An Overview of the RAJA Portability Suite

(the slides are available under “Presentation Materials” in the above URL)
Date: March 10, 2021
Presented by: Arturo Vargas (Lawrence Livermore National Laboratory)

Q. How does it work for nested loops?

A. RAJA has two interfaces for this that will be described in the talk, RAJA::kernel and RAJA::launch (or Teams). Please see the presentation for code examples.

Q. Is there a concept of managed memory in Umpire?

A. Yes, managed memory is a memory type supported by Umpire.

Q. GIven that GPGPU memory is typically constrained - is there any support for async memory movement and scheduling of these moves independently from the main stream? Also how about async alloc/dealloc?

A. Yes, RAJA provides support for scheduling and coordinating kernel execution and memory operations in different GPU streams. There is an example of this shown in the slide presentation.

Q. On slide 27, how is the RAJA execution time shorter than CUDA?

A. “Don’t have a good explanation, but it was a nice surprise.” We have an extensive and growing suite of kernels that we use as a tool to interact with vendors and study RAJA performance (see reference to RAJA Performance Suite in the last slide). Depending on the compiler and version and execution back-end, RAJA kernel execution can be faster or slower than baseline variants of the same kernel. Since most compiler optimizations are based on internal heuristics, it’s difficult to give a solid explanation for this sort of thing.

Q. Is this some sort of MPI Reduce method (map reduce) applied on loops and abstracted in API?

A. RAJA is independent of MPI and is designed to execute kernels within a single MPI rank, which is how it is used in practice. Our GPU (CUDA and HIP) reduction classes have been tuned for those architectures and perform pretty well. We are prototyping a new reduction interface that integrates much better with OpenMP and performs better than our current reduction capability, which cannot use OpenMP reduction machinery directly. We are also looking into support for new reduction patterns, such as arrays of reduction variables, use supplied comparators, etc.

Q. Slide 42: Did any of the projects that adopted RAJA have nice examples of initial planning documents?

A. For ECP application projects, you would have to ask them. For ASC programmatic codes at LLNL, extensive detailed planning for porting was performed and documented. Unfortunately, most of that information is sensitive and cannot be shared.

Q. Are there any examples of sparse iterative solvers with RAJA? Particularly interested in AMG-preconditioned krylov methods.

A. The hypre preconditioner library developed at LLNL uses RAJA internally for some things. You would have to ask a hypre developer for details.

Q. Any BLAS-like primitives for sparse systems in RAJA?

A. We don’t have blas-like primitives in RAJA. We’ve discussed building up a library of similar methods and others as a supplement to RAJA. However, we don’t have the resources to develop and support it appropriately. Also, none of our users have requested it.

As a side note, we are prototyping matrix/vector primitives to exploit SIMD optimization and special GPU hardware (e.g., DGEMM), but it’s experimental at this point.

Q. Is Raja purely a library or is it also a customized compiler? I’m asking because the loop body remains the same even with different execution policies. I’m wondering if this is supported by the compiler rather than the library.

A. RAJA is a (mostly) header-only library. To use it, you only need a C++11 compliant compiler (soon we will require C++14) that also provides support for the execution back-end you wish to use; e.g. CUDA, HIP, OpenMP, etc.

Q. Does Raja support applications spread across multiple nodes using programming models like MPI?

A. RAJA is a single-node programming model, independent of MPI. That said, all RAJA applications use it with an MPI distributed memory parallelization model.

Q. is the outer lambda using the CUDA extended lambda capability or does it do something of its own - how does it map to HIP?

A. Yes, we use the CUDA extended lambda capability. Same as Kokkos. RAJA maps to HIP the same way it maps to CUDA. In fact, RAJA execution policies for HIP and CUDA are essentially the same. Please see the RAJA User Guide referenced on the last presentation slide for details.

Q. Does RAJA provide wrappers (like Kokkos-kernels) for vendor-provided sparse libraries like cuSparse?

A. Please see the answer about blas-like routines above. We would consider it if it is requested and there is sufficient interest. RAJA has a smaller team than Kokkos and we focus almost exclusively on the high-priority needs of our users.

Q. is there any support for the kernel launch latency hiding - wrapping the cudagraphs type APIs?

A. The kernel fusion approach shown in the slides is something we provide for reducing launch overhead for small kernels.

Q. Is the reason why RAJA is portable because it provides an abstraction for shared memory management? For the sake of argument: How can a code written for Nvidia A100 which has 80 Gb of memory be run on a lower level GPU with, say, only 12 Gb of memory using RAJA?

A. RAJA will not handle those sorts of hardware differences for you automagically. That said, you can encapsulate some of those things in RAJA execution policies. For example, you can define a hardware-dependent size of a shared memory block and use it along with loop tiling to keep your data in shared memory while it is being used. Putting such policies in a header file and choosing which one to use at compile-time is a common approach to this for RAJA users.

We want users to be explicit about how they use RAJA. We don't want users to be unaware of things we would do under the covers to avoid producing unexpected results.

Q. Can I provide my own malloc/free routines to Umpire?

A. I believe umpire allocators can be specialized for custom alloc/dealloc routines, but I would have to verify this.

Q. Is Umpire a standalone library I can use without RAJA?

A. Yes, Umpire can be used standalone. We have Fortran apps using Umpire, which can't use RAJA directly, for example.

Q. is the default CUDA backend compiler nvcc or does the framework also support the CLANG PTX?

A. The RAJA CUDA back-end will work with any compiler that can compile CUDA code. We regularly test with the clang-cuda compiler, but it causes runtime issues with the way we use shared memory in our CUDA reduction objects.

Q. Is there any support for load balancing between CPU and GPU - such that all the resources remain fully loaded? I was just quietly hoping somebody solved that for me.

A. This is a difficult application-specific problem. We've explored it in the context of some real applications, but saw little benefit. Memory coordination is especially difficult. If you figure it out, please let us know.

One place we have found some promise is for patch-based structured adaptive mesh refinement (SAMR). Small patches run on CPU, big patches run on GPU. Target the workload to the compute resource it is best suited to.

Q. RAJA works with any versions of CUDA?

A. Any version of CUDA 9.2 and later

Q. Are there any recommendations for tools that measure performance that integrate with CI? Would something like TAU/HPCToolkit work?

A. Large applications at LLNL, whether they use RAJA or not, tend to instrument their code with Caliper (available on GitHub) and track performance on a commit-by-commit basis using the SPOT web browser based tool, which supports a lot of interesting analysis.

Q. Aurora is an Intel machine. Intel already has oneAPI. What are the advantages of using RAJA and Chai in that case?

A. Not intending to be glib, but do you trust your code to work on all platforms well based on a compiler from one vendor, especially those from a different vendor? If you think it will work for you, go for it. Personally, my career is littered with situations where I've needed options to work around compiler deficiencies.

Q. Are there any major differences between your approach and what Kokkos do? What's different here between other approaches, e.g. Kokkos?

A. RAJA and Kokkos are attacking a similar problem and are similar in spirit. In fact, we work together on internal implementation features that we can share in common. However, there are notable differences in APIs, features supported, etc. This is mainly due to the fact that RAJA and Kokkos grew out of the needs of different user bases.

One major difference is that Kokkos has a memory model and array data structure support within it. RAJA does not. This was based on the fact that RAJA users prefer to define their own memory management. The RAJA team supports things like Umpire and CHAI, described in this talk, for memory management, and Views and Layouts that allow users to wrap their own data allocations and modify access patterns without changing their application source code.

Another major difference is that RAJA requires users to be more explicit about their choice of execution policies for specific hardware. Our users prefer this for performance tuning. So, for example, RAJA gives fine-grained control over GPU block-thread mapping.

Q. Does RAJA have a slack channel?

A. RAJA has a slack channel, but it is used little. This is mainly due to the fact that Slack is not a “blessed” chat application at LLNL.

Q. With the abstraction layer as Raja, are you running into compiler limitations in - specifically with NVCC where the extended lambda is very heavy on the symbol generation?

A. No. Not really. We can match native CUDA performance in most cases. This wasn’t automatic and we worked closely with NVIDIA during the Sierra procurement to address issues. We work with all major vendors to address compiler support issues as we uncover them.