\Orchestrating a brighter world NEC

SX-Aurora TSUBASA

Introduction of SX-Aurora TSUBASA and its programming models

All models are available now

Supercomputer Model
■For large scale configuration
■DLC with 40°C water

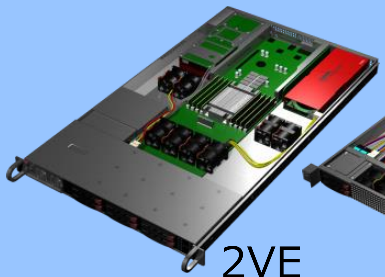
Rack Mount Model
■Flexible configuration
■Air Cooled

Tower Model
■For developer/programmer
■Tower implementation

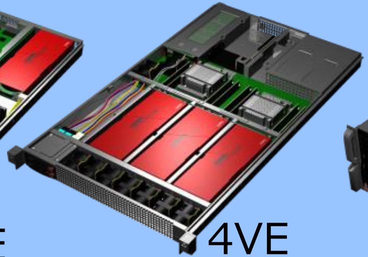
A500 series

64VE-

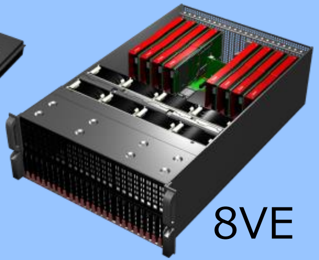
A300 series



2VE



4VE



8VE

A100 series

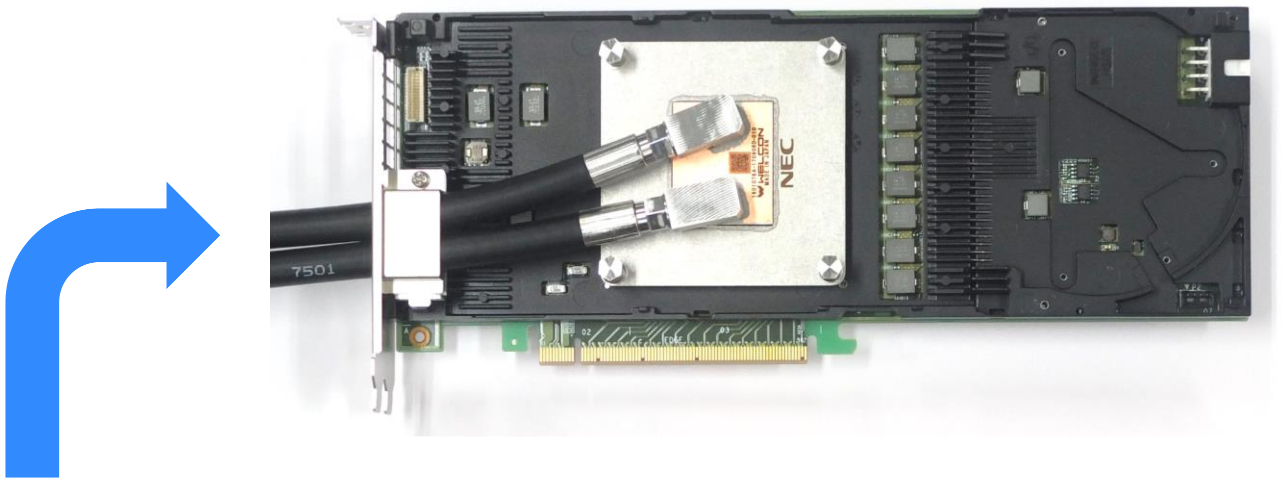
1VE

Supercomputer model becomes available

SX-Aurora TSUBASA

64 Vector Engines cooled by DLC are put into a rack

Direct liquid cooling



40°C water

The high end VE card is available by this DLC super computer

Design Concept

POINT 1 Memory Bandwidth

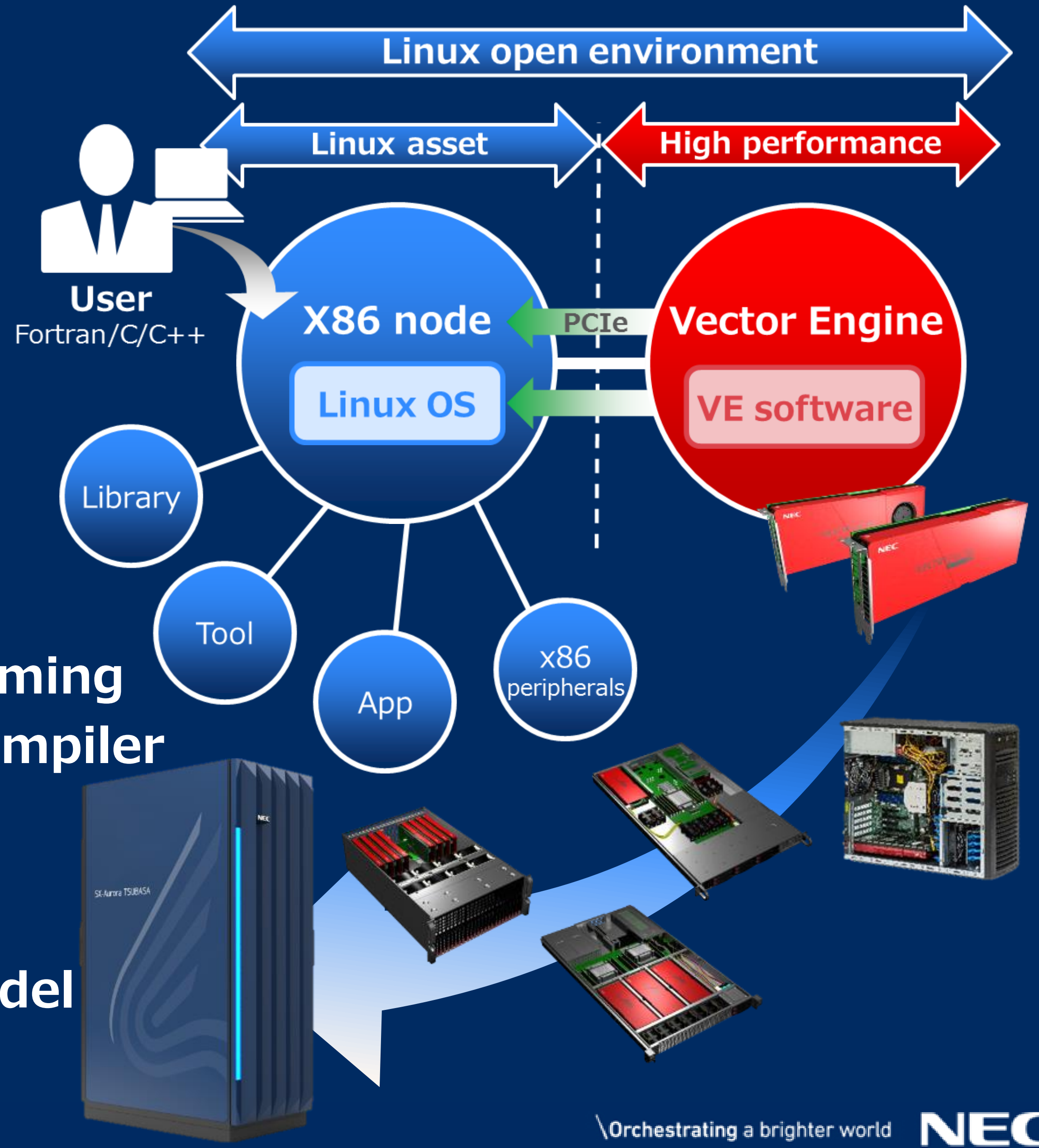
- ✓ 1.22TB/s per processor
- ✓ 78.64TB/s per rack

POINT 2 Easy to Use

- ✓ x86/Linux open environment
- ✓ Fortran/C/C++ standard programming
- ✓ Automatic vectorization by NEC compiler

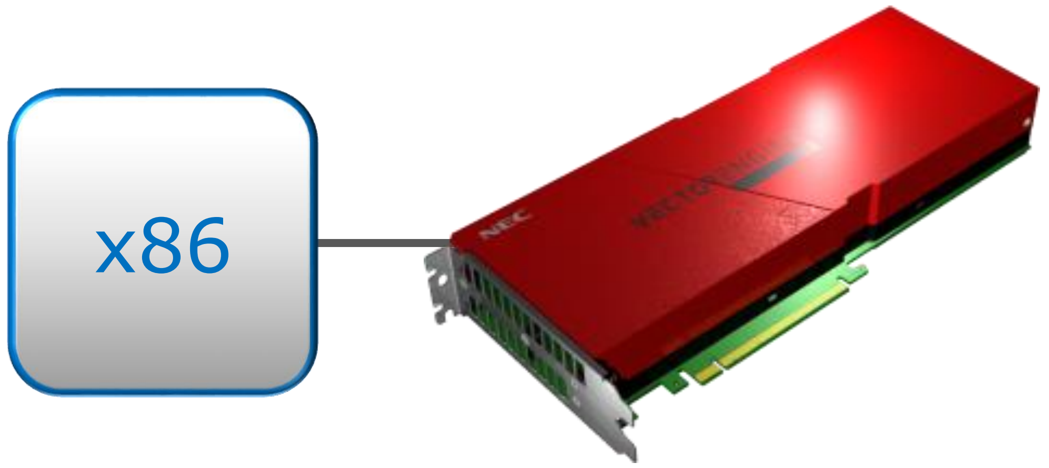
POINT 3 Flexible and Scalable

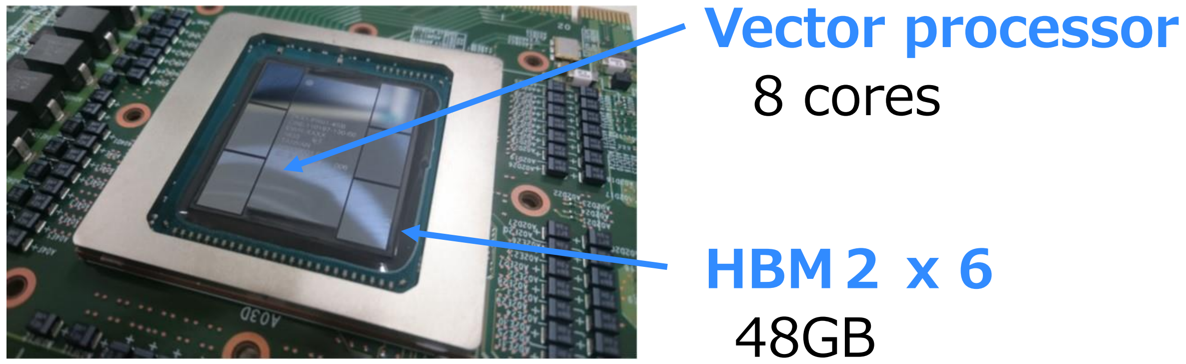
- ✓ From Tower to Supercomputer model
- ✓ InfiniBand interconnect for MPI

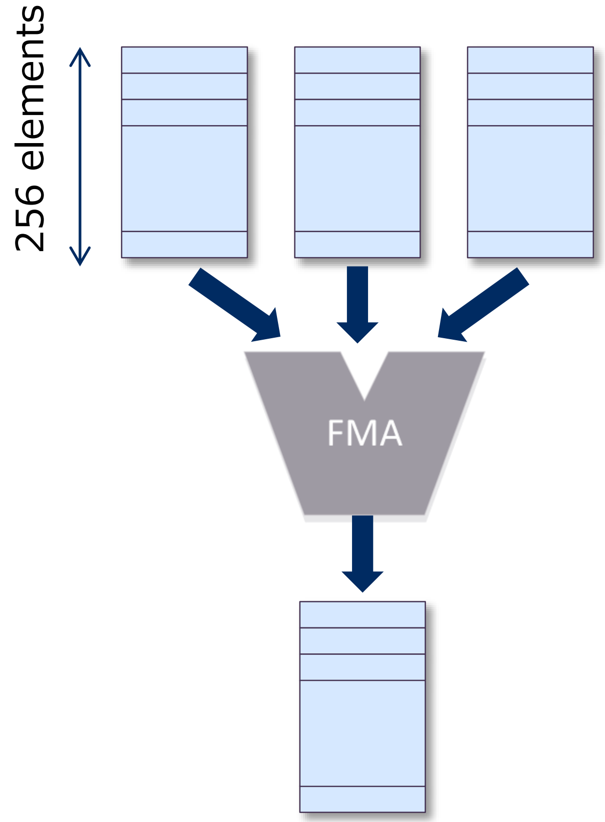


Architecture

Vector processor and HBM2 memory are implemented into Vector Engine





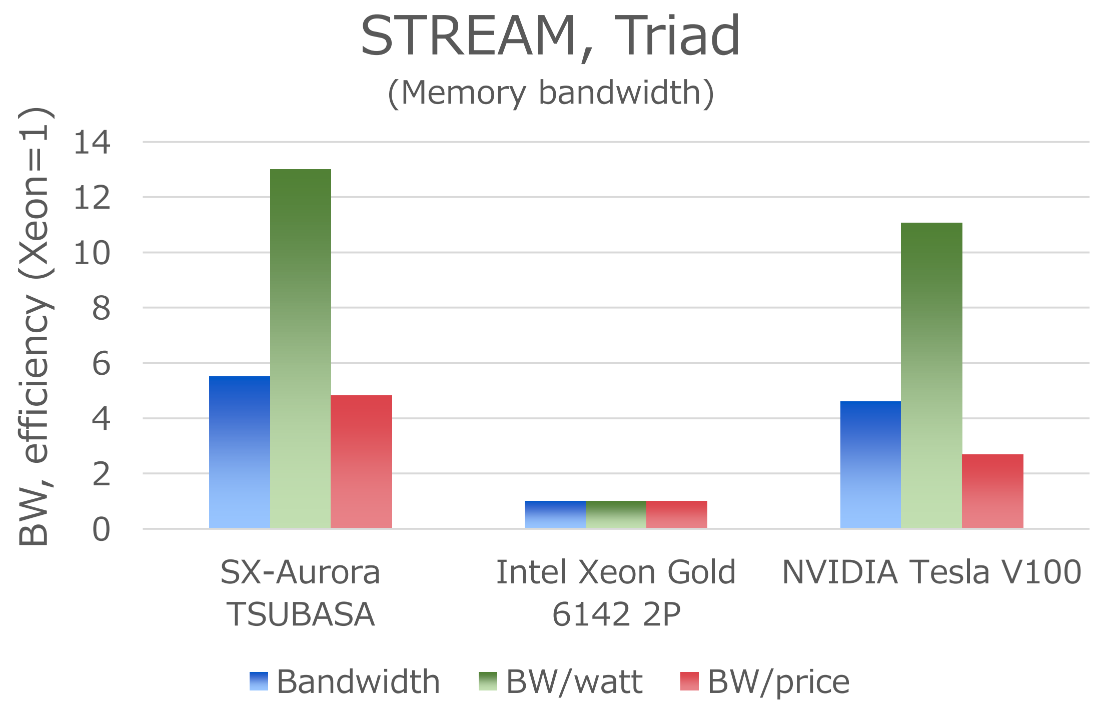


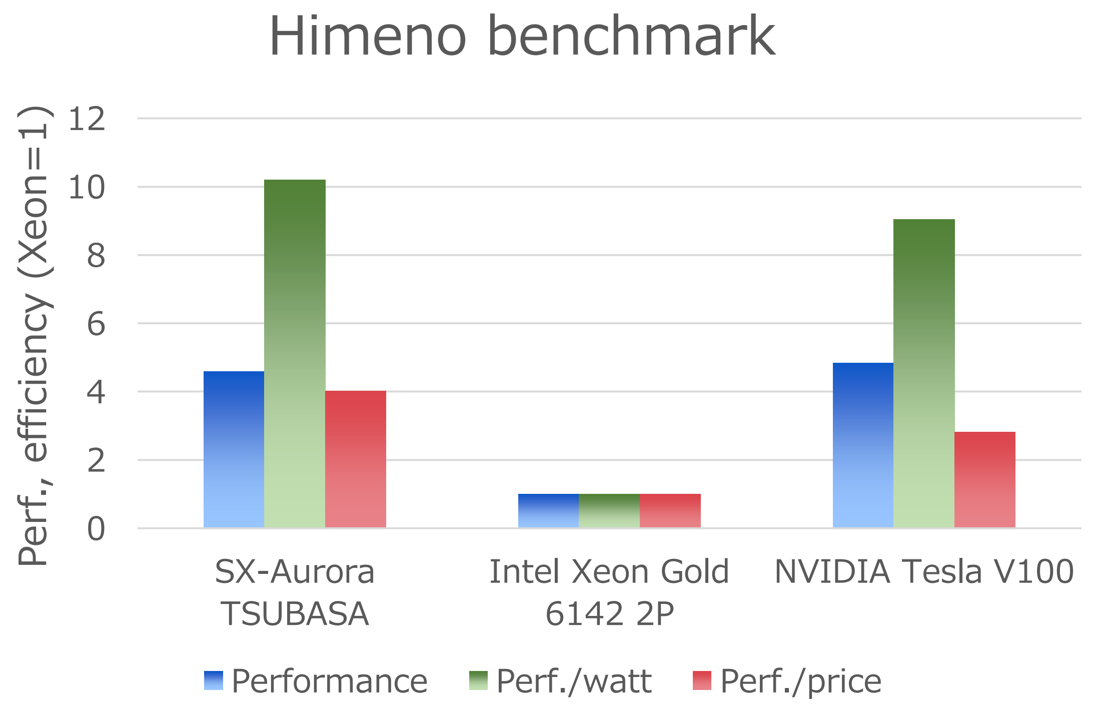
one instruction controls operations for 256 elements

Performance

▌STREAM : Industry leading memory access performance and efficiency

▌Himeno benchmark : Competitive performance and efficiency





- The STREAM benchmark is a benchmark designed to measure the sustainable memory bandwidth.
- The Himeno benchmark is a linear solver of pressure Poisson using a point-Jacobi method and known to be highly memory intensive and bound by memory bandwidth.

How to use SX-Aurora TSUBASA

Just coding standard C/C++/Fortran program

Step1: Coding

```
$ cat test.c
#include <stdio.h>
int main() {
    double a[1000], b[1000], c[1000];
    int i;
    for(i=0; i<1000; ++i){
        a[i] = b[i] + c[i];
    }
    printf("Hello, Aurora!\n");
    return 0;
}
```

- ✓ Standard C/C++/Fortran language
- ✓ Standard library functions
  printf(), fread()
  malloc(), mmap()
  socket(), etc
- ✓ Transparent access to x86 file system

Step2: Compilation

```
$ ncc test.c
ncc: vec( 101): test_v.c, line 5: Vectorized loop.
```

- ✓ NEC compiler automatically vectorize loops in a program

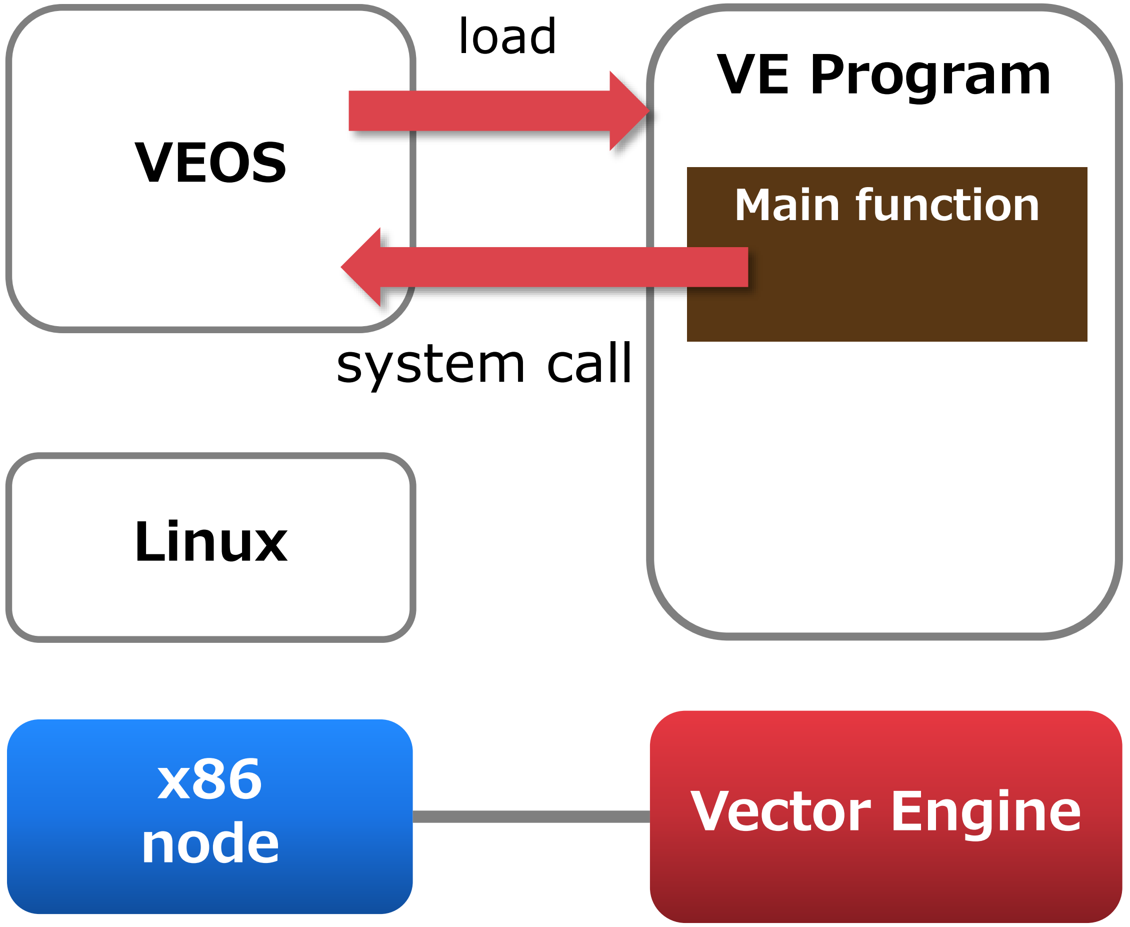
Step3: Execution

```
$ ./a.out
Hello, Aurora!
```

- ✓ The program is loaded into Vector Engine and executed transparently

How a program is executed

Vector engine executes an entire VE program

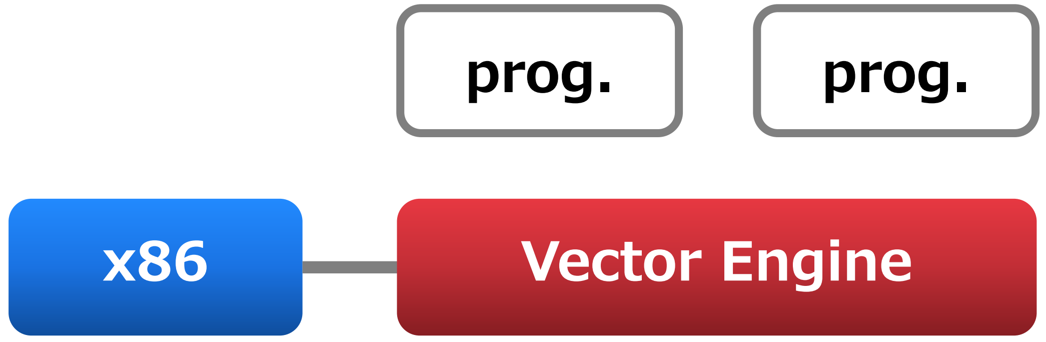


1. VEOS loads an VE program into vector engine
2. Vector engine executes the entire VE program
3. VEOS handles system call requests from the VE program
   For example: read(), write(), mmap()

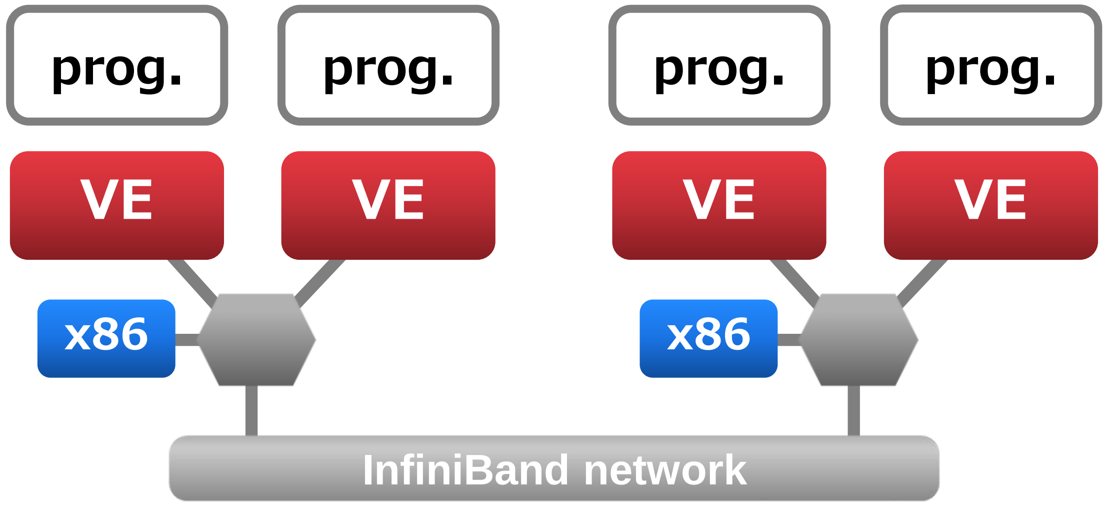
Parallelization

VE program can be parallelized to get results fast

Single VE



Multiple VEs

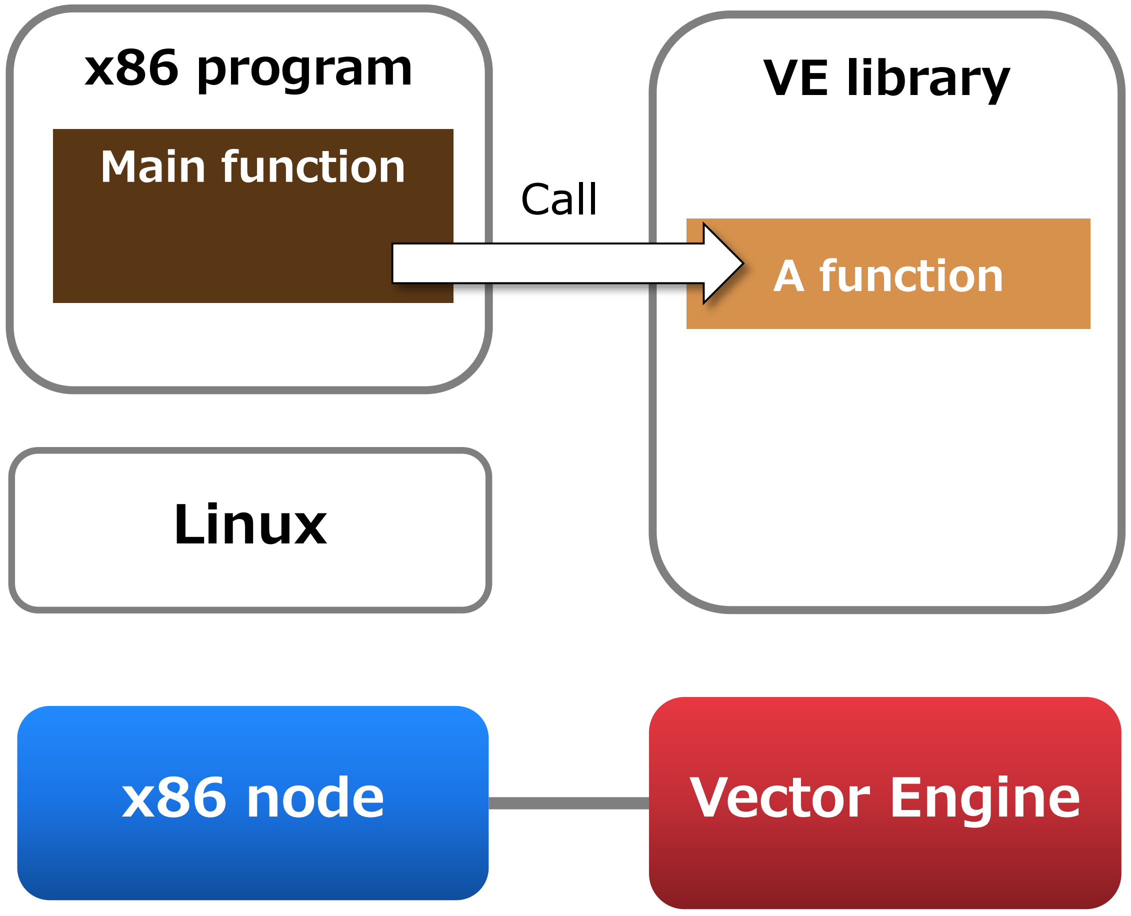


- ✓ Automatic parallelization by compiler
- ✓ OpenMP
- ✓ Pthread

- ✓ MPI
  RDMA supported

Using vector engines as accelerators

x86 program can offload heavy workload to VE



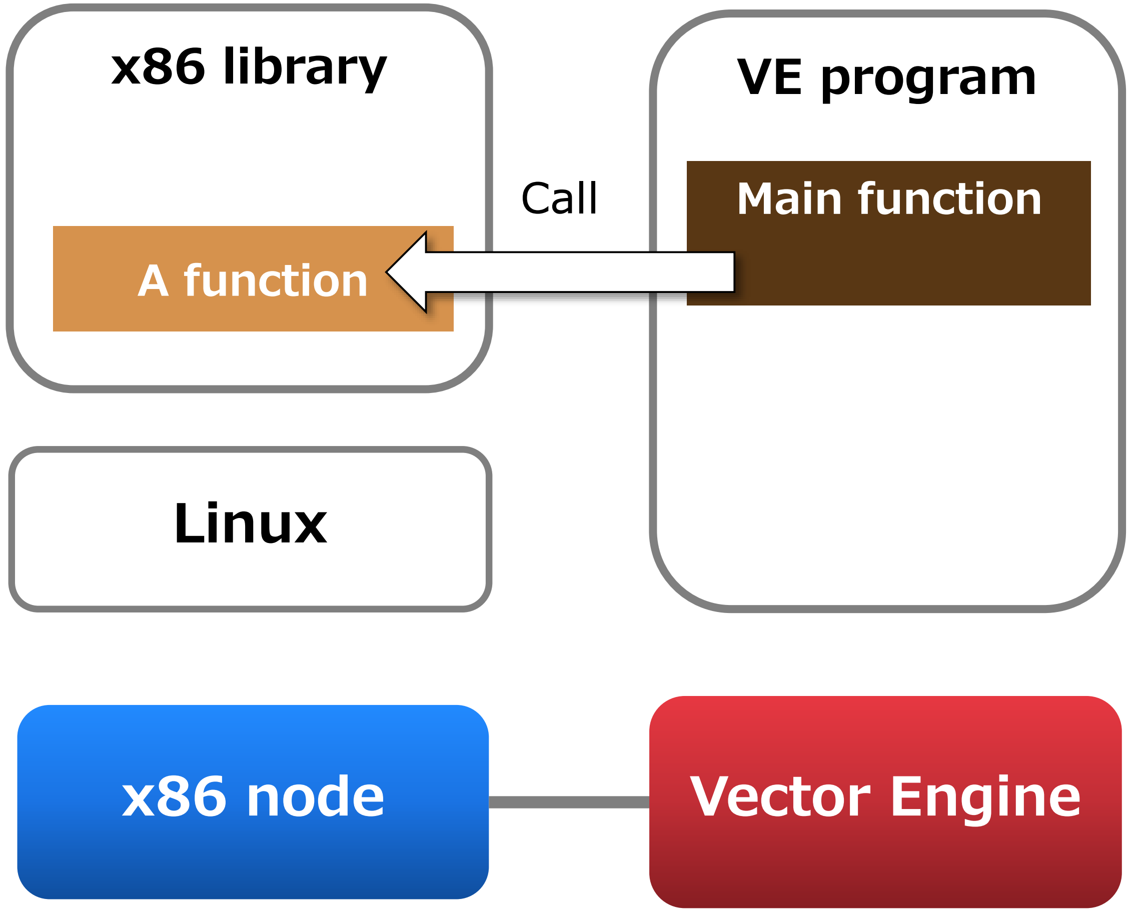
x86 program can call a function in VE library using “VE offloading” API

The tutorial and API reference are available on github
https://veos-sxarr-nec.github.io/veoffload/

Anyone can implement middleware such as python binding, heterogeneous computing framework

Offloading from VE to x86

VE program can offload pre/post operation to x86



VE program can call a function in a x86 library using "VH call" API

The tutorial and API reference are available on github.
https://veos-sxarr-nec.github.io/libsysve/

Your existing library can be called at x86 node

Conclusion

The super computer model with DLC becomes available

- 40℃ hot water cooling

Standard programming model

- Fortran/C/C++ language
- Automatic vectorization

Offloading programming models

- Using VEs as Accelerators
- Offloading from VE to x86

Utilize the performance of SX-Aurora TSUBASA!

\Orchestrating a brighter world

NEC