SYCL Programming Model

Abhishek Bagusetty
Performance Engineering Group
Argonne Leadership Computing Facility

abagusetty@anl.gov

SYCL – Specification

- SYCL is “not” a programming model but a “language specification”

– Heuristics looks similar to OpenCL-C bindings
– C++ single source (coexists host and device source code)
– Two distinct memory models (USM and/or Buffer)
– Asynchronous programming (overlaps device-compute, copy, host operations)
– Portability (functional and performance)
– Productivity

SYCL – Motivation

oneAPI Implementation of SYCL = C++ and SYCL* standard and extensions

Based on modern C++

✓ C++ productivity features and familiar constructs

Standards-based, cross-architecture

✓ Incorporates the SYCL standard for data parallelism and heterogeneous programming

SYCL* extensions

Productivity

- ➢Simple things should be simple to express
- ➢Reduce verbosity and programmer burden enhance performance

- •Give programmers control over program execution
- •Enable hardware-specific features

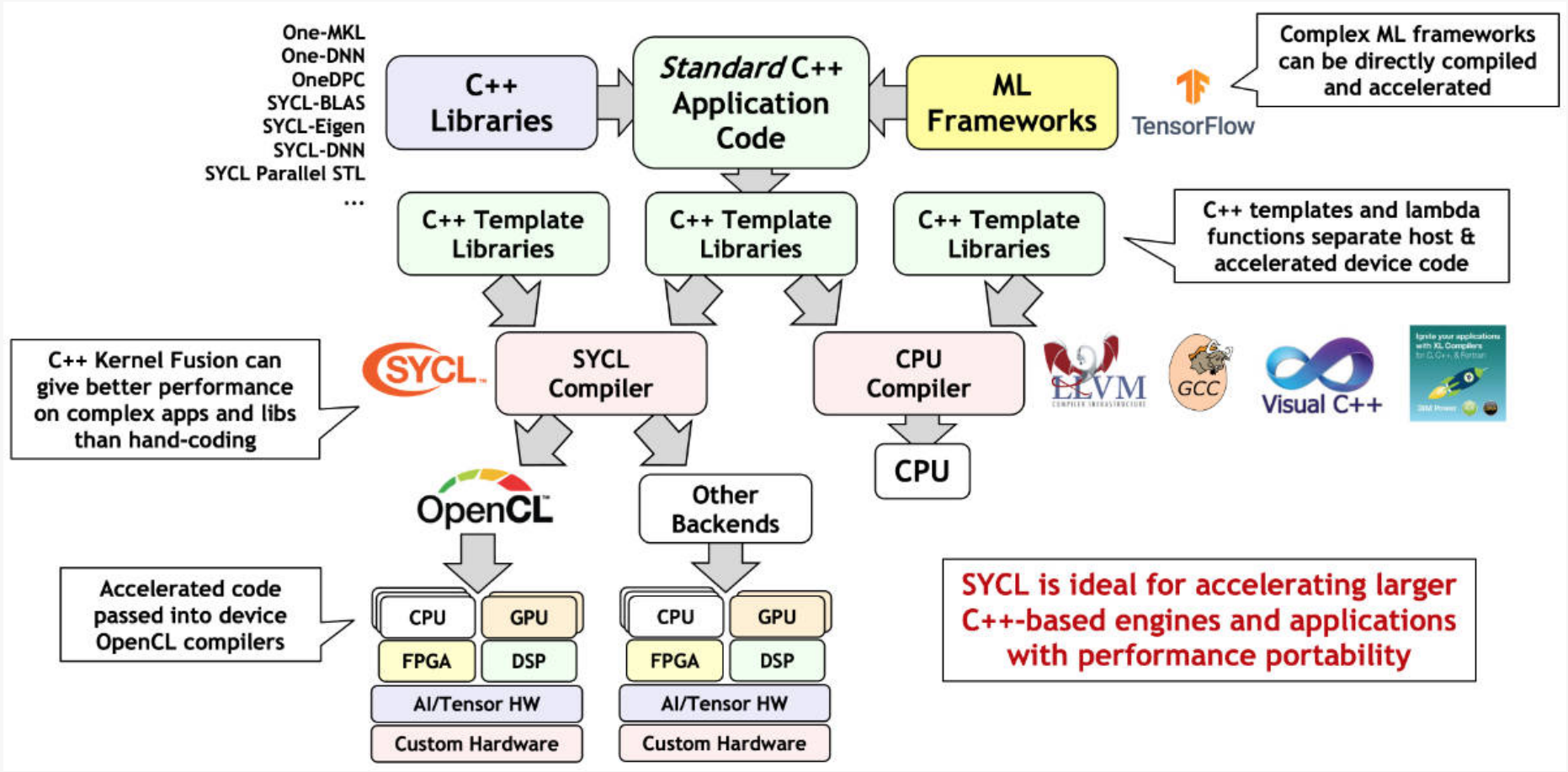
Fast-moving open collaboration feeding into the SYCL* standard

- ✓Open source implementation with goal of upstream LLVM
- ✓Extensions aim to become core SYCL*, or Khronos* extensions

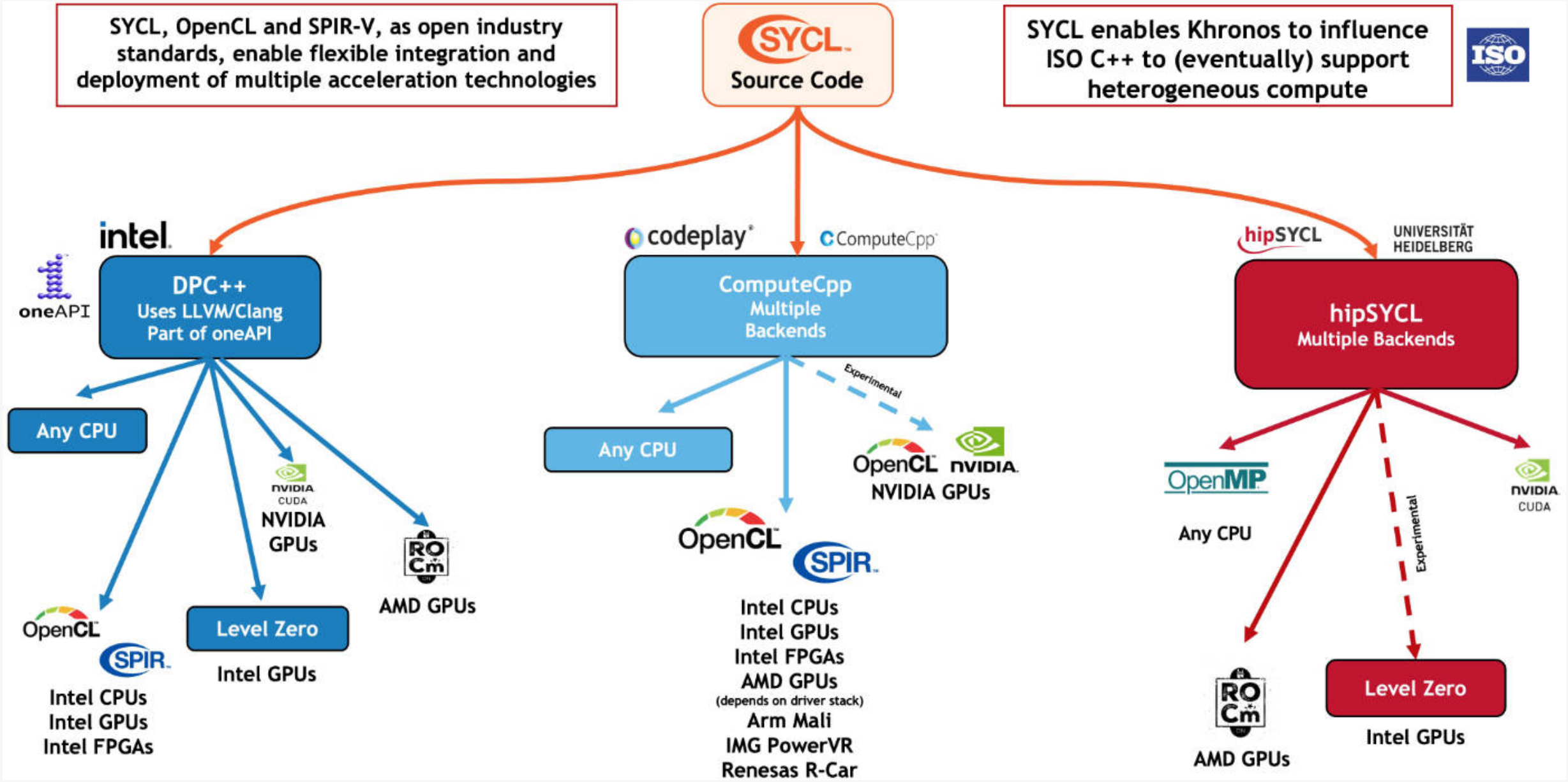
SYCL – A Portable Programming Model

A C++-based programming model for intra-node parallelism

- SYCL is a specification and “not” an implementation, currently compliant to C++17 ISO standards
- Cross-platform abstraction layer, heavily backed by industry
- Open-source, vendor agonistic
- Single-source model



SYCL – Compiler Players



SYCL @ NERSC

- Collaboration between ALCF, NERSC and Codeplay to enable support for NVIDIA A100 GPUs in LLVM DPC++/SYCL2020

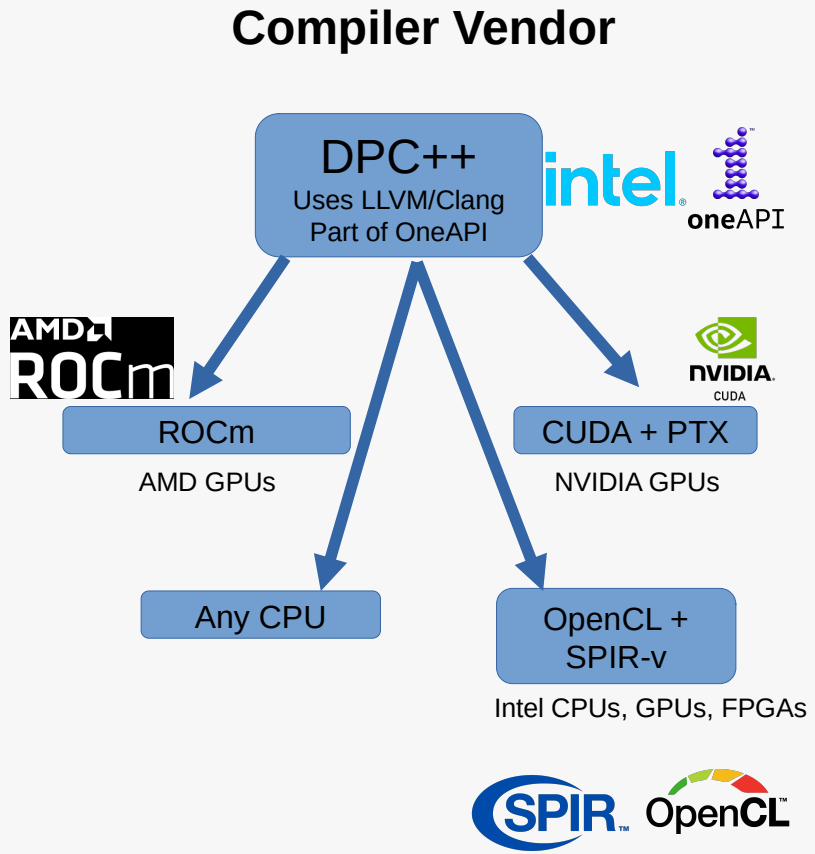
- Initial scope of work complete
  - support for tensor cores, USM, atomics, and more available

- Current focus on performance, upstreaming features to LLVM, tracking library support (e.g. FFT, oneMKL)

PrgEnv-llvm for CPE

NERSC has developed an additional PrgEnv which adds to the Cray Programming Environment (CPE) that HPE provides.

- LLVM compiler with support for OpenMP offload, SYCL



Argonne NATIONAL LABORATORY

NERSC National Energy Research Scientific Computing Center

codeplay

https://docs.nersc.gov/development/programming-models/sycl/

Credits: Brandon Cook (NERSC)

Devices

- Devices are the target for acceleration offload

SYCL sub-devices ↔ CUDA Multi-Instance GPU (MIG) mode ↔ OpenCL sub-devices

- Explicit Scaling: Partitioning of a SYCL root device into multiple sub-devices based on NUMA boundary
  - ✓ SYCL queues are further created based on “sub-devices” (better performance)
- Implicit Scaling: SYCL unpartitioned/root device is directly used to create a SYCL queue

```
sycl::queue Que;


// EXPLICIT SCALING (better performance)

sycl::platform platform(sycl::gpu_selector{});
auto const& gpu_devices = platform.get_devices(sycl::info::device_type::gpu);
for (auto const& gpuDev : gpu_devices) {
  if(gpu_dev.get_info<sycl::info::device::partition_max_sub_devices>() > 0) {
    auto SubDev = gpuDev.create_sub_devices<sycl::info::partition_property::partition_by_affinity_domain>(sycl::info::partition_affinity_domain::numa);

    for (auto const& tile : SubDev) {
      Que = sycl::queue(tile);
    }
  }
}

// IMPLICIT SCALING

sycl::platform platform(sycl::gpu_selector{});
auto const& gpu_devices = platform.get_devices(sycl::info::device_type::gpu);
for (auto const& gpuDev : gpu_devices) {
   Que = sycl::queue(gpuDev);
}
```

Queues & Contexts

- “SYCL Queues” provide mechanism to **submit** work to a **device** or **sub-device**
- “SYCL Contexts” is well known to be over-looked

```
sycl::queue Que;  // implicitly creates a SYCL context
```

- **Context**
  - Contexts are used for resources isolation and sharing
  - A SYCL context may consist of one or multiple devices
  - Both root-devices and sub-devices can be within single context (all from same SYCL platform)
  - Memory created can be shared only if their associated queue(s) are created using the same context

- **Queue** **(aka CUDA Stream)**
  - SYCL queue is always attached to a single device in a possibly multi-device context
    - ✓ Executes “**asynchronously**” from host code
    - ✓ SYCL queue can execute tasks enqueued in either “**in-order**” or “**out-of-order (default)**”
    - ✓ SYCL queue (in-order) is similar to CUDA stream (FIFO)

NERSC Perlmutter: System Overview

https://docs.nersc.gov/systems/perlmutter/

Perlmutter is a "**Shasta**" system: the non-compute **services are deployed as containers** using Kubernetes for orchestration.

Phase 1 (In Operation now)

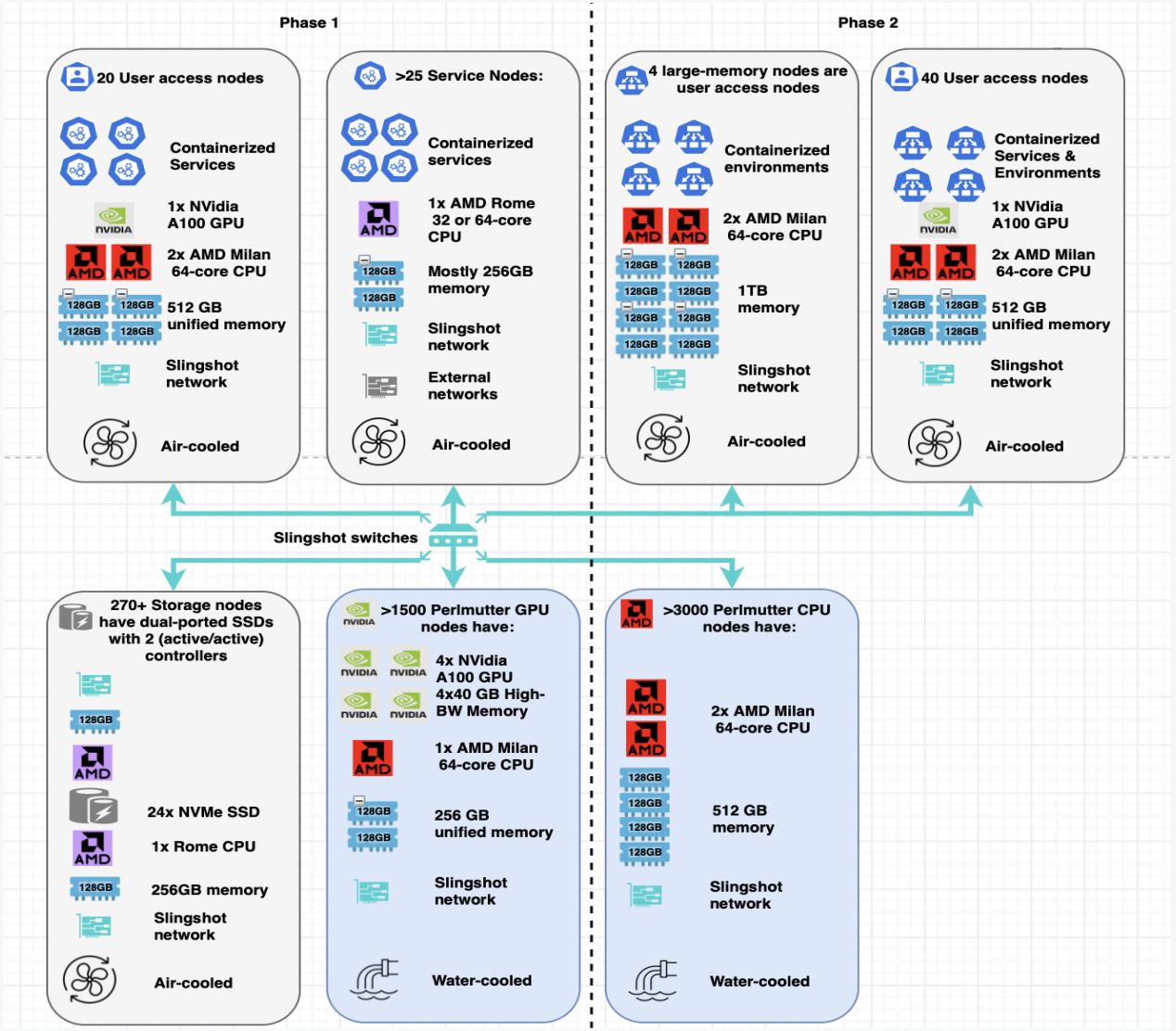
- Non-compute nodes (Login, Service nodes, Workflow and Large Memory)
- Storage (35 PB Lustre, all flash)
- GPU compute: 1536 nodes with 4x NVidia A100 GPUs
- Slingshot-10 Network

Phase 2 (Being installed)

- CPU compute: 3072 nodes with 2x AMD 64-core CPU

Slingshot-11 Network

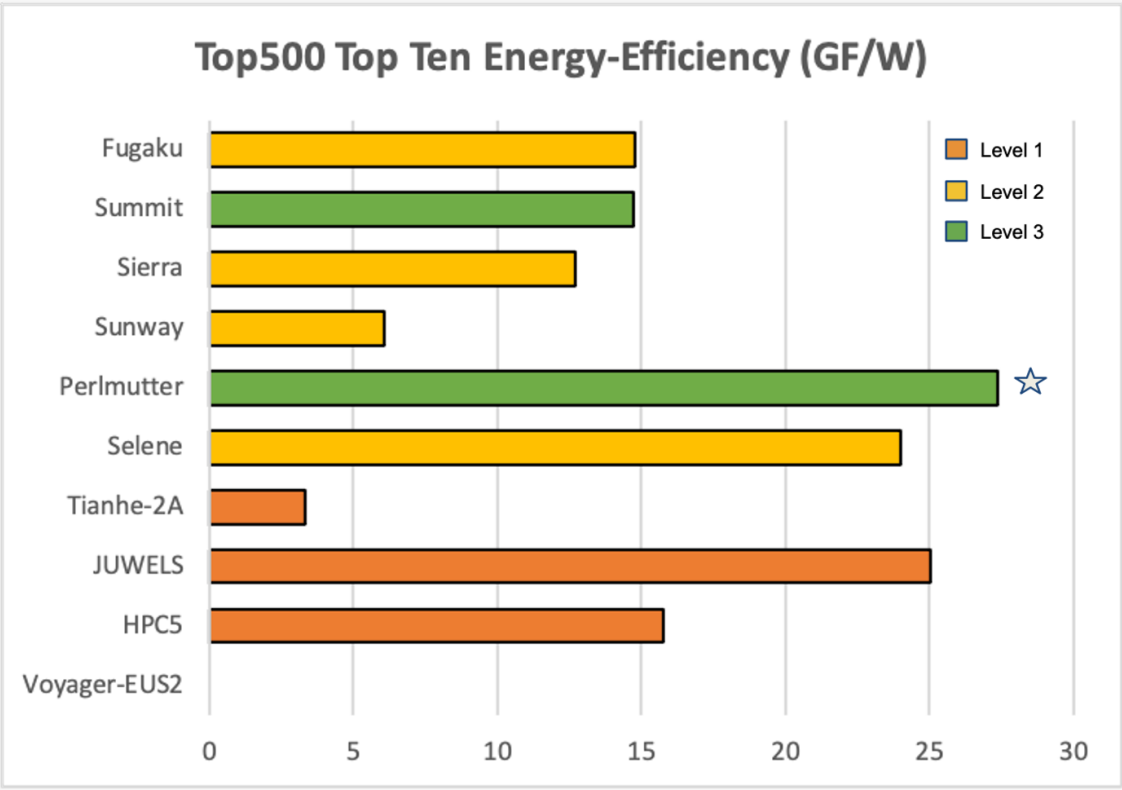
"Slingshot" network - improvements over Aries: faster, better traffic control, Ethernet-compatible



NERSC Perlmutter: System Overview

The only Top 500 Top 10 system in the Top 10 of the Green 500

https://docs.nersc.gov/systems/perlmutter/



Nov 2021:

HPL Performance: 70.87 PF
Energy Efficiency: 27.37 GF/W (core phase)

NERSC Perlmutter: Logging in

https://docs.nersc.gov/systems/perlmutter/

To get a training account:

Please register at https://iris.nersc.gov/train
Training code: b82m
If your organization is not listed in the dropdown menu, select "NERSC".

Note: (Important)
Please make sure to save **username** and **password**, it might be hard to retrieve otherwise

To login:

```
ssh username@perlmutter-p1.nersc.gov
Enter the password:
```

To request an interactive node (https://docs.nersc.gov/jobs/interactive/):

```
salloc -N 1 -C gpu -t 60 -c 10 -G 4 -A ntrain3 --reservation=atpsec_aug4
```

Build Your Own Compiler

- ✓ Build llvm-based SYCL compiler
- ✓ SYCL compiler for Nvidia hardware
- ✓ Nvidia A100 – NERSC Perlmutter

Build Your Own Compiler (~30 mins, plan accordingly)

Get the source code: (a big tar-ball, untar takes a while)

wget https://github.com/intel/llvm/archive/refs/tags/sycl-nightly/20220802.tar.gz
pigz -dc 20220802.tar.gz | tar xf -

Build & Install: (takes a while too)

```
module load cudatoolkit/11.5
module list

export DPCPP_HOME=$HOME

cd llvm-sycl-nightly-20220802

export CUDA_LIB_PATH=/opt/nvidia/hpc_sdk/Linux_x86_64/21.11/cuda/lib64/stubs
CC=`which gcc` CXX=`which g++` python $DPCPP_HOME/llvm-sycl-nightly-20220802/buildbot/configure.py --cuda --cmake-gen="Unix Makefiles" --
cmake-opt="-DCUDA_TOOLKIT_ROOT_DIR=/opt/nvidia/hpc_sdk/Linux_x86_64/21.11/cuda/11.5"

python $DPCPP_HOME/llvm-sycl-nightly-20220802/buildbot/compile.py
```

Where are my SYCL compilers installed ?

train515@nid001608:~/llvm-sycl-nightly-20220802/build/bin>

Experimental Support for CUDA and ROCm devices

Compiling With DPC++ for CUDA GPUs

The following command can be used to compile your code using DPC++ for CUDA backend:

```
clang++ -std=c++17 -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice -Xsycl-target-backend '--cuda-gpu-arch=sm_80' simple-sycl-app.cpp -o simple-sycl-app-cuda
```

Compiling With DPC++ for ROCm GPUs*

The following command can be used to compile your code using DPC++ for HIP backend:

```
clang++ -fsycl -fsycl-targets=amdgcn-amd-amdhsa -Xsycl-target-backend --offload-arch=gfx9xx simple-sycl-app.cpp -o simple-sycl-app-rocm
```

*Currently tested for ROCm 4.2.0, gfx906 and gfx908 for MI50 and MI100 GPU targets respectively

Test your BYOC SYCL Compiler

```
export PATH=$HOME/llvm-sycl-nightly-20220802/build/bin:$PATH
export LD_LIBRARY_PATH=$HOME/llvm-sycl-nightly-20220802/build/lib:$LD_LIBRARY_PATH
```

SYCL Source code to test:

```
#include <sycl/sycl.hpp>

int main() {
  auto const& gpu_devices = sycl::device::get_devices(sycl::info::device_type::gpu);
  std::cout << "Number of GPUs: " << gpu_devices.size() << std::endl;

  for(const auto& d : gpu_devices)
    std::cout << "Found device " << d.get_info<sycl::info::device::name>() << std::endl;

  sycl::queue Queue(sycl::gpu_selector{});
  return 0;
}
```

Expected output:

```
train515@nid001608:~> ./a.out
Number of GPUs: 4
Found device NVIDIA A100-SXM4-40GB
Found device NVIDIA A100-SXM4-40GB
Found device NVIDIA A100-SXM4-40GB
Found device NVIDIA A100-SXM4-40GB
```