Process/Thread Pinning Overview

Pinning, the binding of a process or thread to a specific core, can improve the performance of your code by increasing the percentage of local memory accesses.

Once your code runs and produces correct results on a system, the next step is performance improvement. For a code that uses multiple cores, the placement of processes and/or threads can play a significant role in performance.

Given a set of processor cores in a PBS job, the Linux kernel usually does a reasonably good job of mapping processes/threads to physical cores, although the kernel may also migrate processes/threads. Some OpenMP runtime libraries and MPI libraries may also perform certain placements by default. In cases where the placements by the kernel or the MPI or OpenMP libraries are not optimal, you can try several methods to control the placement in order to improve performance of your code. Using the same placement from run to run also has the added benefit of reducing runtime variability.

Pay attention to maximizing data locality while minimizing latency and resource contention, and have a clear understanding of the characteristics of your own code and the machine that the code is running on.

Characteristics of NAS Systems

NAS provides two distinctly different types of systems: Pleiades, Aitken, Electra, and Merope are cluster systems, and Endeavour is a global shared-memory system. Each type is described in this section.

Pleiades, Aitken, Electra, and Merope

On Pleiades, Aitken, Electra, and Merope, memory on each node is accessible and shared only by the processes and threads running on that node. Pleiades is a cluster system consisting of different processor types: Sandy Bridge, Ivy Bridge, Haswell, and Broadwell. Merope is a cluster system that currently consists of Westmere nodes that have been repurposed from Pleiades. Electra is a cluster system that consists of Broadwell and Skylake nodes, and Aitken is a cluster system that consists of Cascade Lake nodes.

Each node contains two sockets, with a symmetric memory system inside each socket. These nodes are considered non-uniform memory access (NUMA) systems, and memory is accessed across the two sockets through the Quick Path Interconnect. So, for optimal performance, data locality should not be overlooked on these processor types.

However, compared to a global shared-memory NUMA system such as Endeavour, data locality is less of a concern on the cluster systems. Rather, minimizing latency and resource contention will be the main focus when pinning processes/threads on these systems.

For more information on Pleiades, Aitken, Electra, and Merope, see the following articles:

- Pleiades Configuration Details
- Aitken Configuration Details
- Electra Configuration Details
- Merope Configuration Details

Endeavour

Endeavour comprises two hosts (endeavour1 and endeavour2). Each host is a NUMA system that contains several dozen Sandy Bridge nodes, with memory located physically at various distances from the processors.
that access the data on memory. A process/thread can access the local memory on its node and the remote memory across nodes through the NUMAlink, with varying latencies. So, data locality is critical for achieving good performance on Endeavour.

Note: When developing an application, we recommend that you initialize data in parallel so that each processor core initializes the data it is likely to access later for calculation.

For more information, see Endeavour Configuration Details.

### Methods for Process/Thread Pinning

Several pinning approaches for OpenMP, MPI and MPI+OpenMP hybrid applications are listed below. We recommend using the Intel compiler (and its runtime library) and the SGI MPT software on NAS systems, so most of the approaches pertain specifically to them. You can also use the `mbind` tool for multiple OpenMP libraries and MPI environments.

**OpenMP codes**

- Using Intel OpenMP Thread Affinity for Pinning
- Using SGI's `dplace` Tool for Pinning
- Using SGI's `omplace` Tool for Pinning
- Using the `mbind` Tool for Pinning

**MPI codes**

- Setting SGI MPT Environment Variables
- Using SGI's `omplace` Tool for Pinning
- Using the `mbind` Tool for Pinning

**MPI+OpenMP hybrid codes**

- Using SGI's `omplace` Tool for Pinning
- Using the `mbind` Tool for Pinning

### Checking Process/Thread Placement

Each of the approaches listed above provides some verbose capability to print out the tool's placement results. In addition, you can check the placement using the following approaches.

**Use the ps Command**

```
ps -C executable_name -L -opsr,comm,time,pid,ppid,lwp
```

In the generated output, use the core ID under the PSR column, the process ID under the PID column, and the thread ID under the LWP column to find where the processes and/or threads are placed on the cores.
Note: The `ps` command provides a snapshot of the placement at that specific time. You may need to monitor the placement from time to time to make sure that the processes/threads do not migrate.

**Instrument your code to get placement information**

- Call the `mpi_get_processor_name` function to get the name of the processor an MPI process is running on
- Call the Linux C function `sched_getcpu()` to get the processor number that the process or thread is running on

For more information, see Instrumenting your Fortran Code to Check Process/Thread Placement.