Evaluating and exploiting the potential of the 2nd-generation SX-Aurora TSUBASA

Cyberscience Center, Tohoku University

Hiroyuki Takizawa

<takizawa@tohoku.ac.jp>

Outline

- **Performance Evaluation**
  - 2nd generation SX-Aurora TSUBASA (SX-AT)

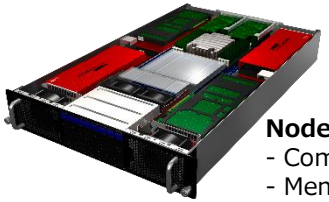
- **Ongoing Research Projects for SX-AT**
  - neoSYCL
    - A SYCL implementation for SX-Aurora TSUBASA
  - Xevolver-C
    - A domain-specific language to describe vectorization-aware code tuning in C

AOBA

System Configuration

Subsystem AOBA-A

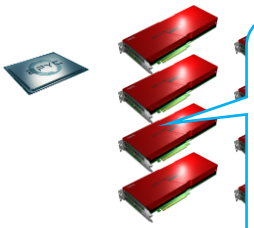
NEC SX-Aurora TSUBASA B401-8 x 72



Node Performance
- Comp. : **20.7 TF**
- Mem size : 640 GB
- Mem BW : **12.4 TB/s**

2nd generation

AMD EPYC 7402P x 1
NEC Vector Engine Type 20B x 8



Vector Engine Core x 8
HBM2 Module x 6

Subsustem AOBA-B

NEC LX 406Rz-2 x 68

Node Performance
- Comp. : **4.1 TF**
- Mem size : 256 GB
- Mem BW : **0.41 TB/s**

AMD EPYC 7702P x 2

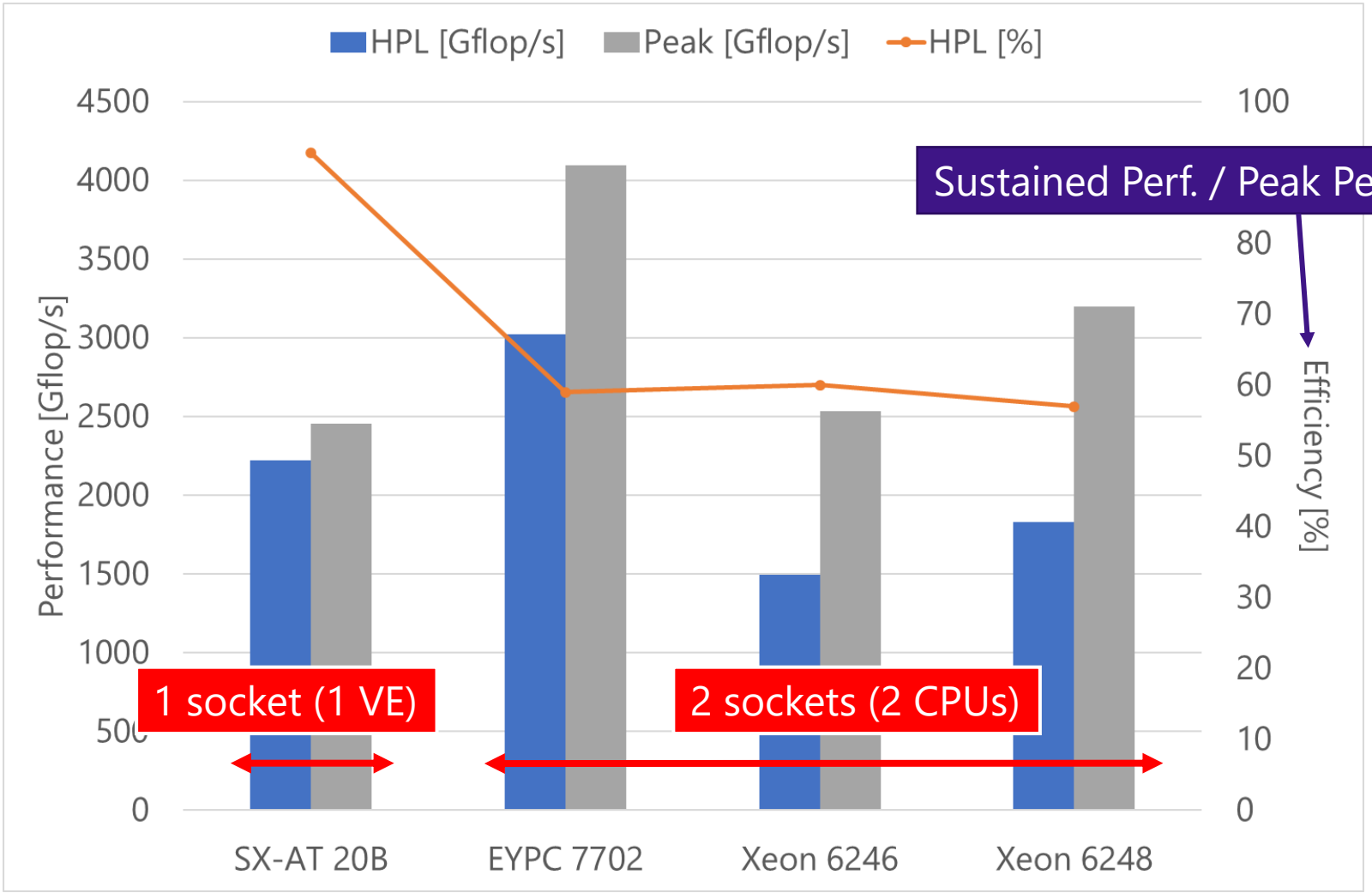
Overall System Performance: 1.78 Pflop/s, 924 TB/s

Evaluation Setup

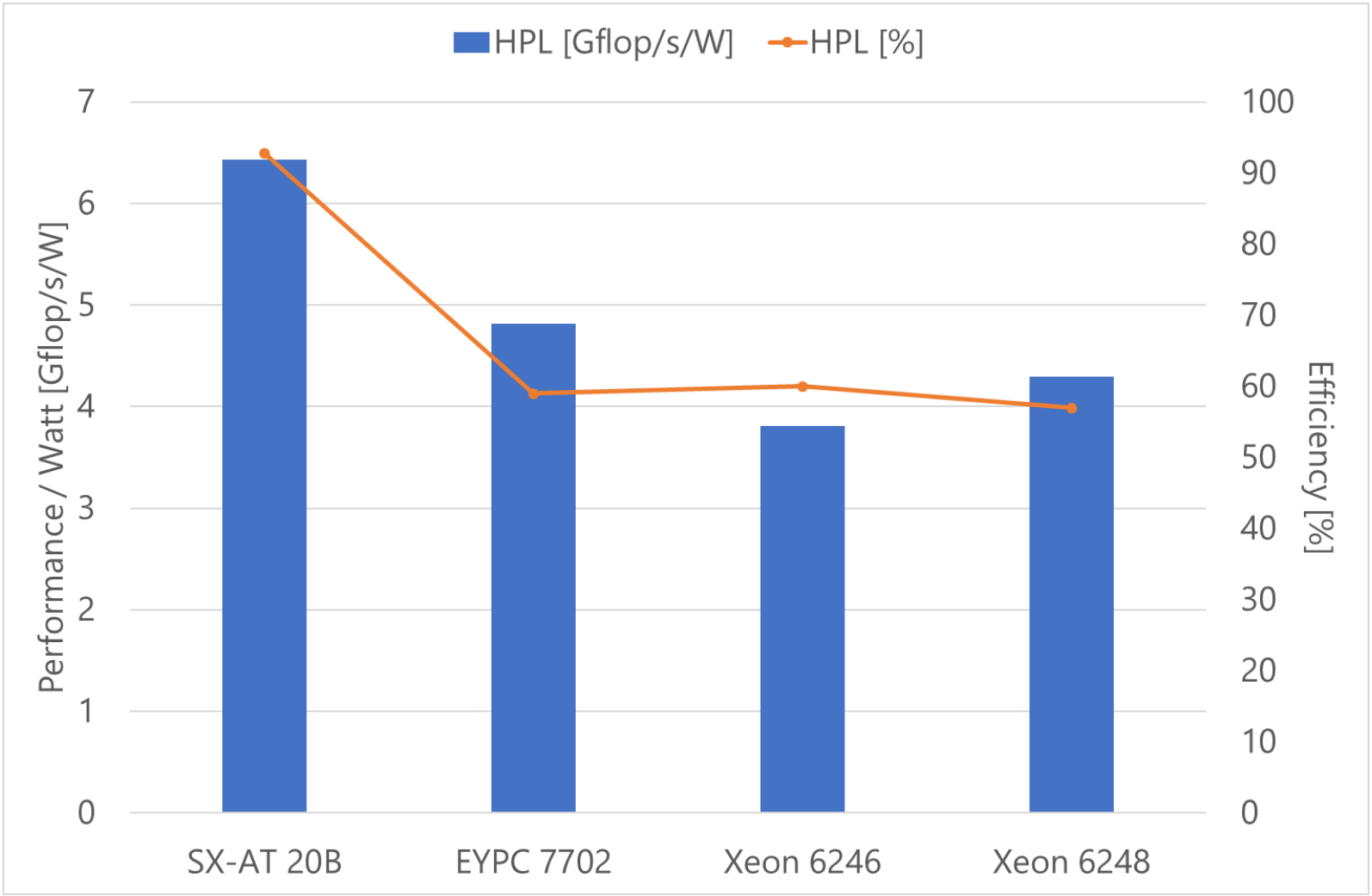
		SX-ACE	SX-AT 10B	SX-AT 10AE	SX-AT 20B	EPYC 7702	Xeon 6246	Xeon 6248
Core	Clock Freq. [GHz]	1	1.4	1.58	1.6	2	3.3	2.5
	Peak Perf. [Gflop/s]	64	268	304	307	32	105	80
Socket	No. of Cores	4	8	8	8	64	12	20
	Peak Perf. [Gflop/s]	256	2150	2433	2457	2048	1267	1600
	Mem BW [GB/s]	256	1228	1352	**1536**	204	140	140
Node	No. of Sockets	1	1	1	1	2	2	2
	Peak Perf. [Gflop/s]	256	2150	2433	2457	4096	2534	3200
	System B/F	1	0.56	0.56	**0.62**	0.1	0.11	0.09

AOBA

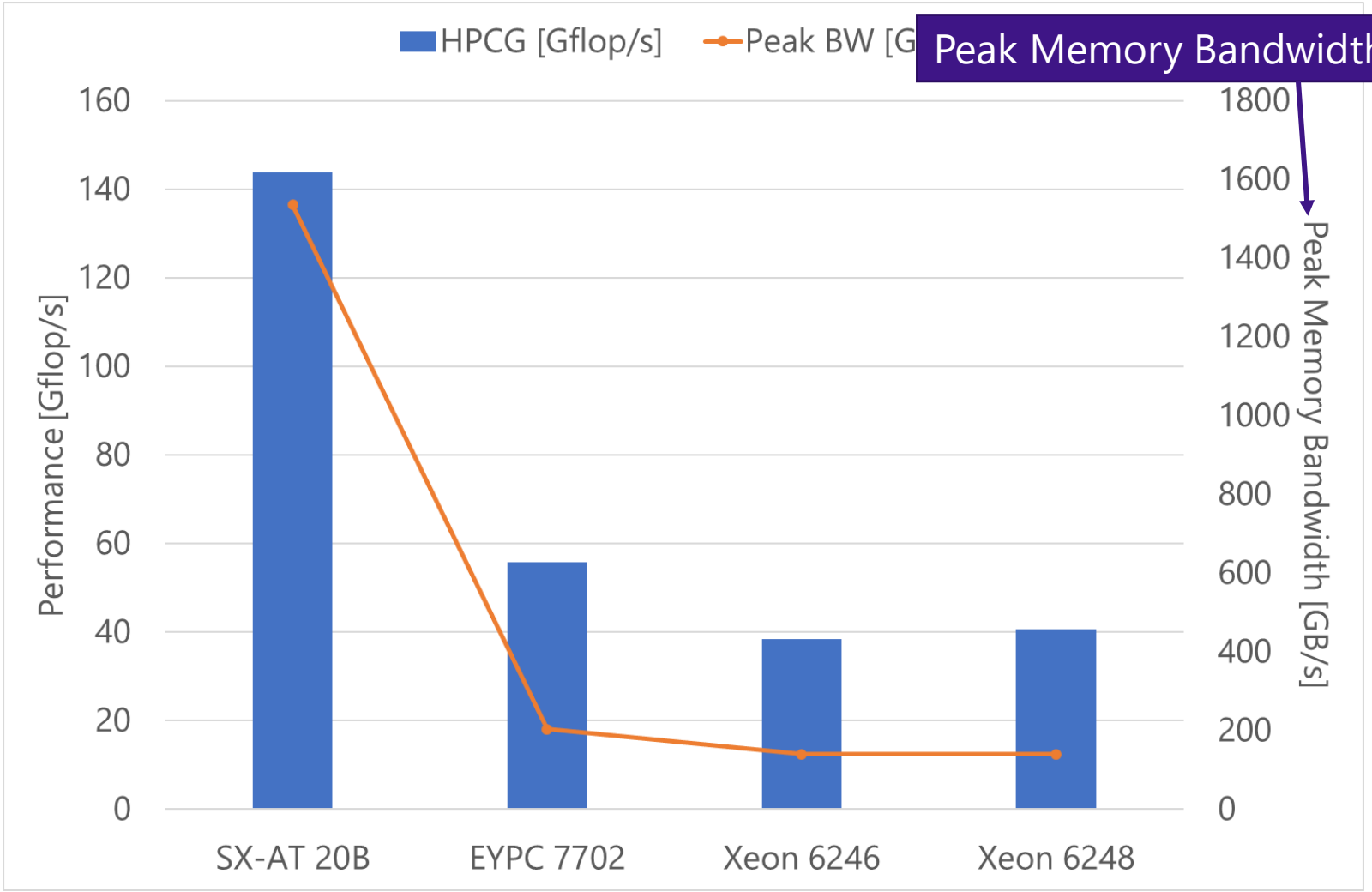
HPL Performance



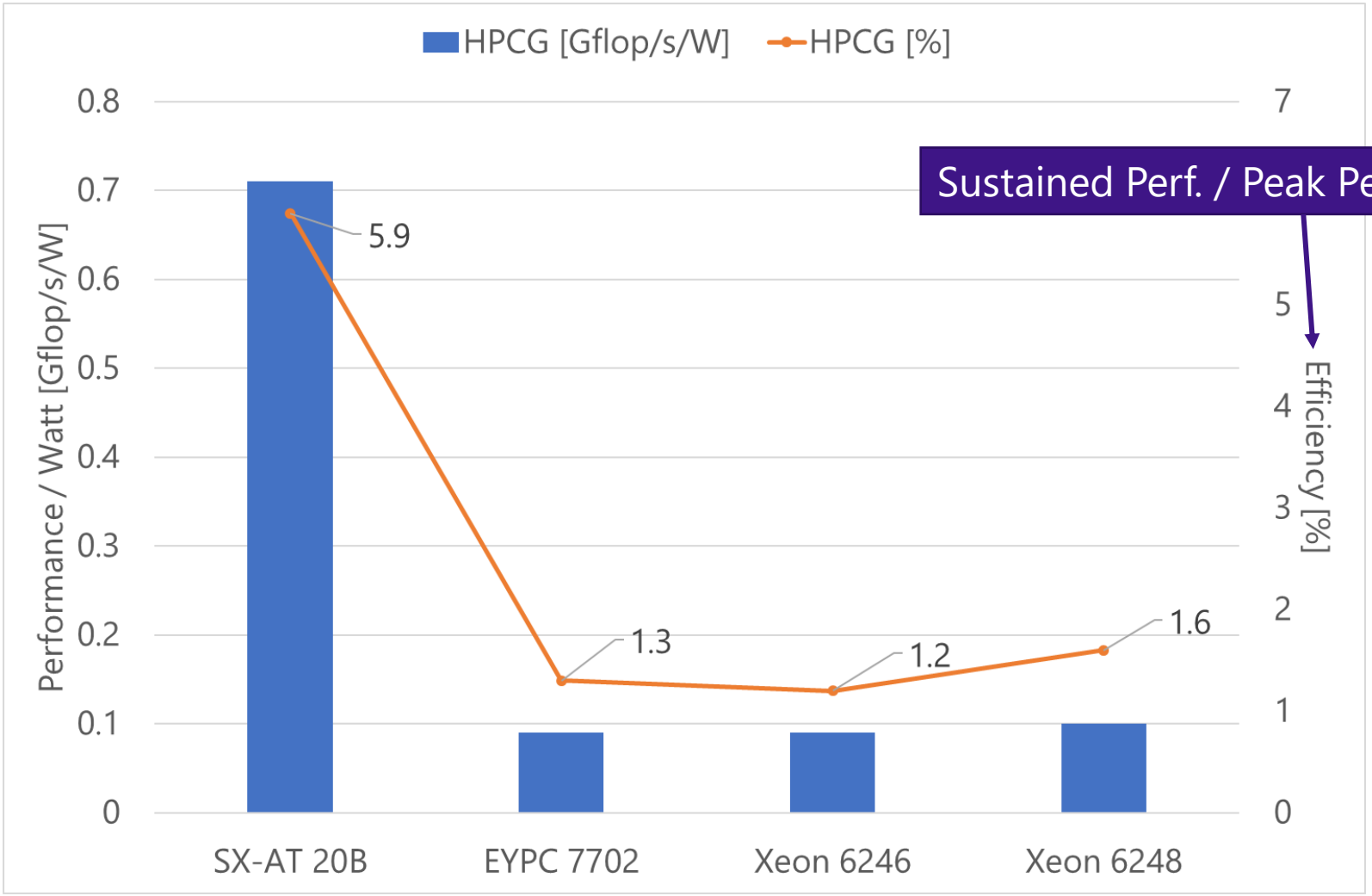
HPL Power Efficiency



HPCG Performance



HPCG Power Efficiency



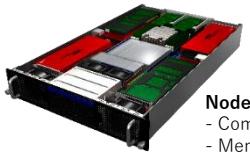
HPCG & HPL on AOBA

Egawa et al.@PMBS20

AOBA

Subsystem AOBA-A

NEC SX-Aurora TSUBASA B401-8 x 72



Node Performance
- Comp. : 20.7 TF
- Mem size : 640 GB
- Mem BW : 12.4 TB/s

【SX-Aurora】
72VH(576VE) Peak 1415.6TF

HPL : 1140.1TF (Efficiency 80.5%)

HPCG : 75.3TF (Efficiency 5.32%)

Subsystem AOBA-B

NEC LX 406Rz-2 x 68

Node Performance
- Comp. : 4.1 TF
- Mem size : 256 GB
- Mem BW : 0.41 TB/s

【AMD Rome 7702】
68node Peak 278.5TF

HPL : 161.7TF (Efficiency 58.08%)

HPCG : 2.55TF (Efficiency 0.92%)

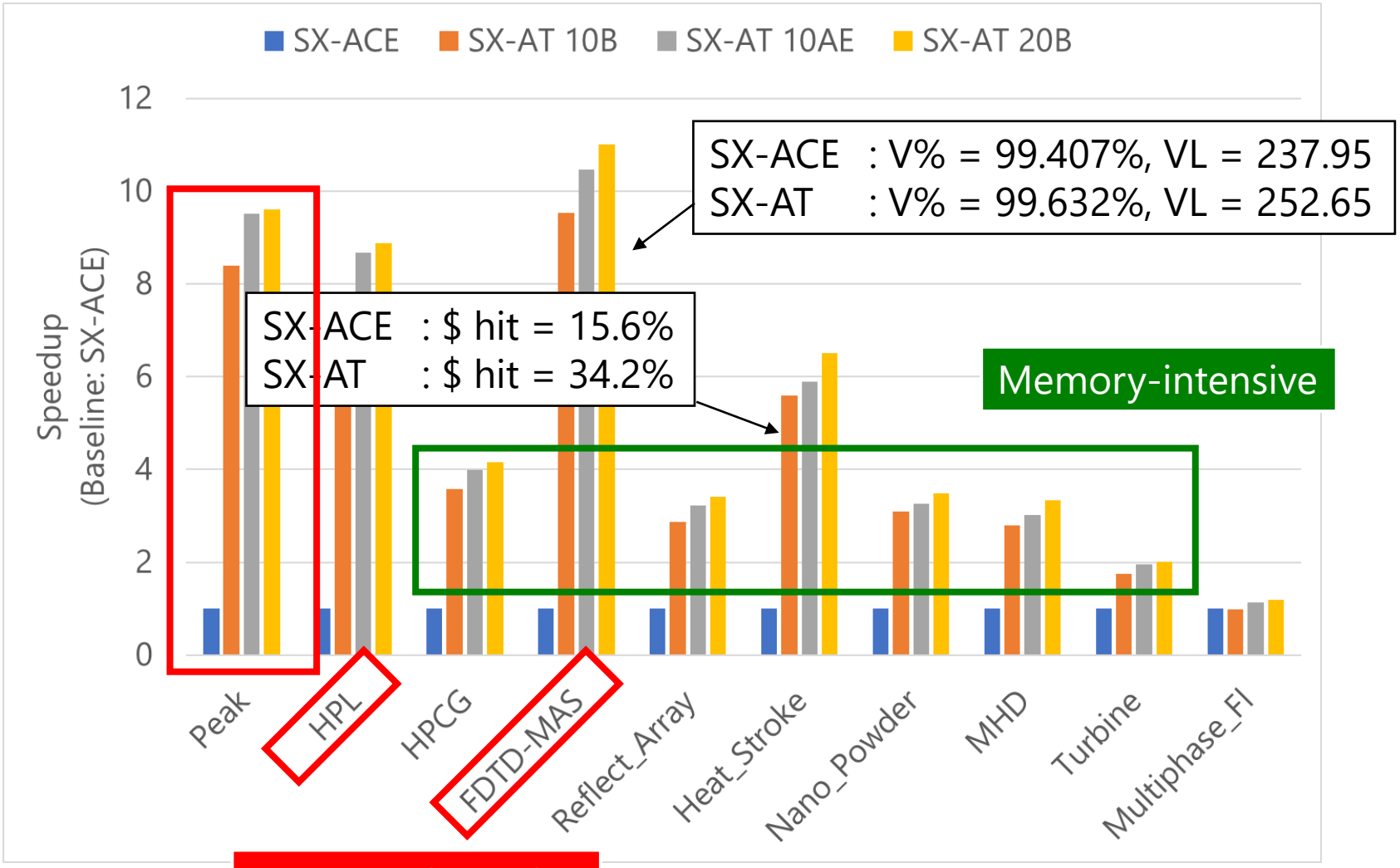
Key Applications

- **FDTD-MAS (Konno et al. 2016)**
  - Analysis of noises on a PCB board
- **Reflext_Array (Kasuga et al. 2005)**
  - Simulation of a reflect array
- **Heat_Stroke (Kojima et al. 2018)**
  - Evaluation of the heat stroke risk
- **Nano_Powder (Shigera et al. 2018)**
  - Simulation of generating silicon nanoparticles
- **MHD (Fukazawa et al. 2016)**
  - Simulation of radiation band electrons around Jupiter
- **Turbine (Yamamoto et al. 2011)**
  - Design of steam and gas turbines
- **Multiphase_Fl (Yoshida 2020)**
  - Simulation of dust explosion phenomena

Application Summary

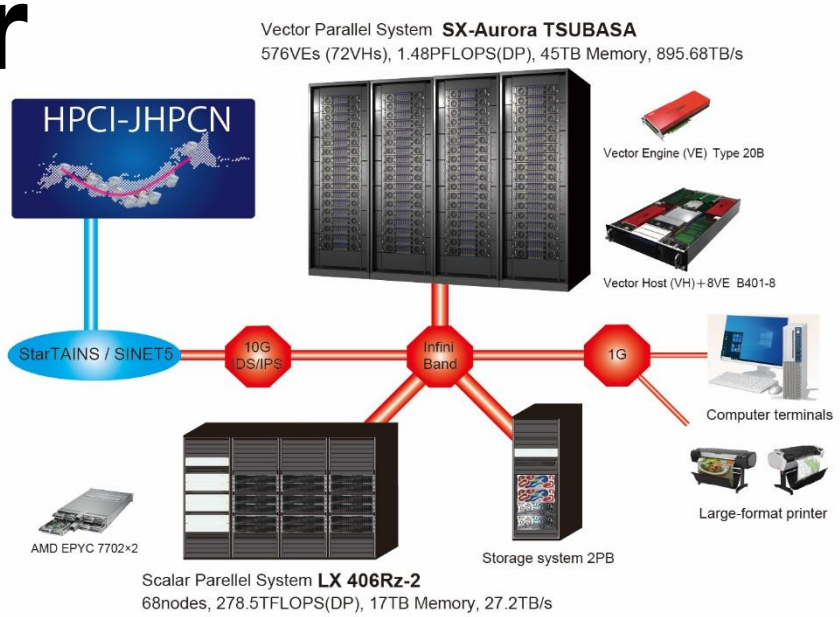
	Vectorization [%]	Ave Vec Len	No. of Processes	No. of VEs	B/F
HPL	99.3	254.4	256	32	0.31
HPCG	99.0	237.5	Compute-intensive	32	11.08
FDTD-MAS	99.6	252.7	1	1	0.29
Reflect_Array	98.8	216.1	8	1	3.74
Heat_Stroke	99.3	214.2	160	20	1.87
Nano_Powder	98.9	255.1	32	32	2.40
MHD	99.6	249.6	Memory-intensive	32	1.33
Turbine	92.0	175.5	159	20	2.52
Multiphase_Fl	29.1	18.0	1	1	12.88

Performance Evaluation Results



Summary so far

■ Supercomputer AOBA



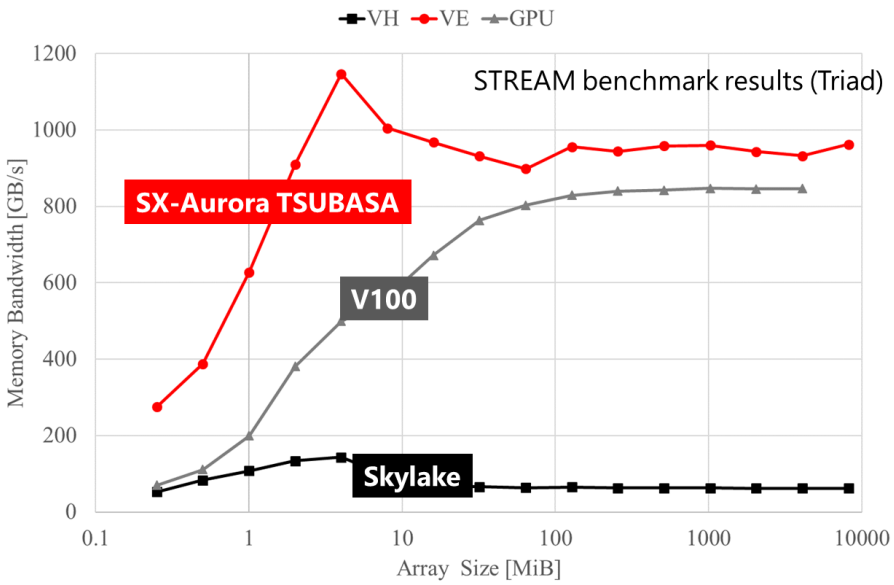
- Start operation on **October 1, 2020**.
- **The world's first system** of **2nd Generation** SX-Aurora TSUBASA
  - Standard Software + Special Hardware
  - High Memory Bandwidth
  - Vector Engine Type 20B
    → Higher B/F Rate = **Higher Comp.&Power Efficiencies**

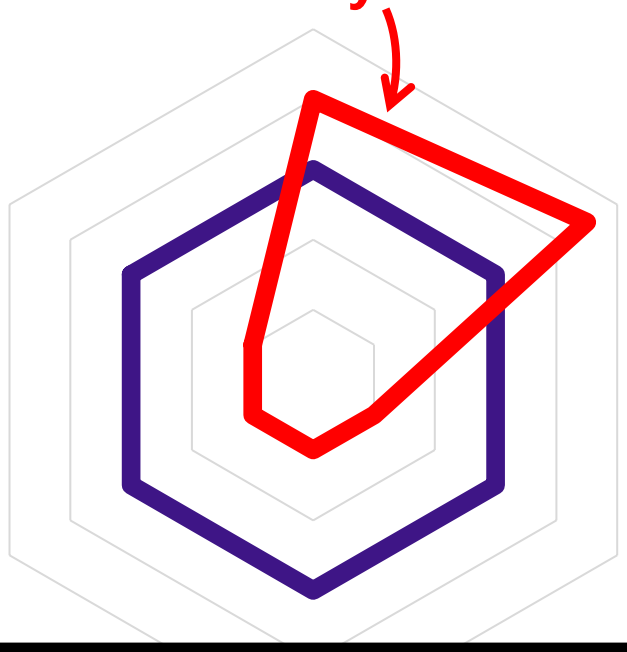
neoSYCL

A SYCL implementation for SX-Aurora TSUBASA

Motivation

- **VH and VE are**
  - Stand-alone, **general-purpose processors**
    - Fully supporting **standard languages**, e.g. C/C++/Fortran
  - Different performance characteristics
    - VE works much faster only for **vector-friendly** workloads





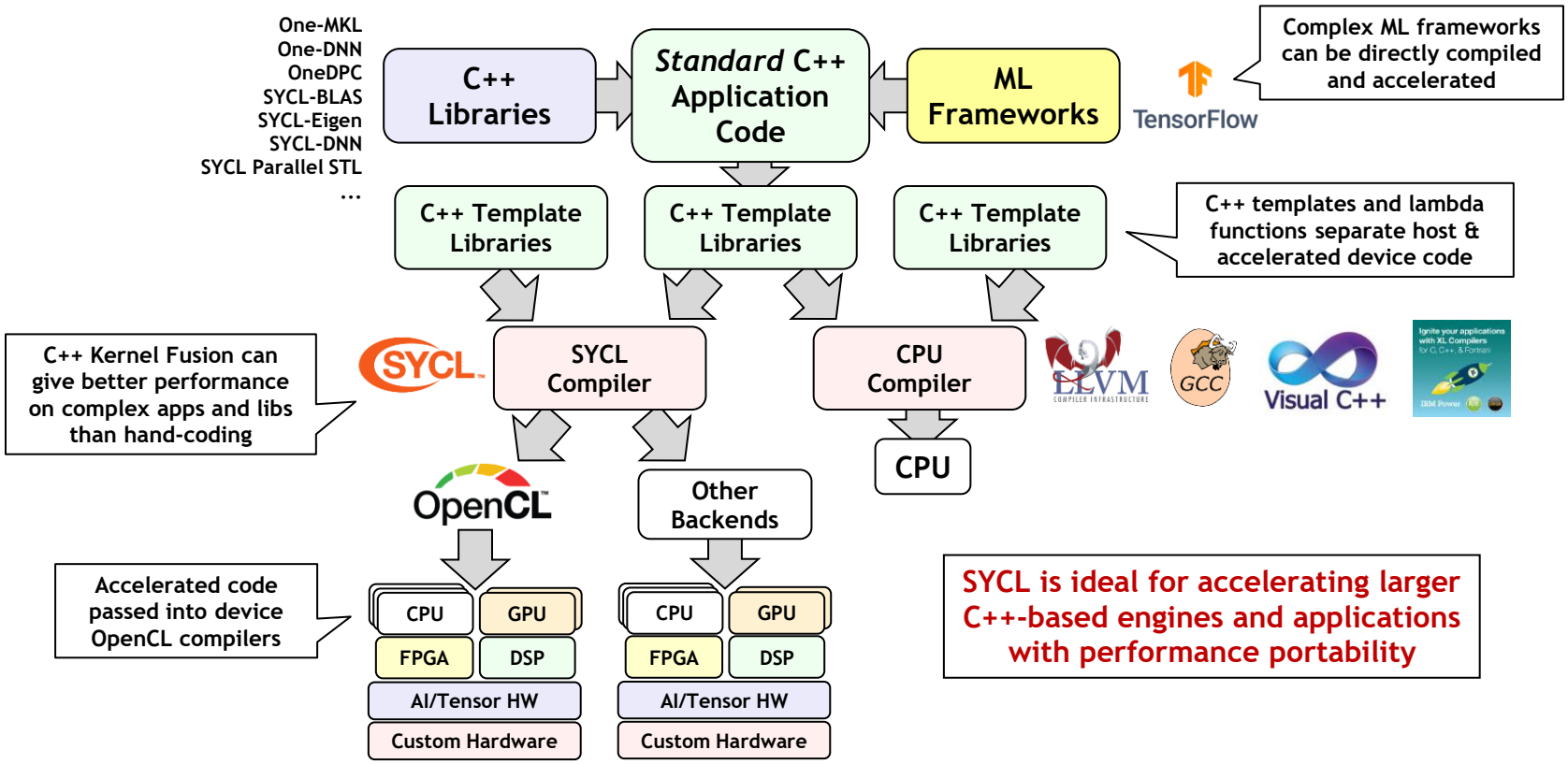
The **right processor** should be assigned to the **right task.**

Related Work

- **Vector Engine Offloading (VEO) and VH Call**
  - ○ VH-VE collaboration
    - Data transfers
    - Function calls
    - Synchronization
    - VH-initiated (VEO) and VE-initiated (VH call)
  - ✕ NEC's proprietary programming tools =application code needs to be **specialized only to NEC SX-Aurora TSUBASA**

Open standard for offload programming = SYCL

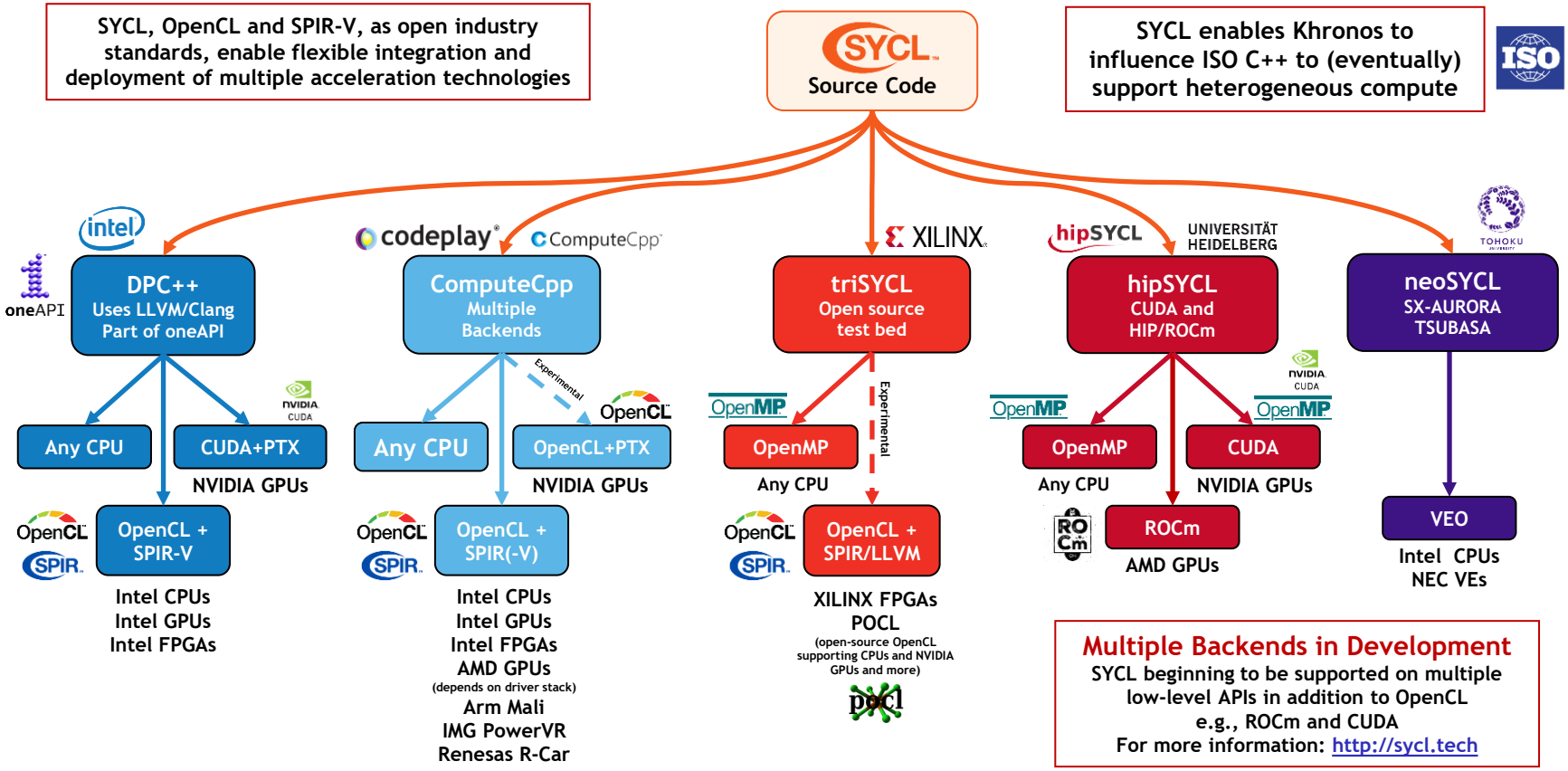
SYCL Single Source C++ Parallel Programming



This work is licensed under a Creative Commons Attribution 4.0 International License

SYCL Implementations in Development

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies



This work is licensed under a Creative Commons Attribution 4.0 International License

Kernel Outlining

Simple Triad implementation

```
for (int i = 0; i < array_size; i++) {
  b[i] = scalar * c[i];
}
```

SYCL Triad implementation

```
queue.submit([&](handler &cgh) {
  auto ka = buf_a.template get_access<access::mode::write>(cgh);
  auto kb = buf_b.template get_access<access::mode::read>(cgh);
  auto kc = buf_c.template get_access<access::mode::read>(cgh);
  cgh.parallel_for<class triad_kernel>(range<1>(array_size), [=](const id<1> &i) {
    ka[i] = kb[i] + scalar * kc[i];
  });
});
```

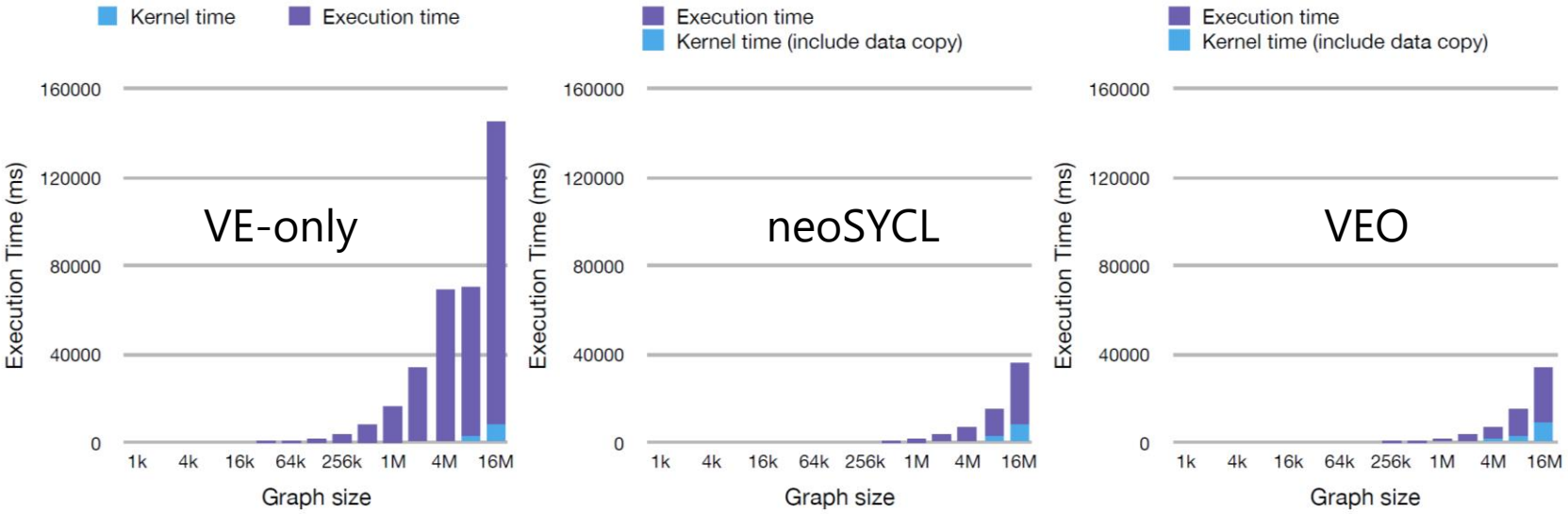
Kernel Outlining by neoSYCL

```
int triad_kernel(double *ka, double *kb, double *kc, int N) {
  for (int i = 0; i < N; i++) {
    ka[i] = kb[i] + 0.4 * kc[i];
  }
  return 0;
}
```

A kernel is translated into a loop in a separate function
→ VEO can invoke it on VE.

Performance Evaluation

BFS (Rodinia)



Programming with SYCL leads to lower code complexity

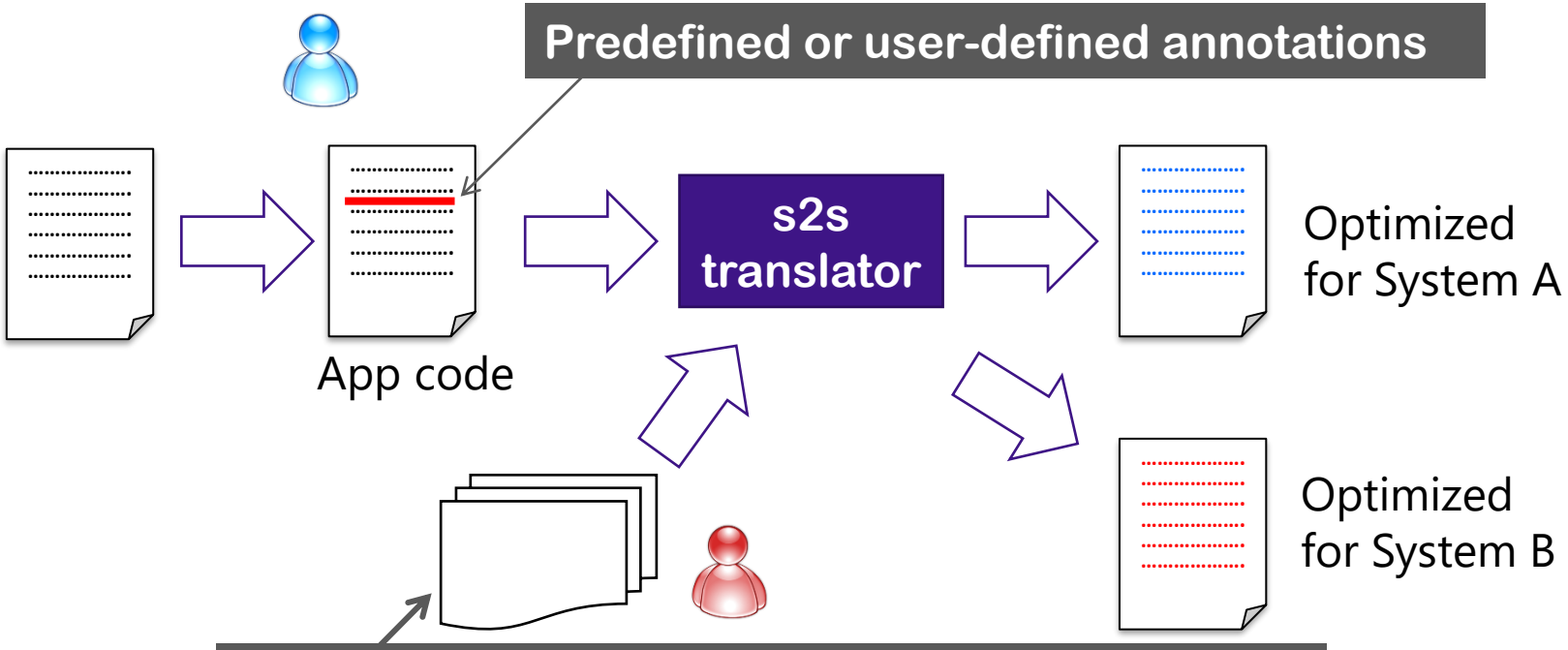
Applicaton	VERSION	NLOC	AvgCCN	Avg.token
STREAM	SYCL	148	1.2	96
	VE-Offload	296	3.8	159.7
N-body	SYCL	86	2.7	233.3
	VE-Offload	166	5	240.2
	Origin	66	3.3	173.7
BFS	SYCL	133	4.5	248
	VE-Offload	225	7.4	302
	Origin	116	4.5	196.2

Xevolver-C

A DSL for describing performance engineering

Xevolver Framework

Various transformations are required for replacing **arbitrary code modifications**.
= cannot be expressed by combining predefined transformations.
→ Xevolver : a framework for custom code transformations



Ongoing

Xevolver-C

- **DSL for user-defined code transformation**
  - C langauge syntax with some special features

```
int i,j;
xev_stmt* any_stmt;

int main(int argc, char** argv){
#pragma xev stmt src("label1")
  {
    for(i=0;i<10;i++){
      for(j=0;j<10;j++){
        any_stmt;
      }
    }
  }
#pragma xev stmt dst("label1");
  {
    for(int ij=0;ij<100;ij++){
      i=ij%10;
      j=ij/10;
      any_stmt;
    }
  }
}
```

Original loop structure

All the statements in the loop body are copied

Transformed loop structure

Code patterns are written in the rule → **easily customizable** for individual cases

Ongoing

Performance Evaluation

■ HPC refactoring catalog (Egawa et al. 2017)

- A collection of 33 code optimization cases
  - Effective for SX-9 and (probably) SX-ACE
  - Sample codes are written in Fortran

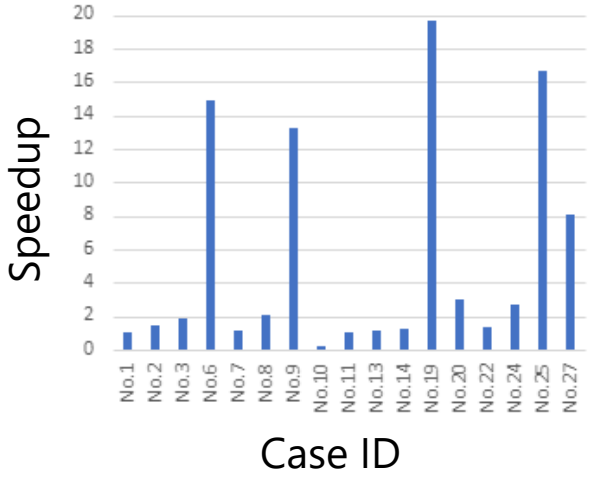
How about SX-AT ?

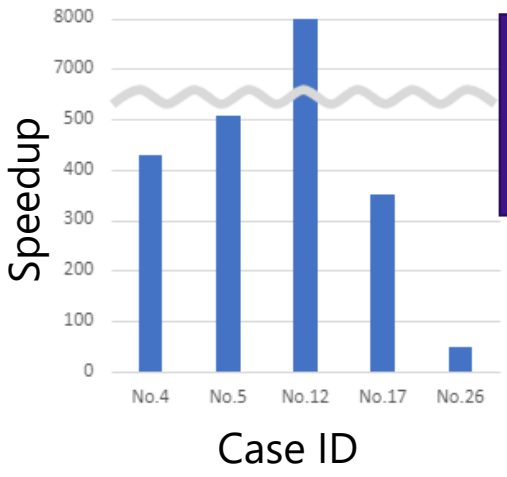
How about C ?

→23 optimizations have been translated into C

→22 optimizations can be expressed by Xev-C

→20 optimizations can improve SX-AT performance





Performance did not change in 2 cases because the newer compiler for SX-AT works well even for the original version.
= Optimization not needed for the 2 cases.

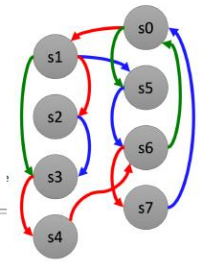
… and many more

Other ongoing projects

"Quantum Compiler"

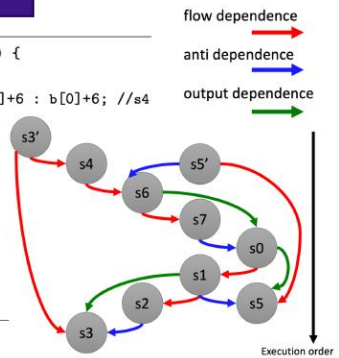
Non-vectorizable loop

```
1   for (i = 1; i < 16*1024; i++) {
2       a[i] = d[i]+2; //s0
3       b[i] = a[i]+3; //s1
4       e[i] = b[i]+4; //s2
5       b[i] = f[i]+5; //s3
6       c[i] = b[i-1]+6; //s4
7       a[i] = a[i+1]+7; //s5
8       a[i+1] = c[i]+8; //s6
9       h[i] = a[i+1]+9; //s7
10      }
```



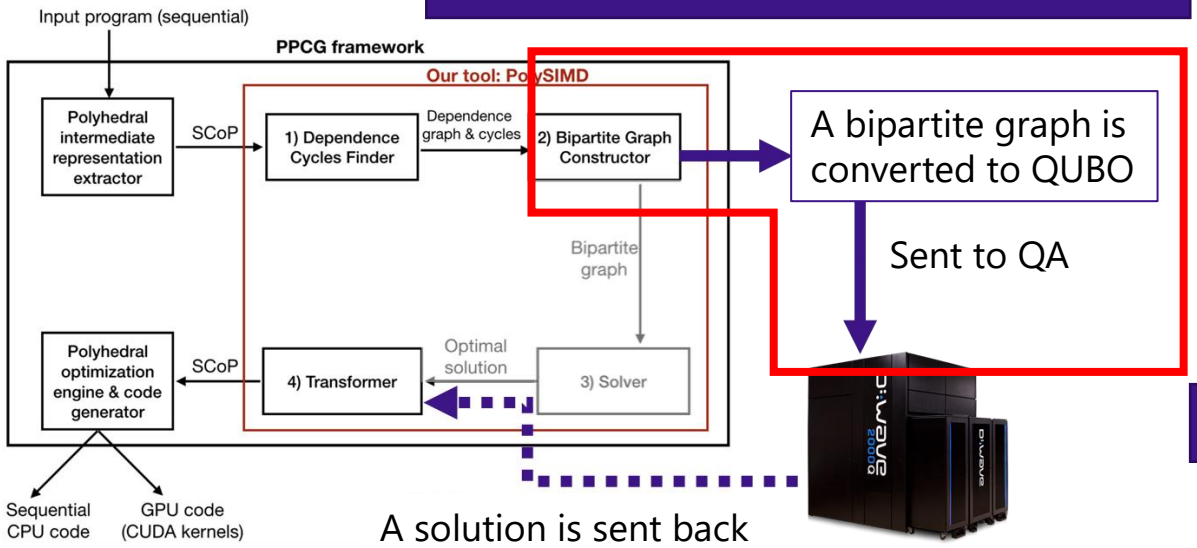
Vectorizable loop

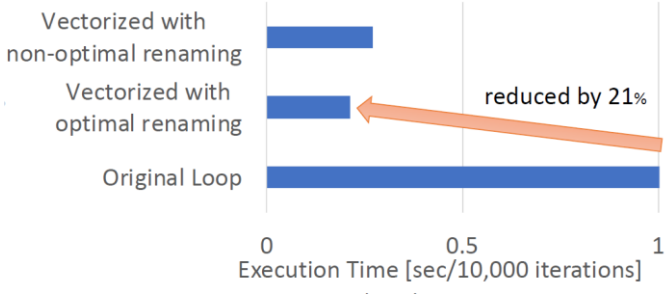
```
1   for (i = 1; i < 16*1024; i++) {
2       temp[i] = f[i] + 5;//s31
3       c[i] = (i >= 1)? temp[i-1]+6 : b[0]+6; //s4
4       k = a[i+1]; //s51
5       a[i + 1] = c[i] + 8;//s6
6       h[i] = a[i+1] + 9; //s7
7       a[i] = d[i] + 2; //s0
8       b[i] = a[i] + 3; //s1
9       e[i] = b[i] + 4; //s2
10      b[i] = temp[i]; //s3
11      a[i] = k + 7; //s5
12  }
```



Loop vectorization needs to solve a combinatorial problem (NP-hard)

A part of the compilation process is offloaded to D-Wave machine





Loop becomes vectorizable!

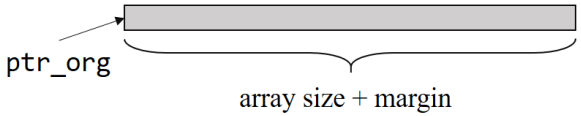
Abstraction of conflict avoiding

Automatic data padding to avoid conflicts

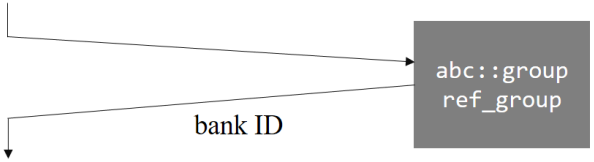
The constructor of `abc::array` is called.

Arguments
array size (integer) : `Ni, Nj, …`
reference of `abc::group` : `ref_group`

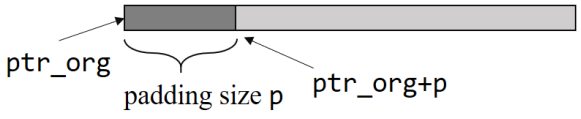
`malloc` for array size + margin.



The `abc::group` assigns a bank ID.



Calculate the padding size `p` and pad `ptr_org`.



Then, `ptr_org+p` is the initial address of the array region.

We can avoid spike-like performance drops

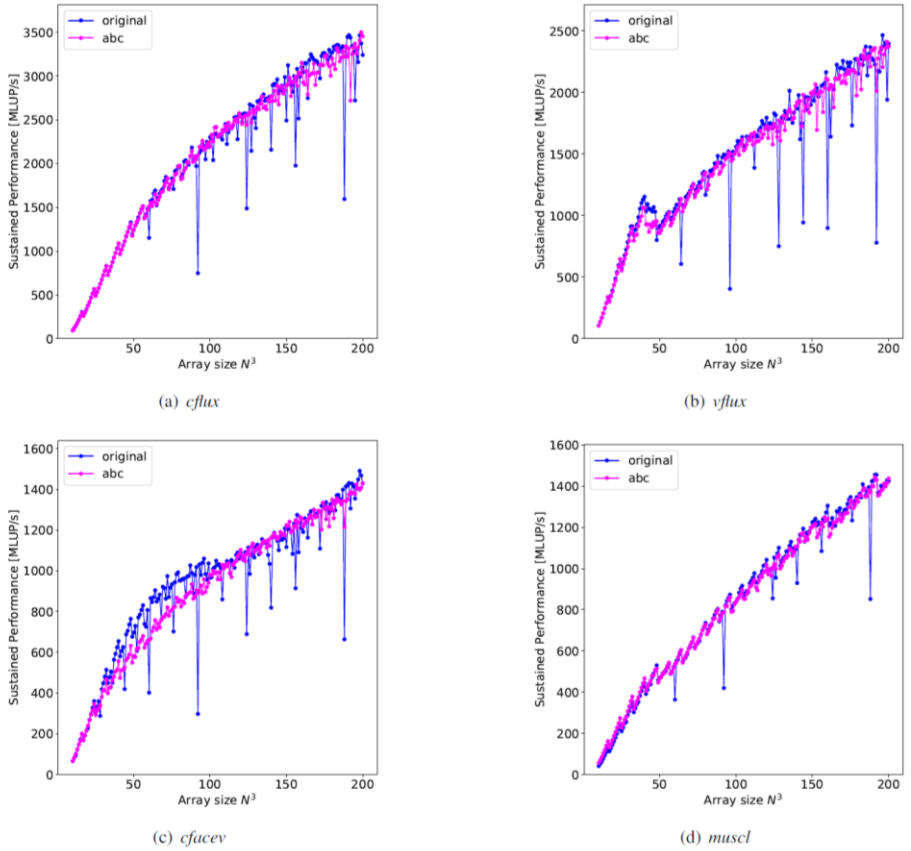
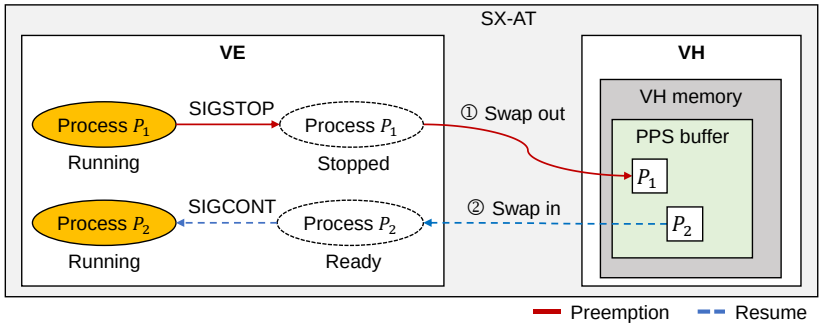


Figure 8. The evaluation results of UPACS-Parts.

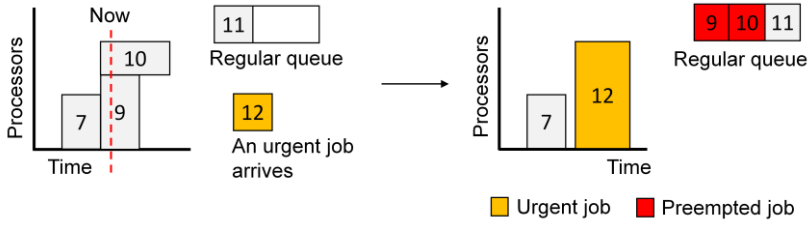
Urgent Computing

Agung et al. IEEE Access, 2021

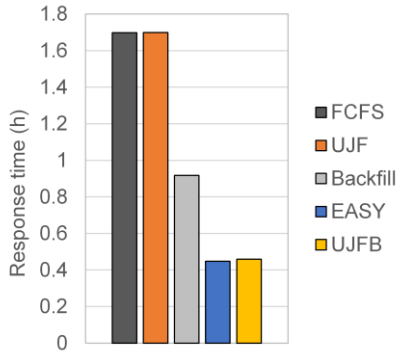
- **Use of an HPC system at Emergency**
  - Preemptive execution of urgent jobs on a heterogeneous system (such as AOBA)



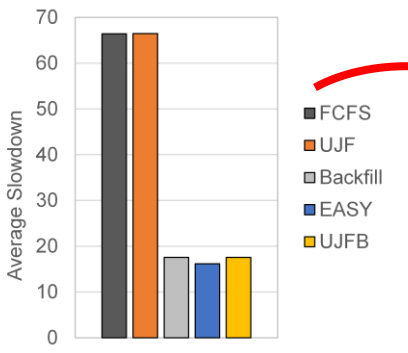
A process running on VE is partially moved to VH (Partial Process Swapping)



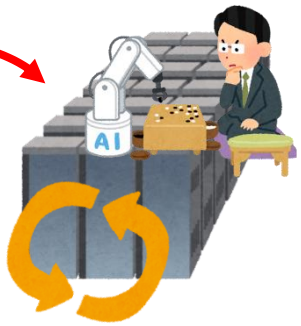
  - Scheduling urgent jobs while keeping the efficiency of regular jobs

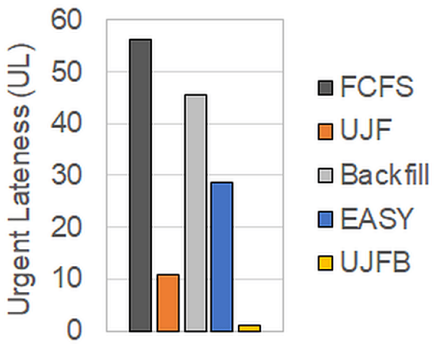


Regular job execution time



Regular job slowdown





Urgent job latencies

Summary

- **Lately we have started many research projects for NEC SX-Aurora TSUBASA**
  - Performance evaluation (AOBA)
  - Programming model (neoSYCL)
  - Performance tuning tool (Xevolver-C)
  - Urgent computing
  - Job scheduling
  - Machine learning
  - Simulated annealing …

Stay tuned, there is more to come!

Acknowledgments

- **Many thanks to the collaborators.**
  - Ryusuke Egawa (Tokyo Denki University)
  - Souya Fujimoto (NEC)
  - Yoko Isobe (NEC)
  - Yoichi Shimomura (NEC)
  - Tsuyoshi Yamashita (CC, Tohoku University)
  - Daisuke Sasaki (CC, Tohoku University)
  - Joint-Research Division of High Performance Computing (NEC), CC, Tohoku University

    Led by Professor Hiroaki Kobayashi

This work is partially supported by MEXT Next-Generation High-Performance Computing Infrastructures and Applications R&D Program "R&D of a Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications," and Grant-in-Aid for Scientific Research(A) #20H00593.