Exploiting the Node: OpenMP for All?

Barbara Chapman
Stony Brook University
Brookhaven National Lab

eResearch, Auckland, February, 2019

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Acknowledgments

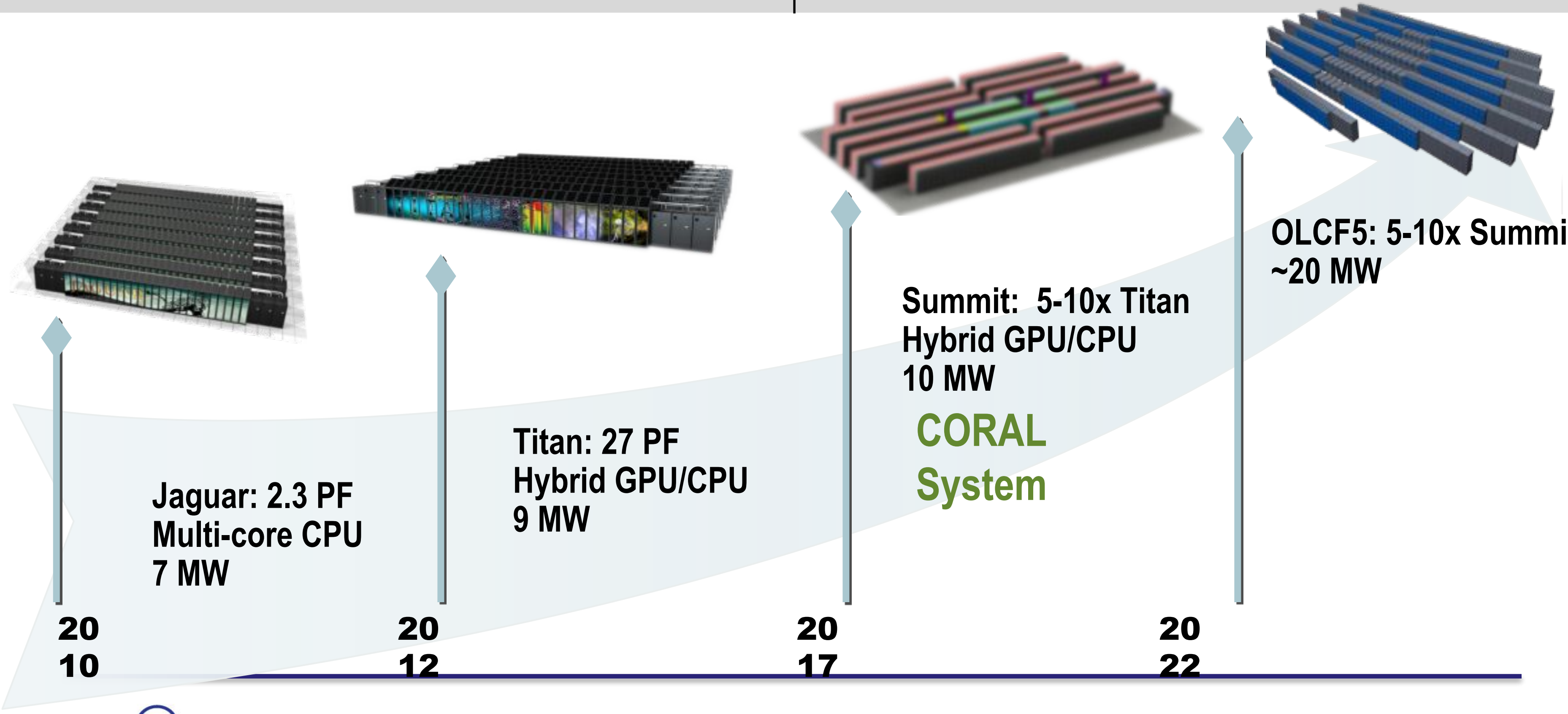
This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations - the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative.

www.ExascaleProject.org

Roadmap to Exascale (ORNL View)

Since clock-rate scaling ended in 2003, HPC performance has been achieved through **increased parallelism**. Jaguar scaled to 300,000 CPU cores.

Titan and beyond deliver **hierarchical parallelism** with very powerful nodes. MPI plus thread level parallelism through **OpenMP or OpenACC** plus vectors



2010

Jaguar: 2.3 PF
Multi-core CPU
7 MW

2012

Titan: 27 PF
Hybrid GPU/CPU
9 MW

2017

Summit: 5-10x Titan
Hybrid GPU/CPU
10 MW

CORAL
System

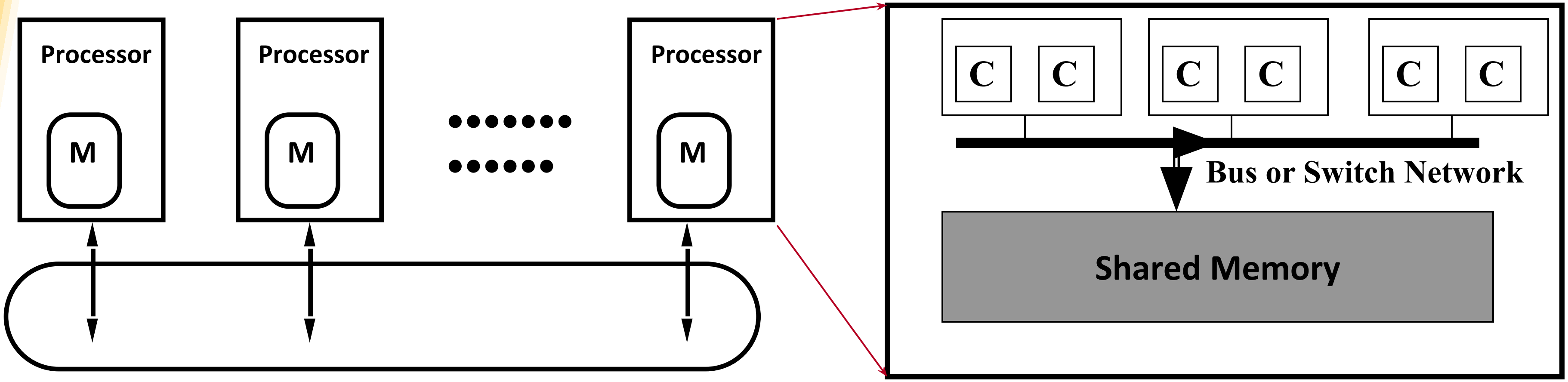
2022

OLCF5: 5-10x Summit
~20 MW

HPC Cluster Architecture

- Distributed Memory Machine (HPC cluster)
- Shared Memory Node (SMP)
- Accelerators (GPUs)

SMP Multicore Architecture



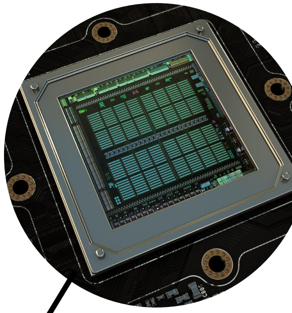
Network for Data Exchange

Programmed using MPI

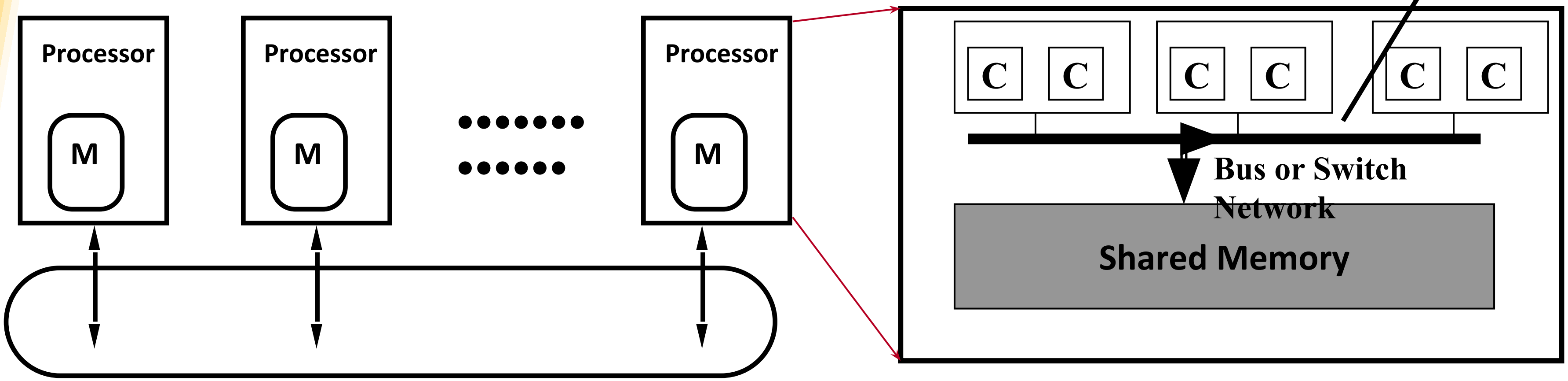
Programmed using on-node API such as OpenMP

HPC Cluster Architecture

- Distributed Memory Machine (HPC cluster)
- Shared Memory Node (SMP)
- Accelerators (GPUs)



SMP Multicore, attached accelerator



Network for Data Exchange

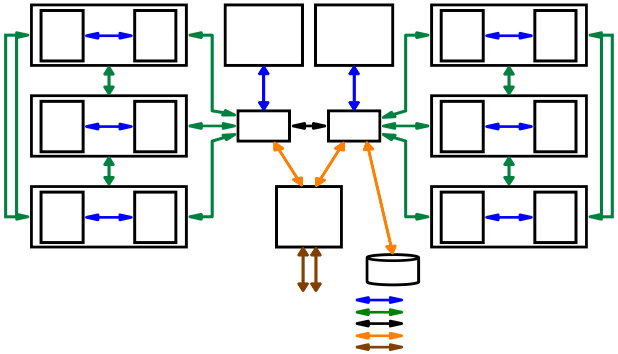
Programmed using MPI

Programmed using on-node API such as OpenMP

Summit 😊

Summit: World’s Fastest Supercomputer

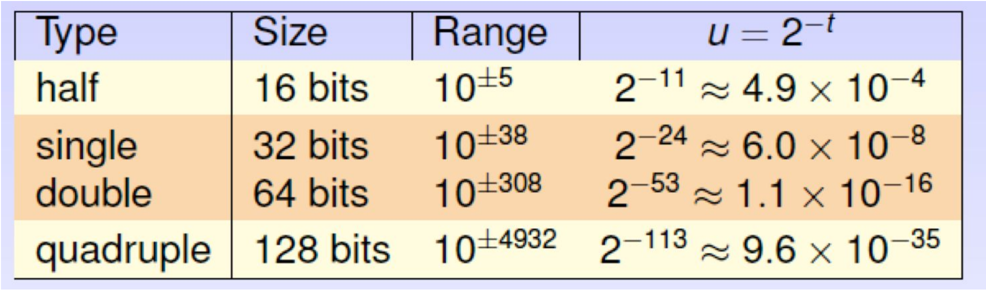
2 Power9s, 6 GPUs per node



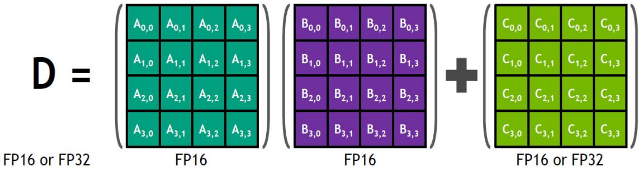
27,648 NVIDIA Tesla V100s, each with:

- 5120 CUDA cores
- 640 Tensor cores
- 300 GB/s BW (NVLink 2.0)
- 20MB registers, 16MB cache, 16GB HBM2 @900 GB/s
- 7.5 DP TFLOPS; 15 SP TFLOPS, 120 FP16 TFLOPS

Tensor cores do mixed precision multiply add of 4x4 matrices



Type	Size	Range	$u = 2^{-t}$
half	16 bits	$10^{\pm 5}$	$2^{-11} \approx 4.9 \times 10^{-4}$
single	32 bits	$10^{\pm 38}$	$2^{-24} \approx 6.0 \times 10^{-8}$
double	64 bits	$10^{\pm 308}$	$2^{-53} \approx 1.1 \times 10^{-16}$
quadruple	128 bits	$10^{\pm 4932}$	$2^{-113} \approx 9.6 \times 10^{-35}$


D =
$A_{0,0}$ $A_{0,1}$ $A_{0,2}$ $A_{0,3}$
$A_{1,0}$ $A_{1,1}$ $A_{1,2}$ $A_{1,3}$
$A_{2,0}$ $A_{2,1}$ $A_{2,2}$ $A_{2,3}$
$A_{3,0}$ $A_{3,1}$ $A_{3,2}$ $A_{3,3}$
$B_{0,0}$ $B_{0,1}$ $B_{0,2}$ $B_{0,3}$
$B_{1,0}$ $B_{1,1}$ $B_{1,2}$ $B_{1,3}$
$B_{2,0}$ $B_{2,1}$ $B_{2,2}$ $B_{2,3}$
$B_{3,0}$ $B_{3,1}$ $B_{3,2}$ $B_{3,3}$
$C_{0,0}$ $C_{0,1}$ $C_{0,2}$ $C_{0,3}$
$C_{1,0}$ $C_{1,1}$ $C_{1,2}$ $C_{1,3}$
$C_{2,0}$ $C_{2,1}$ $C_{2,2}$ $C_{2,3}$
$C_{3,0}$ $C_{3,1}$ $C_{3,2}$ $C_{3,3}$
FP16 or FP32
FP16
FP16
FP16 or FP32

D = AB + C

The Modeling & Simulation community can benefit from utilizing mixed / reduced precision

- Eg: Possible to achieve 4x FP64 peak for 64bit LU on V100 with iterative mixed precision (Dongarra et al.)

Evolution of DOE Leadership Class Systems

Accelerated node 🟦

Name	Titan	Mira	Cori	Theta	Summit	Sierra	Perlmutter
System peak (PF)	27	10	Haswell: 2.81 KNL: 29.5	11.69	200	125	
Peak Power (MW)	9	4.8	4.2	1.7	13.3		6
Total system memory	710TB	768 TB	Haswell: 298.5 TB DDR4 KNL: 1.06 PB DDR4 + High Bandwidth Memory	1475 TB: 843 DDR4 + 70 MCDRAM + 562 SSD	2.8 PB: DDR4, HBM2, PB persistent, memory	1.4 PB DDR4, HBM2, PB persistent, memory	
Node performance (TF)	1.452	0.204	Haswell: 1.178 KNL: 3.046	2.66	>40		
Node Processors	AMD Opteron NVIDIA K20x	64-bit PowerPC A2	Intel Haswell Intel KNL	Intel KNL	2 POWER9 6 NVIDIA Volta GPUs	2 POWER9 4 NVIDIA Volta GPUs	AMD EPYC (Milan) NVIDIA GPU
System Size (nodes)	18,688 nodes	49,152	Haswell; 2,388 nodes KNL: 9,688 nodes	4,392 nodes	~4600 nodes	4320	> 4000 node CPU-only partition
System Interconnect	Gemini	5D Torus	Aries	Aries	Dual Rail EDR-IB	Dual Rail EDR-IB	Cray Slingshot
File System	32 OB 1 TB/s Lustre	26 PB 300 GB/s GPFS	28 PB >700 GB/s Lustre	10 PB 744 GB/s Lustre	120 PB 1 TB/s GPFS		30 PB 4 TB/s Lustre

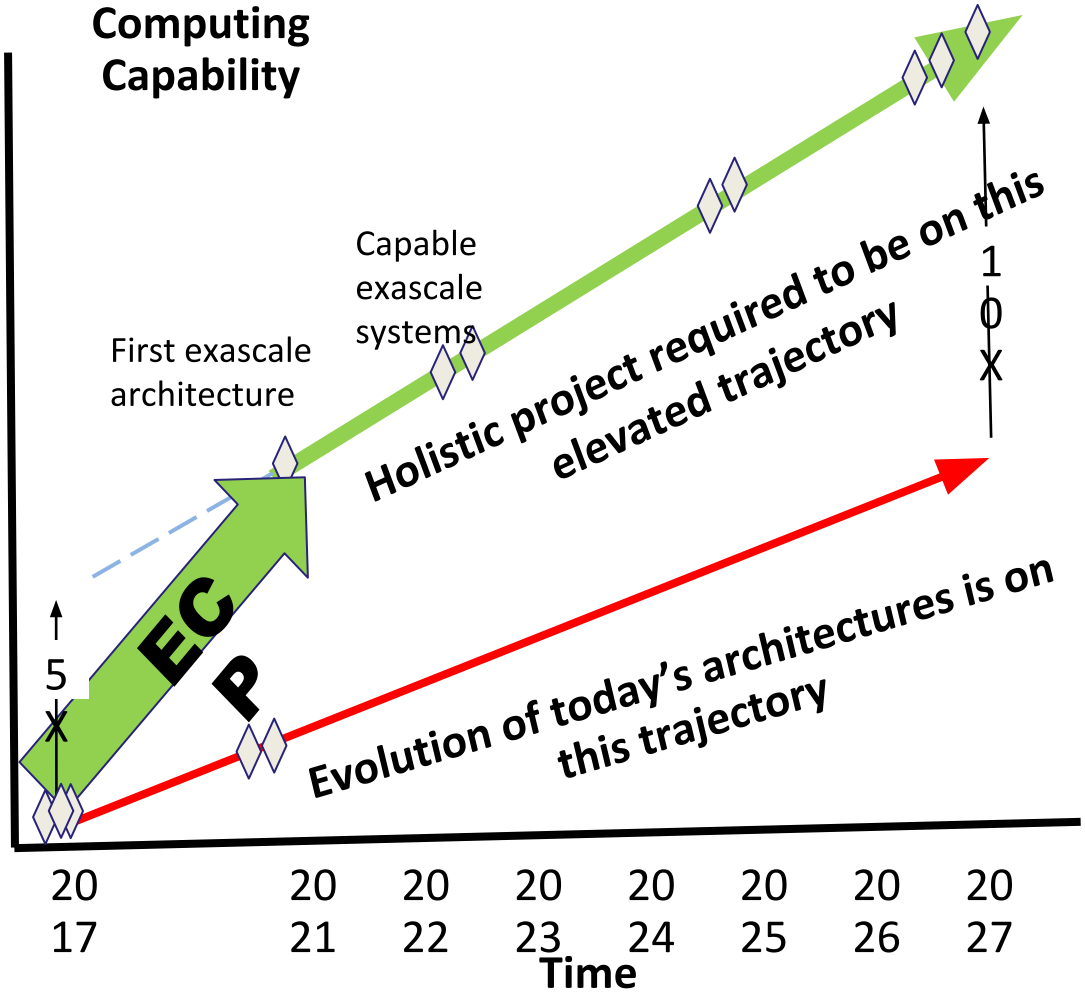
What About Productivity? Portability?

```
// Run one OpenMP thread per device per MPI node
#pragma omp parallel num_threads(devCount) if (initDevice())
 {
      // Block and grid dimensions
      dim3 dimBlock(12,12);
      kernel<<<1,dimBlock>>>();
      cudaThreadExit();
}
else
{
      printf("Device error on %s\n",processor_name);
}
      MPI_Finalize();
      return 0;
}
```

Exascale Computing Project (ECP)

- Major US DOE* project
- Deliver 2 **capable exascale** computing systems
- Exaflop/s rate is 10**18 floating point operations per second
  - ca. 5 X SUMMIT
- Acceptable power
- Develop applications that utilize them
- Develop software to make them usable
- Big part of Exascale Computing Initiative

** Department of Energy*

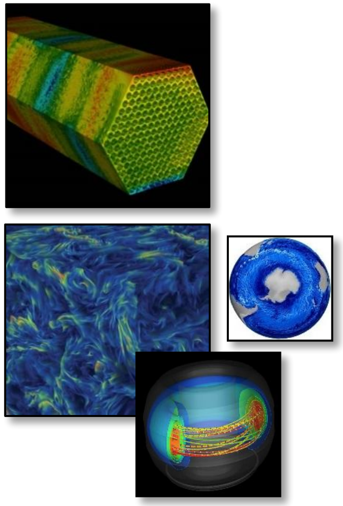


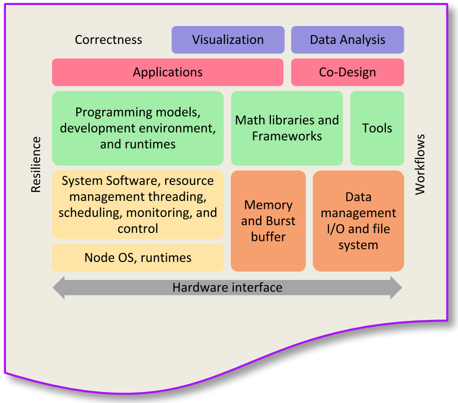
Computing Capability
Capable exascale systems
First exascale architecture
Holistic project required to be on this elevated trajectory
1 0 X
5 X
ECP
Evolution of today's architectures is on this trajectory
2017
2021
2022
2023
2024
2025
2026
2027
Time

"Accelerating innovation with exascale simulation and data science solutions that enhance US economic competitiveness, improve our quality of life, and strengthen our national security."

ECP: From Application Design to Integrated Hardware

Application Development	Software Technology	Hardware Technology	Exascale Systems
Science and mission applications	Scalable and productive software stack	Hardware technology elements	Integrated exascale supercomputers





Correctness

Visualization

Data Analysis

Applications

Co-Design

Resilience

Programming models, development environment, and runtimes

Math libraries and Frameworks

Tools

Workflows

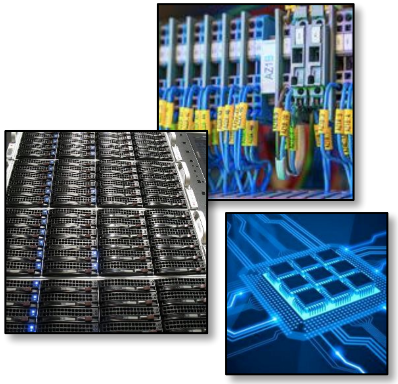
System Software, resource management threading, scheduling, monitoring, and control

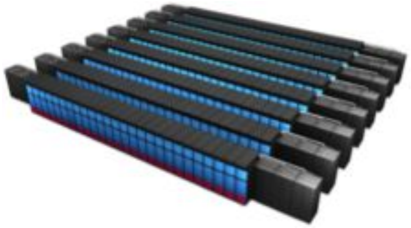
Memory and Burst buffer

Data management I/O and file system

Node OS, runtimes

Hardware interface



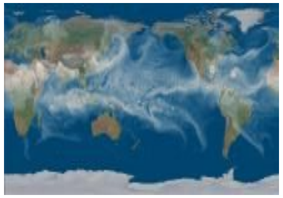


ECP's work encompasses applications, system software, hardware technologies and architectures, and workforce development

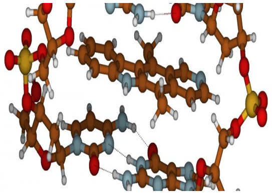
Diverse set of applications including codes from: Chemistry and Materials, Energy, Earth and Space Science, National Security, Data Analytics; Nuclear Stockpile Stew.

ECP Applications

- Deliver improved and impactful science and engineering
- High performance on problem of national importance
- Ensure applications makes effective use of exascale system
    - Needs to execute on potentially diverse set of architectures
    - Strong or weak scaling, ensemble computing
    - Based on latest software technologies
    - How to program?



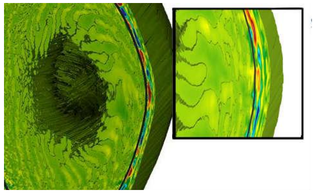
ACME



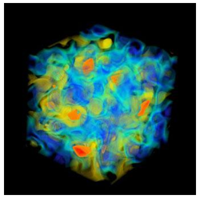
QMCPACK

SLATE

CEES

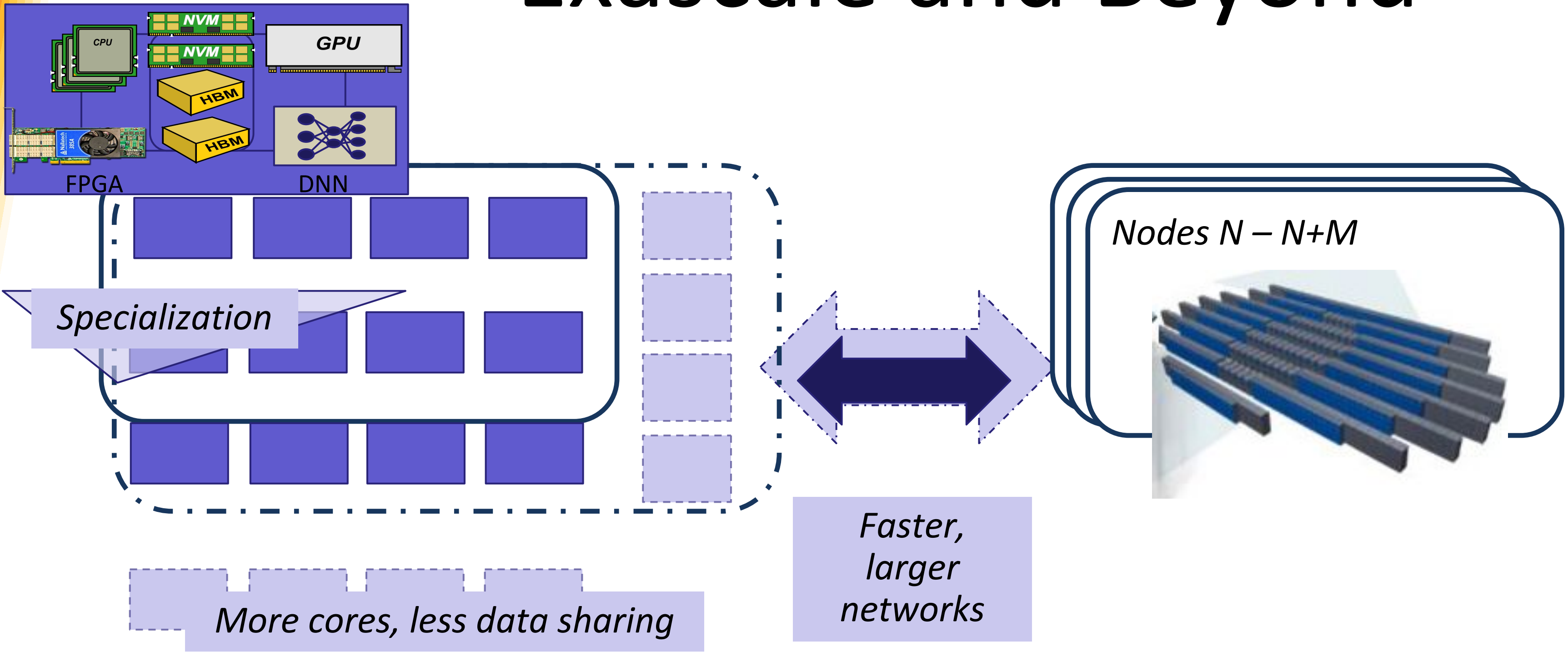


Fusion XGC



LQCD

Exascale and Beyond



- Outside of quantum, neuromorphic, architectures expected to evolve into "extreme" versions of today's systems
  - 3D stacked processors, less cache, more on-die memory, more specialization, optical interconnects
- We need programming languages that meet tomorrow's needs as well as today's application goals
  - **Address needs of systems with diverse, extremely complex memory hierarchies**
  - **Able to handle more (and more kinds of) devices and high core counts**
  - **Facilitate interoperability, especially with internode approaches**

Programming Model Ingredients

- Performance
  - Parallelism, lots of it; load balance; minimize waits
- Power-saving
  - Avoid too much data motion
- Portability
  - Across different vendor platforms
- Performance portability
  - Across diverse heterogeneous systems

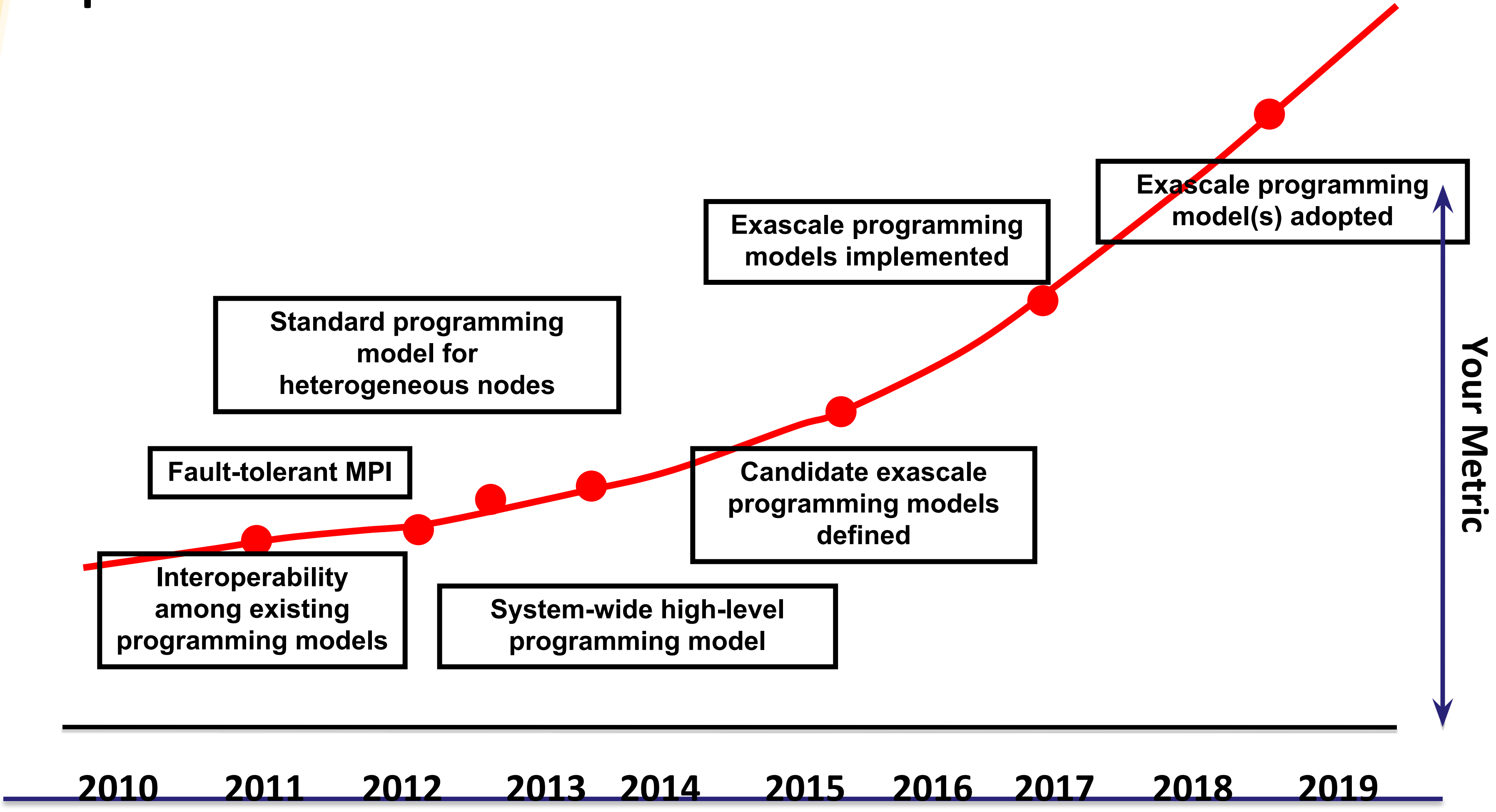
Practical: requires robust implementations and an ecosystem

A pot full of "P"s – we also strive for Productivity

IESP Programming Models

International Exascale Software Project, 2010-11

Proposed timeline



Interoperability among existing programming models

Fault-tolerant MPI

Standard programming model for heterogeneous nodes

System-wide high-level programming model

Candidate exascale programming models defined

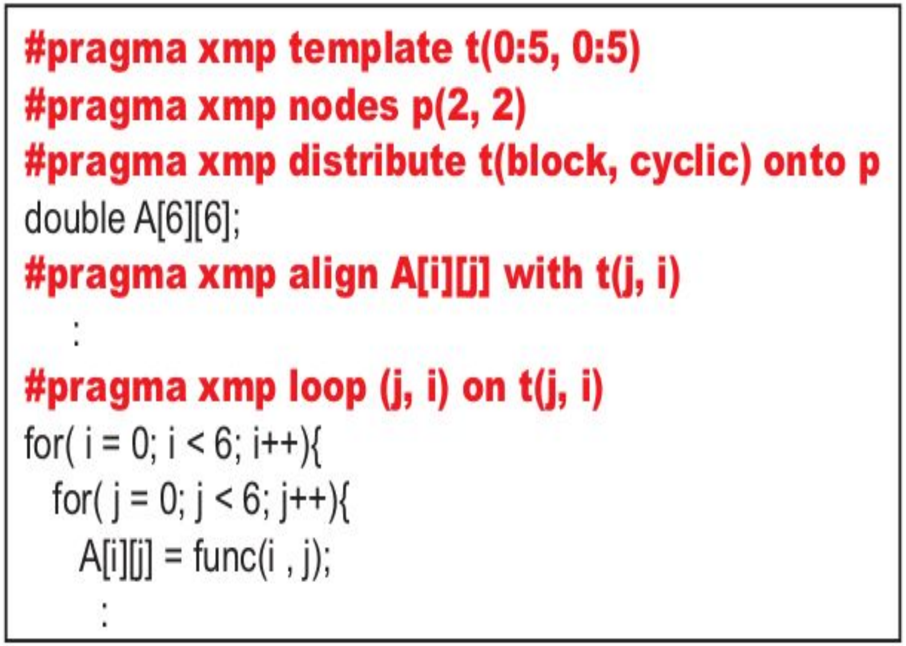
Exascale programming models implemented

Exascale programming model(s) adopted

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

XcalableMP

- Exascale pragmas from Tsukuba/RIKEN
- Directives:
  - Specify data mapping and alignments (HPF)
  - Coordinate distributed computation
- Asynchronous tasks

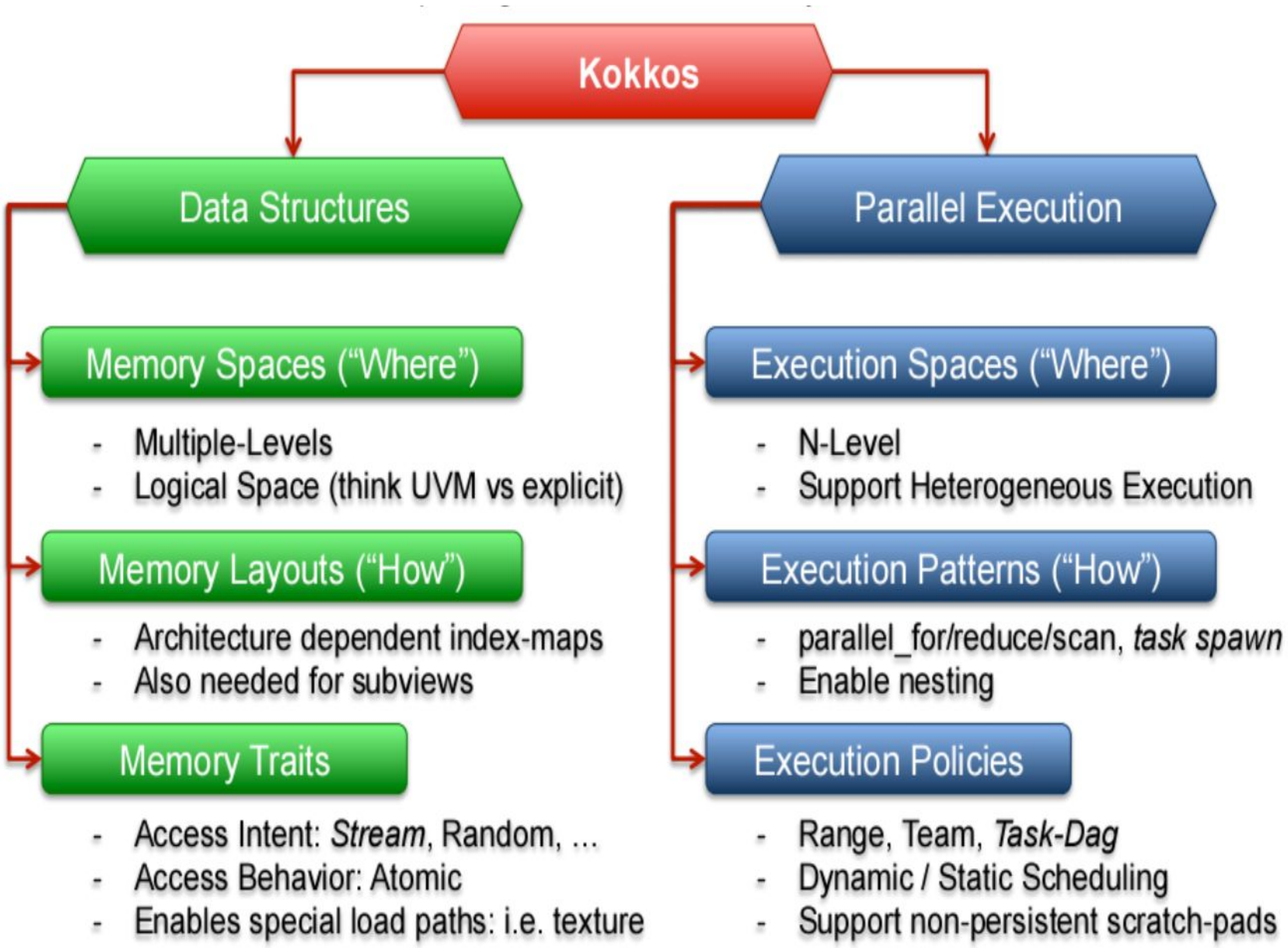


```
#pragma xmp template t(0:5, 0:5)
#pragma xmp nodes p(2, 2)
#pragma xmp distribute t(block, cyclic) onto p
double A[6][6];
#pragma xmp align A[i][j] with t(j, i)
    :
#pragma xmp loop (j, i) on t(j, i)
for( i = 0; i < 6; i++){
  for( j = 0; j < 6; j++){
    A[i][j] = func(i , j);
      :
```

Example from Nakao et al. “Productivity and Performance of Global-View Programming with XcalableMP PGAS Language”, 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Kokkos

- Programming on the node
- Performance portability for C++ codes through abstraction
- Multidim array layouts chosen at compile time
- Tools support low-level management
- Targets include CUDA, pthreads, OpenMP



Kokkos

Data Structures

Memory Spaces ("Where")

- Multiple-Levels
- Logical Space (think UVM vs explicit)

Memory Layouts ("How")

- Architecture dependent index-maps
- Also needed for subviews

Memory Traits

- Access Intent: *Stream*, Random, …
- Access Behavior: Atomic
- Enables special load paths: i.e. texture

Parallel Execution

Execution Spaces ("Where")

- N-Level
- Support Heterogeneous Execution

Execution Patterns ("How")

- parallel_for/reduce/scan, *task spawn*
- Enable nesting

Execution Policies

- Range, Team, *Task-Dag*
- Dynamic / Static Scheduling
- Support non-persistent scratch-pads

Legion

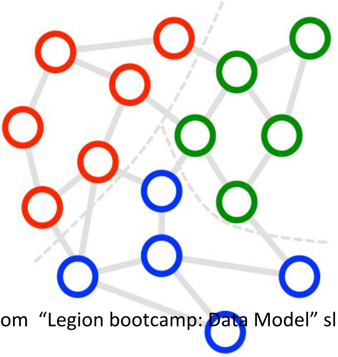
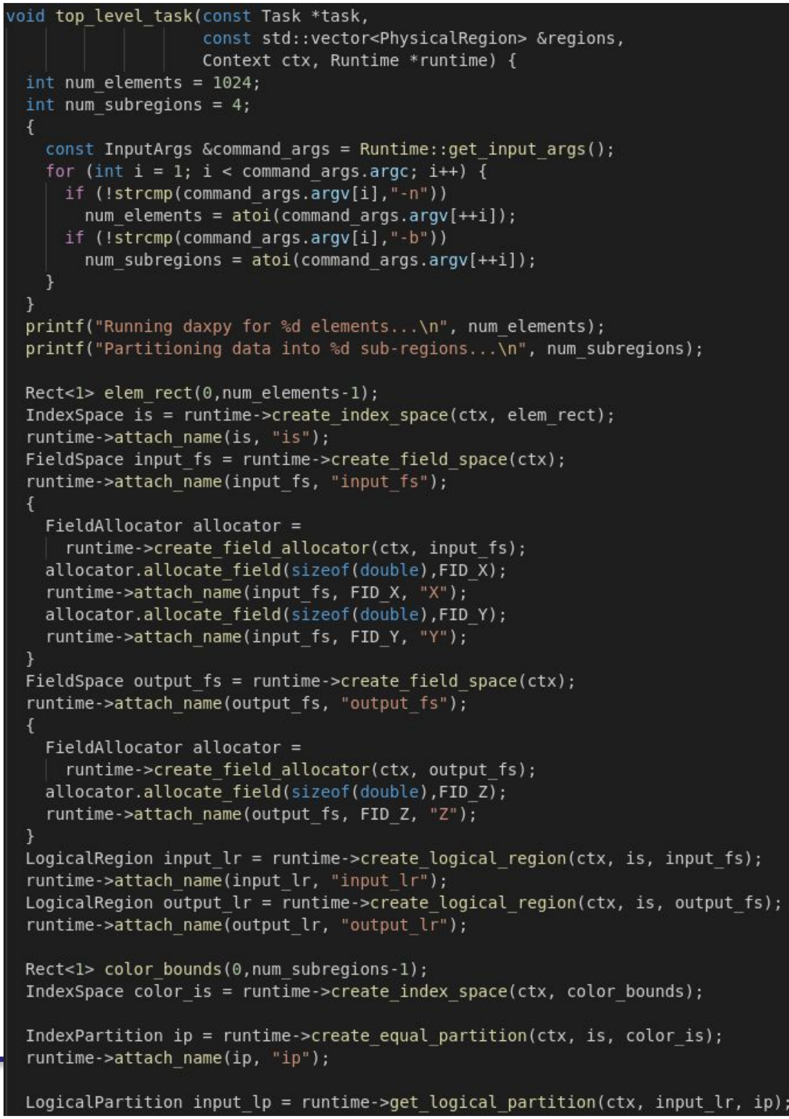


Image from “Legion bootcamp: Data Model” slides by Sean Treichler

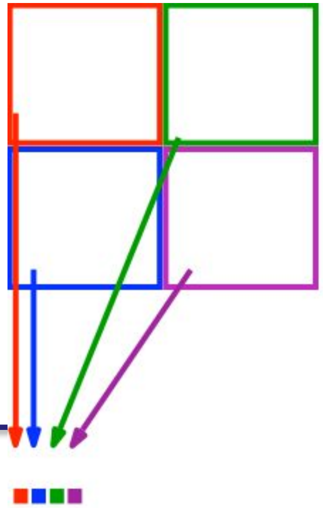
- Development for heterogeneous, distributed systems
- Data is divided into regions
  - can overlap, be recursively defined
  - read/write permissions are set
- Regions are connected by a function/kernel, which specifies the computation
- Runtime divides work in functions on the regions into tasks to be executed
- Legion constructs embedded in C++



```
void top_level_task(const Task *task,
                    const std::vector<PhysicalRegion> &regions,
                    Context ctx, Runtime *runtime) {
  int num_elements = 1024;
  int num_subregions = 4;
  {
    const InputArgs &command_args = Runtime::get_input_args();
    for (int i = 1; i < command_args.argc; i++) {
      if (!strcmp(command_args.argv[i],"-n"))
        num_elements = atoi(command_args.argv[++i]);
      if (!strcmp(command_args.argv[i],"-b"))
        num_subregions = atoi(command_args.argv[++i]);
    }
  }
  printf("Running daxpy for %d elements...\n", num_elements);
  printf("Partitioning data into %d sub-regions...\n", num_subregions);

  Rect<1> elem_rect(0,num_elements-1);
  IndexSpace is = runtime->create_index_space(ctx, elem_rect);
  runtime->attach_name(is, "is");
  FieldSpace input_fs = runtime->create_field_space(ctx);
  runtime->attach_name(input_fs, "input_fs");
  {
    FieldAllocator allocator =
      runtime->create_field_allocator(ctx, input_fs);
    allocator.allocate_field(sizeof(double),FID_X);
    runtime->attach_name(input_fs, FID_X, "X");
    allocator.allocate_field(sizeof(double),FID_Y);
    runtime->attach_name(input_fs, FID_Y, "Y");
  }
  FieldSpace output_fs = runtime->create_field_space(ctx);
  runtime->attach_name(output_fs, "output_fs");
  {
    FieldAllocator allocator =
      runtime->create_field_allocator(ctx, output_fs);
    allocator.allocate_field(sizeof(double),FID_Z);
    runtime->attach_name(output_fs, FID_Z, "Z");
  }
  LogicalRegion input_lr = runtime->create_logical_region(ctx, is, input_fs);
  runtime->attach_name(input_lr, "input_lr");
  LogicalRegion output_lr = runtime->create_logical_region(ctx, is, output_fs);
  runtime->attach_name(output_lr, "output_lr");

  Rect<1> color_bounds(0,num_subregions-1);
  IndexSpace color_is = runtime->create_index_space(ctx, color_bounds);

  IndexPartition ip = runtime->create_equal_partition(ctx, is, color_is);
  runtime->attach_name(ip, "ip");

  LogicalPartition input_lp = runtime->get_logical_partition(ctx, input_lr, ip);
```

GRID Programming Framework

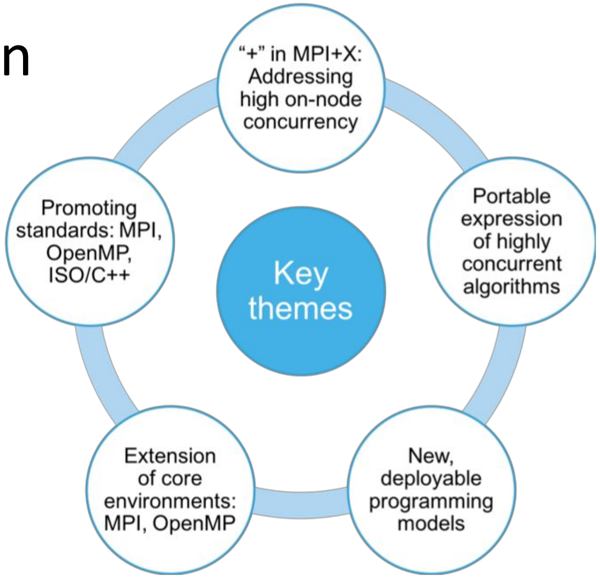
- High level, data parallel programming model
- Facilitates use of multiple types of parallelism
  - Currently supports MPI, OpenMP and short vector parallelism (SSE, AVX, AVX2, FMA4, IMCI and AVX512)
- Under development by Peter Boyle (EPCC) since summer 2014
  - Planned use by RBC, UKQCD on Cori, Theta, BlueWaters and future UK systems (DiRAC-3).
  - Development part-funded by Intel
- Data layout, mappings, memory optimizations
  - “Borrows” from ideas in CM-Fortran and HPF
  - Avoids re-engineering of arrays and structures
  - Automatic generation of target-specific code



http://arxiv.org/abs/1512.03487

ECP Programming Models

- Exploit system with acceptable effort
  - Address challenges posed by new hardware
    - Multiple levels of parallelism, including vectorization
  - Avoid reprogramming for different hardware
- Programming features and implementation
- Multiple efforts under way
  - Including Kokkos, Legion, Grid++
  - Custom frameworks
  - MPI "+ X"
- Majority of ECP applications use MPI and OpenMP
  - MPI and OpenMP are evolving too


"+" in MPI+X: Addressing high on-node concurrency
Portable expression of highly concurrent algorithms
New, deployable programming models
Extension of core environments: MPI, OpenMP
Promoting standards: MPI, OpenMP, ISO/C++
Key themes

OpenMP®

Portable parallel programming since 1997

- Compiler directives
- Data, task, SIMD parallelism
- Multicores, GPUs
- User specifies the strategy, not the details

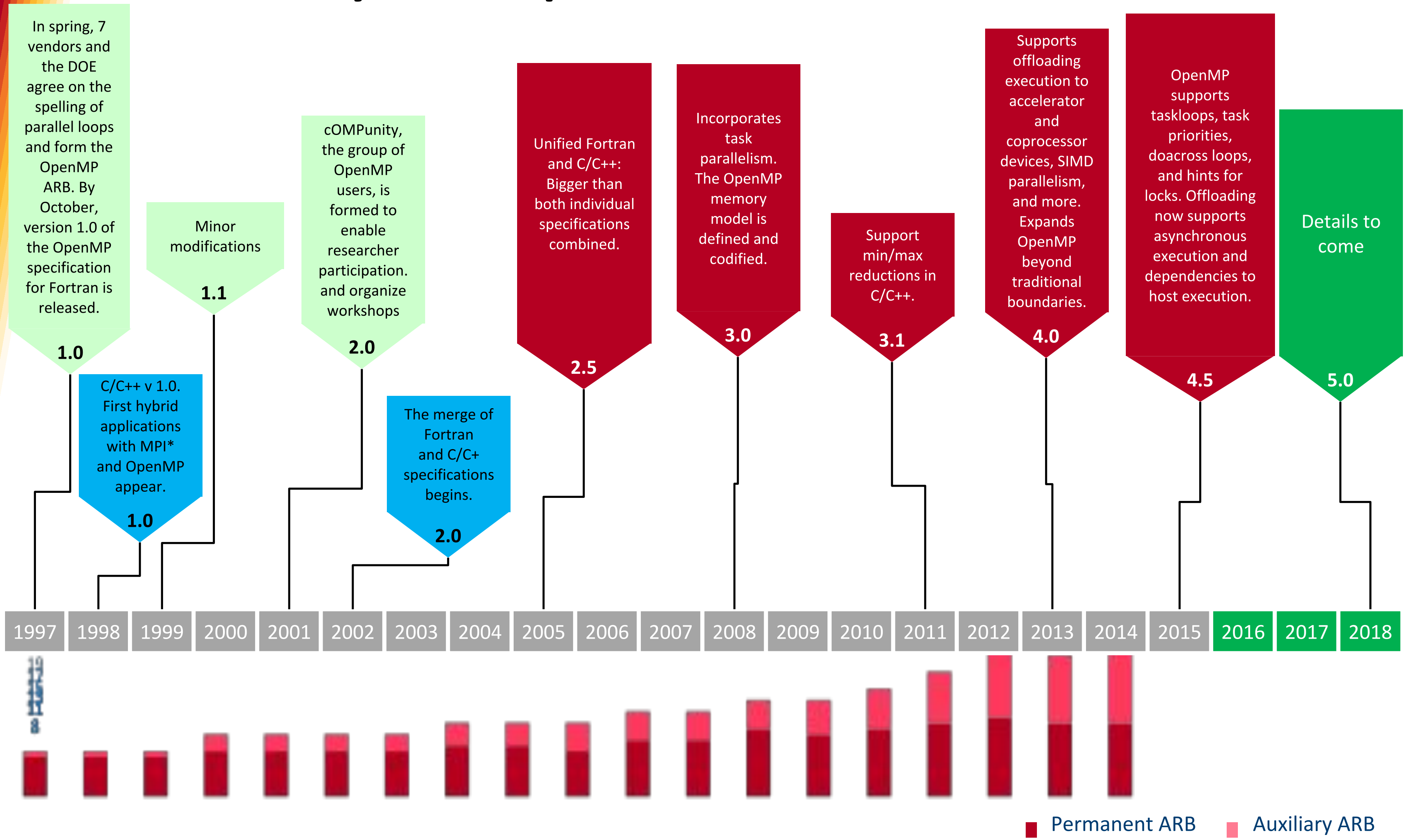
Maintained by industry consortium

- It is now easy for academics to join

AMD
Argonne NATIONAL LABORATORY
arm
BSC
BROOKHAVEN NATIONAL LABORATORY
OMPunity
CRAY THE SUPERCOMPUTER COMPANY
epcc
FUJITSU
IBM
Inria
intel
Los Alamos NATIONAL LABORATORY
MARVELL
MAUI HIGH PERFORMANCE COMPUTING CENTER DSRC
Micron
NASA
NEC
NVIDIA
ORACLE
redhat
RWTH AACHEN UNIVERSITY
SUSE We adapt. You succeed.
TACC
TI
UNIVERSITY OF DELAWARE
UH
MANCHESTER 1824 The University of Manchester

The mission of the OpenMP ARB (Architecture Review Board) is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable.

History of OpenMP: 1997 - 2018



1.0 — In spring, 7 vendors and the DOE agree on the spelling of parallel loops and form the OpenMP ARB. By October, version 1.0 of the OpenMP specification for Fortran is released.

1.0 — C/C++ v 1.0. First hybrid applications with MPI* and OpenMP appear.

1.1 — Minor modifications

2.0 — cOMPunity, the group of OpenMP users, is formed to enable researcher participation. and organize workshops

2.0 — The merge of Fortran and C/C+ specifications begins.

2.5 — Unified Fortran and C/C++: Bigger than both individual specifications combined.

3.0 — Incorporates task parallelism. The OpenMP memory model is defined and codified.

3.1 — Support min/max reductions in C/C++.

4.0 — Supports offloading execution to accelerator and coprocessor devices, SIMD parallelism, and more. Expands OpenMP beyond traditional boundaries.

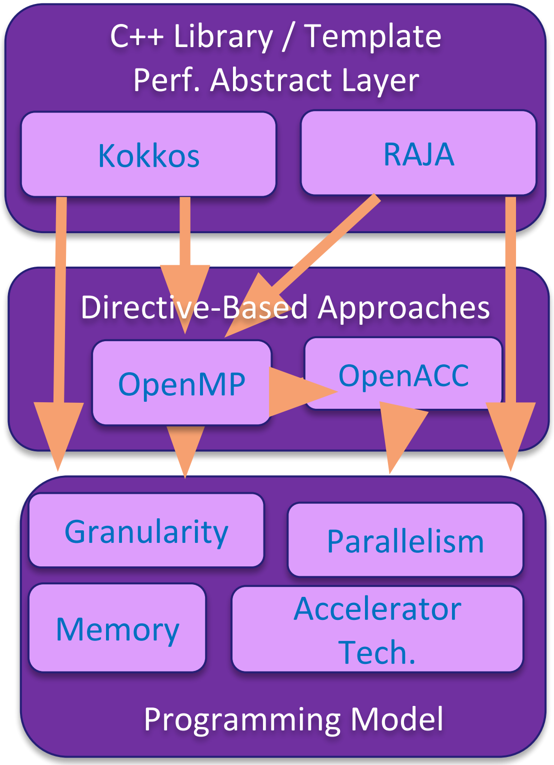
4.5 — OpenMP supports taskloops, task priorities, doacross loops, and hints for locks. Offloading now supports asynchronous execution and dependencies to host execution.

5.0 — Details to come

Intra-node: Complementary Technologies

- ECP’s OpenMP project goal is to deliver enhanced OpenMP to address programming needs of entire node, including heterogeneity, complex memory hierarchies, and SIMD parallelism
- Related technologies include Kokkos, RAJA, OpenACC, CUDA

	CUDA	Kokkos	OpenACC	OpenMP	RAJA
Languages	C/C++	C/C++	C/C++/Fortran	C/C++/Fortran	C/C++
Prog. Style		Template Meta-programming, C++11 lambdas	Directive-based	Directive-based	C++11 lambdas
Parallelism	SIMT	OpenMP, Pthreads, CUDA	SIMD, CUDA, Fork-Join	SPMD, SIMD, Tasks, CUDA, Fork-Join	OpenMP, CUDA
Licensing/ Accessibility	Propriet-ary	Open-source	Proprietary	Open-source	Open-source
Abstraction Level	Low	High	High/Medium	High/Medium	High



Service Layers

OpenMP 4.5 – Accelerator Model

- OpenMP 4+ supports heterogeneous systems (accelerators/devices)
- Accelerator model
  - Host device and attached devices

With attached accelerator(s)

Host Device
(CPU Multicore)

Single device attached

Xeon Phi(s) –
(Accelerator and self-hosted)

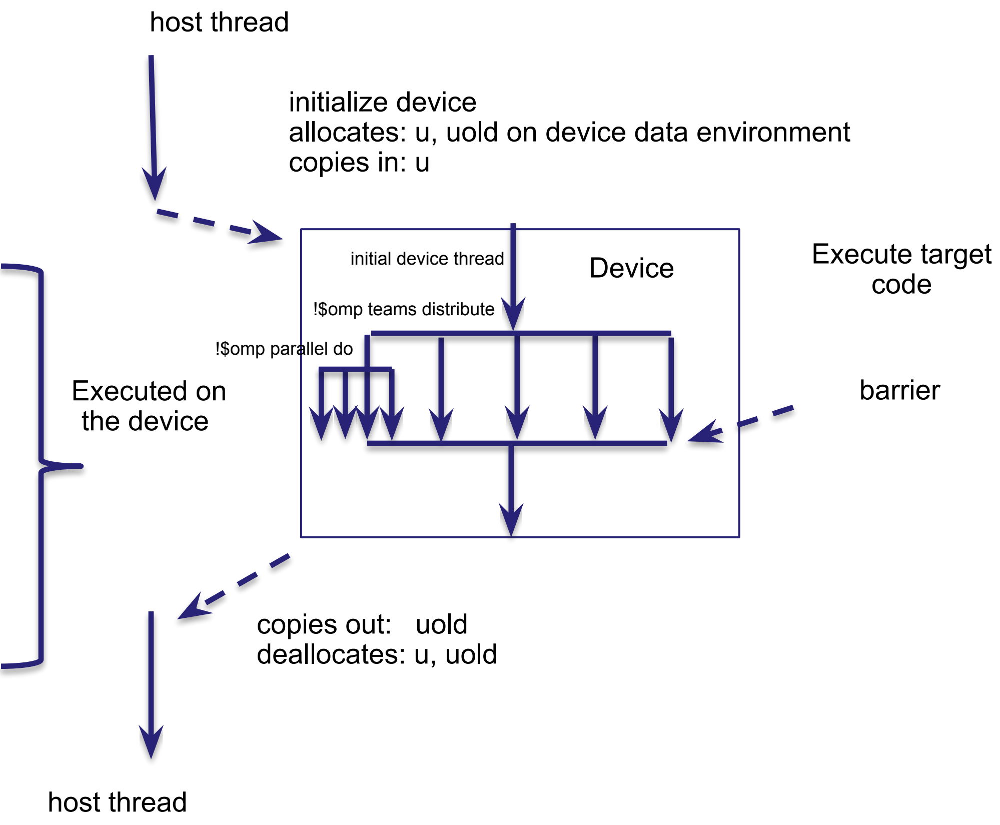
Multiple devices attached

GPU(s)

OpenMP Target Example

An example of OpenMP 4.5 for accelerators

```
..
!$omp target map(to:u) map(from:uold)

!$omp teams distribute parallel do collapse(2)
        do j=1,m
               do i=1,n
                 uold(i,j) = u(i,j)
           enddo
        enddo

!$omp end target
..
```



host thread

initialize device
allocates: u, uold on device data environment
copies in: u

Executed on the device

initial device thread

Device

!$omp teams distribute

!$omp parallel do

Execute target code

barrier

copies out: uold
deallocates: u, uold

host thread

Use target construct to:

- Transfer control from the host to the target device
- Map variables to/from the device data environment

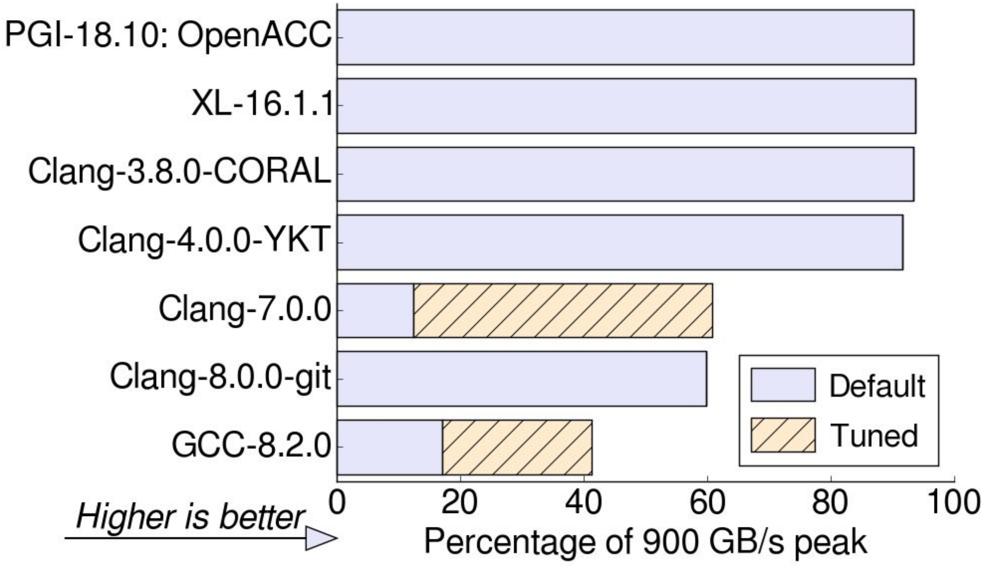
Host thread waits until target region completes

- Use nowait for asynchronous execution

Several compilers achieve close to the GPU memory bandwidth peak in the STREAM Triad kernel

Recommended combination of OpenMP directives:

```
#pragma omp target teams distribute parallel for
for (j=0; j<N; j++)
  a[j] = b[j] + scalar*c[j];
```



Sometimes additional tuning is needed:

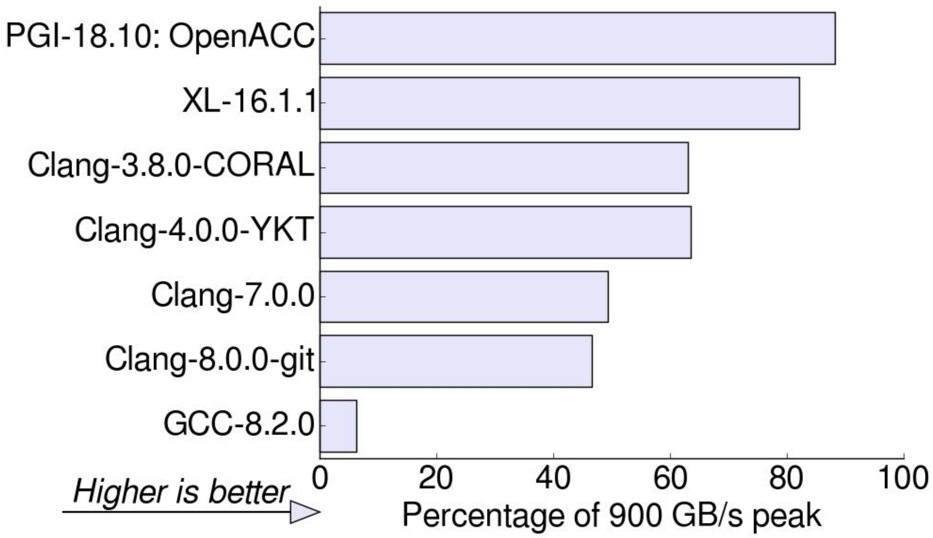
- Clang-7.0.0: requires "schedule(static,1)"
- GCC-8.2.0: must add the "simd" construct

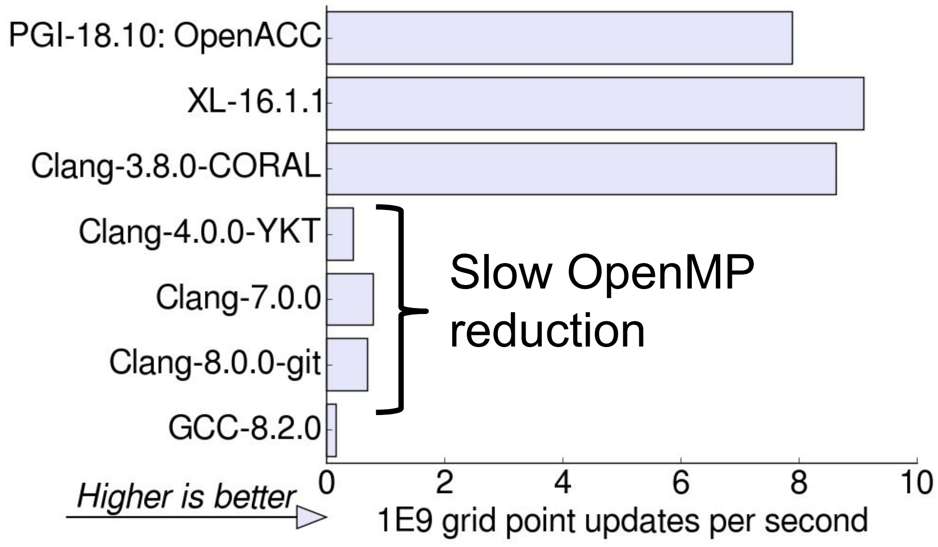
Courtesy Chris Daley, NERSC

Proprietary compilers tested consistently performed well in other micro-benchmarks

Transpose benchmark: tiled loops and use of the OpenMP “collapse” clause

Laplace equation benchmark: frequent kernel launches and data reductions





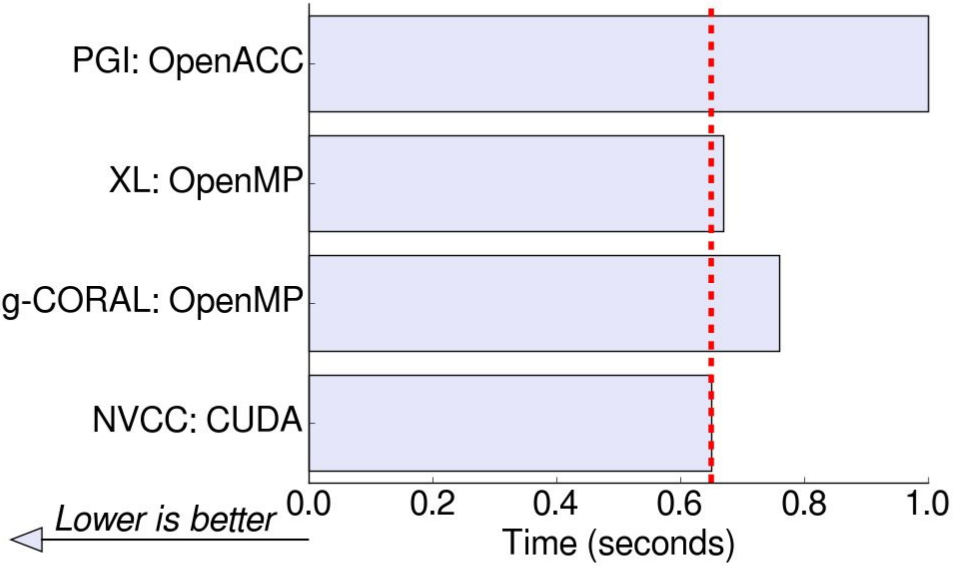
Slow OpenMP reduction

Chris Daley, NERSC

Case study: BerkeleyGW mini-application named GPP

- BerkeleyGW is a C++ application which computes the excited state properties of materials
- GPP contains the self-energy computation: large matrix reductions over complex arrays in a single loop nest of 4 loops

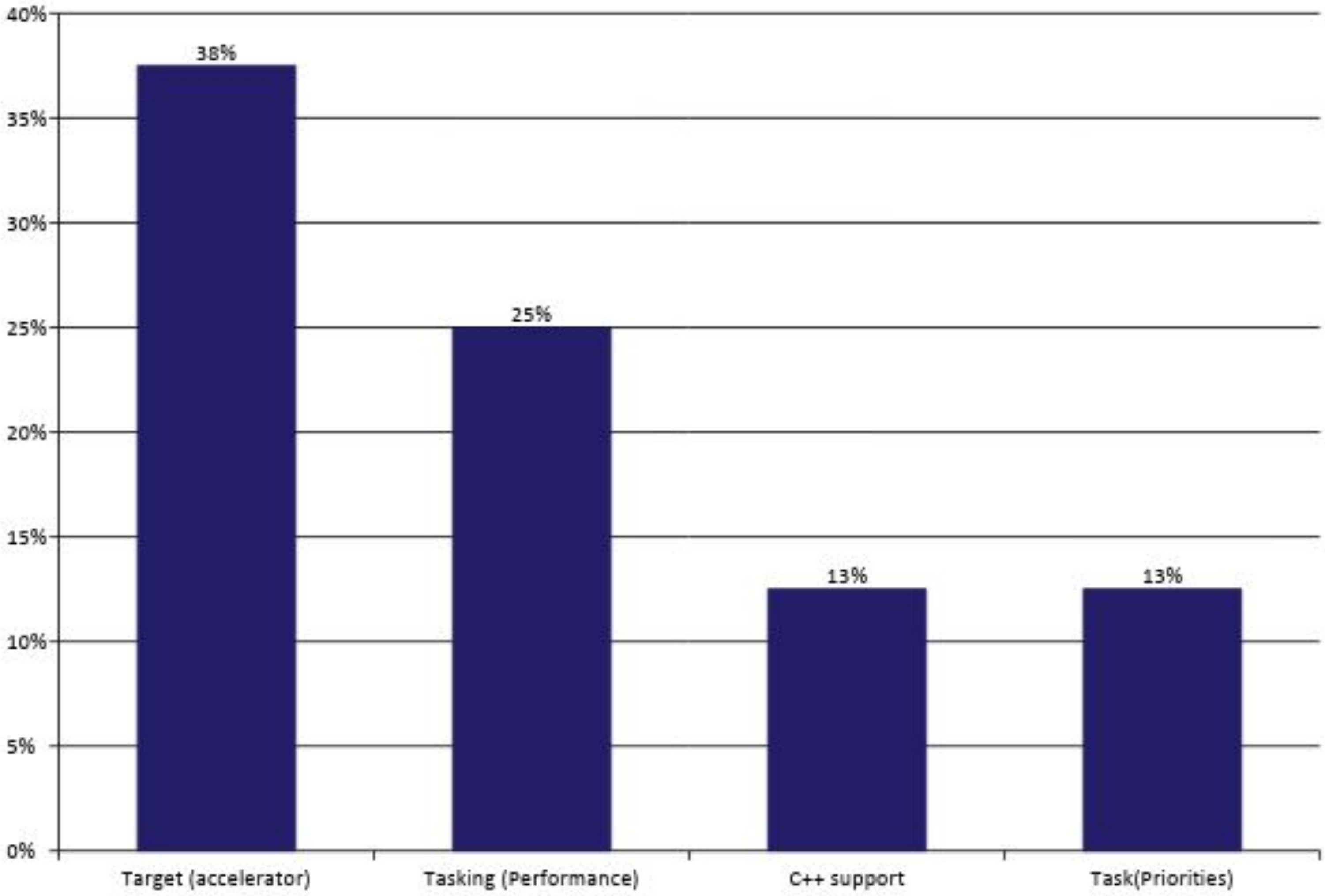
The OpenMP implementation with XL compiler achieves approximately the same run time as a tuned CUDA implementation



Results from: Rahulkumar Gayatri, "A Case Study for Performance Portability Using OpenMP 4.5", WACCPD-18

Chris Daley, NERSC

ECP Survey : OpenMP Challenges



QMCPACK – OpenMP Needs

- Exploring OpenMP for on-node parallelism
- Offload programming to exploit GPUs
- Performance portability is needed
- Also considering other approaches

Key problem areas:

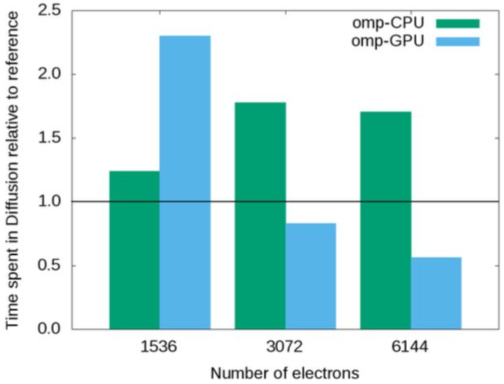
- Performance Portability
  - Between CPU and GPU
  - SIMD performance varies
- OpenMP / Libraries
  - Nested parallelism
  - Affinity control
- Deep copy needed
  - Vector of vectors classes
  - Shadow data handling
- Exposing GPU queues/streams
  - Library interoperability

Co-design interactions:

- #pragma omp loop
- BOLT runtime for nested parallelism
- Meta directive to generate customized directives and transform loops by compilers
- Custom Mappers for Deep copy
- Prototype for exposing streams in OpenMP 5.1

QMCPACK

Preliminary Diffusion Kernel via miniQMC



Summit

NiO with 128, 256, 512 atoms, namely 1536, 3072 and 6144 electrons are studied.

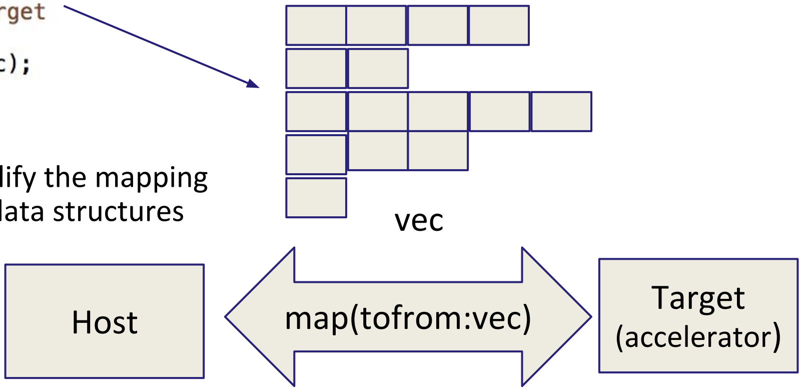
Courtesy Ye Luo, ANL

Deep Copy Application Requirements

- Requires ability to map C++ vectors in target regions
- Uses vector<vector *> QMCdata
  - E.g. Splines[i]->coef[i]
- Deep copy to/from host
  - Uses vector::data() to access data pointers
- Specializes vector class to map data
- A simplified solution is needed

Solution: custom mappers (OpenMP 5.0)

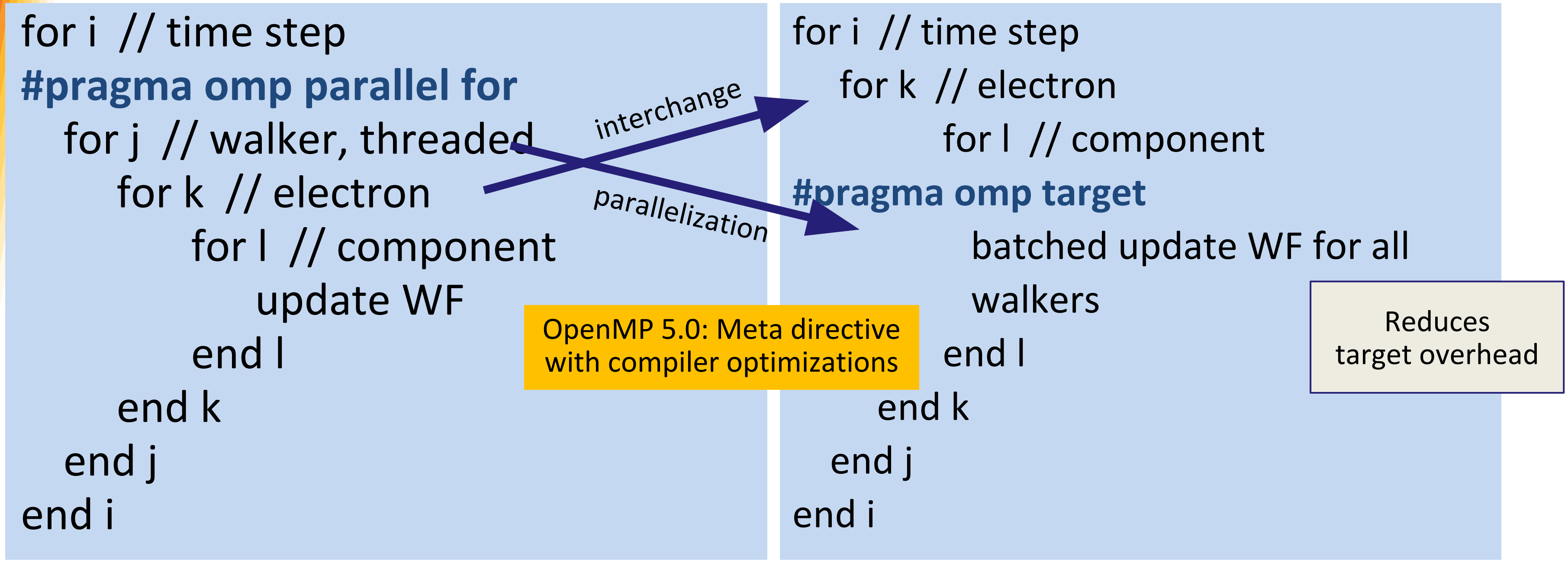
```
template<typename T>
class vector {
    T *_begin = nullptr;
    T *_end = nullptr;
    size_t len = 0;
public:
    vector() = default;
    //…
    #pragma omp declare mapper(vector<T> v) \
            map(v, v._begin[0:len], v._end[-len:len])
};

vector<vector<double>> vec;
populate_vec(vec);
#pragma omp target
{
    use_vec(vec);
}
```



Mappers simplify the mapping of complex data structures

OpenMP Performance Portability Challenges

QMCPACK



CPU strategy

```
for i  // time step
#pragma omp parallel for
    for j  // walker, threaded
        for k  // electron
              for l  // component
                    update WF
              end l
        end k
    end j
end i
```

GPU strategy

```
for i  // time step
    for k  // electron
              for l  // component
#pragma omp target
                 batched update WF for all
                 walkers
              end l
        end k
    end j
end i
```

OpenMP 5.0: Meta directive with compiler optimizations

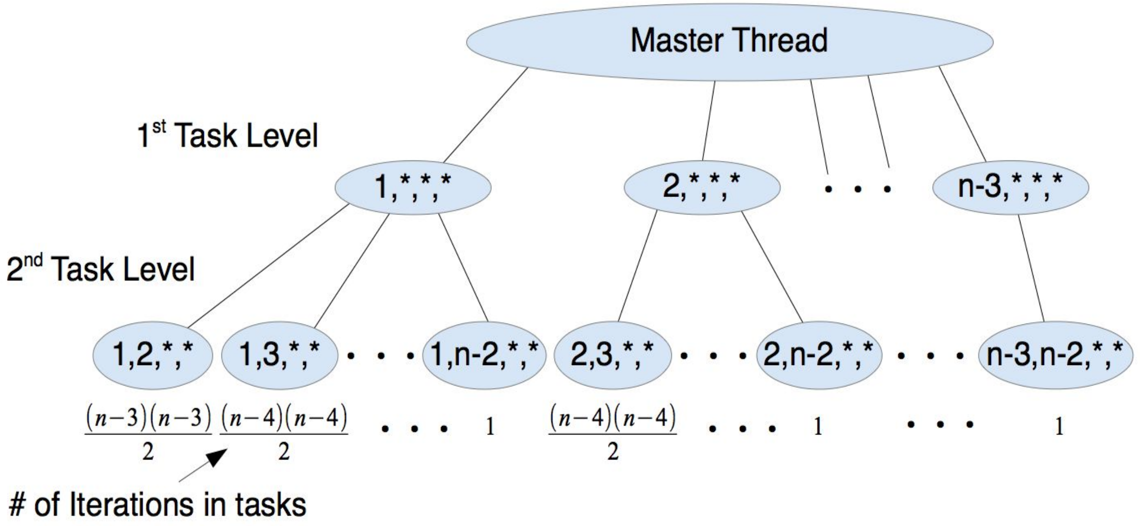
Reduces target overhead

Courtesy Ye Luo, ANL

Tasking in OpenMP

- Task-based programming models suitable for parallelizing many data analytics and graph analysis algorithms
- OpenMP tasks can scale --- but the price of load balancing is the potential loss of data locality

- Large amount of load balancing at high thread counts observed in some implementations
- Needs NUMA-aware scheduling



Aaron B. Adcock, Blair D. Sullivan, Oscar R. Hernandez, and Michael W. Mahoney. 2013. Evaluating OpenMP tasking at scale for the computation of graph hyperbolicity. In OpenMP in the Era of Low Power Devices and Accelerators. Lecture Notes in Computer Science, Vol. 8122. Springer, 71--83.

Using OpenMP Tasks

SLATE

- ScaLAPACK replacement
- dense linear algebra
  - BLAS
  - LU, LLT, QR/LQ
  - SVD, EVP
- distributed memory
- multicore
- multi-GPU

OMP Tasks

- nested tasking
- top: `omp task depend` (DAG scheduling)
  - using task dependencies, not data dependencies
- bottom: `omp task` (independent)
- all activities are tasks
  - CPU work
  - GPU kernel launches
  - GPU communication
  - MPI communication
- heavily using task priorities

OMP Offload

- OMP directives
  - not quite sufficient
  - somewhat host centric
    - SLATE tries to be more symmetric
- OMP target API
  - basically sufficient

streams and asynchronicity

- duality of streams
  - synchronize within each stream
  - desynchronize among streams
- we only need the latter
  - GPU kernel are launched as tasks
  - tasks are scheduled using OMP deps
  - need to enable concurrent execution
    - requires management of CUDA streams
    - ideally GPU actions would be async by default

device memory allocation

- need async device memory allocator
  - `cudaMalloc()` is sync
- rolled out our own
  - simplistic but async

Courtesy Jakub Kurzak, UTK

ECP Application OpenMP Requirements

Application Requirements	Addressed by White Paper	Addressed by OpenMP 5.0	Addressed by Implementations	Addressed by V&V
Quality of implementation for OpenMP 4.5	NA	NA	Extensions, optimi-zations, bug fixes	V&V 4.5 accelerator tests
OpenACC->OpenMP	!$omp loop	!$omp loop	Performance portability optimizations	Combined directives in OpenMP 4.5
Non-contiguous data sections for target data	Restriction lifted	Yes	FY19	FY19
Implicit declare target routines	Implicit declare target	Yes	Inline optimizations	FY19
Support for GPU shared memory	Memory Management API	Yes (revised)	FY19	FY19
Fine-grained synchronization (tasks)	Taskwait depend	Yes	FY19	FY19
Support for pinned memory	Memory Management API	Yes (revised)	FY19	FY19

ECP Application OpenMP Requirements (ctd)

Application Requirements	Addressed by White Paper	Addressed by OpenMP 5.0	Addressed by Implementation	Addressed by V&V
C++ Lambdas / target	No.	No	Through Raja/Kokkos	None.
Data placement across NUMA nodes	Memory Management API	Yes (revised)	FY19	5.0
Releasing runtime memory from GPU	Pause resource	Yes	FY19	5.0
Deep copy support	Declare mappers	Yes (revised)	FY19	5.0
Support for static data members in C++ on device	Yes	Yes	FY19	5.0
Support for task priorities	Already in 4.5	Already in 4.5	Implementation issue	4.5 – FY19
Improved performance for nested parallelism	NA	NA	BOLT runtime	4.5 – FY19
Full support of latest C++ and Fortran standards	Work in progress	Work in progress	FY19-FY21	
Interoperability with libs, other OpenMP code	No	No	Beyond 5.0	Beyond 5.0

OpenMP 5.0

Ratified November 2018; download from www.openmp.org

OpenMP 5.0 New Features

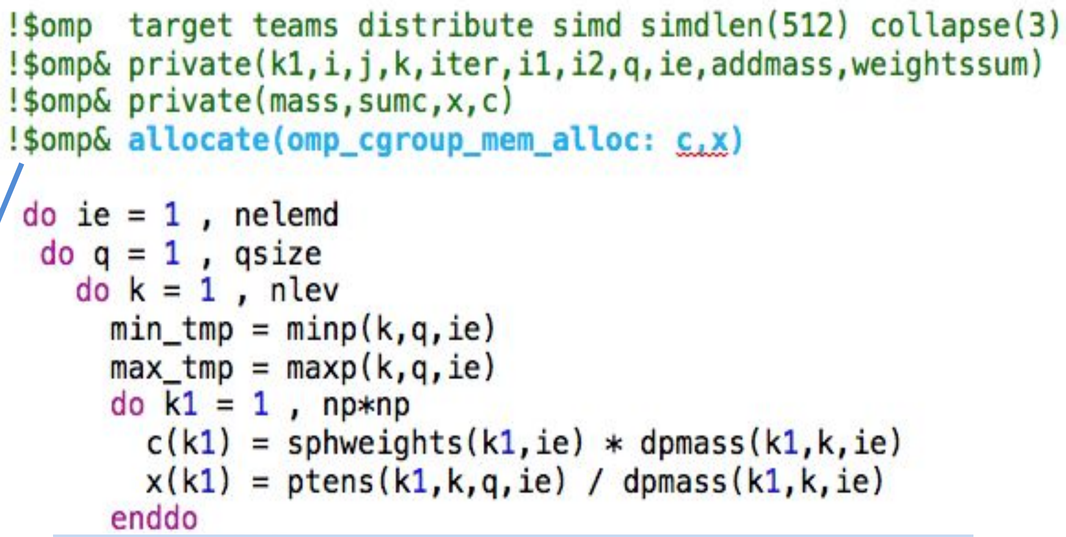
- New added normative base languages
  - Fortran 2008, C11, C++11, C++14, C++17
- Task enhancement
  - Task reduction
  - Data affinity
  - Expand dependency mechanism
- Descriptive parallelism
  - The loop construct
  - Allow the compiler to select the appropriate parallelism form
- For/Do loop enhancement
  - Add scan operations
  - Collapse of non-rectangular loops
- Tool interface
  - Allow 3rd party performance and debugging tools

- Reverse offloading
  - Offload from device to host
- Memory management APIs
  - Support complex memory hierarchy
  - Runtime APIs and allocate directive
- Release/acquire semantics
  - Support more memory models
- User defined mapper
  - Support deep copy within map clauses
- The requires directive
  - Support specialization for devices
  - E.g., unified shared memory
- Function variants
  - Allow to define multiple versions for the same function

Ratified November 2018; download from www.openmp.org

Memory Management API

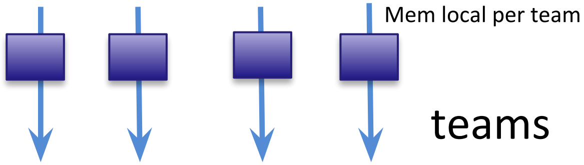
- APIs to access different types of node memories
- Pre-defined allocators for
  - Large capacity mem.
  - Read-only memory
  - High bandwidth mem.
  - Low latency mem.
  - Local memory in the same contention group
    - #pragma omp teams -- per local team
  - Threads in the same parallel team
    - #pragma omp parallel



```
!$omp  target teams distribute simd simdlen(512) collapse(3)
!$omp& private(k1,i,j,k,iter,i1,i2,q,ie,addmass,weightssum)
!$omp& private(mass,sumc,x,c)
!$omp& allocate(omp_cgroup_mem_alloc: c,x)

do ie = 1 , nelemd
 do q = 1 , qsize
   do k = 1 , nlev
     min_tmp = minp(k,q,ie)
     max_tmp = maxp(k,q,ie)
     do k1 = 1 , np*np
       c(k1) = sphweights(k1,ie) * dpmass(k1,k,ie)
       x(k1) = ptens(k1,k,q,ie) / dpmass(k1,k,ie)
     enddo
```

Use of memory on the same contention group (e.g. GPU shared memory)
6x speedup on K20x (Titan) a kernel from ACME.

#pragma omp target teams



omp_cgroup_mem_alloc allocates memory local to a contention group   (#pragma omp teams)

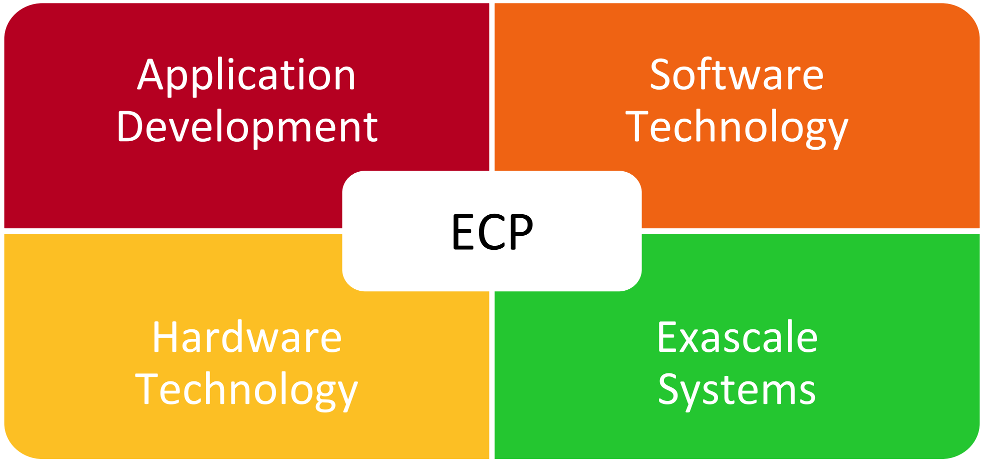
The Path Forward

- OpenMP 5.1 will be released in November 2020
  - Proceedings of the IEEE article: “The Ongoing Evolution of OpenMP”
  - Broadly support on-node performant, portable parallelism
  - OpenMP 5.1 will refine how OpenMP 5.0 realizes it, will not break existing code
- Clarifications, corrections possibly minor extensions
  - Improved native device support (e.g., CUDA streams)
  - May add `taskloop` dependences
  - Address issues arising from detailed implementation and use of OpenMP 5.0
- Plan is to release OpenMP 6.0 in November 2023

What Might OpenMP 6.0 Include?

- Deeper support for descriptive and prescriptive control
- More support for memory affinity and complex hierarchies
- Support for pipelining, other computation/data associations
- Continued improvements to device support
  - Extensions of deep copy support (serialize/deserialize functions)
- Task-only, unshackled or free-agent threads
- Event-driven parallelism
- Completing support for new normative references

Co-design is Key



Application Development

Software Technology

ECP

Hardware Technology

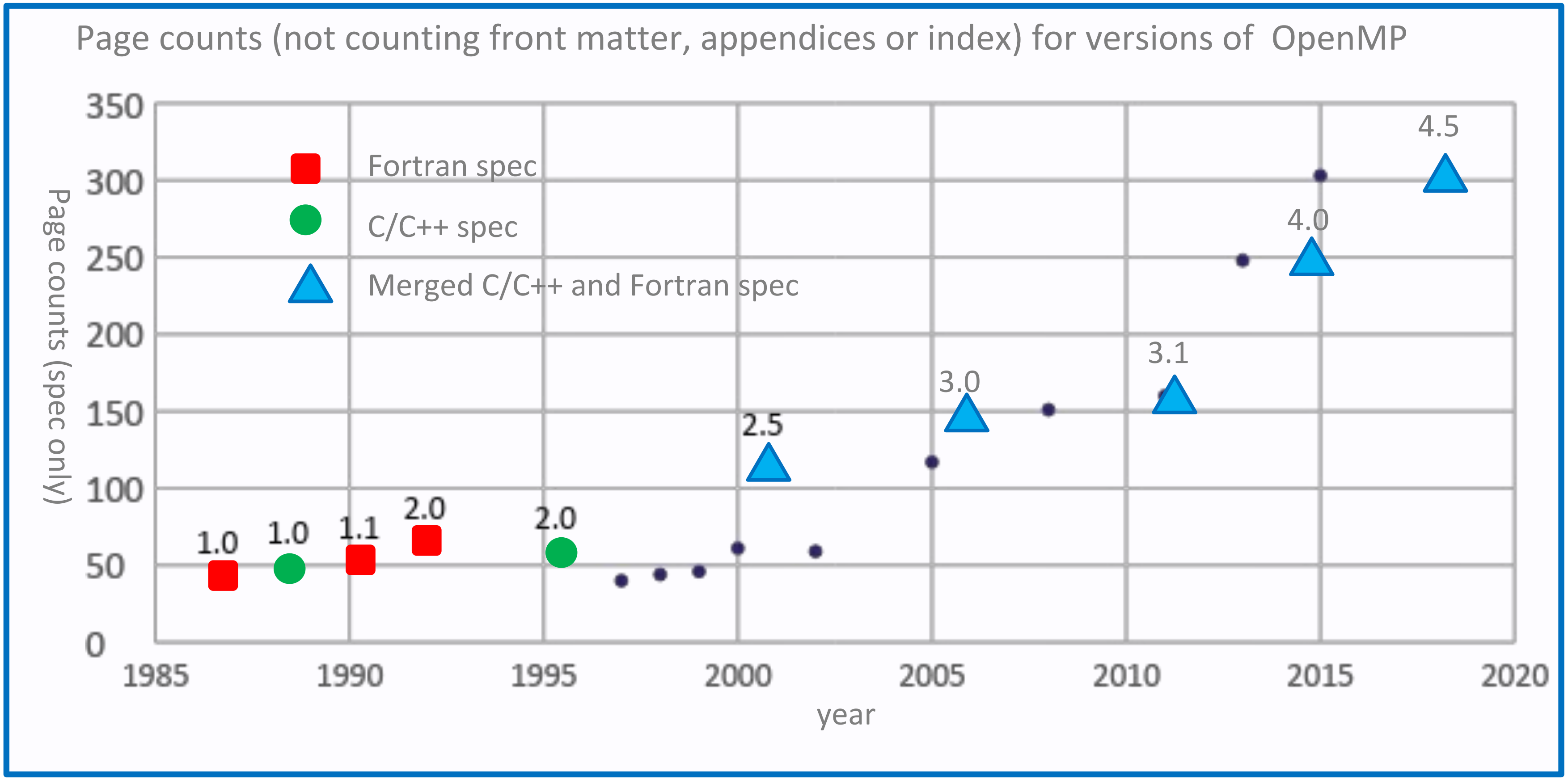
Exascale Systems

- Enabled rapid identification of critical needed features
- Rapid prototyping of potential solutions
- Provision of case studies to vendors
- Addition to standard within 18 months

The Growth of Complexity in OpenMP

- OpenMP started out as a simple interface for the application programmers more versed in their area of science than computer science.

- The complexity has grown considerably over the years!

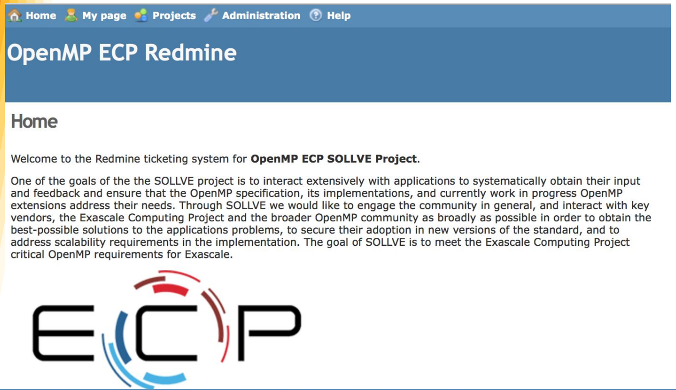


Major need for training, especially in more recent features, but also in techniques for performance

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Central Resource for OpenMP Users

http://openmp-ecp.ornl.gov



OpenMP ECP Redmine

Home

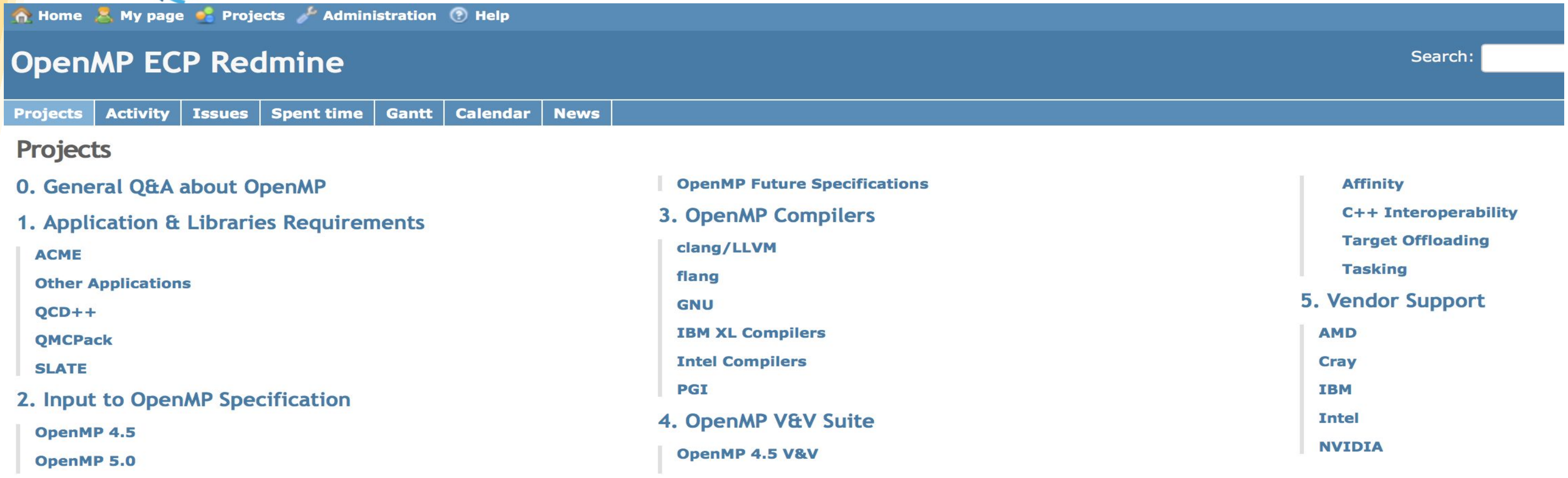
Welcome to the Redmine ticketing system for **OpenMP ECP SOLLVE Project**.

One of the goals of the the SOLLVE project is to interact extensively with applications to systematically obtain their input and feedback and ensure that the OpenMP specification, its implementations, and currently work in progress OpenMP extensions address their needs. Through SOLLVE we would like to engage the community in general, and interact with key vendors, the Exascale Computing Project and the broader OpenMP community as broadly as possible in order to obtain the best-possible solutions to the applications problems, to secure their adoption in new versions of the standard, and to address scalability requirements in the implementation. The goal of SOLLVE is to meet the Exascale Computing Project critical OpenMP requirements for Exascale.

Launched in Jan 2019

Central location for entire community of users

- General Q&A
- Specification interpretation
- App Use-cases for missing capabilities
- Implementation availability and prioritization
- Connect with the rest of community (Vendors,)



OpenMP ECP Redmine

Projects

0. General Q&A about OpenMP

1. Application & Libraries Requirements
- ACME
- Other Applications
- QCD++
- QMCPack
- SLATE

2. Input to OpenMP Specification
- OpenMP 4.5
- OpenMP 5.0
- OpenMP Future Specifications

3. OpenMP Compilers
- clang/LLVM
- flang
- GNU
- IBM XL Compilers
- Intel Compilers
- PGI

4. OpenMP V&V Suite
- OpenMP 4.5 V&V
  - Affinity
  - C++ Interoperability
  - Target Offloading
  - Tasking

5. Vendor Support
- AMD
- Cray
- IBM
- Intel
- NVIDIA

OpenMPCon 2019 & IWOMP 2019

OpenMP Developers Conference 2019 & 15th International Workshop on OpenMP

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

PARALLEL AND RECONFIGURABLE COMPUTING LAB

OpenMPCon: 9th-10th September, 2019
Tutorial: 11th September, 2019
IWOMP:12th -13th September, 2019
In Auckland, New Zealand

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

OpenMPCon 2019

http://parallel.auckland.ac.nz/openmpcon2019/
9th -10th September, 2019 in Auckland, NZ

IWOMP 2019

http://parallel.auckland.ac.nz/iwomp2019/
12th -13th September, 2019 in Auckland, NZ

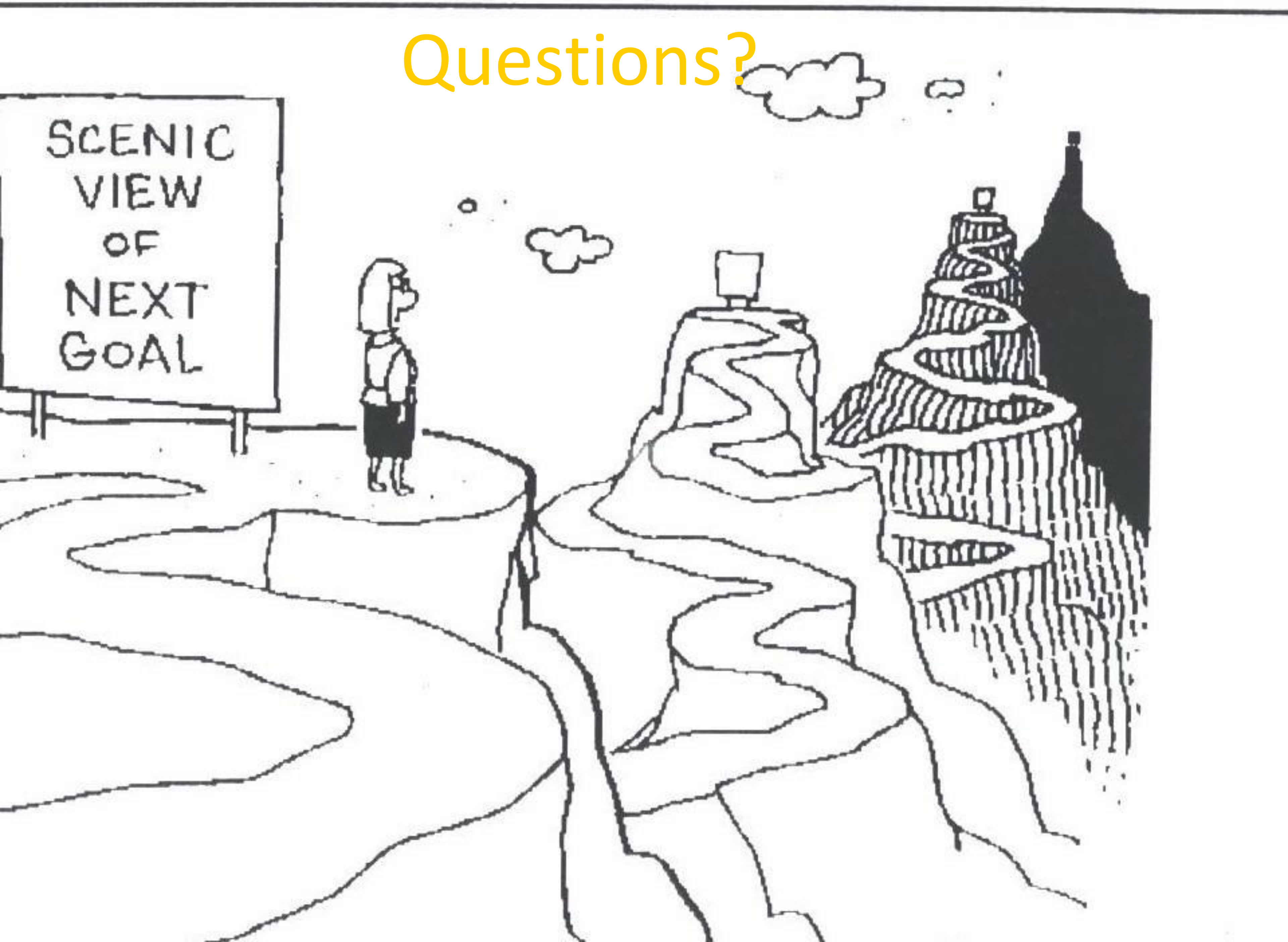
General Chair

Dr. Oliver Sinnen
PARC lab
Department of Electrical and Computer Engineering
University of Auckland
o.sinnen@Auckland.ac.nz

OpenMPCon 2019

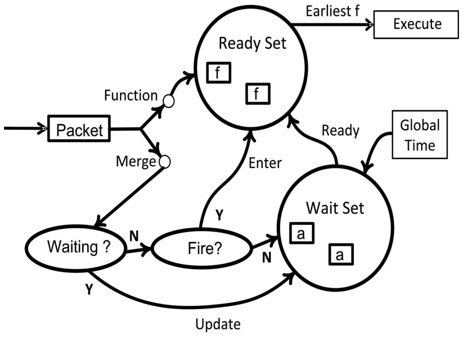
IWOMP 2019

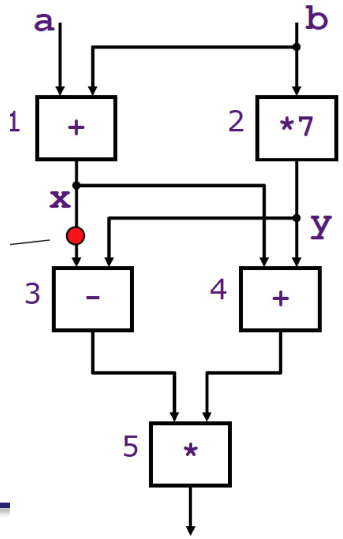
Questions?



One Size Fits All? Tasks and Data Flow

- Increasingly promoted by CS as parallel programming approach
- Inspired by the data flow execution model
  - Cilk, TBB, OpenMP,…
  - Legion, HPX, Parsec,…
  - Google Cloud Dataflow, Tensorflow, ….
- Node represents a task; edges represents inter-task dependencies
  - Main idea is to minimize synchronization
  - A task can begin execution only if all its predecessors have completed execution





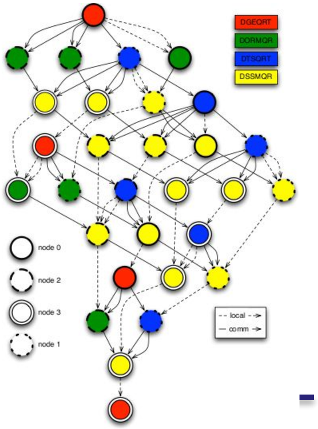


Fig. 5. DAG of QR for a 4x4 tile matrix.