NUMA Control for Hybrid Applications

Hang Liu
TACC
October, 23rd, 2012
Hybrid Applications

• Typical definition of hybrid application
  – Uses both message passing (MPI) and a form of shared memory algorithm (OMP)
  – Runs on multicore systems

• Hybrid execution does not guarantee optimal performance
  – Multicore systems have multilayered, complex memory architecture
  – Actual performance is heavily application dependent

• **Non-Uniform Memory Access -NUMA**
  – Shared memory with underlying multiple levels
  – Different access latencies for different levels
  – Complicated by asymmetries in multisocket, multicore systems
  – More responsibility on the programmer to make application efficient
Modes of Hybrid Operation

1. Pure MPI
   - Thread on each Core

2. 1 MPI Task
   - 12 threads/task

3. 2 MPI Tasks
   - 6 threads/task

4. 1 MPI Tasks
   - 12 threads/task

Legend:
- Green: MPI Task on Core
- Purple: Slave Thread of MPI Task
- Green: Master Thread of MPI Task

TACC
Texas Advanced Computing Center
The University of Texas at Austin
Needs for NUMA Control

• Asymmetric multi-core configuration on node requires better control on core affinity and memory policy.
  – Load balancing issues on node

• Slowest CPU/core on node may limit overall performance
  – use only balanced nodes, or
  – employ special in-code load balancing measures

• Applications performance can be enhanced by specific arrangement of
  – tasks (process affinity)
  – memory allocation (memory policy)
NUMA Operations

- Each process/thread is executed by a core and has access to a certain memory space
  - Core assigned by process affinity
  - Memory allocation assigned by memory policy

- The control of process affinity and memory policy using NUMA operations
  - NUMA Control is managed by the kernel (default).
  - Default NUMA Control settings can be overridden with numaclt.
NUMA Operations

• Ways Process Affinity and Memory Policy can be managed:
  – Dynamically on a running process (knowing process id)
  – At process execution (with wrapper command)
  – Within program through F90/C API

• Users can alter Kernel Policies by manually setting Process Affinity and Memory Policy
  – Users can assign their own processes onto specific cores.
  – Avoid overlapping of multiple processes
Affinity and Policy can be changed externally through `numactl` at the socket and core level.

Command: `numactl <options> ./a.out`

- Process affinity: socket references and core references
- Memory policy: socket references
# numactl Options on Lonestar

<table>
<thead>
<tr>
<th>Affinity Type</th>
<th>Command</th>
<th>Option</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket Affinity</td>
<td>numactl</td>
<td><code>-N</code></td>
<td><code>{0,1}</code></td>
<td>Only execute process on cores of this (these) socket(s).</td>
</tr>
<tr>
<td>Memory Policy</td>
<td>numactl</td>
<td><code>-l</code></td>
<td><code>{no argument}</code></td>
<td>Allocate on current socket.</td>
</tr>
<tr>
<td>Memory Policy</td>
<td>numactl</td>
<td><code>-i</code></td>
<td><code>{0,1}</code></td>
<td>Allocate round robin (interleave) on these sockets.</td>
</tr>
<tr>
<td>Memory Policy</td>
<td>numactl</td>
<td><code>--preferred=▢</code></td>
<td><code>{0,1}</code></td>
<td>Select only one; allocate on this socket; fallback to any other if full.</td>
</tr>
<tr>
<td>Memory Policy</td>
<td>numactl</td>
<td><code>-m</code></td>
<td><code>{0,1}</code></td>
<td>Only allocate on this (these) socket(s).</td>
</tr>
<tr>
<td>Core Affinity</td>
<td>numactl</td>
<td><code>--physcpubind</code></td>
<td><code>{0,1,2,3, 4,5,6,7, 8,9,10,11}</code></td>
<td>Only execute process on this (these) Core(s).</td>
</tr>
</tbody>
</table>
General Tips for Process Affinity and Memory Policies

Process affinity:
• MPI tasks shall be evenly populated on multi sockets
• Threads per task shall be evenly loaded on multi cores

Memory policy:
• MPI – local is best
• SMP – Interleave may be the best for large, completely shared arrays
• SMP – local may be the best for private arrays
• Once allocated, memory structure is fixed
Hybrid Runs with NUMA Control

- A single MPI task (process) is launched and becomes the “master thread”.
- It uses any `numactl` options specified on the launch command.
- When a parallel region forks the slave threads, the slaves inherit the affinity and memory policy of the master thread (launch process).
Hybrid Batch Script 12 threads

- Make sure 1 MPI task is created on each node
- Set number of OMP threads for each node
- Can control only memory allocation
- No simple/standard way to control thread-core affinity

```
job script (Bourne shell)
...
#!/pe 1way 96
...
export OMP_NUM_THREADS=12
ibrunch numactl –i all ./a.out
```
Hybrid Batch Script   2 tasks, 6 threads/task

job script  (Bourne shell)

...  
#!/bin/bash
# Unset any MPI Affinities
export MV2_USE_AFFINITY=0
export MV2_ENABLE_AFFINITY=0
export VIADEV_USE_AFFINITY=0
export VIADEV_ENABLE_AFFINITY=0
# Get rank from appropriate MPI API variable
[ "x$MPIRUN_RANK" != "x" ] && myrank=$MPIRUN_RANK
[ "x$PMI_ID" != "x" ] && myrank=$PMI_ID
[ "x$OMPI_COMM_WORLD_RANK" != "x" ] &&
myrank=$OMPI_COMM_WORLD_RANK
[ "x$OMPI_MCA_ns_nds_vpid" != "x" ] && myrank=$OMPI_MCA_ns_nds_vpid
localrank=$(( $myrank % 2 ))
socket=localrank
exec numactl --cpunodebind $socket -m $socket ./a.out
Hybrid Batch Script with tacc affinity

- Simple setup for ensuring evenly distributed core setup for your hybrid runs.
- tacc_affinity is not the single magic solution for every application out there - you can modify the script and replace tacc_affinity with yours for your code.

```bash
job script (Bourne shell)
...
#!/ -pe 4way 192
...
export OMP_NUM_THREADS=3
ibrun tacc_affinity ./a.out
```
#!/bin/bash
# -*- shell-script -*-

# First determine "wayness" of PE
myway=`builtin echo $PE | /bin/sed s/way//`

export MV2_USE_AFFINITY=0
export MV2_ENABLE_AFFINITY=0
export VIADEV_USE_AFFINITY=0
export VIADEV_ENABLE_AFFINITY=0

my_rank=($( $(PMI_RANK-0) + $(PMI_ID-0) + $(MPIRUN_RANK-0) + $(OMPI_COMM_WORLD_RANK-0) + $(OMPI_MCA_ns_nds_vpid-0) ))
local_rank=$(( my_rank % myway ))

if [ $myway -eq 1 ]; then
  numnode="0,1"
elif [ $myway -eq 2 ]; then
  numnode=$local_rank
elif [ $myway -le 4 ]; then
  numnode=$(( local_rank / 2 ))
elif [ $myway -le 6 ]; then
  numnode=$(( local_rank / 3 ))
elif [ $myway -le 8 ]; then
  numnode=$(( local_rank / 4 ))
elif [ $myway -le 10 ]; then
  numnode=$(( local_rank / 5 ))
elif [ $myway -le 12 ]; then
  numnode=$(( local_rank / 6 ))
fi

exec numactl --cpunodebind=$numnode --membind=$numnode $*

---

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

TEXAS ADVANCED COMPUTING CENTER
Summary

• NUMA control ensures hybrid jobs to run with optimal core affinity and memory policy.
• Users have global, socket, core-level control for process and threads arrangement.
• Possible to get great return with small investment by avoiding non-optimal core/memory policy.