SMP

Twelve of the Ocelote compute nodes are aggregated together using sophisticated software from ScaleMP. This creates a virtual SMP node that presents a single image with 336 cores and 2TB memory.

As the vSMP product from ScaleMP is a unique tool, any job run there is likely to need some simple but special configuration in order to take full advantage of its capabilities. This customization depends on the type of application you are looking to run, so please ask our consultants for assistance.

Using the vSMP node

As the vSMP product from ScaleMP is a unique tool, any job run there is likely to need some simple but special configuration in order to take full advantage of its capabilities. This customization depends on the type of application you are looking to run, so please ask our consultants for assistance.

You will use either the standard or windfall queues. And you need to specify "vsmp=1" in the select line as in the examples below.

Binding

The concept of binding in a SMP environment is that you generally need a few CPU's (cores) and lots of memory. The job will run significantly faster if the cores are located on as few boards as possible (each of the 12 nodes in the vSMP cluster is considered a board and contains 2 CPU sockets with 14 cores each). Binding will concentrate the cores together in one of two ways: static is for when you can name which board you will use, and dynamic when you do not. We will always use dynamic to support a multi-user environment where you do not know or care what other jobs are running.

Further in this concept is that you will ask for as few cores as required and do not keep to the 6GB memory per core relationship (pcmem). For example in the Trinity case below, if you need 960GB memory don't request 160 cores if 28 will do the job. You are keeping all the CPU workload on one board and dramatically improving performance.

Numabind

The examples below use numabind with a bind configuration script. The job starts then runs in the background while the numabind function finds the tasks then aggregates them.

Sample PBS Scripts

Trinity
#!/bin/bash
#PBS -N TrinSMP1006
#PBS -m bea
#PBS -M netid@email.arizona.edu
#PBS -W group_list=PI
#PBS -l select=1:ncpus=160:mem=960gb:vsmp=1
#PBS -l pvmem=960gb
#PBS -q standard
#PBS -l walltime=100:0:0
#PBS -l cput=1600:0:0
#PBS -V

# Setting stacksize to unlimited. Optional; does not hurt but sometimes helps.
ulimit -s unlimited

# Path to numabind command
export PATH=/opt/ScaleMP/numabind/bin:$PATH

# This will get a comma delimited list of 28 CPUs.
cpulist="`numabind --offset 28 --flags=best`"

# This will set the affinity of the job script to those CPUs.
taskset -pc $cpulist $$

# Now we run Trinity (and it will "inherit" the same affinity)
cd $PBS_O_WORKDIR
module load trinity
Trinity --seqType fq --CPU 28 --max_memory 160G --output /dev/shm/trinity960_out \
--left pair.AGTTCC.fastq.gz --right pair.AGTTCC.fastq.gz &
sleep 3
numabind --config serialconfig
cp -prd /dev/shm/trinity960_out .

Sample serialconfig file

name=serial-mm pattern=serial-mm verbose=0 process_allocation=cpu
task_affinity=cpu rule=RULE-throughput.so
#!/bin/bash

#PBS -q standard
#PBS -l select=1:ncpus=210:mem=1260gb:pcmem=6gb:vsmp=1
#PBS -N generic-smp
#PBS -W group_list=PI
#PBS -l cput=37800:00:00
#PBS -l walltime=180:00:00

# Setting stacksize to unlimited. Optional; does not hurt but sometimes helps.
ulimit -s unlimited

# Path to numabind command
export PATH=/opt/ScaleMP/numabind/bin:$PATH

source Python_Files/virtenv/bin/activate

cd Python_Files/

python strategies.py &
sleep 3

numabind --offset 28 --flags=best

---

Serial

Below is an example script to run a serial job.
#!/bin/bash

#PBS -q standard
#PBS -l select=1:ncpus=1:mem=6gb:vsmp=1
#PBS -N s-v
#PBS -W group_list=PI
#PBS -l cput=0:10:00
#PBS -l walltime=0:10:00

# Setting stacksize to unlimited. Optional; does not hurt but sometimes helps.
ulimit -s unlimited

# PATH to numabind
export PATH=/opt/ScaleMP/numabind/bin:$PATH

cd ~netid/vsmp

./serial-mm &
sleep 3
numabind --offset 28 --flags=best
wait

Throughput
Below is an example script to run a throughput job which means multiple instances of one job using a "for" loop
#!/bin/bash
#PBS -q standard
#PBS -l select=1:ncpus=8:mem=48gb:vsmp=1
#PBS -N t-v
#PBS -W group_list=hpcteam
#PBS -l cput=0:20:00
#PBS -l walltime=0:5:00

# Setting stacksize to unlimited. Optional; does not hurt but sometimes helps.
ulimit -s unlimited

# PATH to numabind
export PATH=/opt/ScaleMP/numabind/bin:$PATH

# Number of throughput jobs to launch
np=8

for i in `seq 1 $np`;do
  log=log-throughput-nbind-$i.txt
  /usr/bin/time ./throughput-mm > $log 2>&1 &
done

# Wait few seconds for jobs to be created
sleep 2

# Place the jobs using numabind
echo 'throughput-mm: process affinity by numabind' > numabind.log
numabind --offset 28 --flags=best
wait

OpenMP

Below is an example script for running your OpenMP code on the scalemp machine.
#!/bin/bash
#PBS -q standard
#PBS -l select=1:ncpus=8:mem=48gb:vsmp=1
#PBS -N openmp-v
#PBS -W group_list=hpcteam
#PBS -l cput=0:20:00
#PBS -l walltime=0:5:00

# Setting stacksize to unlimited. Optional; does not hurt but sometimes helps.
ulimit -s unlimited
# PATH to numabind
export PATH=/opt/ScaleMP/numabind/bin:$PATH

cd ~chrisreidy/vsmp/OpenMP
export MKL_VSMP=1

for np in 8 16 32 ; do
   # Dynamic binding of OpenMP threads using numabind.
   export KMP_AFFINITY=compact,verbose,0,`numabind --offset $np 2> /dev/null`
   export OMP_NUM_THREADS=$np
   /usr/bin/time ./openmp-mm > log${np}.txt 2>&1
done

**Threaded**

Below is an example script for running a job using Pthreads.
#!/bin/bash

#PBS -q standard
#PBS -l select=1:ncpus=8:mem=48gb:vsmp=1
#PBS -N openmp-v
#PBS -W group_list=hpcteam
#PBS -l cput=0:20:00
#PBS -l walltime=0:5:00

cd ~netid/vsmp/Pthread

# PATH to numabind.
export PATH=/opt/ScaleMP/numabind/bin:$PATH

# Specify sleep duration for each pthread. Default = 60 sec if not set
export SLEEP_TIME=120

# 16 pthreads would be created
NP=16

./ptest $NP > log-pthread-app-$NP 2>&1 &

# Waiting for 15 seconds for all the threads to start
sleep 15

# Start numabind with a config file that has a rule for pthread,
# which would place all threads to consecutive cpus.
numabind --offset 28 --flags=best

wait