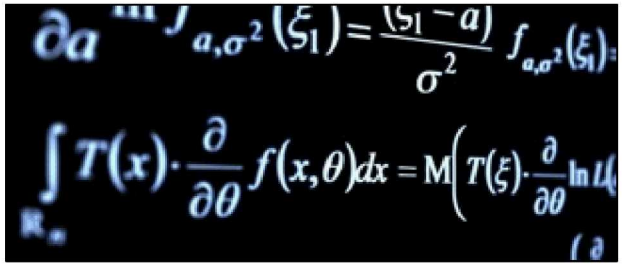
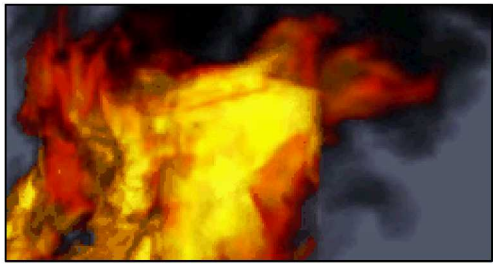
SAND2019-7116C



$$\frac{\partial}{\partial a} f_{a,\sigma^2}(\xi_1) = \frac{(\xi_1 - a)}{\sigma^2} f_{a,\sigma^2}(\xi_1)$$

$$\int T(x) \cdot \frac{\partial}{\partial \theta} f(x,\theta) dx = \mathrm{M}\left(T(\xi) \cdot \frac{\partial}{\partial \theta} \ln L \right.$$



Kokkos: Are We Prepared for Extreme Heterogeneity?

Unclassified Unlimited Release

D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang

Christian R. Trott, - Center for Computing Research

Sandia National Laboratories/NM

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

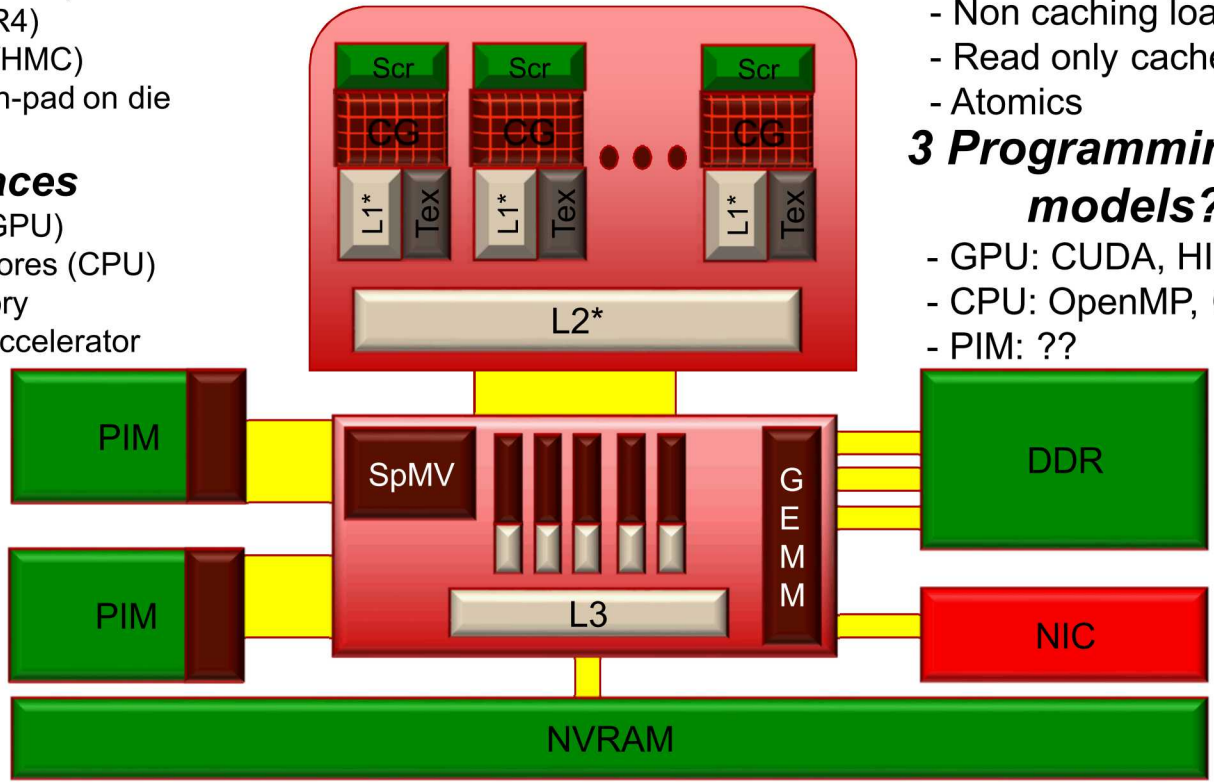
A Vision of the future

4 Memory Spaces

- Bulk non-volatile (Flash?)
- Standard DDR (DDR4)
- Fast memory (HBM/HMC)
- (Segmented) scratch-pad on die

3 Execution Spaces

- Throughput cores (GPU)
- Latency optimized cores (CPU)
- Processing in memory
- SpMV and GEMM accelerator



Special Hardware

- Non caching loads
- Read only cache
- Atomics

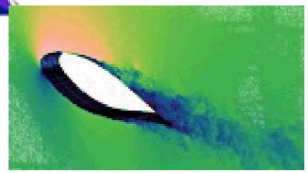
3 Programming models??

- GPU: CUDA, HIP, SyCL, OpenMP
- CPU: OpenMP, OpenACC
- PIM: ??

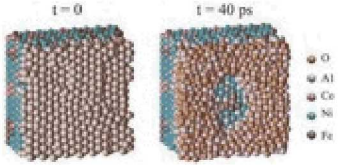
Applications

Libraries

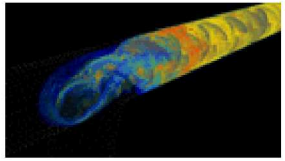
Frameworks



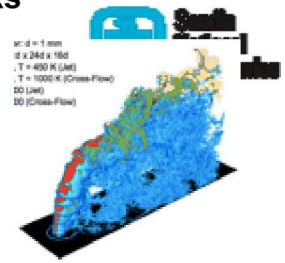
SNL NALU
Wind Turbine CFD



SNL LAMMPS
Molecular Dynamics



UT Uintah
Combustine



ORNL Raptor
Large Eddy Sim

Kokkos

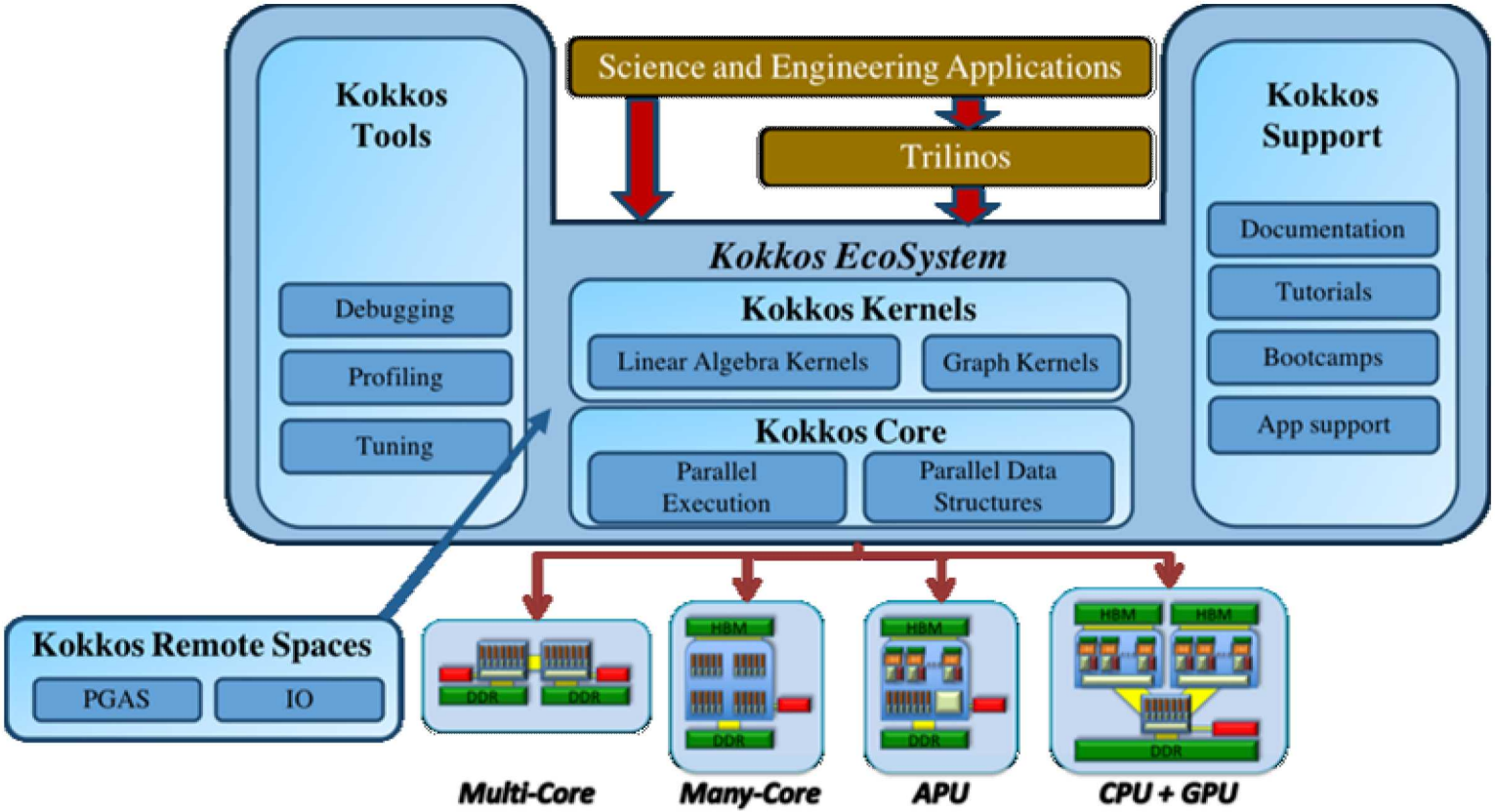
ORNL Summit
IBM Power9 / NVIDIA Volta

LANL/SNL Trinity
Intel Haswell / Intel KNL

ANL Aurora
Intel Xeon CPUs + Intel Xe Compute

SNL Astra
ARM Architecture

Kokkos EcoSystem

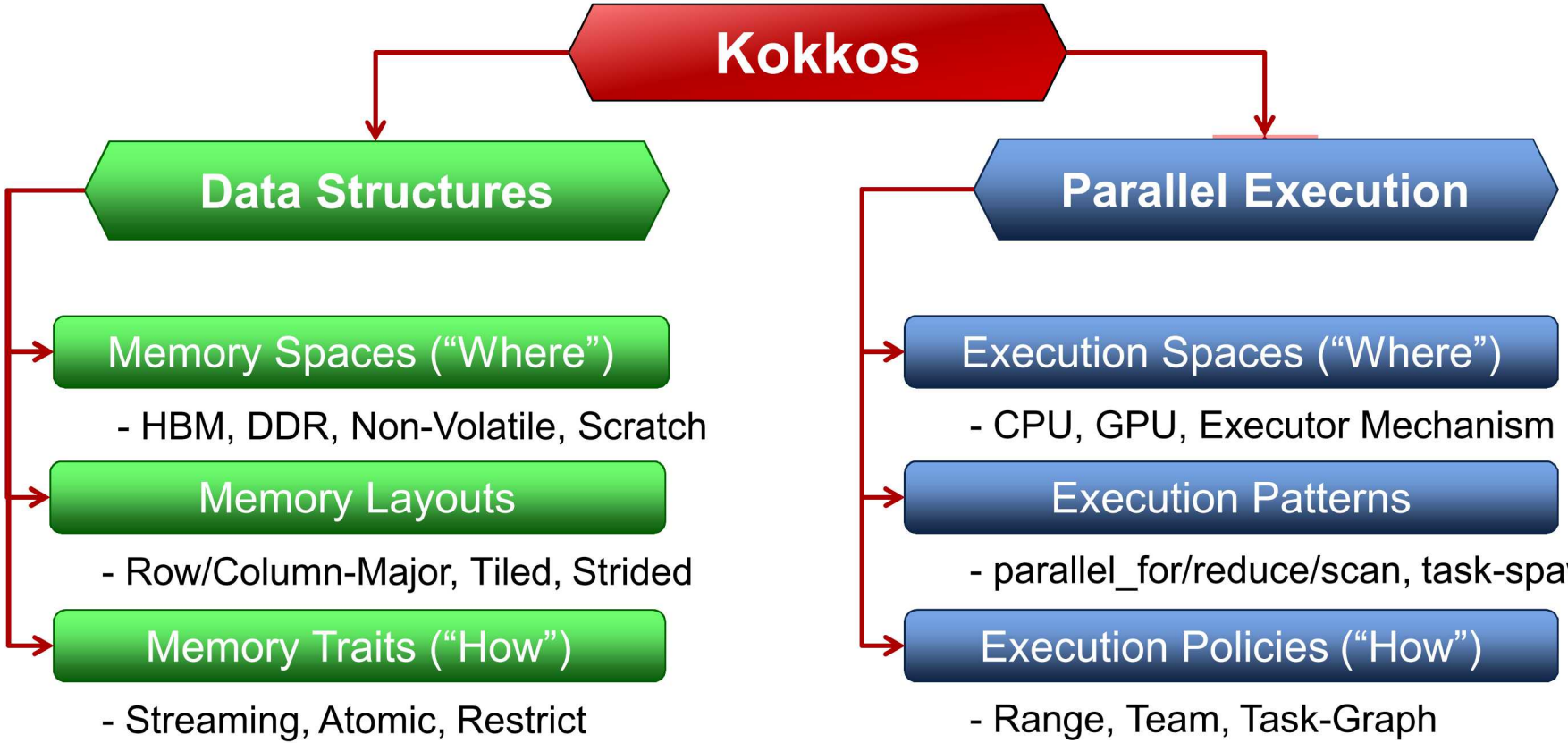


Kokkos Development Team

- Dedicated team with a number of staff working most of their time on Kokkos
  - Main development team at Sandia in CCR

Kokkos Core:	***C.R. Trott,*** *D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang, Mikael Simberg, H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin*
	former: ***H.C. Edwards****, D. Labreche, G. Mackey, S. Bova*
Kokkos Kernels:	***S. Rajamanickam,*** *N. Ellingwood, K. Kim, C.R. Trott, V. Dang, L. Berger, J. Wilke, W. McLendon*
Kokkos Tools:	***S. Hammond,*** *C.R. Trott, D. Ibanez, S. Moore; soon:* ***D. Poliakoff***
Kokkos Support:	***C.R. Trott,*** *G. Shipman, G. Lopez, G. Womeldorff,*
	former: ***H.C. Edwards****, D. Labreche, Fernanda Foertter*

Kokkos Core Abstractions



Kokkos Core Capabilities

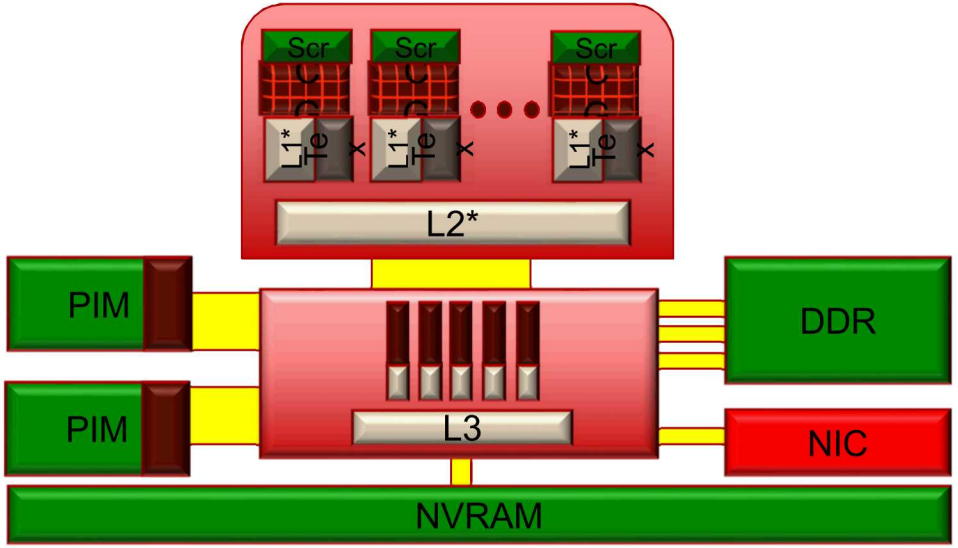
Concept	Example
Parallel Loops	**parallel_for**(N, **KOKKOS_LAMBDA** (int i) { ...BODY… });
Parallel Reduction	**parallel_reduce**(**RangePolicy**<ExecSpace>(0,N), **KOKKOS_LAMBDA** (int i, double& upd) { …BODY... upd += ... }, Sum<>(result));
Tightly Nested Loops	**parallel_for**(**MDRangePolicy**<**Rank**<3> > ({0,0,0},{N1,N2,N3},{T1,T2,T3}, **KOKKOS_LAMBDA** (int i, int j, int k) {…BODY...});
Non-Tightly Nested Loops	**parallel_for**(**TeamPolicy**<**Schedule**<**Dynamic**>>(N, TS), **KOKKOS_LAMBDA** (Team team) { … COMMON CODE 1 ... **parallel_for**(**TeamThreadRange**(team, M(N)), [&] (int j) { ... INNER BODY... }); … COMMON CODE 2 ... });
Task Dag	**task_spawn**(**TaskTeam**(scheduler , priority), **KOKKOS_LAMBDA** (Team team) { … BODY });
Data Allocation	**View**<double**, Layout, MemSpace> a(“A”,N,M);
Data Transfer	**deep_copy**(a,b);
Atomics	atomic_add(&a[i],5.0); View<double*,MemoryTraits<AtomicAccess>> a(); a(i)+=5.0;
Exec Spaces	Serial, Threads, OpenMP, Cuda, HPX (experimental), ROCm (experimental)

Dealing with Exectuion

```
for(int i=0; i<N; i++) {
  /* Loop Body */
}
```

How to decide where to execute?

How to express that?



Don't. *Explicitly.* *Describe.* *Compilers.*

Dealing with Execution

- Default Parallel Loop

```
parallel_for("MyLoop", N,
  KOKKOS_LAMBDA (const int i) { /* Loop Body */ });
```

- Requesting a latency or throughput optimized execution space

```
parallel_for("MyLoop", RangePolicy<LatencyOptimizedExecSpace>(0,N),
  KOKKOS_LAMBDA (const int i) { /* Loop Body */ });
```

- Instead of explicitly requesting, describe what the algorithm needs
  - Providing hints which are statically mapped to exec spaces.

```
parallel_for("MyLoop", RangePolicy<>(0,N).require(Hints(BandwidthLimited,NonTemporal)),
  KOKKOS_LAMBDA (const int i) { /* Loop Body */ });
```

- Let the compiler figure out what the most optimal core is to put this.
  - The compiler for example knows flops to bytes ratio (though not caching)!

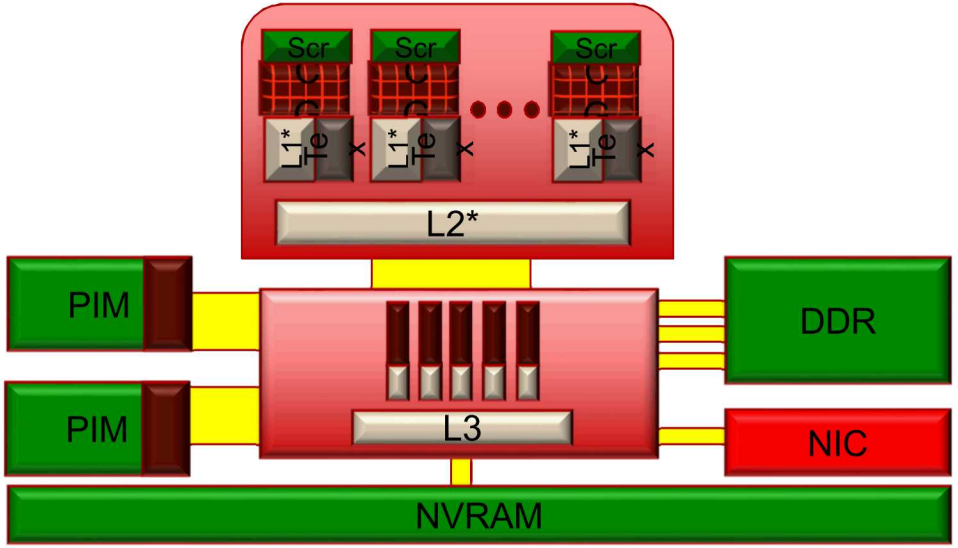
```
parallel_for("MyLoop", RangePolicy<AUTO>(0,N),
  KOKKOS_LAMBDA (const int i) { /* Loop Body */ });
```

Dealing with Memory

```
double** A = new double[N*M];
double* x = new double[M];
```

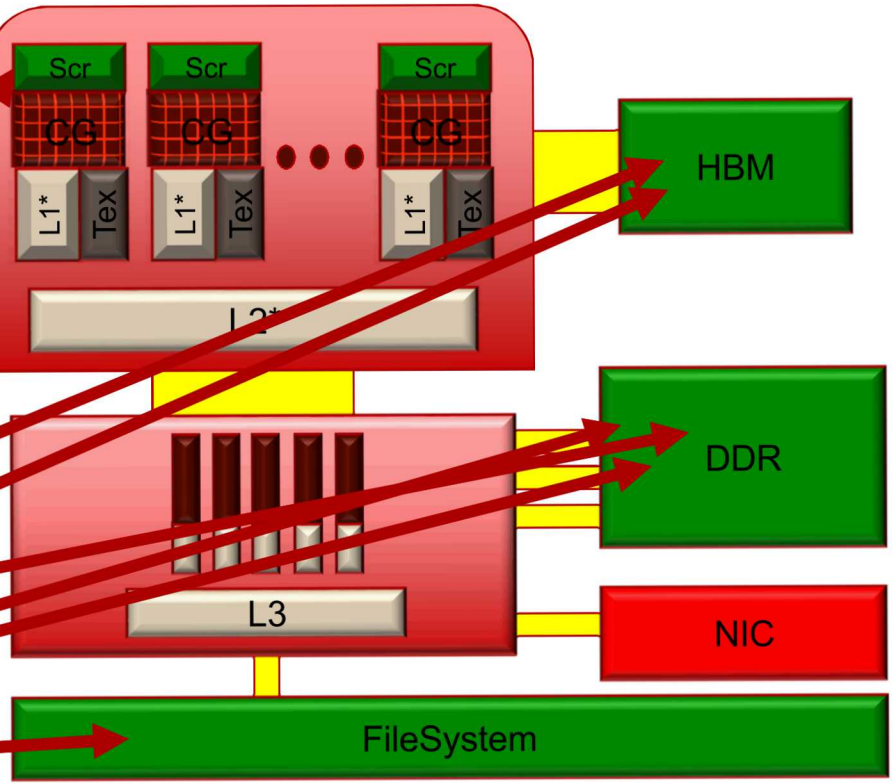
Where to allocate?

How to Layout?



MemorySpaces: More than Storage

- Who can access it?
- Page migratable?
- Persistence Scope?
- Bandwidth/Latency?




```
View<double**,Cuda::scratch_memory_space>
View<double**,CudaSpace>
View<double**,CudaUVMSpace>
View<double**,CudaHostPinnedSpace>
View<double**,HostSpace>
View<double**,HDF5Space>
```

Doing Data Layout

- GEMV parallelized trivially over rows:

```
parallel_for("GEMV",num_rows, KOKKOS_LAMBDA (int i) {
  double ysum = 0.0;
  for(int j=0; j<num_cols; j++)
    ysum += A(i,j) * x(j);
  y(i) = ysum;
}
```

- How to store A?
  - GPUs: Column-Major

```
View<double**,LayoutLeft> A("A",N,M);
```

  - CPUs: Row-Major

```
View<double**,LayoutRight> A("A",N,M);
```

How To Expose Special Function Units?

Libraries!

- Easy to use for applications
- Connect with memory info
  - Is the data accessible and the correct layout?
- KokkosKernels has interface with all necessary information
  - Matrix in main GPU memory
  - RHS vector created on the fly in scratch memory
  - LHS vector in Host accessible memory

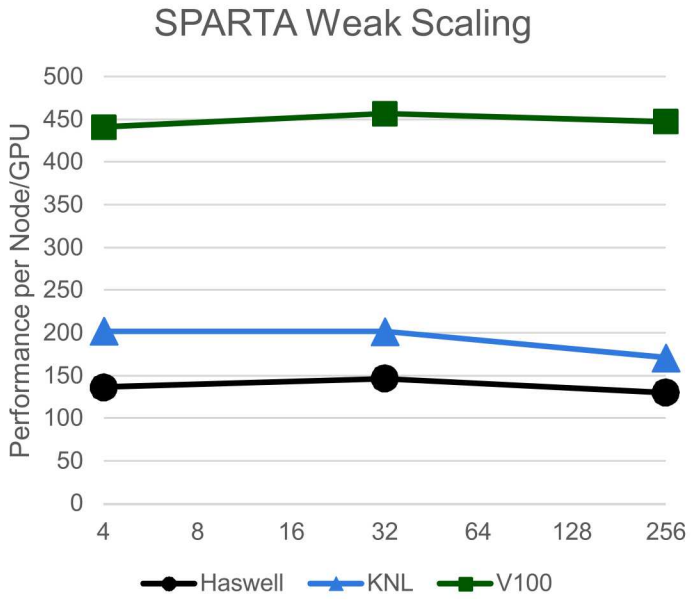
```
View<double**,CudaSpace> A = /*...*/;
View<double*,CudaHostPinnedSpace> y = /*...*/;
View<double*,Cuda::scratch_memory_space> x = /*...*/;
gemv(y,A,x); /* Execute in Cuda Space since it can access all data. */
```

Key Things to Help Compilers/Runtimes

- Encode information at compile time (as part of the type system)
  - Where does data life.
  - How do you access it.
  - Properties of algorithms.
- Be descriptive – not prescriptive
  - Say what you want to happen and give properties (see above)
  - Let the compiler/runtime figure out how to use that info
- Provide graceful fallbacks and defaults
- Make it possible to provide incrementally more information

Sparta: Production Simulation at Scale

- **S**tochastic **PA**rallel **R**arefied-gas **T**ime-accurate **A**nalyzer
- A direct simulation Monte Carlo code
- Developers: *Steve Plimpton, Stan Moore, Michael Gallis*
- Only code to have run on all of Trinity
  - 3 Trillion particle simulation using both HSW and KNL partition in a single MPI run
- Benchmarked on 16k GPUs on Sierra
  - Production runs now at 5k GPUs
- Co-Designed Kokkos::ScatterView



That's Great But I Don't Trust TPLs

- Good News! We are working on contributing to the C++ standard!
- Executors for heterogeneous environements
  - Control where and how stuff executes
  - Property mechanism to provide more information
  - Hierarchical executors for supporting hierarchical hardware
- MDSpan for multi-dimensional arrays with accessors
  - Templated on scalar, extents, layout and accessor

```
basic_mdspan<double,extents<dynamic_extent,8>,layout_left,basic_accessor<double>>
```

  - Extent accessors to provide typesafe info about storage place

```
basic_mdspan<double,extents<8,4>,layout_right,memspace_accessor<double,HBM>>
```

- BLAS support in the works: point to get SpMV or GEMM accelerator support