Lab: Hybrid Programming and NUMA Control

Steve Lantz

Introduction to Parallel Computing
May 19, 2010

Based on materials developed by Bill Barth at TACC
What You Will Learn

• How to use numactl in the execution of serial, MPI and 4xN hybrid (i.e., 4 tasks, each with N threads) codes

• How to structure communications in a 2x16 hybrid code that involves threaded MPI calls between 2 nodes
  – MPI calls from serial region
  – MPI calls from master thread in a parallel region
  – MPI calls from all threads in a parallel region

• Measuring performance of the above codes

• Determining performance implications from using numactl and threaded MPI
Getting Started

• Untar the file numahybrid.tar
  – cd ~                   (start in your home directory)
  – tar xvf ~train400/numahybrid.tar   (extract files)
  – cd numahybrid
numactl_serial

- Run the memory intensive daxpy program on four different sockets using local, interleave and off-socket-memory policies.
  - Use the commands below to make the daxpy executable and run it with numa control commands.
  - See the job script and the table on the next page for the numa options.
  - Run the job and report the times and relative performance.

- Procedure:
  - cd numactl_serial (change directory to numactl_serial)
  - module unload mvapich; module swap pgi intel; module load mvapich
  - make
  - qsub job (submits job)
numactl_serial – Results

- From the job output fill in the table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>numactl –l -C 0</td>
<td></td>
</tr>
<tr>
<td>numactl –l -C 1</td>
<td></td>
</tr>
<tr>
<td>numactl –l -C 2</td>
<td></td>
</tr>
<tr>
<td>numactl –l -C 3</td>
<td></td>
</tr>
<tr>
<td>numactl –i all –C 0</td>
<td></td>
</tr>
<tr>
<td>numactl –i all –C 1</td>
<td></td>
</tr>
<tr>
<td>numactl –i all –C 2</td>
<td></td>
</tr>
<tr>
<td>numactl –i all –C 3</td>
<td></td>
</tr>
<tr>
<td>numactl –m 3 –C 0</td>
<td></td>
</tr>
</tbody>
</table>

Rank the performance of local, interleave, and off-socket-memory policies, from best to poorest:
1.)
2.)
3.)
numactl_mpi

• Run the memory intensive daxpy program on four different sockets simultaneously using local, interleave and off-socket-memory policies.
  – Use the commands below to make the daxpy executable and run it with numa control commands.
  – See the job script and the table on the next page for the numa options.
  – Run the job and report the times and relative performance.

• Procedure:
  – cd numactl_mpi  (change directory to numactl_mpi)
  – if you have done this already, don’t do it again:
    module unload mvapich; module swap pgi intel; module load mvapich
  – make
  – qsub job  (submits job)
numactl_mpi – Results

- From the job output fill in the table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>numactl –l</td>
<td></td>
</tr>
<tr>
<td>numactl –i all</td>
<td></td>
</tr>
<tr>
<td>numactl tacc_affinity</td>
<td></td>
</tr>
</tbody>
</table>

Rank the performance of local, interleave, and tacc_affinity policies, from best to poorest
1.)
2.)
3.)
What is tacc_affinity?

It's a script: /share/sge6.2/default/pe_scripts/tacc_affinity

#!/bin/bash
MODE=`/share/sge6.2/default/pe_scripts/getmode.sh`
# First determine "wayness" of PE
myway=`echo $PE | sed s/way//` # Determine local compute node rank number
if [ x"$MODE" == "xmvapich2_ssh" ]; then
    export MV2_USE_AFFINITY=0
    export MV2_ENABLE_AFFINITY=0
my_rank=$PMI_ID
elif [ x"$MODE" == "xmvapich1_ssh" ]; then
    export VIADEV_USE_AFFINITY=0
    export VIADEV_ENABLE_AFFINITY=0
my_rank=$MPIRUN_RANK
else
    echo "TACC: Could not determine MPI stack. Exiting!"
    exit 1
fi
local_rank=$(( $my_rank % $myway ))
...
What is tacc_affinity? – Part 2

# Based on "wayness" determine socket layout on local node
# if less than 4-way, offset to skip socket 0
if [ $myway -eq 1 ]; then
    numnode="0,1,2,3"
# if 2-way, set 1st task on 0,1 and second on 2,3
elif [ $myway -eq 2 ]; then
    numnode="$( ( 2 * $local_rank ) ),$( ( 2 * $local_rank + 1 ) )"
else
    numnode="$( ( $local_rank + 1 ) )"
# if 4-way to 12-way, spread processes equally on sockets
elif [ $myway -lt 13 ]; then
    numnode="$( ( $local_rank / ( $myway / 4 ) ) )"
# if 16-way, spread processes equally on sockets
elif [ $myway -eq 16 ]; then
    numnode="$( ( $local_rank / ( $myway / 4 ) ) )"
# Offset to not use 4 processes on socket 0
else
    numnode="$( ( $local_rank + 1 ) / 4 )"
fi
#echo "TACC: Running $my_rank on socket $numnode"
exec /usr/bin/numactl -c $numnode -m $numnode $*
numactl_4x1, numactl_4x4

- Run the daxpy program as 4 tasks in a node (4x1) and 4 tasks with 4 threads in a node (4x4), following the instructions below.
  - Use the commands below to make the daxpy executable and run it with numa control commands.
  - See the job script and the table on the next page for the numa options.
  - Run the job and report the times and relative performance.

- Procedure:
  - cd numactl_4x1 or numactl_4x4 (change directory as needed)
  - if you have done this already, don’t do it again:
    module unload mvapich; module swap pgi intel; module load mvapich
  - make
  - qsub job (submits job)
numactl_4x1, numactl_4x4 – Results

- From the job output fill in the table.

<table>
<thead>
<tr>
<th>Command (4x1)</th>
<th>Time (secs)</th>
<th>Rank 4x1 performance from best to poorest</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;no numactl&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>numactl –l</td>
<td></td>
<td>1.)</td>
</tr>
<tr>
<td>numactl –i all</td>
<td></td>
<td>2.)</td>
</tr>
<tr>
<td>numactl tacc_affinity</td>
<td></td>
<td>3.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command (4x4)</th>
<th>Time (secs)</th>
<th>Rank 4x4 performance from best to poorest</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;no numactl&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>numactl –l</td>
<td></td>
<td>1.)</td>
</tr>
<tr>
<td>numactl –i all</td>
<td></td>
<td>2.)</td>
</tr>
<tr>
<td>numactl tacc_affinity</td>
<td></td>
<td>3.)</td>
</tr>
</tbody>
</table>
Communications in Hybrid Codes

• The tmpi (threaded mpi) code illustrates different ways of doing point-to-point and broadcast communications in a hybrid code. Using both mvapich and openmpi, we will:
  – check to make sure the code performs correctly
  – measure the cost for sending a single large message in the serial region
  – compare the cost for sending 16 small messages in the parallel region

• Procedure:
  – cd threaded_mpi
  – if you have done this already, don’t do it again:
    module unload mvapich; module swap pgi intel; module load mvapich
  – ./build.sh (this builds tmpi.mvapich1 and tmpi.openmp)
Hybrid Job Script

Script for 10 interactive minutes of 2 nodes (32/16), 1 task per node (1way), 2 tasks total, in the development queue. 16 threads (OMP_NUM_THREADS 16) are launched on each node.

```bash
#!/bin/tcsh
#
#$ -V                    # inherit submission environment
#$ -cwd                  # use submission directory
#$ -N threadedmpi       # jobname (threadedmpi)
#$ -j y                  # stdout/err combined
#$ -o $JOB_NAME.o$JOB_ID # output name jobname.ojobid
#$ -pe 1way 32           # 1 task/node, 32 cores total
#$ -q development        # queue name !! can use normal
#$ -l h_rt=00:10:00      # request 10 minutes
## -M <myemail_addr>     # Mail address !! your own mail
## -m be                 # send email at begin/end of job
set echo                # echo cmds, use "set -x" in sh
setenv MY_NSLOTS 2
setenv OMP_NUM_THREADS 16
ibrun ./tmpi < input
```

If # of tasks is not equal to a multiple of 16, set value here.
Submit the Batch Job

% qsub job
...
Welcome to TACC's Ranger System, an NSF Teragrid Resource
  --> Submitting 2 tasks...
  --> Submitting 1 tasks/host...
  --> Submitting exclusive job to 2 hosts...
...
Your job 18073 ("threadedmpi") has been submitted

% qstat
job-ID prior  name user state submit/start at queue slots
--------------------------------------------------------------------------------
  18075 0.00001 threadedmp milfeld r  01/17/2008 22:48:54 normal@i104-408 32

% showq
...

Communication from Serial Region

include "mpif.h"

... call MPI_Init_thread(MPI_THREAD_MULTIPLE, iprovided,ierr) call MPI_Comm_size(MPI_COMM_WORLD,nranks, ierr) call MPI_Comm_rank(MPI_COMM_WORLD,irank,ierr)

if(irank == 0) then
    call mpi_send(as,n,MPI_REAL8, 1,9,MPI_COMM_WORLD, ierr)
    call mpi_recv(as,n,MPI_REAL8, 1,1,MPI_COMM_WORLD, istatus,ierr)
else if (irank == 1) then
    call mpi_recv(as,n,MPI_REAL8, 0,9,MPI_COMM_WORLD, istatus,ierr)
    call mpi_send(as,n,MPI_REAL8, 0,1,MPI_COMM_WORLD, ierr)
endif

if(irank .eq. 0) read(*,'(i5)') iread1 call MPI_Bcast(iread1,1,MPI_INTEGER, 0,iwcomm, ierr)

(don’t forget error argument in f90 codes)
Broadcast in Parallel Region

```fortran
!$OMP PARALLEL private(i,ithread,nthreads, icp1, icp2, icpd)

ithread = OMP_GET_THREAD_NUM()

if(ithread == 0) then
  if(irank .eq. 0) read(*,'(i5)') iread2
  call MPI_Bcast(iread2,1,MPI_INTEGER, 0,iwcomm, ierr)
end if
```

(don’t forget error argument in f90 codes)
Point-to-point in Parallel Region

```
!$OMP DO ordered
  do i = 1,nthreads
    !$OMP ordered
      if(irank == 0) then
        call mpi_send(as,ns,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, ierr)
        call mpi_recv(as,ns,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, istatus,ierr)
      else if (irank == 1) then
        call mpi_recv(as,ns,MPI_REAL8, 0,ithread,MPI_COMM_WORLD, istatus,ierr)
        call mpi_send(as,ns,MPI_REAL8, 0,ithread,MPI_COMM_WORLD,ierr)
      endif
    !$OMP end ordered
  end do
if(irank == 0 .and. ithread == 0) then
  call mpi_send(as,n,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, ierr)
  call mpi_recv(ar,n,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, istatus,ierr)
else if (irank == 1 .and. ithread == 0) then
  call mpi_recv(ar,n,MPI_REAL8, 0,ithread,MPI_COMM_WORLD, istatus,ierr)
  call mpi_send(as,n,MPI_REAL8, 0,ithread,MPI_COMM_WORLD,ierr)
endif
!$OMP barrier
!$OMP END PARALLEL
call mpi_finalize(ierr)
```
### Hybrid Communication Cost (Output from tmpi)

**Mvapich1**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Region Ping Pong</td>
<td>0.00509</td>
</tr>
<tr>
<td>Serial Region Broadcast</td>
<td>0.00002</td>
</tr>
<tr>
<td>Parallel Region Broadcast</td>
<td>0.00001</td>
</tr>
<tr>
<td><strong>Parallel region messages:</strong></td>
<td></td>
</tr>
<tr>
<td>One Large message</td>
<td>0.00555</td>
</tr>
<tr>
<td>16 Small messages</td>
<td>0.00548</td>
</tr>
<tr>
<td><strong>individual times:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00042 0.00033 0.00038 0.00033 0.00033 0.00033 0.00033 0.00033 0.00034 0.00033 0.00033 0.00033</td>
</tr>
</tbody>
</table>

**OpenMPI**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Region Ping Pong</td>
<td>0.00501</td>
</tr>
<tr>
<td>Serial Region Broadcast</td>
<td>0.00005</td>
</tr>
<tr>
<td>Parallel Region Broadcast</td>
<td>0.00001</td>
</tr>
<tr>
<td><strong>Parallel region messages:</strong></td>
<td></td>
</tr>
<tr>
<td>One Large message</td>
<td>0.00535</td>
</tr>
<tr>
<td>16 Small messages</td>
<td>0.13446</td>
</tr>
<tr>
<td><strong>individual times:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.12864 0.00038 0.00038 0.00039 0.00038 0.00065 0.00036 0.00036 0.00036 0.00036 0.00038 0.00034 0.00038 0.00037 0.00036 0.00037 0.00036 0.00036 0.00036</td>
</tr>
</tbody>
</table>