LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-MI-692819

DOE-COE Breakouts

J. R. Neely, M. W. Epperly

May 23, 2016

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Performance Portable Abstractions Breakout

- Five questions:
  - What is performance portability? Isn't the real question, what are we targeting for our codes; what is 'good enough'? What have we shown is possible?
  - At what level(s) do we want abstractions? High level such as DSLs and architecture optimized algorithms/libraries? Low level data and task parallel programming models? What are the successes and failures we have seen so far?
  - What are the tradeoffs of different approaches? Directives, Attributes, Language extensions, DSLs? Any other categories? What have been the successes? What challenges remain?
  - What do we need from: Vendors? (Open)Community? What have we been happy with and what do we see as critical gaps?
  - What do we want to see supported by our programs? ASC, ASCR, ECP, etc.

What is performance portability?

- Using John Pennycook's definition:
  - *An **application** is 'performance portable' if it achieves a consistent level of performance across platforms (**relative to the best known implementation on each platform**).*
- With some additional qualifiers:
  - X% of the code must be shared between implementations (e.g. 95%)
  - The implementation must exploit available architecture features

At what level do we want our abstractions?

- Application developers can use library-level abstractions
  - Leave the problem of performance to the library developer
  - One library per architecture is no big deal
  - We need to build these high-performance libraries
- Developers want abstractions as low-level as possible, while remaining portable
  - This could be inside the libraries used by applications
- Legacy applications require incremental adoption
- Fortran cannot be ignored, but what can we do?
  - Pre-processing, RAJA/Kokkos interop with Fortran, `f2c`

What are the tradeoffs of different approaches?

- DSLs provide opportunity for kernel fusion, but inhibit incremental adoption.
- *Success story: IRP DSL generating optimized Fortran from Python*
- RAJA/Kokkos great for rapidly prototyping optimizations
- Abstractions using cutting-edge language features at the mercy of compilers
- Push to add parallelism to language standards makes them more bloated
  - This makes more work for compiler vendors

What do we need from vendors and the community?

- Focus on compilers
  - Support standards
  - Optimize code even through our chosen abstractions
- Tools that support our chosen abstractions (e.g. debug information)
- Better build systems
  - Difficult to combine wrappers, preprocessors and compilers

What do we want to see supported by the programs?

- Explicit support targeting system software (c.f. compiler research)
- Evangelization experts who can spread expertise, visit application teams to share knowledge

To Summarize

- Performance portable: substantial amount of shared code, exploits all target architecture, achieves consistent fraction of best possible performance
- Low-level abstractions for computer scientists
- Library-level abstractions for application developers if possible
- Improved compilers and tools
- Program support for abstraction layer experts