NUMA Control for Hybrid Applications

Hang Liu & Xiao Zhu
TACC
June 19, 2014
**Parallel Paradigms**

- **OpenMP**
  - Run a bunch of threads in shared memory (spawned by a single `a.out`).

- **MPI**
  - Run a bunch of `a.out`'s as distributed memory paradigm.

- **Distributed and Shared Memory Parallel Paradigms in HPC**
  - **MPI**: addresses data movement in distributed memory (between processes-- executables)
  - **OpenMP**: addresses data access in shared memory (among threads in an executable)
Hybrid Program Model

- Start with **special** MPI initialization
- Create **OMP** parallel regions within **MPI** task (process).
  - Serial regions are the master thread or MPI task.
  - MPI rank is known to all threads
- Call MPI library in serial or parallel regions.
- Finalize MPI
Hybrid Applications

• Typical definition of hybrid application
  – Uses both message passing (MPI) and a form of shared memory algorithm (OpenMP), e.g., uses MPI task as a container for OpenMP threads
  – 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

Pure MPI

16 MPI Tasks

2 MPI Tasks
8 threads/task

1 MPI Tasks
16 threads/task

Master Thread of MPI Task
- MPI Task on Core
- Master Thread of MPI Task
- Slave Thread of MPI Task
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
**numactl Syntax**

- Affinity and Policy can be changed externally through **numactl** at the socket and core level.

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

---

**Stampede computing node**

- **Process affinity:** socket references and core references
- **Memory policy:** socket references

```
0,1,2,3, 8,9,10,11,
4,5,6,7, 12,13,14,15
```

- 0: `core core core core`
- 1: `core core core core`
- 1: `core core core core`
- 1: `core core core core`
# numactl Options on Stampede

<table>
<thead>
<tr>
<th>Category</th>
<th>cmd</th>
<th>option</th>
<th>arguments</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Socket Affinity</strong></td>
<td>numactl</td>
<td>-N --cpunodebind=</td>
<td>{0,1}</td>
<td>Only execute process on cores of this (these) socket(s).</td>
</tr>
<tr>
<td><strong>Memory Policy</strong></td>
<td>numactl</td>
<td>-l --localalloc</td>
<td>{no argument}</td>
<td>Allocate on current socket.</td>
</tr>
<tr>
<td><strong>Memory Policy</strong></td>
<td>numactl</td>
<td>-i --interleave=</td>
<td>{0,1}</td>
<td>Allocate round robin (interleave) on these sockets.</td>
</tr>
<tr>
<td><strong>Memory Policy</strong></td>
<td>numactl</td>
<td>--preferred=</td>
<td>{0,1}</td>
<td>Allocate on this socket; fallback to any other if full.</td>
</tr>
<tr>
<td><strong>Memory Policy</strong></td>
<td>numactl</td>
<td>-m --membind=</td>
<td>{0,1}</td>
<td>Only allocate on this (these) socket(s).</td>
</tr>
<tr>
<td><strong>Core Affinity</strong></td>
<td>numactl</td>
<td>--physcpubind=</td>
<td>{0,1,2,3,4,5,6, 7,8,9,10,11, 12,13,14,15}</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 16 threads

- Make sure 1 MPI task is created on each node
- Set number of OMP threads for each MPI task
- Can control only memory allocation

Number of MPI task on each node: \( n/N \)

**job script (Bourne shell)**

```bash
#SBATCH –N 6
#SBATCH –n 6

export OMP_NUM_THREADS=16

# Unset any MPI Affinities
export MV2_USE_AFFINITY=0
export MV2_ENABLE_AFFINITY=0
export VIADEV_USE_AFFINITY=0
export VIADEV_ENABLE_AFFINITY=0

ibrun numactl –i all ./a.out
```
Hybrid Batch Script  2 tasks, 8 threads/task

job script  (Bourne shell)
...
#SBATCH –N 6
#SBATCH –n 12
...
export OMP_NUM_THREADS=8

 numa.sh:  
#!/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
myrank=$(( $( ${PMI_RANK -0} + ${PMI_ID -0} +
${MPIRUN_RANK -0} + ${OMPI_COMM_WORLD_RANK -0} +
${OMPI_MCA_ns_nds_vpid -0} )))

localrank=$(( $myrank % 2 ))

socket=$localrank

exec numactl --cpunodebind $socket --membind $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.

```
job script (Bourne shell)
...
#SBATCH --N 6
#SBATCH --n 24
...
export OMP_NUM_THREADS=4
ibrunt tacc_affinity ./a.out
```
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.
#!/bin/bash
# -*- shell-script -*-

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} ))

# If running under "ibrun", TACC_pe_ppn will already be set
# else get info from SLURM_TASKS_PER_NODE
if [ -z "$TACC_pe_ppn" ]
  then
    myway=`echo $SLURM_TASKS_PER_NODE | awk -F ':' '{print $1}'`
  else
    myway=$TACC_pe_ppn
fi
local_rank=$(( $my_rank % $myway ))
EvenRanks=$(( $myway % 2 ))

if [ "$SYSHOST" = stampede ]; then
    # if 1 task/node, allow memory on both sockets
    if [ $myway -eq 1 ]; then
        numnode="0,1"
    # if 2 tasks/node, set 1st task on 0, 2nd on 1
    elif [ $myway -eq 2 ]; then
        numnode="$local_rank"
    # if even number of tasks per node, place memory on alternate chips
    elif [ $EvenRanks -eq 0 ]; then
        numnode=$(( $local_rank % 2 ))
    # if odd number of tasks per node, do nothing -- i.e., allow memory on both sockets
    else
        numnode="0,1"
    fi
fi

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