Three Things to Consider After Intial Speedups

Many people like to try a small kernel with ArBB first. If you feed this small kernel a lot of data, you will initia see speedups on par or greater than the "quick and dirty" threading/vectorization tools you have used in the pas But what should you do next to get the large speedups and high scalability across cores you're expecting with ArBB?

0. If you are trying to code something that is already available in MKL, IPP, or another optimized library, pleas to choose another algorithm that is not a "canned" function to benchmark with ArBB. ArBB is not a substitute those heavily optimized libraries. ArBB is to be used when those "canned" functions do not apply and the algor you are trying to code up is

a) difficult to code up with the conventional C++ syntax

b) even harder to both vectorize and thread across multiple cores

and

c) hardest to make portable to future architectures

1. Most of the time you are not doing anything incorrectly. The next step to really ramp up in speedup is to add much more computation, "do much more" (i.e. compute something meaningful in the context of a more comple algorithm). ArBB is well suited for doing large computations and doing more operations inside ArBB functions where your small kernel is just one piece of a much larger puzzle.

There are some overheads might be overshadowing the benefit of your small kernel. Those overheads are negli for more realistic examples (for example, if someone is doing a saxpy to compute something meaningful in the context of a more complex algorithm). By the way, if you are trying something analogous to saxpy with ArBB, a nice way to learn the ArBB syntax, but please see item 0 above.

2. Have you seen how to use map in conjunction with call? Take a look at some of the samples that use the map function. This allows you to address individual elements without the _for loop, enabling the ArBB runtime to d more parallelization optimization. Vector and map operations generate better threaded code.

3. During the first invocation of 'call,' there is a performance hit due to the JIT. Since the code is cached after th JIT, all subsequent runs do not incur any performance hit. So you should be timing runs 2 through X and rule ou first run. But what about a real-world scenario you might ask where the first run is indeed important?? We have answer to that! You can use manual closures and capturing to record all the operations at a specified time. Mos people like to do it at the beginning and have an initialization phase where all of the ArBB functions are compil (kind of like a video game having a 'loading' screen). A few articles in our Knowledge Base are already written

about the JIT overhead and how to use closures and capturing to have full control over the runtime compilation process.

Do you need more help?
Click tags links for related articles Search Knowledge Base Visit User Forums Get other Support options

This article applies to: Intel® Array Building Blocks Knowledge Base