AURORA EXASCALE SYSTEM OVERVIEW

THOMAS APPLENCOURT – APL@ANL.GOV

Argonne NATIONAL LABORATORY

DOE* HPC LANDSCAPE

	Years at the floor	Vendors
Summit	2018	IBM + NVIDIA
Sierra	2018	IBM + NVIDIA
Perlmutter	2020	AMD + NVIDIA
Aurora	**2021**	**Intel + Intel**
Frontier	2021	AMD + AMD
El Captain	2022	AMD + X

DOE HPC LANDSCAPE

- Heterogenous Computing (CPU + Accelerator)
- No, announced, NVIDIA Exascale System
  - Need to port your CUDA code

AURORA: AN INTEL-CRAY SYSTEM

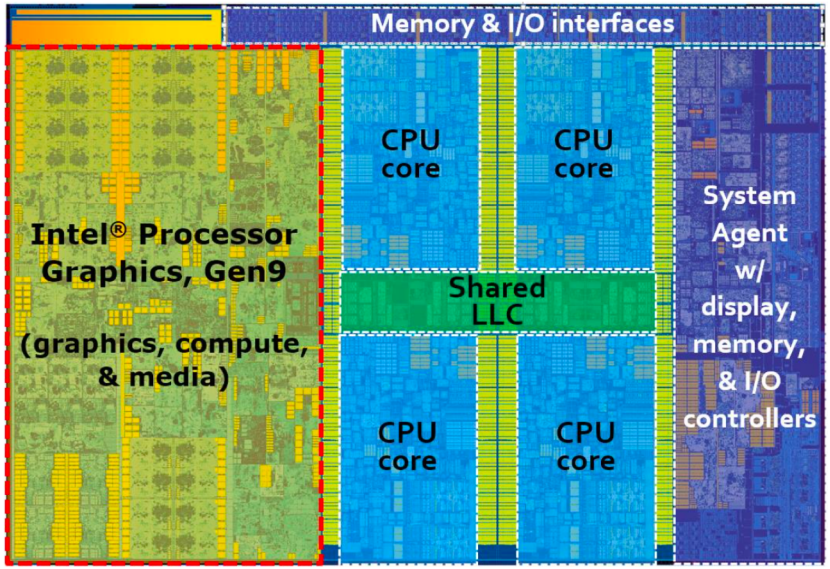
AURORA: A HIGH LEVEL VIEW

- Architecture:
  - Intel/Cray machine arriving at Argonne in 2021
  - Sustained Performance greater than 1 ExaFlops
  - Greater than 10 PB of total memory
  - Intel Xeon processors and Intel X^{e} GPUs
  - Cray Slingshot network and Shasta platform
- Software (Intel One API umbrella):
  - Intel compilers (C,C++,Fortran)
  - Programming models: SYCL*, OpenMP, OpenCL
  - Libraries: MKL, MKL-DNN, DAAL
  - Tools: VTune, Advisor
  - Python!
- IO:
  - Uses Lustre and Distributed Asynchronous Object Store IO (DAOS)
  - Greater than 230 PB of storage capacity and 25 TB/s of bandwidth

HARDWARE

INTEL GPUS

- Intel has been building GPUs integrated with CPUs for over a decade which are widely available in:
  - Laptops (e.g. MacBook pro)
  - Desktops
  - Servers
- Recent and upcoming integrated Generations:
  - “Gen 9” – current products
  - “Gen 11” – later this year in Ice Lake
- Double precision peak performance: 100-300 GF
  - Low by design due to power and space limits



Architecture components layout for an Intel Core i7 processor 6700K for desktop systems (91 W TDP, 122 mm)

INTEL GPU BUILDING BLOCKS

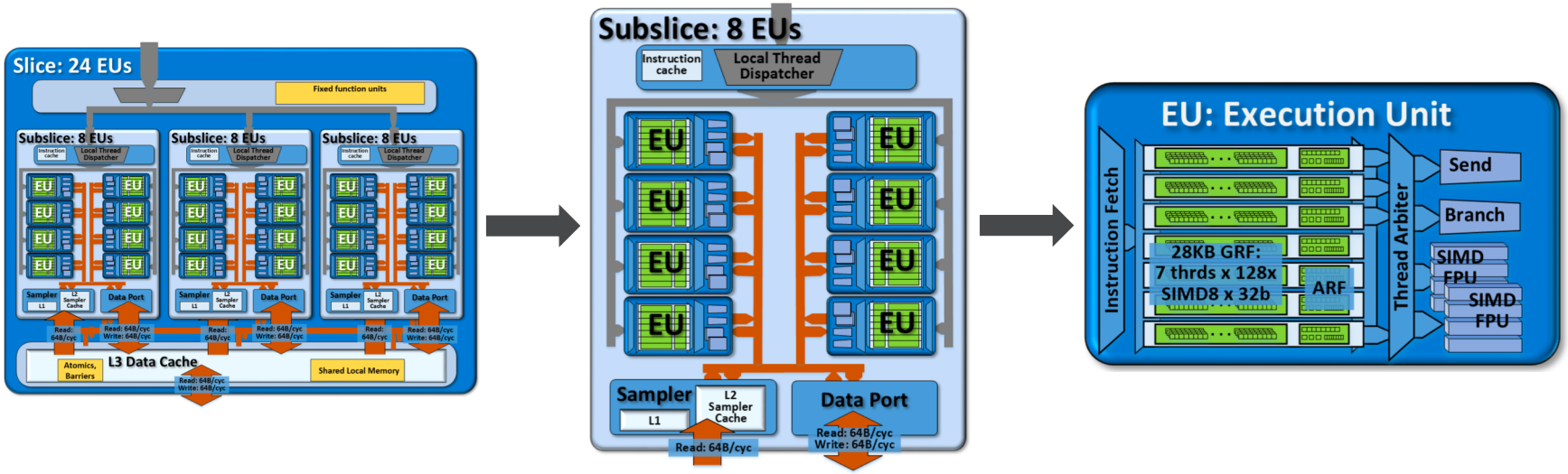
Slice

Sub-Slice

- Streaming Multiprocessors (NVDIA)
- Compute Unit (AMD)

Execution Unit (EU)

- Many cuda-core (NVIDA)



IRIS (GT4) GEN 9 BY THE NUMBERS

	Value	Derivation
Clock	1.15 GHz	
Slices	3	
EUs	72	3 slice * 3 sub-slices * 8 EUs
Hardware Threads	504	72 EUs * 7 threads
Concurrent Kernel Instances	16,128	504 thread * SIMD-32 compile
L3 Data Cache Size	1.5 MB	3 slices * 0.5 MB/slice
Max Shared Local Memory	576 KB	3 slice * 3 sub-slices * 64 KB/sub-slice
Last Level Cache Size	8 MB	
eDRAM size	128 MB	
32b float FLOPS	1152 FLOPS/cycle	72 EUs * 2 FPUs * SIMD-4 * (MUL + ADD)
64b float FLOPS	288 FLOPS/cycle	72 EUs * 1 FPU * SIMD-2 * (MUL + ADD)
32b integer IOPS	576 IOPS/cycle	72 EUs * 2 FPUs * SIMD-4

“BIG” GEN 9 CONFIGURATION

Hypothetically scale up a Gen 9 GPU by:

- Using 27 slices
- Upgrade the FP32 only FPU to a FP64 FPU
- Bump clock from 1.15 GHz to 1.5 GHz
- Connect it to HBM2 memory with 900 GB/s of bandwidth

Metric	Big Gen 9	V100
FP64 (Glops/s)	7900	7800
FP32 (Glops/s)	15860	15600
FP32 Function Units (CUDA cores)	5184	5120
Sub-slices (SS)/SM	81	80
Register File Size per SS/SM (KB)	224	256
BW (GB/s)	900	900

CRAY SLINGSHOT NETWORK

- Dragonfly topology:
  - Three network hops (for up to a quarter-million endpoints)
  - Only one hops uses optical cables
- High bandwidth switch:
  - 64 ports at 200 Gb/s in each direction
  - Total bandwidth of 25.6 Tb/s per switch
- Adaptive routing:
  - Avoids congestion by allowing packet to take different routes
  - Low diameter network allows responsive adaptive routine
- Congestion control:
  - Reduces message latency variation
  - Temporarily throttles injection from nodes causing congestion
- Traffic classes:
  - Allow overlaid virtual networks with traffic prioritization

PROGRAMMING MODELS

THREE PILLARS

Simulation	Data	Learning
HPC Languages	Productivity Languages	Productivity Languages
Directives	Big Data Stack	DL Frameworks
Parallel Runtimes	Statistical Libraries	Statistical Libraries
Solver Libraries	Databases	Linear Algebra Libraries

Compilers, Performance Tools, Debuggers

Math Libraries, C++ Standard Library, libc

I/O, Messaging

Containers, Visualization

Scheduler

Linux Kernel, POSIX

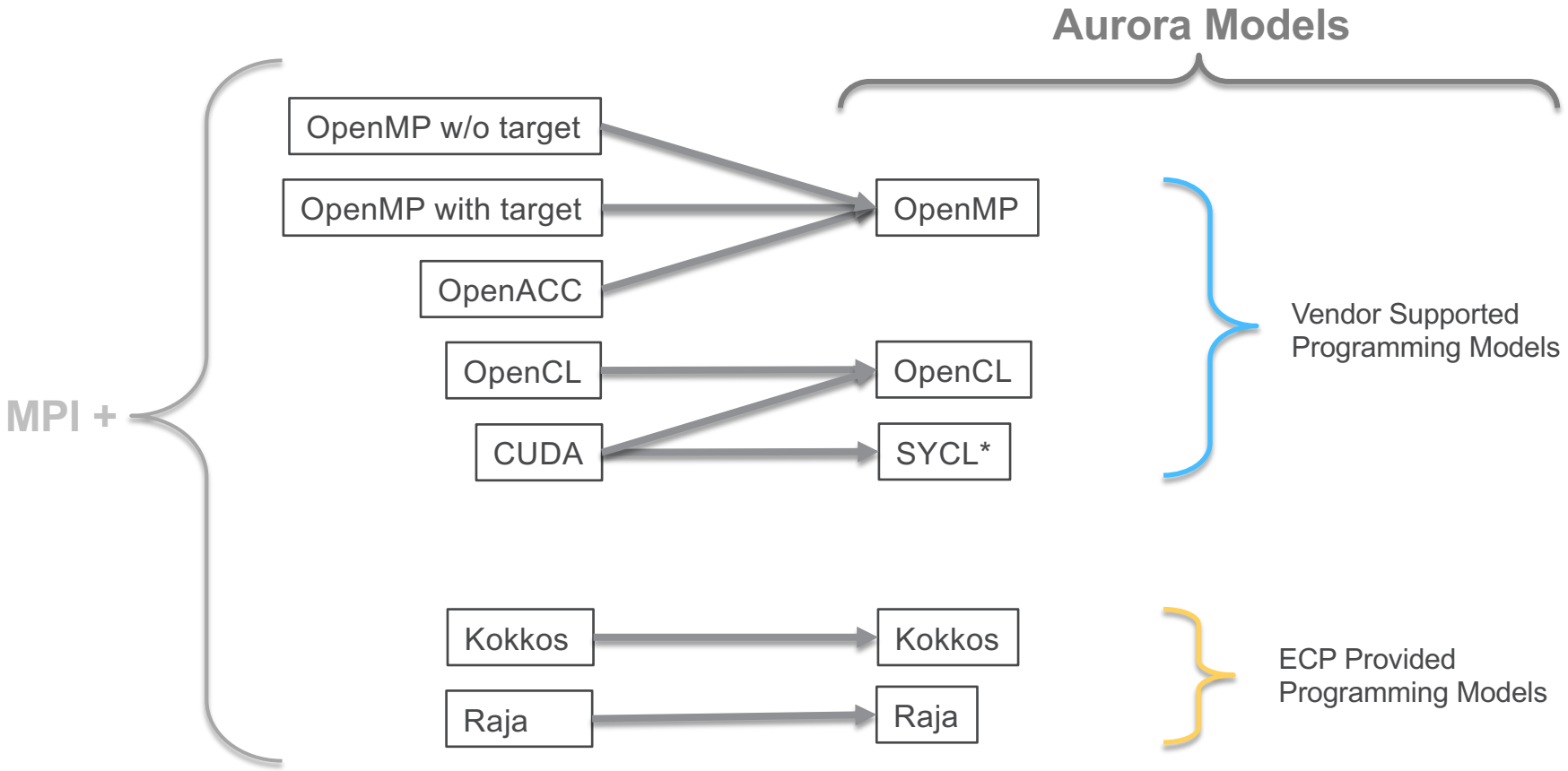
HETEROGENOUS SYSTEM PROGRAMMING MODELS

- Applications will be using a variety of programming models for Exascale:
  - CUDA
  - OpenCL
  - HIP
  - OpenACC
  - OpenMP
  - SYCL
  - Kokkos
  - Raja
- Not all systems will support all models
- Libraries may help you abstract some programming models.

AURORA PROGRAMMING MODELS

- Programming models available on Aurora:
  - CUDA
  - OpenCL
  - HIP
  - OpenACC
  - OpenMP
  - SYCL*
  - Kokkos
  - Raja

MAPPING OF EXISTING PROGRAMMING MODELS TO AURORA



OPENMP 5

- OpenMP 5 constructs will provide directives based programming model for Intel GPUs
- Available for C, C++, and Fortran
- A portable model expected to be supported on a variety of platforms
- Optimized for Aurora
- For Aurora OpenACC codes could be converted into OpenMP
  - ALCF staff will assist with conversion, training, and best practices
  - Automated translation possible through the clacc conversion tool (for C/C++)

OPENMP 4.5/5: FOR AURORA

- OpenMP 4.5/5 specification has significant updates to allow for improved support of accelerator devices

Offloading code to run on Aurora	Distributing iterations of the loop to threads	Controlling data transfer between devices
#pragma omp target *[clause[[,] clause],…]* *structured-block* **#pragma omp declare target** *declarations-definition-seq* **#pragma omp declare variant***(*variant-func-id) clause new-line* *function definition or declaration*	**#pragma omp teams** *[clause[[,] clause],…]* *structured-block* **#pragma omp distribute** *[clause[[,] clause],…]* *for-loops* **#pragma omp loop*** *[clause[[,] clause],…]* *for-loops*	**map** *([map-type:] list)* *map-type*:=alloc | tofrom | from | to | … **#pragma omp target data** *[clause[[,] clause],…]* *structured-block* **#pragma omp target update** *[clause[[,] clause],…]*

Runtime support routines:

- void **omp_set_default_device**(int dev_num)
- int **omp_get_default_device**(void)
- int **omp_get_num_devices**(void)
- int **omp_get_num_teams**(void)

Environment variables

- Control default device through OMP_DEFAULT_DEVICE

* denotes OMP 5

OPENMP 5 EXAMPLE

Distributes iterations to the threads, where each thread uses SIMD parallelism

Creates teams of threads in the target device

```
extern void init(float*, float*, int);
extern void output(float*, int);
void vec_mult(float*p, float*v1, float*v2, int N)
{
  int i;
  init(v1, v2, N);
  #pragma omp target teams distribute parallel for simd \
                          map(to: v1[0:N], v2[0:N]) map(from: p[0:N])
  for (i=0; i<N; i++)
  {
      p[i] = v1[i]*v2[i];
  }
  output(p, N);
}
```

Controlling data transfer

Hierarchical thread organization/parallelism

- Each OpenMP thread executes (vectorized) loop bodies
- OpenMP threads are organized into teams

The number of threads in a team and number of teams can be expressed with OpenMP clauses num_teams() and thread_limit()

https://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf

SYCL/DPC++

- SYCL
  - SYCL is a C++ based abstraction layer
    - Builds on OpenCL concepts (but single-source)
  - **Runs on today's CPUs and nVidia, AMD and Intel GPUs**
  - Tools exist for translating CUDA to SYCL
  - Current Implementations of SYCL:
    - ComputeCPP™ (www.codeplay.com)
    - Intel SYCL (github.com/intel/llvm)
    - triSYCL (github.com/triSYCL/triSYCL)
    - hipSYCL (github.com/illuhad/hipSYCL)
- DPC++
  - Intel extension of SYCL to support new innovative features
  - Not meant to be specific to Aurora
  - But Implementation will be optimized for Aurora
- Example:
  - https://github.com/alcf-perfengr/sycltrain

SYCL Example

Get a device

SYCL buffer using host pointer

Data accessor

Kernel

Buffer out of scope

```
#include <CL/sycl.hpp>
#include <iostream>
int main() {
  using namespace cl::sycl;
  int A[1024]; // Allocate data to be worked on

  default_selector selector; // Selectors determine which device to dispatch to.
  // Create your own or use {cpu,gpu,accelerator}_selector
  {
    queue myQueue(selector); // Create queue to submit work to, based on selector

    // Wrap data in a sycl::buffer
    buffer<cl_int, 1> bufferA(A, 1024);

    myQueue.submit([&](handler& cgh) {

        //Create an accessor for the sycl buffer.
        auto writeResult = bufferA.get_access<access::mode::discard_write>(cgh);

        // Kernel
        cgh.parallel_for<class hello_world>(range<1>{1024}, [=](id<1> idx) {
          writeResult[idx] = idx[0];
        }); // End of the kernel function
      }); // End of the queue commands
   }  // End of scope, wait for the queued work to stop.

  for (size_t i = 0; i < 1024; i++)
    std::cout<< "A[ " << i <<" ] = " << A[i] << std::endl;
  return 0;
}
```

OPENCL

- Open standard for heterogeneous device programming (CPU, GPU, FPGA)
  - Utilized for GPU programming
- Standardized by multi-vendor Khronos Group, V 1.0 released in 2009
  - AMD, Intel, nVidia, …
  - Many implementations from different vendors
- Intel implementation for GPU is Open Source (https://github.com/intel/compute-runtime)
- SIMT programming model
  - Distributes work by abstracting loops and assigning work to threads
  - Not using pragmas / directives for specifying parallelism
  - Similar model to CUDA
- Consists of a C compliant library with kernel language
  - Kernel language extends C
  - Has extensions that may be vendor specific
- Programming aspects:
  - Requires host and device code be in different files
  - Typically uses JIT compilation
- Example: https://github.com/alcf-perfengr/alcl

TOOLS AND LIBRARIES

VTUNE AND ADVISOR

- VTune
  - Widely used performance analysis tool
  - Currently supports analysis on Intel integrated GPUs
  - Will support future Intel GPUs
- Advisor
  - Provides roofline analysis
  - Offload analysis will identify components for profitable offload
    - Measure performance and behavior of original code
    - Model specific accelerator performance to determine offload opportunities
    - Considers overhead from data transfer and kernel launch

INTEL MKL

- Highly tuned algorithms
- Optimized for every Intel compute platform

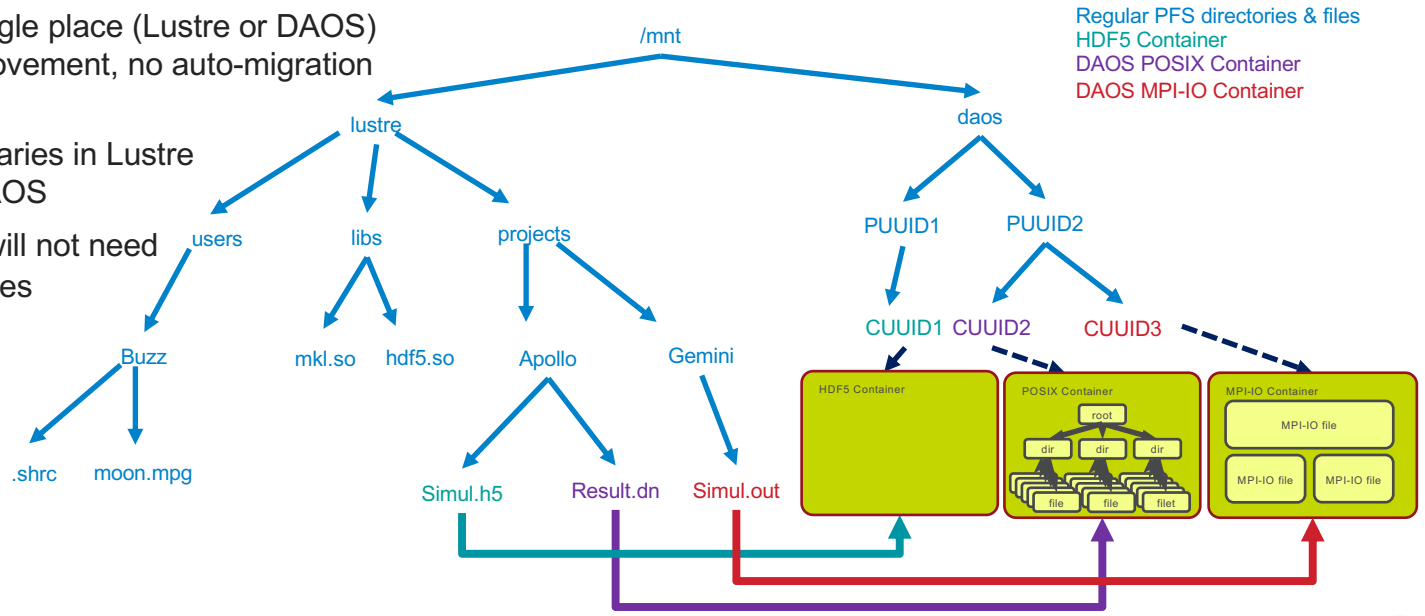
AI AND ANALYTICS

- MKL-DNN
  - High Performance Primitives to accelerate deep learning frameworks
  - Powers Tensorflow, PyTorch, MXNet, Intel Caffe, and more
  - Running on Gen9 today (via OpenCL)
- DAAL
  - Classical Machine Learning Algorithms
  - Easy to use one line daal4py Python interfaces
  - Powers Scikit-Learn
- Apache Arrow

I/O

DAOS & LUSTRE

- DAOS spec:
  - Greater than 230 PB of storage capacity
  - Greater than 25 TB/s of bandwidth
- User see single namespace which is in Lustre
  - “Links” point to DAOS containers within the /project directory
  - DAOS aware software interpret these links and access the DAOS containers
- Data resides in a single place (Lustre or DAOS)
  - Explicit data movement, no auto-migration
- Users keep:
  - Source and binaries in Lustre
  - Bulk data in DAOS
- Applications codes will not need changes in most cases



TRY IT NOW

www.anl.gov

Argonne NATIONAL LABORATORY

CURRENT HARDWARE AND SOFTWARE

- Argonne's Joint Laboratory for System Evaluation (JLSE) provides testbeds for Aurora – available today
  - 20 Nodes of Intel Xeons with Gen 9 Iris Pro [No NDA required]
    - SYCL, OpenCL, Intel VTune and Advisor are available.
  - Intel's Aurora SDK that already includes many of the compilers, libraries, tools, and programming models planned for Aurora. [NDA required]

- If you have workloads that are of great interest to you, please send them to us. We'll make sure they run well™ on Aurora!

QUESTIONS?