RAJA Portability Suite Update

P3HPC Virtual Forum

September 1-2, 2020

EXASCALE COMPUTING PROJECT

Rich Hornung, LLNL, with contributions from many others

LLNL-PRES-813321

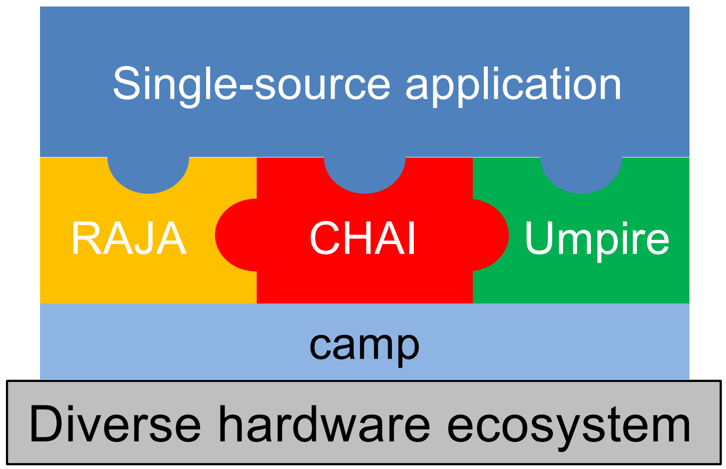
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Our open source software tools enable applications to run on HPC systems in a performance portable way

RAJA: C++ kernel execution abstractions

- Enable single-source application source code insulated from hardware and programming model details



Umpire: Memory management

- High performance memory operations, such as pool allocations, with native C++, C, Fortran APIs

CHAI: C++ array abstractions

- Automate data copies giving look and feel of unified memory

https://github.com/LLNL/RAJA

https://github.com/LLNL/CHAI

https://github.com/LLNL/Umpire

https://github.com/LLNL/camp

camp: C++ metaprogramming facilities

- Focused on HPC compiler compatibility

Applications rely on these tools to run on current systems and evolve to future platforms (Frontier, Aurora, El Capitan, …)

- LLNL WSC production codes use the tools in a variety of combinations
  - Applications: multiple rad-hydro, deterministic & MC transport, ICF, engineering
  - Support libraries: EOS, TN burn, HE chemistry, sliding contact, AMR, mesh-to-mesh linking
- Other LLNL program apps: NIF VBL, magnetic confined plasma simulations, …
- The tools are part of the ECP ST ecosystem used in ECP apps and libraries, such as
  - SW4 (AD-EQSIM)
  - GEOS (AD-Subsurface)
  - ExaSGD (AD)
  - LLNL ATDM
  - SUNDIALS, DevilRay (ST-Alpine), MFEM (CEED co-design)

LLNL institutionally-funded RADIUSS effort facilitates adoption across the lab.

Notable RAJA accomplishments since last year's meeting

- 5 releases on GitHub
- New released features include:
  - Initial **HIP back-end** support for all RAJA features (AMD GPUs)
  - Initial support for **asynchronous kernel execution** (also works with Umpire and CHAI)
  - Work groups (**fuse many small GPU kernels** into one kernel launch)
  - A "Multi-view" abstraction (enabling multiple arrays to share indexing arithmetic)
  - Multiple **sort algorithms**
  - Expanded GPU capabilities and performance improvements
    - Block-direct, thread, warp, bitmask execution policies, atomic local array type for atomics in GPU shared mem
  - **Dynamic "plug-in" support** and integration with Kokkos performance tools (J. Hynes – summer)
- Notable external engagements
  - Tutorials at ATPESC 2019, ECP Annual Meeting 2020; ExaSGD, SW4 Hack-a-thons
  - Created DESUL (DoE Standard Utility Library) org and repo (https://github.com/desul/desul)
    - Collaboration with Sandia and Oak Ridge, eventually other C++ projects
  - Working with Marvell on RAJA Perf Suite optimization for ARM processors

New RAJA support for GPU streams enables asynchronous execution (M. Davis, T. Scogland, D. Beckingsale)

```
chai::ManagedArray<double> a1(N);   chai::ManagedArray<double> a2(N);

RAJA::resource::Cuda cuda1;
RAJA::resource::Cuda cuda2;

auto event1 = forall<cuda_exec_async>(&cuda1, RangeSegment(0, N),
                [=] RAJA_DEVICE (int i) { a1[i] = ... } );

auto event2 = forall<cuda_exec_async>(&cuda2, RangeSegment(0, N),
                [=] RAJA_DEVICE (int i) { a2[i] = ... } );

cuda1.wait_on(&event2);   // or event2.wait();

forall<cuda_exec_async>(&cuda1, RangeSegment(0, N),
  [=] RAJA_DEVICE (int i) { a1[i] *= a2[i]; } );

forall<seq_exec>(RangeSegment(0, N),
  [=] (int i) { printf("a1[%d] = %f \n", i, a1[i]); } );
```

Resource objects passed to RAJA execution methods.

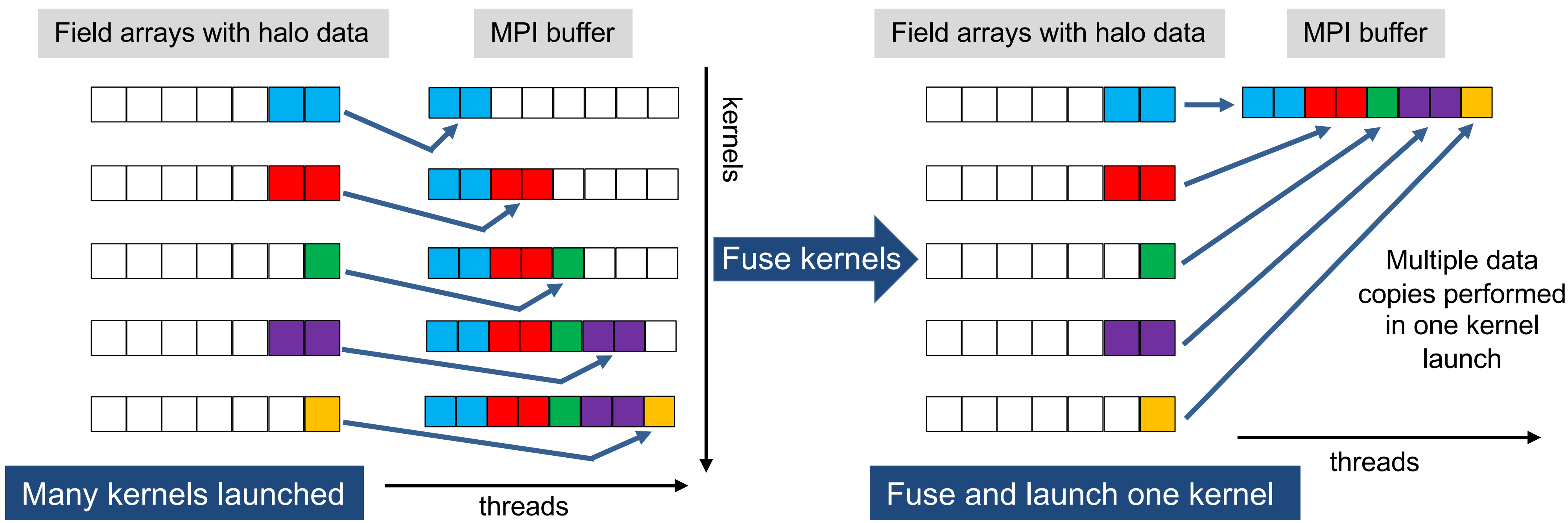
RAJA execution methods return event objects that can be queried or waited on.

CHAI arrays know which resources are using them so host-device data transfers can happen as soon as data is ready.

Plan to explore similar extensions to other programming model back-ends.

RAJA "work groups" enable many small kernels to be packed into one GPU launch to reduce overhead (J. Burmark, P. Robinson)

Key use case: Packing/unpacking halo data for MPI comm**



****In 2 production apps, this yields 5-15% overall performance boost**

Notable Umpire accomplishments since last year’s meeting

- 6 releases on GitHub
- New released features include:
  - Support for **HIP back-end** (AMD GPUs)
  - Support for **OpenMP target back-end** (alternative mechanism for managing data)
  - Support for **OneAPI back-end** (SYCL support for Intel GPUs)
  - **Zero-byte allocations** supported as a native concept (track allocations to specific allocator)
  - Asynchronous copy and memset operations
    - Works with RAJA and CHAI to overlap data transfer and compute operations
  - Improved “replay” support
    - Binary capability to handle large (100G+) replay files
    - Additional memory operations (copy, etc.) are now recorded also
  - Added **multi-device support** for CUDA, HIP, and OpenMP target

Umpire accomplishments, ctd…

- New released features include:
  - **Backtrace support** (e.g., track allocations in app codes)
  - Additional memory operations, such as prefetch for NVIDIA GPUs
  - **NV memory file allocation support** (A. Perez – summer)
- Notable external engagements
  - Tutorial at ECP Annual Meeting 2020; ExaSGD, SW4 Hack-a-thons

Notable CHAI accomplishments in the past year

- 3 releases on GitHub
- New released features include:
  - Support for **AMD HIP** programming model (AMD GPUs)
  - Use of Umpire's fast Judy array-based map implementation to store pointer records
  - Transition to unified logging across Umpire and CHAI
  - Adoption of **RAJA "plugin" mechanism** which allows CHAI to automatically integrate with RAJA when libraries are built together
  - "Managed pointer" simplifies use of **virtual class hierarchies** across memory spaces
  - **Eviction capability** is easy to use and enables integrated apps to run larger problems by avoiding pool fragmentation

CHAI ‘managed_ptr’ solution simplifies use of virtual class hierarchies across host/device memories (P. Robinson, A. Dayton)

- New CHAI ‘managed_ptr’ enables such code to be run on GPUs without a major refactor

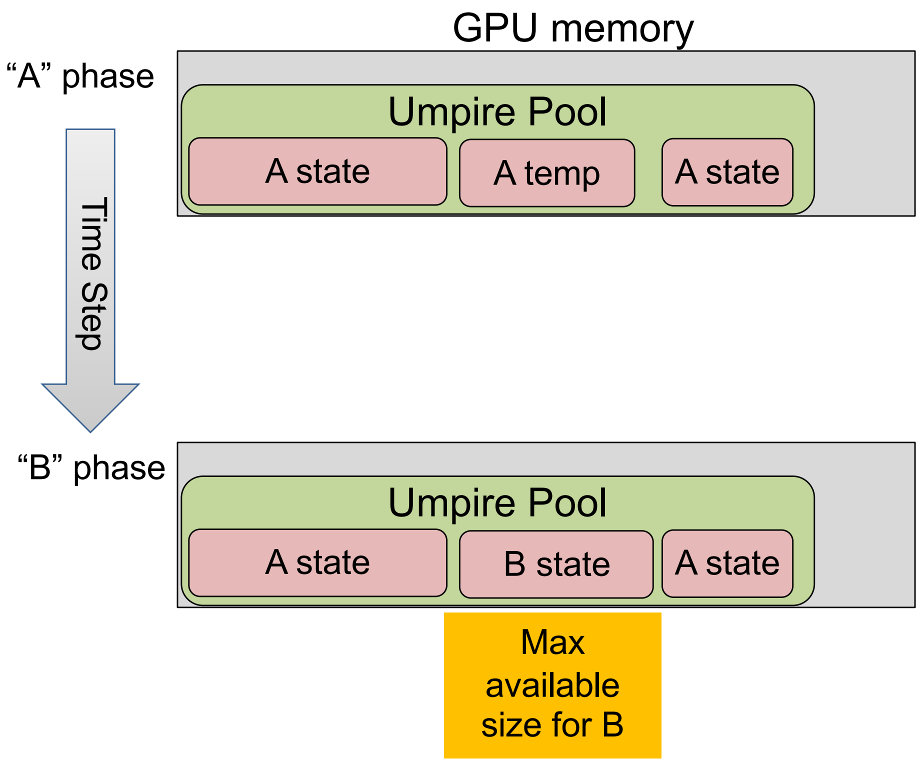
```
void overlay( Shape* shape, double* mesh_data ) {
  chai::managed_ptr< Shape > mgd_shape = shape->makeManaged();
  RAJA::forall< cuda_exec > ( ...  {
    mgd_shape->processData(mesh_data[i]);
  });
  mgd_shape.free();
}
```

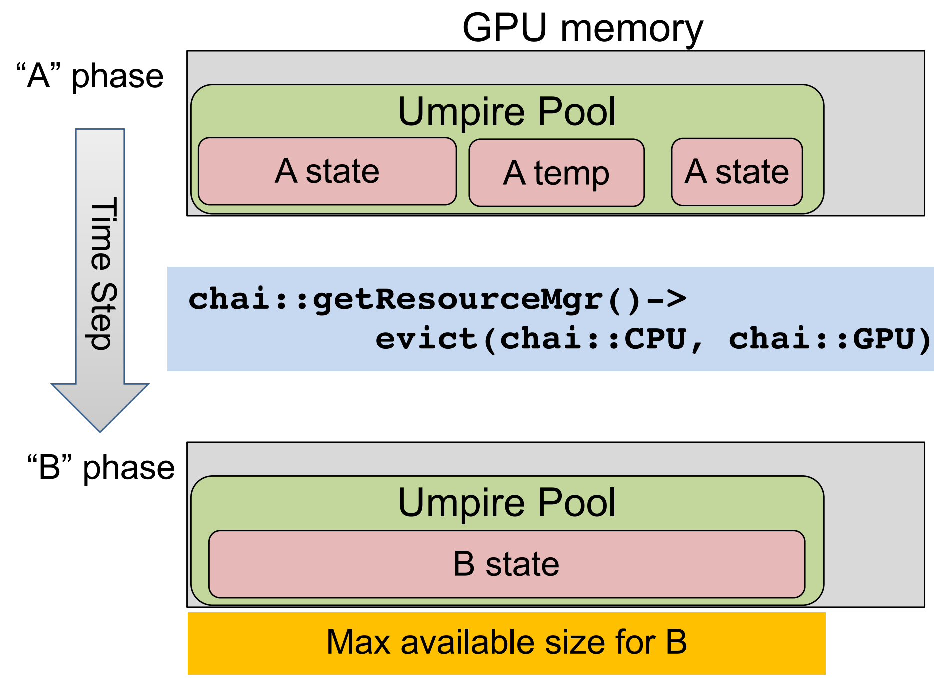
- This requires methods to clone objects and host-device decorations on constructors

```
chai::managed_ptr< Shape > Sphere::makeManaged( ) { ... }

__host__ __device__ Sphere::Sphere( ... ) { ... }
```

CHAI eviction capability yields significant memory and execution time benefits for multiphysics apps (P. Robinson, A. Dayton)






```
chai::getResourceMgr()->
        evict(chai::CPU, chai::GPU);
```

Other features in development

RAJA vector interface can encapsulate vector intrinsics so code will SIMD-ize transparently (A. Kunen)

```
using Vec_type = RAJA::StreamVector<double, 2>;
using Vecidx_type = RAJA::VectorIndex<int, Vec_type>;

RAJA::kernel< KernelPolicy<...vector_exec<...>...> >( segments,
  [=] (int m, int d, int g, Vecidx_type z) {
     phi(m, g, z) += L(m, d) * psi(d, g, z);
});
```

registers

Vectorize operations along ‘z’ index

L loads scalar

psi loads vector in z

Bcast scalar to vector

phi loads vector in z

SIMD multiply

SIMD add

phi stores vector in z

RAJA can automatically fuse these into a single FMA instruction

RAJA "Teams" prototype being developed in collaboration with the MFEM team (A. Vargas)

- Potentially more flexible than RAJA::kernel interface
  - Simpler run time policy selection (e.g., CPU or GPU)
  - New nested loop patterns
- Opens up hierarchical parallelism opportunities
- Aligns well with MFEM algorithm structures macro layers

"Upper triangular" loop pattern example

```
int N = ...;
launch<launch_policy>(select_cpu_or_gpu,
                      Resources(Teams(N), Threads(N)),
  [=] RAJA_HOST_DEVICE(LaunchContext ctx) {
    loop<teams_pol>(ctx, RangeSegment(0, N), [&](int r) {

      loop<threads_pol>(ctx, RangeSegment(r, N), [&](int c) {
        M(r, c) = ...;
      });

      loop<threads_pol>( ... );

    });  // teams loop (r)
  }   // outer lambda
);  // launch
```

Our Argonne ECP collaborators have made substantial progress toward a RAJA SYCL back-end (B. Homerding, et al.)

- ECP SW4 app and much of RAJA Perf Suite working
- Performance looks promising
- Working through memory management issues
- Work remains to support all RAJA features and resolve performance issues

Mean Runtime Report (sec.)

Kernel	Base_SYCL	RAJA_SYCL
Basic_DAXPY	0.263559	0.247529
Basic_IF_QUAD	0.192065	0.204579
Basic_INIT3	0.407615	0.371487
Basic_MULADDSUB	0.284180	0.261049
Basic_NESTED_INIT	0.468795	0.491765
Lcals_DIFF_PREDICT	0.495185	0.488606
Lcals_EOS	0.382110	0.458804
Lcals_FIRST_DIFF	0.865176	0.796497
Lcals_GEN_LIN_RECUR	0.716076	0.658505
Lcals_HYDRO_1D	0.688161	0.627595
Lcals_HYDRO_2D	1.251182	1.122538
Lcals_INT_PREDICT	0.580458	0.627150
Lcals_PLANCKIAN	0.044052	0.050743
Lcals_TRIDIAG_ELIM	0.584795	0.670069
Apps_DEL_DOT_VEC_2D	0.138256	0.201736
Apps_ENERGY	0.664540	0.599901
Apps_FIR	0.278217	0.413339
Apps_PRESSURE	0.877004	0.827753
Apps_VOL3D	0.110188	0.194256

Acknowledgements

- RAJA
  - Rich Hornung (PL)
  - David Beckingsale
  - Jason Burmark
  - Noel Chalmers (AMD)
  - Robert Chen
  - Mike Davis
  - Jeff Hammond (Intel)
  - Brian Homerding (ANL)
  - Holger Jones
  - Will Killian (Millersville U.)
  - Adam Kunen
  - Olga Pearce
  - Tom Scogland
  - Arturo Vargas
- Umpire
  - David Beckingsale (PL)
  - Noel Chalmers (AMD)
  - Johann Dahm (IBM)
  - Mike Davis
  - Marty McFadden
- CHAI
  - David Beckingsale (PL)
  - Alan Dayton
  - Adam Kunen
  - Peter Robinson
- camp
  - Tom Scogland (PL)
  - Mike Davis
  - Adam Kunen
  - David Beckingsale

Lawrence Livermore National Laboratory