

# Using Intel OpenMP Thread Affinity for Pinning

## Category: Process Pinning

### Columbia Phase Out:

As of Feb. 8, 2013, the Columbia21 node has been taken offline as part of the [Columbia phase out process](#). Columbia22-24 are still available. If your script requires a specific node, please make the appropriate changes in order to ensure the success of your job.

**Summary:** The Intel compiler's OpenMP runtime library has the ability to bind OpenMP threads to physical processing units. Depending on the system (machine) topology, application, and operating system, thread affinity can have a dramatic effect on the code performance. For most OpenMP codes, **type=scatter** would provide the best performance, as it minimizes cache and memory bandwidth contention for Nehalem-EP, Westmere, and Sandy Bridge. For Harpertown, using an explicit **proclist** should give the best performance.

---

### Recommended Approaches

Two approaches are recommended for using the Intel OpenMP thread affinity capability:

#### Use the KMP\_AFFINITY Environment Variable

The thread affinity interface is controlled using the KMP\_AFFINITY environment variable.

#### Syntax

For **cs**h and **tc**sh:

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

For **sh**, **bash**, and **ksh**:

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

#### Use the Compiler Flag -par-affinity Compiler Option

Starting with the [Intel compiler](#) version 11.1, thread affinity can also 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.

## Syntax

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

For both of these approaches, **type** is the only required argument, and it indicates the type of thread affinity to use. Descriptions of the arguments (**type**, **modifier**, **permute**, and **offset**) can be found on Intel's [Thread Affinity Interface web page](#).

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

## Possible Values of type

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**

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, **dp1ace()** 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:

```
# 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.

Usage example:

```
# 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:

```
# 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 hyper-threading (such as Nehalem-EP, Westmere, and Sandy Br), you can use the **granularity** modifier to choose 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.

For most OpenMP codes, **type=scatter** should provide the best performance, as it minimizes cache and memory bandwidth contention for Nehalem-EP, Westmere, and Sandy Bridge nodes. For Harpertown nodes, using an explicit **proclist** should give the best performance.

## 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.

## Harpertown

Note that every two cores (indicated with same color) in Harpertown share L2 cache.

Four threads running on one node (eight physical cores) of Harpertown will get the following thread placement:

setting of KMP_AFFINITY	Processor id	0	2	4	6	1	3	5	7
compact,verbose	thread id	0	1	2	3				
scatter,verbose	thread id	0	2			1	3		
"explicit,proclist=[0,1,4,5],verbose"	thread id	0		2		1			3

## Nehalem-EP

Note that four physical cores (indicated with same color) in Nehalem-EP share the same L3 cache.

Four threads running on one node (eight physical cores and 16 logical cores due to hyper-threading) of Nehalem-EP will get the following thread placement:



There are two cores per node (indicated with same color, below) on Columbia21, while there are four cores per node on C22-24. In the following example, 8 consecutive cores (cores 4-11) are allocated on Columbia21.

Four threads running on 8 cores of Columbia21 will get the following thread placement:

setting of KMP_AFFINITY	Processor id	4	5	6	7	8	9	10	11
compact,verbose	thread id	0	1	2	3				
scatter,verbose	thread id	0		1		2		3	
"explicit,proclist=[5,7,9,11],verbose"	thread id		0		1		2		3

---

Article ID: 285

Last updated: 14 Feb, 2013

Computing at NAS -> Best Practices -> Process Pinning -> Using Intel OpenMP Thread Affinity for Pinning

<http://www.nas.nasa.gov/hecc/support/kb/entry/285/?ajax=1>