Orchestrating a brighter world

NEC

VEOS Intro

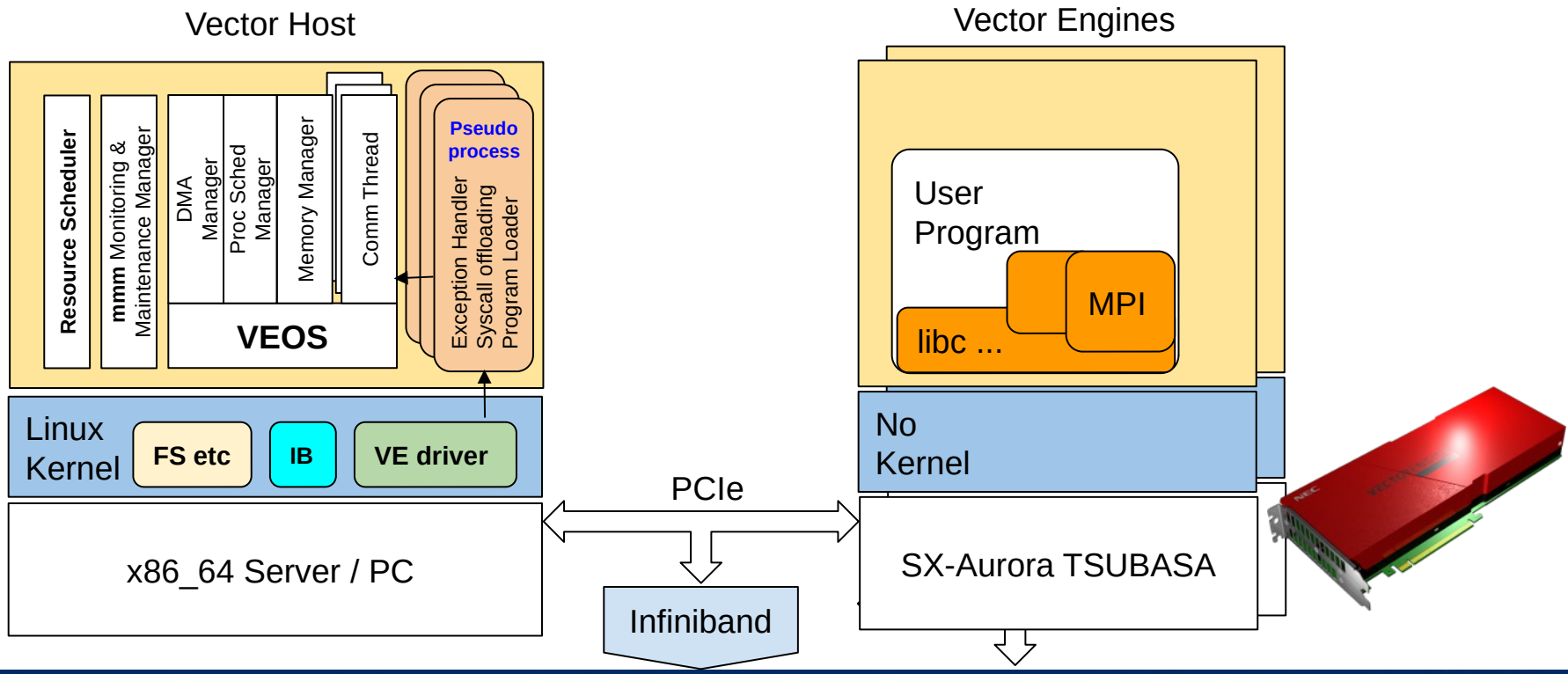
VE Deep Dive, RWTH Aachen, May 2-3 2019

Dr. Erich Focht, HPC Europe, NEC Deutschland GmbH

NEC Corporation

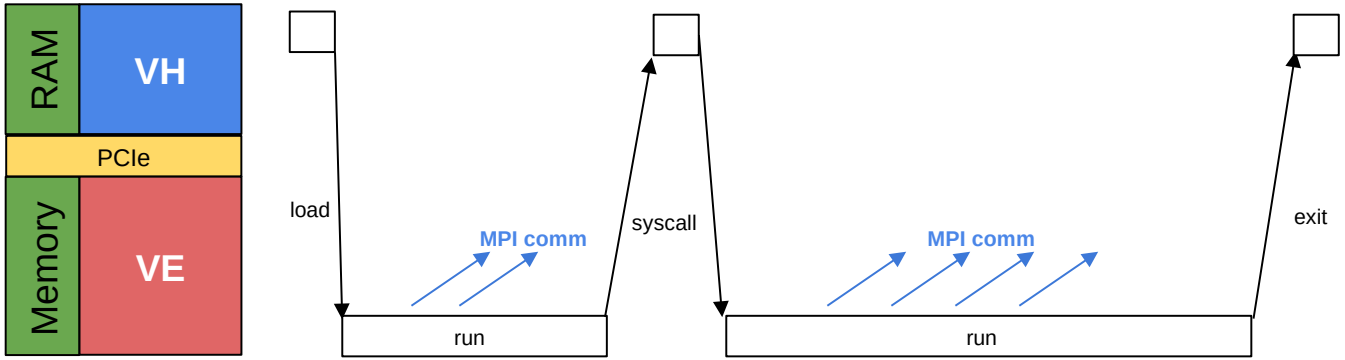
SX-Aurora System Software

VH + VE



Main Targeted Usage Model

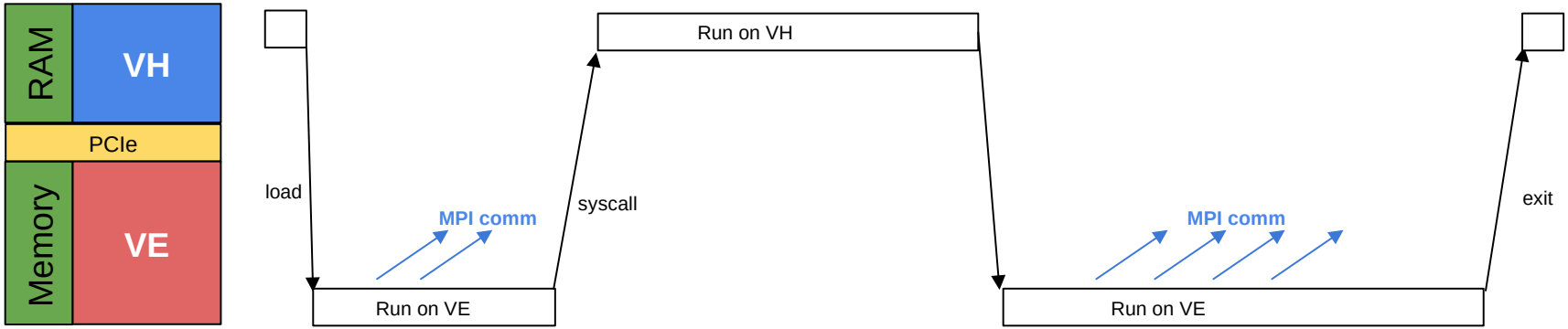
Native VE Program



- Compile for VE
  - Fortran, C, C++, (project: LLVM)
  - Auto-vectorize, “easy”
- Run on VE
  - Rare syscalls
  - Rare or no context switches
  - IO
  - OpenMP, MPI

Main Targeted Usage Model

Native VE Program with VH offloading

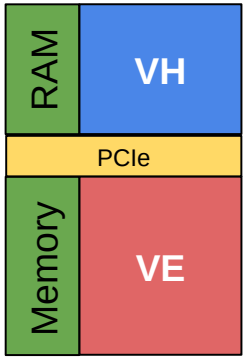


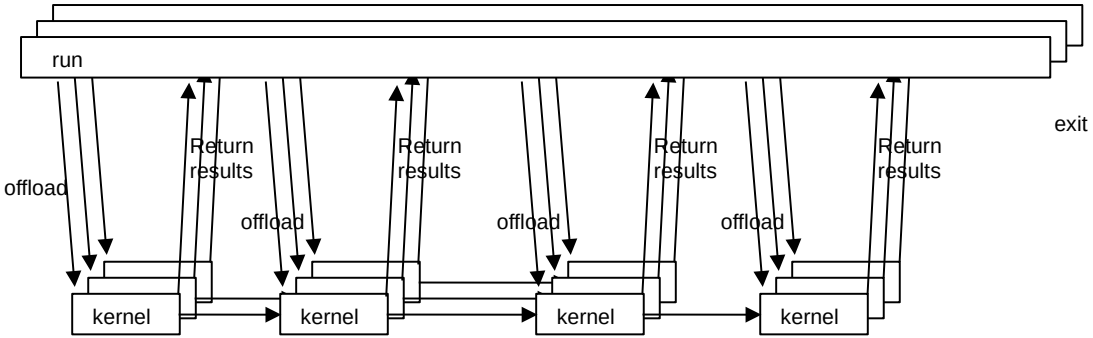
- Compile for VE
  - Fortran, C, C++, (project: LLVM)
  - Auto-vectorize, “easy”
- Run on VE
  - Rare syscalls
  - Rare or no context switches
  - IO
  - OpenMP, MPI

- Special syscall: ve_sysve (316)
- **VHcall**
  - Load dynamic library on VH
  - Call function inside dynamic library, run it on VH!
  - Transfer data from/to VE
  - A little peculiar API (will be improved soon)
  - Synchronous!
  - *Optimal for IO, pre- and post-processing*

VE as an Accelerator: VEO

Hybrid VH - VE Program





- **Main program runs on VH**
  - Multi-threaded, or interactive (!)
  - Any language (in principle...)
  - Uses x86_64 capabilities
  - Any MPI, any interconnect
- **VE Offloaded kernels**
  - Single thread or OpenMP
  - Simpler than full program

- **VE Offloaded kernels**
  - No MPI, no significant IO, no networking
- **Yes, PCIe is a bottleneck**
  - Keep data on VE, if possible
- **Many accelerated applications out there!**
- ***Let the VE do what it's good at!***
- ***And the VH too!***

Run Program on both: VE and VH

Compile VE program with *mpicc*

- Eg. built binary xyz.ve

Compile VH program with *mpicc -vh*

- Eg. built binary named xyz.vh

Run the program on VEs and VHs

```
mpiexec –node 0 –np 8 –ve 0 xyz.ve : -node 0 –np 1 –vh xyz.vh : -node 3 –np 2 –vh xyz.vh
```

Be careful with MPI Types

- Data layout and alignment is different!

… beta …

This and That

Little helpers...

- Scheduler tuning
- Log4crc
- strace
- Veperf
- Veprof

VE Scheduler Tuning

VEOS Daemon Launcher Options

- **`/etc/opt/nec/ve/veos/ve-os-launcher.d`**
- Root can change options in file **`veos_timer.options`**
- Restart veos!

```
# systemctl restart ve-os-launcher@*.service
```

When to use it?

- Multithreaded program starts slowly
- Fork() : schedule child first
- If child doesn’t yield CPU immediately, it spends at least 1s on it
- More threads than cores

```
#The interval between invocation of the timer handler of the scheduler, in milliseconds.
#The period of time for which a thred of a process is allowed to run, in milliseconds.
#ve-os-launcher@*=--timer-interval=100 –time-slice=1000
ve-os-launcher@*=--timer-interval=10 --time-slice=20
```

log4crc

VEOS logging facility

- Logging for VEOS daemons: **`/var/opt/nec/ve/veos`**
- Root can change log levels in **/etc/opt/nec/ve/veos/log4crc**

Debug Pseudo process, VEO

- Put a log4crc file in your current working directory. Edit log levels!
- Run your VE program → Look at the logs
- Sample log4crc : copy it from **/etc/opt/nec/ve/veos/log4crc**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE log4c SYSTEM "">

<log4c>
        <config>
                <bufsize>0</bufsize>
                <debug level="0"/>
                <nocleanup>0</nocleanup>
        </config>

        <category name="veos.pseudo_process" priority="INFO" appender="ve_exec_appender" />
        <appender name="ve_exec_appender" layout="ve" type="stream" />

        <category name="veos.vemmd" priority="INFO" appender="vemmd_appender" />
        <rollingpolicy name="veos_command_rp" type="sizewin" maxsize="4194304" maxnum="10" />
```

strace

VEOS version of strace

- Like the normal linux version…
- Trace of VE side system calls
- With all normal options (time stamps, etc...)
- Can attach to existing VE process
- Could do more with this… like generate histogram of syscalls or read/write sizes

Others...

py-veosinfo

- E. Focht
- Python bindings to libveosinfo
- veperf
- https://github.com/SX-Aurora/py-veosinfo

```
# veperf -d 2 -n 0
pid          USRSEC  EFFTIME     MOPS    MFLOPS  AVGVL    VOPRAT  VTIMERATIO  L1CMISS  PORTCONF   VLLCHIT  comm
-> VE0
376756       52.23s    1.000   276641    267393    255     99.4%       100.0%       0%         0%      100%  ./p
376799       50.43s    1.000   276629    267382    255     99.4%       100.0%       0%         0%      100%  ./p
376872       47.23s    1.000   416945    532519    255     99.4%       100.0%       0%         0%      100%  ./sgemm
376964        1.40s    0.017     1126         0      0      0.0%         0.0%      32%         0%        0%  ../bin/mpc_ve
377026       39.72s    0.997    64282     44803    256     99.8%       100.0%       0%         1%      100%  ./vector.sx-mpi
SUM VE0: MOPS=1035623  MFLOPS=1112096
pid          USRSEC  EFFTIME     MOPS    MFLOPS  AVGVL    VOPRAT  VTIMERATIO  L1CMISS  PORTCONF   VLLCHIT  comm
-> VE0
376756       54.23s    1.000   276324    267087    255     99.4%       100.0%       0%         0%      100%  ./p
376799       52.43s    1.000   276325    267088    255     99.4%       100.0%       0%         0%      100%  ./p
376872       49.24s    1.000   416555    532021    255     99.4%       100.0%       0%         0%      100%  ./sgemm
376964        2.01s    0.308     4311      1296    222     94.1%        20.1%      69%         1%      100%  ../bin/mpc_ve
377026       41.72s    1.000    61982     43218    256     99.8%       100.0%       0%         1%      100%  ./vector.sx-mpi
SUM VE0: MOPS=1035497  MFLOPS=1110709
pid          USRSEC  EFFTIME     MOPS    MFLOPS  AVGVL    VOPRAT  VTIMERATIO  L1CMISS  PORTCONF   VLLCHIT  comm
-> VE0
376756       56.23s    1.000   275573    266361    255     99.4%       100.0%       0%         0%      100%  ./p
376799       54.43s    1.000   275574    266361    255     99.4%       100.0%       0%         0%      100%  ./p
376872       51.24s    1.000   415472    530636    255     99.4%       100.0%       0%         0%      100%  ./sgemm
376964        4.01s    1.000     4380      1318    222     95.3%        20.8%      71%         2%      100%  ../bin/mpc_ve
377026       43.72s    1.000    56542     39425    256     99.8%       100.0%       0%         1%      100%  ./vector.sx-mpi
SUM VE0: MOPS=1027540  MFLOPS=1104101
```

Others...

py-veosinfo

- E. Focht
- Python b
- veperf
- https://gi

```
/**
 * @brief This function will be used to communicate with VEOS and get the
 * register values of a process according to the IDs in regid.
 *
 * @param nodeid[in] VE node number
 * @param pid[in] PID of VE process
 * @param numregs[in] number of registers to retrieve
 * @param regid[in] int array with register indices to retrieve
 * @param regval[out] uint64_t array filled with retrieved register values
 *
 * @return 0 on success and -1 on failure
 */
int ve_get_regvals(int nodeid, pid_t pid, int numregs,
                   int *regid, uint64_t *regval);
```

```
# veperf -
pid
-> VE0
376756
376799
376872
376964
377026
SUM VE0: M
pid
-> VE0
376756
376799
376872
376964
377026
SUM VE0: MOPS=1035497  MFLOPS=1110709
pid          USRSEC  EFFTIME      MOPS     MFLOPS  AVGVL    VOPRAT  VTIMERATIO  L1CMISS  PORTCONF   VLLCHIT  comm
-> VE0
376756       56.23s    1.000    275573     266361    255     99.4%       100.0%       0%        0%      100%  ./p
376799       54.43s    1.000    275574     266361    255     99.4%       100.0%       0%        0%      100%  ./p
376872       51.24s    1.000    415472     530636    255     99.4%       100.0%       0%        0%      100%  ./sgemm
376964        4.01s    1.000      4380       1318    222     95.3%        20.8%      71%        2%      100%  ../bin/mpc_ve
377026       43.72s    1.000     56542      39425    256     99.8%       100.0%       0%        1%      100%  ./vector.sx-mpi
SUM VE0: MOPS=1027540  MFLOPS=1104101
```

Others...

veprof

- H. Berger
- External profiler for VE programs
- Running on VH
- Low overhead, 100Hz or more
- Samples performance counters, too
- (limited value for very short functions)
- https://github.com/SX-Aurora/veprof

```
FUNCTION                 SAMPLES    TIME     TIME VECTIME  VTIME VARITHM    VLOAD    VOP MFLOPS    MOPS     AVG    L1$ PRTCNF  LLC$E
                               %       %      [s]     [s]      %     [s]      [s]      %                    VLEN  MISS%    [s]   HIT%
main                       82.86   83.13   192.82    0.35   0.18    0.20     0.15   5.15    291    2230 176.62   15.76    0.00 100.00
__fast_sincos               8.62    8.61    19.98    0.00   0.00    0.00     0.00   0.00    365    2066   0.00    0.31    0.00   0.00
rotate(double&  double&  double)
                            3.18    3.20     7.42    0.00   0.00    0.00     0.00   0.00    365    2067   0.00    0.11    0.00   0.00
occupancy(double  double  double  double)
                            2.16    2.12     4.93    0.00   0.02    0.00     0.00   0.55    366    2063 180.40    0.12    0.00 100.00
__fast_sqrt                 2.24    2.07     4.81    0.00   0.07    0.00     0.00   1.98    360    2066 175.20    0.20    0.00 100.00
length(double  double)
                            0.75    0.76     1.75    0.00   0.00    0.00     0.00   0.00    359    2079   0.00    0.03    0.00   0.00
_init                       0.17    0.10     0.23    0.00   0.61    0.00     0.00  18.03    272    2177 180.93    0.06    0.00 100.00
triangleOccupancy(double  double  double)
                            0.01    0.01     0.02    0.00   0.00    0.00     0.00   0.00    378    2032   0.00    0.00    0.00   0.00
```

News: VEOS Feature

VH-VE-Shared Memory

- Allocate SysV shared memory segment on VH
- Find segment on VE
- Register it in DMAATB → get VEHVA address of VH buffer
- VE read/write from/to VH shared mem segment with LHM/SHM instructions

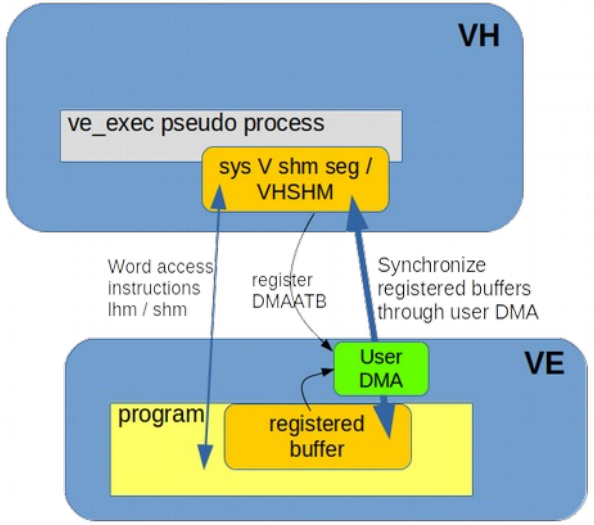
- Allocate buffer on VE (same size as VH sysV shm seg
- Register buffer with DMAATB → get VEHVA address of VE buffer
- Program DMA to transfer data between VE and VH buffers

- Works with User DMA Descriptors (2 per Core)
  - Used by IB, MPI, ScateFS
  - As opposed to privileged DMA descriptors (1 per CPU), used by VEOS
  - User descriptors: “registered” memory, i.e. VEHVA
  - Priv descriptors: physical memory, unregistered, needs translation

News: VEOS Feature

VE Accelerated IO

- Why needed?
- Normal IO:
  - VE Process
  - Pseudo process on VH
  - VEOS → system DMA descriptors
  - VE Driver Kernel Module
- New: user DMA descriptors!



VE Accelerated IO: HOWTO

```
sudo sysctl vm.nr_hugepages=$((256*16))
```

To make this setting permanent create for example a file in `/etc/sysctl.d`:

```
echo >/etc/sysctl.d/10-hugepages.conf <<EOF
#
# Set number of huge pages to 4096
#
vm.nr_hugepages = 4096
EOF
```

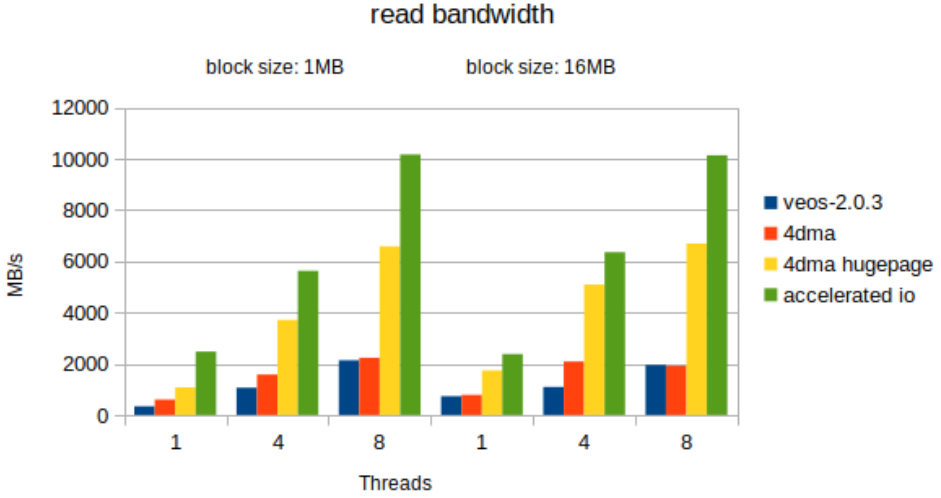
and reload the sysctl settings:

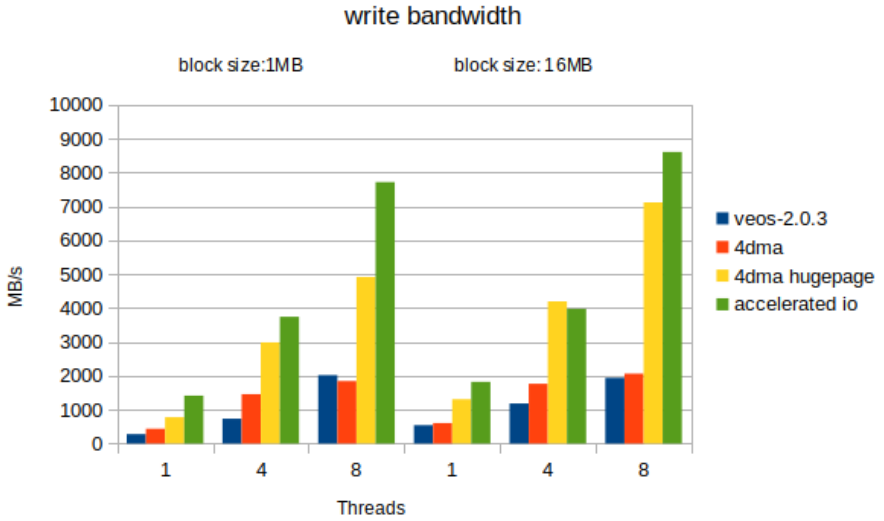
```
sudo sysctl --system
```

Now you're ready for VE IO acceleration. Before executing a VE program make the loader preload the accelerated IO library:

```
export VE_LD_PRELOAD=libveaccio.so.1
```

VE Accelerated IO: Performance





News: VEOS Feature

VE AIO: Async IO from VE

See https://veos-sxarr-nec.github.io/libsysve/md_doc_VEAIO.html

API:

- **ve_aio_init()** initializes and returns new context for VE AIO.
- **ve_aio_fini()** release the context for VE AIO.
- **ve_aio_read()** starts asynchronous read operation for VE.
- **ve_aio_write()** starts asynchronous write operation for VE.
- **ve_aio_query()** gets state of read/write operation.
- **ve_aio_wait()** waits and gets result of read/write request.

News: Tunings

DMA-bulk-hugepages

- **Patch by EF**
- **Significantly improves VEOS System DMA (privileged descriptors)**
  - ve_read_data(), ve_write_data(), used for syscalls, IO, VEO data transfer
  - Bulk virt to phys address computation
  - Async DMA descriptor generation and transfer
  - Performance without this patch: max. 1.5GB/s
- **With huge pages: up to 11GB/s VE-VH with big buffers**
- **With 4k pages: max. 5.5GB/s**
- **Included since veos 2.1.0**
- **Cheap “tuning”:**

```
hugectl --heap <your_ve_executable> ...
```

News: NUMA

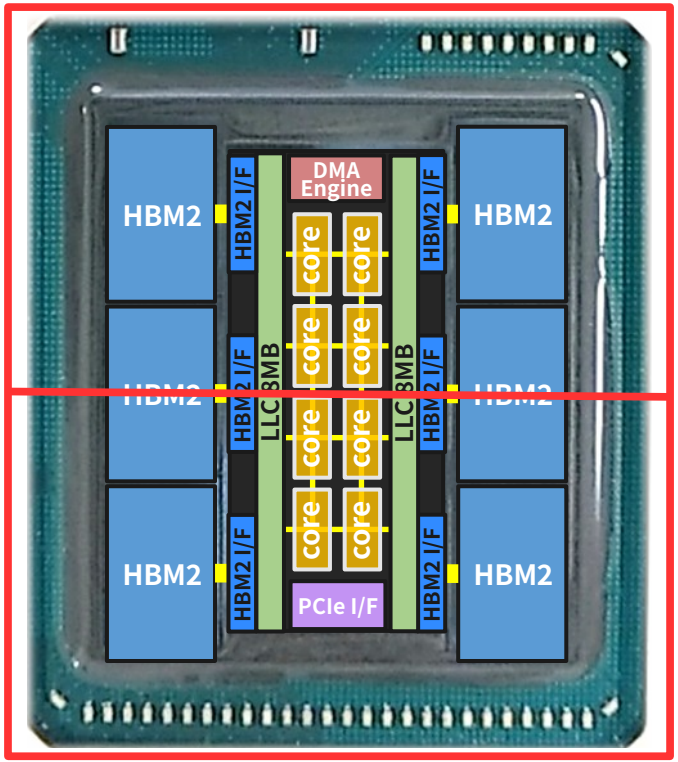
Split one VE “node” into two NUMA domains

- Reduce memory network conflicts
- Less unwanted evictions from shared LLC
- Check with

```
vecmd info
   Partitioning mode                : ON
       NUMA node0                   : 0-3
       NUMA node1                   : 0-3
```

- Scheduler: chooses node with least load
- Memory: on local node, if free space
- Enforce with env variable:

```
…
VE_NUMA_OPT="--cpunodebind=0 --localmembind"
```



News: NUMA

Split one VE “node” into two NUMA domains

Enable:

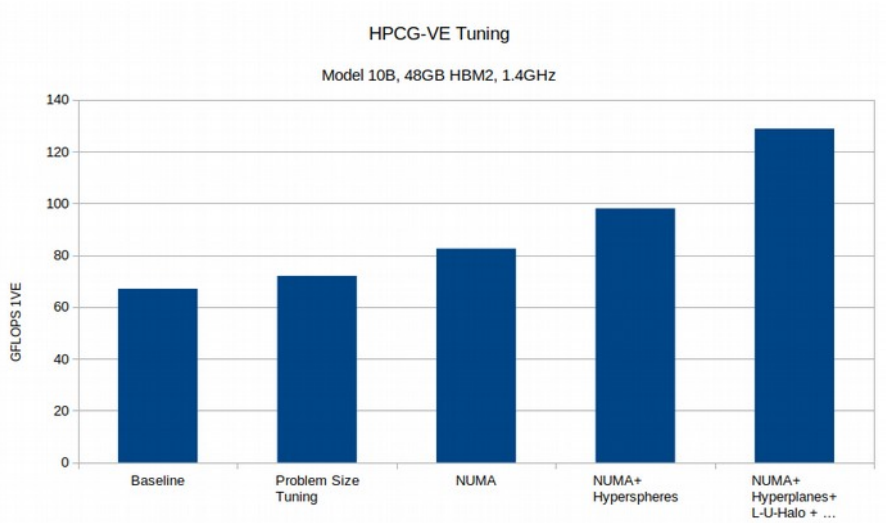
```
vecmd vconfig set partitioning_mode on
vecmd state set off
vecmd state set mnt
vecmd reset card
```

Disable:

```
vecmd vconfig set partitioning_mode off
vecmd state set off
vecmd state set mnt
vecmd reset card
```

Impact:

- 15% with HPCG (8 MPI processes)
- Can be more (~20%) with 2 OpenMP process



Documentation and Forum

Aurora Forum Site: www.hpc.nec

Documentation of all products is open

- https://www.hpc.nec/documents
- Links to SDK, MPI, VEOS, Scatefs, guides...
- Info on DMA is missing (ISA Guide)

Use the forum!

-

Check out blogs :-)

- https://sx-aurora.github.io
- https://sx-aurora-dev.github.io


Orchestrating a brighter world
NEC