u blaenger3

sched-coc-ice.pace.gatech.edu:
                                                                  Req'd       Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory      Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ --------- --------- - ---------
481890.sched-coc-ice.p  blaenger3   coc-ice  STDIN             71838     8     48       4gb  02:00:00 R  01:47:23
[blaenger3@atl1-1-02-003-19-l hpl]$
[blaenger3@atl1-1-02-003-19-l hpl]$
[blaenger3@atl1-1-02-003-19-l hpl]$
```