Wikiprint Book

Title: Programming with OmpSs?-2

Subject: DEEP - Public/User_Guide/OmpSs-2

Version: 53

Date: 20.05.2024 00:40:00

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.

Additional information about the OmpSs?-2 programming model can be found at:

- 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 repository. ?https://pm.bsc.es/gitlab/ompss-2/examples
- OmpSs?-2 manual with 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 for an interactive session, 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
```

Note that the command above is consistent with the actual hardware configuration of the cluster module with **hyper-threading enabled**.

OmpSs?-2 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:$modulepath"`
- `modulepath="/usr/local/software/skylake/Stages/2018b/modules/all/MPI/intel/2019.0.117-GCC-7.3.0/psmpi/5.2.1-1-mt:$module`
- `export MODULEPATH="$modulepath:$MODULEPATH"`
- `module load OmpSs-2`

Remember that OmpSs?-2 uses a **thread-pool** execution model which means that it **permanently uses all the threads** present on the system. Users are strongly encouraged to always 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 in the second socket will remain idle. Therefore, **the system affinity showed above is not valid since it does not represent the resources requested via SLURM.**

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. However, if the affinity does not correspond to the resources requested like in the previous example, it should be reported to the system administrators.
- Via the command `numactl`.
- Via the command `taskset`.

Repository with Examples

All the examples shown here are publicly available at ?https://pm.bsc.es/gitlab/ompss-2/examples. Users must clone/download each example's repository and then transfer it to a DEEP working directory.

System configuration

Please refer to section Quick Setup on DEEP System to get a functional version of OmpSs?-2 on DEEP. It is also recommended to run OmpSs?-2 on a cluster module (CM) node.

Building and running the examples

All the examples come with a Makefile already configured to build (e.g. `make`) and run (e.g. `make run`) them. You can clean the directory with the command `make clean`.

Controlling available threads

In order to limit or constraint the available threads for an application, the Unix **taskset** tool can be used to launch applications with a given thread affinity. In order to use taskset, simply precede the application's binary with taskset followed by a list of CPU IDs specifying the desired affinity:

```
taskset -c 0,2-4 ./application
```

The example above will run **application** with 4 cores: 0, 2, 3, 4.

Dependency graphs

Nanos6 allows for a graphical representation of data dependencies to be extracted. In order to generate said graph, run the application with the **NANOS6** environment variable set to **graph**:

`NANOS6=graph ./application`

By default graph nodes will include the full path of the source code. To remove these, set the following environment variable:

`NANOS6_GRAPH_SHORTEN_FILENAMES=1`

The result will be a PDF file with several pages, each representing the graph at a certain point in time. For best results, we suggest to display the PDF with **single page** view, showing a full page and to advance page by page.

Obtaining statistics

Another equally interesting feature of Nanos6 is obtaining statistics. To do so, simply run the application as:

`NANOS6=stats ./application` or also `NANOS6=stats-papi ./application`

The first collects timing statistics while the second also records hardware counters (compilation with PAPI is needed for the second). By default, the statistics are emitted standard error when the program ends.

Tracing with Extrae

A **trace.sh** file can be used to include all the environment variables needed to get an instrumentation trace of the execution. The content of this file is as follows:

```
#!/bin/bash
export EXTRAE_CONFIG_FILE=extrae.xml
export NANOS6="extrae"
$*
```

Additionally, you will need to change your running script in order to invoke the program through this trace.sh script. Although you can also edit your running script adding all the environment variables related with the instrumentation, it is preferable to use this extra script to easily change between instrumented and non-instrumented executions. When in need to instrument your execution, simply include trace.sh before the program invocation. Note that the **extrae.xml** file, which is used to configure the Extrae library to get a Paraver trace, is also needed.

---

multisaxpy benchmark (OmpSs?-2)

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

Description

This benchmark runs several SAXPY operations. SAXPY is a combination of scalar multiplication and vector addition (a common operation in computations with vector processors) and constitutes a level 1 operation in the Basic Linear Algebra Subprograms (BLAS) package.

There are **7 implementations** of this benchmark.

Execution instructions

`./multisaxpy SIZE BLOCK_SIZE INTERATIONS`

where:

- `SIZE` is the number of elements of the vectors used on the SAXPY operation.
- The SAXPY operation will be applied to the vector in blocks that contains `BLOCK_SIZE` elements.

- `ITERATIONS` is the number of times the SAXPY operation is executed.

Example output

```
$ make clean
rm -f 01.multisaxpy_seq 02.multisaxpy_task_loop 03.multisaxpy_task 04.multisaxpy_task+dep 05.multisaxpy_task+weakdep 06.mu

$ make
mcxx --ompss-2 01.multisaxpy_seq.cpp main.cpp -o 01.multisaxpy_seq -lrt
mcxx --ompss-2 02.multisaxpy_task_loop.cpp main.cpp -o 02.multisaxpy_task_loop -lrt
mcxx --ompss-2 03.multisaxpy_task.cpp main.cpp -o 03.multisaxpy_task -lrt
03.multisaxpy_task.cpp:3:13: info: adding task function 'axpy_task' for device 'smp'
03.multisaxpy_task.cpp:12:3: info: call to task function '::axpy_task'
03.multisaxpy_task.cpp:3:13: info: task function declared here
mcxx --ompss-2 04.multisaxpy_task+dep.cpp main.cpp -o 04.multisaxpy_task+dep -lrt
04.multisaxpy_task+dep.cpp:3:13: info: adding task function 'axpy_task' for device 'smp'
04.multisaxpy_task+dep.cpp:12:3: info: call to task function '::axpy_task'
04.multisaxpy_task+dep.cpp:3:13: info: task function declared here
mcxx --ompss-2 05.multisaxpy_task+weakdep.cpp main.cpp -o 05.multisaxpy_task+weakdep -lrt
05.multisaxpy_task+weakdep.cpp:3:13: info: adding task function 'axpy_task' for device 'smp'
05.multisaxpy_task+weakdep.cpp:12:3: info: call to task function '::axpy_task'
05.multisaxpy_task+weakdep.cpp:3:13: info: task function declared here
mcxx --ompss-2 06.multisaxpy_task_loop+weakdep.cpp main.cpp -o 06.multisaxpy_task_loop+weakdep -lrt
mcxx --ompss-2 07.multisaxpy_task+reduction.cpp main.cpp -o 07.multisaxpy_task+reduction -lrt
07.multisaxpy_task+reduction.cpp:14:13: info: reduction of variable 'yy' of type 'double [elements]' solved to 'operator +
<openmp-builtin-reductions>:1:1: info: reduction declared here
07.multisaxpy_task+reduction.cpp:21:13: info: reduction of variable 'y' of type 'double [N]' solved to 'operator +'
<openmp-builtin-reductions>:1:1: info: reduction declared here

$ make run
./01.multisaxpy_seq 16777216 8192 100
size: 16777216, bs: 8192, iterations: 100, time: 3.2982, performance: 0.508678
NANOS6_SCHEDULER=fifo ./02.multisaxpy_task_loop 16777216 8192 100
size: 16777216, bs: 8192, iterations: 100, time: 0.40835, performance: 4.10854
./03.multisaxpy_task 16777216 8192 100
size: 16777216, bs: 8192, iterations: 100, time: 0.646697, performance: 2.59429
./04.multisaxpy_task+dep 16777216 8192 100
size: 16777216, bs: 8192, iterations: 100, time: 1.00903, performance: 1.6627
./05.multisaxpy_task+weakdep 16777216 8192 100
size: 16777216, bs: 8192, iterations: 100, time: 1.17464, performance: 1.42829
NANOS6_SCHEDULER=fifo ./06.multisaxpy_task_loop+weakdep 16777216 8192 100
size: 16777216, bs: 8192, iterations: 100, time: 3.81836, performance: 0.439382
./07.multisaxpy_task+reduction 16777216 8192  100
size: 16777216, bs: 8192, iterations: 100, time: 4.26565, performance: 0.39331
```

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/multisaxpy
- ?https://pm.bsc.es/ftp/ompss-2/doc/examples/local/sphinx/03-fundamentals.html
- ?https://en.wikipedia.org/wiki/AXPY

---

dot-product benchmark (OmpSs?-2)

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

Description

This benchmark runs a dot-product operation. The dot-product (also known as scalar product) is an algebraic operation that takes two equal-length sequences of numbers and returns a single number.

There are **3 implementations** of this benchmark.

Execution instructions

```
./dot_product SIZE CHUNK_SIZE
```

where:

- `SIZE` is the number of elements of the vectors used on the dot-product operation.
- The dot-product operation will be applied to the vector in blocks that contains `CHUNK_SIZE` elements.

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/dot-product
- ?https://en.wikipedia.org/wiki/Dot_product

---

mergesort benchmark (OmpSs?-2)

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

Description

This benchmark is a recursive sorting algorithm based on comparisons.

There are **6 implementations** of this benchmark.

Execution instructions

```
./mergesort N BLOCK_SIZE
```

where:

- `N` is the number of elements to be sorted. Mandatory for all versions of this benchmark.
- `BLOCK_SIZE` is used to determine the threshold when the task becomes *final*. If the array size is less or equal than `BLOCK_SIZE`, the task will become final, so no more tasks will be created inside it. Mandatory for all versions of this benchmark.

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/mergesort
- ?https://en.wikipedia.org/wiki/Merge_sort

---

nqueens benchmark (OmpSs?-2)

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

Description

This benchmark computes, for a NxN chessboard, the number of configurations of placing N chess queens in the chessboard such that none of them is able to attack any other. It is implemented using a branch-and-bound algorithm.

There are **7 implementations** of this benchmark.

Execution instructions

```
./n-queens N [threshold]
```

where:

- `N` is the chessboard's size. Mandatory for all versions of this benchmark.
- `threshold` is the number of rows of the chessboard that will generate tasks.

The remaining rows (N - threshold) will not generate tasks and will be executed in serial mode. Mandatory from all versions of this benchmark except from 01 (sequential version) and 02 (fully parallel version).

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/nqueens
- ?https://en.wikipedia.org/wiki/Eight_queens_puzzle

---

matmul benchmark (OmpSs?-2)

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

Description

This benchmark runs a matrix multiplication operation C = A?B, where A has size N?M, B has size M?P, and the resulting matrix C has size N?P.

There are **3 implementations** of this benchmark.

Execution instructions

```
./matmul N M P BLOCK_SIZE
```

where:

- `N` is the number of rows of the matrix A.
- `M` is the number of columns of the matrix A and the number of rows of the matrix B.
- `P` is the number of columns of the matrix B.
- The matrix multiplication operation will be applied in blocks that contains `BLOCK_SIZE`?`BLOCK_SIZE` elements.

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/matmul
- ?https://pm.bsc.es/ftp/ompss-2/doc/examples/local/sphinx/02-examples.html
- ?https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

---

Cholesky benchmark (OmpSs?-2+MKL)

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

Description

This benchmark is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. This Cholesky decomposition is carried out with OmpSs?-2 using tasks with priorities.

There are **3 implementations** of this benchmark.

The code uses the CBLAS and LAPACKE interfaces to both BLAS and LAPACK. By default we try to find MKL, ATLAS and LAPACKE from the MKLROOT, LIBRARY_PATH and C_INCLUDE_PATH environment variables. If you are using an implementation with other linking requirements, please edit the `LIBS` entry in the makefile accordingly.

The Makefile has three additional rules:

- **run:** runs each version one after the other.
- **run-graph:** runs the OmpSs?-2 versions with the graph instrumentation.
- **run-extrae:** runs the OmpSs?-2 versions with the extrae instrumentation.

For the graph instrumentation, it is recommended to view the resulting PDF in single page mode and to advance through the pages. This will show the actual instantiation and execution of the code. For the extrae instrumentation, extrae must be loaded and available at least through the `LD_LIBRARY_PATH` environment variable.

Execution instructions

```
./cholesky SIZE BLOCK_SIZE
```

where:

- `SIZE` is the number of elements per side of the matrix.
- The decomposition is made by blocks of `BLOCK_SIZE` by `BLOCK_SIZE` elements.

References

- ?https://pm.bsc.es/gitlab/ompss-2/examples/cholesky
- ?https://pm.bsc.es/ftp/ompss-2/doc/examples/02-examples/cholesky-mkl/README.html
- ?https://en.wikipedia.org/wiki/Eight_queens_puzzle

---

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.

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://pm.bsc.es/ftp/ompss-2/doc/examples/local/sphinx/02-examples.html
- ?https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

---