Migrating to Heterogeneous Computing: Lessons Learned in the Sierra and El Captan Centers of Excellence
(the slides are available under “Presentation Materials” in the above URL)
Date: October 13, 2021
Presented by: David Richards (Lawrence Livermore National Laboratory)

Q. What portable abstractions are recommended for Fortran codes?

A. The only portable abstraction for Fortran and GPU offload that I know of is OpenMP.

Q. What about switching to APIs which utilize the various compute units (e.g. Chapel, Intel OneAPI, OpenCL, Xilinx Versal, etc.) from a single codebase?

A. Some codes and some teams may find great value in the various APIs you mention. There are many tradeoffs to evaluate in choosing a parallel programming abstraction. I strongly recommend RAJA and Kokkos for C++ codes because they are vendor agnostic, are standard C++, and have a proven record of single source portability across multiple CPU and GPU vendors.

Q. Is it possible to use Raja API to target both CPU and GPU simultaneously? If yes, how?

A. Yes. The Apollo Project is one example of a project that shows how a single code can use multiple RAJA execution policies and hence target multiple devices.

Q. What about using DMA-like operations from the GPU to avoid copying data back to the CPU or system MMU to have GPU & CPU units access the same memory to avoid performance degradation?

A. I’ll start with the observation that data needs to be in GPU memory to get high performance GPU computing. Hence, GPU memory needs to be the “home” of practically any data to avoid performance degradation. (Yes, I know that some algorithms like DGEMM can cover all memory transfers. However, these algorithms are the exception, not the rule). We do have DMA access to GPU memory from certain NICs, but as far as I know, none support strided or irregular access. I have suggested this feature to vendors. And some hardware can support “zero-copy” CPU directly to GPU memory. This isn’t exactly DMA access, but it does avoid a copy. Of course, it is still limited in both latency and bandwidth by whatever bus connects the CPU and GPU.

Q. Can you provide the name of the kernels talk, or a link to it, please?

A. The Importance of Kernels for Performance Portability, or How I Learned to Stop Looping and Love the Kernel, by Tom Scogland at P3HPC Forum 2020, Slides and Video.

The link to the video is also included on the last slide of my webinar

Q. Why is it better to write kernels than loops?

A. See Tom Scogland's talk for a full explanation.

The short version is that a loop is an inherently sequential code construct that implies a strict ordering of the iterations. A compiler can attempt to parallelize a loop, but that requires proving that it is safe to do so and that is not always easy. A kernel, at least in the sense I'm using it here, is an inherently parallel construct that allows a compiler much more flexibility in generating parallel code.

Q. P3 is a hot topic, but 'R' (reproducibility) is not often mentioned (but of course important in sciences). How does the 'R' fit in, and is it possible to some extent/precision? For example, RNG on one architecture will likely differ on another. Are custom kernels a potential resolution to ensure 'R'?

A. Reproducibility in parallel code is hard. As soon as you start to admit asynchrony and the resulting variable ordering of computations it is very difficult or impossible to guarantee bit-wise reproducibility. There is no magic bullet to solve this problem. One advantage of RAJA and Kokkos is that you can run exactly the same kernels with progressively more challenging execution policies. For example, you can start with a single thread sequential CPU policy, then move on to multi-thread CPU execution. You can test on the CPU where thread correctness tools are often more capable. Once you are confident that the code is running correctly on the CPU, you can move to GPU execution policies. The ability to change execution policies without changing code provides a number of powerful debugging and error checking options.

C. The latest issue of Computing in Science & Engineering (Volume 23, Issue 5, Sept.-Oct. 2021) is devoted to performance portability.

Q. OpenACC / Fortran is a successful option as well. Why no mention here?

A. I understand that OpenACC has been very successful for some codes. However, OpenACC is used by very few codes at LLNL. We recommend that such codes convert to OpenMP because we believe OpenMP compilers will be better supported, especially on future platforms.

Q. When a code is ported to GPUs, how do you test it? Is regression testing affected by porting? What tools do you use for testing in GPUs?

A. We use standard unit and regression testing practices as well as continuous integration for both GPU and CPU codes. I don't have any specific tool recommendations.

Q. How does one reconcile advice for performance with good software development practices? Particularly in Fortran.

A. Writing code that is portable, maintainable, and performant is challenging and there is continuous tension between those three objectives. Good parallel abstractions are one essential ingredient, but every team is different and every team needs to find their own way. I do think the problem is harder in Fortran.

Q. Is using Kokkos/RAJA kernels in a library performant? For performance, would one want to have the Kokkos/Raja loops in the host code or in the library, when kernels are defined in the library?

A. There are several examples at LLNL of libraries that use RAJA and obtain high performance. Some provide interfaces to pass entire data arrays to the library and the RAJA loops are in the library, others provide device callable functions that are called from kernels in the host code. Both ways work and can deliver high performance.

Q. How about Fortran + Raja? Or Kokkos?

A. Fortran and RAJA/Kokkos can interoperate just like any other Fortran and C/C++ code. Getting the linking step right in the build system can be a little tricky.

Q. Do you have many Fortran users?

A. There are some Fortran codes at LLNL, but they are in the minority. We see only two viable choices for Fortran codes on GPU: Either use OpenMP or re-write offload kernels in C/C++, preferably in RAJA or Kokkos, and call into C/C++ from Fortran. Both techniques can work.

Q. What is the lab plan for codes that, for algorithmic reasons (e.g. lots of branching), will not translate well to GPUs

A. There are multiple paths. LLNL maintains significant CPU-only cluster resources for codes and problems that don't run well on GPUs. Where CPU-only platforms don't meet performance needs we seek improved implementations of such algorithms or new algorithms and methods.

Q. For Fortran or for OpenACC users, is the gnu toolchain just not going to be a permitted option?

A. If the gcc toolchain gives you the capabilities you need then go ahead and use it. As far as I have been able to determine, there is no support for current generation AMD devices in gcc's OpenACC implementation. That is a show-stopper for Frontier and El Capitan.

Q. In this case (slide 19/20) what are examples of Degrees of Freedom?

A. In this context it is probably easiest to think of degrees of freedom as a way of describing problem size. It is the number of dependent variables in the calculation.

Q. How do smaller code development groups, especially at universities, get buy-in for collaboration from groups with more resources, such as the centers of excellence? Alternatively where can they get the resources to do this refactoring and porting work?

A. This is a difficult question with no easy answer. My best suggestion is to identify and focus on a unique value or capability that you offer.

Q. Can you elaborate on any issues that you had because of the differences of the two underlying APIs (and platforms) that RAJA/Kokkos is built on? CUDA on PPC64LE with cache line granularity (if used) and working managed memory vs ROCm on x86_64 with HMM just becoming available. Also, regarding the DoF/s vs DoF plots, even more telling is a plot that looks at overall time per timestep vs DoF - if one needs to go to very high DoF/GPU to get high speed, the time per timestep may end up being high despite the high speed.

A. The RAJA and Kokkos teams are both working closely with HPE and AMD to exploit new hardware capabilities and to identify and resolve problems in the compiler toolchain. This is a typical process for new systems and we are making good progress.

There are of course many ways to analyze performance and time per timestep is an important metric for many applications. We have found the throughput plots to be a good way to identify and understand the strong scaling behavior of our hardware and understand the regimes where the hardware is used at maximum efficiency.