LLNL-MI-692819

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

# DOE-COE Breakouts

J. R. Neely, M. W. Epperly

May 23, 2016

## Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

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

# Managing the Memory Hierarchy
# Breakout Session 1

Douglas Doerfler (LBL), et al,

DOE Center of Excellence Performance Portability Meeting

April 18-21, 2016

# Breakout Charge Questions

- What are the practical limitations of using current programming models for managing the memory hierarchy
  - Do you plan to integrate multi-level memory support into your code?
  - What are your memory capacity requirements in the 2020 timeframe?
  - Can you live with 16, 32, 64 GB per node? Per NUMA domain?
  - How much effort are you willing to do to support multi-level memory?
- Languages, directives, attributes, other?
  - Are you willing to use a "non-standard" memory management programing model?
  - Do you need memory management interoperability of C, C++ and Fortran in a common code?
  - Would you like to see a type attribute for variables to declare fast memory storage?
- What is the proper balance between user control and runtime control for memory placement and management?
  - Did Ian's presentation cover all of the possibilities here?

# Setting the Stage

- Assumptions, boundary conditions for the discussion
  - On package memory (MCDRAM, HBM)
  - Off package, bulk capacity, memory (DDR)
  - Byte addressable non-volatile memory (future NV technologies)
- Quick survey: Are you actively integrating multi-level memory (MLM) into your code?
  - About ¼ of application developers said yes
  - About ¼ said will be in the near future
  - About ½ were not developers
- Are we really sure we need MLM concepts in next-generation machines?
  - No clear indication we can avoid MLM in future machines
  - Skeptical that on-package memory only can satisfy adequate Byte/FLOP balance ratios

# Practical Limitations of using Current Programming Models

- What's wrong with memkind?
  - Assumes that were data resides is static, but real codes go through multiple phases so you want to dynamically change data attributes
  - Memkind solution is completely developer managed
  - Not sure why one would want a library-based solution
  - But still want a way for for developers to to this at a low level
- What developers really want is to able to describe the attributes of data and have introspection of the node to help manage data placement
  - Some combination of the compiler and a "runtime" to manage the data
- Also need a higher level, higher productivity solution
  - CHAI style?
  - UVM?
  - OpenMP?

- There is a desire and a need for variable type attribute extensions to specify “memory characteristics”
  - Attributes (vs declarations) allow type characteristics to propagate through the system
  - Some disagreement that a declarative statement is sufficient, but there was some argument that the extra semantics would help in using a data structure with this information
- This is a language issue and is just as applicable to Fortran as C/C++
  - However, changes in type system in languages will take time to get through the language committee
- Some discussion that the attribute should not be “fast”, but instead “doesn’t need to fast”.
  - May also want to capture other attributes such as latency
- Action: Cray has agreed to explore the attribute feature
  - Group can send suggestions to Luis De Rose (ldr@cray.com)

- Would appreciate not just a programming model but also a tool to tell us what data structures would benefit most from fast memory
  - This is my hotspot for memory accesses
  - Is this a latency bound access, or BW bound
  - This still may have the limitation that the results will change with input deck and phase changes in the code
- Action: Recommendation the the CoE have tutorials for tools available today
  - Cray does have some capability in this area based on L2 misses, release sometime this year
  - Nvidia's nvprof can already see memory migrations (UVM)

# Languages, directives, attributes, other?

- How do get C/C++ and Fortran to use the same mechanisms for data attributes
- Much of the previous discussion covered this topic area
- This propagates down to the libraries too
- We need cross standard standards!

# Proper balance between User control and Runtime control for memory management

- Statement: “Doing memory management by hand is hard, we did that on RoadRunner for the SPEs …”
- Statement: “Would like to see a hierarchy of approches from use it as a cache right through to low-level programmer driven
- Again, much of this topic was discussed in prior topics
- Continued to make the case runtime control
- Impacts MPI, O/s, libraries, I/O buffers, etc
- Analogy with binding processes and threads to cores is similar
  - Except we really don’t want to bind due to dynamic nature of an application
- A brief discussion regarding the ability of the O/S to be part of the memory management
  - Certainly could, but should it be? -> NO