Software Design for Longevity with Performance Portability
(the slides are available under “Presentation Materials” in the above URL)
Date: December 9, 2020
Presented by: Anshu Dubey (Argonne National Laboratory and University of Chicago)

Q. You suggest “to design with reproducibility in mind”. What are the greatest challenges to design with reproducibility in mind given the different available platforms, compilers and programing models?

- **A.** That depends upon what one’s reproducibility definition is. If one wants bit-wise reproducibility, then the challenge in a parallel environment is huge. Compilers do provide options to run deterministically, but that is not a very good or sustainable option. So one should understand what correctness implies, develop diagnostics that can give confidence about correct behavior and make provision of such diagnostics in the design of the code itself.

Q.Could you give an example of the abstractions used in this intermediate solution?

- **A.** They are there in the slides

Q. Why use a new language for these “keys” instead of a “standard” preprocessor ?
I mean the @ syntax, that you referred to
Yes, but it’s not FPP or GNU CPP, right ? it’s some in-house thing

- **A.** That is not a new language, it is just an escape character that is highlighting where the keys are used in the example. The example is meant to illustrate the idea of how to identify the opportunities for using abstractions and how to deploy them within the code. So that was a bit of simplified pseudo-code.
  Where we actually use keys in the code they are allowed to have arguments, be inlined anywhere in the code, and have recursion. Syntax definition needs 3 lines of documentation, a comprehensive list of examples on how to use the tool and keys in the code is 1/3rd of a page. And the tool itself is a python code that can be embedded in our configuration process which uses a configuration DSL and is also written in python, so additional dependence is added.

Q. What do you think of the potential of very high level languages (like Julia), where the compiler can figure out optimal implementations by analyzing the full dependencies of the code?

- **A.** That would be utopia. However, compilers and languages have a long history of not being able to deliver on the promise of optimized code without some solid cooperation from the developers. Compilers always have to make the most conservative

assumptions, and therefore if the code is not well thought out in terms of being explicit about dependencies, the compiler is faced with an impossible task. So yes, it would be great to have a compiler which can do such analysis, but it is not going to do well with a code whose developers did not invest in good design.