LLNL-MI-692819

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

DOE-COE Breakouts

J. R. Neely, M. W. Epperly

May 23, 2016

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

OpenMP Futures Breakout

David Richards, LLNL

Sriram Swaminarayan, LANL

Glendale AZ, 4/2016

Many thanks to Charles Ferenbaugh for taking notes

Three Big Takeaways

- Not enough users/ apps people on standards groups
- We need a High-level way of mem mgmt. across platforms
- <politics> Accelerator subgroup needs to talk to affinity subgroup

Best Practices

- **Are not clear yet -- Standard changing so rapidly, hard to keep compilers up to speed – both in implementation and in performance**
- ***Standard comes out before it's been tried in anger***
- ***OPenaACC went through growing pains, OpenMP will likely do the same thing and we are seeing it now.***
- **Some tutorials are out there Here are some suggested ones:**
  - Van der Pas & Chapman (3.1)?
  - OMP website? Sometimes picking up examples and using them will slow down your code (not an OMP problem, but wrong usage). Examples on OMP website are maintained –but these are how to use the features – not for advanced optimization
  - SC tutorials
- **Gotchas listing would be nice:**
  - **– crossing NUMA domains; OMP overhead; Need a document of common gotchas? Even NAS benchmarks sometimes do this wrong - Some users don't understand how memory systems work – they're won't understand how to use OMP correctly**
  - **NERSC also has tutorials; videos from GTC**

Composability with third party libraries is an issue

- interop with other threading systems (e.g., pthreads in a LLNL code),
- different packages using OMP differently, OMP in oversubscription mode (AWE), MPI + MPI (PETSC),
- OpenMP + C++ STL ("broken by standard")
- We would like OpenMP to work with other tasking models, e.g. Legion
- Interoperability with MPI needed but best method not known – best practice is not known yet – Is MPI_THREAD_MULTIPLE needed for our threads? Need someone to try before you can see if this is the right way

Performance portability depends on compute bound or memory bound

- For memory bound code, OMP can approach CUDA on GPUs; but not on compute bound code – some codes can be perf portable in OMP, but can all?
- We would like to be more descriptive about execution and memory
- Current OMP is prescriptive – We want to be able to specify where you expect highly parallel vs. highly serial code
- On other hand, some things can't be done in OACC because it's not prescriptive
- *Need a data model/handle shared data properly*

Performance portability continued

- No easy path from OMP3 – OMP4.5: Someone with OMP 3 code doesn't have a way to come with a heterogeneous system and get performance
- OMP now has a simple memory model, tied to offload paradigm – how you treat Multi level memory on the Phi is not clear – no way the user can explicitly say what they want – could you use "map" that way? Maybe, but that's not the intent – memory hints? Places setup? Intents of use (e.g., I'm going to stream this, put it in HLM)
- Defaults that behave in similar, reasonable ways would make perf portability easier (e.g., CPU vs. GPU, static(1) vs. static) - some sensible defaults are not available – OMP is sometimes more restrictive than it needs to be (e.g., collapse)
- Heterogeneous computing not allowed i.e. no spawning threads across host and accelerator. Proposal exists
- *With all the different flags, does config space explode? OACC already has this, OMP will end up there*
- In most cases you can use defaults until you're trying to get the last few % of performance – decide how much perf you want vs. how much pain

Optimization is a myth

- **Control over sequential optimizations might be a compiler responsibility:**
  - Asking for unrolling, collapsing, etc., don't really fit philosophy of OMP – should be part of compiler/language instead
  - Compiler can tile a loop, but can't do analysis to see whether tiling is safe
  - Tiling not in OMP yet – not yet clear what that would mean – what iterations would end up in your team if you do tiling?
  - Specifying dependencies between loops: Can't tell compiler about this, especially across translation units – OMP can't cross translation units
- **Loop fusion: end up with 10/20/400 parallel regions right next to each other - trying to make Raja merge them so threads don't join/restart/join… - nowait works if you have an outer parallel region, but if you don't what does it even mean? Can threads teleport to the next parallel region?**
  - But If parallel regions are humongous, that can make GPU code harder to write
- **Can we use dependencies to run asynchronously? The machinery exists, but it's hard to do manually before your brain explodes – would need an automated system of some kind**

Rest are original slides

Topics to cover

- Performance Portability
- Usability
- Memory management
- Execution
- …

Performance Portability Related

- **What is missing in OpenMP for performance portability?**
- **Is the OpenMP abstract machine adequate to reason about performance?**
  - Is it possible to define an adequate abstract machine or do we need to explicitly support/be aware of architectural differences?
- **There are different practices for different architectures**
- **Do you need better support for STL?**

Usability Related

- **Best practices**
  - Does a ‘best practices’ document exist somewhere? Who maintains it?
  - What programming patterns work in OpenMP?
  - What are the characteristics of codes that admit such patterns?
  - Do we now need a new set of practices? Will community adopt it?
  - Different annotations for different architectures
- **What are most important interoperability concerns for OpenMP regarding:**
  - third party libraries?
  - other programming models?
  - other threading systems?
  - MPI?

Memory Management

- Is OpenMP adequate to manage the lower levels of the memory hierarchy?
- Do we need additional facilities and what would they look like?
- Could code written with such features ever be considered performance portable?

Execution, Runtime, etc…

- **Do you need more control over sequential optimization?**
  - Tiling of loops?
  - Automated unrolling?
  - Support for wavefront loops?
- **Do we need directives for specifying dependencies between loops?**
  - Loop fusion?
  - Pipelining?
  - Intermediate variables?
  - Dependency list?

Backups in case discussion dies…

Usability Improvements

- **Best practices**
  - Does a 'best practices' document exist somewhere? Who maintains it?
  - Do we now need a new set of practices? Will community adopt it?
- **It is hard to do OpenMP well**
  - How can I do OpenMP wrong? Let me count the ways…
  - Different annotations for different architectures
  - No heuristics easy to find (see best practices above)
- **Should OpenMP be more descriptive?**
- **Is accelerator support adequate?**
- **Is support for function pointers needed?**
  - Do we need access to function pointers generated by compiler?
- **Is support for lambdas needed?**
- **Do we need API routines to understand how the device works?**
- **Are OpenMP tasks usable?**
- **Are overheads too high? What would you give up to get lower overheads?**

Memory Management

- **Do we need Futures in OpenMP?**
  - Easier specification of dependencies
- **What memory management improvements do you want to see?**
  - Hierarchies?
  - Affinity? (NUMA aware)
  - Deep copies?
  - Allocations? (is target / map enough)
- **Should OpenMP infer target directives based on the context?**