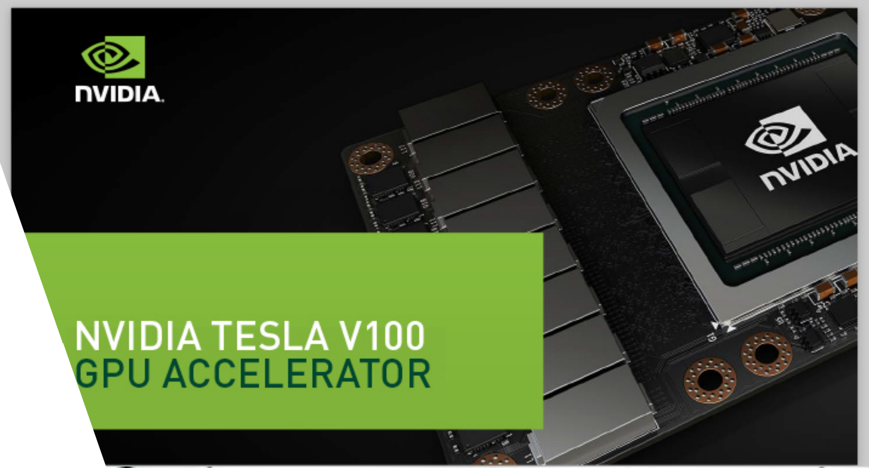
Programming for the Future : Are We There Yet?

Barbara Chapman
Stony Brook University
Brookhaven National Lab

SPPEXA Workshop, Versailles, March 21, 2019

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

We Live in Interesting Times



Development of large-scale applications is challenging. It is becoming much more so.

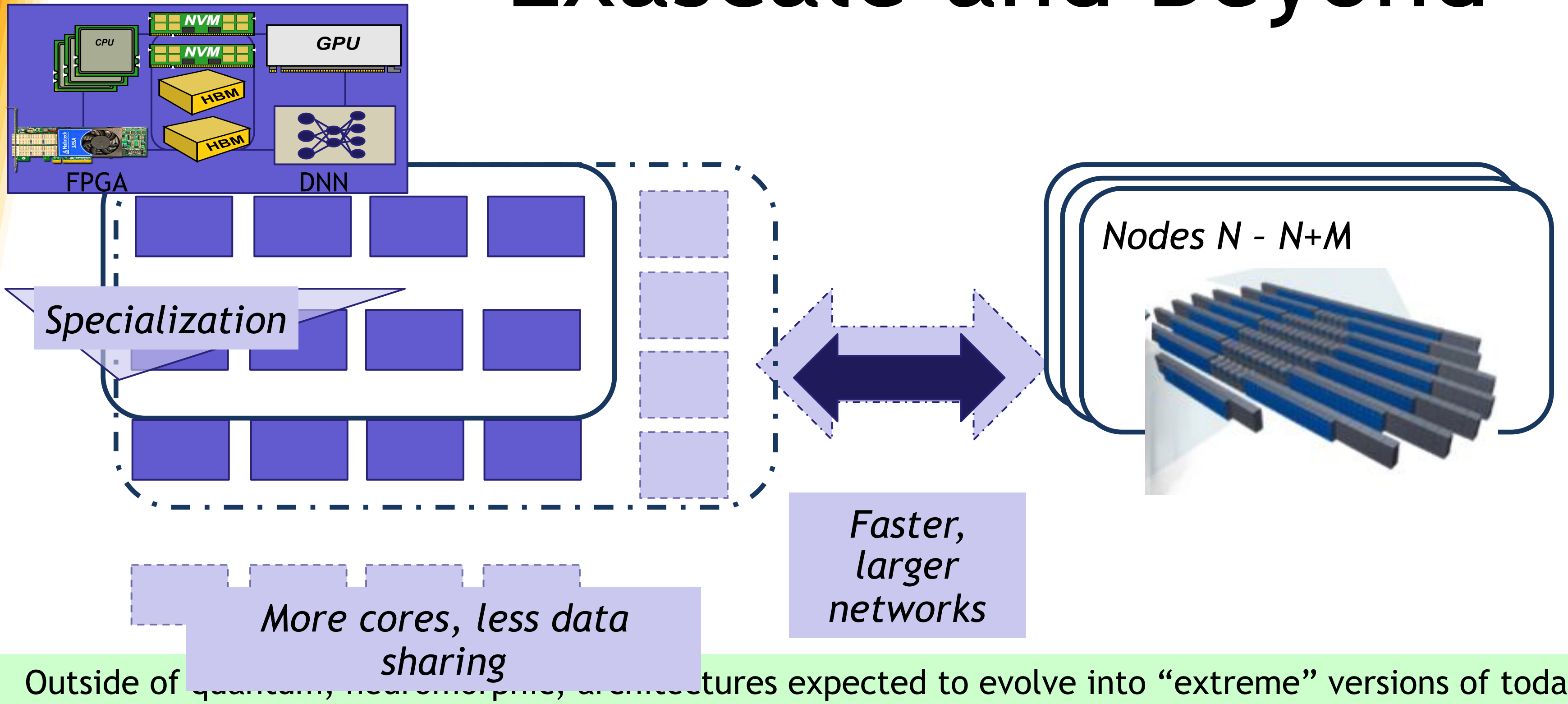
- ❑ Increasing complexity of computer hardware
  - ❑ Diversity in large-scale platforms
- ❑ Growing diversity of applications and users
  - ❑ Traditional scientific computations, AI, combination, workflows
- ❑ Increasing need for dynamic program adaptation
  - ❑ To handle changes in computation or resources
- ❑ Changing expectations on part of application developers
  - ❑ Python, TensorFlow, PyTorch, …
- ❑ Scalable performance, performance portability, productivity; power saving

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

Exascale and Beyond



Specialization

More cores, less data sharing

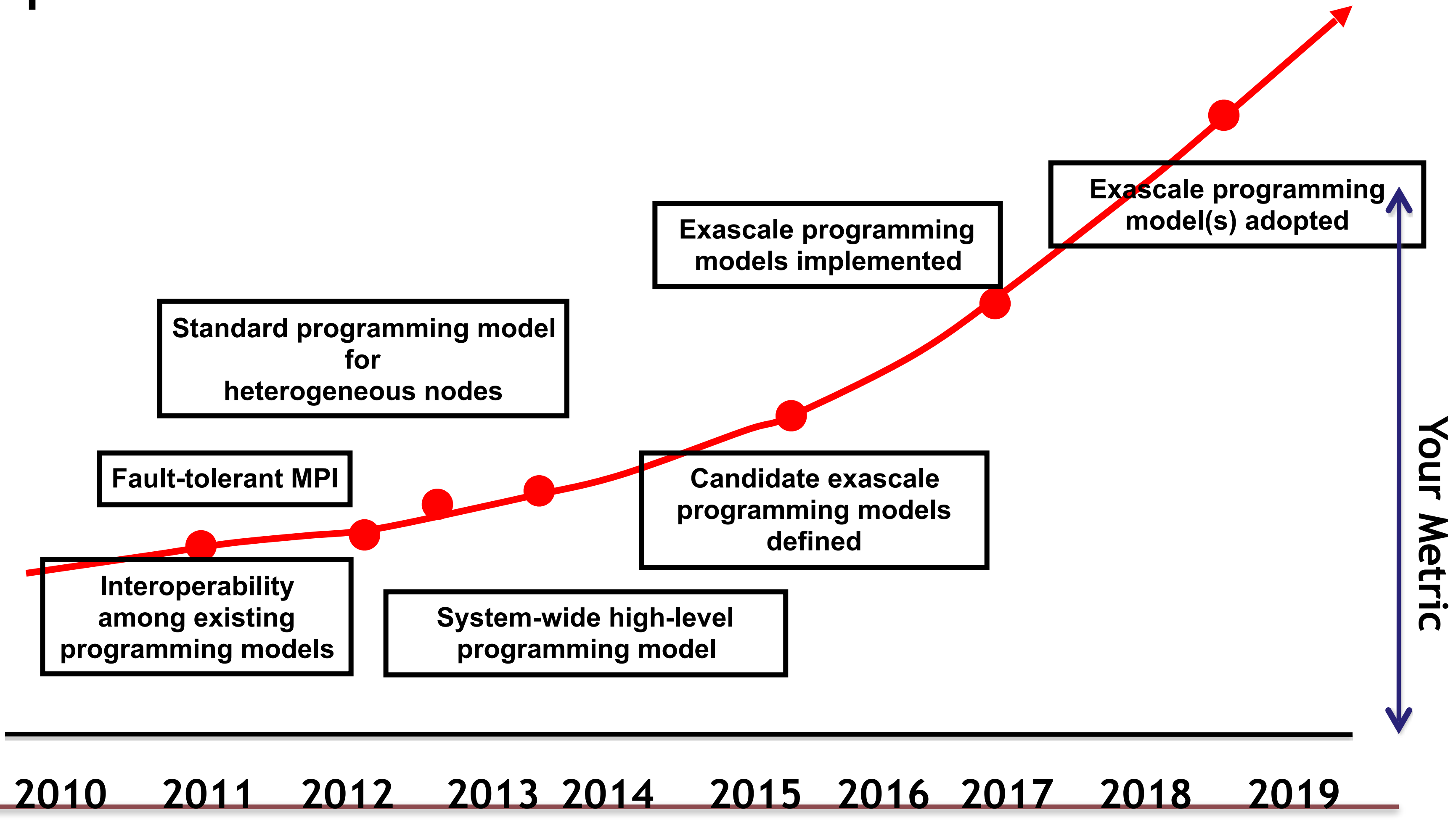
Faster, larger networks

- 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**

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

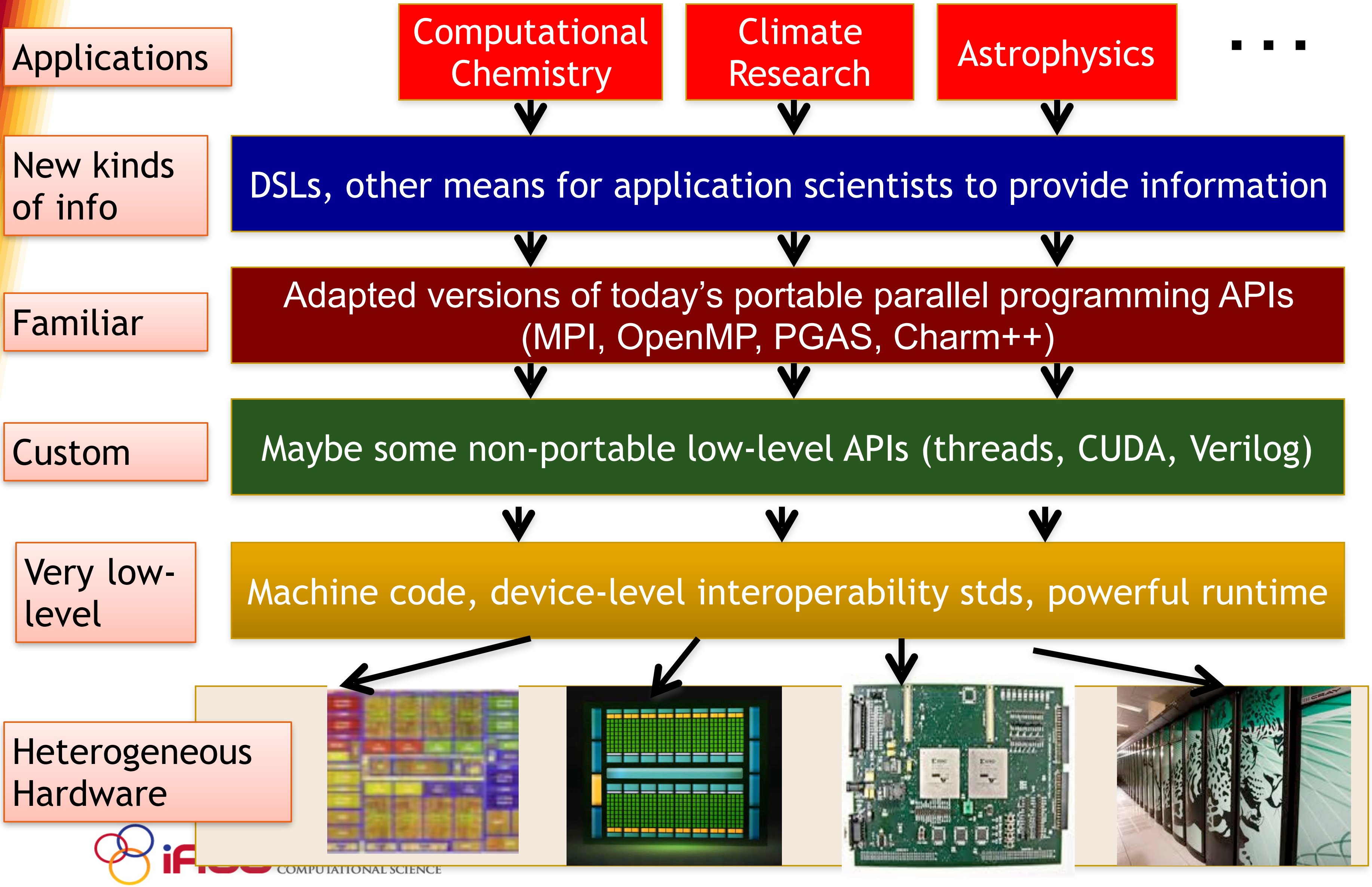
System-wide high-level programming model

Candidate exascale programming models defined

Exascale programming models implemented

Exascale programming model(s) adopted

A Layered Programming Approach



Applications

Computational Chemistry

Climate Research

Astrophysics

. . .

New kinds of info

DSLs, other means for application scientists to provide information

Familiar

Adapted versions of today's portable parallel programming APIs (MPI, OpenMP, PGAS, Charm++)

Custom

Maybe some non-portable low-level APIs (threads, CUDA, Verilog)

Very low-level

Machine code, device-level interoperability stds, powerful runtime

Heterogeneous Hardware

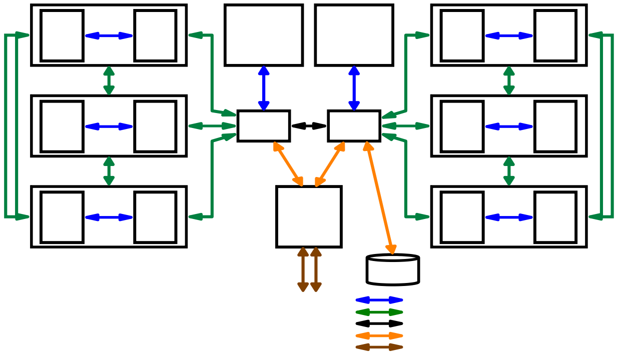
Summit

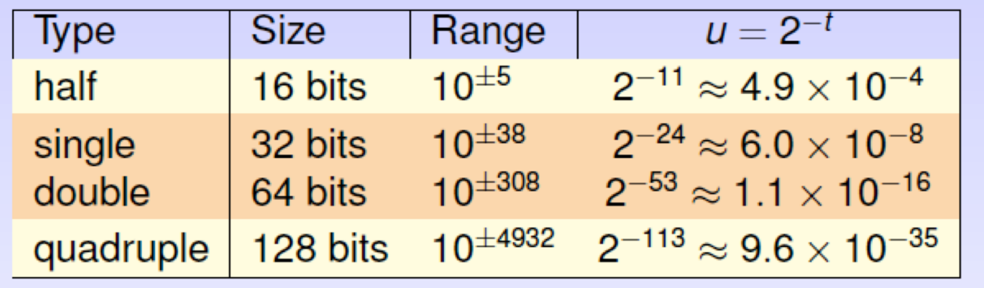
2 Power9s, 6 GPUs per node

27,648 NVIDIA Tesla V100s, each with:

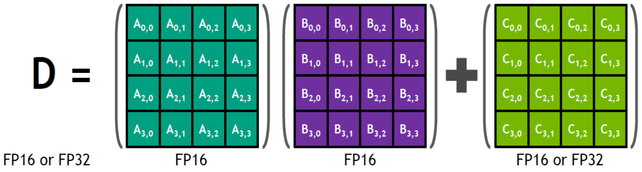
- 5120 CUDA cores
- 640 Tensor cores
- 300 GB/s BW (NVLink 2.0)
- 20MB registers, 16MB cache, 16GB HBM2 @900 GB/s
- 7.5 DP TFLOPS; 15 SP TFLOPS, 120 FP16 TFLOPS

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	$10^{\pm 4932}$	$2^{-113} \approx 9.6 \times 10^{-35}$



$$D = \left(\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix}\right) + \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{bmatrix}$$

D = AB + C

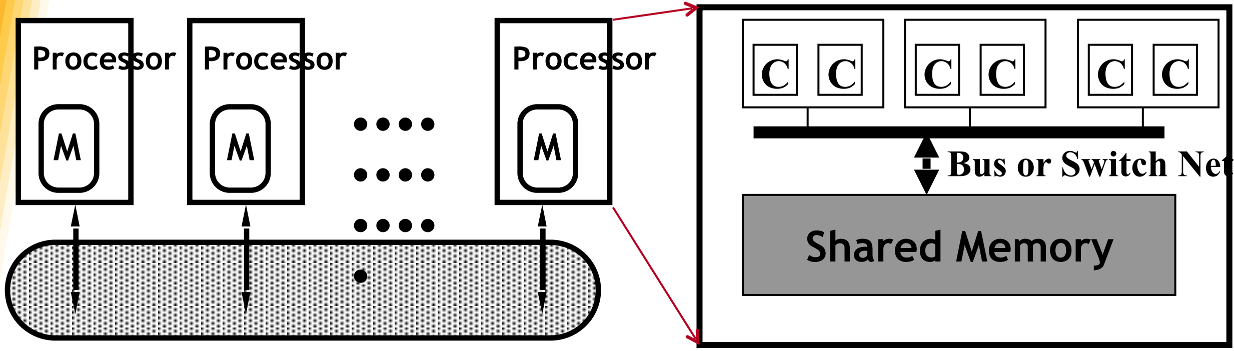
The Modeling & Simulation community can benefit from utilizing mixed / reduced precision

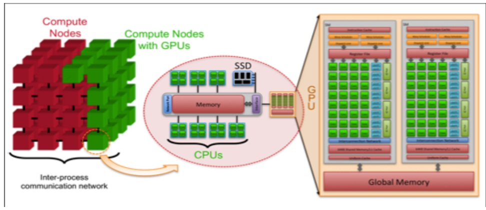
- Eg: Possible to achieve 4x FP64 peak for 64bit LU on V100 with iterative mixed precision (Dongarra et al.)

Programming: HPC vs. DL

Nodes in a cluster

SMP Multicore Architecture

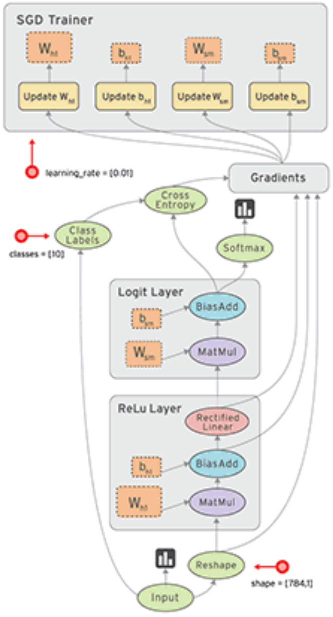


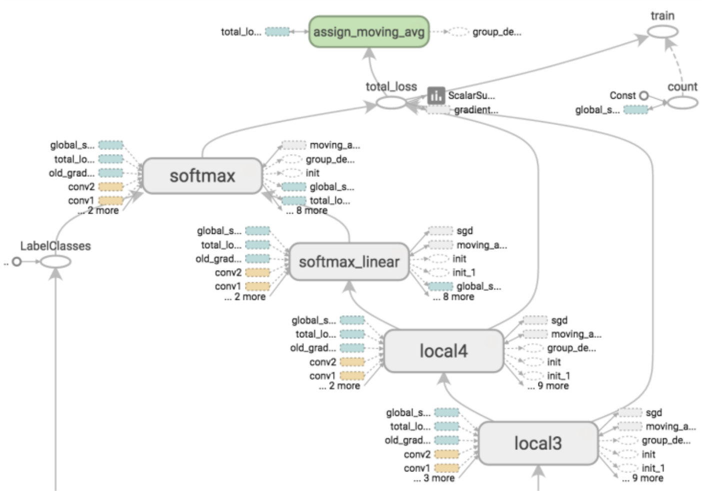


Network for Data Exchange

Programmed using MPI

Programmed using OpenMP





TensorFlow programming interface

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

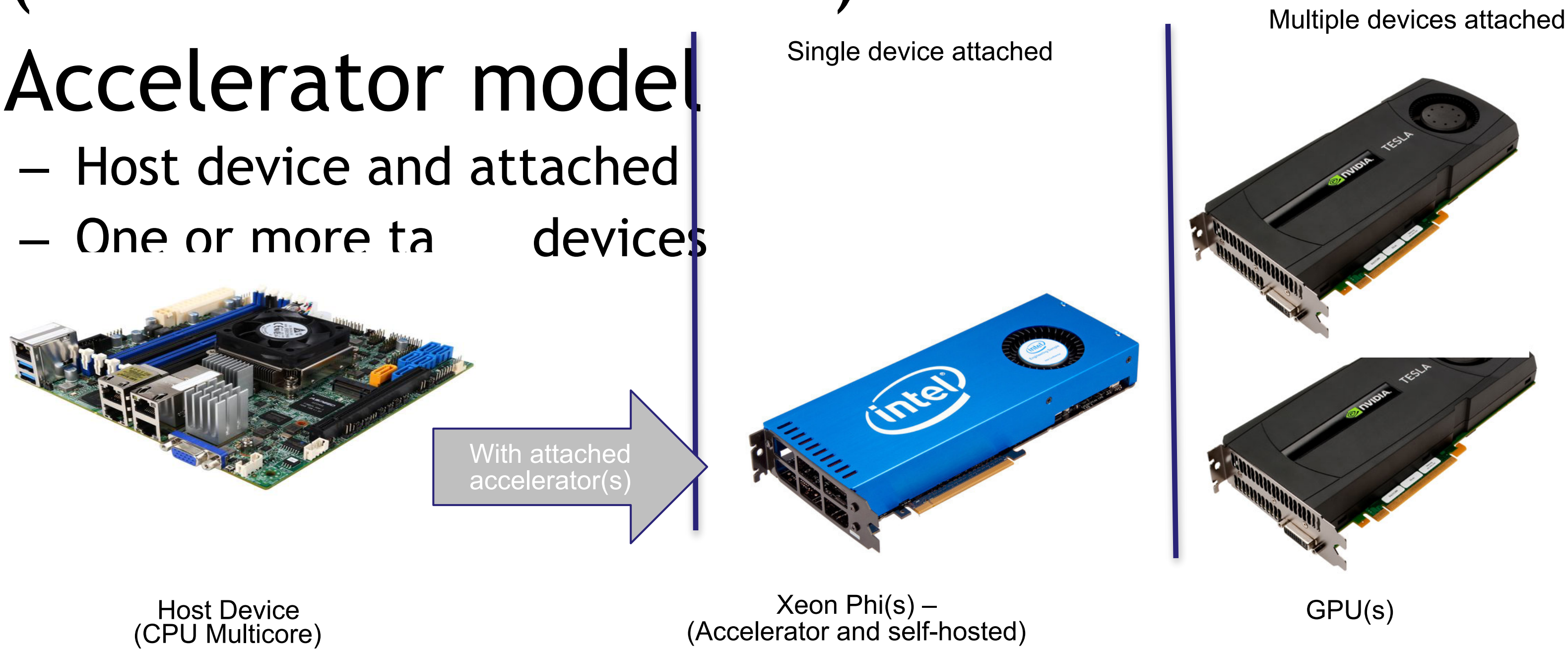
- It is now easy for academics to join

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
UNIVERSITY OF DELAWARE
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.

OpenMP 4.5 – Accelerator Model

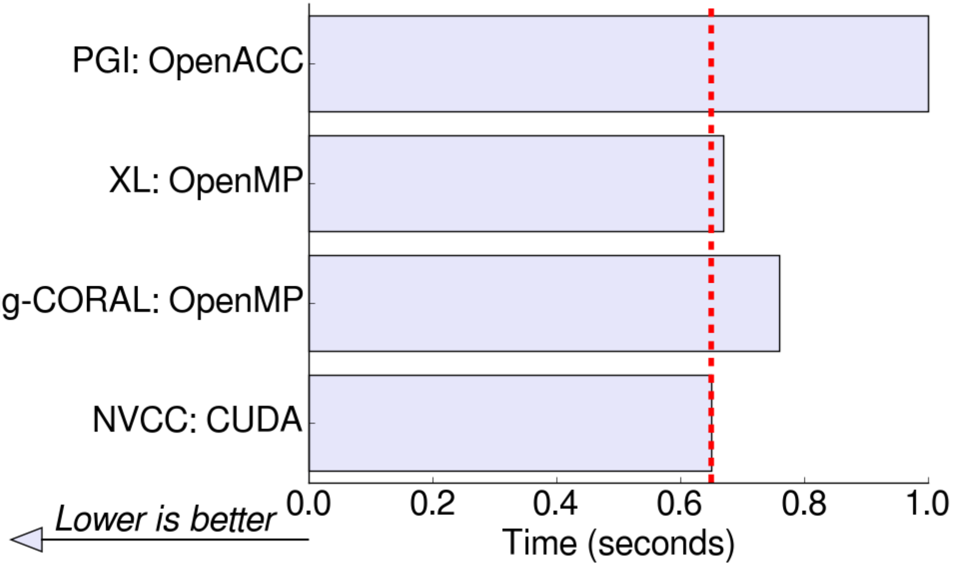
- OpenMP 4+ supports heterogeneous systems (accelerators/devices)
- Accelerator model
  - Host device and attached
  - One or more ta devices


Single device attached
Multiple devices attached
With attached accelerator(s)
Host Device
(CPU Multicore)
Xeon Phi(s) –
(Accelerator and self-hosted)
GPU(s)

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

Emerging OpenMP Features

- New features in OpenMP 5.0
  - Memory model: deal with memory heterogeneity

```
#pragma omp allocate(A) allocator(omp_high_bw_mem_alloc)
```

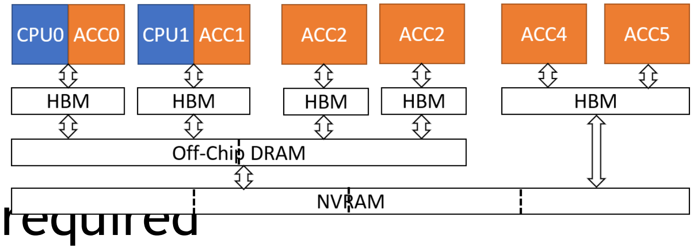
  - “concurrent” directive: descriptive parallel method

```
#pragma omp for order(concurrent)  for (int i = 0; i < N; i++) {…}
```

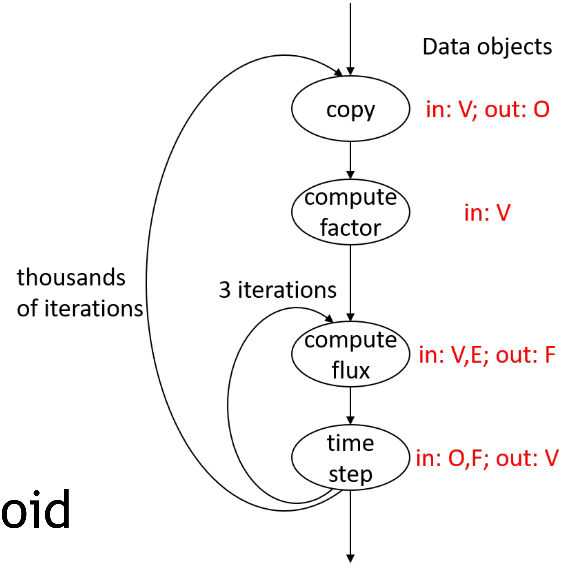
- Also support for unified memory, deep copy of data, metadirectives, task affinity, and more

- Significant implementation effort to support these well

Extreme Heterogeneity: Memory Management

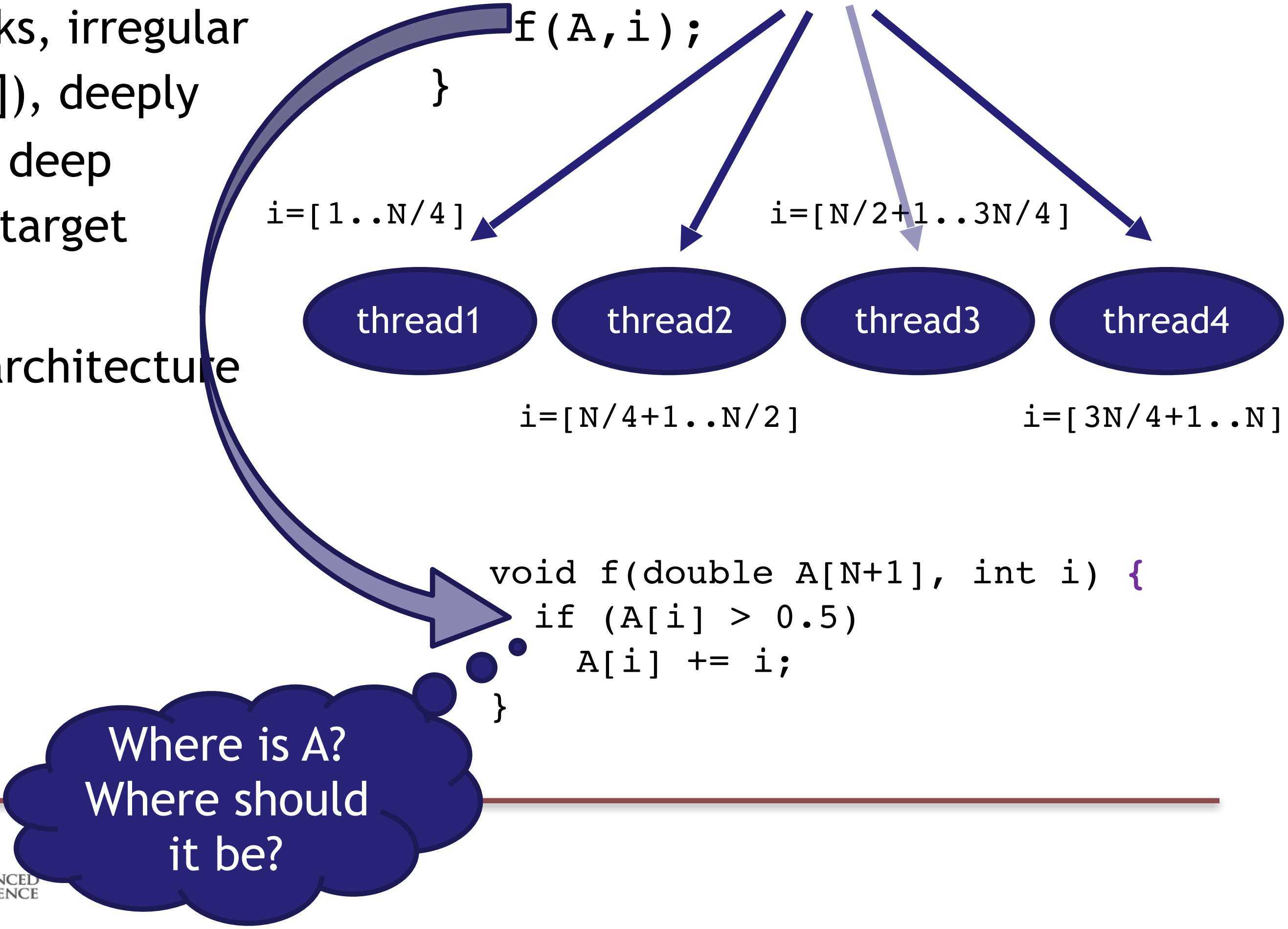


- Expect significantly more complex memory systems in the future
- Programming environment innovations are required to cope with such complexity
- Programming model
  - Provide high level data abstractions
  - Improve memory systems' programmability
  - Integrate performance modeling
- Compiler and runtime
  - Unified memory optimizations (prefetching, pinning) to avoid thrashing
  - Coarse grained data optimization
  - Better usage of hardware capabilities e.g. avoid caching of non-temporal accesses



OpenMP Loop Feature

- Work-sharing directive: split loop iterations among threads
- Significant challenges: large reduction basic blocks, irregular accesses (e.g. a[b[i]]), deeply nested conditionals, deep copy / allocation at target (where is my data?)
- Strategies vary per architecture



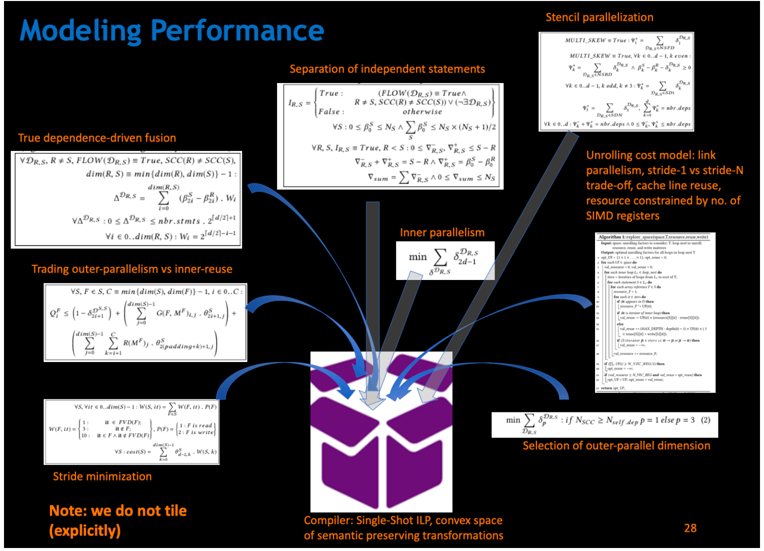
```
#pragma omp for order(concurrent
for (i = 1; i < N; i++) {
  f(A,i);
}
```

```
void f(double A[N+1], int i) {
  if (A[i] > 0.5)
    A[i] += i;
}
```

Model-driven, Composable and Multi-Target Compiler Optimizations

Motivation: reduce performance gap between general purpose and domain specific compiler frameworks

Impact: substantial performance improvements, reduced application tuning time



Modeling Performance

True dependence-driven fusion

$$\forall \mathcal{D}_{R,S},\ R \neq S,\ FLOW(\mathcal{D}_{R,S}) \equiv True,\ SCC(R) \neq SCC(S),$$
$$dim(R,S) \equiv min\{dim(R), dim(S)\} - 1 :$$
$$\Delta^{\mathcal{D}_{R,S}} = \sum_{i=0}^{dim(R,S)} (\beta_{2i}^{S} - \beta_{2i}^{R})\ .\ W_i$$
$$\forall \Delta^{\mathcal{D}_{R,S}} : 0 \leq \Delta^{\mathcal{D}_{R,S}} \leq nbr.stmts\ .\ 2^{\lceil d/2 \rceil + 1}$$
$$\forall i \in 0..dim(R,S) : W_i = 2^{\lceil d/2 \rceil - i - 1}$$

Separation of independent statements

$$I_{R,S} = \begin{cases} True : & (FLOW(\mathcal{D}_{R,S}) \equiv True \wedge \\ & R \neq S, SCC(R) \neq SCC(S)) \vee (\neg \exists \mathcal{D}_{R,S}) \\ False : & otherwise \end{cases}$$
$$\forall S : 0 \leq \beta_0^S \leq N_S \wedge \sum_S \beta_0^S \leq N_S \times (N_S + 1)/2$$
$$\forall R, S, I_{R,S} \equiv True, R < S : 0 \leq \nabla_{R,S}^{-}, \nabla_{R,S}^{+} \leq S - R$$
$$\nabla_{R,S}^{-} + \nabla_{R,S}^{+} = S - R \wedge \nabla_{R,S}^{+} = \beta_0^S - \beta_0^R$$
$$\nabla_{sum} = \sum \nabla_{R,S}^{-} \wedge 0 \leq \nabla_{sum} \leq N_S$$

Stencil parallelization

$$MULTI\_SKEW \equiv True : \Psi_1^{+} = \sum_{\mathcal{D}_{R,S} \in NSFD} \delta_1^{\mathcal{D}_{R,S}}$$
$$MULTI\_SKEW \equiv True, \forall k \in 0..d-1, k\ even :$$
$$\Psi_k^{+} = \sum_{\mathcal{D}_{R,S} \in NSBD} \delta_k^{\mathcal{D}_{R,S}} \wedge \beta_k^S - \beta_k^R - \delta_k^{\mathcal{D}_{R,S}} \geq 0$$
$$\forall k \in 0..d-1, k\ odd, k \neq 3 : \Psi_k^{+} = \sum_{\mathcal{D}_{R,S} \in SD1} \delta_k^{\mathcal{D}_{R,S}}$$
$$\Psi_3^{+} = \sum_{\mathcal{D}_{R,S} \in SDN} \delta_3^{\mathcal{D}_{R,S}},\ \sum_{k=0}^{d} \Psi_k^{+} = nbr.deps$$
$$\forall k \in 0..d : \Psi_k^{-} + \Psi_k^{+} = nbr.deps \wedge 0 \leq \Psi_k^{-}, \Psi_k^{+} \leq nbr.deps$$

Unrolling cost model: link parallelism, stride-1 vs stride-N trade-off, cache line reuse, resource constrained by no. of SIMD registers

```
Algorithm 1: explore_space(space,T,resource,reuse,write)
Input: space: unrolling factors to consider; T: loop nest to unroll;
       resource, reuse, and write matrices
Output: optimal unrolling factors for all loops in loop nest T
opt_UF = {1 × 1 × ... × 1}; opt_reuse = 0;
for each UF ∈ space do
    val_resource = 0; val_reuse = 0;
    for each inner loop $L_i$ ∈ loop_nest do
        iters = iterators of loops from $L_i$ to root of T;
        for each statement $S \in L_i$ do
            for each array reference $F \in S$ do
                resource_F = 1;
                for each it ∈ iters do
                    if (it appears in F) then
                        resource_F *= UF(it);
                    if (it is iterator of inner loop) then
                        val_reuse -= UF(it) × (resource[S][it] - reuse[S][it]);
                    else
                        val_reuse += (MAX_DEPTH - depth(it) + 1) × UF(it) × ( 3
                          × reuse[S][it] + write[S][it]);
                    if (∃ iterator jt ∈ iters s.t. it → jt or jt → it) then
                        val_reuse = −∞;
                val_resource += resource_F;
    if ($\prod_i$ UF(i) ≥ N_VEC_REG/2) then
        opt_reuse = −∞;
    if (val_resource ≤ N_VEC_REG and val_reuse > opt_reuse) then
        opt_UF = UF; opt_reuse = val_reuse;
return opt_UF;
```

Inner parallelism

$$\min \sum_{\delta^{\mathcal{D}_{R,S}}} \delta_{2d-1}^{\mathcal{D}_{R,S}}$$

Trading outer-parallelism vs inner-reuse

$$\forall S, F \in S, C \equiv min\{dim(S), dim(F)\} - 1, i \in 0..C :$$
$$Q_i^F \leq \left(1 - \delta_{2i+1}^{\mathcal{D}^{S,S}}\right) + \left(\sum_{j=0}^{dim(S)-1} G(F, M^F)_{i,j}\ .\ \theta_{2i+1,j}^{S}\right) +$$
$$\left(\sum_{j=0}^{dim(S)-1} \sum_{k=i+1}^{C} R(M^F)_j\ .\ \theta_{2(padding+k)+1,j}^{S}\right)$$

$$\forall S, \forall it \in 0..dim(S)-1 : W(S,it) = \sum_{F \in S} W(F,it)\ .\ P(F)$$
$$W(F,it) = \begin{Bmatrix} 1 : & it \in FVD(F); \\ 3 : & it \notin F; \\ 10 : & it \in F \wedge it \notin FVD(F) \end{Bmatrix},\ P(F) = \begin{Bmatrix} 1 : F\ is\ read \\ 2 : F\ is\ write \end{Bmatrix}$$
$$\forall S : cost(S) = \sum_{k=0}^{dim(S)-1} \theta_{d-2,k}^{S}\ .\ W(S,k)$$

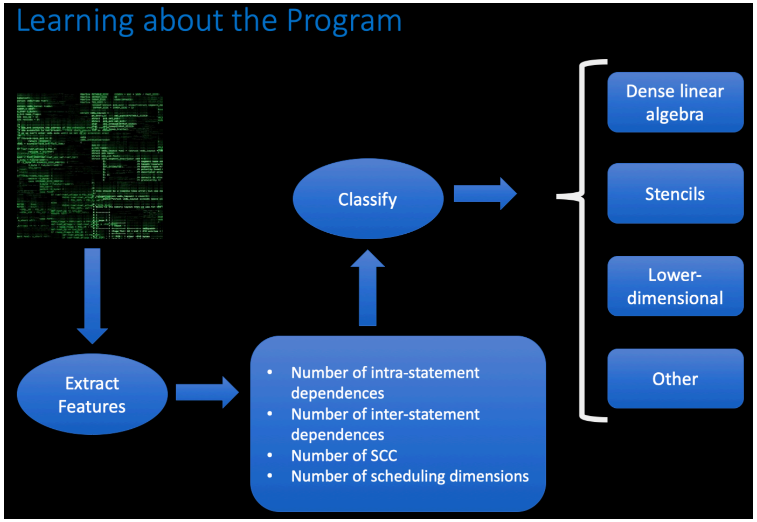
Stride minimization

$$\min \sum_{\mathcal{D}_{R,S}} \delta_p^{\mathcal{D}_{R,S}} : if\ N_{SCC} \geq N_{self.dep}\ p = 1\ else\ p = 3 \quad (2)$$

Selection of outer-parallel dimension

Note: we do not tile (explicitly)

Compiler: Single-Shot ILP, convex space of semantic preserving transformations



Learning about the Program

Extract Features

- Number of intra-statement dependences
- Number of inter-statement dependences
- Number of SCC
- Number of scheduling dimensions

Classify

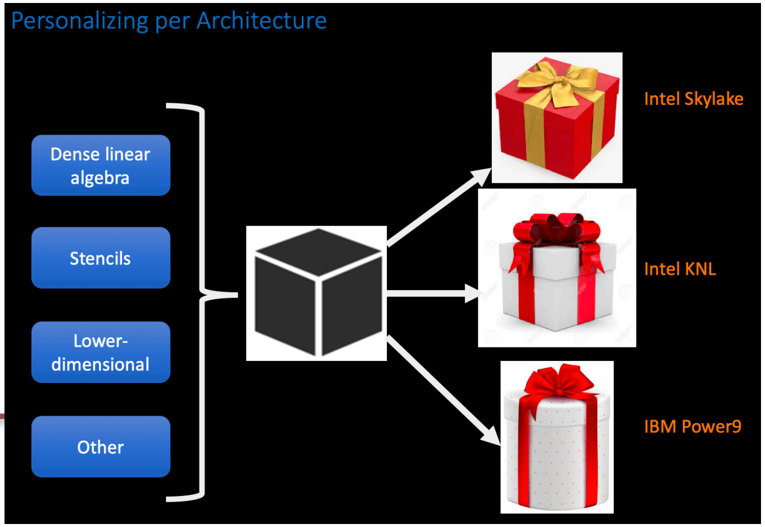
Dense linear algebra

Stencils

Lower-dimensional

Other

Martin Kong: To appear in ACM Programming Language Design and Implementation, PLDI'19



Personalizing per Architecture

Dense linear algebra

Stencils

Lower-dimensional

Other

Intel Skylake

Intel KNL

IBM Power9

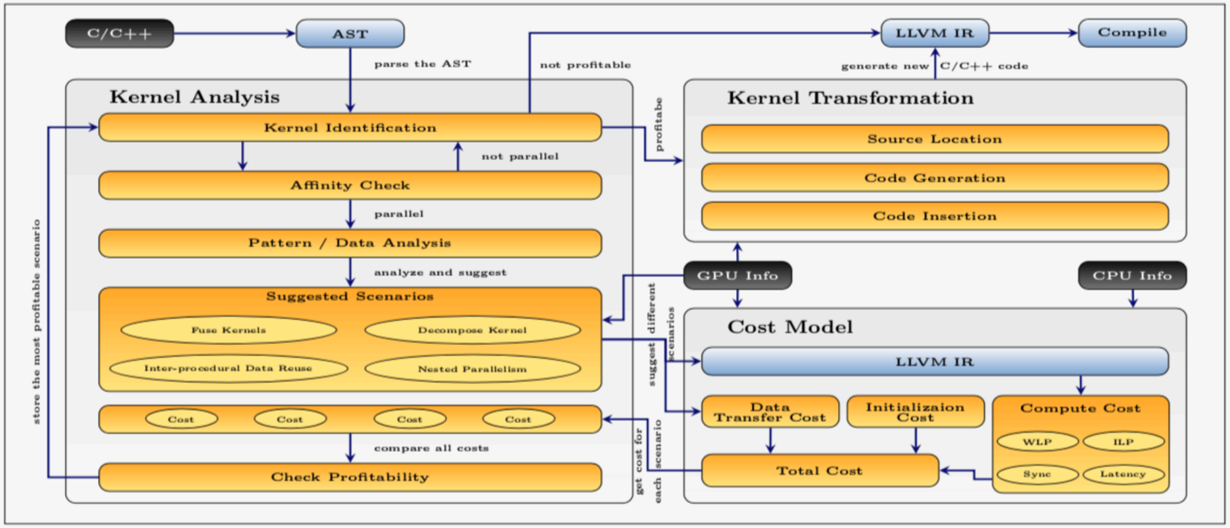
iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Kernel Fusion/Decomposition for Automatic GPU-offloading

Motivation: automatic GPU-offloading capabilities in LLVM to maximize application performance and user productivity

Impact: immediate benefit for medium to large scale mathematical libraries (e.g. Grid++ Lattice QCD parallel library)

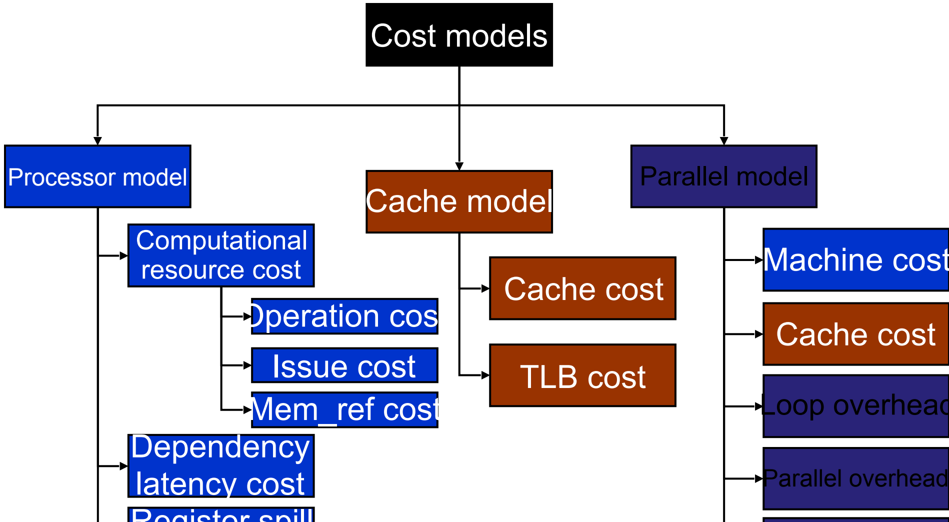
High Level Flow

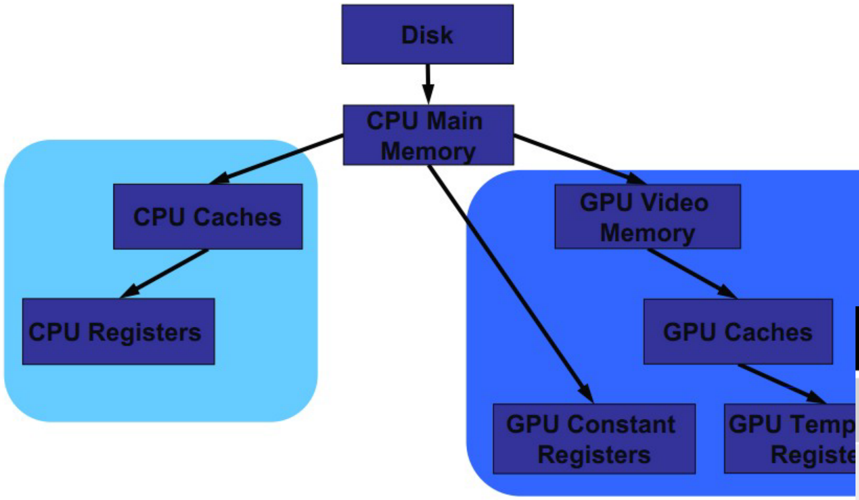


In the backend we use LLVM/Clang tools. Analysis Pass identifies all possible kernels in the code and suggests several variants for offloading. Cost Model statically compares the potential performance amongst the various kernels generated. Kernel Transformation adds pertinent OpenMP code to the kernel to support offloading.

Third Place, ACM Student Research Competition at the International Conference of Code Generation and Optimization (CGO’19, Washingtor DC)
Alok Mishra, Martin Kong, Barbara Chapman

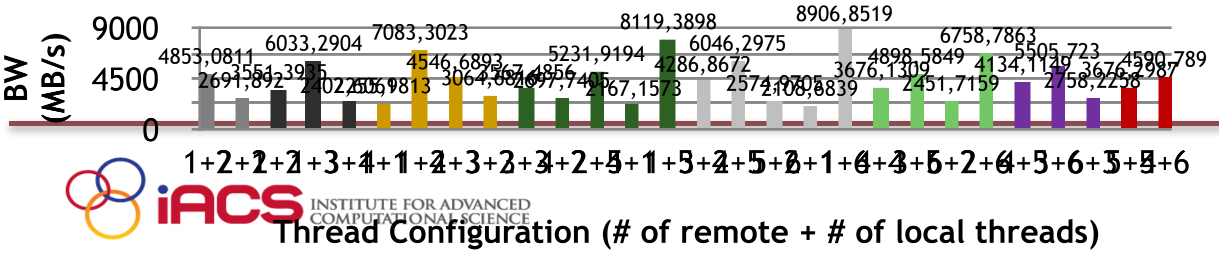
Model-Driven Translation





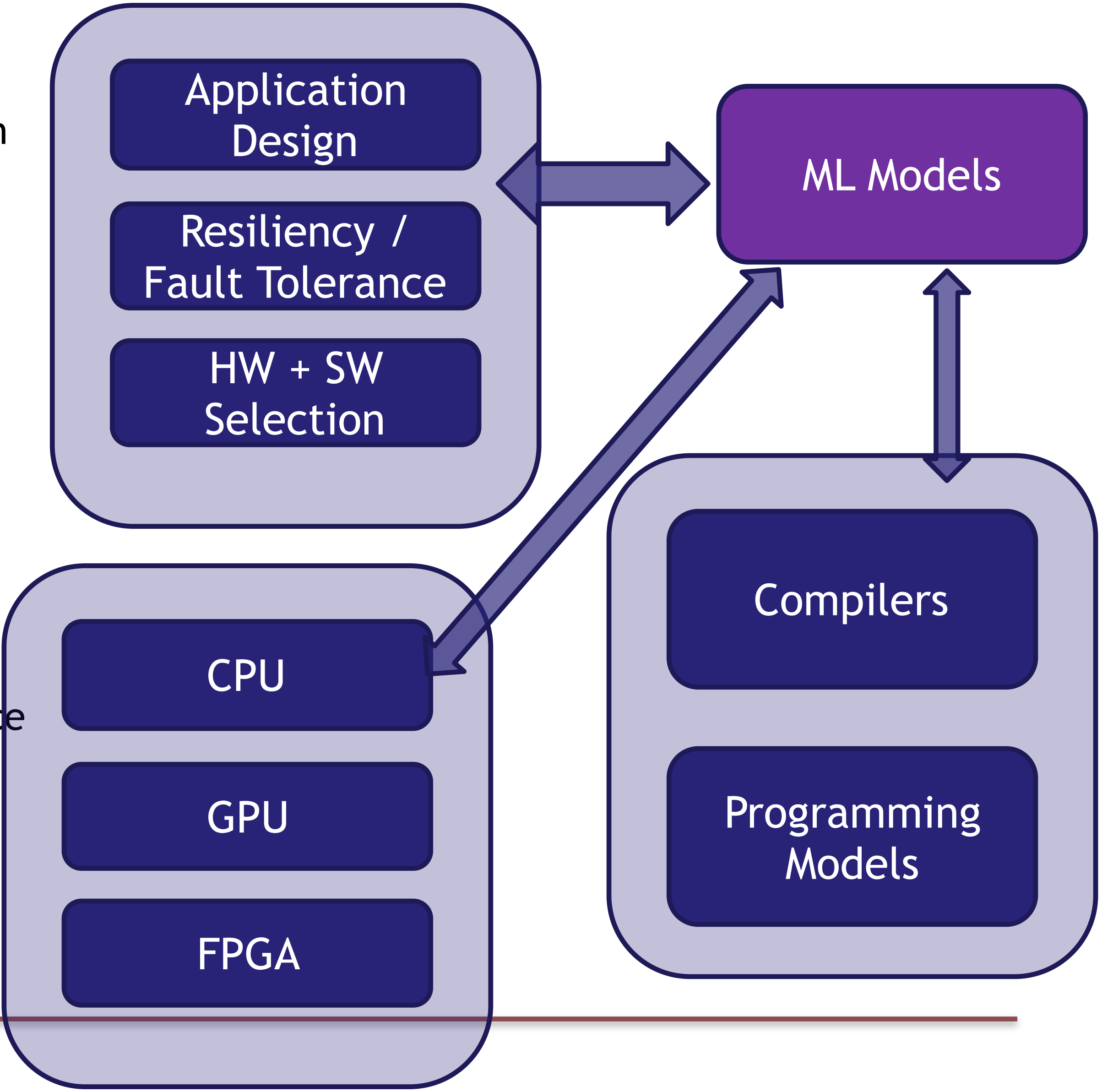
Operation	Energy (pJ)	DP FLOPs	Insts*
I$ Fetch	33	0.67	2.0
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360

HT3 BW vs Threads



Exploiting ML in Software Stack

- ❑Design: determine best target machine given inherent application traits (memory or compute driven, accuracy?)
- ❑Compiler/runtime support for stringent power caps
- ❑Resiliency and fault tolerance: when and what to checkpoint?
- ❑Compiler options and runtime features embody a large and complex optimization space
- ❑Impact on application performance
- ❑Different application classes require different optimization strategies



Application Design

Resiliency / Fault Tolerance

HW + SW Selection

ML Models

CPU

GPU

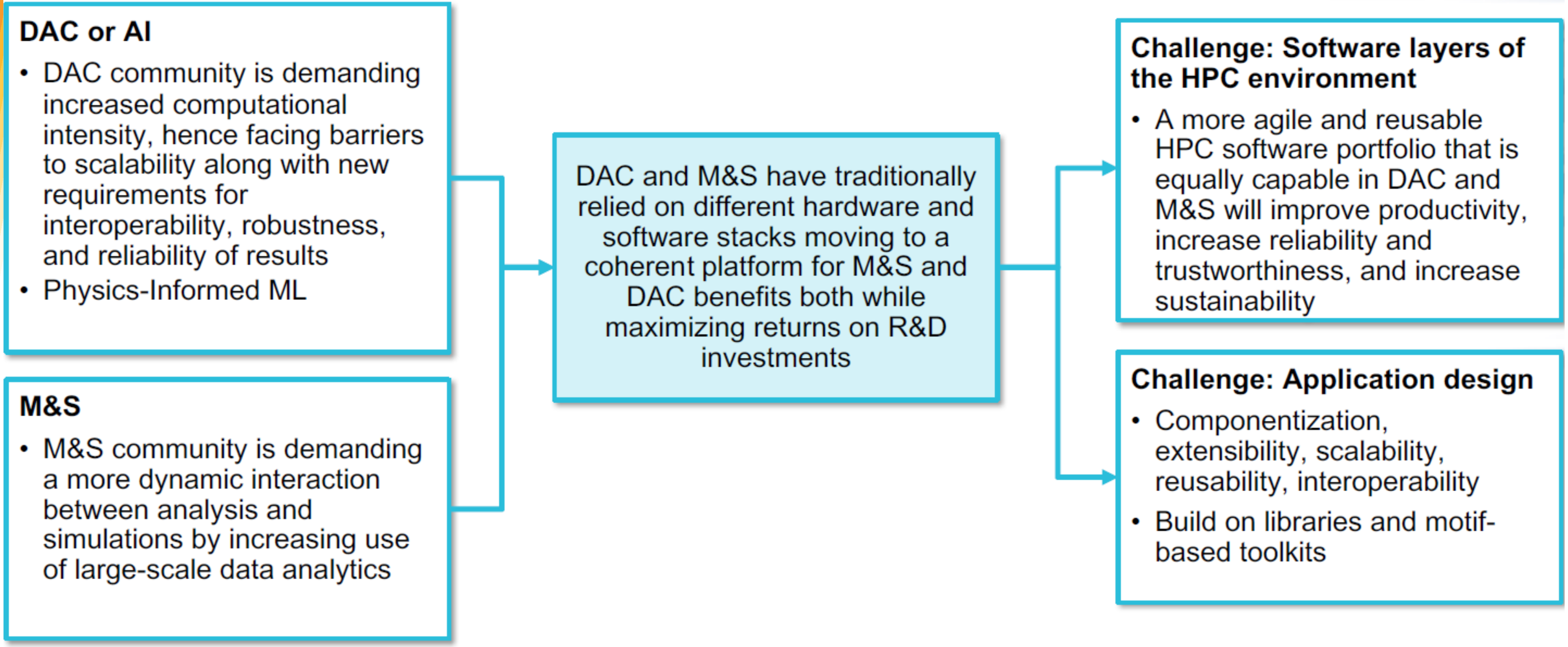
FPGA

Compilers

Programming Models

Changing Workloads

Data Analytic Computing (DAC) and Modeling and Simulation (M&S)



DAC or AI

- DAC community is demanding increased computational intensity, hence facing barriers to scalability along with new requirements for interoperability, robustness, and reliability of results
- Physics-Informed ML

M&S

- M&S community is demanding a more dynamic interaction between analysis and simulations by increasing use of large-scale data analytics

DAC and M&S have traditionally relied on different hardware and software stacks moving to a coherent platform for M&S and DAC benefits both while maximizing returns on R&D investments

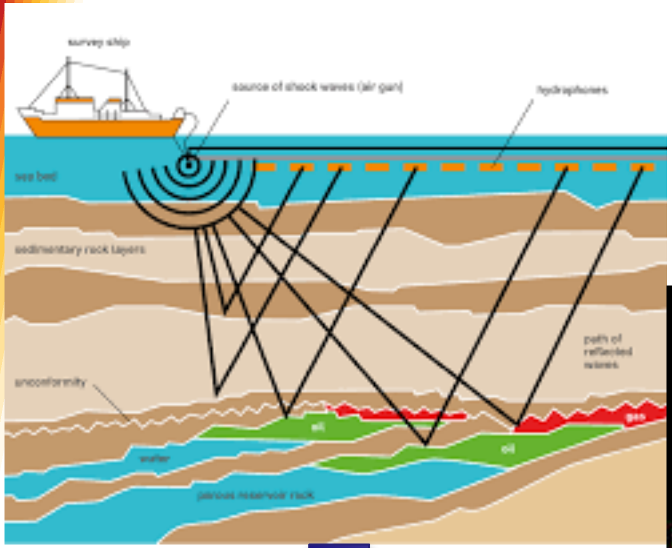
Challenge: Software layers of the HPC environment

- A more agile and reusable HPC software portfolio that is equally capable in DAC and M&S will improve productivity, increase reliability and trustworthiness, and increase sustainability

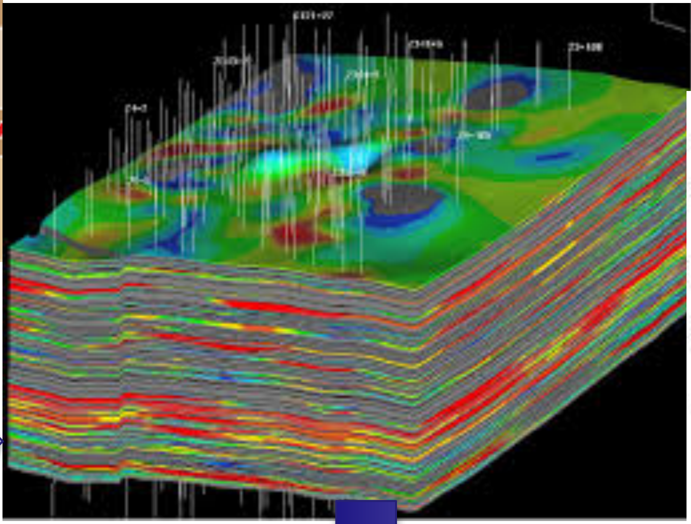
Challenge: Application design

- Componentization, extensibility, scalability, reusability, interoperability
- Build on libraries and motif-based toolkits

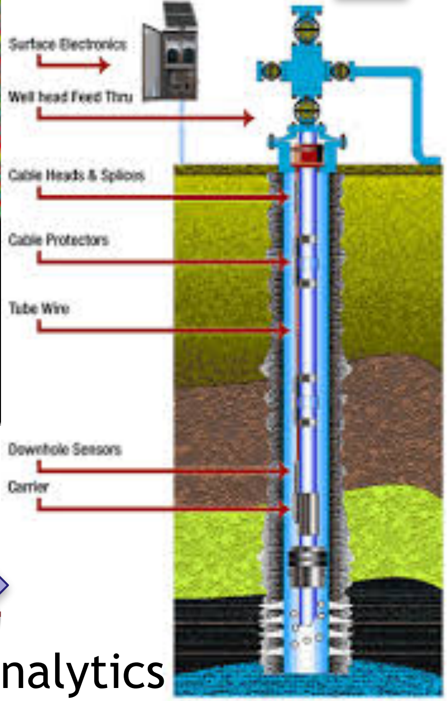
Large Scale Petroleum: Combining HPC and Data Analytics



Seismic Data Processing and Interpretation



Reservoir Simulation

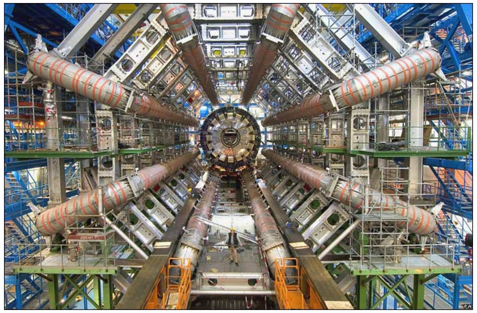


Oilfield Data Analytics

- A combination of Physics and Data Science

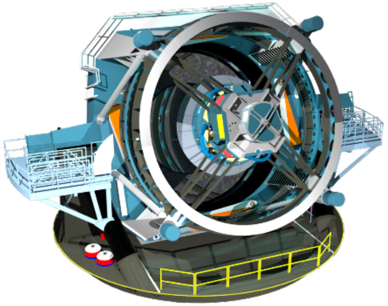
Scalable Learning Matters!

Large Hadron Collider



25GB/s, >200PB

LSST



20TB/night, 73PB

Transmission EM

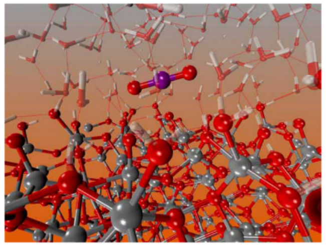
3GB/s

ExaFEL (ECP)

LCLS

10PB

MD Trajectories



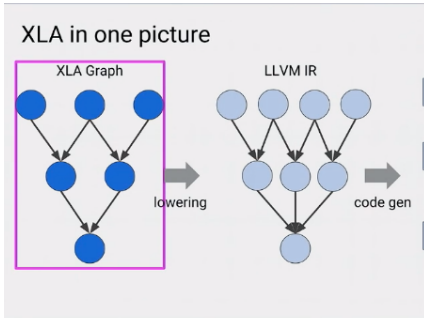
~32PB / simulation

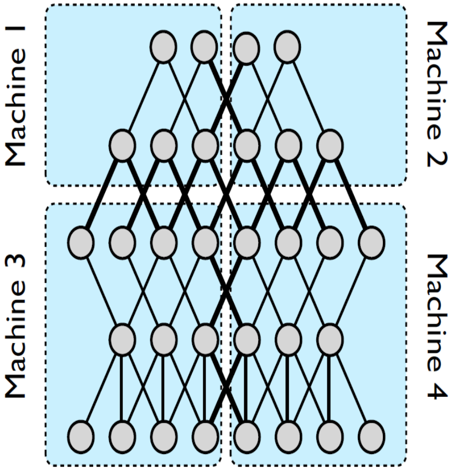
Meta-genomics

~15GB / sample

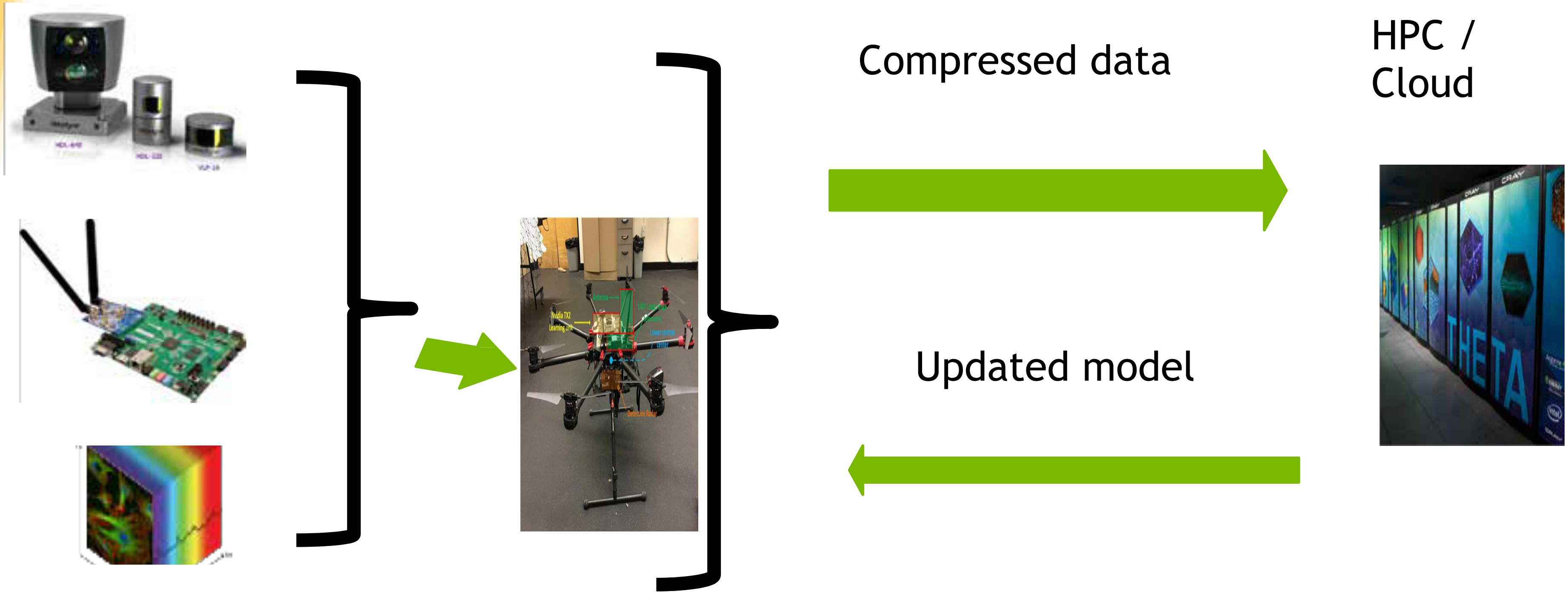
Machine Learning for Scientific Applications

- Today's machine learning frameworks are not easy to scale
  - Expensive communication / synchronization in SGD
  - Significant effort optimizing frameworks for CPU, GPU
  - Use of OpenMP
  - Research making advances in distributed parallelization
- State of art
  - TensorFlow, PyTorch,…
  - Heavy optimization of operations
  - Otherwise, much lacking in compiler technology
- Language and compiler enhancements
  - Improved single node translation
  - Efficient translation of associated user code in Python
  - DSL features for enhanced ML?
  - Path toward integration with scientific application code?
  - Representation of data and computation flow for wholistic optimization?





ML at the Edge



Compressed data

HPC / Cloud

Updated model

Inference at (potentially many) edge devices, major training in central computer. Regular updates of model must be sent back to edge.

One Size Fits All? Tasks and Data Flow

- Inspired by the data flow execution model (Dennis, 1970s)
  - Cilk, TBB, OpenMP,…
  - Legion, HPX, Parsec,…
  - Google Cloud Dataflow, Tensorflow, ….
- Not a “natural” approach for many applications
  - Implementations often forget data locality

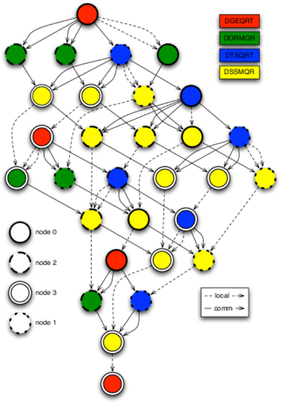
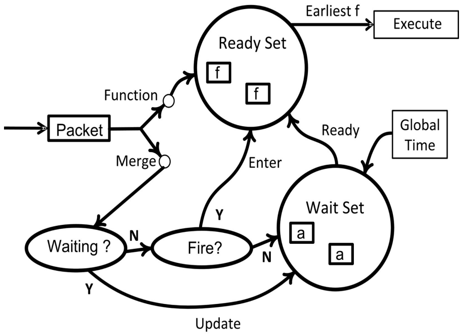
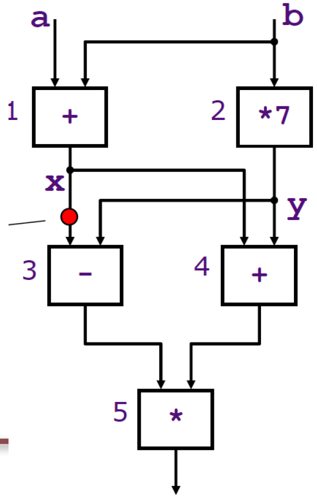
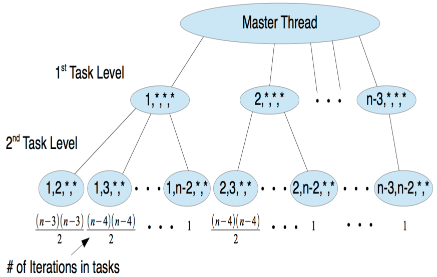


Fig. 5. DAG of QR for a 4x4 tile matrix.





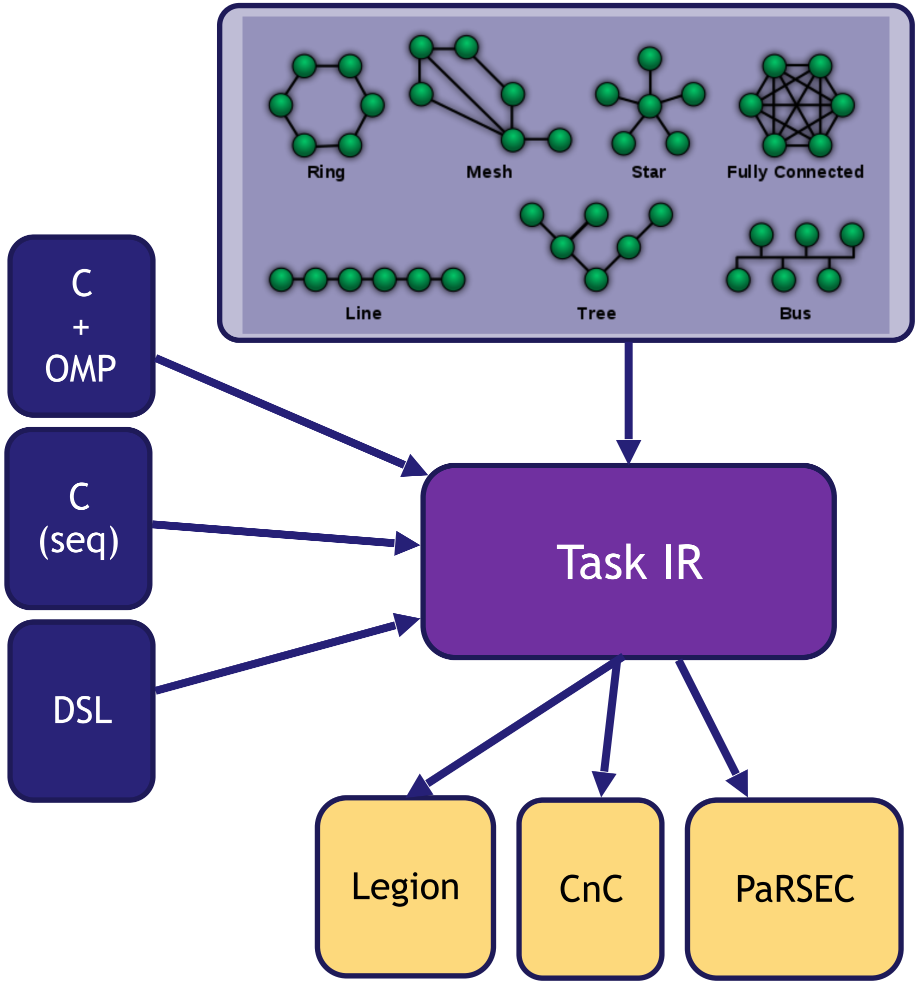


High-Performance and High-Productivity Programming With Task-Based Execution

The performance promise of task / dataflow runtimes is hampered by the lack of high-productivity systems that encourage their use

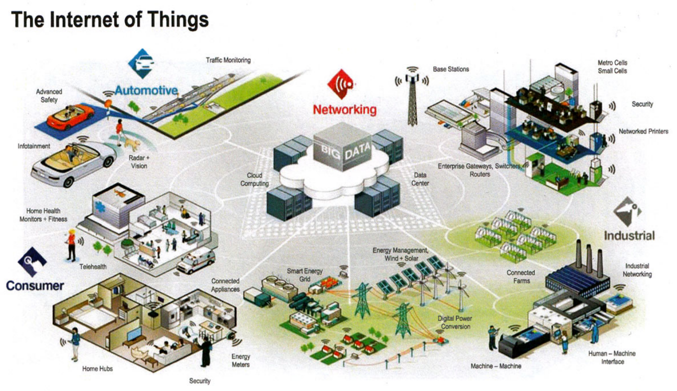
Can we design a unified system for distributed scientific applications targeting tasking / dataflow runtimes?:

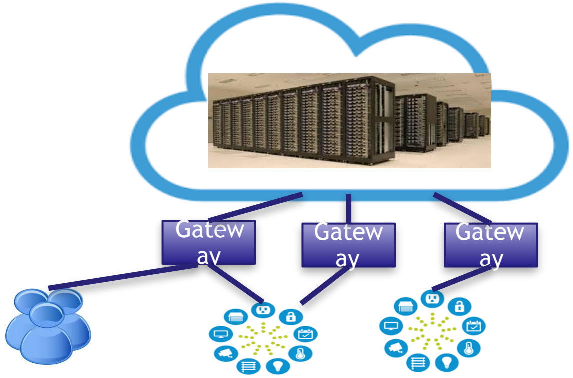
- ❑ Implement compiler transformations that optimize programs for a variety of target runtimes (Legion, Concurrent Collections, ParSEC) and platforms
- ❑ Design a common intermediate representation to allow for several input specifications: including sequential, explicitly parallel (e.g OpenMP) and domain specific languages
- ❑ Embed topological information and exploit it to find optimal task and data mappings that take locality fully into account
- ❑ Develop intra-node work and data partitioning strategies to exploit multi-GPU execution
- ❑ Can we extend this to include ML frameworks?

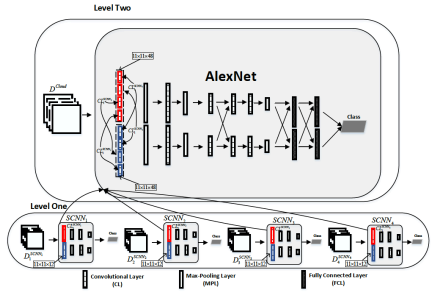


Are We There Yet?

- High Performance Computing, Cloud Computing, Edge Computing, Fog Computing
  - Computing anywhere, anytime, any devices
  - Extreme Heterogeneity, Deep Memory Hierarchy
  - Performance, Resilience, Elasticity, Productivity, Power
- Data-driven, scientific and AI code in (peaceful?) co-existence







How do we program these complex systems?

Questions?