Wikiprint Book

Title: Usage of TAMPI

Subject: DEEP - Public/User_Guide/TAMPI

Version: 12

Date: 19.04.2025 23:37:32

Table of Contents

Usage of TAMPI **3**
Quick Overview **3**
Quick Setup on DEEP System for a Hybrid Application **3**
Nbody Benchmark (MPI+OmpSs-2+TAMPI) **6**
Description 6
Execution Instructions 7
References 7
Heat Benchmark (MPI+OmpSs-2+TAMPI) **7**
Description 7
Execution Instructions 7
References 7
Krist Benchmark (OmpSs-2+CUDA) **7**
Description 8
Execution Instructions 8
References 8

Usage of TAMPI

Table of contents:

- Quick Overview
- Quick Setup on DEEP System for a Hybrid Application
- Using the Repositories
- Examples:
  - 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 a 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.

In most cases a truly hybrid application should simply execute two MPI ranks, each one on a different NUMA socket in order 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 instance of the same binary.

The command below requests an entire CM node for an interactive session with 2 MPI ranks (1 MPI rank per NUMA socket) and each rank using the 12 **physical cores** available on each socket (i.e. multi-threading ignored):

```
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`:

```
$ srun -p dp-cn -N 1 -n 2 -c 12 --pty /bin/bash -i
$ 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 (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 together with the corresponding system affinity report:

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

```
$ 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
```

Finally, in case you would like to take advantage of multi-threading, we do recommend to run your application as standards jobs via the command `srun`. For example, requesting 1 CM node with 2 MPI ranks (1 MPI rank per socket) and 24 threads per socket (multi-threading) via a regular job submission to check the system affinity:

```
srun -p dp-cn -N 1 -n 2 -c 24 --ntasks-per-node=2 --ntasks-per-socket=1 numactl --show
```

yields the following system affinity:

```
$ srun -p dp-cn -N 1 -n 2 -c 24 --ntasks-per-node=2 --ntasks-per-socket=1 numactl --show
policy: bind
preferred node: 0
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 24 25 26 27 28 29 30 31 32 33 34 35
cpubind: 0
nodebind: 0
membind: 0
policy: bind
preferred node: 1
physcpubind: 12 13 14 15 16 17 18 19 20 21 22 23 36 37 38 39 40 41 42 43 44 45 46 47
cpubind: 1
nodebind: 1
membind: 1
```

which indicates that each MPI rank is binded to a single NUMA socket.

On the other hand, when allocating an interactive session for the same purpose:

```
srun -p dp-cn -N 1 -n 2 -c 24 --ntasks-per-node=2 --ntasks-per-socket=1 --pty /bin/bash -i
```

one realises that the binding somehow remains interleaved between the two NUMA sockets thus yielding **suboptimal performance**, which should be avoided:

```
$ srun -p dp-cn -N 1 -n 2 -c 24 --ntasks-per-node=2 --ntasks-per-socket=1 --pty /bin/bash -i
$ srun numactl --show
policy: bind
preferred node: 0
physcpubind: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
cpubind: 0 1
nodebind: 0 1
membind: 0 1
policy: bind
preferred node: 0
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

Nbody Benchmark (MPI+OmpSs-2+TAMPI)

Users must clone/download this example's repository from ?https://pm.bsc.es/gitlab/ompss-2/examples/nbody and transfer it to a DEEP working directory.

Description

This benchmark represents an N-body simulation to numerically approximate the evolution of a system of bodies in which each body continuously interacts with every other body. A familiar example is an astrophysical simulation in which each body represents a galaxy or an individual star, and the

bodies attract each other through the gravitational force.

There are **7 implementations** of this benchmark which are compiled in different binaries by executing the command `make`. These versions can be blocking, when the particle space is divided into smaller blocks, or non-blocking, when it is not.

The interoperability versions (MPI+OmpSs-2+TAMPI) are compiled only if the environment variable `TAMPI_HOME` is set to the Task-Aware MPI (TAMPI) library's installation directory.

Execution Instructions

The binaries accept several options. The most relevant options are the number of total particles (`-p`) and the number of timesteps (`-t`). More options can be seen with the `-h` option. An example of execution could be:

```
mpiexec -n 4 -bind-to hwthread:16 ./nbody -t 100 -p 8192
```

in which the application will perform 100 timesteps in 4 MPI processes with 16 hardware threads in each process (used by the OmpSs-2 runtime). The total number of particles will be 8192 so that each process will have 2048 particles (2 blocks per process).

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/nbody
- ?https://en.wikipedia.org/wiki/N-body_simulation

Heat Benchmark (MPI+OmpSs-2+TAMPI)

Users must clone/download this example's repository from ?https://pm.bsc.es/gitlab/ompss-2/examples/heat and transfer it to a DEEP working directory.

Description

This benchmark uses an iterative Gauss-Seidel method to solve the heat equation, which is a parabolic partial differential equation that describes the distribution of heat (or variation in temperature) in a given region over time. The heat equation is of fundamental importance in a wide range of science fields. In mathematics, it is the parabolic partial differential equation par excellence. In statistics, it is related to the study of the Brownian motion. Also, the diffusion equation is a generic version of the heat equation, and it is related to the study of chemical diffusion processes.

There are **9 implementations** of this benchmark which are compiled in different binaries by executing the command `make`.

The interoperability versions (MPI+OmpSs-2+TAMPI) are compiled only if the environment variable `TAMPI_HOME` is set to the Task-Aware MPI (TAMPI) library's installation directory.

Execution Instructions

The binaries accept several options. The most relevant options are the size of the matrix in each dimension (`-s`) and the number of timesteps (`-t`). More options can be seen with the `-h` option. An example of execution could be:

```
mpiexec -n 4 -bind-to hwthread:16 ./heat -t 150 -s 8192
```

in which the application will perform 150 timesteps in 4 MPI processes with 16 hardware threads in each process (used by the OmpSs-2 runtime). The size of the matrix in each dimension will be 8192 (8192^2 elements in total), this means that each process will have 2048x8192 elements (16 blocks per process).

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/heat
- ?https://pm.bsc.es/ftp/ompss-2/doc/examples/local/sphinx/04-mpi+ompss-2.html
- ?https://en.wikipedia.org/wiki/Heat_equation

Krist Benchmark (OmpSs-2+CUDA)

Users must clone/download this example's repository from ?https://pm.bsc.es/gitlab/ompss-2/examples/krist and transfer it to a DEEP working directory.

Description

This benchmark represents the krist kernel, which is used in crystallography to find the exact shape of a molecule using Rntgen diffraction on single crystals or powders.

There are **2 implementations** of this benchmark, *krist* and *krist-unified* using regular and unified CUDA memory, repectively.

Execution Instructions

```
./krist N_A N_R
```

where:

- `N_A` is the number of atoms (1000 by default).
- `N_R` is the umber of reflections (10000 by default).

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/krist