\Orchestrating a brighter world NEC

SX-Aurora TSUBASA

October 9, 2018

NEC Corporation

Agenda

- Overview of SX-Aurora TSUBASA
- Performance and use cases
- Roadmap

SX-Aurora TSUBASA Overview

History of Vector computing

NEC has always provided high sustained performance by vector supercomputer SX series

Good, but…
- large
- expensive
- special

like dinosaurs

Vector Engine (PCI c

Change!

Earth Simulator 3

Earth Simulator 2

Earth Simulator

SX-ACE

SX-9

SX-8

SX-7

SX-6

SX-5

SX-4

SX-3

SX-2

Hardware innovations

Software innovations

Performance

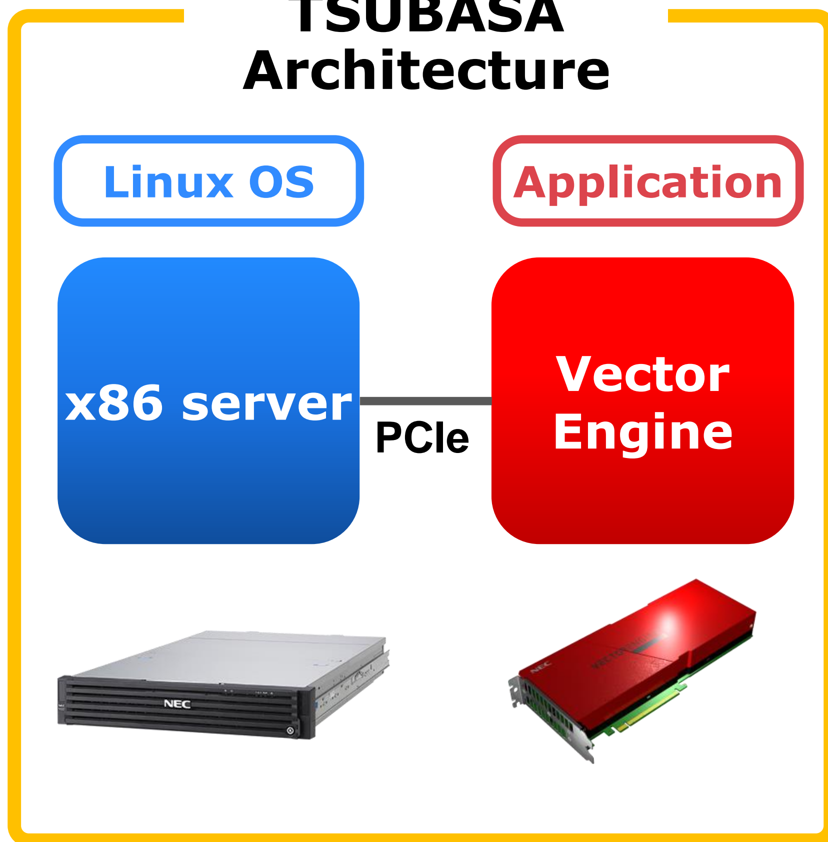
1990 2000 2010

Packed vector technologies accumulated over 35 years into PCI card

© NEC Corporation 2018

New Architecture

- SX-Aurora TSUBASA = Standard x86 + Vector Engine
- Linux + standard language (Fortran/C/C++)
- Enjoy high performance with easy programming



Hardware

- Standard x86 server + Vector Engine

Software

- Linux OS
- Automatic vectorization compiler
- Fortran/C/C++
  → No special programming like CUDA

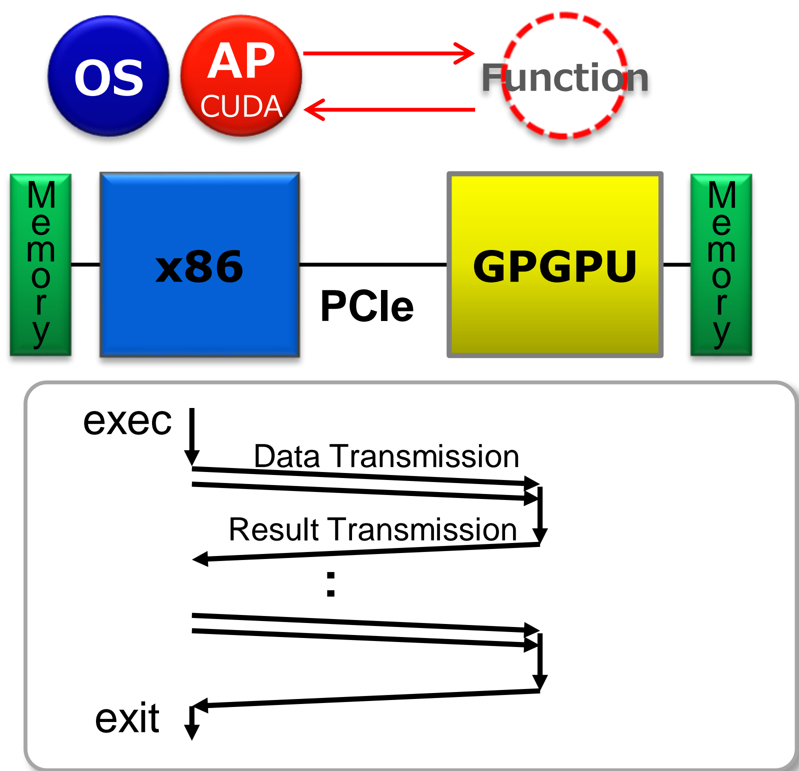
Interconnect

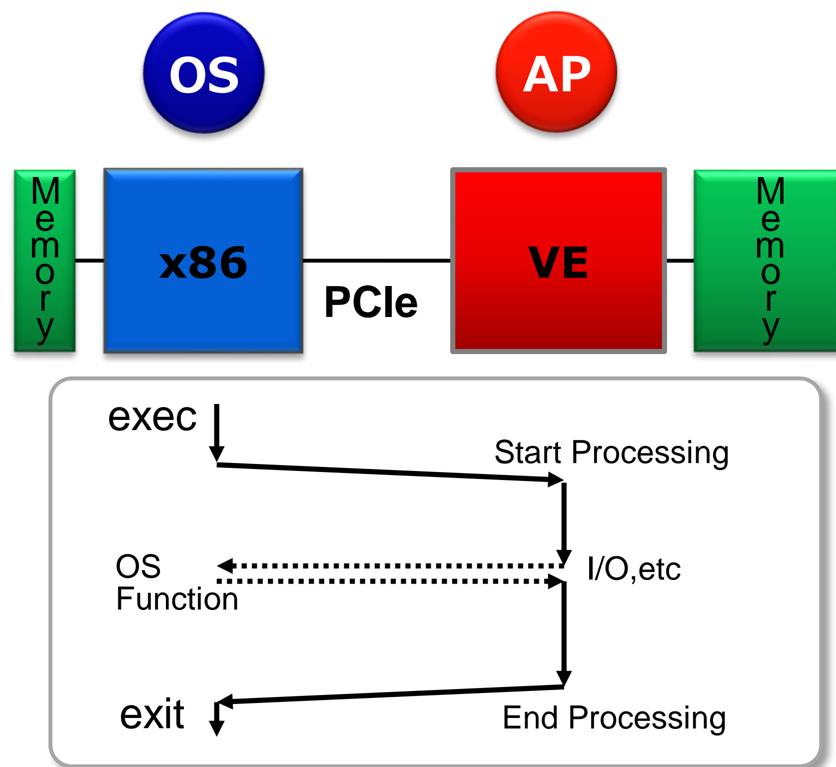
- InfiniBand for MPI
- VE-VE direct communication support

Easy programming (standard language) ▶ Automatic vectorization compiler ▶ Enjoy high performance

© NEC Corporation 2018

GPGPU and VE





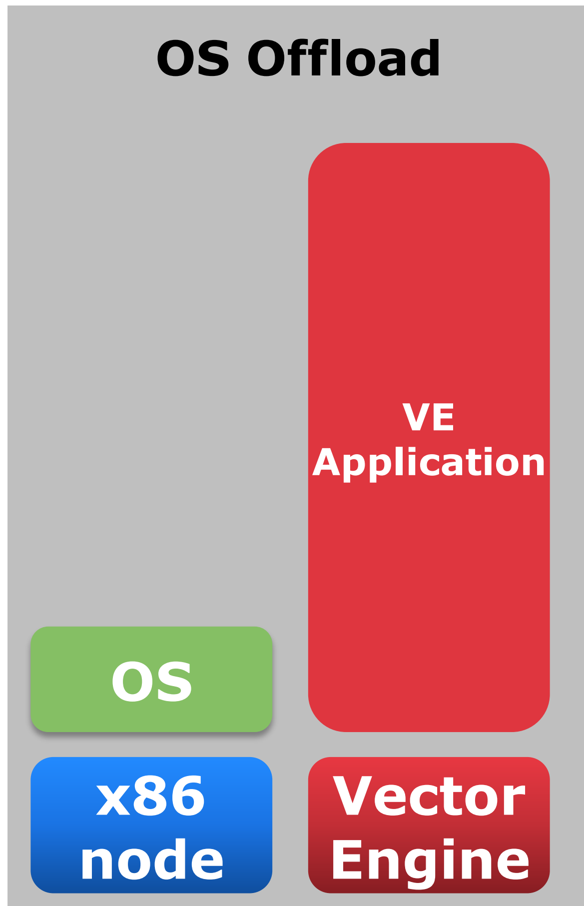
disadvantage
- PCIe bottleneck
- Programming difficulty

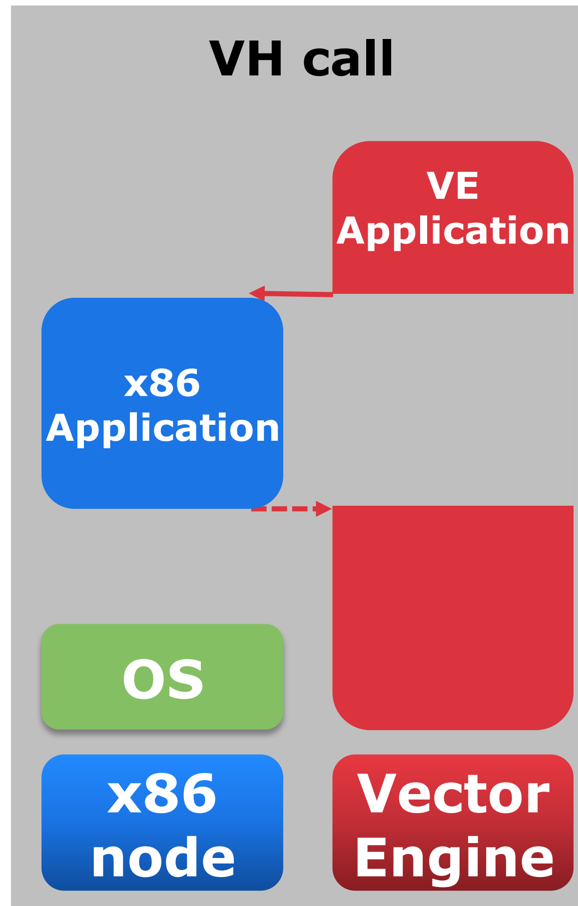
Advantage
- Avoiding PCIe bottleneck
- Standard language

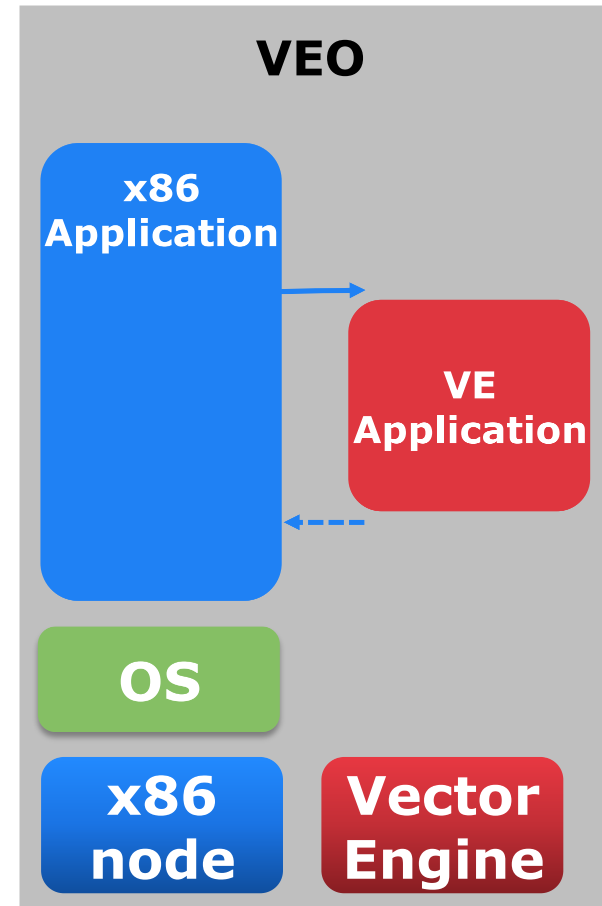
© NEC Corporation 2018

VEOS offload models

Run the application in the right way

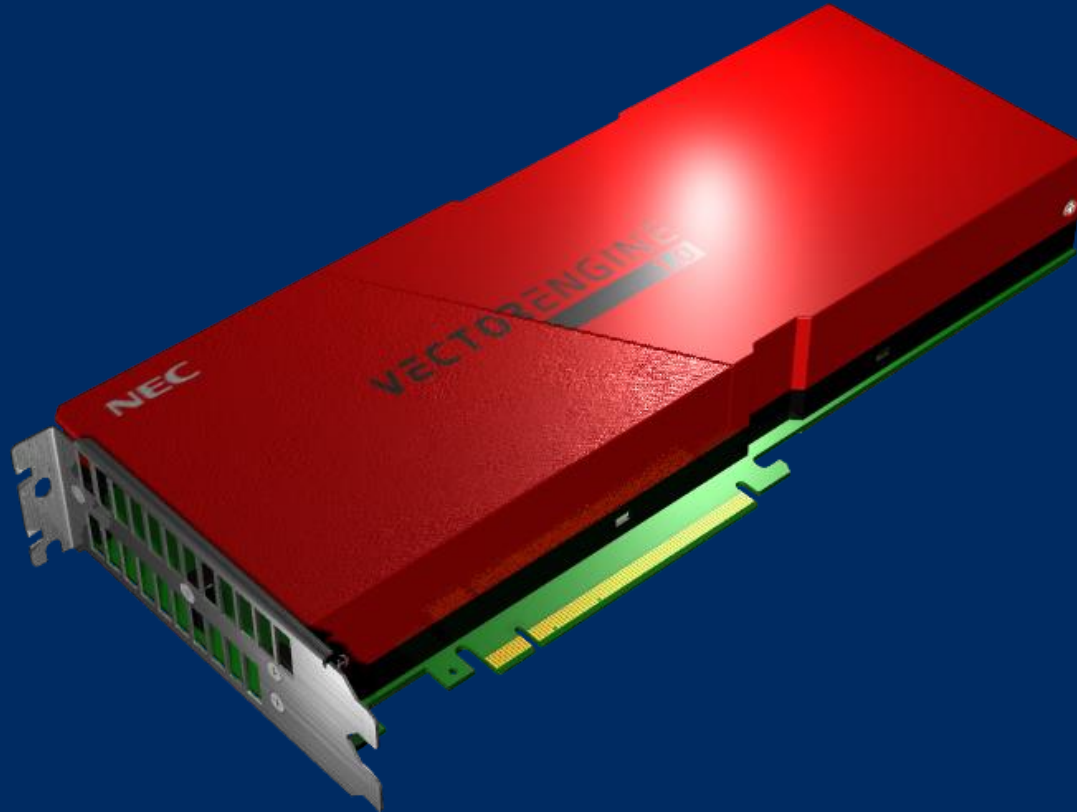






© NEC Corporation 2018

Vector Processor on The Card



- New Developed Vector Processor
- PCIe Card Implementation
- 8 cores / processor
- 2.45TF performance
- 1.22TB/s memory bandwidth
- Normal programing with Fortran/C/C++

© NEC Corporation 2018
\Orchestrating a brighter world NEC

Product Lineup

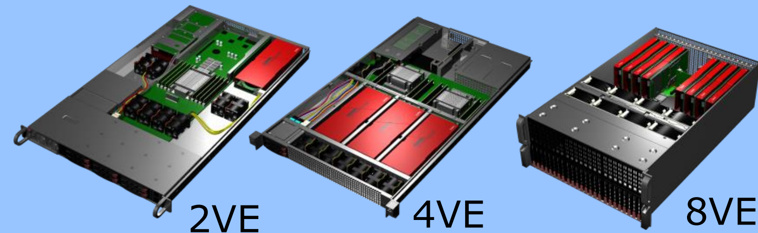
- Product: VEs + x86/Linux server
- 3 series, A100, A300, and A500 for various users

Supercomputer Model
- For large scale configuration
- DLC with 40℃ water

A500 series

Rack Mount Model
- Flexible configuration
- Air Cooled

A300 series



Tower Model
- For developer/programmer
- Tower implementation

A100 series

© NEC Corporation 2018

Usability

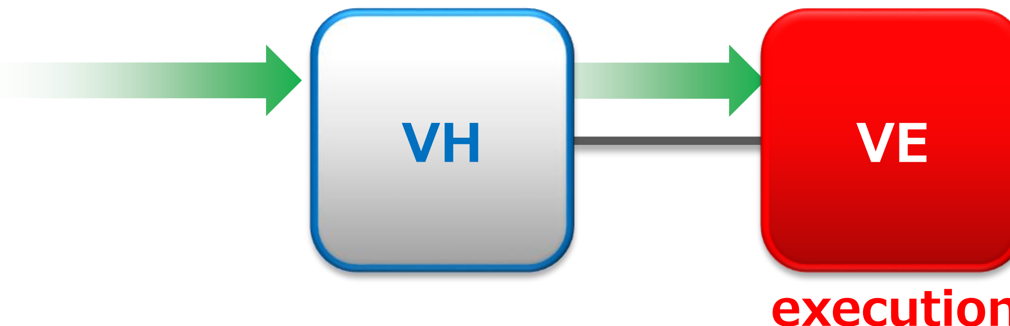
Programing Environment

Vector Cross Compiler

- automatic vectorization
- automatic parallelization

Fortran:	F2003, F2008
C/C++:	C11/C++14
OpenMP:	OpenMP4.5
Library:	MPI 3.1, libc, BLAS, Lapack, etc
Debugger:	gdb, Eclipse parallel tools platform
Tools:	PROGINF, FtraceViewer

Execution Environment



© NEC Corporation 2018

Vector programming

- Standard C/Fortran programming (No need to change program)
- Automatic vectorization feature and various tools help vectorization

```
void matmul(float *A, float *B, float *C, int l, int m, int n){
    int i, j, k;
    for (i = 0; i < l; i++) {
        for (j = 0; j < n; j++) {
            float sum = 0.0;
            for (k = 0; k < m; k++)
                sum += A[i * m + k] * B[k * n + j];
            C[i*n+j] = sum;
        }
    }
}

void alloc_matrix(float **m_h, int h, int w){
    *m_h = (float *)malloc(sizeof(float) * h * w);
}

// other function definitions …

int main(int argc, char *argv[]){
    float *Ah, *Bh, *Ch;
    struct timeval t1, t2;

    // prepare matrix A
    alloc_matrix(&Ah, L, M);
    init_matrix(Ah, L, M);
    // do it again for matrix B
    alloc_matrix(&Bh, M, N);
    init_matrix(Bh, M, N);
    // allocate spaces for matrix
    alloc_matrix(&Ch, L, N);

    // call matmul function
    matmul(Ah, Bh, Ch, L, M, N);

    return 0;
}
```

No modification is necessary for vectorization

Just compile, and loops are vectorized automatically

[Compiler diagnostic message]

```
$ ncc sample.c -O4 -report-all -fdiag-vector=2
ncc: opt(1589): sample.c, line 11: Outer loop moved inside inner loop(s).: j
ncc: vec( 101): sample.c, line 11: Vectorized loop.
ncc: opt(1592): sample.c, line 13: Outer loop unrolled inside inner loop.: k
ncc: vec( 101): sample.c, line 13: Vectorized loop.
ncc: vec( 128): sample.c, line 14: Fused multiply-add operation applied.
ncc: vec( 101): sample.c, line 15: Vectorized loop.
```

[Format list]

```
 8:              void matmul(float *A, float *B, float *C, int l, int m, int n){
 9:                  int i, j, k;
10: +------>         for (i = 0; i < l; i++) {
11: |X----->             for (j = 0; j < n; j++) {
12: ||                       float sum = 0.0;
13: ||V---->                 for (k = 0; k < m; k++)
14: ||V---- F                    sum += A[i * m + k] * B[k * n + j];
15: ||                       C[i*n+j] = sum;
16: |X-----              }
17: +------          }
18:              }
```


© NEC Corporation 2018

Profiler tools : Ftrace

User can obtain performance information for each function as well as user specified regions with Ftrace

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	VLD LLC HIT E.%	PROC.NAME
1012	49.093(24.0)	48.511	23317.2	14001.4	96.97	83.2	42.132	5.511	0.000	80.32	funcA
160640	37.475(18.3)	0.233	17874.6	9985.9	95.22	52.2	34.223	1.973	2.166	96.84	funcB
160640	30.515(14.9)	0.190	22141.8	12263.7	95.50	52.8	29.272	0.191	2.544	93.23	funcC
160640	23.434(11.5)	0.146	44919.9	22923.2	97.75	98.5	21.869	0.741	4.590	97.82	funcD
160640	22.462(11.0)	0.140	42924.5	21989.6	97.73	99.4	20.951	1.212	4.590	96.91	funcE
53562928	15.371(7.5)	0.000	1819.0	742.2	0.00	0.0	0.000	1.253	0.000	0.00	funcG
8	14.266(7.0)	1783.201	1077.3	55.7	0.00	0.0	0.000	4.480	0.000	0.00	funcH
642560	5.641(2.8)	0.009	487.7	0.2	46.45	35.1	1.833	1.609	0.007	91.68	funcF
2032	2.477(1.2)	1.219	667.1	0.0	89.97	28.5	2.218	0.041	0.015	70.42	funcI
8	1.971(1.0)	246.398	21586.7	7823.4	96.21	79.6	1.650	0.271	0.000	2.58	funcJ
54851346	204.569(100.0)	0.004	22508.5	12210.7	95.64	76.5	154.524	17.740	13.916	90.29	total
62248	37.709(18.4)	0.606	2200.2	1026.4	0.00	0.0	0.000	0.532	0.000	20.00	loop#1
2032	4.834(2.4)	2.379	415.8	0.0	28.61	6.3	4.098	0.246	0.000	0.00	loop#2

...

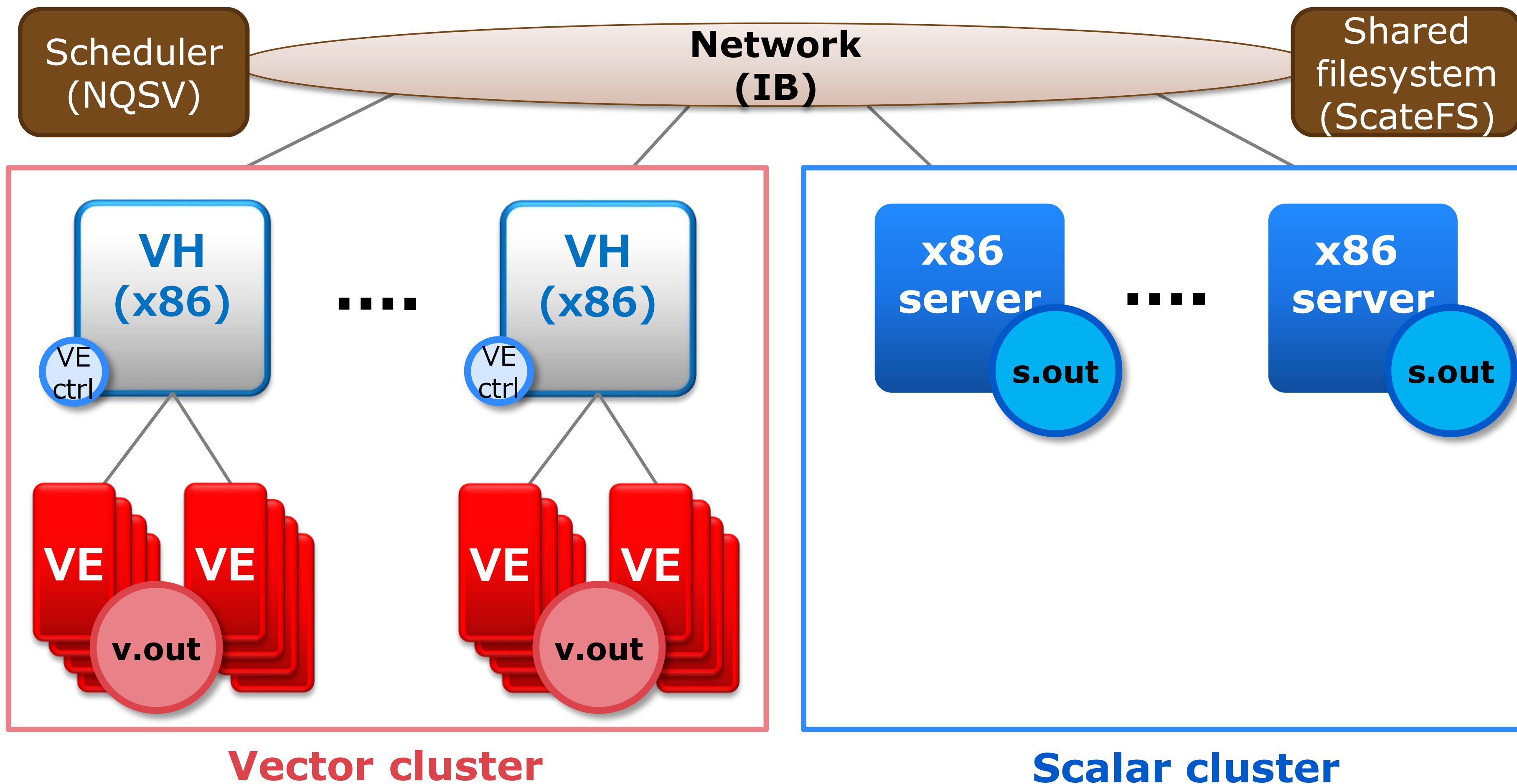
```
#include <ftrace.h>
...
(void) ftrace_region_begin("loop#1") // outside region begin

for (i = 0; i < n; i++) {
    ...
}

(void) ftrace_region_begin("loop#2") // inside region begin
for (j = 0; j < n; j++) {
    ...
}
(void) ftrace_region_end(“loop#2”)  // inside region end

(void) ftrace_region_end(“loop#1”)  // outside region end
```

User specified regions

© NEC Corporation 2018

Hybrid system

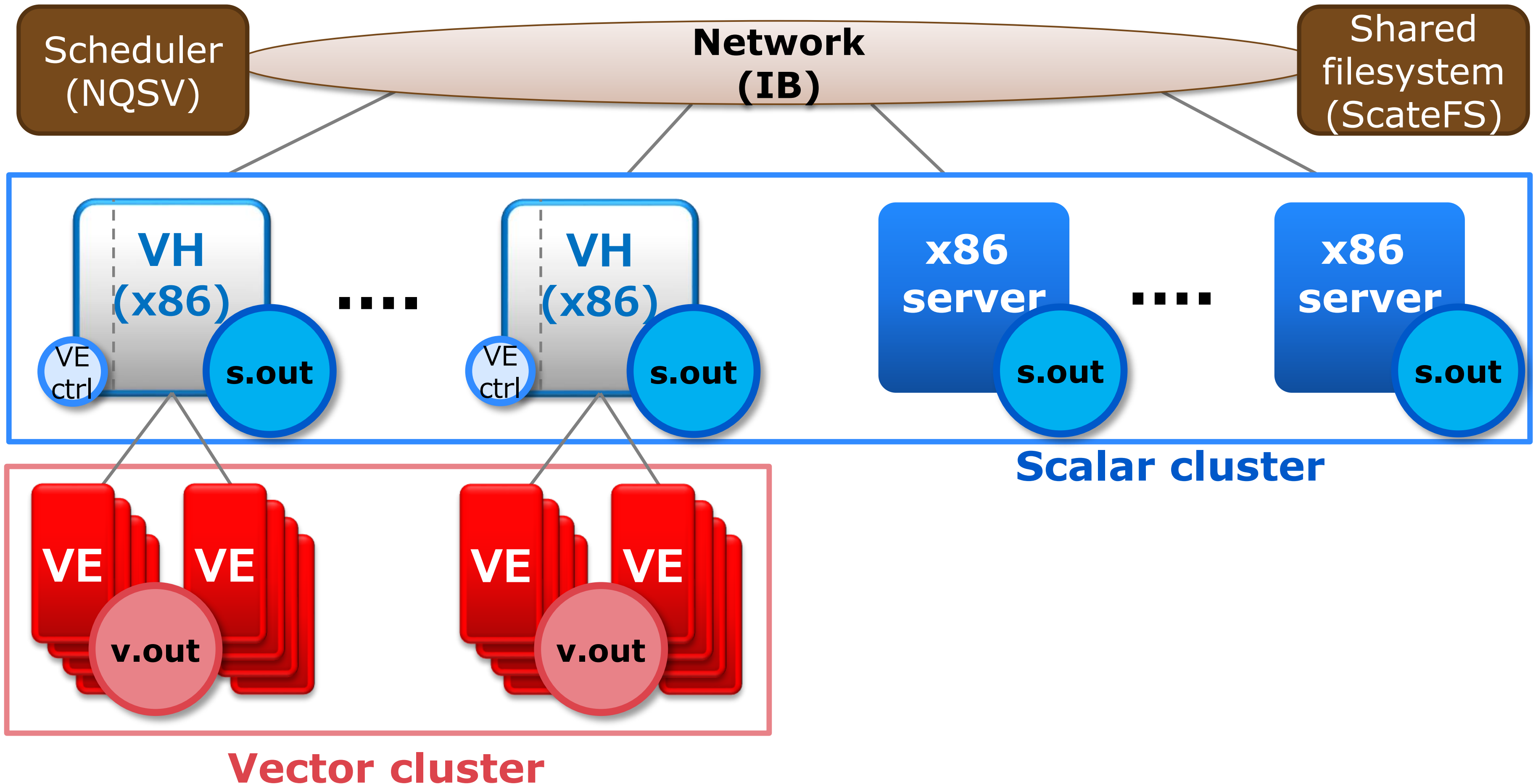
There are various types of applications
→ Run the right application on the right platform



© NEC Corporation 2018

Hybrid system

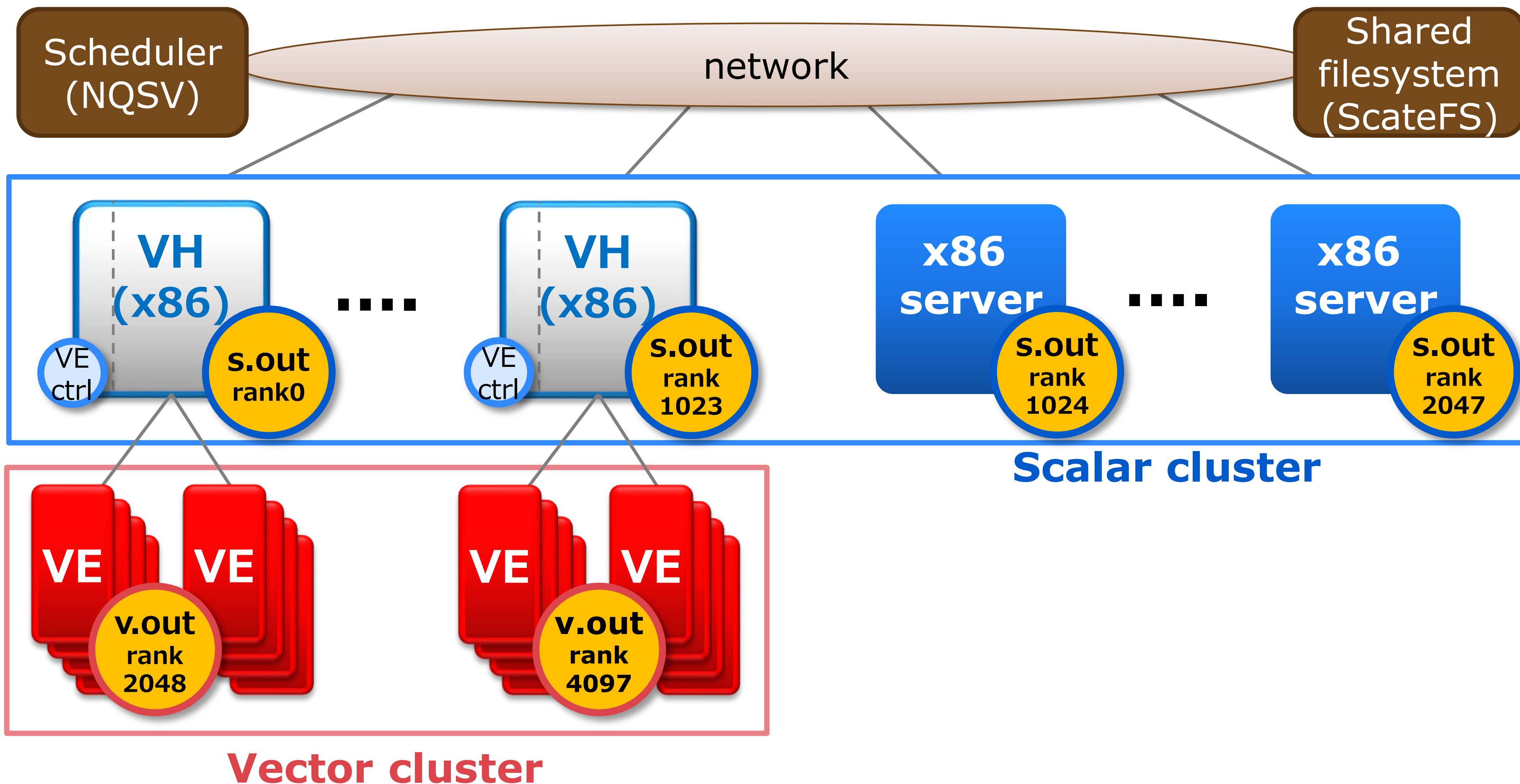
Incorporate VH as part of scalar cluster
→ Higher performance, Higher system utilization



© NEC Corporation 2018

Hybrid MPI

MPI process running on both VE node and x86 node as a single MPI program.

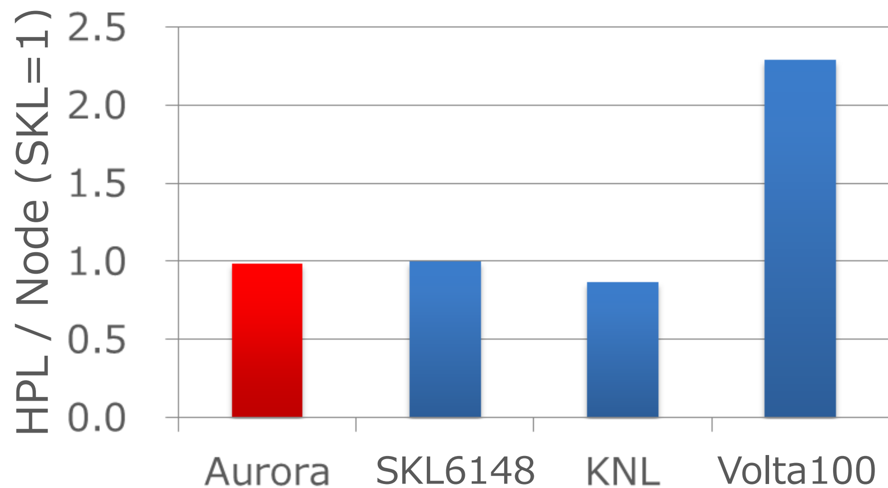


© NEC Corporation 2018

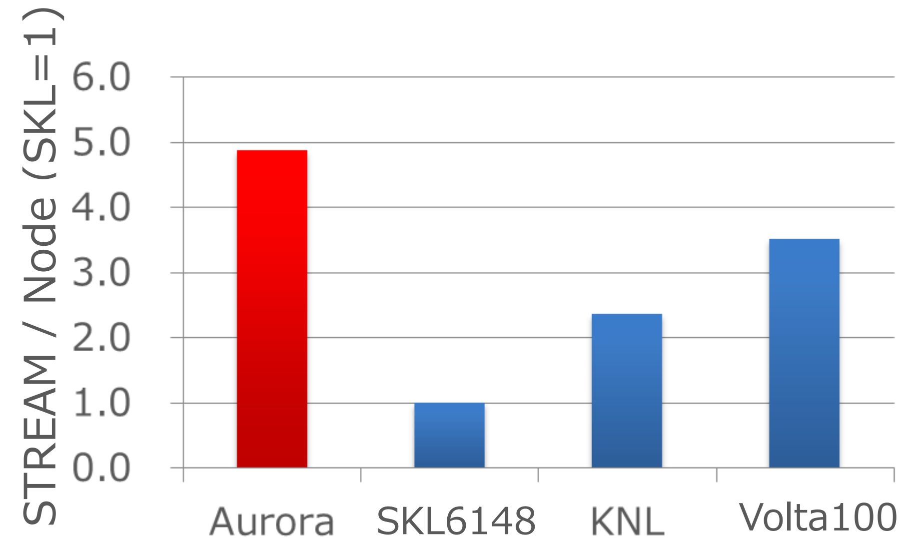
Performance and use cases

HPL and STREAM

- HPL : Aurora provides competitive FLOPS capability
- STREAM : Aurora provides highest sustained memory bandwidth



- Aurora provides same range HPL sustained performance as SKL/KNL



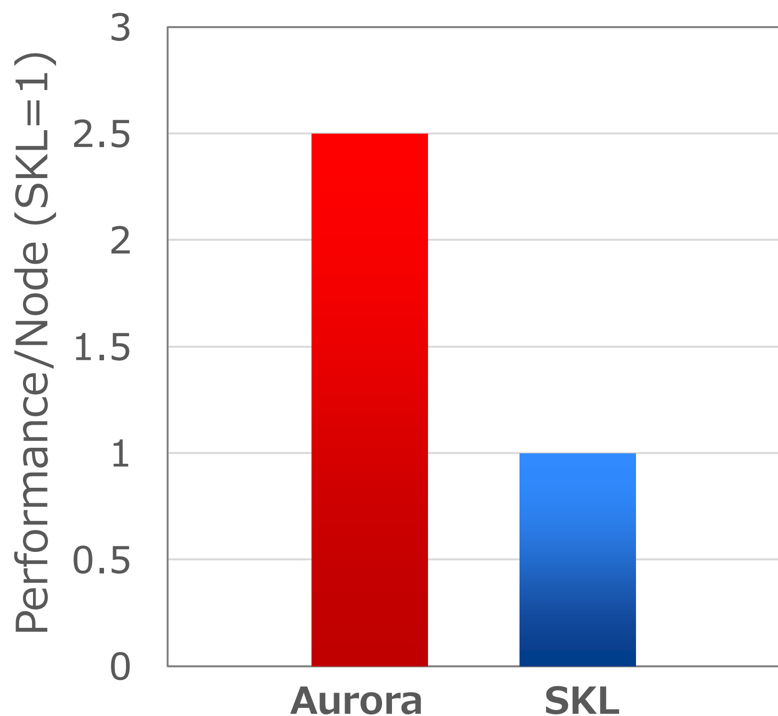
- Aurora provides the highest memory bandwidth

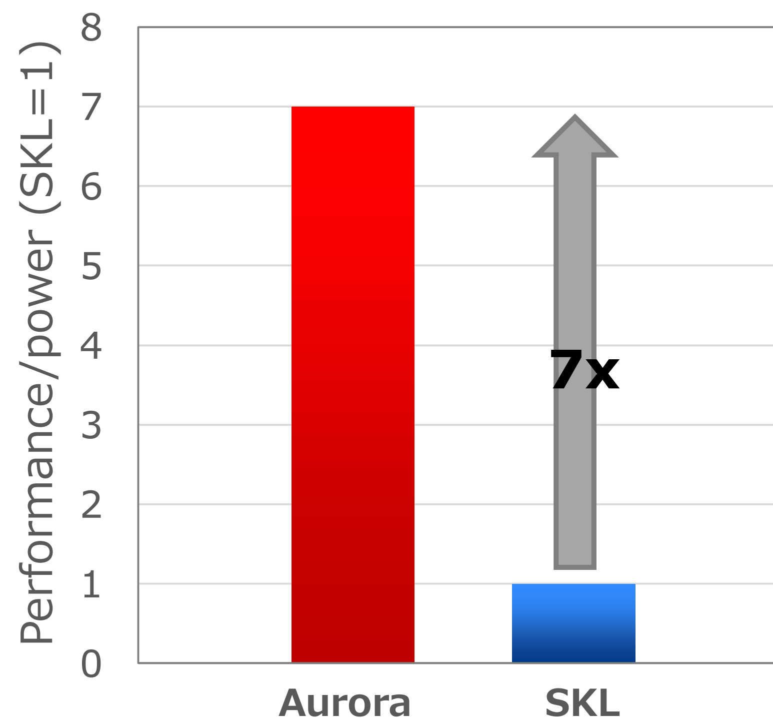
- Aurora is Vector Engine Type 10-B (1.4GHz, 8core)
- SKL is Intel Skylake 6148 Xeon x2/node
- KNL is Intel Knight Landing x1/node
- V100 is NVIDIA Tesla V100 x1/node

© NEC Corporation 2018

HPCG

Performance/power of Aurora shows 7 times better than SKL



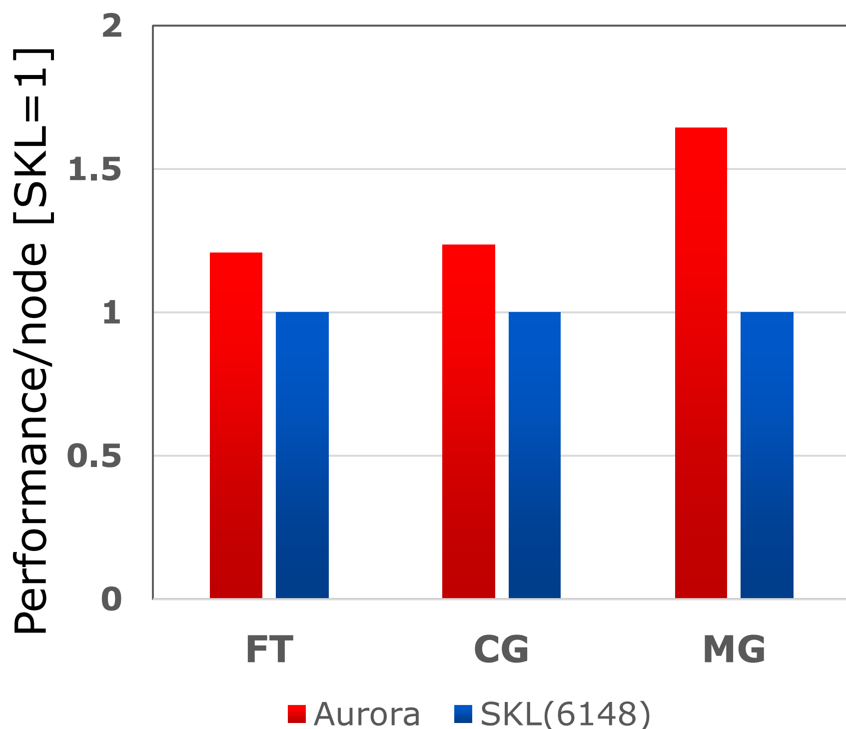


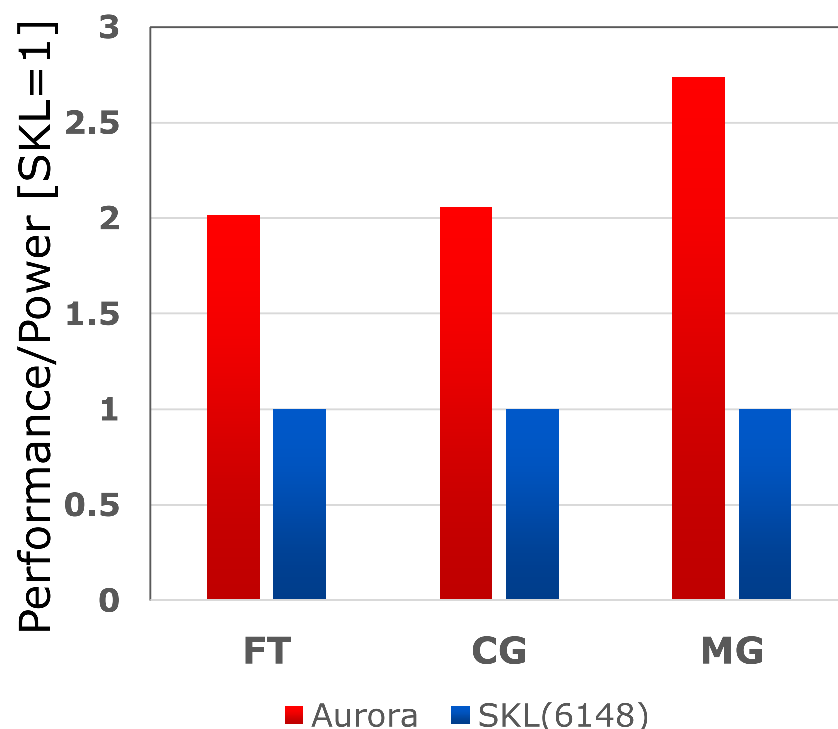
- Aurora is Vector Engine Type 10-B (1.4GHz, 8core)
- SKL is Intel Skylake 6148 Xeon x2/node

© NEC Corporation 2018

NAS Parallel Benchmark (Class C)

Aurora shows over 2x performance/power compared to SKL(6148)



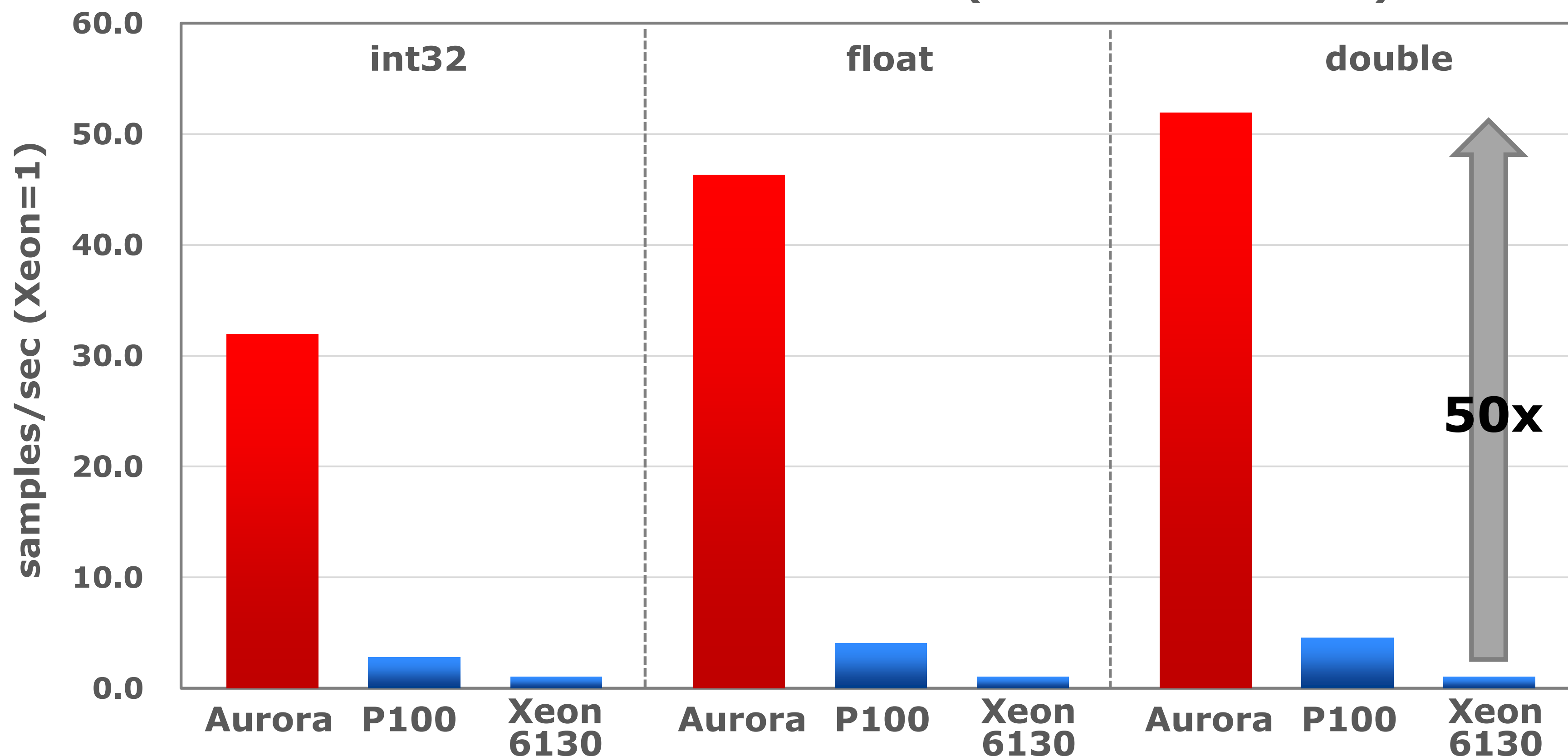


- Aurora is Vector Engine Type 10-B (1.4GHz, 8core)
- SKL is Intel Skylake 6148 Xeon x2/node

© NEC Corporation 2018

Library performance : Random Generation (MT)

Aurora shows over 50x performance compared to Xeon



- Aurora: Vector Engine Type 10-B (1.4GHz, 8core) using MT19937
- P100: NVIDIA Tesla P100 x1 node using MT19937
- Xeon6130: Intel Xeon Gold 6130 using MT19937

➢MT19937 : A high-quality pseudorandom number generation algorithm. Frequently used.

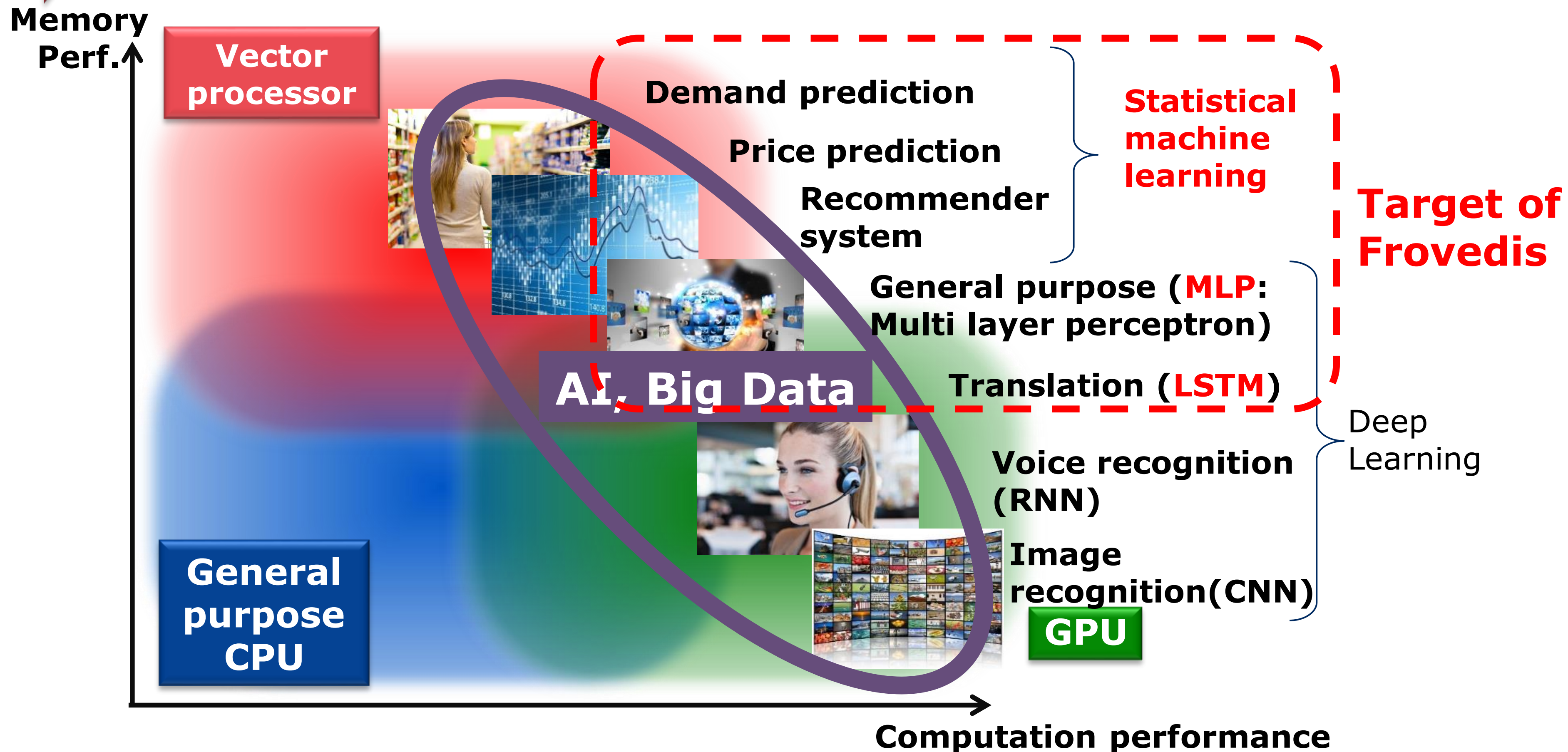
© NEC Corporation 2018

AI and BigData

We target accelerating memory intensive workloads for HPC and AI

✓ Several statistical MLs and some Neural Networks (MLP, LSTM)

Created middleware that speeds up Spark for statistical ML



© NEC Corporation 2018

Frovedis: NEC Middleware for statistical machine learning

Frovedis: NEC middleware compatible to Spark

- Same API
- Only 3 lines to change original spark program
- OSS developed by NEC
  https://github.com/frovedis

Original Spark program: logistic regression

```
…
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
…
val model = LogisticRegressionWithSGD.train(data)
…
```

Change to call NEC middleware implementation

```
…
import com.nec.frovedis.mllib.classificaiton.LogisticRegressionWithSGD
…
FrovedisServer.initialize(…)
val model = LogisticRegressionWithSGD.train(data)
FrovedisServer.shut_down()
…
```

Change import

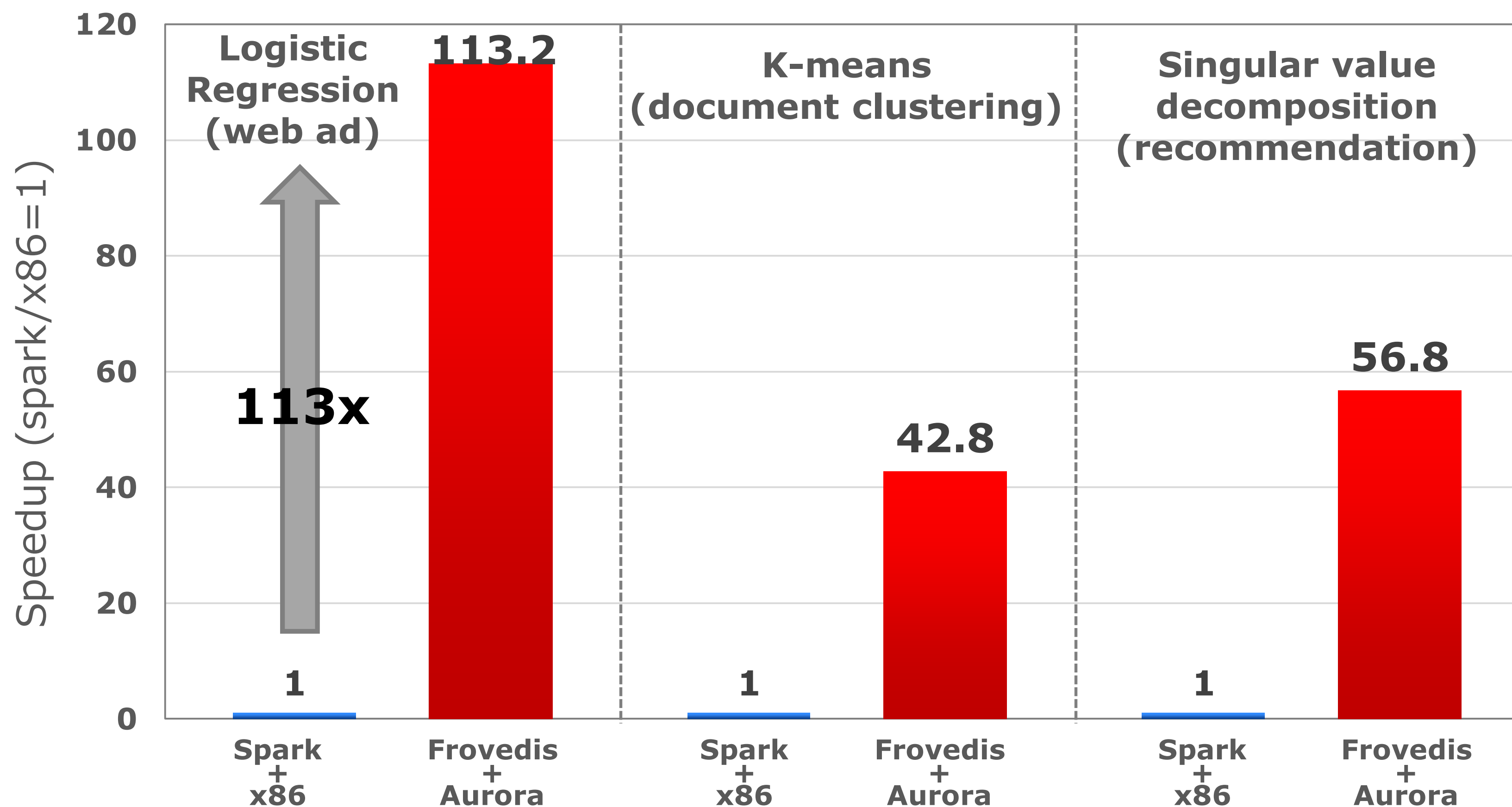
Same API (no change)

Start/Stop server

© NEC Corporation 2018

Performance of Frovedis

Frovedis + VE shows over 100x performance compared to Spark + x86



- x86: Intel Xeon Gold 6126 x1 socket
- Aurora: Vector Engine Type 10-B (1.4GHz, 8core) x1
- Performance comparison does not include I/O time

© NEC Corporation 2018

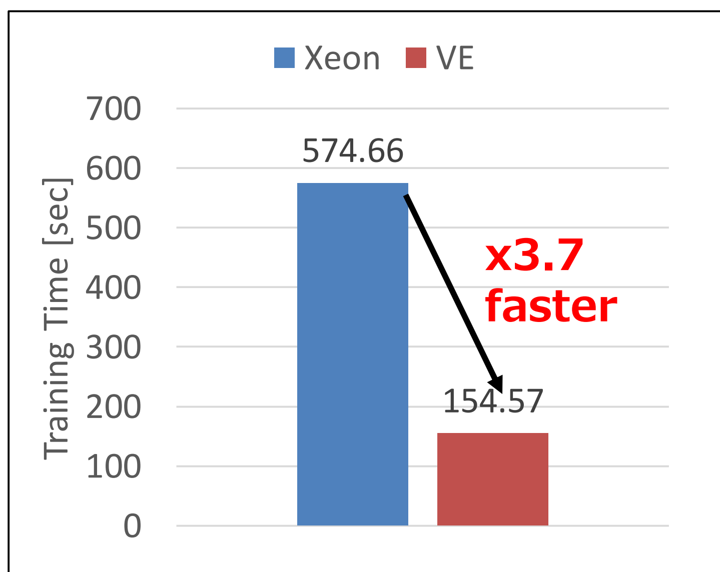
Use case: Malware Detection

- Malware detection training used to take few days → Daily training
- Detect new malware rapidly → Reduce malware damage

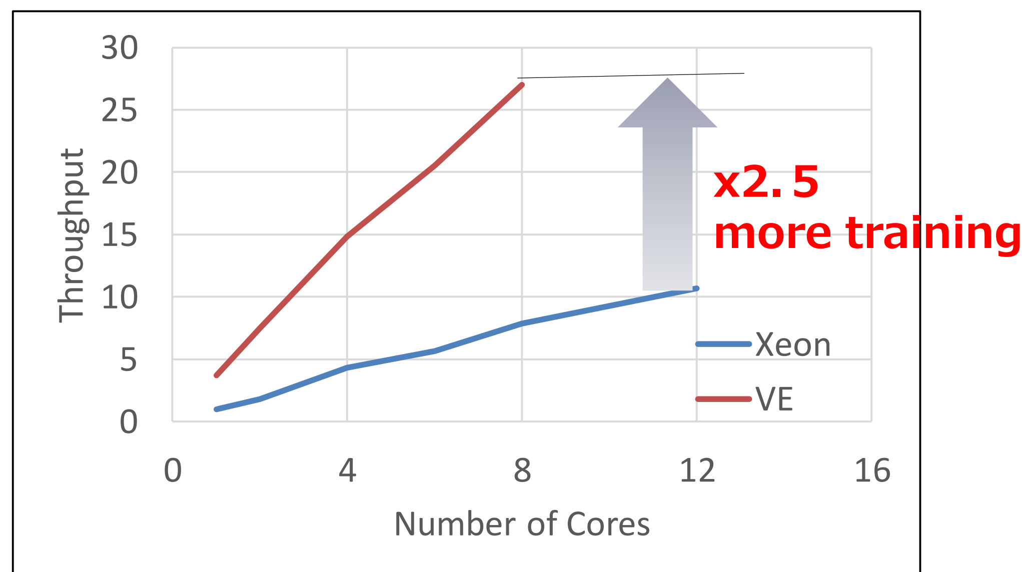
Analyze application binary and detect malware

Using NEC machine learning application

Training time using 1 core
(required training time for 100 epoch)



Performance per socket
(training throughput)



Xeon Gold 6126 (12c, 2.6GHz/1.7GHz): 652.8GFlops(DP)
VE(8c, 1.4GHz): 2150.4 GFlops(DP)

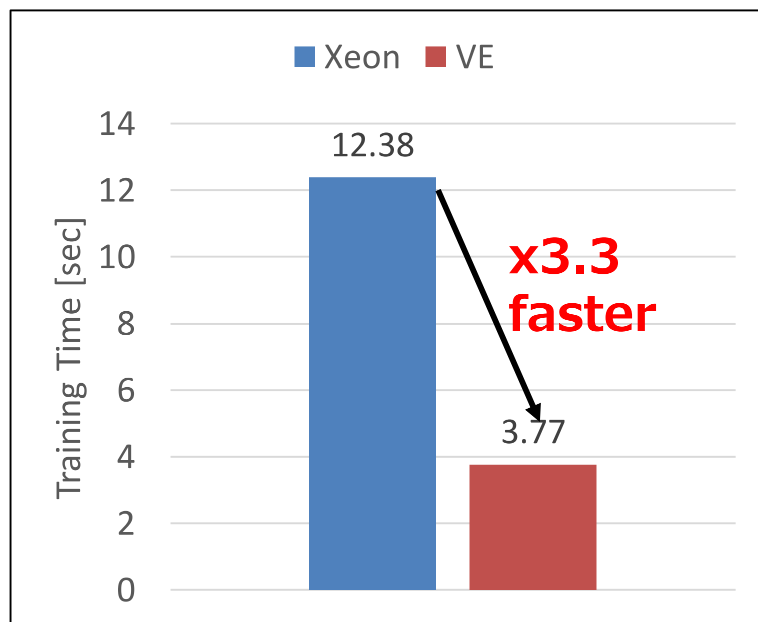
Assign independent training to each core
(assuming parallel training with different parameters)

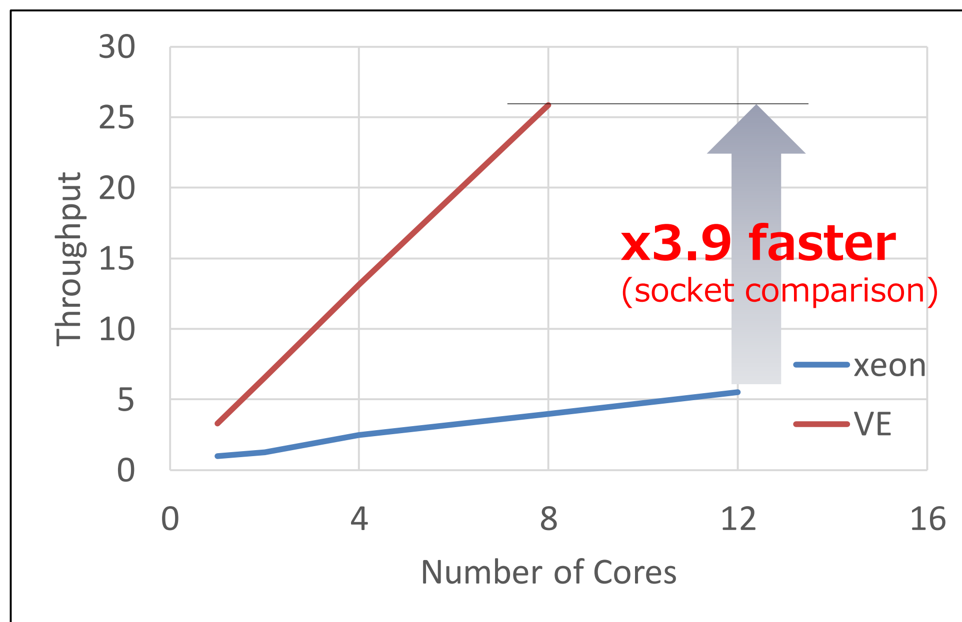
© NEC Corporation 2018

Use case: Financial Option Pricing

European option pricing (Monte Carlo)

- Using Intel MKL financial option pricing example
  https://software.intel.com/en-us/mkl_cookbook_samples





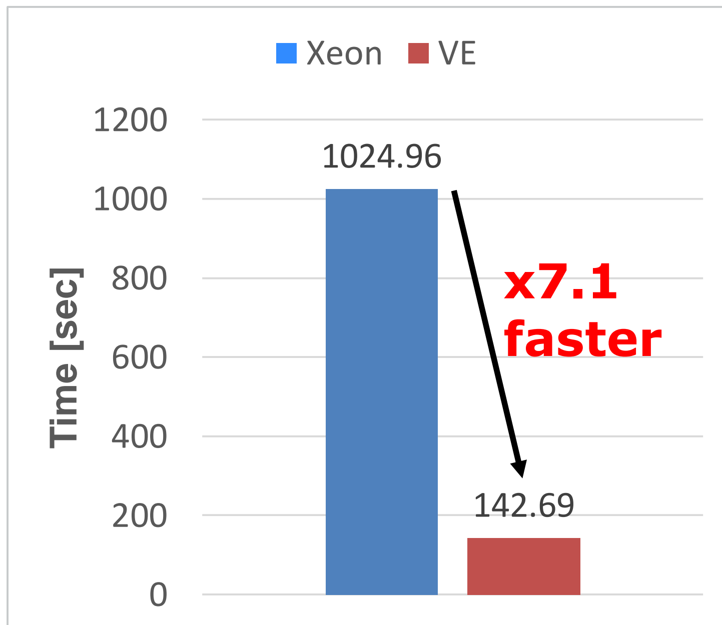
Xeon Gold 6126 (12c, 2.6GHz/1.7GHz): 652.8GFlops(DP)
VE(8c, 1.4GHz): 2150.4 GFlops(DP)

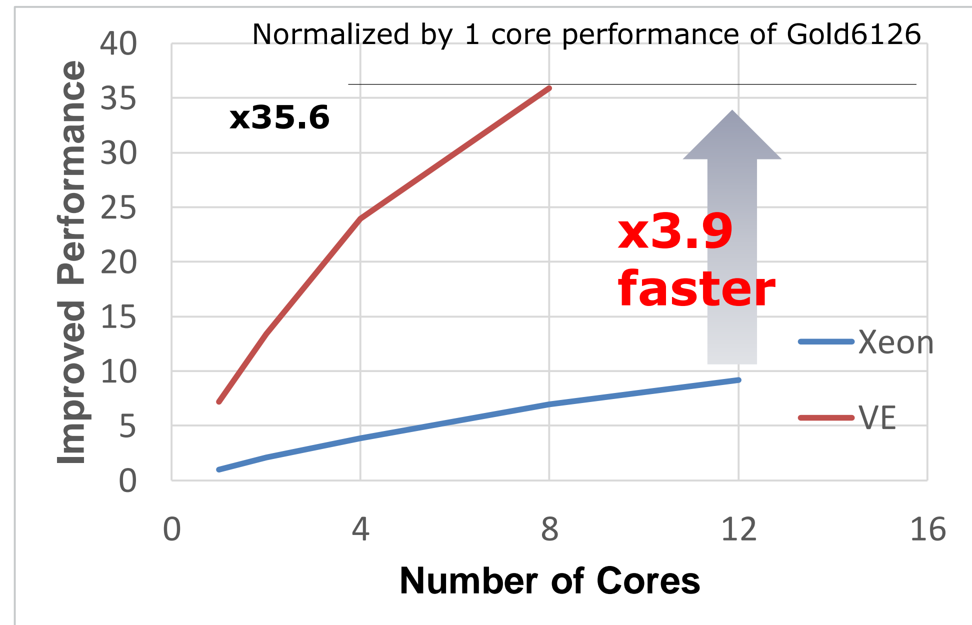
© NEC Corporation 2018

CT Scan Program Benchmark Result (MBIR)

~4x higher performance than 1 socket Xeon

Model-Based Iterative Reconstruction (MBIR)



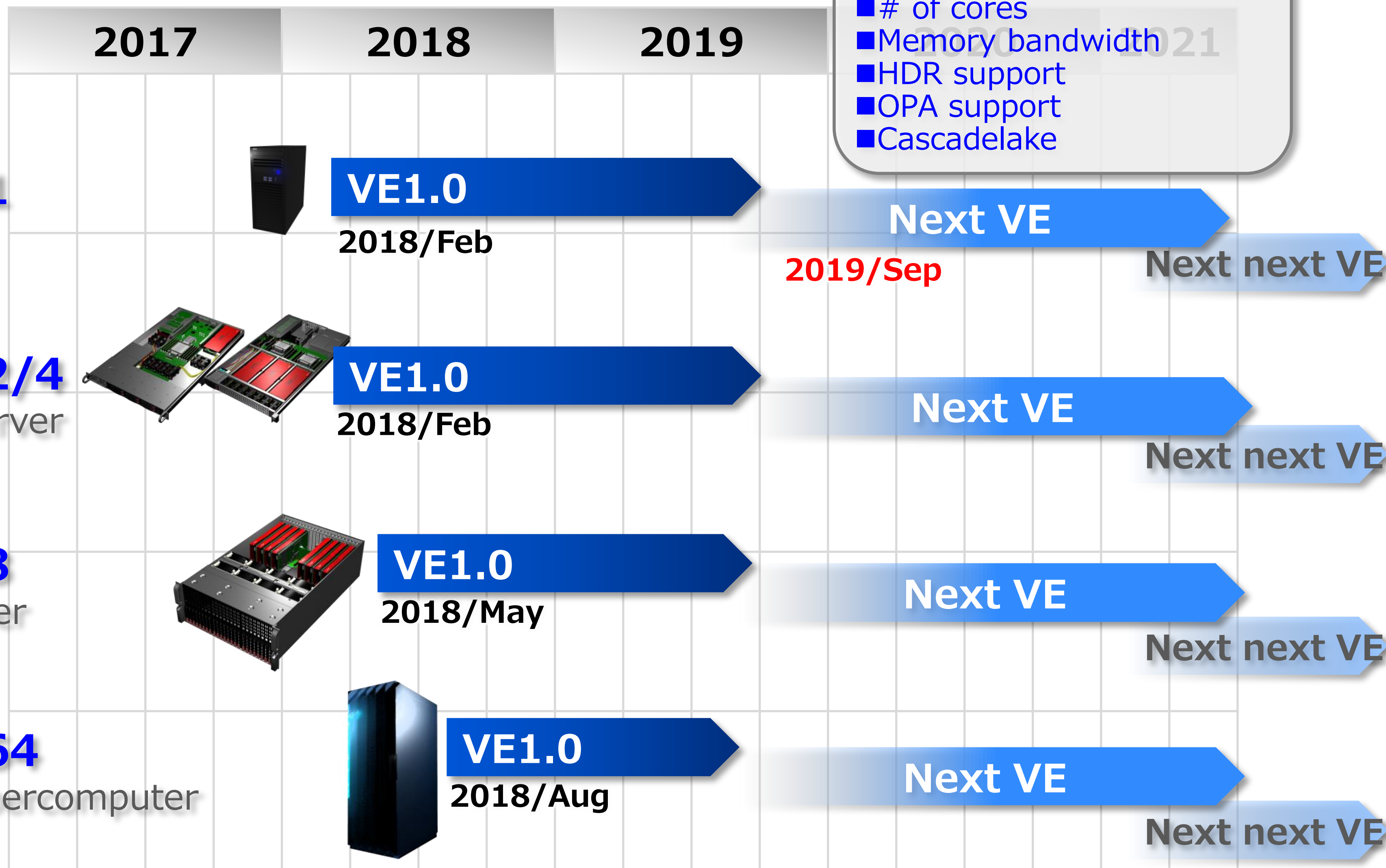


- NEC tested CT Program with Model-Based Iterative Reconstruction (MBIR) algorithm
- NEC compared VE performance to Gold 6126(2.6GHz)
- Test Program is OSS provided by Purdue University

© NEC Corporation 2018

Roadmap

Hardware Roadmap



 © NEC Corporation 2018 Confidential \Orchestrating a brighter world NEC

Next VE

VE1.0

<table>
<tr><th rowspan="2">VE Type</th><th rowspan="2">Freq. (GHz)</th><th>core</th><th colspan="5">processor</th></tr>
<tr><th>GF</th><th>cores</th><th>DP TF</th><th>TB/s</th><th>GB</th></tr>
<tr><td>Type 10A</td><td>1.6</td><td>307</td><td>8</td><td>2.45</td><td rowspan="2">1.22</td><td rowspan="2">48</td></tr>
<tr><td>Type 10B</td><td rowspan="2">1.4</td><td rowspan="2">269</td><td rowspan="2">8</td><td rowspan="2">2.15</td></tr>
<tr><td>Type 10C</td><td>0.75</td><td>24</td></tr>
</table>

Next VE Target spec

VE Type	Freq. (GHz)	core	processor			
		GF	cores	DP TF	TB/s	GB
Type 15A	1.6	307	10	3.07	1.50	48
Type 15B	1.6	307	8	2.45	1.50	48
Type 15C	1.6	307	8	2.45	0.75	24

© NEC Corporation 2018

\Orchestrating a brighter world
NEC