Process and thread affinity

Thread affinity allows software threads (e.g., OpenMP threads) to execute within the scope of specific processing resources. For example, when running two MPI tasks on a 2-socket node, one may want to bind all threads from task 0 to the first socket and the threads from task 1 to the second socket. It is also possible to bind threads at finer granularities such as cores (instead of sockets) or even hardware threads.

There are several ways to bind processes and threads to processing units:

1. Using the portable hardware locality layer (hwloc).
2. Using the Linux sched_setaffinity and sched_getaffinity API.
3. Using environment variables (OpenMP threads).

This document describes how to bind OpenMP threads to processing units (PUs) for Intel and GNU compilers. Before discussing threads, a simple understanding of process affinity in SLURM is necessary and is described in the next section.

All examples in this document use a TLCC2 node which consists of 2 sockets, each with 8 cores (total of 16 cores). The first socket contains cores 0-7 and the second 8-15.

Process affinity in SLURM

By default SLURM uses the auto-affinity module to bind MPI tasks (processes) to PUs. The following examples show these mappings.

```
## 1 task per node
## Task 0 executes on the entire node
$ srun -N1 -n1 -ppdebug myprog
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## 2 tasks per node
## Task 0 executes on socket 0, task 1 executes on socket 1
$ srun -N1 -n2 -ppdebug myprog
physcpubind: 0 1 2 3 4 5 6 7
physcpubind: 8 9 10 11 12 13 14 15

## 4 tasks per node
## Task 0 executes on cores 0-3, task 1 on 4-7, task 2 on 8-11, and task 3 on 12-15
$ srun -N1 -n4 -ppdebug myprog
physcpubind: 0 1 2 3
physcpubind: 4 5 6 7
physcpubind: 8 9 10 11
physcpubind: 12 13 14 15

## 8 tasks per node
## Each task executes exclusively on two cores
$ srun -N1 -n8 -ppdebug myprog
physcpubind: 0 1
physcpubind: 2 3
physcpubind: 4 5
physcpubind: 6 7
physcpubind: 8 9
physcpubind: 10 11
physcpubind: 12 13
physcpubind: 14 15

## 16 tasks per node
## Task i executes on core i
$ srun -N1 -n16 -ppdebug myprog
physcpubind: 0
physcpubind: 1
physcpubind: 2
physcpubind: 3
physcpubind: 4
physcpubind: 5
physcpubind: 6
physcpubind: 7
physcpubind: 8
physcpubind: 9
physcpubind: 10
physcpubind: 11
physcpubind: 12
physcpubind: 13
physcpubind: 14
physcpubind: 15
```

If SLURM's auto-affinity does not meet a user's binding requirements, one can disable or customize affinity using the

`--auto-affinity` flag.

```
## Disable auto-affinity on a 4-task job
## Unlike the example above, every process can execute on any core (no affinity defined)
$ srun -N1 -n4 -ppdebug --auto-affinity=off myprog
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



## Get more information about using auto-affinity
$ srun --auto-affinity=help
auto-affinity: Automatically assign CPU affinity using best-guess defaults.

The default behavior attempts to accomodate multi-threaded apps by
assigning more than one CPU per task if the number of tasks running
on the node is evenly divisible into the number of CPUs. Otherwise,
CPU affinity is not enabled unless the cpus_per_task (cpt) option is
specified. The default behavior may be modified using the
--auto-affinity options listed below. Also, the srun(1) --cpu_bind option
is processed after auto-affinity, and thus may be used to override any
CPU affinity settings from this module.

Option Usage: --auto-affinity=[args...]

where args... is a comma separated list of one or more of the following
help >>> Display this message.
v(erbose) >>> Print CPU affinty list for each remote task

off >>> Disable automatic CPU affinity.

start=N >>> Start affinity assignment at CPU [N]. If assigning CPUs in reverse, start [N] CPUs from the
last CPU.
rev(erse) >>>Allocate last CPU first instead of starting with CPU0.
cpus_per_task=N >>> Allocate [N] CPUs to each task.
cpt=N >>> Shorthand for cpus_per_task.

The following options may be used to explicitly list the CPUs for each
task on a node.
cpus=LIST >>> Comma-separated list of CPUs to allocate to tasks
masks=LIST >>> Comma-separated mask of CPUs to allocate to each task

Where the cpu and mask lists are of the same format documented in the
cpuset(4) manpage in the FORMATS section. Masks may be optionally followed
by a repeat count (e.g. 0xf0*2 == 0xf0,0xf0)

If one of the cpus= or masks= options is used, it must be the last option
specified, and any 'reverse' or 'start' option will be ignored
```

OpenMP thread affinity with Intel's icc

The Intel runtime library provides OpenMP thread affinity through the environment variable `KMP_AFFINITY`. For a detailed description of this variable please refer to Intel's online documentation. This section is limited to a few common usage examples.

The basic usage is very simple: `KMP_AFFINITY=<affinity_policy>`. For example, `KMP_AFFINITY=scatter` distributes and binds threads as evenly as possible among the available PUs while `KMP_AFFINITY=compact` distributes and binds threads as close to each other as possible.

Thread affinity can be used with or without process affinity. For example, a common scenario is to use SLURM's process affinity to bind an MPI task to a socket and then use thread affinity to bind threads to cores within that socket. The following examples demonstrate how to use thread affinity with and without SLURM's auto-affinity. First, SLURM's cpu-affinity is disabled to show the difference between the scatter and compact thread binding policies.

```
## The following two examples do not use SLURM's auto-affinity

## 1 task per node; 8 threads per task
## Disable SLURM's affinity (any task can run on any PU)
## 'scatter' binds every even threads to first socket and odd threads to second socket, i.e.,
## threads 0,2,4,6 execute on socket 0 - cores 0,1,2,3 respectively).
$ export OMP_NUM_THREADS=8
$ export KMP_AFFINITY=verbose,scatter
$ srun -N1 -n1 --auto-affinity=off -ppdebug ./omp_hello.icc.tlcc2
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {8}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {1}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {9}
OMP: Info #147: KMP_AFFINITY: Internal thread 4 bound to OS proc set {2}
OMP: Info #147: KMP_AFFINITY: Internal thread 5 bound to OS proc set {10}
OMP: Info #147: KMP_AFFINITY: Internal thread 6 bound to OS proc set {3}
OMP: Info #147: KMP_AFFINITY: Internal thread 7 bound to OS proc set {11}

## 1 task per node; 8 threads per task
## Disable SLRUM's affinity (any task can run on any PU)
## 'compact' binds threads to available PUs consecutively, i.e.,
## threads 0-7 execute on cores 0-7 respectively (all threads are running on first socket)
$ export OMP_NUM_THREADS=8
$ export KMP_AFFINITY=verbose,compact
$ srun -N1 -n1 --auto-affinity=off -ppdebug ./omp_hello.icc.tlcc2
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {1}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {2}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {3}
OMP: Info #147: KMP_AFFINITY: Internal thread 4 bound to OS proc set {4}
OMP: Info #147: KMP_AFFINITY: Internal thread 5 bound to OS proc set {5}
OMP: Info #147: KMP_AFFINITY: Internal thread 6 bound to OS proc set {6}
OMP: Info #147: KMP_AFFINITY: Internal thread 7 bound to OS proc set {7}
```

The following examples keep the number of MPI tasks and number of threads per task fixed at 2 and 4 respectively.

```
## No process affinity, no thread affinity
## Since there is no process or thread affinity, any thread can run on any core.
$ export OMP_NUM_THREADS=4
$ export KMP_AFFINITY=verbose,none
$ srun -N1 -n2 -ppdebug --auto-affinity=v,off ./omp_hello.icc.tlcc2
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}


## SLURM auto-affinity on, no thread affinity.
## Due to process affinity, task 0 is bound to cores 0-7 (socket 0) while task 1 is bound to cores 8-15
(socket 1).
## Even though the threads are not bound, they can only execute in the PUs associated with their parent
task, i.e.,
## threads from task 0 can execute on any core from socket 0 while threads from task 1 can execute on
any core from socket 1.
$ export OMP_NUM_THREADS=4
$ export KMP_AFFINITY=verbose,none
$ srun -N1 -n2 -ppdebug --auto-affinity=v ./omp_hello.icc.tlcc2
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0,1,2,3,4,5,6,7}
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {0,1,2,3,4,5,6,7}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {0,1,2,3,4,5,6,7}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {8,9,10,11,12,13,14,15}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {0,1,2,3,4,5,6,7}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {8,9,10,11,12,13,14,15}
auto-affinity: local task 0: CPUs: 0-7
auto-affinity: local task 1: CPUs: 8-15
```

```
## SLURM auto-affinity on, thread affinity scatter.
## Due to process affinity, task 0 is bound to socket 0 while task 1 is bound to socket 1.
## Due to thread affinity (scatter), threads 0,1,2,3 from task 0 are bound to cores 0,1,2,3 (in socket 0)
while threads 0,1,2,3 from task 1 are bound to cores 8,9,10,11 (in socket 1).
$ export OMP_NUM_THREADS=4
$ export KMP_AFFINITY=verbose,scatter
$ srun -N1 -n2 -ppdebug --auto-affinity=v ./omp_hello.icc.tlcc2
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {8}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {1}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {9}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {2}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {10}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {3}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {11}
auto-affinity: local task 0: CPUs: 0-7
auto-affinity: local task 1: CPUs: 8-15
```

OpenMP thread affinity with GNU's gcc

For those programs compiled with GNU's gcc, the environment variable `GOMP_CPU_AFFINITY` can be used to bind threads to PUs. This variable is not as flexible as Intel's `KMP_AFFINITY`; it requires a specific list of PU identifiers (as shown in `/proc/cpuinfo`) where threads should be bound to. As a consequence, every process (e.g., MPI task) needs to set this variable with the appropriate PU identifiers. Thus, when using more than one process per node, it is **not** recommended to use SLURM's auto-affinity (enabled by default) in tandem with GOMP_CPU_AFFINITY. If using a multi-process, multi-threaded program, consider (1) disabling SLURM's auto-affinity (`--auto-affinity=off`) and (2) setting `GOMP_CPU_AFFINITY` on a per process basis.

As shown below, this environment variable takes a list of PU identifiers separated by a comma or a space.

```
## Bind threads alternating sockets.
## Bind threads 0,2 to cores 0,1 (socket 0) and threads 1,3 to cores 8,9 (socket 1).
$ export OMP_NUM_THREADS=4
$ export GOMP_CPU_AFFINITY=0,8,1,9
$ srun -N1 -n1 -ppdebug ./omp_hello.gcc.tlcc2

## Bind threads to second socket.
## Bind threads 0,1,2,3 to cores 8,9,10,11 (in socket 1) respectively
$ export OMP_NUM_THREADS=4
$ export GOMP_CPU_AFFINITY=8-15
$ srun -N1 -n1 -ppdebug ./omp_hello.gcc.tlcc2
```

Todo

Add section for IBM compilers and associated flag: XLSMPOPTS