NEC SX-Aurora Tsubasa: User Guide

Last revision: 2020-04-09 by M. Hermanowicz <m.hermanowicz@icm.edu.pl>

This document aims to provide basic information on how to use the NEC SX-Aurora Tsubasa system available at ICM UW computational facility. The contents herein are based on a number of documents, as referenced in the text, to provide a concise *quick start* guide and suggest further reading material for the ICM users.

Contents

1 Introduction

NEC SX-Aurora Tsubasa, announced in 2017, is a vector processor (vector engine, VE) belonging to the SX architecture line which has been developed by NEC Corporation since mid-1980s [1]. Unlike its stand-alone predecessors, Tsubasa has been designed as a PCIe card working within and being operated by an x86_64 host server (vector host, VH) running a distribution of the GNU/Linux operating system. The latter provides a complete software development environment for the connected VEs and runs Vector Engine Operating System (VEOS) which, in turn, serves as operating system to the VE programs. [2].

ICM UW provides its users with a NEC SX-Aurora Tsubasa installation as part of the Rysy cluster – see Table 1 – within the **ve** partition.

	Vector Host (VH)	Vector Engine (VE)
CPU Model	Intel Xeon Gold 6126	NEC SX-Aurora Tsubasa A300-8
CPU Cores	2×12	8×8
RAM [GB]	192	8×48

Table 1: NEC SX-Aurora Tsubasa installation at ICM UW: the **ve** partition of the Rysy cluster

2 Basic usage

To use the Tsubasa installation users must access the login node first at `hpc.icm.edu.pl` through SSH [3] and then establish a further connection to the Rysy cluster as in Listing 1. Alternatively, the `-J` command line option can be passed to the OpenSSH client to specify a jump host (here the `hpc` login node) through which the connection to Rysy will be established (issue `man ssh` command for details).

```
$ ssh username@hpc.icm.edu.pl
$ ssh rysy
```

Listing 1: Accessing NEC SX-Aurora Tsubasa installation at ICM UW

The system runs Slurm Workload Manager for job scheduling [4] and Environment Modules [5] to manage software. The single compute node (`PBaran`) of the **ve** partition can be used interactively – see Listing 2 – or as a batch job (see further below).

```
$ srun -A GRANT_ID -p ve --gres=ve:1 --pty bash -l
```

Listing 2: Running interactive Slurm session on Rysy/PBaran

Option	Description
`-c`	create object file
`-o`	output file name
`-I/path/to/include`	include header files
`-L/path/to/lib`	include libraries
`-g`	debugger symbols
`-Wall`	enable syntax warnings
`-Werror`	treat warnings as errors
`-O[0-4]`	optimisation levels
`-ftrace`	use the profiler
`-proginf`	enable execution analysis
`-report-all`	report diagnostics
`-traceback`	provides traceback information
`-fdiag-vector=[0-3]`	level of details for vector diagnostics

Table 2: Several basic options for the NEC compilers

Once the interactive shell session has started, the environmental variable `$VE_NODE_NUMBER` is being automatically set to control which VE card is to be used by the user programs. This variable can be read and set manually with `echo` [6] and `export` [7] commands, respectively. The software used to operate the VEs – including binaries, libraries, header files, etc. – is installed in `/opt/nec/ve` directory. Its effective use requires modification of the environmental variables [8], such as `$PATH`, `$LD_LIBRARY_PATH` and others, which can be done conveniently with the `source` command [9]:

```
$ source /opt/nec/ve/mpi/2.2.0/bin/necmpivars.sh
```

Listing 3: Sourcing VE environmental variables

Sourcing the variables (Listing 3) makes various VE tools accessible within the user environment. This includes the NEC compilers for `C`, `C++`, and `Fortran` languages that can be invoked by `ncc`, `nc++`, and `nfort`, respectively, or by their respective MPI wrappers: `mpincc`, `mpinc++`, and `mpinfort`. Please note that several compiler versions are currently installed and it might be necessary to include a version number in your command, e.g. `ncc-2.5.1`. The general usage is consistent with the GNU GCC: `<compiler> <options> <source file>`. Table 2 lists several standard options for the NEC compilers – see documentation for details. The last four of them, marked in red, are used for performance analysis and allow for efficient software development. Some of these, apart from being used as command line options at compile time, also rely on dedicated environmental variables that need to be set at runtime. For a full list of performance-related options, variables, as well as their output description, see PROGINF/FTRACE User's Guide [10] and the compiler-specific documentation [11, 12].

The binaries can be run directly by specifying the path or by using the VE loader program (`ve_exec`) – a few examples including parallel execution are gathered in Listing 4. For a full listing of options available for `mpirun` see the corresponding manual page [13] or issue `mpirun -h` command.

```
$ ./program
$ ve_exec ./program
$ mpirun ./program
$ mpirun -v -np 2 -ve 0-1 ./program # which enables the use of VE cards 0 and 1
```

Listing 4: Executing serial and parallel VE programs

Full documentation for SX-Aurora Tsubasa, its hardware and software components, is available at the NEC website [15]. An accessible introduction to using Tsubasa is also provided on the dedicated blog [16].

Another, non-interactive, mode of operation is a batch mode which requires a script to be submitted to Slurm. An example job script is shown in Listing 5. It specifies the name of the job (`-J`), requested number of nodes (`-N`), CPUs (`--ntasks-per-node`), memory (`-mem`; here in Megabytes), wall time limit (`--time`), grant ID (`-A`), partition (`-p`), generic resources (`--gres`), output file (`--output`), and the actual commands to be executed once the resources are granted. See Slurm documentation for an extensive list

of available options [14].

```
#!/bin/bash -l
#SBATCH -J name
#SBATCH -N 1
#SBATCH --ntasks-per-node 1
#SBATCH --mem 1000
#SBATCH --time=1:00:00
#SBATCH -A <Grant ID>
#SBATCH -p ve
#SBATCH --gres=ve:1
#SBATCH --output=out

./program
```

Listing 5: Example Slurm job script

Listing 6 provides a few basic example commands used to work with job scrips: submitting the job (`sbatch`) which returns the ID number assigned to the it by the queuing system, listing the user's jobs along with their status (`squeue`), listing the details of the specified job (`scontrol`), cancelling execution of the job (`scancel`). Consult the documentation for more [14].

```
$ sbatch job.sl # submits the job
$ squeue -u $USER # lists the user's current jobs
$ scontrol show job <ID> # lists the details of the job specified by given <ID>
$ scancel <ID> # cancels the job with given <ID>
```

Listing 6: Example Slurm commands

Since there's no dedicated filesystem to be used for calculations on the Rysy cluster, in contrast to other ICM systems, the jobs should be run from within the `$HOME` directory. The **ve** partition (PBaran compute node) is intended for jobs utilizing VE cards, and as such it should not be used for intensive CPU-consuming tasks.

3 SOL: Transparent Neural Network Acceleration

The SOL project aims at accelerating neural network tasks [17]. Working as middleware, it integrates with PyTorch, TensorFlow and MxNet, and supports NEC SX-Aurora Tsubasa as well as standard CPU and GPU architectures (x86, ARM64, NVIDIA) [18].

The software, including offline documentation, is located in the `/apps/nec/sol` directory. SOL is provided as a Python Wheel package (no system module available) and needs to be installed locally (via `pip`) by each user – see Listing 7. Consult the project website for use cases, papers, and presentations [17].

```
$ pip3 install --user /apps/nec/sol/sol-0.1.8-py3-none-any.whl # installs SOL
$ pip3 install --user torch torchvision numpy # installs other (example) requirements
```

Listing 7: SOL installation

References

[1] NEC SX Vector Supercomputer
`https://www.nec.com/en/global/solutions/hpc/sx/index.html`

[2] VEOS: Vector Engine Operating System
`https://github.com/veos-sxarr-NEC/veos`

[3] SSH: Secure Shell
`https://en.wikipedia.org/wiki/Secure_Shell`

[4] Slurm Workload Manager
https://slurm.schedmd.com/overview.html

[5] Environment Modules
https://modules.readthedocs.io/en/latest

[6] `echo` (command)
https://en.wikipedia.org/wiki/Echo_(command)

[7] `export` command
https://ss64.com/bash/export.html

[8] Environment variable
https://en.wikipedia.org/wiki/Environment_variable

[9] `source` command
https://ss64.com/bash/source.html

[10] PROGINF/FTRACE User's Guide
https://www.hpc.nec/documents/sdk/pdfs/g2at03e-PROGINF_FTRACE_User_Guide_en.pdf

[11] NEC C/C++ Compiler User's Guide
https://www.hpc.nec/documents/sdk/pdfs/g2af01e-C++UsersGuide-016.pdf

[12] NEC Fortran Compiler User's Guide
https://www.hpc.nec/documents/sdk/pdfs/g2af02e-FortranUsersGuide-016.pdf

[13] `mpirun` command
https://www.open-mpi.org/doc/v4.0/man1/mpirun.1.php

[14] Slurm Workload Manager: Documentation
https://slurm.schedmd.com/documentation.html

[15] NEC SX-Aurora Tsubasa Documentation
https://www.hpc.nec/documents/

[16] NEC Blog: First Steps with the SX-Aurora Tsubasa vector engine
https://sx-aurora.github.io/posts/VE-first-steps

[17] SOL: Transparent Neural Network Acceleration
http://sysml.neclab.eu/projects/sol

[18] SOL: Talks/Publications
http://sysml.neclab.eu/projects/sol/talks