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.

Managing the Memory Hierarchy
Breakout #2

Bronson Messer (ORNL), et al.

What are the practical limitations of using current programming models for managing the memory hierarchy?

- We don't know what we really **need** now, therefore the limitations are not fully known.
- Strong consensus that many users have identified a need to manage the hierarchy
  - Electronic structure, lattice QCD, deterministic transport, other multi-physics codes given as examples of codes that will not fit into small, fast memories.
- Memory footprint needs a normalization to be meaningful, e.g. bytes/peak TFLOP or size/bandwidth.
- For many codes, there is a minimum amount of memory required per MPI rank.

What are the practical limitations of using current programming models for managing the memory hierarchy?

- The “COE platforms” (SC ASCR 2018-era platforms at NERSC, OLCF, and ALCF and ASC platforms around same time) will have HW features that smaller platforms will not share (e.g. Linux clusters with older GPUs)

- One possibility is to use HBM as a cache, but managing that cache explicitly as a user seems daunting.
- MPI SHM provides a picture of how bad things can be.
  - Can lead directly to writing a hand-made memory manager

What are the practical limitations of using current programming models for managing the memory hierarchy?

- True shared address space—where the system "finds" the data for you—opens up several possibilities for tools and finer user control.
- Important to separate the distinct, but related, issues of data placement and data layout
  - Data placement is physical memory location
  - Data layout is how a program traverses does data structures (e.g. SoA vs. AoS)

Languages, directives, attributes, other?

- Libraries and consistent API's preferred to directives.
  - Portability is the primary driver here.
  - Not dependent on compiler support.
  - Indeed, you can package a library with the code.
  - Importantly, you can hide directives with macros. It is more a compiler-support issue.
- Waiting for language standards to take hold is not a realistic strategy for the future that is upon us.
  - Also, probably premature to design features for standards (cf. earlier comment about what is needed).

Languages, directives, attributes, other?

- Wrapping platform-dependent allocation and placement methods is already common.

- We do need some guidance from the vendors to determine the scale of the gap we need to bridge between what is possible and what is desired.
  - But, the vendors have to optimize over finite development resources to effectively answer the question.
  - Nevertheless, a concise list of guarantees would help.

What is the proper balance between user control and runtime control for memory placement and management?

- "Balance" is perhaps the wrong word.
- Nevertheless, strong consensus for a combination of reasonable defaults and the possibility of fine control
- There are always tradeoffs between absolute performance, maintainability, resources, and portability.
- Opaqueness for performance is a huge potential problem.
  - Already a problem today in, e.g., PGAS languages

What is the proper balance between user control and runtime control for memory placement and management?

- Page faults or allocations exceeding device memory need to be able to be seen.
  - Even to the point of stopping program execution
- If people have to confront full complexity to get started that is a problem.
  - It was suggested this was a lesson learned with the Cell processor.
- Tool writers will need to pick some abstractions to provide information between “fail” and gritty details of paging.
  - These chosen abstractions might need to be the same as those under user control.

What is the proper balance between user control and runtime control for memory placement and management?

- We often talk about memory management and mean dynamic memory management: That is a big assumption.
  - We should discuss static memory and how the runtime will manage it, particularly thread-static

- If you maintain your codebase frozen in amber, you have to accept that new capabilities and HW features may well be beyond your reach to exploit (e.g. the use of Fortran COMMON blocks).