Parallel Computing on Agate

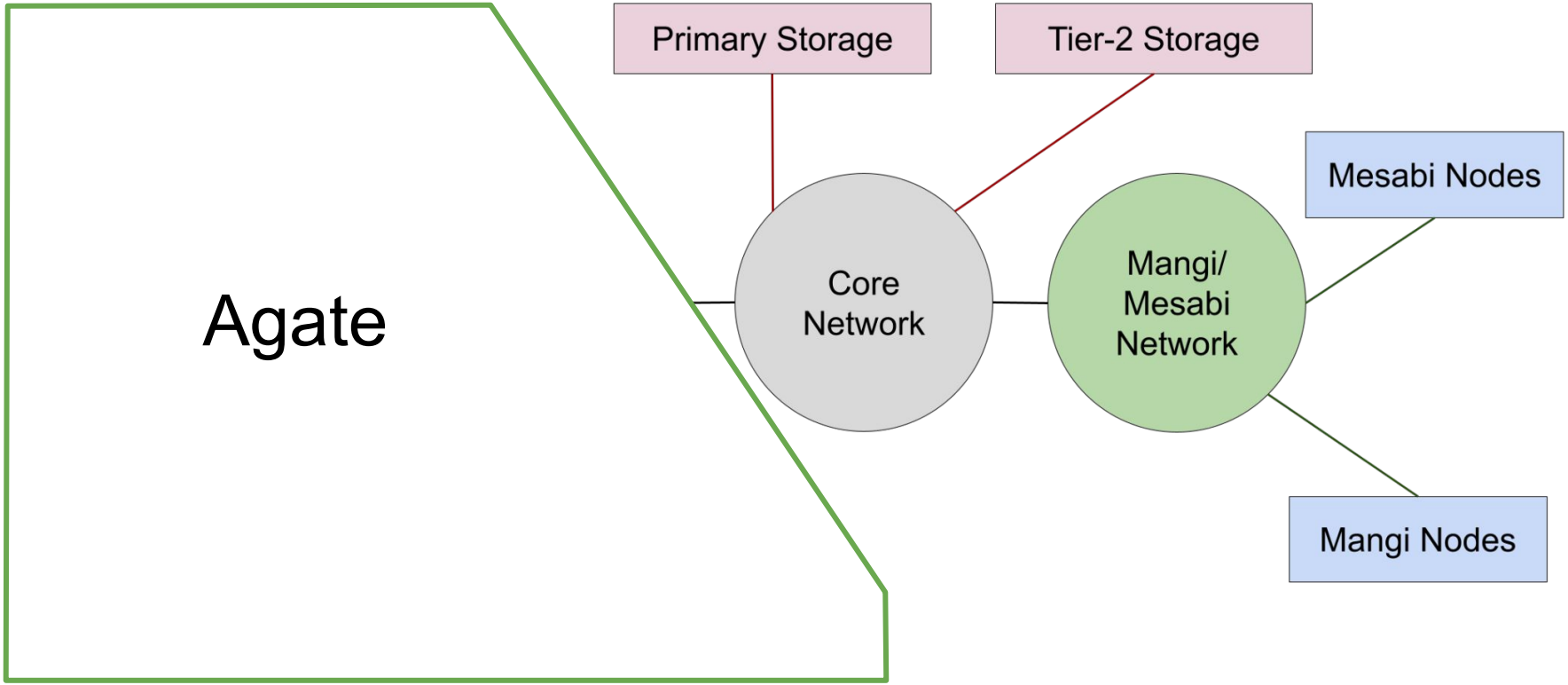
David Porter, Benjamin Lynch

Overview

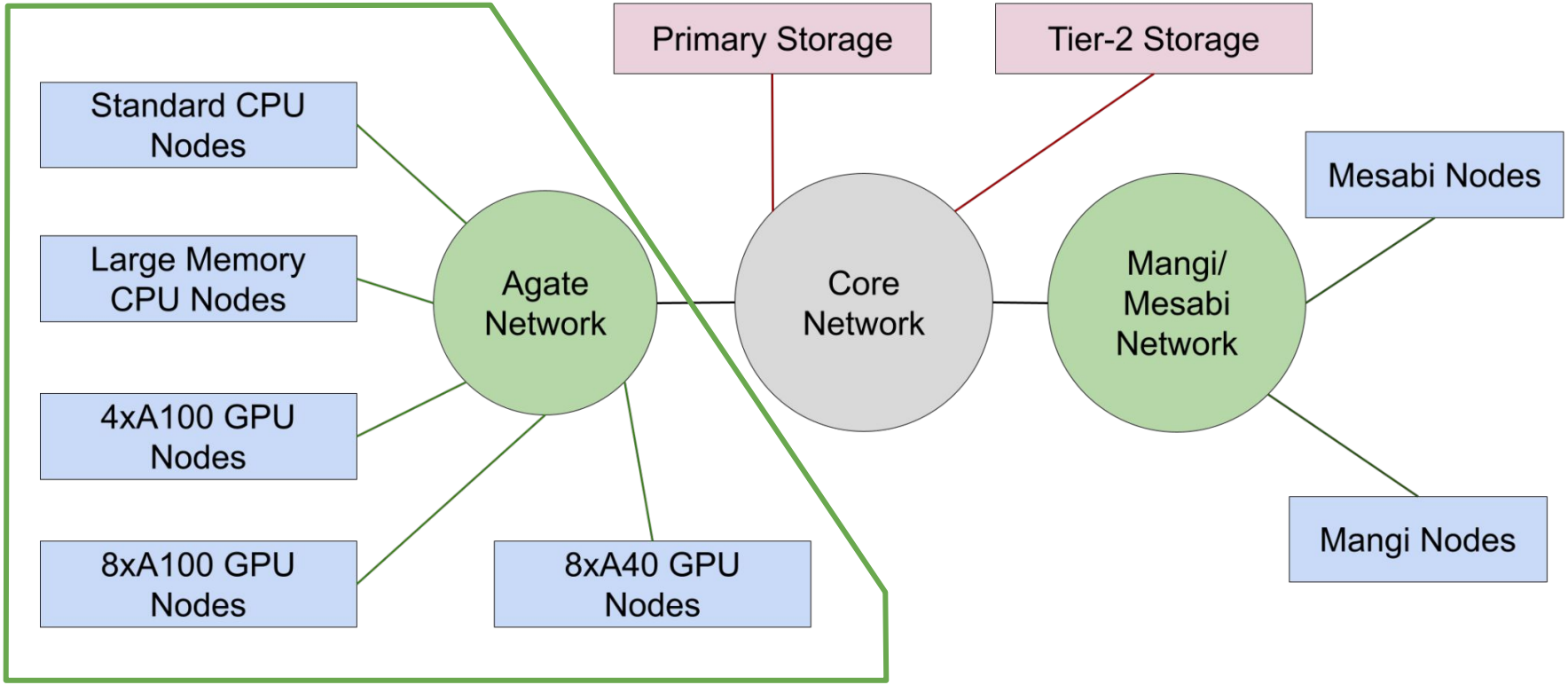
- Introduction to Agate
- GPU Jobs
- Job Arrays
- Thread Parallel
- MPI Parallel

https://public.s3.msi.umn.edu/tutorials/march_31_2022.tar.gz

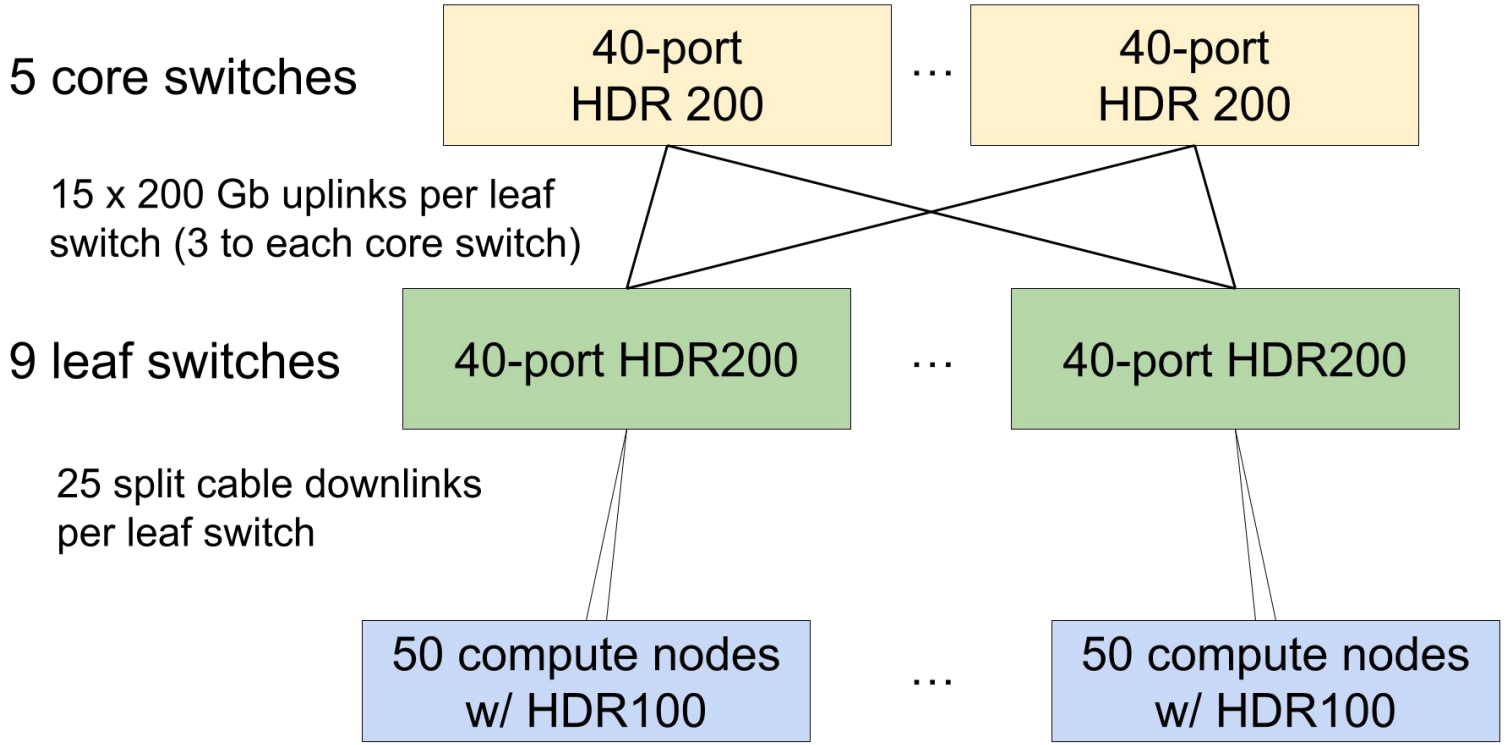
HPC Resources



HPC Resources

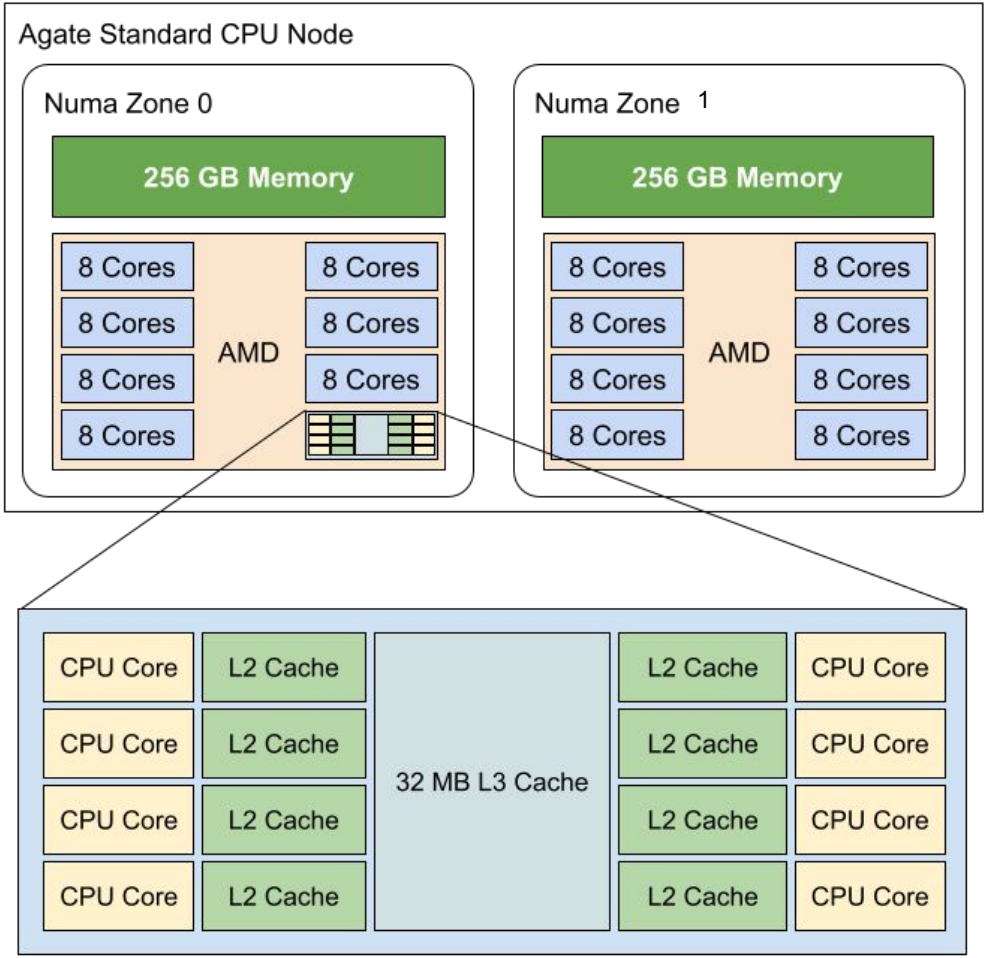


Agate Infiniband Topology



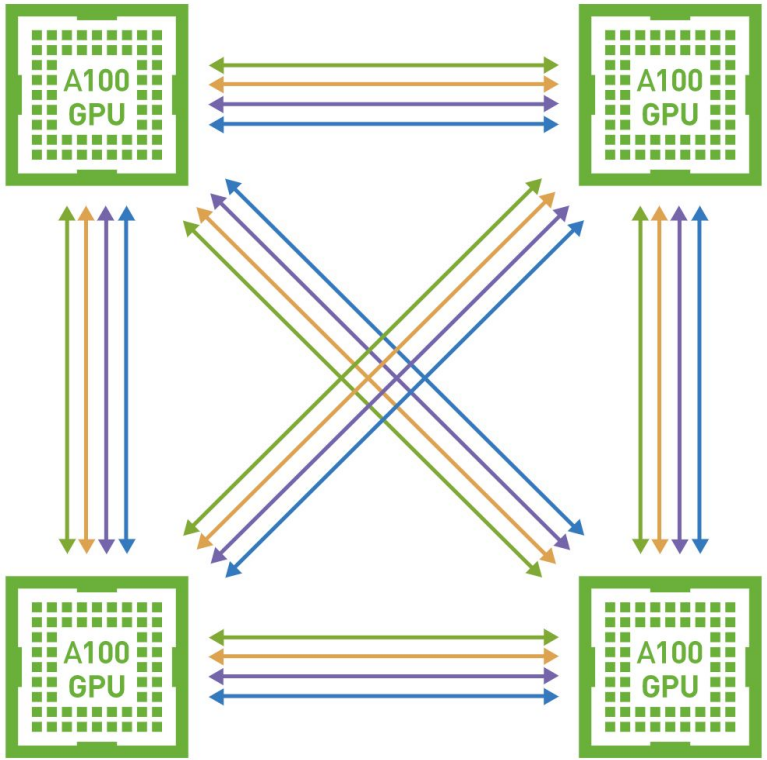
Standard Agate CPU node

- 128 CPU cores
- 512 GB of memory
- 850 GB local SSD

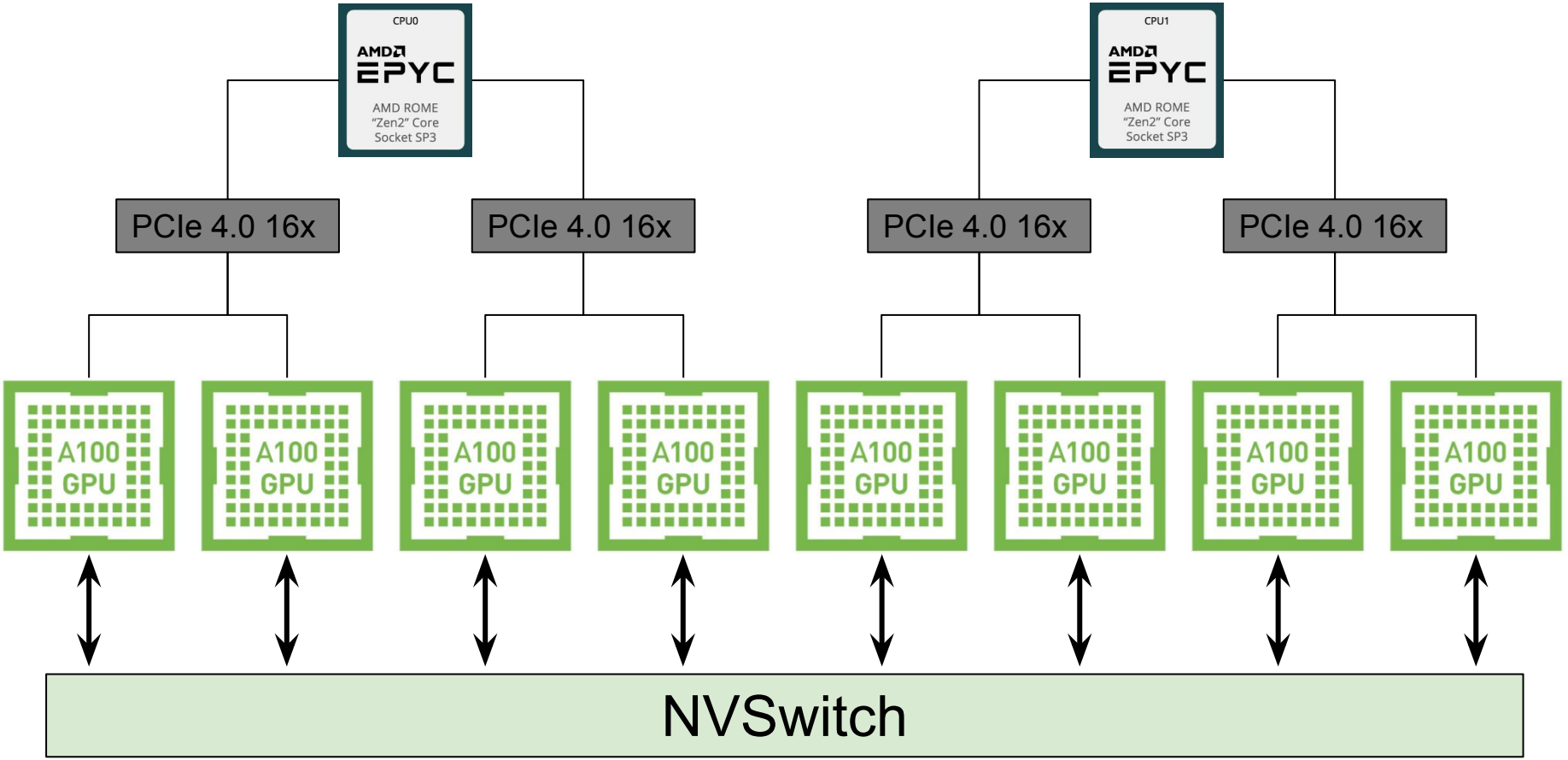


A100-4

- 512 GB memory
- 64 CPU Cores
- 4 A100 GPUs
  - 3rd Generation NVLink
  - 600 GB/s bandwidth per GPU



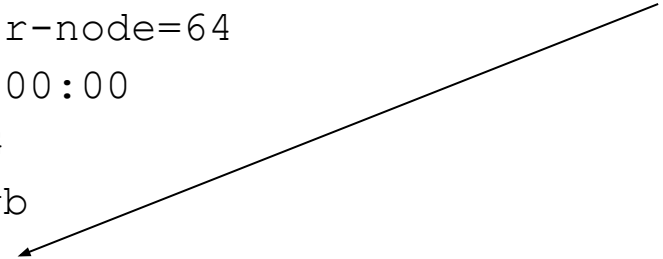
A100-8



Slurm Script

```
#!/bin/bash -l
#
#SBATCH --mail-type=NONE
#SBATCH -N 1
#SBATCH --tasks-per-node=64
#SBATCH --time=24:00:00
#SBATCH --mem=400G
#SBATCH --tmp 400gb
#SBATCH -p a100-4
#SBATCH --gres=gpu:a100:4

do_something
```

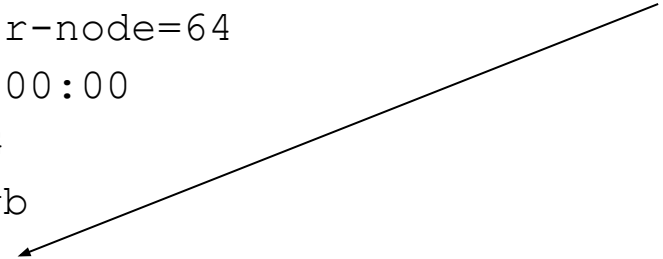


Specify the 4-way GPU nodes if you want to ensure your job lands on this node type.

Slurm Script

```
#!/bin/bash -l
#
#SBATCH --mail-type=NONE
#SBATCH -N 1
#SBATCH --tasks-per-node=64
#SBATCH --time=24:00:00
#SBATCH --mem=400G
#SBATCH --tmp 400gb
#SBATCH -p msigpu
#SBATCH --gres=gpu:a100:4

do_something
```



Specify the **msigpu** partition if you don't care of the job lands on a **a100-4** or an **a100-8** node.

GPU Jobs on Agate

A Few Notes

- Always use CUDA 11+ libraries
- `nvidia-smi` gives you useful information on the GPUs and the applications running on them
- Many ways to parallelize you job!
  - NCCL
  - openacc
  - cuda
  - Combine methods above with OpenMP & MPI

OpenACC

Simple

- Similar to OpenMP, OpenACC directives are a simple means of parallelizing your code.
- As a programmer, you do not need to explicitly copy data to the GPU device.

Portable

- OpenACC code is portable across GPUs. Code can be compiled and run on CPU-only systems as well as computers with NVIDIA, AMD or Intel GPUs.

OpenACC

```
#pragma acc kernels
{
    #pragma acc loop independent
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {

            Anew[j*m+i] = 0.25f * ( A[j*m+i+1] + A[j*m+i-1]
                                  + A[(j-1)*m+i] + A[(j+1)*m+i]);

            error = fmaxf( error, fabsf(Anew[j*m+i]-A[j*m+i]));
        }
    }
```

Task Parallel: Job Arrays

- Basic SLURM Job Script for an array of similar tasks
- Submit job array to queue
- Example: A data driven workflow

Simple Task Array

SLURM script

- 1 task (core) on 1 node
- Up to 3800 MB per task
- Up to 10 min.

- Submit task array
- Output
  - logs/task_ouput.######_#

do_one_task.sh

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem-per-cpu=3800
#SBATCH -t 00:10:00
#SBATCH -p agsmall
#SBATCH -o logs/task_output.%A_%a

echo "Task " ${SLURM_ARRAY_TASK_ID} "  on " $(hostname)
```

```
ahl03:task_arrays % sbatch --array=0-9 do_one_task.sh
ahl03:task_arrays % grep Task logs/*
logs/task_output.67134323_0:Task  0  on  acn06
logs/task_output.67134323_1:Task  1  on  acn06
logs/task_output.67134323_2:Task  2  on  acn06
logs/task_output.67134323_3:Task  3  on  acn06
logs/task_output.67134323_4:Task  4  on  acn06
logs/task_output.67134323_5:Task  5  on  acn06
logs/task_output.67134323_6:Task  6  on  acn06
logs/task_output.67134323_7:Task  7  on  acn06
logs/task_output.67134323_8:Task  8  on  acn06
logs/task_output.67134323_9:Task  9  on  acn06
```

Data Driver Workflow

Sentinel 2 data

- Top of atmosphere
- Focus on one tile: **T14TSQ** for 6 months (2020 May-Oct)
- Variation in distribution of values in band 3 (Green)

Data in Tier-2 bucket

86 tiles

- S2B_MSIL1C_20200502T171849_N0209_R012_T14TQS_20200502T204806.zip
- S2A_MSIL1C_20200504T170901_N0209_R112_T14TQS_20200504T204849.zip
- S2B_MSIL1C_20200509T170849_N0209_R112_T14TQS_20200509T203631.zip
- S2A_MSIL1C_20200507T171901_N0209_R012_T14TQS_20200507T205829.zip

  …

Split list

Split full list of work items into 20 pieces

- 20 files: x000 - x019
- 4-5 items per file

Each job in array

- Process one file

```
split -a 3 -d -n l/20 zip_list.txt

ls x*
x000  x001  x002  x003  x004  x005  x006  x007  x008  x009
x010  x011  x012  x013  x014  x015  x016  x017  x018  x019

cat x005
S2A_MSIL1C_20200626T171901_N0209_R012_T14TQS_20200626T222141.zip
S2B_MSIL1C_20200628T170849_N0209_R112_T14TQS_20200628T204257.zip
S2A_MSIL1C_20200703T170901_N0209_R112_T14TQS_20200703T204911.zip
S2B_MSIL1C_20200701T171859_N0209_R012_T14TQS_20200701T204911.zip
```

SLURM Script

- **Resources**
  - 1 core on 1 node
- **Work directory**
  - 1 per worker
- **Conda environment**
  - Supports rasterio
- **Loop over data**
  - Uses split file

s2_stats.sh

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem-per-cpu=3800
#SBATCH -t 00:10:00
#SBATCH -p agsmall
#SBATCH -o logs/task_output.%A_%a

workdir=w${SLURM_ARRAY_TASK_ID}
mkdir -p $workdir
cd $workdir

source ../conda_init_script
zlist=$(printf "x%03d" ${SLURM_ARRAY_TASK_ID})
for z in $(cat ../$zlist)
do
  bucket="s3://dhp-S2MSI1C"
  s3cmd get $bucket/$z
  unzip $z
  python ../s2stats.py >> ../B03_stats_$zlist
  rm -rf $(echo $z | cut -d'.' -f 1).*
done
conda deactivate
```

Python Script

Get Stats of an S2 Image:

- Path to the image file
- Date – from file name
- Read – use rasterio
- Calculate percentiles
- Write

s2stats.py

```
import rasterio
import glob
import numpy as np

# Get path to image file & extract date from name
jp2_path =
glob.glob("*/GRANULE/*/IMG_DATA/*B03.jp2")[0]
jp2_file = jp2_path.split("/")[-1]
date = jp2_file[7:11]+'-'+jp2_file[11:13]+'-'+jp2_file[13:15]

# Read in image values as a numpy ndarray
with rasterio.open(jp2_path) as src:
    a = src.read(1)

# Report percentile values (excluding nodata values)
b = np.extract(a > 0, a)
v01,v25,v50,v75,v99 = np.percentile(b, [1,25,50,75,99])
print(date, v01,v25,v50,v75,v99)
```

Run Task Array

Submission

sbatch –array=0-19 script

- 20 parallel workers
- script – can do a lot

Monitor

squeue -l -u $USER

- Long listing
- Only your jobs

```
ahl03:ta1 % sbatch  --array=0-19   s2_stats.sh
sbatch: Setting account: dhp
Submitted batch job 67134525

ahl03:ta1 % squeue -l -u $USER
             JOBID     PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES
       67134525_0    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_1    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_2    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_3    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_4    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_5    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_6    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_7    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_8    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
       67134525_9    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_10    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_11    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_12    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_13    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_14    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_15    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_16    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_17    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_18    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
      67134525_19    agsmall s2_stats      dhp  RUNNING       0:09      10:00      1 acn06
```

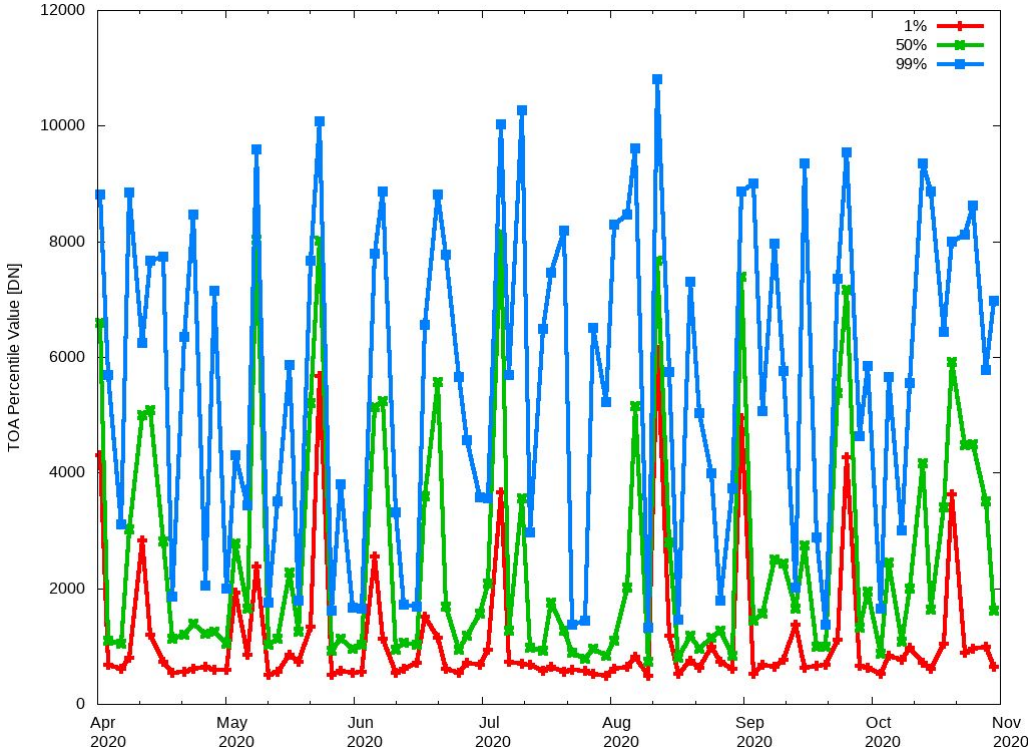
Output

Output files

B03_stats_x000 B03_stats_x001 B03_stats_x002
B03_stats_x003 B03_stats_x004 B03_stats_x005
B03_stats_x006 B03_stats_x007 B03_stats_x008
B03_stats_x009 B03_stats_x010 B03_stats_x011
B03_stats_x012 B03_stats_x013 B03_stats_x014
B03_stats_x015 B03_stats_x016 B03_stats_x017
B03_stats_x018 B03_stats_x019

cat B03* | sort > B03stats_full.txt

```
2020-04-02   4309   5792   6597   7596   8803 1.000
2020-04-04    671    918   1091   1376   5687 0.457
2020-04-07    608    864   1033   1255   3099 1.000
2020-04-09    805   1643   3017   5289   8839 0.364
2020-04-12   2831   4588   4993   5391   6235 1.000
2020-04-14   1201   3979   5088   5985   7670 0.463
2020-04-17    728   1422   2813   4907   7741 1.000
2020-04-19    542    955   1135   1324   1847 0.459
...
```



Job Arrays & Queues

SLURM Job Arrays:

https://slurm.schedmd.com/job_array.html

MSI Queues (Partitions):

https://www.msi.umn.edu/partitions#slurm

https://www.msi.umn.edu/content/choosing-job-partition#slurm

Thread Parallel : OpenMP

- **Agate compute nodes** — enable larger thread parallel applications
  - 128 cores
  - 512 GB — Shared memory between all 128 cores
  - 4 MB/core L3 — 32 MB shared between sets of 8 cores

- **SLURM job scripts**
- **OpenMP example** — Demonstrate good scaling

OpenMP Script

Resource request

- 1 node
- agsmall partition
- Up to 128 cores
- up to ~500GB

Commands

- load module(s)
- Set shell variables
- run application

```
#SBATCH --time=04:20:00
#SBATCH --ntasks-per-node=128
#SBATCH -N 1
#SBATCH --mem-per-cpu=3900
#SBATCH -p agsmall

module load intel/cluster/2018

export OMP_NUM_THREADS=128

./application.exe
```

Example: N-Body Simulation

- Simple
- Computationally intensive
- Tightly coupled
- Same or similar to
  - Galaxy Clustering
  - Stellar Dynamics
  - Electrostatic plasmas
  - Molecular Dynamics
  - Genomics

$$\{ (m_i, \vec{X}_i, \vec{V}_i) : i = 1, N \}$$

$$\frac{d\vec{X}_i}{dt} = \vec{V}_i$$

$$\frac{d\vec{V}_i}{dt} = \frac{1}{m_i} \vec{F}_i$$

$$\vec{F}_i = G m_i \sum_{j=1}^{N} \frac{m_j (\vec{X}_i - \vec{X}_j)}{\left(\left| \vec{X}_j - \vec{X}_i \right|^2 + s^2 \right)^{3/2}}$$

State Variables

Header file: nb.h

- Particle: m, X, V
- N-Body run parameters
- N up to 10 million
- Number of threads

Divide list of particles

- chunks: cache re-use
- i ranges:
  - one range per thread

```
integer nmax      ! Max number of particles
parameter(nmax = 10000000)
integer nsteps    ! Number of time steps
real grav         ! Gravitational constant
real s2           ! Force softening
real dt           ! Time step
integer n         ! Number of particles
real m(  nmax)    ! Particle masses
real x(3,nmax)    ! Particle positions
real v(3,nmax)    ! Particle velocities
real a(3,nmax)    ! Particle accelerations
common /nb_com/ grav, s2, dt, m, x, v, a, n, nsteps

integer nthreads
integer nchunk
integer iranges(0:1000)
common /nb_com/ nthreads, nchunk, iranges
```

Inputs & Initialization

Get run parameters

- Number of particles
- Number of threads
- Chunk size

Thread ranges

- Each thread gets a range of particles to update

Set number of threads

First Touch

Initialize state variables

- Positions
- Velocities

```
subroutine init()
include 'nb.h'

read (5,*) n, nsteps, nthreads, nchunk
iseed = 1003
s2    = 0.0001
dt    = 0.01
grav  = 1.0
m(:)  = 1.0 / float(n)
call set_range(nthreads, n, iranges)
call OMP_SET_NUM_THREADS(nthreads)
call first_touch()

call setrmy(iseed)
call fill_sphere(n, 1.0, x)
call fill_sphere(n, 5.0, v)

return
end
```

First Touch

Memory commit

- on first touch

NUMA

- Banks of memory
- Processors
  - cores
  - caches
  - directly connected memory

Threads touch memory they will most frequently reuse.

```
      subroutine first_touch()
      include 'nb.h'
      integer OMP_GET_THREAD_NUM

!$omp parallel private(mythread,i1,i2)  shared(iranges,x,v,a)
      mythread = OMP_GET_THREAD_NUM()
      i1 = iranges(mythread  ) + 1
      i2 = iranges(mythread+1)
      x(:, i1:i2) = 0.0
      v(:, i1:i2) = 0.0
      a(:, i1:i2) = 0.0
!$omp barrier
!$omp end parallel

      return
      end
```

Main routine

Initialization

Parallel Region

- i1,i2 – range of particles for thread
- Loop over time steps
  - Calculate forces
  - barrier
  - Update X & V
  - barrier

End Parallel Region

```
      include 'nb.h'
      integer OMP_GET_THREAD_NUM
      call init()

!$omp parallel private(istep,mythread,i1,i2)
      mythread = OMP_GET_THREAD_NUM()
      i1 = iranges(mythread  ) + 1
      i2 = iranges(mythread+1)
      do istep = 1, nsteps
        call calc_a(i1,i2)
!$omp barrier
        call update(i1,i2)
!$omp barrier
      enddo
!$omp end parallel

      stop
      end
```

Force Calculation

Calculate accelerations: a(:)

- from all force pairs

Thread particle ranges:

- Targets: i = i1, i2
- Sources: j = 1, N

Nested loops over

- Source range: j1,j2
  - Target: i
    - Source: j
  - Reuse J1,j2 in cache

```
subroutine calc_a(i1,i2)
include 'nb.h'
real aa(3), dx(3)

a(:, i1:i2) = 0.0
do j1 = 1, n, nchunk
 j2 = min(n, j1+nchunk-1)
 do i = i1, i2
  aa(:) = 0.0
  do j = j1, j2
   dx(:) = x(:,j) - x(:,i)
   rsinv = 1.0 / sqrt(dot(dx,dx) + s2)
   aa(:) = aa(:) + grav * m(j) * dx(:) * (rsinv**3)
  enddo
  a(:,i) = a(:,i) + aa(:)
 enddo
enddo

return
end
```

Particle Update

Update XYZ positions

- Use old velocity
- Use acceleration

Update Velocities

- Use acceleration

```
subroutine update(i1,i2)
include 'nb.h'
real dv(3)

do i = i1, i2
  dv(:)  = dt * a(:,i)
  x(:,i) = x(:,i) + dt * (v(:,i) + 0.5 * dv(:))
  v(:,i) = v(:,i) + dv(:)
enddo

return
end
```

Run N-Body Code

Job request

- 128 cores on 1 node
- Up to 475 GiB total
- L3 cache: 512 MB

N-body code uses

- 128 cores
- 524.288 particles
- ~20 GiB

```
#!/bin/bash -l
#SBATCH --time=04:20:00
#SBATCH --ntasks-per-node=128
#SBATCH -N 1
#SBATCH --mem-per-cpu=3900
#SBATCH -p agsmall

module load intel/cluster/2018

steps=20
chunk=512
n=524288
threads=128
echo "$n $steps $threads $chunk" | ./nb
```

Time vs. Threads

Time per full time step

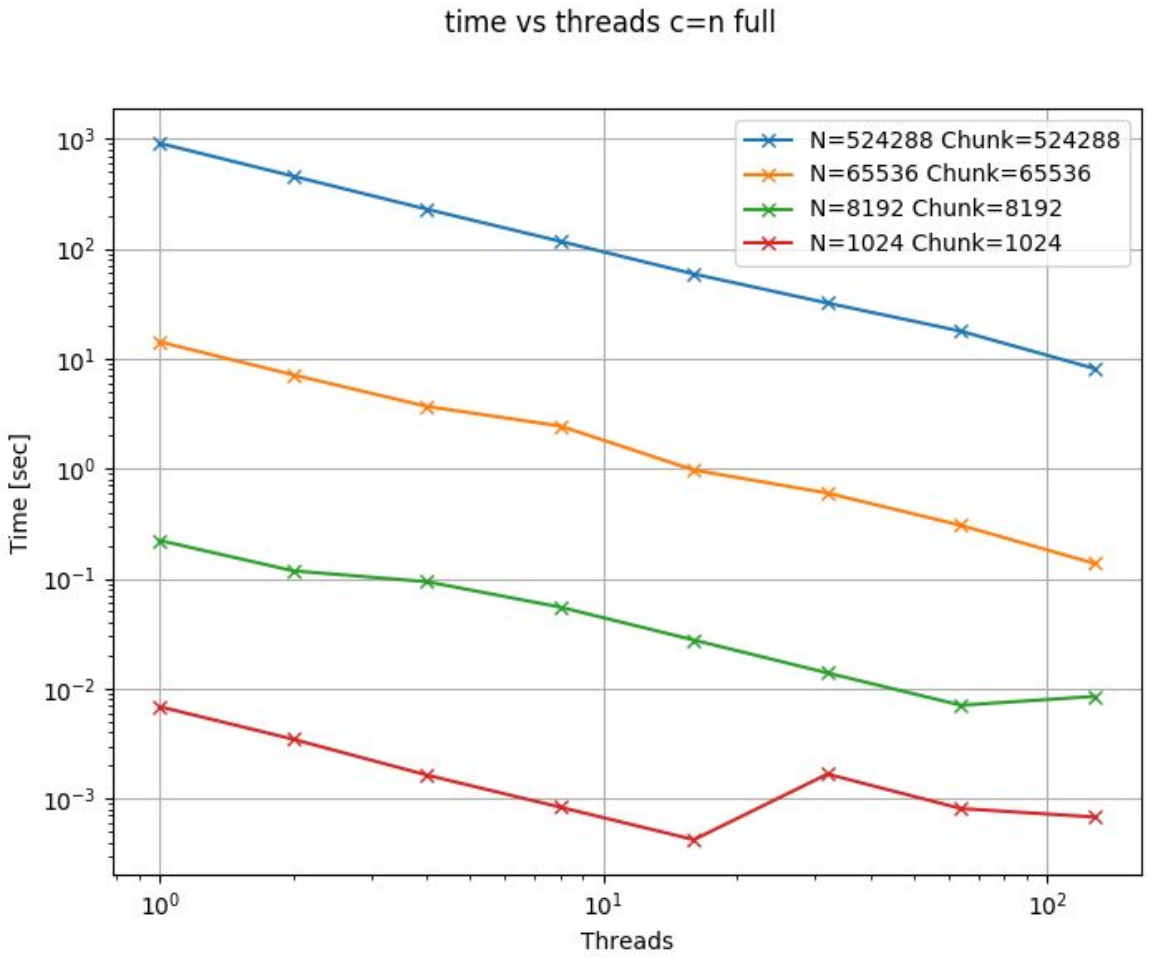
- threads: 1 to 128
- N: 1024 to 524,288

Two larger problems

- Speedup: 112

Limit on speedup: ~1ms

- Thread management
- Time slice



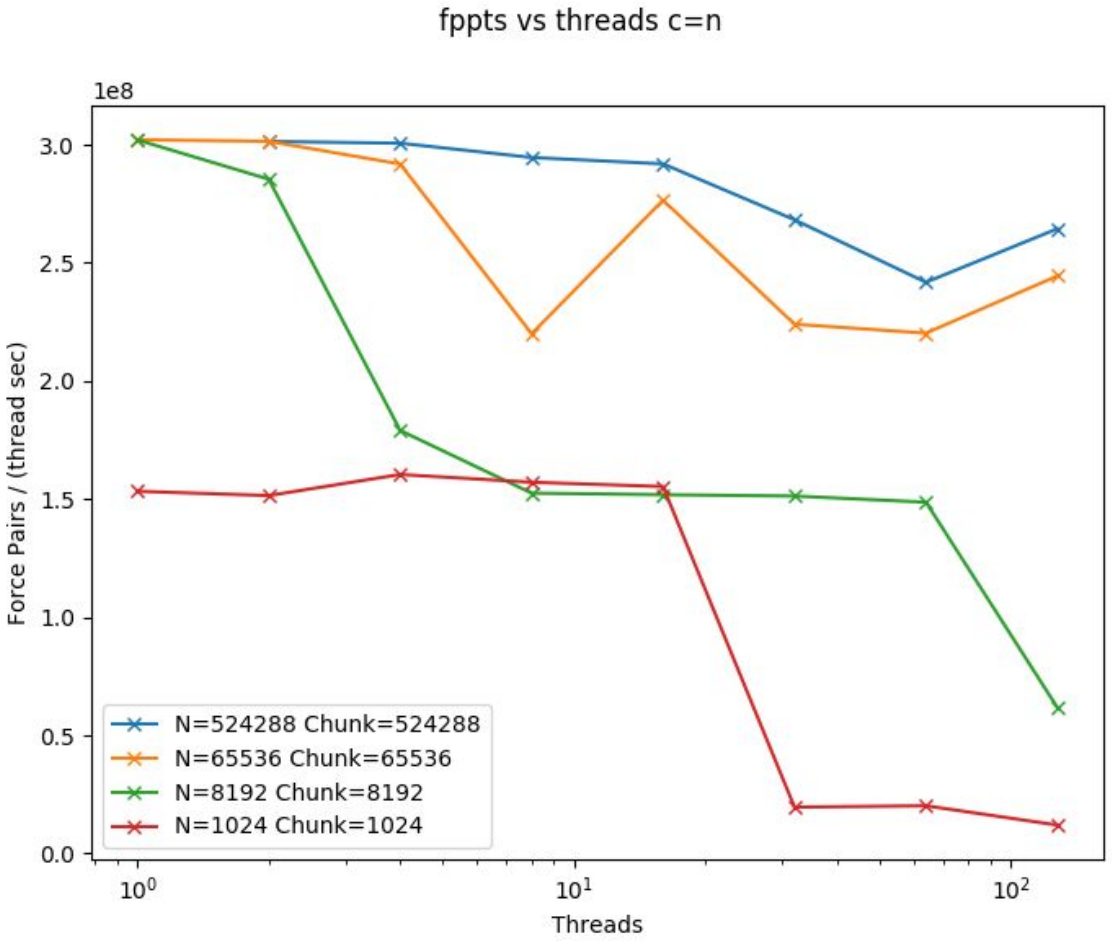
Strong Scaling

Total number of force pairs

- scale with number threads
- scale with wall clock time

Force Pairs pre thread per sec

- Perfect strong scaling
- Horizontal line



Tuning Performance

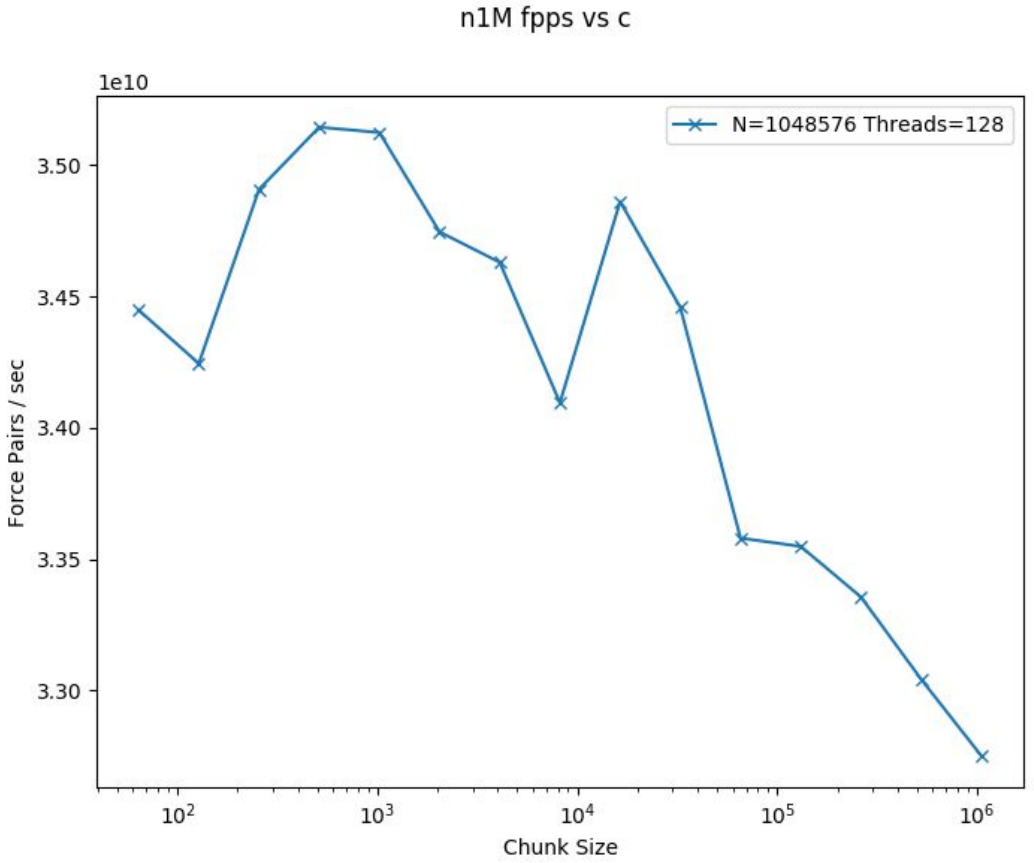
Force pairs per second

- 1 Million particles
- 128 threads

Optimal chunk size:

- 512
- Fits in L2 cache
- All million particles fit in L3

About 6% improvement



Thread Parallel Scaling and Performance

- Give each thread a lot of work to do
- Maximize size of parallel region
- Minimize barriers and synchronization points
- For large memory systems
  - First Touch
  - Cache reuse

OpenMP: Environment, Build, & Run

	Intel	GNU
Modules	module load intel/cluster/2018	module load gcc/7.2.0
Build	ifort -qopenmp code.f -o app icc -qopenmp code.c -o app	gfortran -fopenmp code.f -o app gcc -fopenmp code.c -o app
Shell Variables	export OMP_NUM_THREADS=24 export KMP_AFFINITY=granularity=core,scatter	export OMP_NUM_THREADS=24 export OMP_PROC_BIND=TURE
Run	./app numactl --interleave=0,1 ./app	./app

- **Run on MSI's HPC systems**
- **This is a sample. See documentation on next slide**

OpenMP Documentation & Resources

Tutorials & Articles: https://www.openmp.org/resources/tutorials-articles/

Summary: https://opensees.berkeley.edu/OpenSees/workshops/parallel09/03_ParallelIntroduction.pdf

Intel: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb-documentation.html

GNU: https://gcc.gnu.org/onlinedocs/libgomp/index.html#Top

Specifications (OpenMP Versions 5.2 & earlier):

https://www.openmp.org/specifications/

MPI Applications

- Message Passing Interface
- SLURM Job Scripts
- Examples for using memory and processors across multiple nodes
- References

Basic MPI Script

Job requests:

- 32 nodes
  - Max in aglarge
- 128 cores per node
- 3900 MB per core
- Most Agate nodes
  - 128 cores
  - 512 GB

MPI application uses:

- 4096 ranks (cores)
- Ranks spread across nodes by default

```
#!/bin/bash -l
#SBATCH --time=00:20:00
#SBATCH --ntasks-per-node=128
#SBATCH -N 32
#SBATCH --mem-per-cpu=3900
#SBATCH -p aglarge

module load intel/cluster/2018

mpirun -np 4096 ./mpi_application .exe
```

MPI Memory Test

Main Routine

- Get #MB per rank
- Initi MPI
- Use Memory
- Count hosts
- Report
- Finalize MPI

```
include 'mpif.h'
character*12 cArg
real*8 total, expect

call getarg(1, cArg)
read (cArg,*) nmib

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, nranks, ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD, myrank, ierr)

call calc(myrank, nmib, total)
call count_hosts(nranks, nhosts)

if(myrank .eq. 0) then
  write (6,*) "nranks, nhosts: ", nranks, nhosts
  write (6,*) "MiB per rank  : ", nmib
  write (6,*) "MiB per node  : ", nmib * nranks / nhosts
  expect = dble(131072) * dble(nmib)*dble((nranks*(nranks-1))/2)
  write (6,*) "total  : ", total
  write (6,*) "expect : ", expect
endif

call MPI_FINALIZE(ierr)

 stop
 end
```

Use Memory

subroutine calc

- allocate memory
- commit memory
- BARRIER
  - Ensures all memory is used at same time
- Sum array elements
- MPI_Reduce
  - sum over ranks
- deallocate memory

```
subroutine calc(myrank, nmib, total)
include 'mpif.h'
real*8 a, asum, total
allocatable a(:,:)

allocate(a(131072,nmib))
a(:,:) = dble(myrank)
call MPI_BARRIER(MPI_COMM_WORLD, ierr)

asum = 0.0d0
do j = 1, nmib
do i = 1, 131072
  asum = asum + a(i,j)
enddo
enddo

call mpi_Reduce(asum, total, 1, MPI_DOUBLE_PRECISION,
1               MPI_SUM, 0, MPI_COMM_WORLD,iError)
deallocate(a)

return
end
```

Node Names

Not needed for computing

- name not important

Can be handy for testing

- Slow nodes
- iB bandwidth

Get list of most names

- 1 per rank
- count unique names

```
subroutine count_hosts(nranks, nhosts)
      include 'mpif.h'
      character*12 hostname, hosts(100000), last_host

      hostname = "            "
      call MPI_Get_processor_name(hostname, len, ierr)
      call MPI_Gather(hostname, 12, MPI_CHARACTER,
     1              hosts   , 12, MPI_CHARACTER,
     1              0, MPI_COMM_WORLD, iError)

      nhosts = 0
      last_host = "            "
      do i = 1, nranks
        if(last_host .ne. hosts(i)) then
          nhosts = nhosts + 1
          last_host = hosts(i)
        endif
      enddo

      return
      end
```

Memory Test

Job requests:

- 8 nodes
- 128 cores per node
- 3900 MB per core

Can build app in job

MPI application uses:

- 1024 ranks (cores)
- Input parameter
  - 3800 MB per rank

Memory used per node:

- 475 GiB

```
#!/bin/bash -l
#SBATCH --time=00:20:00
#SBATCH --ntasks-per-node=128
#SBATCH -N 8
#SBATCH --mem-per-cpu=3900
#SBATCH -p aglarge
module load intel/cluster/2018
mpiifort -O3 -g hello_mpi_memory.f -o hello_mpi_memory
mpirun -np 1024 ./hello_mpi_memory 3800
```

```
sbatch: Setting account: dhp
Submitted batch job 67137025

ahl03:mpi % cat test_n8.out
 nranks, nhosts:        1024          8
 MiB per rank  :        3800
 MiB per node  :      486400
 total  :    260878997913600.
 expect : 260878997913600.
```

MPI References & Resources

Tutorials:

https://www.msi.umn.edu/sites/default/files/IntroMPI2019march07.pptx_.pdf
https://hpc-tutorials.llnl.gov/mpi/

MSI Software Pages:

https://www.msi.umn.edu/sw/openmpi
https://www.msi.umn.edu/sw/intel-mpi

Specification, Documentation, & Descriptions:

https://www.mpi-forum.org/docs/
https://en.wikipedia.org/wiki/Message_Passing_Interface

Examples in this document

https://public.s3.msi.umn.edu/tutorials/march_31_2022.tar.gz

Parallel Computing Consierations

- Multi-socket nodes with rapidly increasing core counts.
- Memory bandwidth per core decreasing.
- Network bandwidth per core decreasing.

⇒ Hybrid programming model with 4 levels of parallelism

– Workflow	task parallel	Multiple cases.
– Distributed Memory	MPI	Multiple processes.
– Shared memory	OpenMP	Multiple threads.
– Vectorization	SIMD	Multiple elements / loop indices

Questions?