Wikiprint Book

Title: Programming with OmpSs?-2

Subject: DEEP - Public/User_Guide/OmpSs-2

Version: 53

Date: 20.05.2024 01:27:38

Table of Contents

Programming with OmpSs?-2

Table of contents:

Quick Overview

OmpSs?-2 is a programming model composed of a set of directives and library routines that can be used in conjunction with a high-level programming language (such as C, C++ or Fortran) in order to develop concurrent applications. Its name originally comes from two other programming models: **OpenMP** and **StarSs?**. The design principles of these two programming models constitute the fundamental ideas used to conceive the OmpSs? philosophy.

OmpSs?-2 **thread-pool** execution model differs from the **fork-join** parallelism implemented in OpenMP.

A **task** is the minimum execution entity that can be managed independently by the runtime scheduler. **Task dependences** let the user annotate the data flow of the program and are used to determine, at runtime, if the parallel execution of two tasks may cause data races.

The reference implementation of OmpSs?-2 is based on the **Mercurium** source-to-source compiler and the **Nanos6** runtime library:

- Mercurium source-to-source compiler provides the necessary support for transforming the high-level directives into a parallelized version of the application.
- Nanos6 runtime library provides services to manage all the parallelism in the user-application, including task creation, synchronization and data movement, as well as support for resource heterogeneity.

The reader is encouraged to visit the following links for additional information:

- OmpSs?-2 official website. ?https://pm.bsc.es/ompss-2
- OmpSs?-2 specification. ?https://pm.bsc.es/ftp/ompss-2/doc/spec
- OmpSs?-2 user guide. ?https://pm.bsc.es/ftp/ompss-2/doc/user-guide
- OmpSs?-2 examples and exercises. ?https://pm.bsc.es/ftp/ompss-2/doc/examples/index.html
- Mercurium official website. ?Link 1, ?Link 2
- Nanos official website. ?Link 1, ?Link 2

Quick Setup on DEEP System

We highly recommend to log in a **cluster module (CM)** node to begin using OmpSs?-2. To request an entire CM node interactively, please execute the following command:

```
srun --partition=dp-cn --nodes=1 --ntasks=48 --ntasks-per-socket=24 --ntasks-per-node=48 --pty /bin/bash -i
```

The command above is consistent with the actual hardware configuration of the cluster module with **hyper-threading enabled**. In this particular case, the command `srun --partition=dp-cn --nodes=1 --pty /bin/bash -i` would have yielded a similar request.

OmpSs?-2 has already been installed on DEEP and can be used by simply loading the following modules:

- `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:$module`
- `export MODULEPATH="$modulepath:$MODULEPATH"`
- `module load OmpSs-2`

Remember that OmpSs?-2 uses **thread-pool** execution model which means that it permanently **uses all the threads** present on the system. The reader check the **system affinity** by running the **NUMA command** `numactl --show`:

```
$ 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
```

as well as the **Nanos6 command** `nanos6-info --runtime-details | grep List`:

```
$ nanos6-info --runtime-details | grep List
Initial CPU List 0-11,24-35
NUMA Node 0 CPU List 0-35
NUMA Node 1 CPU List
```

Notice that both commands return consistent outputs and, even though an entire node with two sockets has been requested, only the first NUMA node (i.e. socket) has been correctly bind. As a result, only 48 threads of the first socket (0-11, 24-35), from which 24 are physical and 24 logical (hyper-threading enabled), are going to be utilised whilst the other 48 threads available on the second socket will remain idle. Therefore, **the system affinity showed above is not correct.**

System affinity can be used to specify, for example, the ratio of MPI and OmpSs?-2 processes for a hybrid application and can be modified by user request in different ways:

- Via SLURM: if the affinity does not correspond with the ressources requested like in the example above, then contact the system admin.
- Via the command `numactl`.
- Via the command `taskset`.

File Systems

On the DEEP-EST system, three different groups of filesystems are available:

- the ?JSC GPFS filesystems, provided via ?JUST and mounted on all JSC systems;
- the DEEP-EST (and SDV) parallel BeeGFS filesystems, available on all the nodes of the DEEP-EST system;
- the filesystems local to each node.

The users home folders are placed on the shared GPFS filesystems. With the advent of the new user model at JSC (?JUMO), the shared filesystems are structured as follows:

- $HOME: each JSC user has a folder under `/p/home/jusers/`, in which different home folders are available, one per system he/she has access to. These home folders have a low space quota and are reserved for configuration files, ssh keys, etc.
- $PROJECT: In JUMO, data and computational resources are assigned to projects: users can request access to a project and use the resources associated to it. As a consequence, each user has a folder within each of the projects he/she is part of. For the DEEP project, such folder is located under `/p/project/cdeep/`. Here is where the user should place data, and where the old files generated in the home folder before the JUMO transition can be found.

The DEEP-EST system doesn't mount the $SCRATCH and $ARCHIVE filesystems, as it is expected to provide similar functionalities with its own parallel filesystems.

The following table summarizes the characteristics of the file systems available in the DEEP-EST and DEEP-ER (SDV) systems:

Stripe Pattern Details

It is possible to query this information from the deep login node, for instance:

```
manzano@deep $ fhgfs-ctl --getentryinfo /work/manzano
Path: /manzano
Mount: /work
EntryID: 1D-53BA4FF8-3BD3
Metadata node: deep-fs02 [ID: 15315]
Stripe pattern details:
```

```
+ Type: RAID0
+ Chunksize: 512K
+ Number of storage targets: desired: 4

manzano@deep $ beegfs-ctl --getentryinfo /sdv-work/manzano
Path: /manzano
Mount: /sdv-work
EntryID: 0-565C499C-1
Metadata node: deeper-fs01 [ID: 1]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 512K
+ Number of storage targets: desired: 4
```

Or like this:

```
manzano@deep $ stat -f /work/manzano
 File: "/work/manzano"
   ID: 0        Namelen: 255     Type: fhgfs
Block size: 524288     Fundamental block size: 524288
Blocks: Total: 120178676  Free: 65045470   Available: 65045470
Inodes: Total: 0          Free: 0

manzano@deep $ stat -f /sdv-work/manzano
 File: "/sdv-work/manzano"
   ID: 0        Namelen: 255     Type: fhgfs
Block size: 524288     Fundamental block size: 524288
Blocks: Total: 120154793  Free: 110378947  Available: 110378947
Inodes: Total: 0          Free: 0
```

See ?http://www.beegfs.com/wiki/Striping for more information.

Additional infos

Detailed information on the **BeeGFS Configuration** can be found ?here.

Detailed information on the **BeeOND Configuration** can be found ?here.

Detailed information on the **Storage Configuration** can be found ?here.

Detailed information on the **Storage Performance** can be found ?here.

Notes

- The /work file system which is available in the DEEP-EST prototype, is as well reachable from the nodes in the SDV (including KNLs and KNMs) but through a slower connection of 1 Gig. The file system is therefore not suitable for benchmarking or I/O task intensive jobs from those nodes
- Performance tests (IOR and mdtest) reports are available in the BSCW under DEEP-ER → Work Packages (WPs) → WP4 → T4.5 - Performance measurement and evaluation of I/O software → Jülich DEEP Cluster → Benchmarking reports: ?https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/1382059
- Test results and parameters used stored in JUBE:

```
user@deep $ cd /usr/local/deep-er/sdv-benchmarks/synthetic/ior
user@deep $ jube2 result benchmarks

user@deep $ cd /usr/local/deep-er/sdv-benchmarks/synthetic/mdtest
user@deep $ jube2 result benchmarks
```