The Role of OpenMP in Performance Portability

Bronis R. de Supinski

Chief Technology Officer for Livermore Computing
Chair, OpenMP Language Committee

June 26, 2023

LLNL-PRES-850488

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

A Position Statement on Performance Portability

Bronis R. de Supinski

Chief Technology Officer for Livermore Computing
Chair, OpenMP Language Committee

June 26, 2023

LLNL-PRES-850488

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

What is the easiest way to realize performance portability?

What is the easiest way to realize performance portability?

- Answer: Start with/attempt to achieve poor performance.
- Of course, no one intends for “performance portability” to mean that.
- So what is “performance portability”?

Performance portability is a myth

- Consistent strong performance only achieved in sequential, compiled programs
  - One could argue MPI-everywhere can achieve it also
  - Except it fails for systems with accelerators
- Some cite solutions such as Kokkos and RAJA
  - Reality is that they reduce how much tuning is required
- “Ease of performance attainment” is more realistic

Some requirements for ease of performance attainment

- A compiler that generates good machine code
  - But not reliance on the "magic compiler"
- Mechanisms to guide the compiler
  - Provide it information that the programmer knows but would require complex (or impossible) static analysis
  - Dynamic context-dependent specialization
  - Low-level control (and interoperability) when needed
- Mechanisms to specify appropriate parallelization strategies
- Mechanisms to control use of optimizations
- Diverse abstraction mechanisms
  - The real lesson of Kokkos and RAJA

OpenMP provides essential features for large-scale ease of performance

- OpenMP is supported by all major compilers
- OpenMP supports a wide range of parallelization models, devices
  - Widely used for shared memory parallelism
    - Loop-level support is its most familiar set of features
  - Task-based parallelism has been supported for over ten years
  - Device constructs (e.g., `target`) support heterogeneous nodes (and systems)
    - Does not assume shared memory ➔ distributed memory parallelism
- OpenMP allows programmers to be prescriptive when necessary
- OpenMP is provides interoperability with key mechanisms
  - OpenMP is naturally interoperable with MPI
  - Mechanisms such as the `interop` construct to support low-level device languages

OpenMP `metadirective` supports advanced specialization

- Optimizations are frequently context specific
  - OpenMP metadirective supports appropriate choices

```
#pragma omp metadirective \\
  when(device={arch(nvptx},user={condition(Niters<NV_min)}:target teams loop) \\
  when(user={condition(Niters<min)}: target teams distribute parallel loop)   \\
  otherwise(target teams distribute parallel for simd num_teams(tcount))
for(i = 0; I < Niters; i++)
  do_work(i);
```

- OpenMP contexts cover key system and code features
  - Enclosing OpenMP regions (e.g., is code encountered in a `target` region)
  - Device or target device architecture and other features
  - Implementation-defined contexts
  - User-defined contexts

OpenMP is becoming the language in which to program your compiler

- OpenMP `metadirective` is one example
- OpenMP is adding loop transformation directives to enable standardized prescriptive control of key compiler optimizations
  - OpenMP 5.1 added `tile` and `unroll` directives
  - OpenMP 6.0 will include `reverse` and `interchange` directives (at least)
  - The `apply` clause sill support optimization of transformed code
- OpenMP assumption directives standardize a common mechanism to guide compiler optimization
- Can ensure compiler support for key features with the `requires` directive
- OpenMP is now the best starting point for complex autotuning tools
  - Standardized infrastructure promises to make these tools more portable
  - Past approaches as well as newer AI-based ones could deliver some desired compiler magic

What OpenMP extensions would further ease performance attainment?

- OpenMP 6.0 will support top-level tasking, which will simplify efficient resource utilization
- Is the loop construct useful?

```
#pragma omp loop [clause [[,] clause] …]
```

- Is support needed to use multiple devices on a node?
- Are Fortran users interested in lambda support?
  - OpenMP requires support for outlining and variable capture
- Other missing features?

Lawrence Livermore National Laboratory