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.

Tools Breakout – Brian Friesen, Moderator

Q: What tools exist today that help achieve performance portability?

Our group had trouble answering this question, which indicated strongly that such tools probably don't exist. Or, if they do exist, they are not readily available. For example, ALCF has found that vendor-supplied tools (e.g., from IBM), are rarely sufficient or useful, and as a result they rely almost entirely on 3rd-party tools, e.g., TAU. Unfortunately there is a trend that some of these tools are supported directly or indirectly by DOE projects, which have limited lifetimes (funding). If the developers of such tools can't find a way to support themselves outside of DOE, then they'll be stuck when the project ends. TAU is a success story which has avoided this fate; ParaTools sustains itself independently of DOE. Additionally, the group agreed that the most valuable tools available to everyone in the HPC community are **compilers**. In fact, there were some interesting comments regarding compilers as tools in the "afterward" from some of the vendor representatives at the conclusion of the meeting. John Levesque, for example, cautioned that developers often want to use bleeding-edge features of compilers without fully appreciating the performance implications involved in doing so. Similarly, James Reinders said that vendor compiler teams' goals are often driven by the specifications in the procurement RFPs, which often place more emphasis on support for bleeding-edge features than on performance and stability. (One code team has a battery of 18,000 regression tests for their >1M LOC code, and Intel v13 (!) is the most modern version of the compiler suite which can pass all of the tests.)

Very quickly the discussion then focused on a tool that already is available and is probably the most valuable to everyone: **dissemination of knowledge**. Many developers in the HPC community have followed many different paths in the pursuit of performance portability, but because the knowledge gained from these exploits seems to be shared so infrequently, we often re-invent the wheel, or are never made aware of solutions that have worked well for others who are working on similar problems.

The entire breakout group agreed strongly with the notion that disseminating knowledge and experiences with regard to performance portability strategies needs to be a top priority. We brainstormed several different avenues for addressing this, including

- more frequent (and perhaps less dense) meetings such as the one just concluded in Glenade
- Collaborative forums of some form
- A webinar series

- Discussion forums
- Wiki pages

Things of this sort are beginning to precipitate already: the Trinity COE already has begun monthly KNL meetings, and IXPUG has similar regular meetings. However, a major complication in attempting to implement any of these solutions is the NDAs that each lab has with various vendors. We would therefore need a way to institutionalize these kinds of venues in a way that protects confidential information, e.g., by allowing access only to people with e-mail addresses from corresponding institutions.

What tools/interfaces should vendors provide to extract information interest about memory access patterns, data layout in memory subsystems, etc.?

This is a very difficult task to accomplish, in part because doing do requires access to things like hardware counters. Doing things like tracking loads/stores is possible (e.g., VTune's "memory access" mode), but the complex cache hierarchy in KNL systems makes it difficult to follow individual pieces of data in any more detail than that. As a result, we can obtain statistical samples of somewhat generic memory access patterns (loads and stores), but gleaning insight from those statistics can be challenging. The group also discussed the barrage of available flavors of malloc() and prospects for leveraging them to do memory access analysis.

How can applications and libraries share resources amiably? Can an application specify what portion of available resources it is willing to give up for library calls?

Intel's Math Kernel Library (MKL) is a mixed bag in this regard. On one hand, it allows the user to force MKL routines to use the same set of existing threads from the application that called it, rather than spawning new threads inside each MKL call (leading to unnecessary nested thread parallelism and perhaps degradation in performance). On the other hand, users have found that MKL often creates unnecessary copies of small matrices during calls to certain linear algebra routines.

The group also discussed the prospect of an application allowing (or restricting) a library call's access to various subsystems, e.g., is the library allowed to use high-bandwidth memory, or should it be restricted to DRAM? It's not clear whose responsibility that should be. On one hand, the purpose of libraries is to be of use to a wide range of applications and therefore should support generic, flexible interfaces; on the other hand, library developers should not be expected to re-architect their entire interfaces each time a new architecture emerges.

What role should compiler technology play in performance portability? What features do we already have? What features are still needed?

As discussed in an earlier section, it may not be wise to demand cutting-edge feature support from compilers when they already suffer from stability issues stemming from features which have long since been implemented. (One code team is forced to use v13 of the Intel compiler suite because no newer version can pass their entire regression test suite.)

Developers have often experienced lackluster response from compiler vendors when reporting bugs. To motivate vendors to focus more heavily on compiler stability, perhaps we should take John Levesque's and James Reinders's comments seriously, e.g., by focusing more heavily on compiler support in procurement RFPs. As it stands currently, a common experience is for a site to lose leverage with regard to compiler support once the machine has been accepted. The group gravitated significantly toward this idea in particular, and suggested assembling a suite of benchmarks (composed perhaps of mini-apps contributed from the community) that must compile and run on new systems (with new compilers) *before* they are accepted.

Another common experience with regard to compiler technology has been the direct interaction of compiler engineers with code teams. This has manifested in various forms already, including the "dungeon sessions" which NERSC and others have participated in with Intel over the last year as part of the NERSC-8 procurement. Compiler bugs which can be shown directly to engineers are often elevated to high significance within the vendor software teams and are fixed more readily. Someone also suggested inviting compiler engineers to COE meetings such as this one, both to provide/receive feedback among HPC consumers/developers, and also to provide perspective on what goes into writing performant/stable/cutting-edge compilers.