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.

Performance Portable Abstractions

2016 COEPP Discussion

Rob Hoekstra(SNL)

What is performance portability? Isn’t the real question, what are we targeting for our codes; what is ‘good enough’? What have we shown is possible?

- What is ‘Good Enough’?
  - Varies by community: couple factors OK for some, close to peak for others
  - Climate on one end; CFD & astrophysics on the other; NNSA: ‘happy enough’ in between, long slow incremental improvement is the norm
  - Investment varies dependent on size and complexity of code base
  - Developers have limited time to devote to performance
- Performance portability implies that the code is more maintainable and allows greater productivity than platform specific variant
  - Avoid changing code so much developers can’t be productive
  - Allow different ‘classes’ of developers to focus on different levels of performance optimization
  - Model which allows incremental incorporation of improved performance
- Performance portability: Are you able to run productively on new platforms quickly?
  - Get the code up and running on the machine, baseline it before you can tune it; this is a big initial hurdle

At what level(s) do we want abstractions? High level such as DSLs and architecture optimized algorithms/libraries? Low level data and task parallel programming models? What are the successes and failures we have seen so far?

- High Level vs. Low Level
  - Want customization; especially for higher level constructs this can be very difficult
  - Lower level may give you this to a greater degree
  - Where is the demand? When there is a lot of commonality in need does that make high level abstractions more attractive?
  - Must develop community support in either case!
- When you have dependencies you are left vulnerable to their timetables.
  - What about a corner case that only you care about?
  - Need to be able to work around if it doesn't meet your needs.
- Successes? Directives? Template Metaprogramming? Fortran? DSLs? Thrust?
  - OMP has both been a success and a failure
    - Lack of performance has been a big issue; constantly changing complex standard; lots of overhead; Can't abandon old features
  - Every success has its caveats; no slam dunks
- Failures? POOMA, Fortran Parallel Constructs
  - Fortran: not adapting to the new architectures, new features not being implemented or used; greatest performance portability for traditional CPU architectures; future hiring may be an issue
- Has the calculus changed enough to drive compilers to support parallelism in languages?
  - Who in the community will be pushing for shared memory PMs?
  - Will the Fortran community be a part of that?

What are the tradeoffs of different approaches? Directives, Attributes, Language extensions, DSLs? Any other categories? What have been the successes? What challenges remain?

- Productivity issues in general
  - Example: meta-template programming drives up complexity and compile times
- OpenMP
  - Currently not a very robust, portable, performant solution especially with C++
  - Way too large now and getting worse; over constrained
  - BUT still seems to be a default winner…
- Language vs. Directive
  - Want 'sticky' attributes; need to be able to manage types
  - Need to be able to better inform compilers of your intention
  - Data attributes are not 'first class citizens'; compilers are not designed for full support allowing optimization
- There is a really good reason that MPI is not directive-based!
- What about the LLVM-IR level?
  - Extend with parallel constructs?
- Is there any way to make DSLs more broadly of value?
  - Neuromorphic community is investing in DSLs. Is there anything to learn?

What do we need from: Vendors? (Open)Community? What have we been happy with and what do we see as critical gaps?

- Working with vendors
  - Motivate vendors by asking for things that have broad appeal to the market
  - Helps when the labs can speak with a common voice
  - Vendors are listening but timelines can vary widely
  - Need early access to vendor tools
  - Vendors complain that there is ‘radio silence’ at times
  - Lack of portability with vendor libraries even though data structures are generally the same
  - Open/Shared Performance Suite used by vendors to evaluate compilers/tools
- Compilers are a crucial area
  - Need a forum for developers and compiler folks to talk
- Are there things you want to see move to the open community?
  - We are very happy with the move to LLVM
  - Do we want to work at this level to add language extensions?
    - How do we get them optimized by the vendor backend?
  - Partner with large companies that are producing open source software
    - Example: Google Thread Sanitizer

What do we want to see supported by our programs? ASC, ASCR, ECP, etc.

- Stronger emphasis on software environment in the procurements
  - Compiler updates/support throughout the life for the platform
  - Development environment as a whole
  - Emphasis on performance tuning capability
- Strong engagement by the labs with the standards communities
  - Get the standards to incorporate the things we need
  - We need to have more influence on Fortran standard
  - Broader than just languages standards
- Can we get better collective focus by the programs?
  - Driven by code requirements?

Closing Remarks

- Performance Portability: We know it we see it.
- Maintainability and Productivity are a crucial part of the equation
- Dependencies make you vulnerable
- OpenMP may be winner by default but there are lots of benefits to language based approaches
- Procurements need to emphasis software environment more
- We need to keep engaging the community standards

What is performance portability? Isn't the real question, what are we targeting for our codes; what is 'good enough'? What have we shown is possible?

- Maintainability, productivity
- Varies by community: couple factors OK for some, close to peak for others
  - Climate: factors; CFD: near peak; astrophysics:near peak; NNSA: 'happy enough', long slow improvement
- Investment is very dependent on size and complexity of code base
- Reference implementation necessary? Are proxies good enough to evaluate this?
- Avoid changing code so much developers cant work on it anymore
- Developers have limited time to devote to performance
- Allow different 'classes' of developers to focus on different levels of performance optimization
- Build from high performance 'lego blocks' is the hope
- Get the code up and running on the machine, baseline it before you can tune it; this is a big initial hurdle
- Model which allows incremental incorporation of improved performance
- Outside dependencies can be both beneficial and a barrier
- Are you able to run productively on new platforms?

At what level(s) do we want abstractions? High level such as DSLs and architecture optimized algorithms/libraries? Low level data and task parallel programming models? What are the successes and failures we have seen so far?

- Want customization; especially for higher level constructs this can be very difficult; lower level may give you this to a greater degree
- Where is the demand? When there is a lot of commonality in need then higher level may be more feasible; less commonality requires lower level building blocks
- OMP has both been a success and a failure; lack of performance has been a big issue; constantly changing complex standard; lots of overhead
  - Can’t abandon old features
- Successes? Directives, Template Metaprogramming, Fortran? DSLs? Thrust?
- Fortran: not adapting to the new architectures, new features not being implemented or used; greatest performance portability for traditional CPU architectures; future hiring may be an issue
- Failures? POOMA, Fortran Parallel Constructs
- Has the calculus changed enough to drive compilers to support parallelism in languages?
- Who in the community will be pushing for shared memory PMs, will the Fortran community be a part of that?
- When you have lower level dependencies you are left vunerable to their timetables of these abstraction layers; what about a corner case that only you care about? What is the sweet spot? Need to be able to work around if it doesn’t meet your needs.

What are the tradeoffs of different approaches? Directives, Attributes, Language extensions, DSLs? Any other categories? What have been the successes? What challenges remain?

- Productivity issues; metatemplate programming drives up complexity and compile times
- What about the LLVM level?; LLVM-IR(extend with parallel constructs)
- OpenMP; currently not a very robust, portable, performant solution especially with C++; way too large now and getting worse; overconstrained
- Language vs. Directive; want 'sticky' attributes; need to be able to manage types; need to be able to better inform compilers of your intention
- All of above is algorithm related
- Data attributes are not 'first class citizens'; compilers are not designed for full support allowing optimization
- There is a really good reason that MPI is not directive-based!
- High level constructs like DSLs have proven to not be widely adopted and supported; these need to be a community driven effort.
- Neuromorphic community is investing in DSLs.

What do we need from: Vendors? (Open)Community? What have we been happy with and what do we see as critical gaps?

- Lack of portability with vendor libraries even though data structures are generally the same
- Motivate vendors by asking for things that have broad appeal to the market; Helps when the labs can speak with a common voice
- Vendors are listening but timelines can vary widely
- Open/Shared Performance Suite used by vendors to evaluate compilers/tools
- Early access to vendor tools
- Vendors complain that there is ‘radio silence’ at times
- Forum for developers and compiler folks
- Are there things you want to see move to the open community?
  - We are very happy with the move to LLVM for example
  - Do we want to work at this level to add language extensions? How do we get them optimized by the vendor backend?
  - Leveraging google software development environment? Are there other opportunities? (Thread Sanitizer)
  - Partner with large companites thet are producting open source software