neoSYCL
SX-Aurora TSUBASAのための軽量SYCL実装

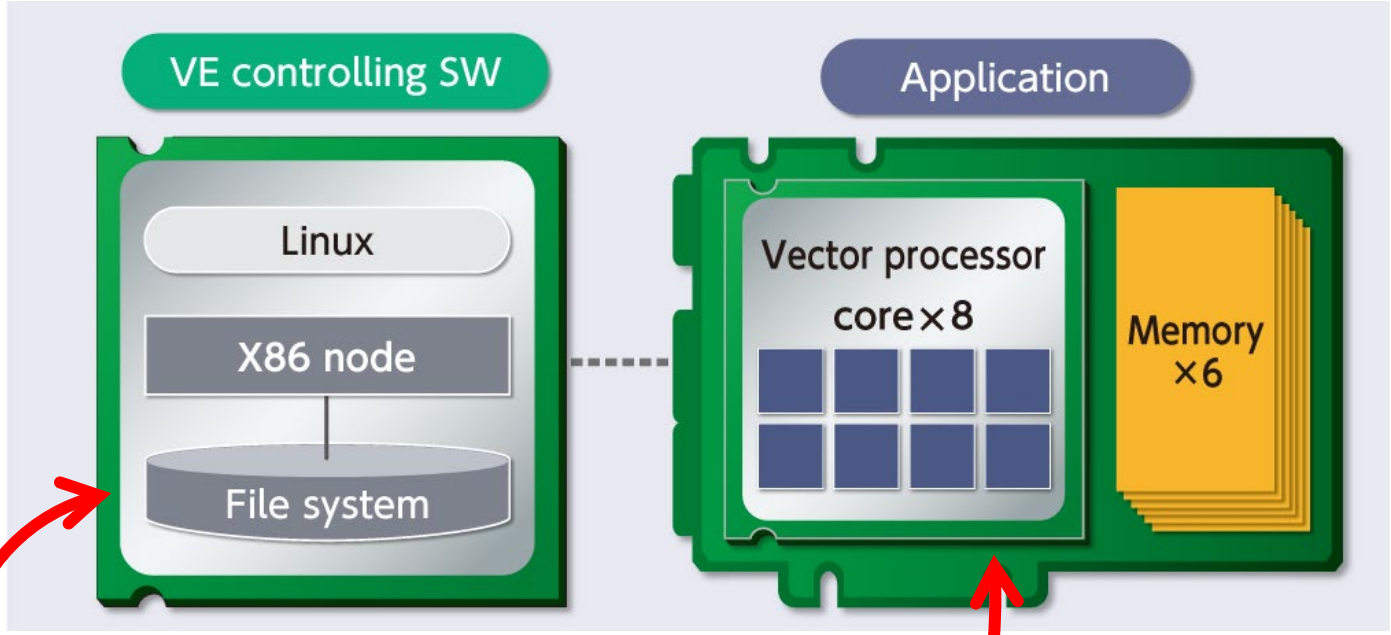
2022年4月15日
PCCC HPC-OSS部会ワークショップ
「高性能クラスタ・プログラミング最前線」

東北大学サイバーサイエンスセンター
滝沢 寛之
<takizawa@tohoku.ac.jp>

もくじ

- 背景と目的
- SX-Aurora TSUBASA向けSYCL実装
- 性能評価
- まとめと今後の展望

背景 ～ ヘテロ構成のベクトルシステム ～



Vector Host (VH)

Vector Engine (VE)

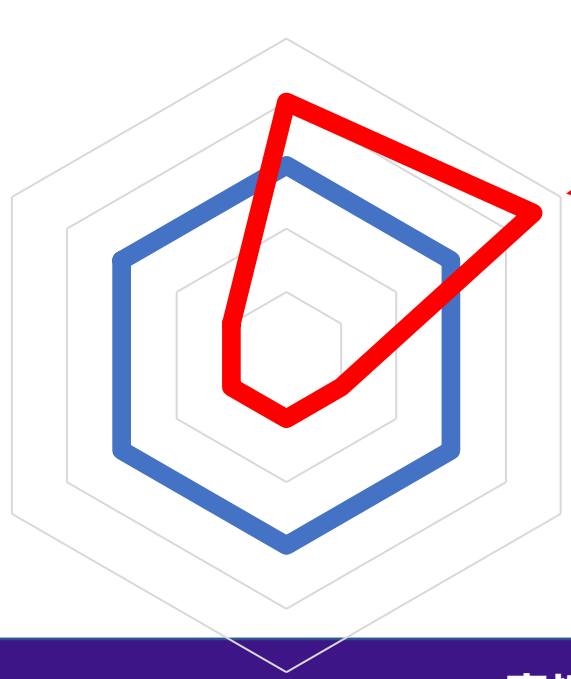
```
$ gcc matmul.c -o matmul
$ ./matmul

$ ncc matmul.c -o matmul
$ ./matmul
```

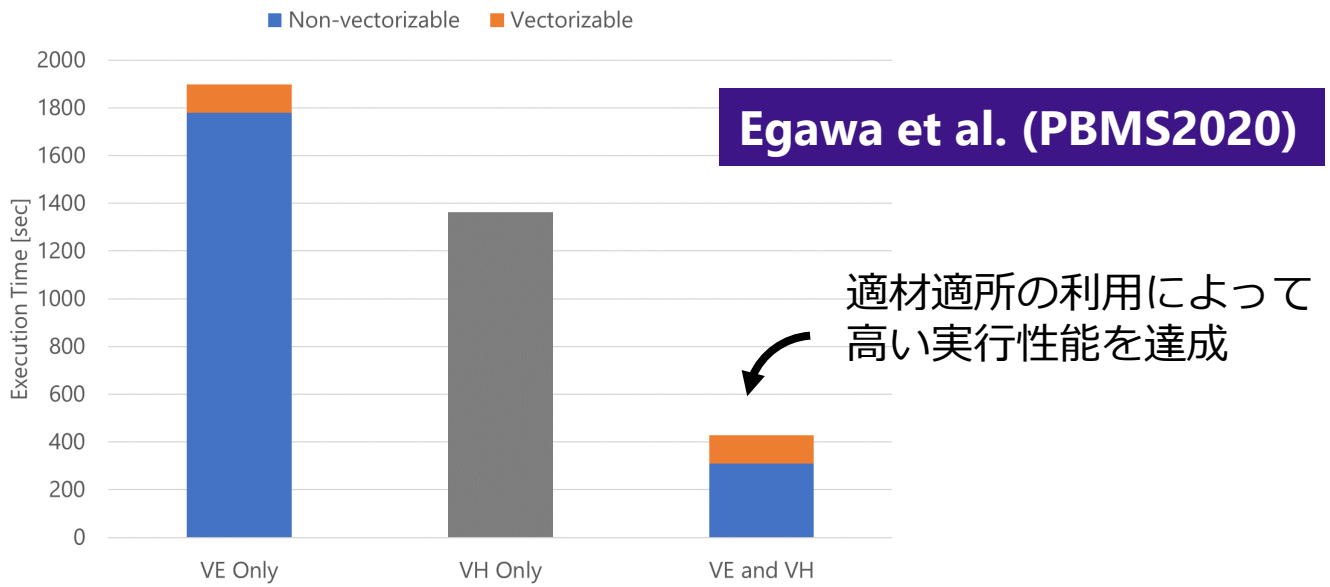
東北大学のスパコンAOBA

NEC SX-Aurora TSUBASAを
中核とするHPCシステム

動機 ～ 適材適所の必要性 ～

- **Vector Host** と **Vector Engine** は両方とも
  - 汎用プロセッサ (単なるアクセラレータではない)
    - 標準的なプログラミング言語を単独で実行可能 (C/C++/Fortran etc)
  - 異なる性能特性
    - **ベクトル向きの処理であれば VE**は高性能を発揮可能





Egawa et al. (PBMS2020)

適材適所の利用によって
高い実行性能を達成

Vector Engine Offloading (VEO)

- カーネル実行をVEにオフロードするための**低レベルな専用API**

Main program for **VH**

```
#include <ve_offload.h>
int main()
{
  /* Load "vehello" on VE node 0 */
  struct veo_proc_handle *proc = veo_proc_create(0);
  uint64_t handle = veo_load_library(proc, "./libvehello.so");
  struct veo_thr_ctxt *ctx = veo_context_open(proc);
  struct veo_args *argp = veo_args_alloc();
  uint64_t id = veo_call_async_by_name(ctx, handle, "hello", argp);
  uint64_t retval;
  veo_call_wait_result(ctx, id, &retval);
  veo_args_free(argp);
  veo_context_close(ctx);
  veo_proc_destroy(proc);
  return 0;
}
```

kernel program for **VE**

```
#include <stdio.h>
#include <stdint.h>
uint64_t hello()
{
  printf("Hello, world¥n");
  return 0;
}
```

このコードをコンパイルして
共有オブジェクト
./libvehello.so
として準備

SX-Aurora TSUBASA専用のアプリになってしまうことが問題

SYCL Implementations in Development (2021/04/27)

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

SYCL Source Code

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

DPC++
Uses LLVM/Clang
Part of oneAPI
- Any CPU
- NVIDIA GPUs (CUDA)
- OpenCL, SPIR: Intel CPUs, Intel GPUs, Intel FPGAs
- Level Zero: Intel GPUs

ComputeCpp
Multiple Backends
- Any CPU
- OpenCL, SPIR: Intel CPUs, Intel GPUs, Intel FPGAs, AMD GPUs (depends on driver stack), Arm Mali, IMG PowerVR, Renesas R-Car
- Experimental: OpenCL, NVIDIA GPUs

triSYCL
Open source test bed
- OpenMP: Any CPU
- TBB: Any CPU
- Experimental: OpenCL, SPIR: XILINX FPGAs, POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more)

hipSYCL
CUDA and HIP/ROCm
- OpenMP: Any CPU
- NVIDIA GPUs (CUDA)
- RO Cm: AMD GPUs
- Level Zero: Intel GPUs

neoSYCL
SX-AURORA TSUBASA
- VEO: Intel CPUs, NEC VEs

Multiple Backends in Development
There is development on supporting SYCL on even more low-level frameworks.
For more information: http://sycl.tech

neoSYCL

- VEOをバックエンドとして利用するSYCLインタフェースの実装
  - Kernel Outlining

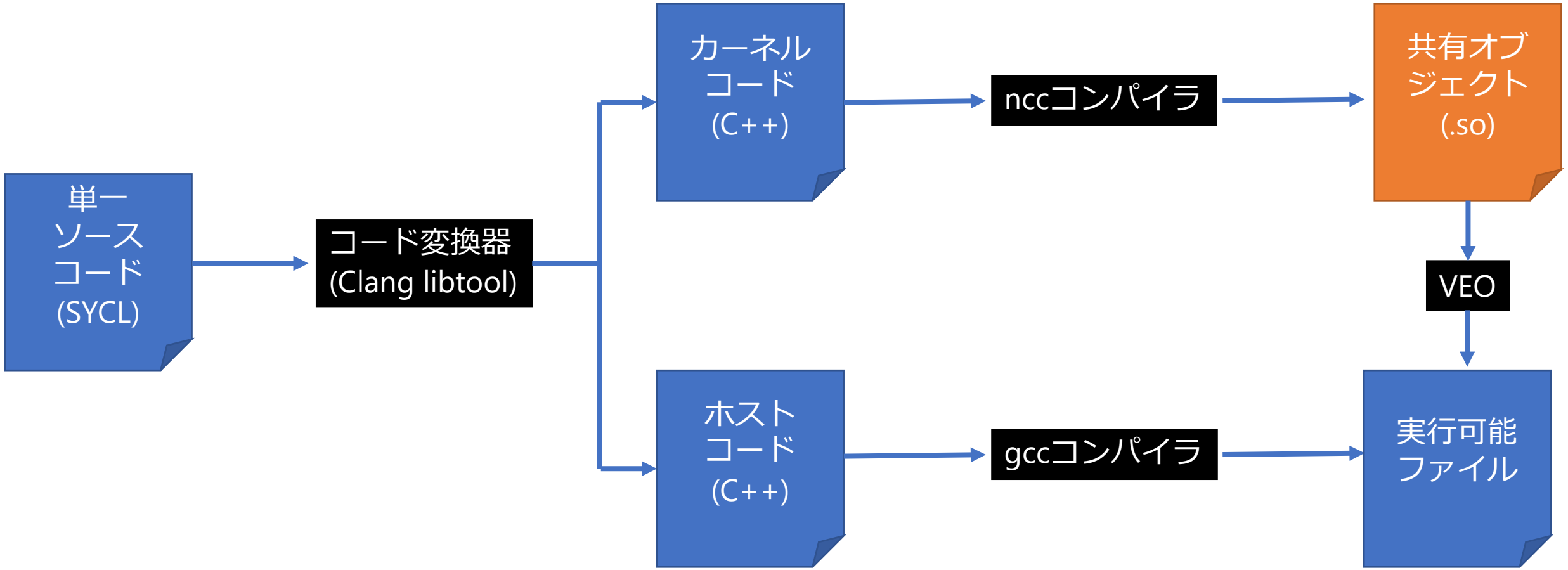
```
queue.submit([&](handler &cgh) {
  auto ka = buf_a.template get_access<access::mode::write>(cgh);
  auto kb = buf_b.template get_access<access::mode::read>(cgh);
  auto kc = buf_c.template get_access<access::mode::read>(cgh);
  cgh.parallel_for<class triad_kernel>(range<1>(array_size), [=](const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

カーネル部分 (functor) を分離して関数内のループに変換
→ **VEOを使うことで、カーネル部分だけをVEにオフロード実行可能**

```
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;
}
```

コード変換器以外はヘッダファイルとして実装
= SYCLインタフェースを提供するための簡易実装

コンパイルの流れ


単一
ソース
コード
(SYCL)
コード変換器
(Clang libtool)
カーネル
コード
(C++)
ホスト
コード
(C++)
nccコンパイラ
gccコンパイラ
共有オブ
ジェクト
(.so)
VEO
実行可能
ファイル

他のSYCL実装との比較

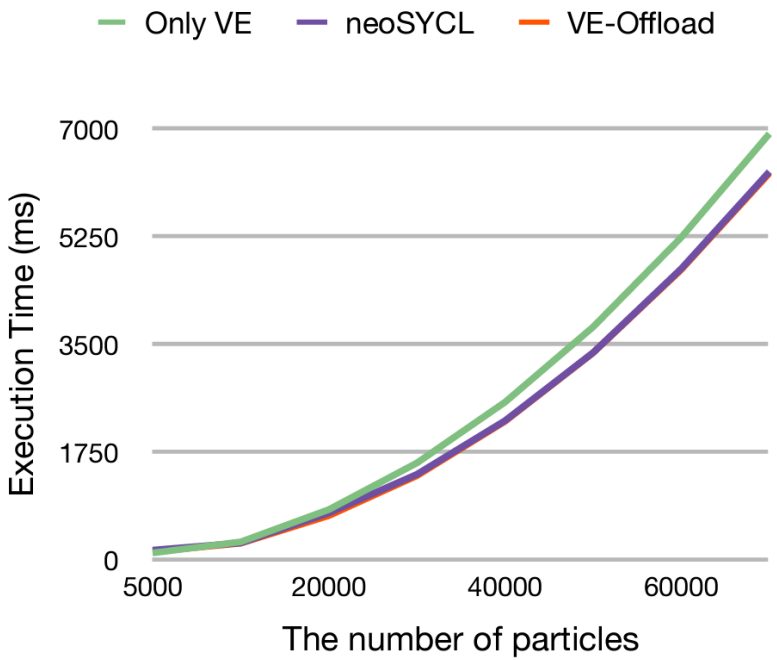
- GPU-STREAM (CPU実行)
  - 1スレッド実行での性能比較
    - OpenMP実装とneoSYCL実装の性能差は平均1.6%程度
    - triSYCLが使っているBoostの多次元配列がメモリ確保とデータコピーを多く実行

N-bodyによる評価

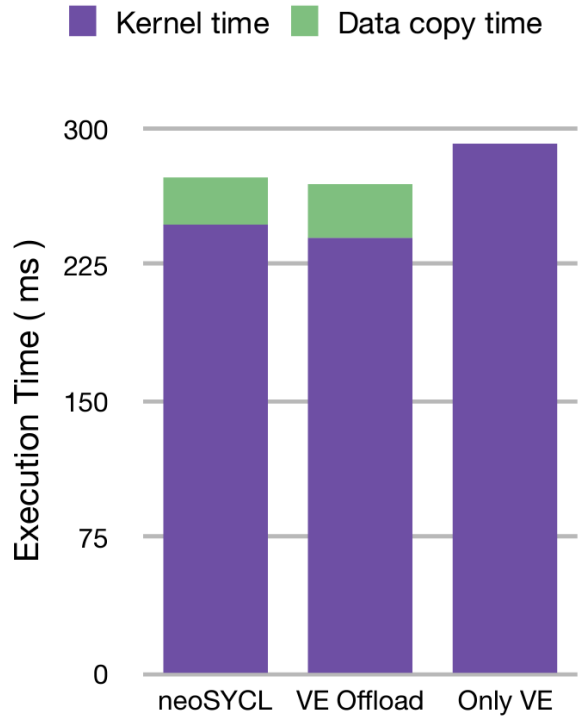
- 2つのループ (速度と位置)をそれぞれカーネルとしてオフロード
  - カーネル以外の部分はVHで実行 (VHの方が高性能)

```
for (int i = 0; i < nBodies; i++) {
  float Fx = 0.0f;
  float Fy = 0.0f;                                    loop-1
  float Fz = 0.0f;
  for (int j = 0; j < nBodies; j++) {
    ...
  }
  p[i].vx += delta * Fx;
  p[i].vy += delta * Fy;
  p[i].vz += delta * Fz;
}
for (int i = 0; i < nBodies; i++) {
  p[i].x += p[i].vx * delta;
  p[i].y += p[i].vy * delta;                          loop-2
  p[i].z += p[i].vz * delta;
}
```

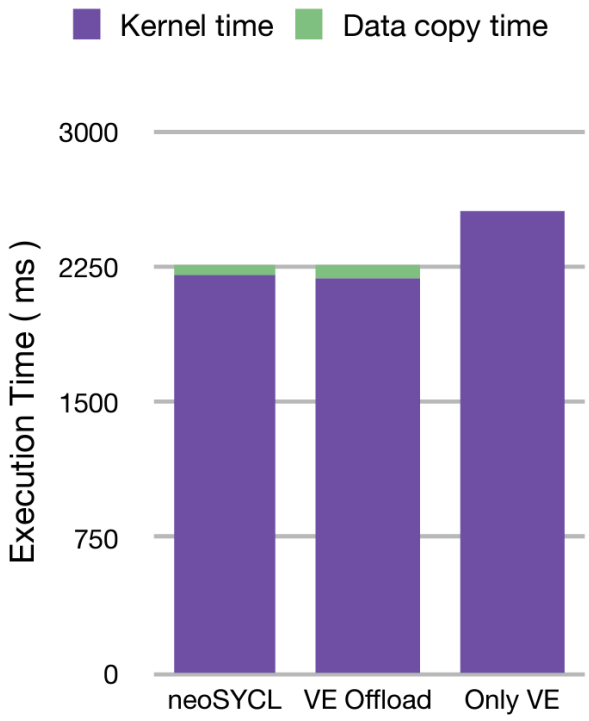
N-bodyによる評価結果



データサイズと実行時間



データサイズ 10,000

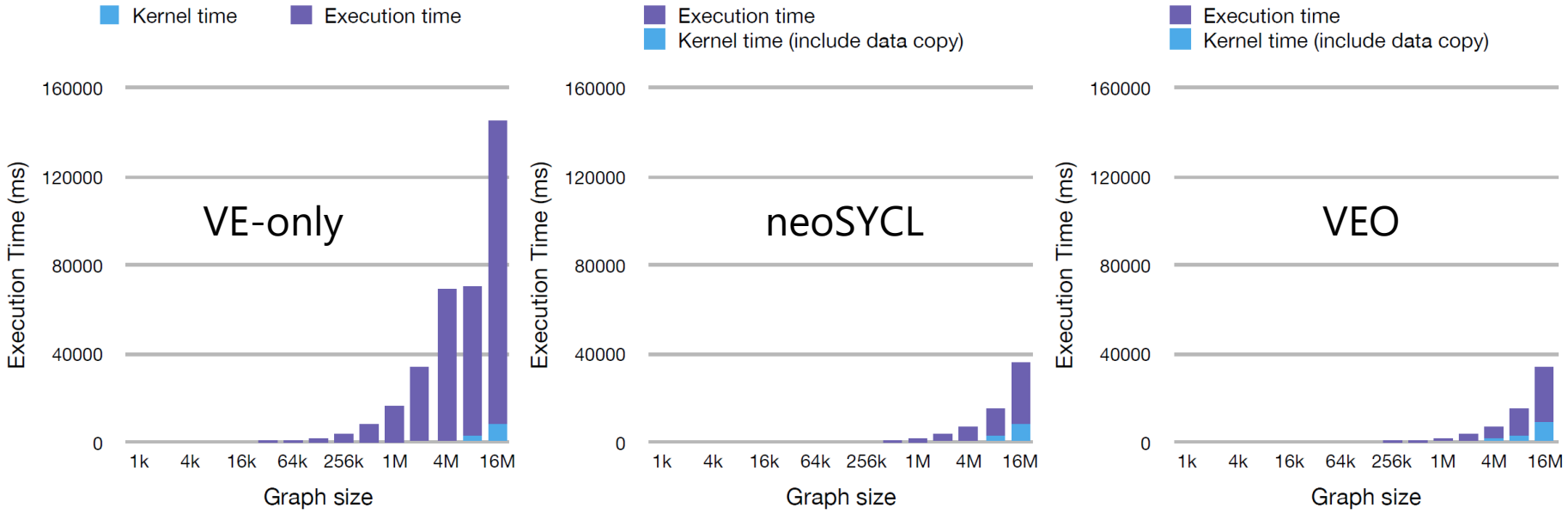


データサイズ 40,000

- データ転送時間を考慮してもオフロードしたほうが高性能
  - いずれの実装でもカーネル実行時間が支配的
  - VEO(低レベルAPI)とneoSYCLはほぼ同等の性能 (性能差は1.7%程度)

BFS(Rodinia)による評価結果

- 初期化部分がベクトルに不向きな実装のため**オフロードが効果的**



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

SYCLインタフェースによりコード複雑さを低減可能

Li et al. (PDCAT2021)

独自ベンチマーク集の作成

- **VEO-SYCL-Bench** https://github.com/Tohoku-University-Takizawa-Lab/veo-sycl-bench
  - SYCL-Bench (Lal et al.@Euro-Par 2020)が動かない(？)
    - DPC++でもneoSYCLでもコンパイルエラー
  - 抽象化によるオーバヘッドの評価にはVEO版も必要

SYCL-Bench's vec_add

```
void setup() {
  input1.resize(args.problem_size);
  input2.resize(args.problem_size);
  output.resize(args.problem_size);
  for (size_t i =0; i < args.problem_size; i++) {
    input1[i] = static_cast<T>(i);
    input2[i] = static_cast<T>(i);
    output[i] = static_cast<T>(0);
  }
  input1_buf.initialize(args.device_queue, input1.data(), s::range<1>(args.problem_
  input2_buf.initialize(args.device_queue, input2.data(), s::range<1>(args.problem_
  output_buf.initialize(args.device_queue, output.data(), s::range<1>(args.problem_
}
void run(std::vector<cl::sycl::event>& events) {
  events.push_back(args.device_queue.submit(
      [&](cl::sycl::handler& cgh) {
    auto in1 = input1_buf.template get_access<s::access::mode::read>(cgh);
    auto in2 = input2_buf.template get_access<s::access::mode::read>(cgh);
    auto out = output_buf.template get_access<s::access::mode::read_write>(cgh);
    cl::sycl::range<1> ndrange {args.problem_size};
    cgh.parallel_for<class VecAddKernel<T>>(ndrange,
      [=](cl::sycl::id<1> gid) {
        out[gid] = in1[gid] + in2[gid];
      });
  }));
}
```

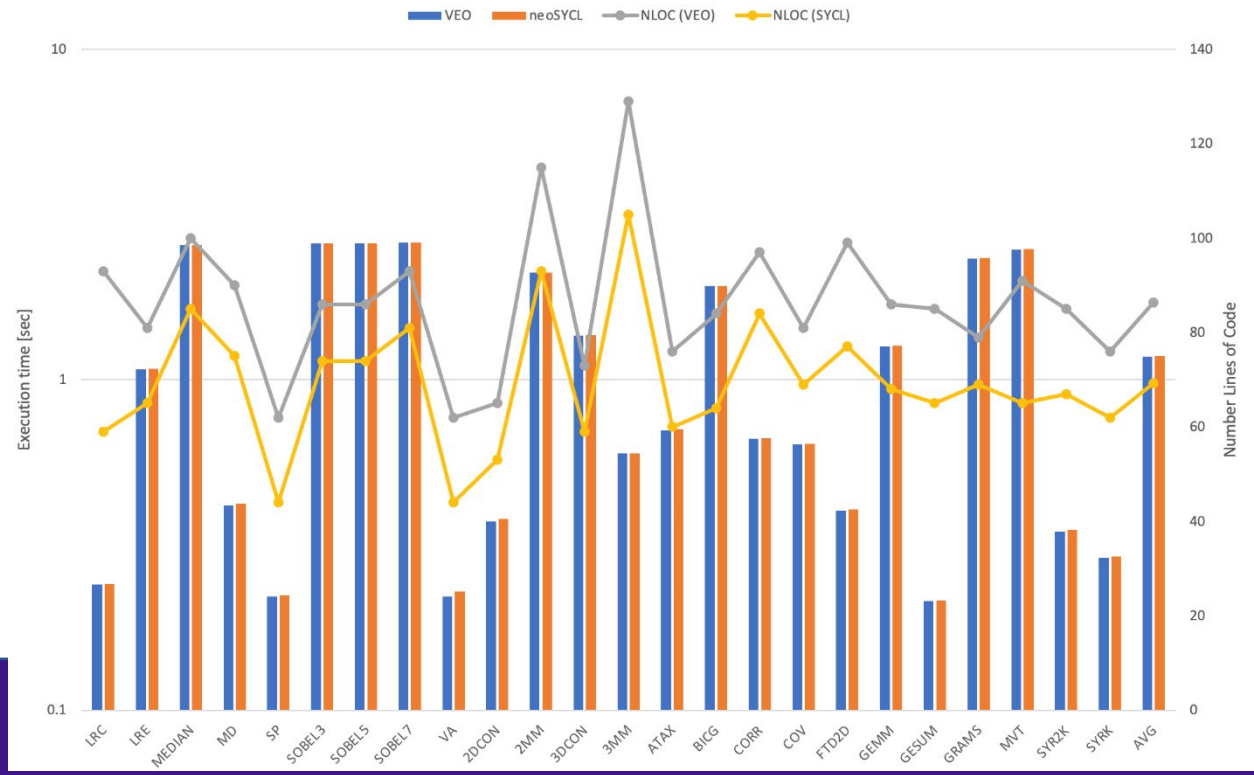
VEO-SYCL-Bench's vec_add

```
int main(int argc, char** argv) {
    accelerator_selector as;
    queue q(as);
    vector<float> input1(LENGTH, 0);
    vector<float> input2(LENGTH, 0);
    vector<float> output(LENGTH, 0);
    for (size_t i = 0; i < LENGTH; i++) {
        input1[i] = static_cast<float>(i);
        input2[i] = static_cast<float>(i);
        output[i] = static_cast<float>(0);
    }
    buffer<float> input1_buf(input1.data(), range<1>(LENGTH));
    buffer<float> input2_buf(input2.data(), range<1>(LENGTH));
    buffer<float> output_buf(output.data(), range<1>(LENGTH));
    q.submit([&](handler& cgh) {
        auto in1 = input1_buf.template get_access<access::mode::read>(cgh);
        auto in2 = input2_buf.template get_access<access::mode::read>(cgh);
        auto out = output_buf.template get_access<access::mode::read_write>(cgh);
        cgh.parallel_for<class VecAddKernel>(range<1>(LENGTH), [=](id<1> gid) {
            out[gid] = in1[gid] + in2[gid];
        });
    });
    q.wait();
    return 0;
}
```

各ベンチマークのカーネル部分だけコピーしてそれ以外はシンプルなコードに書き直し

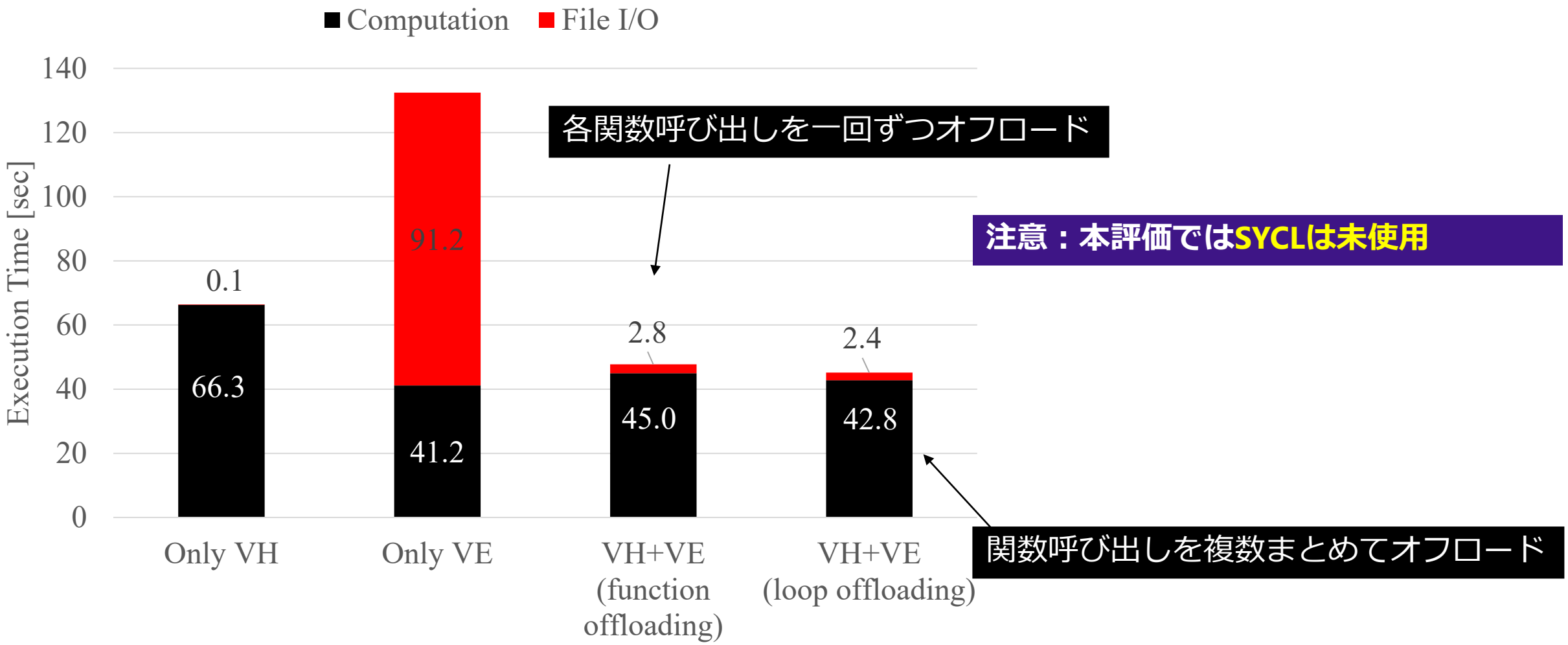
生産性と性能の評価

- neoSYCLとVEOとの性能差は平均**わずか 0.42%**
  - 高抽象化に伴う実行時オーバヘッド (abstraction penalty) は無視できるレベル
- SYCLコードの複雑さはVEOコードよりも**常に低い**
  - SYCLプログラミングにより他プラットフォームとの移植性や生産性を改善



	SYCL	VEO
Avg.NLOC	**69.2**	86.4
Avg.CCN	**10**	12
Avg.token	**653.0**	764.5

VHとVEを適材適所で利用する効果



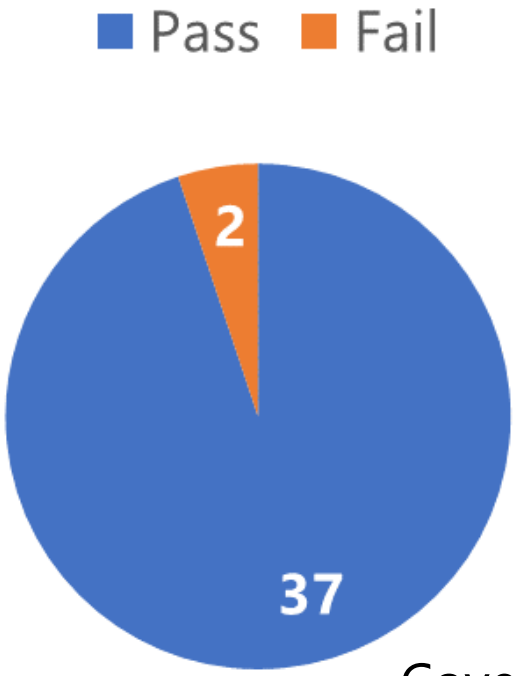
最近の開発状況

- 規格適合性テストとコードリファクタリング
  - DPC++ test cases (basic tests)　`https://github.com/intel/llvm/tree/sycl/sycl/test`

sycl / llvm / sycl / test / basic_tests /　Go to file　Add file

DenisBakhvalov [SYCL] Fix is_device_copyable with range rounding (#4478)　2d28cd4　2 days ago　History

accessor	[SYCL] Transition from ONEAPI/INTEL to ext::oneapi/intel namespaces (#...	2 months ago
buffer	[SYCL] Warning if use sycl without dynamic C++ RT on Windows (#2501)	6 months ago
free_function_queries	[SYCL] Move free function queries to experimental namespace (#4090)	2 months ago
handler	[SYCL] Correct CFE behavior with named lambdas in unnamed-lambda mode (...	2 months ago
marray	[SYCL] Warning if use sycl without dynamic C++ RT on Windows (#2501)	6 months ago
queue	[SYCL] Move tests with dependencies to on-device directory (#2732)	10 months ago
stream	[SYCL] Move tests with dependencies to on-device directory (#2732)	10 months ago
vectors	[SYCL] Deprecate half from global namespace (#4363)	7 days ago
SYCL-2020-spec-const-ids-order.cpp	[SYCL] Switch to using integration footer by default (#3777)	2 months ago
SYCL-2020-spec-constants.cpp	[SYCL] Add missed constexpr for vec (#4324)	last month
address_space_traits.cpp	[SYCL] Sub-group load/store for raw pointers (#3255)	6 months ago
aliases.cpp	[SYCL] Deprecate half from global namespace (#4363)	7 days ago
built-ins.cpp	[SYCL][ROCm] Setup lit tests for ROCm plugin (#4163)	23 days ago
context.cpp	[SYCL] Warning if use sycl without dynamic C++ RT on Windows (#2501)	6 months ago
define_vendors.cpp	[SYCL] Predefine SYCL macros with vendor strings (#4498)	4 days ago
device.cpp	[SYCL] Warning if use sycl without dynamic C++ RT on Windows (#2501)	6 months ago



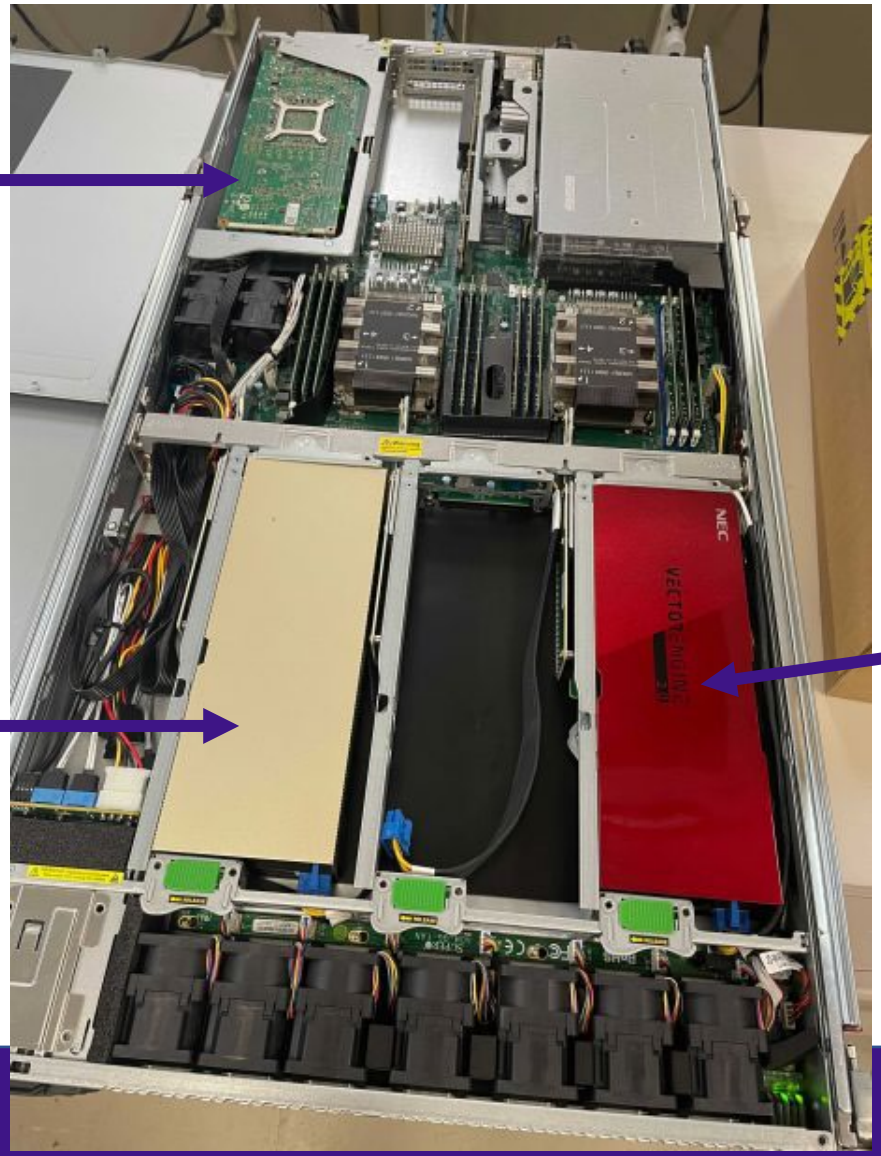
未サポートの機能
- `parallel_for_work_group`
- `parallel_for_work_item`
- `nd_item`
- `ndrange`

OpenCL由来の機能をサポートしても性能を期待できないため*.
* Takizawa et al. (PARCO2021)

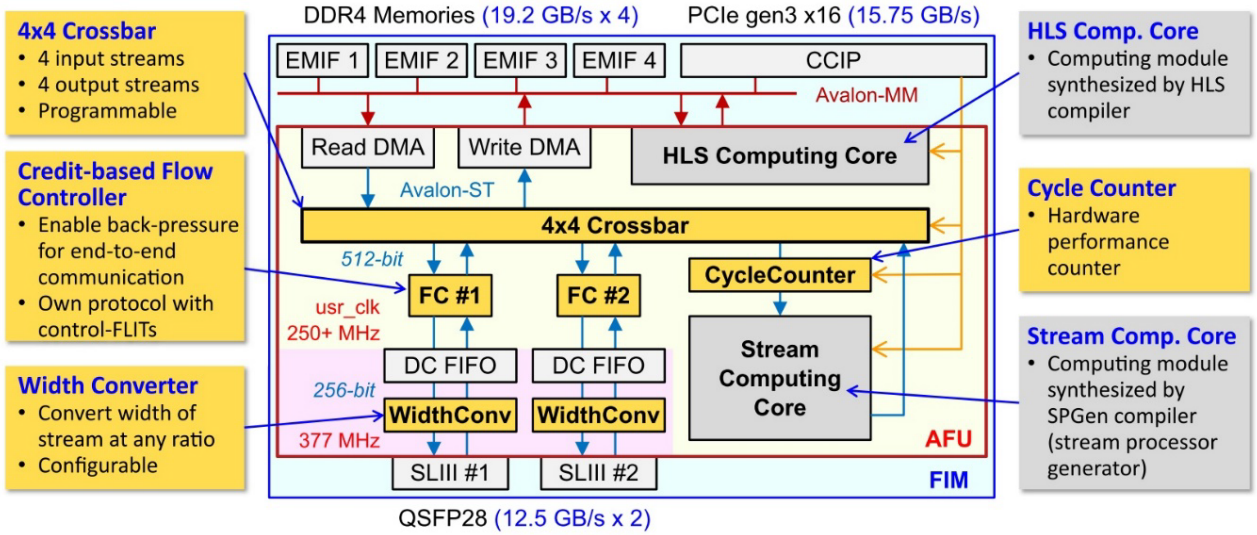
Coverage: **94.8%**

未サポートの機能 (**`parallel_for_work_group`**と**`parallel_for_work_item`**)に関する2件以外はテスト合格

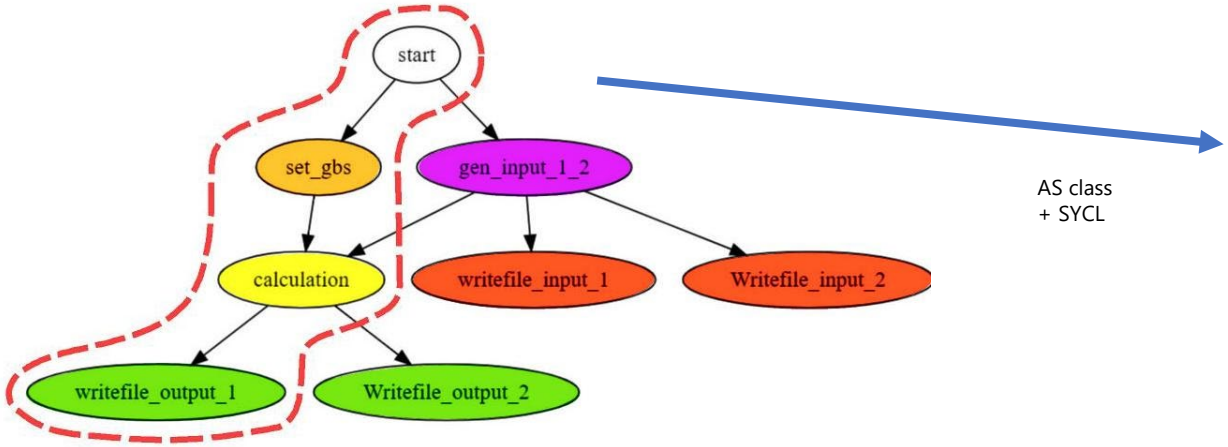
さらに多様なアクセラレータ



SYCL for FPGA

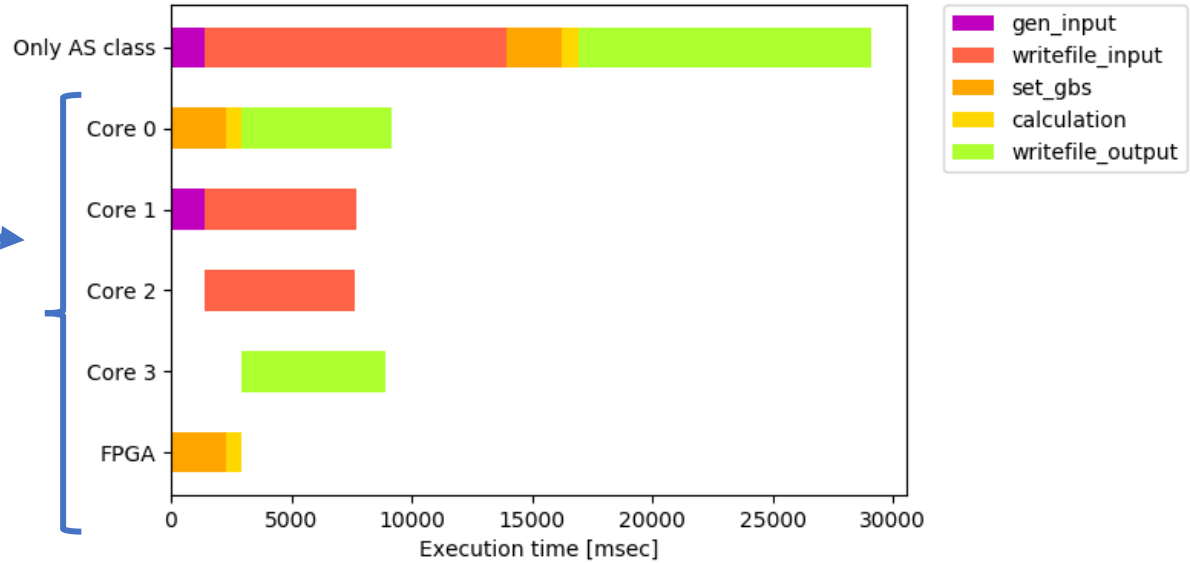


The ESSPER project
FPGAクラスタを利用するためのフレームワーク
led by 佐野健太郎 博士 (RIKEN R-CCS)



AS class
+ SYCL

SYCL for Collaboration between CPU and FPGA



まとめ

- Vector Engine (VE)は**ベクトル向きの処理で効果を発揮**
  - メモリ律速になりがちな科学技術計算での活躍を期待
  - x86とVEはそれぞれ異なる性能特性 → 適材適所の分担が重要
- neoSYCL：SX-ATでの**SYCLによるオフロードプログラミング**
  - カーネル部分を切り出してVEO経由でカーネル呼出しする簡易SYCL実装
    - 実行時のややこしい処理はVEOが担当 ☺
  - 性能と生産性の両面で有望
  - 現在も精力的に開発中！ 近日公開予定 (未定)

- **今後の展開**
  - 他のアクセラレータへのオフロードに利用 (ESSPER)
  - カーネル切り出し時で適切にコード変換することでさらなる性能向上の可能性