



Preparing to Program Aurora at Exascale

Argonne Leadership Computing Facility

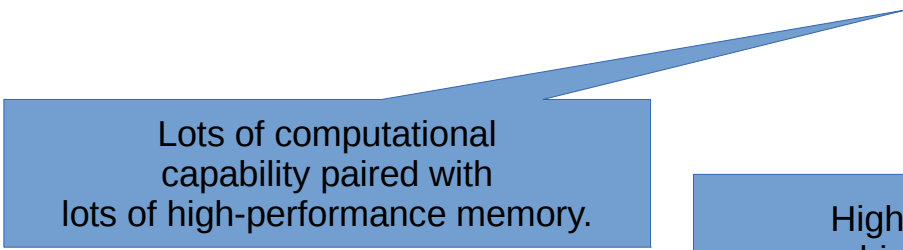
IWOCL, Apr. 28, 2020

Hal Finkel, et al.

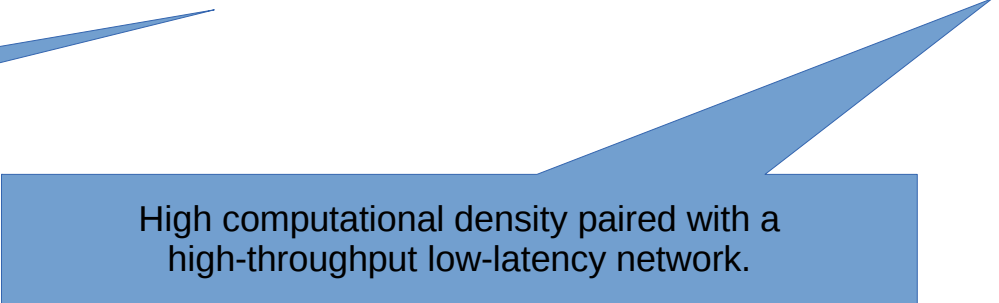
Scientific Supercomputing

What is (traditional) supercomputing?

Computing for large, tightly-coupled problems.



Lots of computational capability paired with lots of high-performance memory.

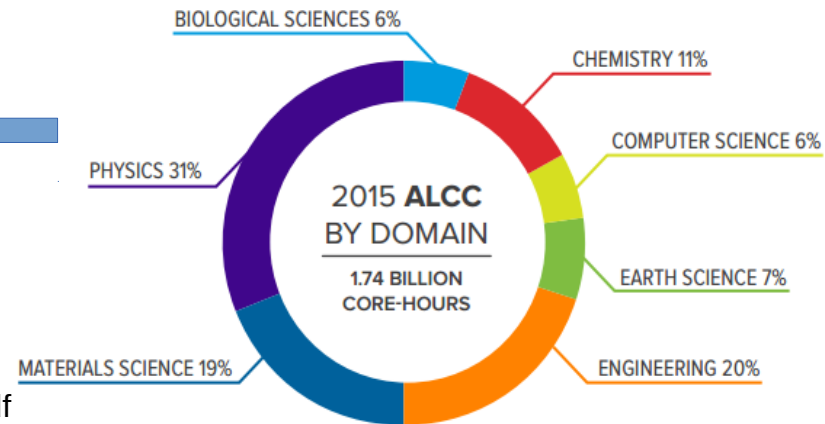
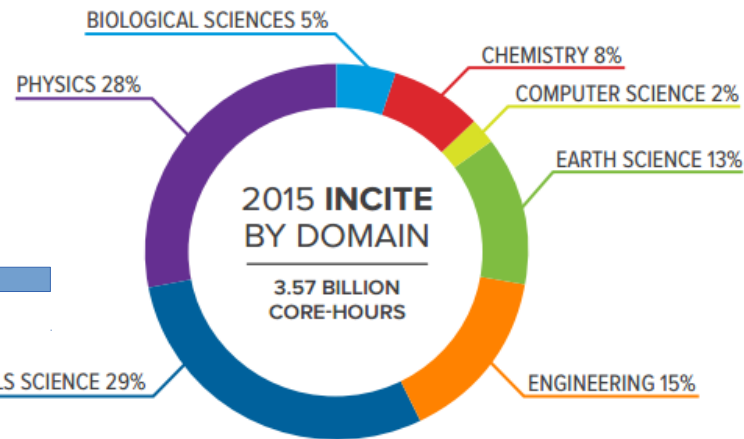


High computational density paired with a high-throughput low-latency network.

Many Scientific Domains



<https://www.alcf.anl.gov/files/alcfscibro2015.pdf>



Common Algorithm Classes in HPC

<i>Algorithm Science areas</i>	<i>Dense linear algebra</i>	<i>Sparse linear algebra</i>	<i>Spectral Methods (FFT)</i>	<i>Particle Methods</i>	<i>Structured Grids</i>	<i>Unstructured or AMR Grids</i>	<i>Data Intensive</i>
Accelerator Science		X	X	X	X	X	
Astrophysics	X	X	X	X	X	X	X
Chemistry	X	X	X	X			X
Climate			X		X	X	X
Combustion					X	X	X
Fusion	X	X		X	X	X	X
Lattice Gauge		X	X	X	X		
Material Science	X		X	X	X		

Common Algorithm Classes in HPC

<i>Algorithm</i> <i>Science areas</i>	<i>Dense linear algebra</i>	<i>Sparse linear algebra</i>	<i>Spectral Methods (FFT)s</i>	<i>Particle Methods</i>	<i>Structured Grids</i>	<i>Unstructured or AMR Grids</i>	<i>Data Intensive</i>
Accelerator Science		High performance memory system	High bisection bandwidth	High performance memory system	High flop/s rate	Low latency, efficient gather /scatter	Storage, Network Infrastructure
Astrophysics							
Chemistry	High Flop/s rate						
Climate							
Combustion							
Fusion							
Lattice Gauge							
Material Science							

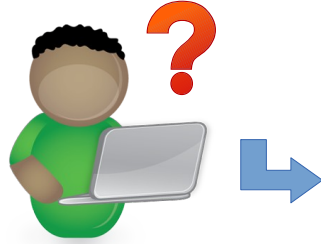
Upcoming Hardware

Toward The Future of Supercomputing

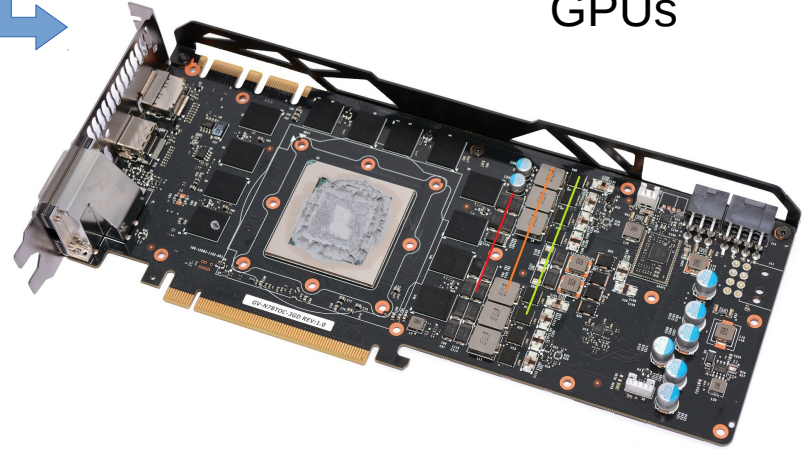
“Many Core” CPUs



<http://www.nextplatform.com/2015/11/30/inside-future-knights-landing-xeon-phi-systems/>



GPUs



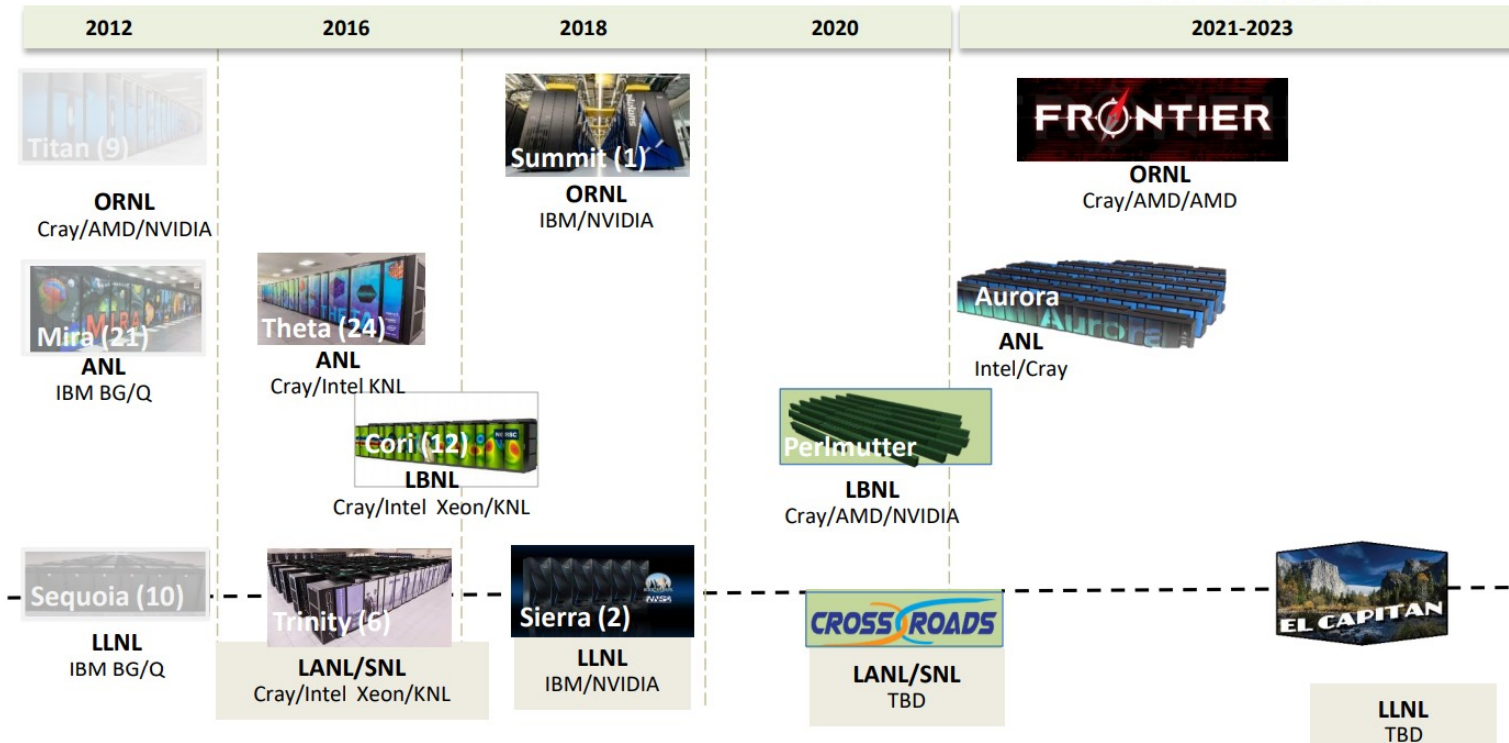
<https://forum.beyond3d.com/threads/nvidia-pascal-speculation-thread.55552/page-4>

All of our upcoming systems use GPUs!

Upcoming Systems

Pre-Exascale Systems [Aggregate Linpack (Rmax) = 323 PF!]

First U.S. Exascale Systems



(https://science.osti.gov/-/media/ascr/ascac/pdf/meetings/201909/20190923_ASCAC-Helland-Barbara-Helland.pdf)

Aurora: A High-level View

- ❑ Intel-Cray machine arriving at Argonne in 2021
 - ❑ Sustained Performance > 1Exaflops
- ❑ Intel Xeon processors and Intel Xe GPUs
 - ❑ 2 Xeons (Sapphire Rapids)
 - ❑ 6 GPUs (Ponte Vecchio [PVC])
- ❑ Greater than 10 PB of total memory
- ❑ Cray Slingshot fabric and Shasta platform
- ❑ Filesystem
 - ❑ Distributed Asynchronous Object Store (DAOS)
 - ❑ ≥ 230 PB of storage capacity
 - ❑ Bandwidth of > 25 TB/s
 - ❑ Lustre
 - ❑ 150 PB of storage capacity
 - ❑ Bandwidth of ~ 1 TB/s



Aurora Compute Node

- ❑ 2 Intel Xeon (Sapphire Rapids) processors
- ❑ 6 Xe Architecture based GPUs (Ponte Vecchio)
 - ❑ All to all connection
 - ❑ Low latency and high bandwidth
- ❑ 8 Slingshot Fabric endpoints
- ❑ Unified Memory Architecture across CPUs and GPUs

Unified Memory and
GPU ↔ GPU connectivity...

Important implications for the
programming model!

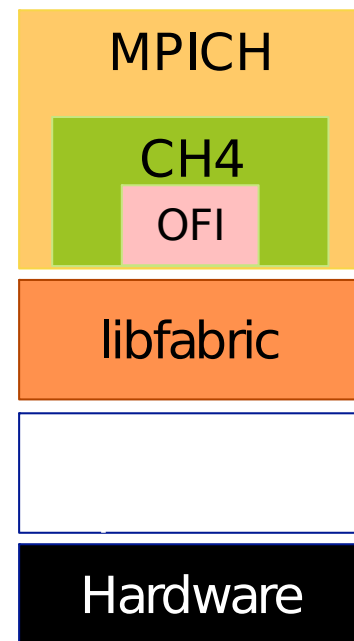
Programming Models (for Aurora)

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		

MPI on Aurora

- Intel MPI & Cray MPI
 - MPI 3.0 standard compliant
- The MPI library will be thread safe
 - Allow applications to use MPI from individual threads
 - Efficient MPI_THREAD_MULTIPLE (locking optimizations)
- Asynchronous progress in all types of nonblocking communication
 - Nonblocking send-receive and collectives
 - One-sided operations
- Hardware and topology optimized collective implementations
- Supports MPI tools interface
 - Control variables



Intel Fortran for Aurora

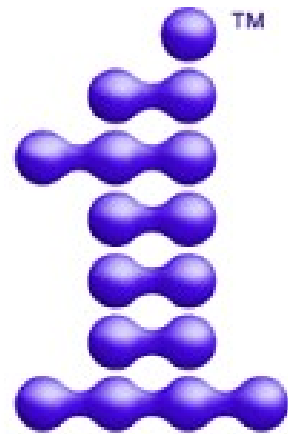
- ❑ Fortran 2008
- ❑ OpenMP 5

- ❑ A significant amount of the code run on present day machines is written in Fortran.
- ❑ Most new code development seems to have shifted to other languages (mainly C++).



oneAPI

- ❑ Industry specification from Intel (<https://www.oneapi.com/spec/>)
 - ❑ Language and libraries to target programming across diverse architectures (DPC++, APIs, low level interface)
- ❑ Intel oneAPI products and toolkits (<https://software.intel.com/ONEAPI>)
 - ❑ Implementations of the oneAPI specification and analysis and debug tools to help programming



oneAPI

Intel MKL - Math Kernel Library

- ❑ Highly tuned algorithms
 - ❑ FFT
 - ❑ Linear algebra (BLAS, LAPACK)
 - ❑ Sparse solvers
 - ❑ Statistical functions
 - ❑ Vector math
 - ❑ Random number generators

- ❑ Optimized for every Intel platform

- ❑ oneAPI MKL (oneMKL)
 - ❑ <https://software.intel.com/en-us/oneapi/mkl>

oneAPI beta includes
DPC++ support

AI and Analytics

- ❑ Libraries to support AI and Analytics
 - ❑ OneAPI Deep Neural Network Library (oneDNN)
 - ❑ High Performance Primitives to accelerate deep learning frameworks
 - ❑ Powers Tensorflow, PyTorch, MXNet, Intel Caffe, and more
 - ❑ Running on Gen9 today (via OpenCL)
 - ❑ oneAPI Data Analytics Library (oneDAL)
 - ❑ Classical Machine Learning Algorithms
 - ❑ Easy to use one line daal4py Python interfaces
 - ❑ Powers Scikit-Learn
 - ❑ Apache Spark MLlib

Heterogenous System Programming Models

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

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 (Aurora, Frontier, Perlmutter, ...)
- ❑ 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 accelerator	Distributing iterations of the loop to threads	Controlling data transfer between devices
<p>#pragma omp target [<i>clause</i>[[,] <i>clause</i>],...] <i>structured-block</i></p> <p>#pragma omp declare target <i>declarations-definition-seq</i></p> <p>#pragma omp declare variant*(<i>variant-func-id</i>) <i>clause new-line</i> <i>function definition or declaration</i></p>	<p>#pragma omp teams [<i>clause</i>[[,] <i>clause</i>],...] <i>structured-block</i></p> <p>#pragma omp distribute [<i>clause</i>[[,] <i>clause</i>], ...] <i>for-loops</i></p> <p>#pragma omp loop* [<i>clause</i>[[,] <i>clause</i>],...] <i>for-loops</i></p>	<p>map ([<i>map-type</i>:] <i>list</i>) <i>map-type</i>:=alloc tofrom from to ...</p> <p>#pragma omp target data [<i>clause</i>[[,] <i>clause</i>],...] <i>structured-block</i></p> <p>#pragma omp target update [<i>clause</i>[[,] <i>clause</i>], ...]</p>

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)

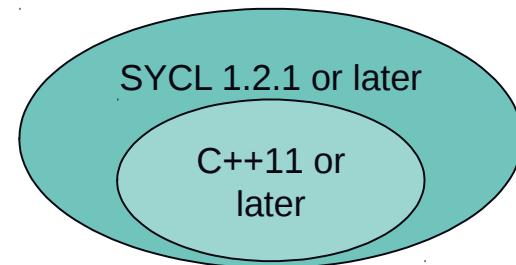
Environment variables

- Control default device through OMP_DEFAULT_DEVICE
- Control offload with OMP_TARGET_OFFLOAD

* denotes OMP 5

DPC++ (Data Parallel C++) and SYCL

- ❑ SYCL
 - ❑ Khronos standard specification
 - ❑ SYCL is a C++ based abstraction layer (standard C++11)
 - ❑ Builds on OpenCL **concepts** (but single-source)
 - ❑ *SYCL is designed to be as close to standard C++ as possible*
- ❑ Current Implementations of SYCL:
 - ❑ ComputeCPP™ (www.codeplay.com)
 - ❑ Intel SYCL (github.com/intel/llvm)
 - ❑ triSYCL (github.com/triSYCL/triSYCL)
 - ❑ hipSYCL (github.com/illuhad/hipSYCL)
- ❑ **Runs on today's CPUs and nVidia, AMD, Intel GPUs**



DPC++ (Data Parallel C++) and SYCL

SYCL

- ❑ Khronos standard specification
- ❑ SYCL is a C++ based abstraction layer (standard C++11)
- ❑ Builds on OpenCL **concepts** (but single-source)
- ❑ *SYCL is designed to be as close to standard C++ as possible*

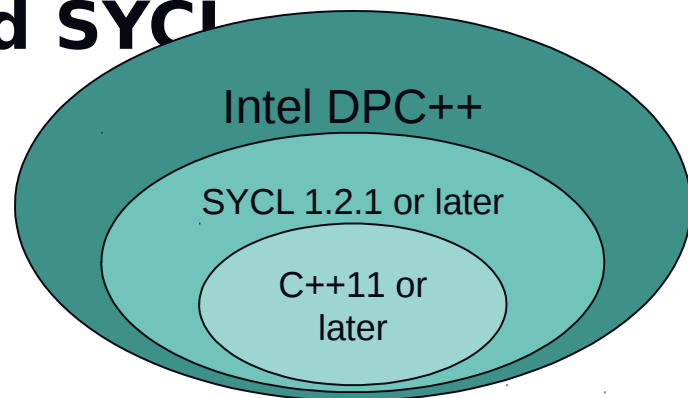
Current Implementations of SYCL:

- ❑ ComputeCPP™ (www.codeplay.com)
- ❑ Intel SYCL (github.com/intel/llvm)
- ❑ triSYCL (github.com/triSYCL/triSYCL)
- ❑ hipSYCL (github.com/illuhad/hipSYCL)

Runs on today's CPUs and nVidia, AMD, Intel GPUs

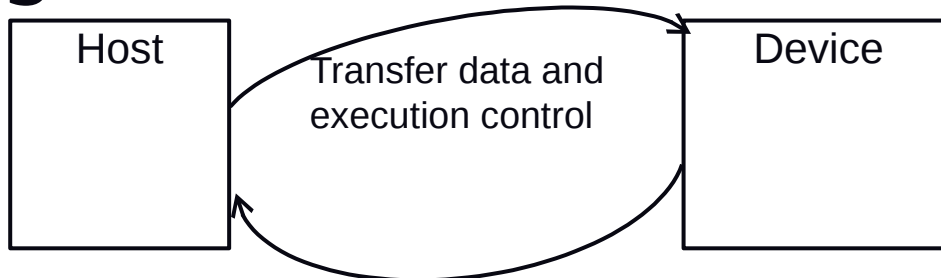
DPC++

- ❑ Part of Intel oneAPI specification
- ❑ Intel extension of SYCL to support new innovative features
- ❑ Incorporates SYCL 1.2.1 specification and Unified Shared Memory
- ❑ Add language or runtime extensions as needed to meet user needs



Extensions	Description
Unified Shared Memory (USM)	defines pointer-based memory accesses and management interfaces.
In-order queues	defines simple in-order semantics for queues, to simplify common coding patterns.
Reduction	provides reduction abstraction to the ND-range form of parallel_for.
Optional lambda name	removes requirement to manually name lambdas that define kernels.
Subgroups	defines a grouping of work-items within a work-group.
Data flow pipes	enables efficient First-In, First-Out (FIFO) communication (FPGA-only)

OpenMP 5



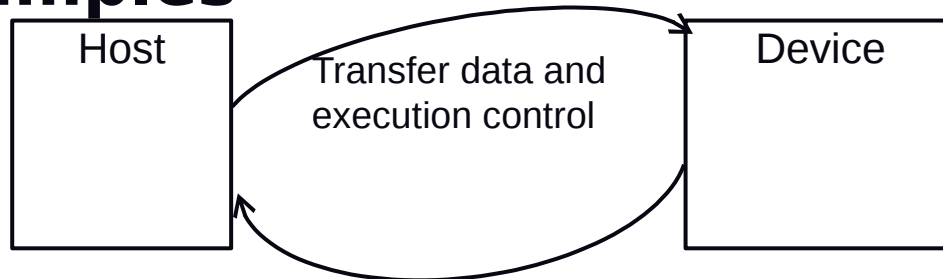
```
extern void init(float*, float*, int);
extern void output(float*, int);
void vec_mult(float*p, float*v1, float*v2, int N)
{
    int i;
    init(v1, v2, N);
    #pragma omp target teams distribute parallel for simd \
        map(to: v1[0:N], v2[0:N]) map(from: p[0:N])
    for (i=0; i<N; i++)
    {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

Creates teams of threads in the target device

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

Controlling data transfer

SYCL Examples



Get a device

SYCL buffer
using host
pointer

Data accessor

Kernel

Queue out of
scope

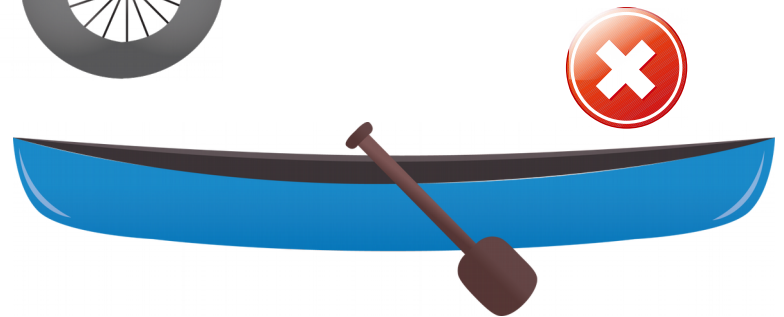
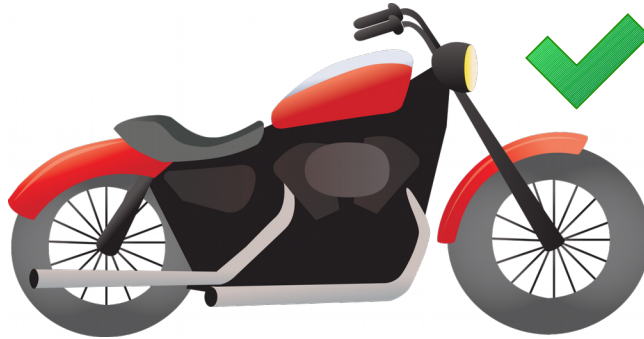
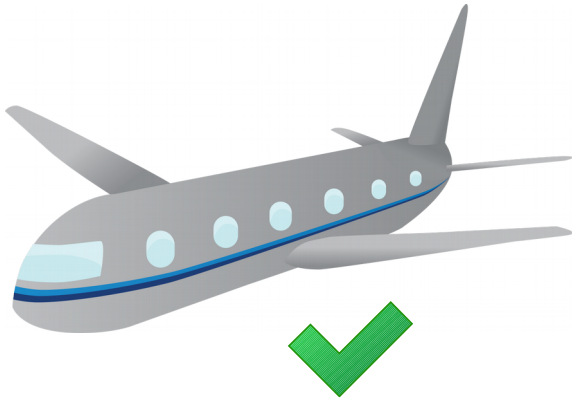
```
void foo( int *A ) {  
    default_selector selector; // Selectors determine which device to dispatch to.  
    {  
        queue myQueue(selector); // Create queue to submit work to, based on selector  
  
        // Wrap data in a sycl::buffer  
        buffer<cl_int, 1> bufferA(A, 1024);  
  
        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.  
    ...  
}
```

Performance Portability

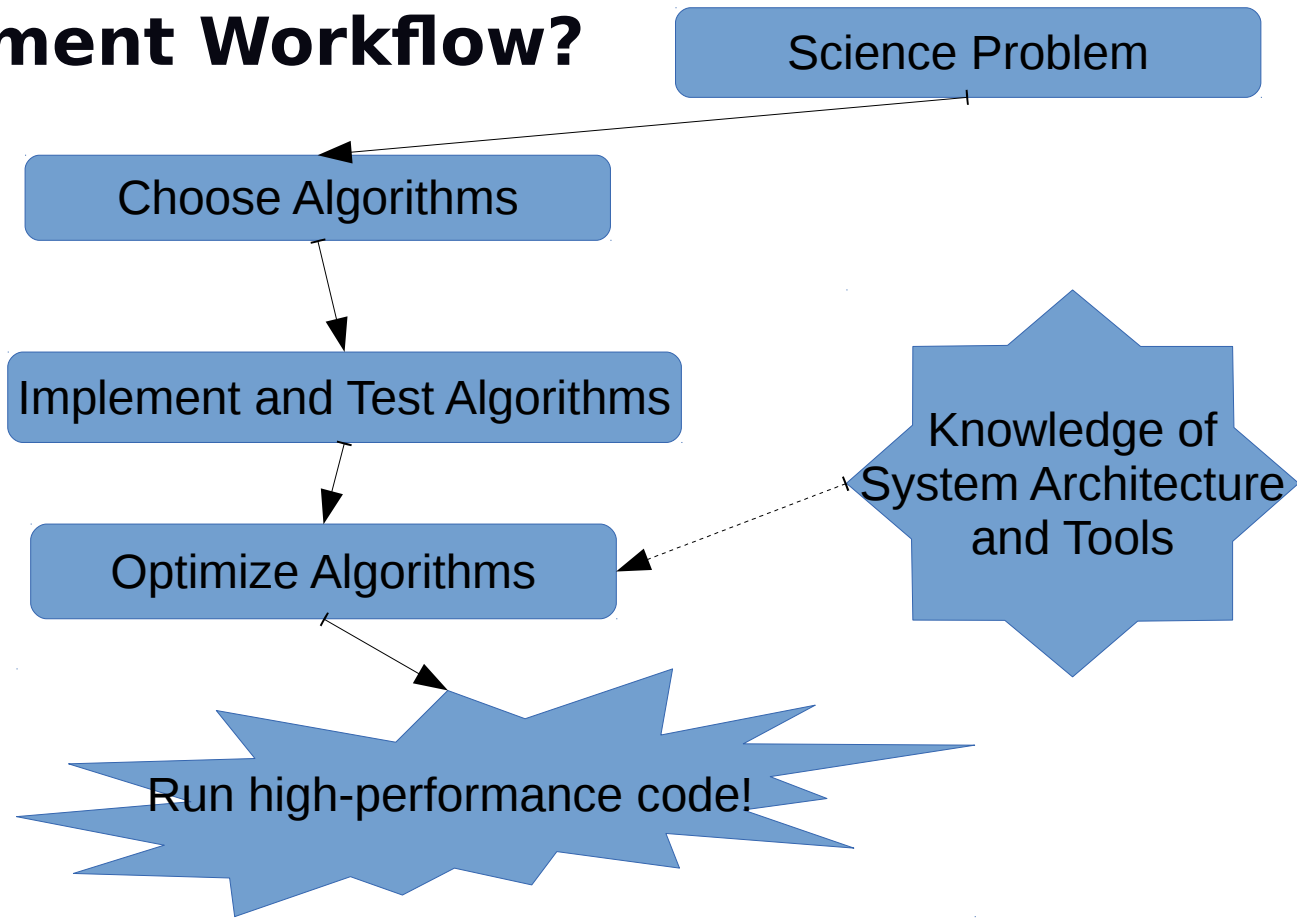
Performance Portability

A performance-portable application...

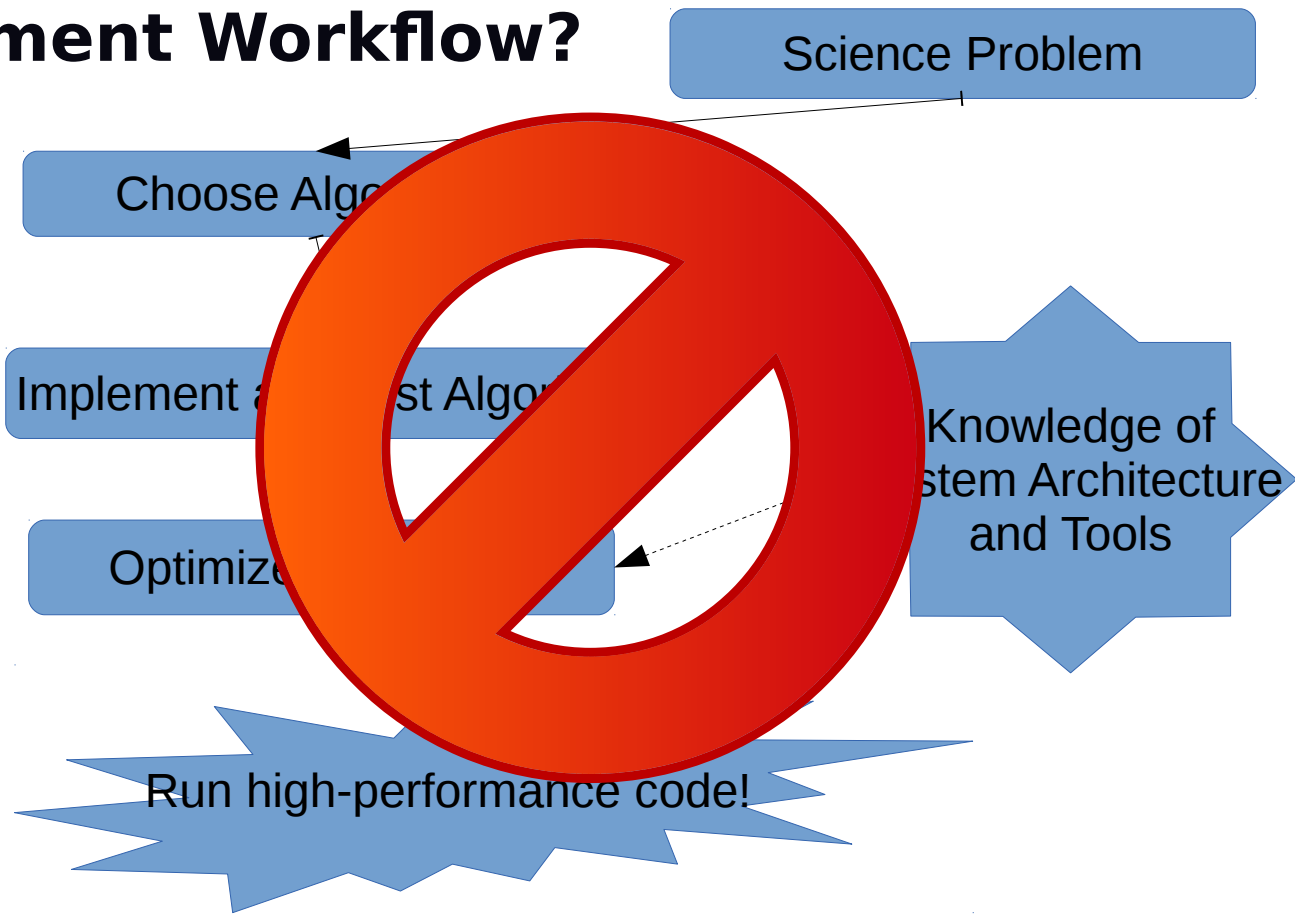
- 1) Is Portable
- 2) Runs on diverse architectures with reasonable performance



The Development Workflow?



The Development Workflow?



Real Workflow...

Science Problem

Choose Algorithms
For the Target Architectures

Implement and Test Algorithms

Optimize Algorithms

Run high-performance code!

Knowledge of
System Architecture
and Tools

Trade-offs between:

- Basis functions
- Resolution
- Lagrangian vs. Eulerian representations
- Renormalization and regularization schemes
- Solver techniques
- Evolved vs computed degrees of freedom
- And more...

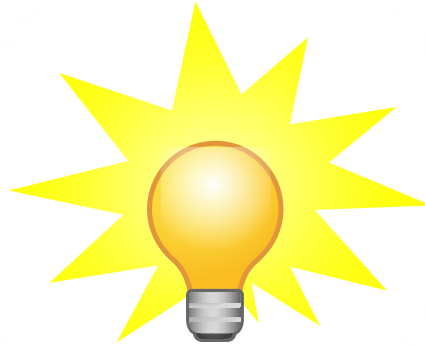
Cannot be made by a compiler!



Performance Portability is Possible!

Does this mean that performance portability is impossible?

No, but it does mean that performance-portable applications tend to be highly parameterizable.

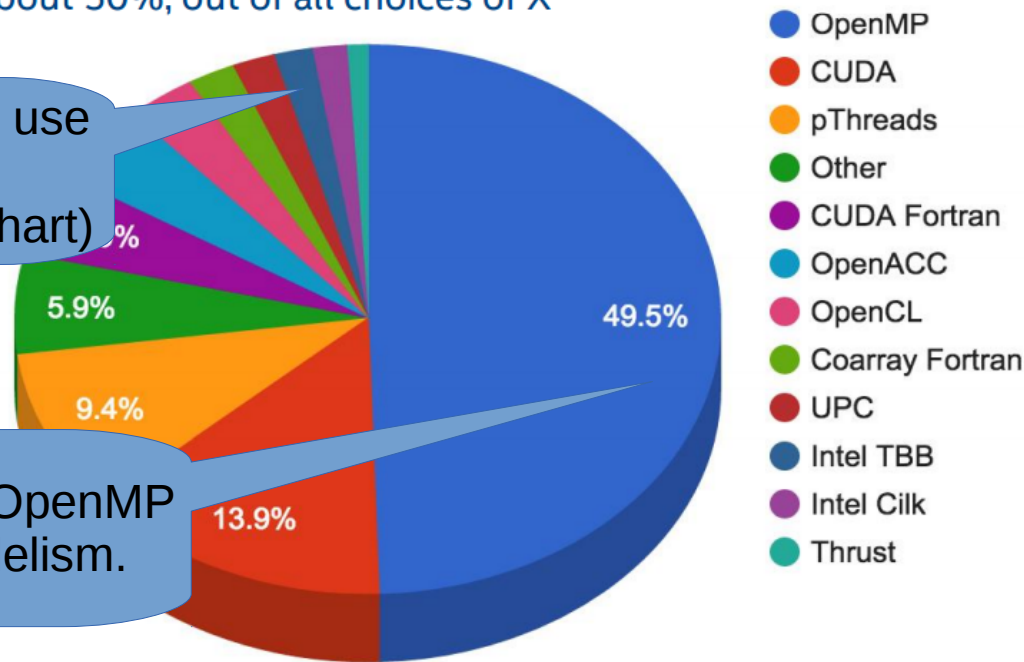


On the Usage of Abstract Models

✓ OpenMP is about 50%, out of all choices of X

A minority of applications use abstraction libraries (TBB and Thrust on this chart)

In 2015, many codes use OpenMP directly to express parallelism.



Courtesy of Yun (Helen) He, Alice Koniges, et. al., (NERSC) at OpenMPCon'2015

On the Usage of Abstract Models

But this is changing...

- We're seeing even greater adoption of OpenMP, but...
- Many applications are not using C++11/14/17/20/23, but still gaining in popularity.

Can use OpenMP and/or other compiler directives under the hood, but probably DPC++/HIP/CUDA.

Use of C++ Lambdas.

- Well established libraries such as TBB and Thrust.
- RAJA (<https://github.com/LLNL/RAJA>)

```
RAJA::ReduceSum<reduce_policy, double> piSum(0.0);

RAJA::forall<execute_policy>(begin, numBins, [=](int i) {
    double x = (double(i) + 0.5) / numBins;
    piSum += 4.0 / (1.0 + x * x);
});
```

- Kokkos (<https://github.com/kokkos>)

```
int sum = 0;
// The KOKKOS_LAMBDA macro replaces
// the capture-by-value clause [=].
// It also handles any other syntax
// needed for CUDA.
Kokkos::parallel_reduce (n, KOKKOS_LAMBDA (const int i,
int& lsum) {
    lsum += i*i;
}, sum);
```

On the Usage of Abstract Models

And starting with C++17, the standard library has parallel algorithms too...

Table 2 — Table of parallel algorithms

adjacent_difference	adjacent_find	all_of	any_of
copy	copy_if	copy_n	count
count_if	equal	exclusive_scan	fill
fill_n	find	find_end	find_first_of
find_if	find_if_not	for_each	for_each_n
generate	generate_n	includes	inclusive_scan
inner_product	inplace_merge	is_heap	is_heap_until
is_partitioned	is_sorted	is_sorted_until	lexicographical_compare
max_element	merge	min_element	minmax_element
mismatch	move	none_of	nth_element
partial_sort	partial_sort_copy	partition	partition_copy
reduce	remove	remove_copy	remove_copy_if
remove_if	replace	replace_copy	replace_copy_if
replace_if	reverse	reverse_copy	rotate
rotate_copy	search	search_n	set_difference
set_intersection	set_symmetric_difference	set_union	sort
stable_partition	stable_sort	swap_ranges	transform
transform_exclusive_scan	transform_inclusive_scan	transform_reduce	uninitialized_copy
uninitialized_copy_n	uninitialized_fill	uninitialized_fill_n	unique
unique_copy			

[Note: Not all algorithms in the Standard Library have counterparts in [Table 2](#). — end note]

```
// For example:  
std::sort(std::execution::par_unseq, vec.begin(), vec.end()); // parallel and vectorized
```

Compiler Optimizations for Parallel Code...

Why can't programmers just write the code optimally?

- Because what is optimal is different on different architectures.
- Because programmers use abstraction layers and may not be able to write the optimal code directly:

```
in library1:  
void foo() {  
    std::for_each(std::execution::par_unseq, vec1.begin(), vec1.end(), ...);  
}
```

```
in library2:  
void bar() {  
    std::for_each(std::execution::par_unseq, vec2.begin(), vec2.end(), ...);  
}
```

```
foo(); bar();
```

Compiler Optimizations for Parallel Code...

```
void foo(double * restrict a, double * restrict b, etc.) {  
#pragma omp parallel for  
    for (i = 0; i < n; ++i) {  
        a[i] = e[i]*(b[i]*c[i] + d[i]) + f[i];  
        m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];  
    }  
}
```



Split the loop

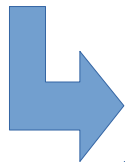


Or should we fuse instead?

```
void foo(double * restrict a, double * restrict b, etc.) {  
#pragma omp parallel for  
    for (i = 0; i < n; ++i) {  
        a[i] = e[i]*(b[i]*c[i] + d[i]) + f[i];  
    }  
#pragma omp parallel for  
    for (i = 0; i < n; ++i) {  
        m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];  
    }  
}
```

Compiler Optimizations for Parallel Code...

```
void foo(double * restrict a, double * restrict b, etc.) {  
  #pragma omp parallel for  
    for (i = 0; i < n; ++i) {  
        a[i] = e[i]*(b[i]*c[i] + d[i]) + f[i];  
    }  
  #pragma omp parallel for  
    for (i = 0; i < n; ++i) {  
        m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];  
    }  
}
```



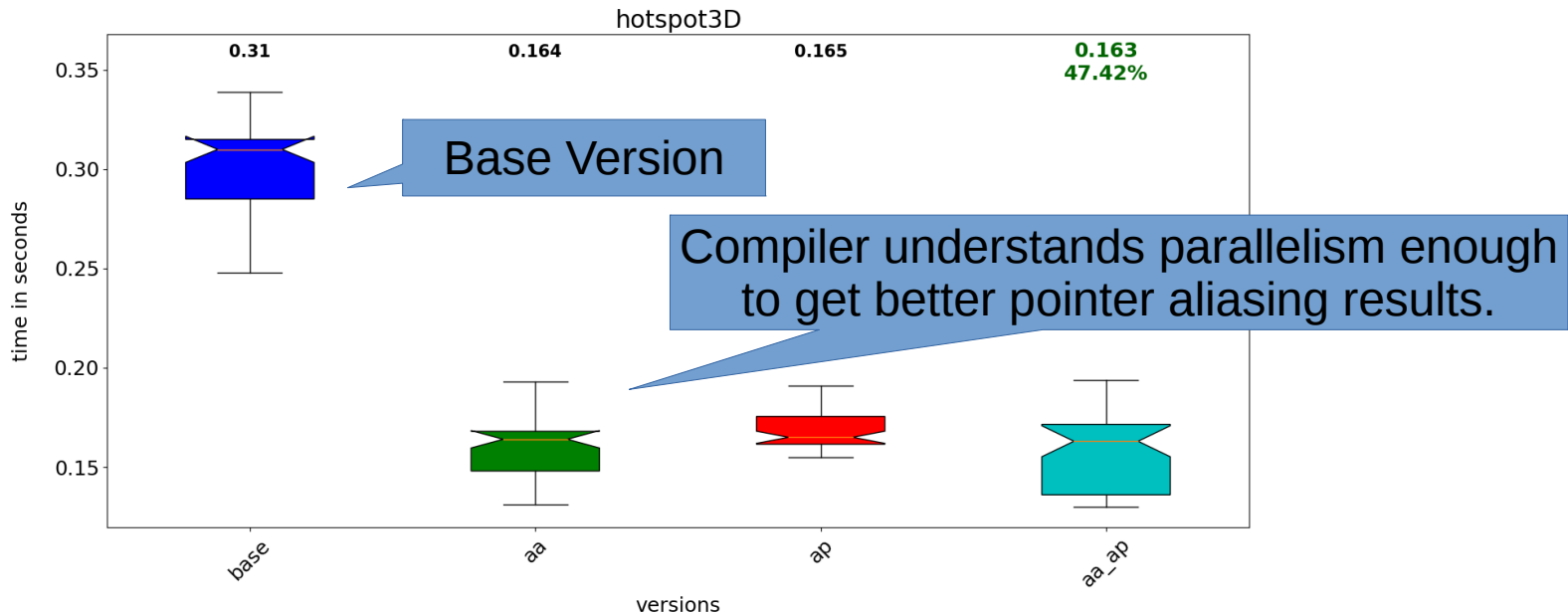
(we might want to fuse
the parallel regions)

```
void foo(double * restrict a, double * restrict b, etc.) {  
  #pragma omp parallel  
  {  
    #pragma omp for  
      for (i = 0; i < n; ++i) {  
          a[i] = e[i]*(b[i]*c[i] + d[i]) + f[i];  
      }  
    #pragma omp for  
      for (i = 0; i < n; ++i) {  
          m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];  
      }  
  }  
}
```

Compiler Understanding Parallelism: It Can Help

Rodinia - hotspot3D

./3D 512 8 100 ../data/hotspot3D/power_512x8 ../data/hotspot3D/temp_512x8



Intel core i9, 10 cores, 20 threads, 51 runs, with and without

- aa => alias attribute propagation
- ap => argument privatization

(Work by Johannes Doerfert, see our IWOMP 2018 paper)

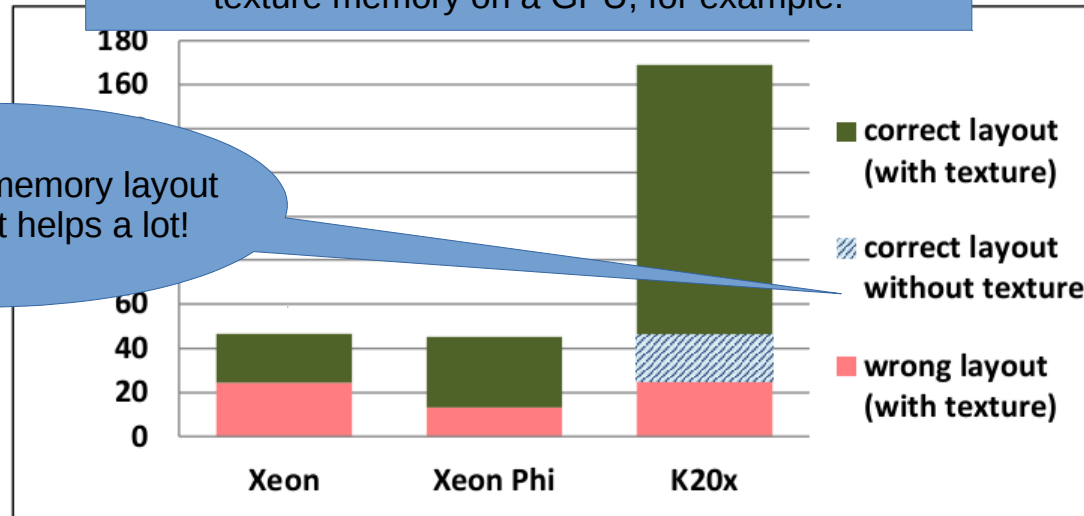
Memory Layout and Placement

It is really hard for compilers to change memory layouts and generally determine what memory is needed where. The Kokkos C++ library has memory placement and layout policies:

```
View<const double ***, Layout, Space, MemoryTraits<RandomAccess>> name (...);
```

Constant random-access data might be put into texture memory on a GPU, for example.

Using the right memory layout and placement helps a lot!



- Large loss in performance with wrong layout

So Where Does This Leave Us?

As you might imagine, nothing is perfect yet...

	OpenMP	DPC++	Kokkos / RAJA
Language	Simple directives have yielded to complicated directives	Modern C++, simple cases will become simpler over time	Modern C++
Default Execution Model	Fork-Join	Work Queue (Probably better for expressing scalable parallelism)	Fork-Join
Compiler Optimization Potential	High	Low (Dynamic work queue obscures structure)	Medium (Greatly depends on underlying backend)

So Where Does This Leave Us?

As you might imagine, nothing is perfect yet...

	OpenMP	DPC++	Kokkos / RAJA
Integrate With Highly-Parameterized Code	Low / Medium	High	High
Helps With Data Layout	No	No (Not Yet)	Yes
Good Accelerator-to-Accelerator Transfer / Dispatch	No (Not Yet)	No (Not Yet)	No (Not Yet)

Conclusion

Conclusions

- Future supercomputers will continue to advance scientific progress in a variety of domains.
- Applications will rely on high-performance libraries as well as parallel-programming models.
- DPC++/SYCL will be a critical programming model on future HPC platforms.
- We will continue to understand the extent to which compiler optimizations assist the development of portably-performant applications vs. the ability to explicitly parameterize and dynamically compose the implementations of algorithms.
- Parallel programming models will continue to evolve: support for data layouts and less-host-centric models will be explored.



Acknowledgements

- ❑ Argonne Leadership Computing Facility and Computational Science Division Staff
- ❑ This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.
- ❑ This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

A blue-tinted photograph of a modern building courtyard. The building has multiple levels with balconies and railings. In the foreground, there is a tree on the left and various plants and rocks in the courtyard. The text "Thank You" is overlaid in the center.

Thank You