Computational Chemistry
USC CARC HPC

NAMD

LAMMPS

GROMACS

QChem

Quantum Espresso

CP2K

Outline

- Part 1: Run atomistic simulations efficiently on HPC
  - a good SLURM job script goes a long way….

- Part 2: Build recipes
  - Makefile
  - CMake

NAMD: Charm++ Runtime

- The Shared-Memory and Network-Based Parallelism (SMP) version of NAMD-CHARM++ parallelization parameters:
  - worker threads = +pN
  - processors per worker thread = ++ppn N
  - MPI Ranks = (# of worker threads)/(# of processors per worker thread)
- In addition: each MPI rank spawns a communication thread (CHARM++ feature)
- So in analysis of the following run line:
  - charmrun +p16 namd2 apoa1.namd ++ppn 2 +setcpuaffinity
- MPI Ranks = (+p16)/(++ppn 2) = 8
- Each MPI Rank spawns 2 worker threads and 1 communication thread:
- total # of cpus used per node = 8 (MPI Ranks) x 3 (2 worker threads + 1 communication thread) = 24 cpus

NAMD: Charm++ Runtime

- The way this translates into SLURM directives is the following:

- **For 1 node**:
  #SBATCH --ntasks=8
  #SBATCH --cpus-per-task=3
  #SBATCH --nodes=1

  charmrun +p16 namd2 apoa1.namd ++ppn 2 +setcpuaffinity

- **For 4 nodes**:
  #SBATCH --ntasks=32
  #SBATCH --cpus-per-task=3
  #SBATCH --nodes=4

  charmrun +p64 namd2 apoa1.namd ++ppn 2 +setcpuaffinity

LAMMPS: Building Packages

- Available as a module:
  $ module load lammps

- Parallelization schemes: MPI + OpenMP + GPU

- Provided: Makefile.mpi
  - modules used: MPI (openmpi; intel-mpi); FFTW3

- command-line:
  $ module purge
  $ module load usc
  $ module load fftw/3.3.8-dp
  $ cp Makefile.mpi lammps-29Oct20/src
  $ cd lammps-29Oct20/src
  $ make mpi

- USER-PACKAGE:
  $ cd lammps-29Oct20/src
  $ make yes-molecule
  $ make mpi
  $ make ps

GROMACS: CPU-to-GPU Ratio

- Available as a module:
  $ module load gromacs

- Parallelization schemes: MPI + OpenMP + GPU

- CPU to GPU ratio recommended to be less than 3.5
  - cpu to/from gpu communication (e.g. data exchange)

- SLURM directives:
  #SBATCH --nodes=1
  #SBATCH --ntasks=6
  #SBATCH --cpus-per-task=2
  #SBATCH --gres=gpu:p100:2

  export OMP_NUM_THREADS=2

QChem: OpenMP Parallelization

- QChem vs Gaussian
  - Gaussian: USC CARC HPC does not have an active license
  - QChem: available to USC researchers associated with the Chemistry Department
- Available as a module:

  $ module load qchem-openmp
- Parallelization schemes: OpenMP
  - intra-node parallelization vs inter-node parallelization
  - export OMP_NUM_THREADS=<num> → --cpus-per-task=<num>
- SLURM directives:

```
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4

export OMP_NUM_THREADS=4
```

Quantum Espresso: Alternative to VASP

- Available as a module:

  $ module load quantum-espresso
- Parallelization schemes: MPI + OpenMP + GPU
- Pseudopotentials: PAW, Ultrasoft, and Norm-conserving (GTH)
- VASP requires four input files:

  INCAR: simulation settings

  POSCAR: structure

  KPOINTS: k-point mesh

  POTCAR: pseudopotentials
- Quantum Espresso:

  one single input file: simulation settings; structure; k-point mesh; pseudopotentials
- Provided: example input file for Quantum Espresso

CP2K: Best of Both Worlds

- Hybrid Gaussian and PlaneWaves (GPW) Method:
  - Molecular Quantum Chemistry
  - Solid-State Chemistry (Periodic Boundary Conditions)
  - GAPW all-electron method: core-spectroscopy simulations (e.g. XAS)
- Classical Molecular Dynamics
- Ab Initio methods: semi-empirical, DFTB, DFT (LSD, GGA, hybrid-GGA, meta-GGA), RPA, MP2, CC
- Available as a module:

  $ module load cp2k
- Parallelization schemes: MPI + OpenMP + GPU
- Look for out weekly CP2K Workshop Series
  - every semester

Multi-Node Jobs: --ntasks-per-node

- MPI Jobs: #SBATCH --exclusive
  - disables node-sharing
  - ideal for MPI codes that scale well (large core counts)
  - increases job queue wait time
- $ sinfo2
  - STATE: mix vs idle vs alloc
- Take advantage of available compute nodes:
  - STATE = mix (usually the majority of compute nodes)
  - use the --ntasks-per-node=<num> slurm directive
- For MPI jobs:
  - strongly recommended to use the same number of tasks (MPI ranks) per node
- Recommended (from extensive testing of massively parallelized codes):
  - #SBATCH --ntasks-per-node=8

Optimize Resource Usage: Scaling Tests

- Main Idea: measure the time-to-solution as a function of increasing parallelism
- OpenMP Jobs:
  - run a series of calculations: export OMP_NUM_THREADS=1, 2, 4, 8, 12, 16, 20, 24, max-cpus-per-node
- MPI Jobs:
  - run a series of calculations: (# of MPI tasks) vs (# of compute nodes)
  - recommendation: # of compute nodes
  - example: #SBATCH --nodes=1, 2, 4, 8, 12, 16, 20
- Hybrid MPI+OpenMP:
  - recommendation from extensive benchmark tests: sweet spot = export OMP_NUM_THREADS=(2 to 6)
  - greater speed-up in time-to-solution is obtained by using more MPI tasks
  - OpenMP threads: great way to increase memory-per-cpu while keeping all cores active
- What is the best calculation type for benchmarking:
  - single point energy calculation

Parallel Jobs: Considerations

- SLURM environment variables:
  - $SLURM_NTASKS = #SBATCH --ntasks=<num>
  - $SLURM_CPUS_PER_TASK = #SBATCH --cpus-per-task=<num>
  - $SLURM_NTASKS_PER_NODE = #SBATCH --ntasks-per-node=<num>
- OpenMP threads:
  - SLURM: --cpus-per-task=<# of OpenMP threads>
  - SLURM: export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
- Hybrid MPI+OpenMP:
  - per node accounting: (--ntasks-per-node) * (--cpus-per-task) <= (max # of cpus per node)
- Process Management Interface (PMI):
  - openmpi: srun --mpi=pmix_v2
  - mpich, mvapich2, intel-mpi: srun –mpi=pmi2
- Task Affinity: binding of processes
  - srun --cpu-bind=cores

SLURM Job Script: OpenMP

```
#!/bin/bash
#SBATCH --mail-user=username@usc.edu
#SBATCH --mail-type=all
#SBATCH --account=<account_id>
#SBATCH --job-name=run_openmp
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=3GB
#SBATCH --time=24:00:00
#SBATCH --partition=main

module purge
module load usc

export OMP_NUM_THREADS=8

cd ${SLURM_SUBMIT_DIR}
srun ./openmp_code.x > output.log
```

SLURM Job Script: MPI+OpenMP

```
#!/bin/bash
#SBATCH --mail-user=username@usc.edu
#SBATCH --mail-type=all
#SBATCH --account=<account_id>
#SBATCH --job-name=run_mpi_openmp
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=12
#SBATCH --ntasks=48
#SBATCH --cpus-per-task=2
#SBATCH --mem=0
#SBATCH --time=24:00:00
#SBATCH --partition=main
#SBATCH --exclusive
#SBATCH --constraint=xeon-4116

module purge
module load usc

export OMP_NUM_THREADS=2

cd ${SLURM_SUBMIT_DIR}
srun --cpu-bind=cores --mpi=pmix_v2 -n $SLURM_NTASKS ./mpi_openmp_code.x > output.log
```

SLURM Job Script: MPI+OpenMP+GPU

```
#!/bin/bash
#SBATCH --mail-user=username@usc.edu
#SBATCH --mail-type=all
#SBATCH --account=<account_id>
#SBATCH --job-name=run_gpu
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=6
#SBATCH --ntasks=18
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=3GB
#SBATCH --time=24:00:00
#SBATCH --partition=main
#SBATCH --gres=gpu:v100:2
#SBATCH --gres-flags=enforce-binding

module purge
module load usc
module load cuda/10.1.243 cudnn/8.0.2-10.1

export OMP_NUM_THREADS=2
nvidia-smi -c DEFAULT

cd ${SLURM_SUBMIT_DIR}
srun --cpu-bind=cores -n $SLURM_NTASKS mpi_openmp_gpu_code.x > output.log
```

Building Applications: Considerations

- login-node vs compute-node
  - compile on compute node via salloc
- Compiler optimizations are very important
- Safest method:
  - compile on oldest compute node hardware
- Most rigorous method:
  - compile code for each different architecture
  - use ‘#SBATCH --constraint=xeon-4116’