

RAJA and multi-physics

(A) How are your application development teams structured?

Software engineers, code physicists, and end-users are typically co-located. When everyone sits together, it is easier to understand common needs. It is also easier to pester someone directly when a solution to a problem is needed; potentially reducing the number of software bugs and helping to ensure features are in line with actual needs rather than imagined needs.

(B) Provide general data about your applications.

Multi-physics applications are typically $O(1,000,000)$ lines of source code, with additional code found in $O(10)$ external libraries. Multi-physics codes support tens of physics domains (hydrodynamics, thermal, chemistry, etc.) and are thus very diverse. For physics that tends to react on fast time scales (e.g. chemistry), an explicit solution technique is required. For operations that happen over long time scales (e.g. thermal conduction), an implicit solution technique is often used, i.e. linear algebra “solver” libraries). For key applications of interest to LLNL, a fast time scale is typically required.

An application can run on a wide range of resources, from a single Celeron laptop processor to hundreds of thousands of very fast cores, as the application requires. Codes such as ALE3D that are typically used by DOD customers can run across all computing scales. Thousands of processors are typically used in a typical run, meaning that supercomputers are commonly needed for typical applications of interest.

There is not a lot of room for prototyping with codes at this scale. That said, some codes use a steering language such as python to drive high-level computation decisions.

(C/D) What abstractions are expressed in your applications? How do you exploit them for portability?

In mesh based physics codes, the majority of:

Computational Work occurs in for-loops

Data Access occurs through arrays

By encapsulating these two concepts, we can map the majority of all computation to a diverse set of hardware architectures.

In multi-physics applications, data and algorithm are tightly bound. Each loop iteration of an algorithm accesses a slice of memory locations. Subsequent loop iterations cause access to subsequent slices of memory. The memory slices in adjacent iterations are typically adjacent in memory. Because of this, in mesh-based science codes, algorithm and data are usually tightly bound. The concept of moving task to data rarely has an opportunity to arise in a multi-physics application, unlike Big Data applications.

There are typically $O(10,000)$ for-loops in a mesh-based multi-physics code, but there are only $O(10)$ iteration patterns that arise. Likewise, there are $O(10)$ mesh subsetting conditions that commonly occur

(e.g. internal mesh entities vs boundary mesh entities) and specific interactions of mesh-subsets also have common patterns. By encapsulating those iteration patterns in a centralized location or with common idioms, it is possible to gain more control over architectural portability. A software engineer acting as an architecture tuner can port $O(10)$ iteration pattern implementations in isolation, allowing those patterns to be leveraged across all $O(10,000)$ loops in the code with minimal impact to code physicists.

(E) Is code re-use (not) possible when exploiting portability across diverse architecture?

We have been exploring RAJA as a portability layer as described in (C/D). So far, essentially all legacy code is reused rather than rewritten. At code sites where we want to use RAJA to control transient fault tolerance, algorithms must be idempotent which can sometimes require small algorithm changes. Idempotence transformations can increase memory movement.

(F) What success have you had at portability, and did you have to change algorithm implementations?

Inter-process communication and file-system interaction still dominates the performance of supercomputer- class applications, and will continue to do so going forward. No matter how efficient on-node algorithms become, computation will not be the resource bottleneck for supercomputer scalability.

In the interest of portability, multi-physics algorithms are almost never specialized for different architectures. The burden of specialization can add too much complexity to an already complex code environment. A human mind can only handle so much complexity before it shuts down and becomes demotivated. That said, it does make sense to specialize the top time-consuming algorithms for a given architecture. Unfortunately, for a hydrodynamics multi-physics code, the top time-consuming algorithm may only account for 7% of runtime, and subsequent “top” algorithms drop off rapidly in the percentage of overall time consumed.

(G) What portability approaches were rejected and why? What approach is a leading contender?

We rejected any approach that did not reflect the underlying algorithms of our problem domain, or any technique that introduced too much “bureaucracy” or code versioning. Kokkos is currently a contender to the RAJA model we are exploring. Furthermore, a new lightweight RAJA has recently been prototyped by a summer student at Lawrence Livermore Lab that could be attractive to some code groups.

(H) What is your greatest fear for application portability and functionality when going to exascale?

Compiler and runtime implementations are of exceedingly low quality for the HPC domain. DOE needs to sponsor direct vendor investment to fix production compilers and runtimes. If something is not done about directly optimizing HPC idioms in current production compilers, DOE will effectively be buying large and expensive space heaters achieving 2% of peak performance. A “CompileForward” engineering initiative that uses actual code from ASC-class applications as a basis for vendor production compiler optimization would have a major impact on achievable efficiency.