

OpenMP on the NEC SX-Aurora Vector Engine

Erich Focht, Simon Moll, NEC Deutschland

OpenMPCon 21

Vector Engine Card

Air Cooled Card

- Two types of packages

Passive Cooling

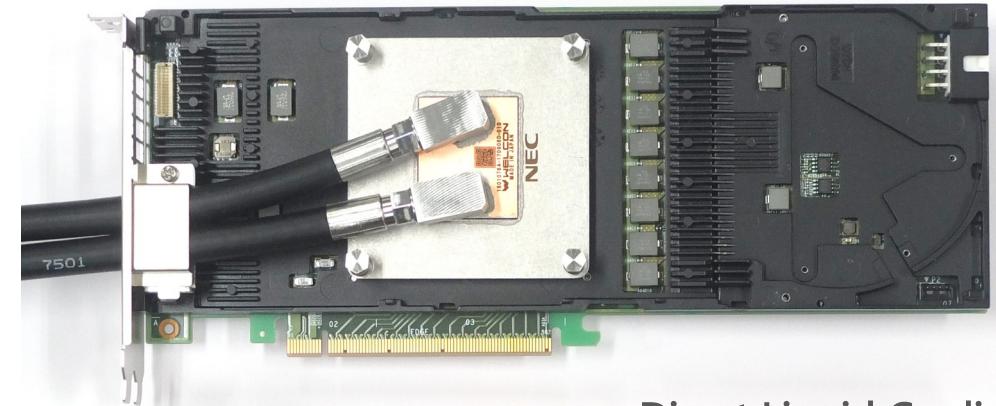
For Servers



Active Cooling
For Tower/Workstation

Water Cooled Card

- Direct liquid cooling
- Hot water cooling available



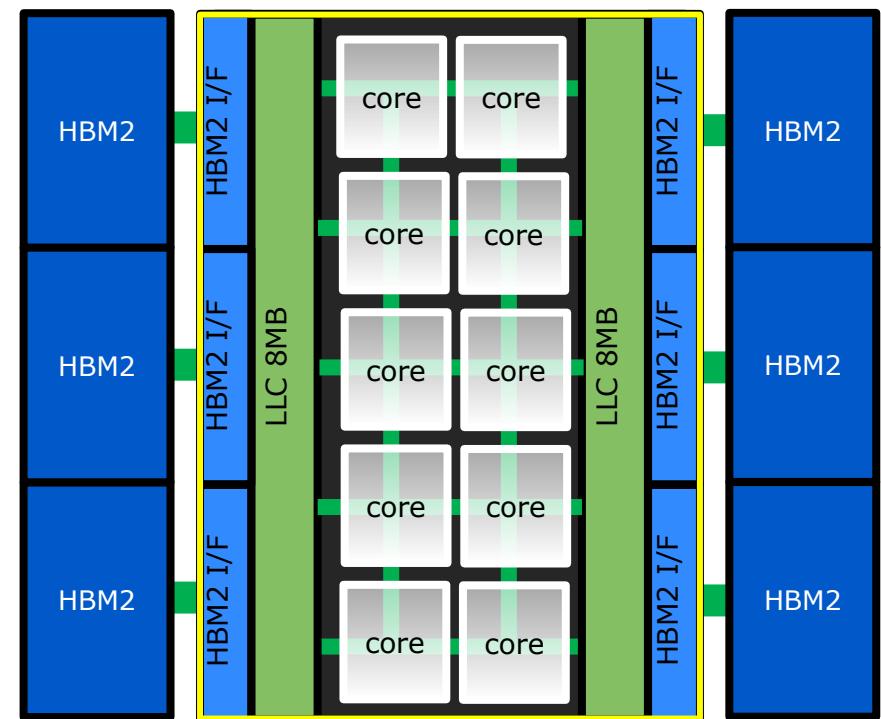
40°C/104°F water

Direct Liquid Cooling
For Supercomputer

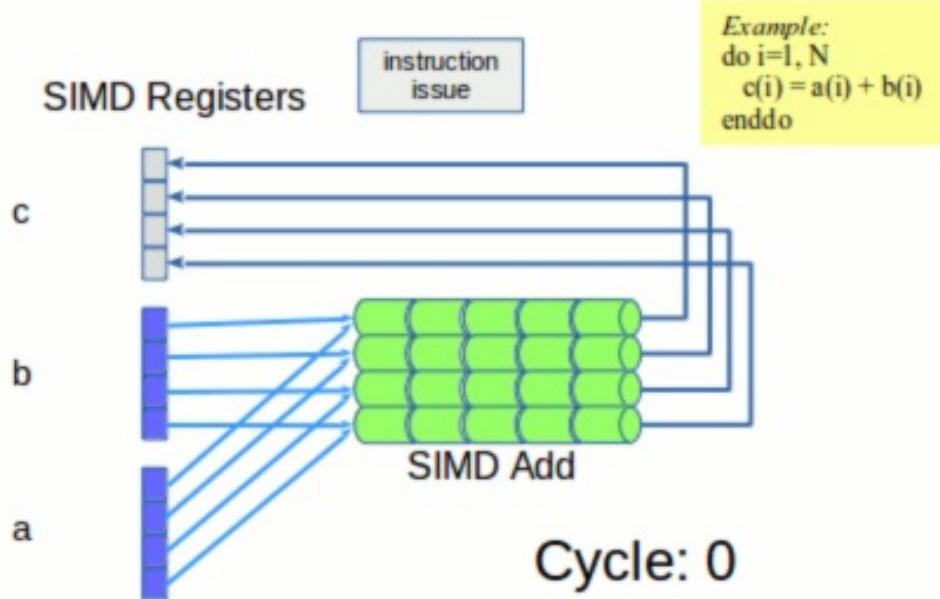
VE20 Processor

VE20 Specifications

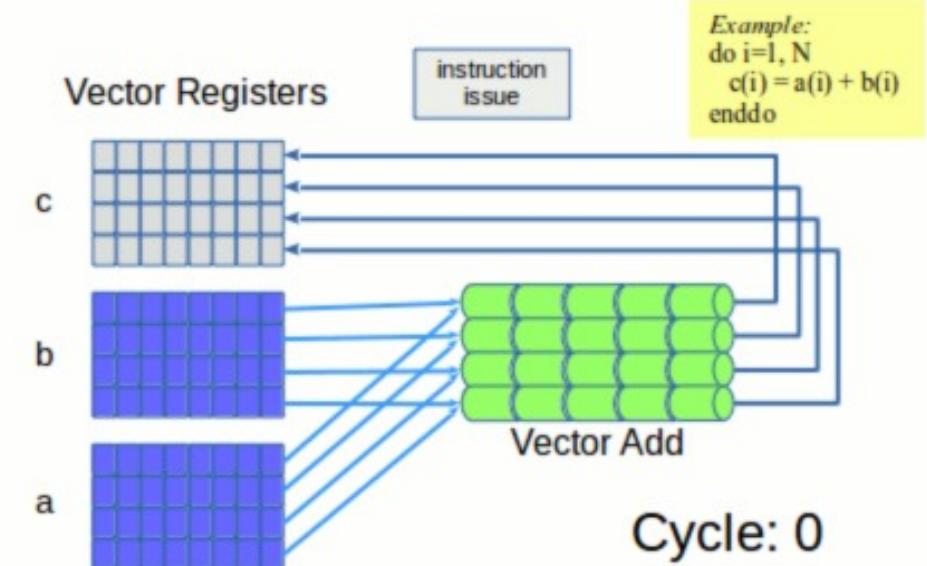
Processor Version	Type 20A	Type 20B
Cores/processor	10	8
Core performance	307GF (DP) 614GF (SP)	
Processor performance	3.07TF (DP) 6.14TF (SP)	2.45TF (DP) 4.91TF (SP)
Cache capacity	16MB	
Cache bandwidth	3TB/s	
Cache Function	Software Controllable	
Memory capacity	48GB	
Memory bandwidth	1.53TB/s	
Power	~300W (TDP) ~200W (Application)	



Long Vector = SIMD + Pipelining



SIMD



Vector

Vector Engine Core

Vector Processing Unit (VPU)

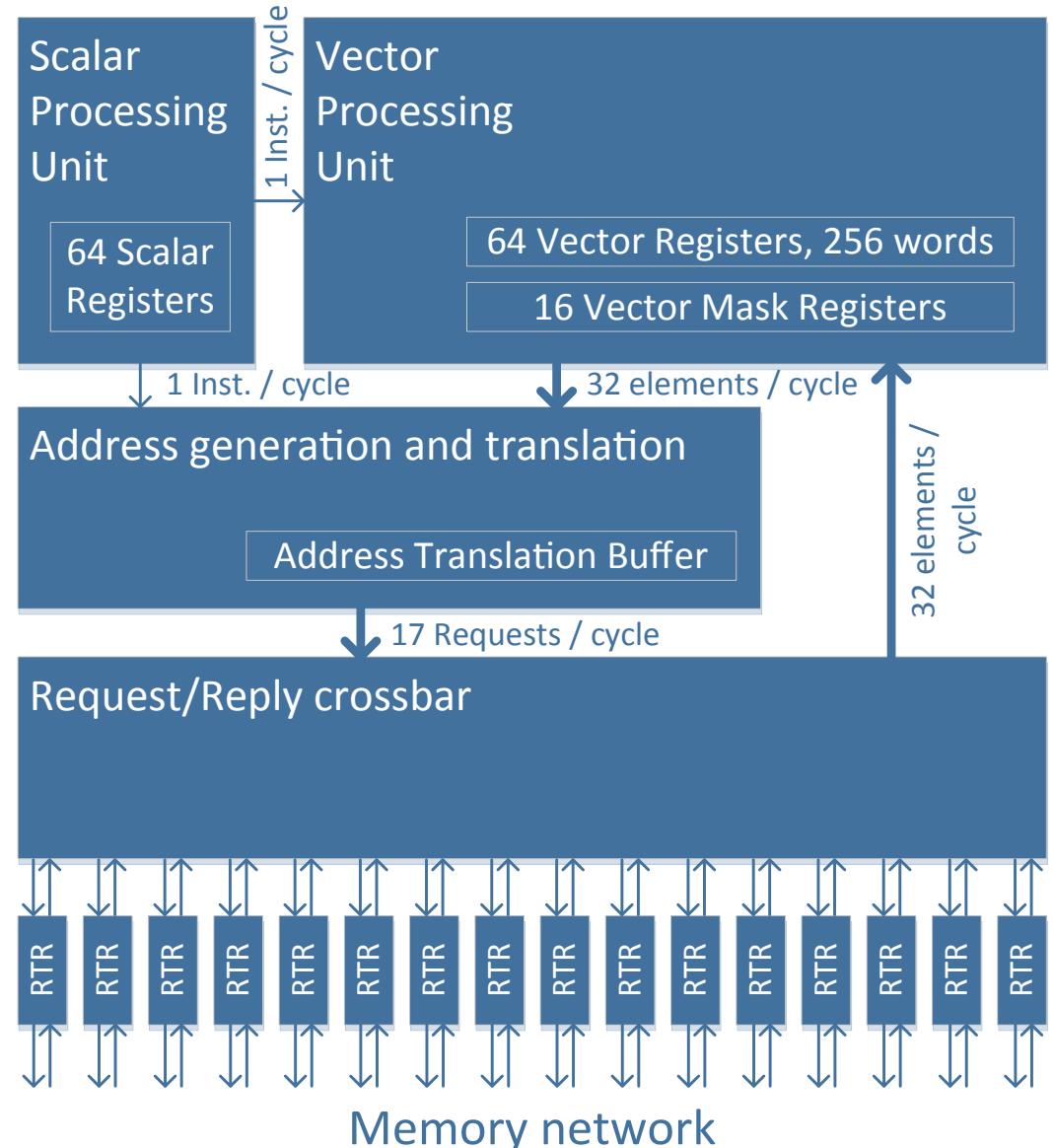
- Powerful
 - 307.2GFLOPS DP / 614.4GFLOPS SP performance
- High bandwidth memory access
 - 409.6GB/sec Load and Store

Scalar Processing Unit (SPU)

- Provides basic processor functions
 - Fetch, decode, branch, add, exception handling, etc...
- Controls the status of complete core

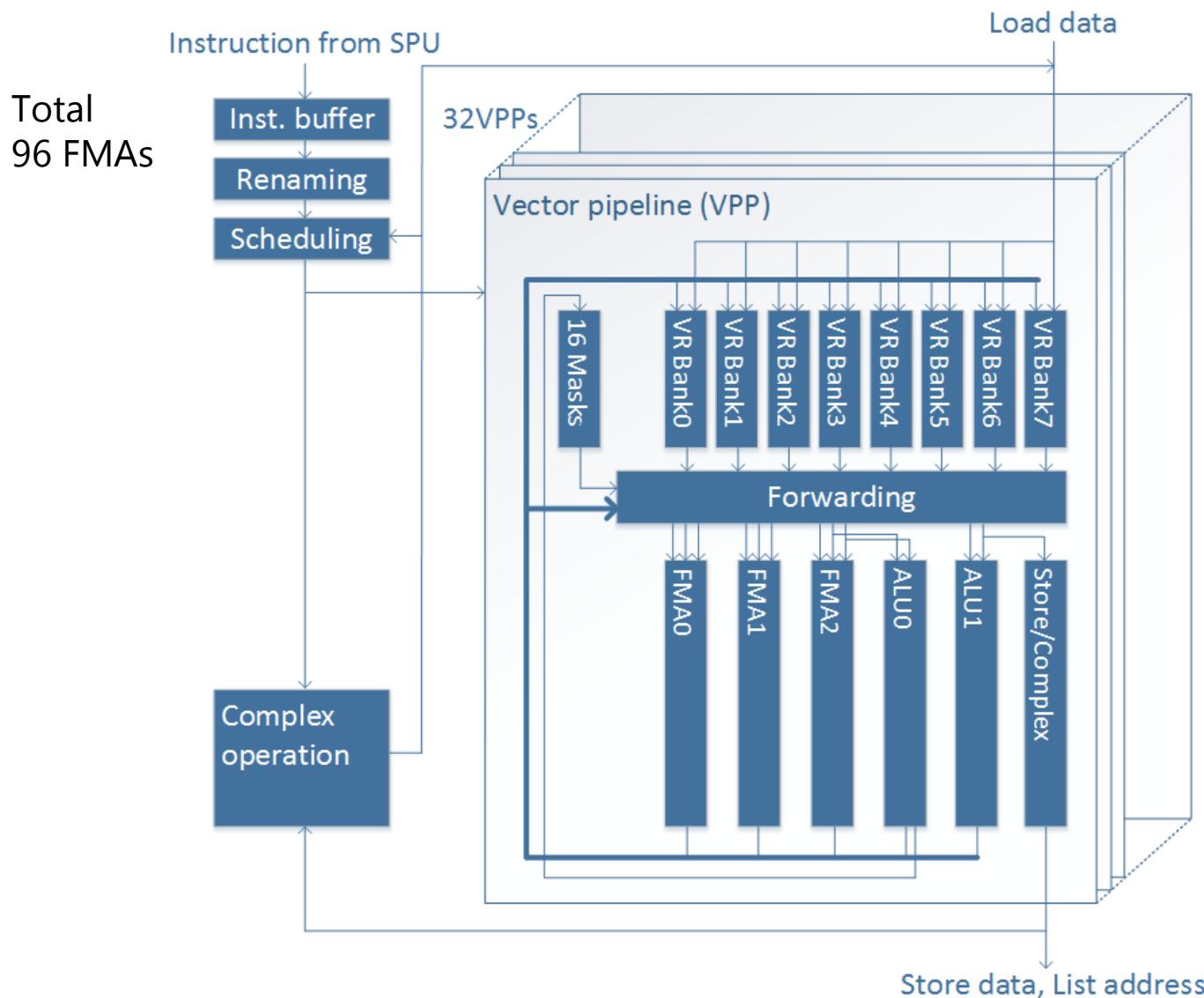
Address translation and data forwarding crossbar

- To support contiguous vector memory access
 - 16 elements/cycle vector address generation and translation, 17 requests/cycle issuing
 - 409.6GB/sec load and 409.6GB/sec store data forwarding



Vector Processing Unit

- Four pipelines, each 32-way parallel
 - FMA0: FP fused multiply-add, integer multiply
 - FMA1: FP fused multiply-add, integer multiply
 - ALU0/FMA2: Integer add, multiply, mask, FP FMA
 - ALU1/Store: Integer add, store, complex operation
- Doubled SP performance by 32bit x 2 packed vector data support
- Vector register (VR) renaming with 192 physical VRs
 - 64 architectural VRs are renamed
 - Enhanced preload capability
 - Avoidance of WAR and WAW dependencies
- OoO scheduling
- Dedicated complex operation pipeline to prevent pipeline stall
 - Vector sum, divide, mask population count, etc.



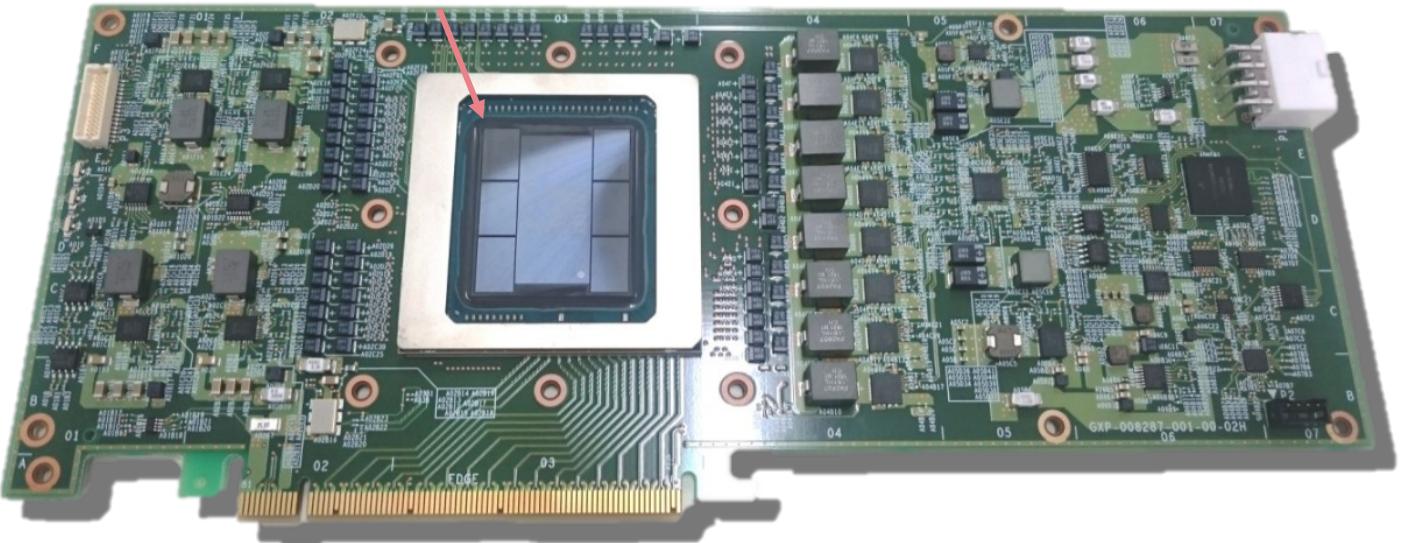
Vector Engine Card

Standard PCIe card

- Full-length full-height card
- Dual slot
- <300W power

Memory capacity	Memory bandwidth			
	1.53TB/s		20B	20A
48GB	1.35TB/s	10BE*	10AE*	
	1.22TB/s	10B		
24GB	1.00TB/s	10CE		
		2.15TF	2.45TF	3.07TF
Frequency	1.4GHz		1.6GHz	
Cores	8core		10core	

Vector engine processor module



High sustained performance

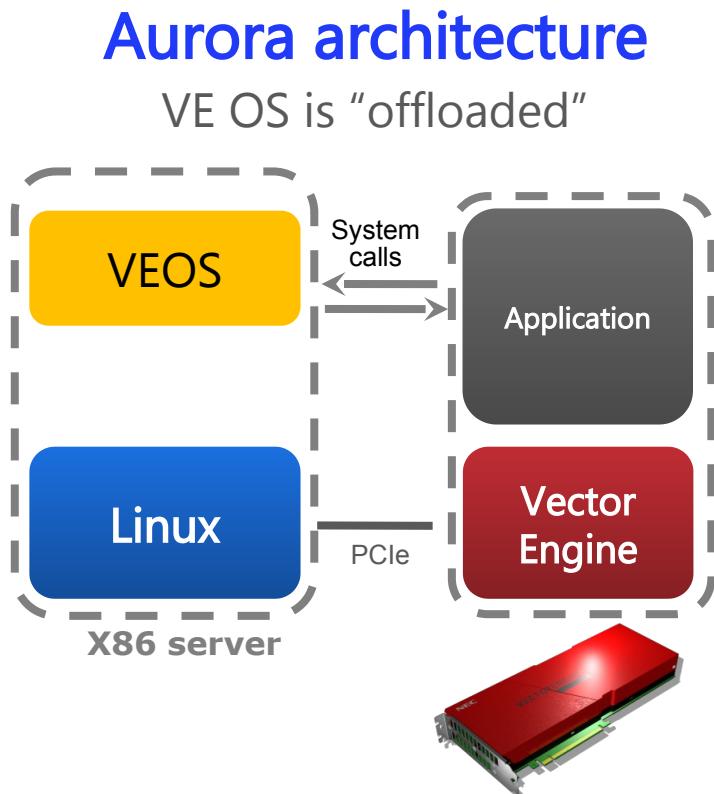
- High B/F ratio (0.62 for VE20B), good balance
- **~6% HPCG performance efficiency**

TCO reduction

- Low power consumption: long vectors
- Increased productivity
(programming, code maintenance)

SX-Aurora TSUBASA: VE Native Mode

- ◆ Programs run on VE in a Linux environment
 - Accelerator that doesn't behave like an accelerator



VEOS features:

- ◆ VE process management
- ◆ VE memory management
- ◆ VE program loading
- ◆ System call handling
- ◆ Signal handling
- ◆ OS commands support
 - gdb, ps, free, top, sar etc.

No OS jitter on Vector Engine
VEOS works completely on Linux/x86.

VE Native Programming Model

◆ C / C++ / Fortran

- With any syscalls
- Proprietary: NEC **ncc**, **nc++**, **nfort**

◆ OpenMP parallelization

- Inside one VE: 8-10 cores
- With NUMA mode: 4-5 cores

◆ MPI

- Infiniband, EDR or HDR
- PeerDirect VE to VE

◆ Hybrid MPI

- Same program compiled for VE and VH, with NEC MPI

NEC ncc/nc++/nfort OpenMP

◆ Compliant to OpenMP 4.5

- With restrictions
 - eg. Cancellation constructs, thread affinity control, array sections, certain clauses
- Environment variables with VE_ prefix take precedence
 - VE_OMP_NUM_THREADS overrides OMP_NUM_THREADS
 - i.e. control threads of host process and VE process separately.

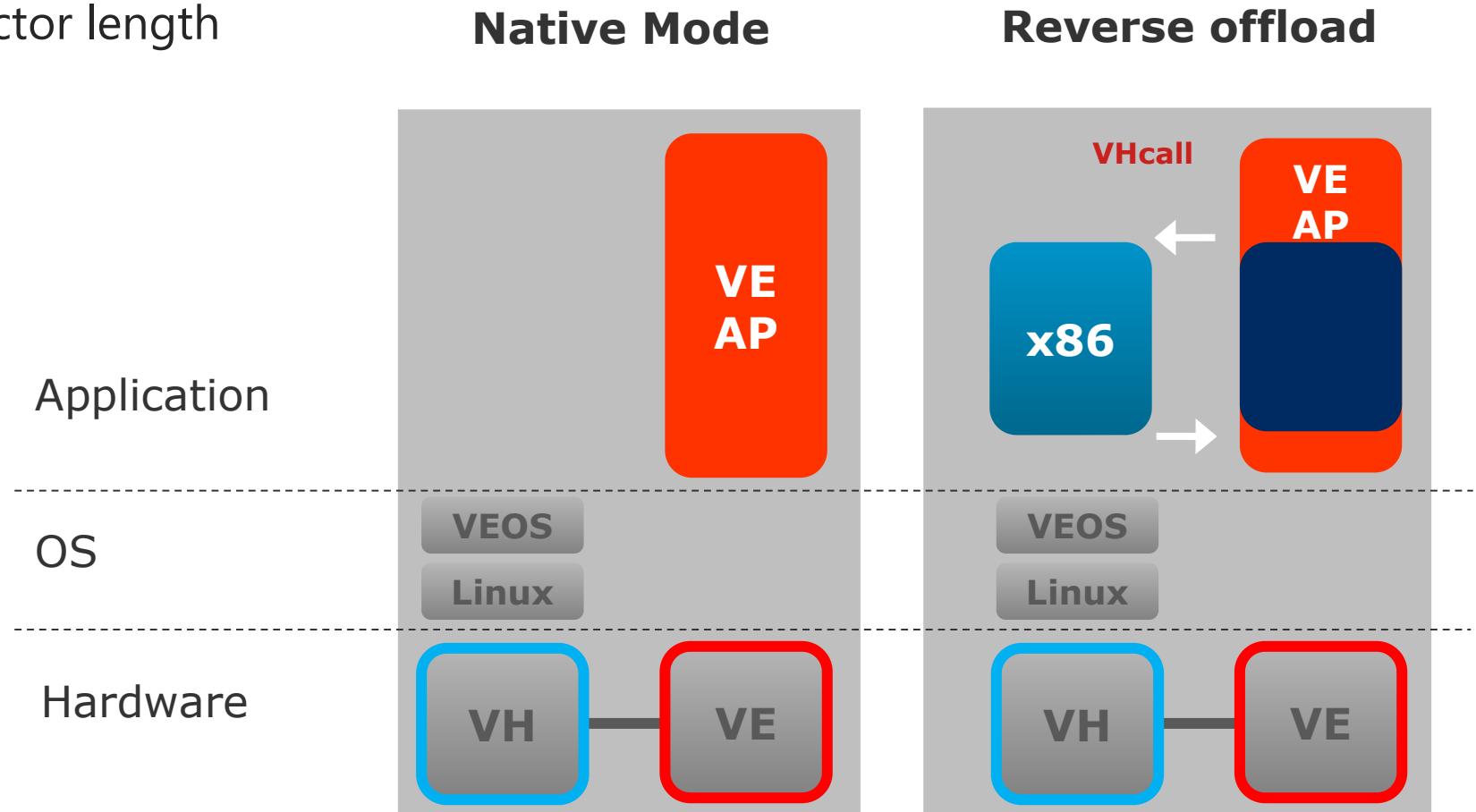
◆ Biggest limitation: no Device Constructs

- No OpenMP Target for accelerator use!
- Remember: VE primarily aimed at native programming
- NEC ncc/nc++/nfort are pure VE cross compilers
 - No code generation capabilities for the host CPU

But... VE Is Actually an Accelerator!

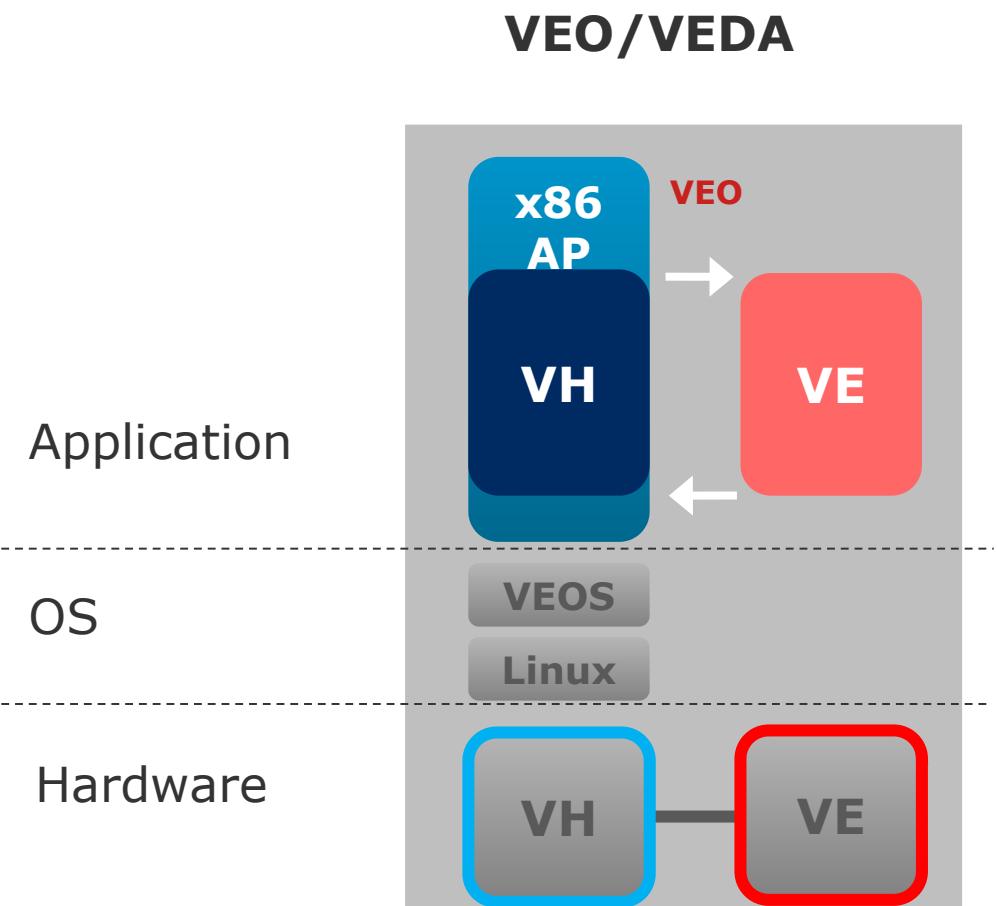
- ◆ While code can be ported as if we'd be using a general purpose CPU

- Some functions won't vectorize
- Or have too short vector length



But... VE Is Actually an Accelerator!

- ◆ Run main program on VH and offload VE kernels to VE...
 - Like OpenCL or CUDA
 - No mechanism available on OS-less device
- ◆ VEOffloading
 - Born 2015, evolved to AVEO in 2020.
- ◆ Core mechanism for accelerator style programming
- ◆ VEDA: Vector Engine Device API
 - Implements CUDA alike API
 - VEDA: <https://docs.nvidia.com/cuda/cuda-driver-api/>
 - VERA: <https://docs.nvidia.com/cuda/cuda-runtime-api/>



OpenMP Target: First Steps

- ◆ Cooperation project with RWTH Aachen
 - Started 2017
 - Tim Cramer, Manoel Römmer, Boris Kosmynin, Marius Behrens. Erich Focht
- ◆ Leveraging LLVM/Clang infrastructure
 - Source-to-source transformation, C language support
 - Target compiler independent
 - Plugin for host – VE communication using VEO

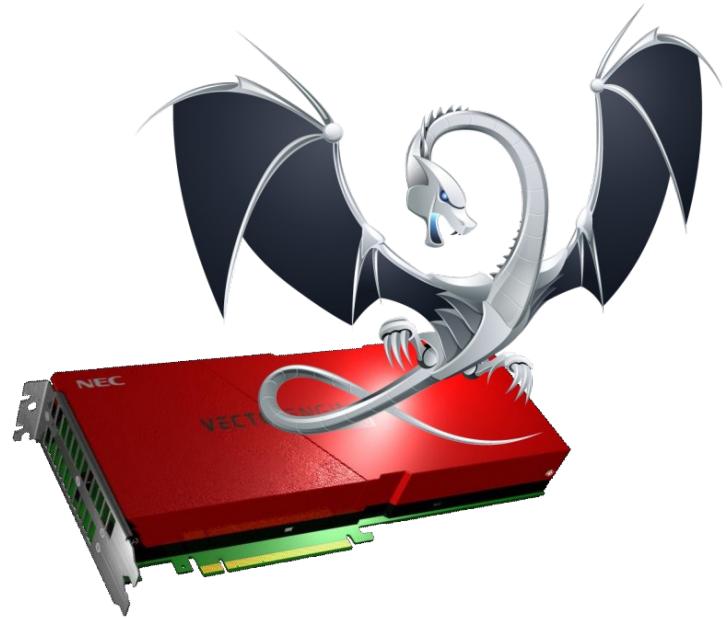


First evaluation results: Parallel Processing and Applied Mathematics, PPAM 2019: OpenMP Target Device Offloading for the SX-Aurora TSUBASA Vector Engine, Tim Cramer et al.

LLVMs of SX-Aurora

◆ LLVM-VE

- Started 2018. Open Source!
- *Inofficial* compiler for SX-Aurora
 - Experimental → upstreaming!
- Highlights
 - Vector Intrinsics
 - Region Vectorizer : outer loop!
 - OpenMP Target

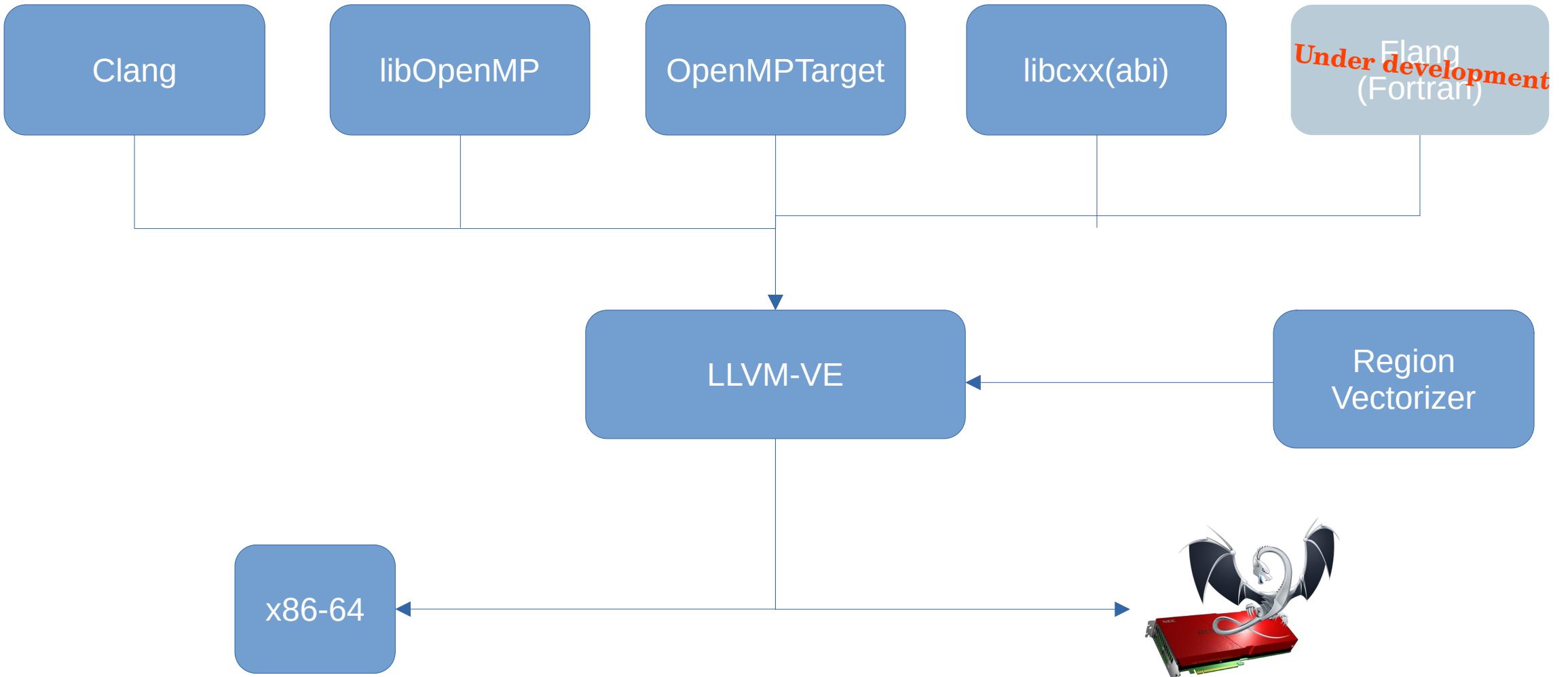


github.com/sx-aurora-dev/llvm-project

◆ llvm-vec NEC LLVM-IR Vectorizer

- Based on proprietary NEC compiler
- First test version: May 2021

LLVM-VE ecosystem



Vectorization

Vectorization in LLVM-VE

- ◆ Clang –target=x86_64-unknown-linux-gnu
 - Uses LLVM upstream vectorizers (LV)
 - Inner loops only
 - Automatic vectorization

- ◆ Clang –target=ve-linux
 - Uses the *Region Vectorizer*
 - Outer loop vectorization
 - Automatic vectorization
 - Best controlled with pragmas

How to vectorize

◆ `#pragma omp simd [simdlen(256) | simdlen(512)]`

- Vectorize this loop
- [Optional] hint for normal (256 wide) or packed mode (512 wide) vectorization

◆ `#pragma omp parallel`

- May trigger vectorization (details next slides)

◆ Some unannotated loops - Automatic vectorization

- Automatic parallel loop detection and vectorization

Region Vectorizer - Outer-loop vectorization

```
#pragma omp simd
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        if (A[i] > 42.0)
            <do stuff>
        if (C[j])
            <do that other thing>
```

Vanilla LLVM cannot vectorize this:

- ◆ Outer loop
- ◆ Control flow (`if` statements) inside

Region Vectorizer can

- ◆ Will retain uniform branch in `C[j]`

Controlling Vectorization

```
#pragma omp parallel for  
for (int i = 0; i < n; ++i)  
[ . . ]  
  
#pragma omp simd  
for (int j = 0; j < m; ++j)  
[ . . ]
```

Parallel execution

Vectorized

Controlling Vectorization

```
#pragma omp parallel for  
for (int i = 0; i < n; ++i)  
[.]
```

Parallel execution

```
for (int j = 0; j < n; ++j)  
[.]
```

May still vectorize, if

- ◆ Loop parallelism detected
- ◆ Better score than pragma parallel loop

Adaptive math vectorization

```
void
foo (double x, ...) {
    #pragma omp simd
    for (int i = 0; i < n; ++i)
        A[i] = pow(B[i], x);
}
```

double pow(double, double) → double pow_vu(double256, double)

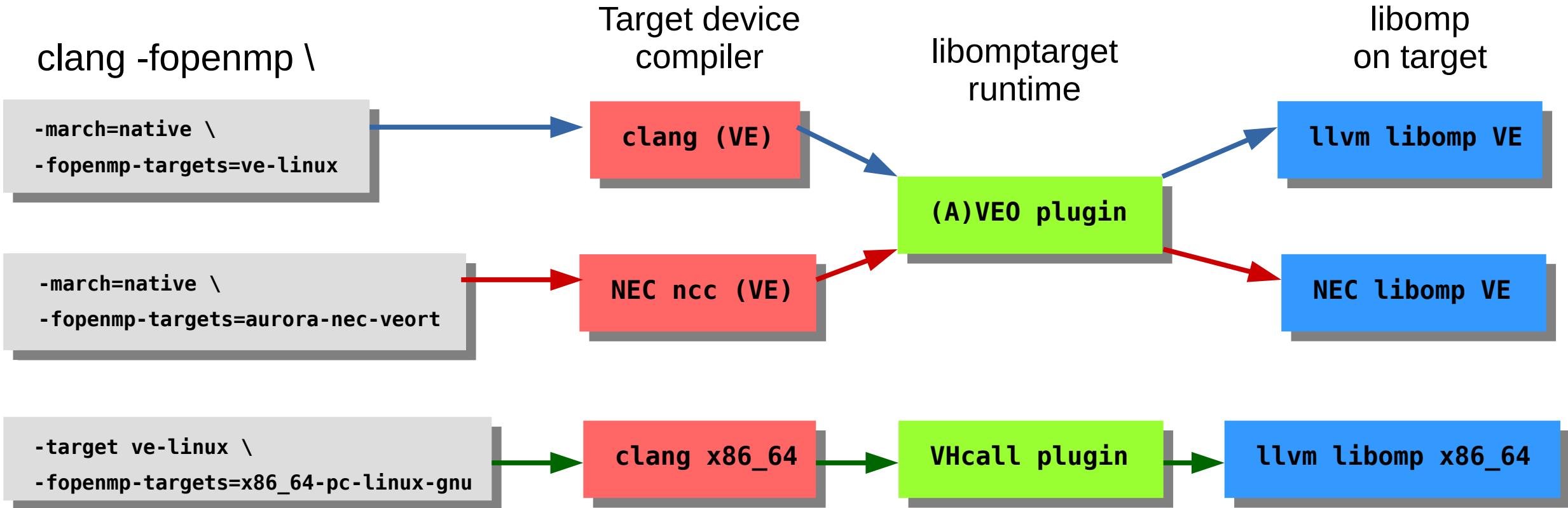
OpenMP [Target]

OpenMP Target for VE

```
int data[N];  
  
#pragma omp target data map(tofrom: data)  
{  
    #pragma omp target  
    for (int i = 0; i < N; i++)  
        data[i] *= 2;  
  
    #pragma omp target update from(data)  
    // ...  
  
    #pragma omp target  
    for (int i = 0; i < N; i++)  
        data[i] += 5;  
}
```

- Where does main() run?
 - Vector host (VH)
 - Vector engine (VE)
- Which compiler
 - For offloaded part
 - For main() part

OpenMP Target for VE



LLVM OpenMP

- libOpenMP `#pragma omp parallel`
 - Implements parallel loops, barriers and reductions
 - Linked against the device code
 - Either as part of VE-native application
 - Or VE-native kernels
- libOpenMPTarget `#pragma omp target`
 - Performs the kernel dispatch, buffer transfers
 - Linked against the host application
 - Plugin mechanism for actual offload

LLVM OpenMP

- Generic LLVM OpenMP library compiled for VE

```
#pragma omp parallel
```

- Pro: Mature OpenMP implementation

Standard OpenMP runtime for x86 for Clang/LLVM

- Con: Not tuned for vector architectures

Based on pthreads, standard synchronization primitives, not hw features

OpenMP – EPCC Syncbench

[μs]	OpenMP (ncc)	LLVM OpenMP (VE)	LLVM OpenMP (x86)
parallel for	6.77	724.4	7.27
barrier	3.74	309.8	1.87
reduction	7.01	608.5	7.5
NEC OpenMP is fast		LLVM OpenMP needs tuning	

LLVM OpenMPTarget

```
#pragma omp target
```

- AVEO plugin
 - VH → VE Offloading
- VHCall plugin
 - VE → VH Offloading
- SOLLVE OpenMP Target Verification suite

OpenMP Target - SOLLVE C

	VH → VE	VE → VH	VH → VE (sotoc)
Compile Error	0	0	7
Runtime Error	11	8	11
Passed	98	101	91
Conformant LLVM code path		Custom source-to-source	

OpenMP Target - SOLLVE C++

	VH → VE	VE → VH	VH → VE (sotoc)
Compile Error	0	0	14
Runtime Error	1	2	0
Passed	13	12	0
Conformant LLVM code path		Source-to-source for C only	

OpenMP Target: SPEC ACCEL

Benchmark	x86→SX (NEC 3.0.8)	x86→V100 (LLVM 12)	x86→V100 (GCC 9)	x86→x86	SX→x86
503.postencil	21.3*	16.8	145	103	137
504.polbm	18.8	36.7	60.1	RE	94.4
514.pomriq	321*	31.4	RE	11822	11976
552.pep	1923	71.1	140	639	667
554.pcg	91.6	88.6	80.2	124	240
557.pcsp	81.6	101	232	167	253
570.pbt	19.6	486	122	114	154

Conclusion

- ◆ Vector Engine is different from other accelerators
 - Can behave like a normal CPU
 - Normal loop based programming in standard languages
- ◆ OpenMP is an important tool for parallelizing inside one VE
- ◆ LLVM is crucial for OpenMP device offloading
 - Two approaches, two target compilers
 - Two offload directions
- ◆ LLVM libomp for VE needs tuning

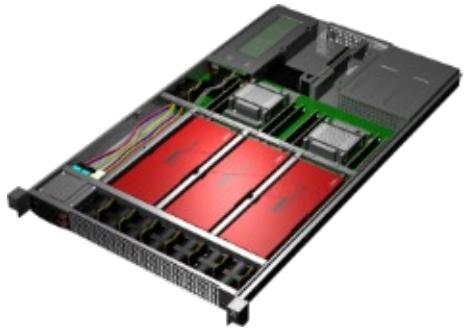
\Orchestrating a brighter world

NEC

NEC SX-Aurora TSUBASA : From tower model to DLC



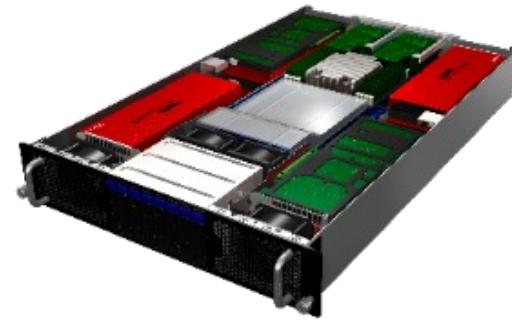
Tower
1VE



Rackmount
4VE/1U



Rackmount
8VE/4U



DLC rackmount
8VE/2U



Built with SX-Aurora Vector Engine

- ◆ NEC vector processor
- ◆ Long vector units: 256 * 64 bit
- ◆ 32-way SIMD and 8 cycle deep pipelining
- ◆ Huge memory bandwidth

Vector Engine Based Heterogeneous HPC Examples

NIFS : Fusion Science
National Institute for Fusion Science



Deutscher Wetterdienst : Weather / Climate
Wetter und Klima aus einer Hand



Tohoku University : Academic



JAMSTEC : Earth Science



Photo of the previous system