OpenMP versus Cilk

Kitur Ganesh (transcribed by Ramesh)

January 9, 2014

- For simple loops, OpenMP performs better for small, very-balanced loops. Also, for non-loop parallelism, Cilk Plus is far easier to work with.
- Of course Cilk Plus has vectorization. All of the loop vectorization support in OpenMP was copied almost verbatim from Cilk Plus. They are almost entirely equivalent at this point. However, Cilk Plus also has array notation, which supports vectorization with a more concise syntax that some people really like.
- Cilk Plus composes better with other things than does OpenMP.
- OpenMP has better support for cache affinity. This comes into play when a parallel loop is executed multiple times, each time working on effectively the same memory locations. Using static scheduling, OpenMP will typically execute the same iterations on the same cores, giving much better performance by reusing the cache. Making the Cilk scheduler more cache-friendly is a current research project that's going on.
- Cilk is easier to program in, but OpenMP has more knobs for fine-tuning. It comes down to how much tuning a developer is willing to do. I think you get more bang for the buck with Cilk, at least initially.

1 From Pablo

Let me make a few points about Cilk Plus: It provides language extensions for task and data parallelism and is a simple yet powerful compiler based solution particularly with low overhead. Please refer to the page at: `http://www.cilkplus.org/faq` especially the sections under "Why do C and C++ need a language extension for parallelism?" which clearly points out the following: "Intel Cilk Plus supports a higher level of abstraction by providing constructs for expressing potential parallelism in an application, not mandating it in the compiled code or runtime. This higher level of abstraction separates the specification of parallelism from scheduling: the programmer is free to focus only on what code is allowed to execute in parallel, not what the underlying system should do to execute that code efficiently. By providing an abstraction for parallelism, Cilk Plus allows developers to express their parallel algorithms in a way that is more natural than many existing parallel libraries. Existing libraries for writing parallel programs such as TBB and OpenMP require the developer to restructure their application to break it into tasks that can be run in parallel. This restructuring can obscure the original algorithm, making it harder for the programmer to reason about and maintain the application. Because the Intel Cilk Plus keywords have serial semantics, developers can often express the parallelism inherent in their algorithm by adding only a few keywords to the original serial code."

Also, please refer to the article here could be helpful in general (especially the table listing the benefits vs caveats), although it needs to be updated to include OMP 4.0

With that said, if you are just interested "in the simple case of parallelizing a for loop where iterations are independent" I don't think there will be much performance difference between Cilk Plus and OpenMP, although as you may need to use dynamic scheduling with OpenMP to address thread imbalance, whereas Cilk Plus will take care of that automatically. In this simple case, no code restructuring is needed, but for either paradigm, you have to figure out where to insert the right pragmas for OMP, or `Cilk_for` or `Cilk_spawn` for Cilk. I would say the development effort is about the same for this simple case

Also, note that Cilk runtime uses work stealing scheduling unlike the OpenMP runtime. Generally work stealing helps in scenarios where there is an imbalance in the distribution of workload across all threads in action. I have attached a document which talks about two scenarios in specific that my peer gave me that he experimented with:

1. Case 1: Performance comparison of OMP parallel for and `_Cilk_for` in case of uniform workload across all 4 threads (OMP does better by

19

2. Case 2: Performance comparison of OMP parallel for and `_Cilk_for` in case of non-uniform workload across all 4 threads (Cilk does better by 18.5

In both the above cases my peer tried static, dynamic and guided scheduling in OpenMP runtime. Best result was found with dynamic scheduling.

Also note that even though OMP supports task parallelism via `#pragma omp task` and `#pragma omp sections`, it is harder to envision the task parallel architecture versus the abstraction offered by Cilk. And now with OMP 4.0, you can exploit instruction level data parallelism via `#pragma omp simd`, something which Cilk cannot do directly. But of course, Cilk tasks can take advantage of vectorization, much the same thing.

Regarding usage scenarios, well as long as the code contains candidates for data and task parallelism, Cilk Plus should do well. Refer to the C/C++ samples page at `http://software.intel.com/en-us/code-samples/intel-c-compiler` which lists some application domain samples using Cilk Plus which should give you some idea.

So, in a nutshell, I'd say start using Cilk Plus and let me know if you have any performance issues which I can file with our developers for investigation and resolution thereof?