

Application Migration and Performance Expectation for Manycore Programming

This was a lively session with ~80 attendees. Ilene Carpenter (NREL), Dan Stanzione (TACC) and John Michalakes (NOAA) each gave a five minute introductory talk, which was spiced up by John arriving with only seconds to spare having become engrossed in judging the student cluster challenge. The floor was then thrown open for questions.

There was general agreement from the speakers that

- porting code to the Intel® Xeon Phi coprocessor is easy, but that getting performance requires non-trivial tuning efforts to ensure good parallelization and vectorization.
- there is no single way of writing code that achieves performance portability
- OpenMP* 4.0 is not (going to be) a magic bullet, though multi-vendor standards are definitely a good thing
- code-owners should be focusing on exposing parallelism in their codes at all levels (vectors, threads, MPI), and making it explicit. They can then choose how to implement it (OpenMP simd pragmas, OpenMP parallelism, OpenCL*, vendor specific languages/pragmas) but they need to know where it is.
- you should beware of quoted speedups. As Dan commented “they’re mostly meaningless because the original code is so atrocious”.

Questions from the floor

How do we normalize the amount of optimization across architectures that use radically different programming models?

Dan said that, even using OpenMP 4.0, there are going to need to be changes for different architectures but it will be less if you get the same person to do the optimization across the different architectures. John noted that one such difference is that Cuda wants to vectorize the outer loop while Carpenter said it would be nice if the Fortran compiler could reverse the loops for you. So, on the original question, the answer is “You can’t easily”

Wouldn’t it be better for users to experiment?

John commented that he is often asked what machine people should buy to run WRF. His answer is that if you are in CS, get the latest and greatest machine and experiment, but if you are a meteorologist and just want to run it to do real (rather than computer) science just get a cluster. Ilene also suggested just starting with a Xeon cluster. Dan suggested that languages won’t get better in the near future but regardless of where you run your code, you must vectorize and thread your code – that is where people should be working now.

What would you like to see going forward? Would it be helpful to add things like locality directives? Would it be helpful to expose things to the user? Or is it best to hide this from the user?

General consensus seemed to be – good question.