Wikiprint Book

Title: Usage of TAMPI

Subject: DEEP - Public/User_Guide/TAMPI

Version: 12

Date: 19.04.2025 23:35:44

Table of Contents

Usage of TAMPI **3**
Quick Overview **3**
Quick Setup on DEEP System for a Hybrid Application **3**

Usage of TAMPI

Table of contents:

- Quick Overview
- Quick Setup on DEEP System for a Hybrid Application
- Using the Repositories
- Examples:
  - A Step-By-Step Detailed Guide to Execute the ???? Benchmark
  - Nbody Nenchmark
  - Heat Benchmark

Quick Overview

The **Task-Aware MPI** or TAMPI library ensures a **deadlock-free** execution of hybrid applications by implementing a cooperation mechanism between the MPI library and the parallel task-based runtime system.

TAMPI extends the functionality of standard MPI libraries by providing new mechanisms for improving the interoperability between parallel task-based programming models, such as **OpenMP** or **OmpSs-2**, and both **blocking** and **non-blocking** MPI operations.

Presently OpenMP programs (based on a derivative version of the LLVM OpenMP, yet to be released) can only make use of the non-blocking mode of TAMPI, whereas OmpSs-2 programs can leverage both blocking and non-blocking modes.

TAMPI is compatible with mainstream MPI implementations that support the **MPI_THREAD_MULTIPLE** threading level, which is the minimum requirement to provide its task-aware features.

Additional information about the TAMPI can be found at:

- OmpSs-2 repository. ?https://github.com/bsc-pm/tampi

Quick Setup on DEEP System for a Hybrid Application

We highly recommend to interactively log in a **cluster module (CM) node** to begin using TAMPI.

Presently, it seems that system affinity is not correctly setup for hybrid applications using multi-threading, therefore multi-threading will be ignored from now on.

A truly hybrid application should simply execute two MPI ranks, one on each NUMA socket to mitigate suboptimal memory accesses. Such an application will then use all the cores/threads available on each NUMA socket to run a shared-memory parallel application.

The command below requests an entire CM node for an interactive session with 2 MPI ranks (one per NUMA socket) and each rank using the 12 **physical** cores available on each socket (multi-threading disabled):

```
srun -p dp-cn -N 1 -n 2 -c 12 --pty /bin/bash -i
```

Once you have entered a CM node you can check the system affinity via the **NUMA command** `srun numactl --show`:

```
policy: bind
preferred node: 1
physcpubind: 12 13 14 15 16 17 18 19 20 21 22 23
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 0
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11
cpubind: 0
nodebind: 0
membind: 0
```

It can be readily seen that the each MPI process is bind to a different socket with no interleaving of processes or memory thus yielding optimal performance.

TAMPI has already been installed on DEEP and can be used by simply executing the following commands:

```
modulepath="/usr/local/software/skylake/Stages/2018b/modules/all/Core:$modulepath"

modulepath="/usr/local/software/skylake/Stages/2018b/modules/all/Compiler/mpi/intel/2019.0.117-GCC-7.3.0:$modulepath"


modulepath="/usr/local/software/skylake/Stages/2018b/modules/all/MPI/intel/2019.0.117-GCC-7.3.0/psmpi/5.2.1-1-mt:$modulepat


export MODULEPATH="$modulepath:$MODULEPATH"

module load TAMPI
```

Note that loading the TAMPI module will automatically load the **OmpSs-2** and **Parastation MPI** modules (notice that this MPI library has been compiled with multi-threading support enabled).

You might want to request more MPI ranks per socket depending on your particular application. See the examples below and the system affinity report (note that all of them ignore multi-threading):

```
srun -p dp-cn -N 1 -n 4 -c 6 --pty /bin/bash -i
```

```
$ srun numactl --show
policy: bind
preferred node: 0
physcpubind: 6 7 8 9 10 11
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 0
physcpubind: 0 1 2 3 4 5
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 1
physcpubind: 18 19 20 21 22 23
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 1
physcpubind: 12 13 14 15 16 17
cpubind: 1
nodebind: 1
membind: 1
```

```
srun -p dp-cn -N 1 -n 12 -c 2 --pty /bin/bash -i
```

```
$ srun numactl --show
policy: bind
preferred node: 0
physcpubind: 0 1
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 0
```

```
physcpubind: 4 5
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 0
physcpubind: 2 3
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 0
physcpubind: 8 9
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 0
physcpubind: 6 7
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 0
physcpubind: 10 11
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 1
physcpubind: 14 15
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 1
physcpubind: 16 17
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 1
physcpubind: 20 21
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 1
physcpubind: 18 19
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 1
physcpubind: 22 23
cpubind: 1
nodebind: 1
membind: 1
policy: bind
preferred node: 1
physcpubind: 12 13
```

```
cpubind: 1
nodebind: 1
membind: 1
```
