Using Intel OpenMP Thread Affinity for Pinning

The Intel compiler's OpenMP runtime library has the ability to bind OpenMP threads to physical processing units. Depending on the system topology, application, and operating system, thread affinity can have a dramatic effect on code performance. We recommend two approaches for using the Intel OpenMP thread affinity capability.

Using the KMP_AFFINITY Environment Variable

The thread affinity interface is controlled using the KMP_AFFINITY environment variable.

Syntax

For `csh` and `tcsh`:

```
setenv KMP_AFFINITY [<modifier>,...]<type>[,<permute>][,<offset>]
```

For `sh`, `bash`, and `ksh`:

```
export KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][,<offset>]
```

Using the Compiler Flag -par-affinity Compiler Option

Starting with the Intel compiler version 11.1, thread affinity can be specified through the compiler option `-par-affinity`. The use of `-openmp` or `-parallel` is required in order for this option to take effect. This option overrides the environment variable when both are specified. See `man ifort` for more information.

Note: Starting with `comp-intel/2015.0.090`, `-openmp` is deprecated and has been replaced with `-qopenmp`.

Syntax

```
-par-affinity=[<modifier>,...]<type>[,<permute>][,<offset>]
```

Possible Values for type

For both of the recommended approaches, the only required argument is `type`, which indicates the type of thread affinity to use. Descriptions of all of the possible arguments (`type`, `modifier`, `permute`, and `offset`) can be found on Intel's Thread Affinity Interface web page.

Recommendation: Use Intel compiler versions 11.1 and later, as some of the affinity methods described below are not supported in earlier versions.

Possible values for `type` are:

- `type = none` (default)
  
  Does not bind OpenMP threads to particular thread contexts; however, if the operating system supports affinity, the compiler still uses the OpenMP thread affinity interface to determine machine topology. Specify `KMP_AFFINITY=verbose,none` to list a machine topology map.

- `type = disabled`

Using Intel OpenMP Thread Affinity for Pinning
Specifying **disabled** completely disables the thread affinity interfaces. This forces the OpenMP runtime library to behave as if the affinity interface was not supported by the operating system. This includes implementations of the low-level API interfaces such as `kmp_set_affinity` and `kmp_get_affinity` that have no effect and will return a nonzero error code.

Additional information from Intel:

"The thread affinity type of KMP_AFFINITY environment variable defaults to none (`KMP_AFFINITY=none`). The behavior for `KMP_AFFINITY=none` was changed in 10.1.015 or later, and in all 11.x compilers, such that the initialization thread creates a "full mask" of all the threads on the machine, and every thread binds to this mask at startup time. It was subsequently found that this change may interfere with other platform affinity mechanism, for example, `dplace()` on SGI Altix machines. To resolve this issue, a new affinity type **disabled** was introduced in compiler 10.1.018, and in all 11.x compilers (`KMP_AFFINITY=disabled`). Setting `KMP_AFFINITY=disabled` will prevent the runtime library from making any affinity-related system calls."

**type = compact**

Specifying **compact** causes the threads to be placed as close together as possible. For example, in a topology map, the nearer a core is to the root, the more significance the core has when sorting the threads.

Usage example:

```sh
# for sh, ksh, bash
export KMP_AFFINITY=compact,verbose

# for csh, tcsh
setenv KMP_AFFINITY compact,verbose
```

**type = scatter**

Specifying **scatter** distributes the threads as evenly as possible across the entire system. Scatter is the opposite of compact.

Note: For most OpenMP codes, **type=scatter** should provide the best performance, as it minimizes cache and memory bandwidth contention for all processor models.

Usage example:

```sh
# for sh, ksh, bash
export KMP_AFFINITY=scatter,verbose

# for csh, tcsh
setenv KMP_AFFINITY scatter,verbose
```

**type = explicit**

Specifying **explicit** assigns OpenMP threads to a list of OS proc IDs that have been explicitly specified by using the `proclist=` modifier, which is required for this affinity type.

Usage example:

```sh
# for sh, ksh, bash
export KMP_AFFINITY="explicit,proclist=[0,1,4,5],verbose"

# for csh, tcsh
setenv KMP_AFFINITY "explicit,proclist=[0,1,4,5],verbose"
```
For nodes that support hyperthreading, you can use the `granularity` modifier to specify whether to pin OpenMP threads to physical cores using `granularity=core` (the default) or pin to logical cores using `granularity=fine` or `granularity=thread` for the compact and scatter types.

**Examples**

The following examples illustrate the thread placement of an OpenMP job with four threads on various platforms with different thread affinity methods. The variable OMP_NUM_THREADS is set to 4:

```
# for sh, ksh, bash
export OMP_NUM_THREADS=4

# for csh, tcsh
setenv OMP_NUM_THREADS 4
```

The use of the `verbose` modifier is recommended, as it provides an output with the placement.

**Sandy Bridge (Pleiades)**

As seen in the configuration diagram of a Sandy Bridge node, each set of eight physical cores in a socket share the same L3 cache.

Four threads running on 1 node (16 physical cores and 32 logical cores due to hyperthreading) of Sandy Bridge will get the following thread placement:

![kb285_sandybridge_table.png](kb285_sandybridge_table.png)

**Ivy Bridge (Pleiades)**

As seen in the configuration diagram of an Ivy Bridge node, each set of ten physical cores in a socket share the same L3 cache.

Four threads running on 1 node (20 physical cores and 40 logical cores due to hyperthreading) of Ivy Bridge will get the following thread placement:

![IvyBridgeTableThumb.png](IvyBridgeTableThumb.png)

**Haswell (Pleiades)**

As seen in the configuration diagram of a Haswell node, each set of 12 physical cores in a socket share the same L3 cache.

Four threads running on 1 node (24 physical cores and 48 logical cores due to hyperthreading) of Haswell will get the following thread placement:

![Using Intel OpenMP Thread Affinity for Pinning]( poignant )
Broadwell (Pleiades and Electra)

As seen in the configuration diagram of a Broadwell node, each set of 14 physical cores in a socket share the same L3 cache.

Four threads running on 1 node (28 physical cores and 56 logical cores due to hyperthreading) of Broadwell will get the following thread placement:

Merope

The six physical cores (indicated with same color) in each of Merope's Westmere processors share the same L3 cache.

Four threads running on one Westmere node (12 physical cores and 24 logical cores due to hyperthreading) will get the following thread placement:

Endeavour

Each Endeavour host has several cores. Based on the number of cores requested by the PBS job, a cpuset is created with the requested number of cores. Depending on availability, PBS may not be able to allocate consecutive cores to a job.

There are 8 cores per node (indicated with same color) on Endeavour. In the following example, 16 consecutive cores (cores 8-23) are allocated on Endeavour1.

Four threads running on 16 cores of Endeavour1 will get the following thread placement: