THIS WEEK IN HPC: 10TH ANNIVERSARY OF INTEL THREADING BUILDING BLOCKS, WITH SPECIAL GUEST JAMES REINDERS

Addison Snell

July 2016

PODCAST

The following is a transcript from the weekly Intersect360 Research podcast, This Week in HPC*, available on iTunes, Stitcher, and through our media partnership with TOP500.org. The full podcast can be found at http://www.intersect360.com/industry/podcasts.php and is hosted at https://www.top500.org/news/10th-anniversary-of-intel-threaded-building-blocks-with-special-guest-james-reinders/.*

In this special podcast episode, originally published on July 7, 2016, host analyst Addison Snell interviews HPC and parallel computing expert James Reinders to discuss Intel's Threaded Building Blocks on its tenth anniversary.

Addison Snell: Hi everyone, thanks for listening in to another episode of This Week in HPC with Intersect360 Research. I'm Addison Snell, and for this special sponsored episode of This Week in HPC I'm joined by James Reinders, who is a parallel programming expert and HPC enthusiast from Intel. James, thanks for coming with me onto the program.

James Reinders: It's my pleasure, Addison. It's always a pleasure to talk with you.

AS: James you and I have known each other a long time around this industry, and knowing you gets ever more relevant because you're one of the guys who knows absolutely the most about what it takes to parallelize things. This has been increasingly important in our research because we're seeing new architectures, new programming models coming into the industry, and optimizing for that is one of the great challenges that end users have in front of them. Meanwhile Intel's got the tenth anniversary of Threading Building Blocks, or Intel TBB, coming onto the market. Can you talk about generally what that is, and what its role is in code modernization?

JR: Absolutely. Yeah, it's been a joy working in the area of parallel programming for decades now, watching it go from somewhat of an obscure topic that a few experts cared about to being very mainstream, very relevant for all programmers.

Intel Threading Building Blocks' coming out 10 years ago was fabulously timed to mesh with advent of multi-core processors – first coming on the market and then becoming dominant. In fact, you'd be hard-pressed to find the processor in anything these days with only one core. What TBB focused on was helping take C++ into the world of parallelism. That did a number of things but the most significant thing at the heart of it is what I would describe as a task-stealing scheduler. The emphasis there really is on breaking down the programming problem of parallelism so that a programmer is just describing opportunities for parallelism, and the runtime of TBB maps that onto the hardware. That's actually becoming more and more relevant as time goes on because the better we can maintain that abstraction, the more interesting things we can do with it. The programmer is not locking the solves into a particular number of cores or particular type of architecture, but rather describing a problem and how it can be broken into parallel pieces and letting the runtime – I like to say – map it onto the hardware, and then you have a lot of flexibility of the hardware that you map onto.

AS: Yes, you mentioned exactly the relevant thing when you were talking about TBB and its timing coming into the market with the advent of multi-core. Now we see multi-core increasing just on its own in terms of the number of cores. We have many-core elements that are available, people looking at FPGAs and other types of accelerators. How does TBB fit into that? Talk about how parallelism is evolving in the market.

"Parallelism is increasingly important, because we're seeing new architectures and new programming models coming into the industry, and optimizing for that is one of the great challenges that end users have in front of them."
– Addison Snell

"Parallelism is increasingly important, because we're seeing new architectures and new programming models coming into the industry, and optimizing for that is one of the great challenges that end users have in front of them."
– Addison Snell

JR: Absolutely. So you know the earliest uses of parallelism were most often marked with very regular problems, especially in scientific calculations, problems stated as a matrix: arrays of data that you process using matrix operations. That regularity is reflected very strongly in OpenMP directives, for instance, which are widely used by HPC programmers, especially for FORTRAN and C. One of the things that's evolving as the use of parallelism expands is there are a lot of more irregular problems that are sort of frustrating a model that's exclusively loop-oriented. TBB with its task stealing scheduler is perfectly poised to take advantage of that. TBB excels when a problem is not as regular. In fact, some of the early criticisms of TBB [came when] people would take a highly regular program that worked with OpenMP and notice that maybe TBB's overhead would reduce the performance slightly. But then TBB would take an irregular problem and make it much more efficient by moving it around on the resources.

Since then we have built on it. The open source community working on TBB has contributed a lot of things. One of them is a movement to support flow graphs. It's an idea that's very timely now but some folks might call it looking at the problem as a data flow problem, which is a phrase that was very common in the '80s that hasn't gone away, but we see parallelism being able to take advantage of that sort of thing's irregularities. If you start thinking about things as far as a data flow goes, or a flow graph, that flow may take you through different devices. Some of the portions of pipeline may be on a CPU, some of it may venture off on an FPGA or another device. It may be on a GPU as well. That flexibility is essentially built into the abstraction of TBB, [so] that it's just a matter of implementing the runtime system to say, "Hey, I can construct this flow to include specialty items up to and including an FPGA that may implement a specific function."

AS: What you're alluding to is diversity of workload or diversity of workflow with all these different types of applications. That's getting true now more than any time before, where we look at things like Big Data and analytics that have come into this space, Internet of Things, [and] machine learning, [which] was just a hot topic at ISC. How does TBB apply to all these new kinds of applications?

JR: It applies extremely well, because what we're seeing is an explosion in different ways to harness the hardware we have available. You're right, we're getting a diversity of hardware solutions explored at the same time that we're exploring spaces like machine learning. Machine learning, in many ways, the algorithms for it are in their infancy. There's a lot of innovation left. People [are] looking at questions of precision, finding some algorithms benefit from higher precision than may be being used now and some of them need less precision. Then they say, "Hey, we could really use a diversity of hardware solutions and what can map to those. What can handle that?" TBB is extraordinarily well structured to do that. The irregularity is great.

AS: We've got these different hardware solutions coming into the market now. Intel has FPGAs now that you're looking at, together with Xeon Phi, in addition to the base Xeon line. Then additionally there are other trends like Python coming into the market increasingly. We saw more Python in our most recent surveys that we've seen before. Does TBB continue to evolve with those trends?

JR: Absolutely, and a great example I was talking about is the dataflow capability. To someone that's used to doing problems say in signal processing space, they often think in terms of data flow and they may do one function and pipe it into another one. The diversity of hardware, the diversity of languages, even plays to that. TBB can even let you code one of your nodes, say, in OpenCL, or conceivably use the math functions we have like MKL or DAAL [Data Analytics Acceleration Library] or even hook Python into it, so you're not limited to one particular coding style. As you envision a pipeline, TBB sort of has the capability to operate at more of a what I call an orchestration level, to orchestrate the parallelism across the heterogeneous system, heterogeneous in terms of both hardware and software programming techniques. And I think we're going to see a lot of that because as these different fields explode people will explore many different ways, looking for what they can accomplish using different tools in different techniques.

AS: You mentioned MKL. Presumably you see a big advantage of using TBB over other threading standards. Do have concrete examples of that?

JR: Well, you know the first places that TBB took hold very firmly is in digital media: One of our first customers was doing the Maya program. We also branched out and had Pixar and Dreamworks and other companies like that find uses for it. Dreamworks Animation had quite a few different articles and even was a lead on a book I participated in on doing multithreading for visual effects. So the visual effects have been very effective with TBB, putting tools together for animators.

Since then we've seen it branch out to other uses of parallelism in the medical manufacturing field, as people look at implementing new algorithms, again especially if they're irregular, or if they're not just a matrix operation, if they're graph-oriented and so forth. TBB has been a very flexible platform for getting high performance. In fact, we initially supported the matrix sort of things pretty well. The flow graph is newer, and some of our customers have transitioned with us – have started on the more regular things that TBB was good at and have moved to the irregular problems. That's been very productive for them, because that's given them a lot more flexibility in their programming style and still maps onto parallel hardware.

AS: Our most recent data show that 88% of HPC users think they're going to be supporting more than one type of processing architecture for HPC over the next couple of years going in the future. So, TBB, is this something that's going to bring commonality to the programming experience across these multiple architectures?

JR: Absolutely. TBB is, for C++ programmers, a very strong tool for doing exactly what you're describing, because it not only helps you with the programming-specific parts of your application, to use parallelism, but it can orchestrate the use of multiple specialty devices so that you can specialize in calling for, say, MKL, for the work you're going to do, perhaps offload some of it to an FPGA if that's appropriate, or a

GPU. It can handle all of that under one umbrella, and it's open-source and has a very large following and community to participate with.

AS: You just updated that licensing model, didn't you?

JR: Yeah. Licensing's always a fun topic. When we introduced it, we decided to go with the then most-common license: we went with the GPLv2 with the class-path exception, which is a fancy way of saying it works in C++. It is not viral to the application that uses it; is the intent of that license. We also had a commercial version that didn't come with source code, for those that had concerns about using GPL. But we've had a lot of feedback, and industry has changed and shifted, and the popularity of non-viral licenses has taken a firm hold as well. [We had] a lot of feedback from customers that they'd like to see us move to a non-viral license like BSD or Apache and so forth.

After a lot of consulting, this year we're moving TBB to the Apache license, which, as I understand it – I'm not a lawyer – but I would call it a non-viral license that also has some provisions about patents being not asserted through the license. You'd have to go read the license to understand that, but I understand that to be one of the reasons our customers encouraged us to go to Apache and not BSD.

AS: We've been speaking with James Reinders, parallel programming expert and HPC enthusiast from Intel, about the 10th anniversary of Intel Threading Building Blocks, or TBB. James, thanks very much for coming on the podcast.

JR: My pleasure, thank you.

AS: And thanks for listening in. You've been listening to *This Week in HPC*.