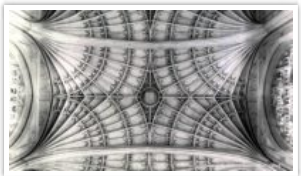
WILL CHAPEL MARK NEXT GREAT AWAKENING FOR PARALLEL PROGRAMMERS?

April 10, 2018 Nicole Hemsoth



Just because supercomputers are engineered to be far more powerful over time does not necessarily mean programmer productivity will follow the same curve. Added performance means more complexity, which for parallel programmers means working harder, especially when accelerators are thrown into the mix.

It was with all of this in mind that DARPA's High Productivity Computing Systems (HPCS) project rolled out in 2009 to support higher performance but more usable HPC systems. This is the era that spawned systems like Blue Waters, for instance, and various co-design efforts from IBM and Intel to make parallelism within broader reach to the wider technical computing base.

At the time when HPCS was getting traction, Brad Chamberlain, the lead architect for the Chapel language effort and a new employee at supercomputer maker, Cray, noticed an opportunity somewhere in between the much higher-level interfaces and down-low approaches like OpenMP, MPI, and as GPUs crept into more supercomputing centers, CUDA. This was before HPC domain scientists were using Python and at a time and during a decades-long stretch where HPC programming was defined by strict modes, most of them very low-level.

"We had then—and now—a mission to grow the HPC market, but if the community is saying that to work in HPC you have to learn C, C++, Fortran, MPI, OpenMP, and CUDA and this is the only way things are done, that sure seems to shut out a huge swath of the modern programming community who are already well versed in Python, Matlab and tools that are more productive and modern," he tells *The Next Platform*. "For HPC to say it is not willing to meet this group along the spectrum is part of the reason HPC has not grown much and it is the reason we have a hard time recruiting and retaining the top software engineers in the field."

"The original goal was to make parallel programming less mechanical, low level, and bottom-up and focus on what people in HPC needed to express. This meant an emphasis on parallelism, locality, and designing from the top-down with higher level abstractions to help users write their codes and move them from a laptop to a cluster to a more specialized Cray supercomputer.

That final point begs an important question—and one that has been the source of confusion since Chapel emerged, even though Chamberlain and his Chapel team manager, Ben Robbins, tell us they constantly make clear. Even though Cray supports the fourteen-person Chapel development team (a big team by HPC development standards but miniscule compared to other mainstream language dev efforts), the platform is not intended to run specifically on Cray supercomputers. It has been designed to be portable, but Chamberlain says of course, when it comes to taking advantage of Cray's custom network and other features in the Cray Linux Environment there are benefits to Chapel that can be wrought elsewhere. Still, he says the need for some middle ground in parallel programming for HPC is still needed.

Chapel supports a multithreaded execution model via high-level abstractions for data parallelism, task parallelism, concurrency, and nested parallelism. Chapel's locale type enables users to specify and reason about the placement of data and tasks on a target architecture in order to tune for locality and affinity. Chapel supports global-view data aggregates with user-defined implementations, permitting operations on distributed data structures to be expressed in a natural manner. In contrast to many previous higher-level parallel languages, Chapel is designed around a multiresolution philosophy, permitting users to initially write very abstract code and then incrementally add more detail until they are as close to the machine as their needs require. Chapel supports code reuse and rapid prototyping via object-oriented design, type inference, and features for generic programming. Existing code can be integrated into Chapel programs (or vice-versa) via interoperability features.

THE NEXT PLATFORM WEEKLY

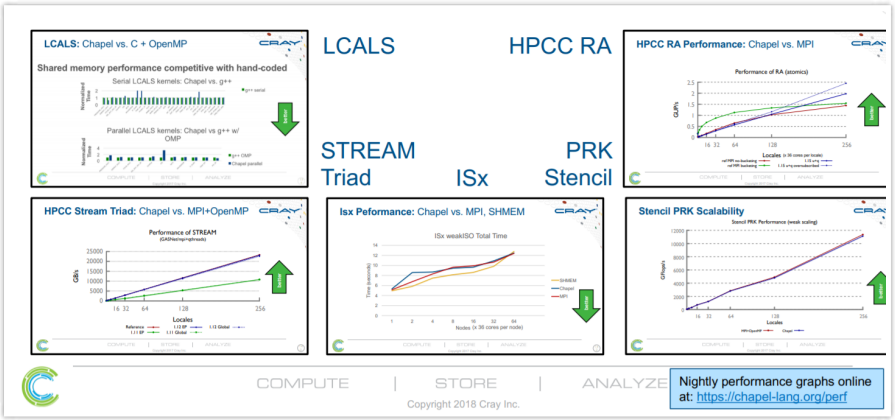
Tap the stack to painlessly subscribe for a weekly email from The Next Platform, featuring highlights, analysis, and stories from the week directly from us to your inbox with nothing in between.

The thing is, some could argue there is already a middle ground language for HPC. The Julia language appeared after Chapel, emerging in 2012. While they might seem to be targeting the same set of HPC users (which they are) the rationale is divided. Julia and Chapel have some of the same aims but they are coming from different worlds—and this makes all the difference, Chamberlain says. "We are both trying to be the 'Python for HPC' but Julia takes the approach of, 'you're already a Python programmer? Let's bring you into our world of HPC,' whereas Chapel is designed from the bottom up to talk to that base of existing HPC parallel programmers that see what they need to control in terms of a machine's resources and how to map efficiently and want to build from higher level abstractions from there," he says. "Most HPC programmers who have dug into Chapel understand there are some high level programmability features but they still want to be able to map down to the machine more efficiently."

In other words, Julia can be considered a bit of an outsider with deeper mainstream roots than Chapel, which is that HPC specific middle ground. Both support GPUs, both provide more flexibility and scalability of performance than Python, but both have their own starting point—something that Chamberlain says is a feature, not a bug.

"We want Chapel to be as accessible and readable as Python is; it should be easy to look at a bit of code and see what's happening pretty quickly, but all the while addressing things in Python that aren't a good fit for HPC and provide better performance. Fortran and MPI are the metrics people expect for scalable performance and portability is also key so we wanted to have a feature-rich language that also has all thing things one would expect from a C++, C#, Java, or Swift, for instance," Chamberlain explains.

The Chapel team has upped the benchmark ante in 2018 with some new graphs highlighting solid performance on a number of HPC metrics, including those below (these are from 2018).



*Chapel vs. MPI/OpenMP performance in recent talks, demonstrating that Chapel is increasingly competitive for local computations (LCALS), embarrassingly parallel computations (stream), random fine-grain communication (RA), stencil communications (Stencil PRK), and bucket exchange patterns (ISx). The distributed memory timings here are on 2×18-core compute nodes (so 256 locales = 256*36 cores)*

Cray also does performance testing nightly to put the results into the public to be used interactively much like a stock ticker—and like a stock ticker, these can be challenging to draw good conclusions from because in addition to Chapel improving or getting worse on a given night, the benchmark itself could be modified, the underlying system or its software could be updated, etc.. Even still, it provides one way to see some rough trends over a longer period of time. There's an "annotation" button under each graph which the team uses to label significant changes that have been diagnosed. As one example, the following link shows some trends over time for 16 nodes of a Cray XC supercomputer.

Good results, but there will always be those in HPC that want to stick as close to the machine as possible. "If you're happy programming with C++, MPI, OpenMP and CUDA you probably don't need Chapel per se. There are programmers in the DoE who work with those tools regularly but want something more like Python to program in," says Ben Robbins, the general manager for Chapel. "Think about the days of Microsoft and Visual Basic—the reason it was so popular was because it was productive; people no longer needed to stay in an elitist band of programmers.

We asked Chamberlain what Cray gets out of supporting what is likely a rather expensive effort with the full time team needed, especially for an open source "product" that works for its users but has other uses outside of Cray if it takes off. His answer was simply that Cray is interested in anything that might make the HPC community larger—or bring more system sales to Cray from outside of traditional HPC or even inside where broader bases of non-parallel programming specialists reside. "It would be the death of any language to only run one system," Chamberlain explains, adding that their implementation is designed to run anywhere with C compiler or LLVM with threads and way to communicate with either Cray's own custom network, or more broadly, InfiniBand and Ethernet.

As with all things open source, it is not simple to get a sense of how wide a platform's user base is. The code itself has over 50,000 commits on Github but that's over a long span and does not

indicate real world use. Clearly there is enough traction to keep Cray investing and Chamberlain insists that the last 18 months in particular have shown remarkable performance jumps—an important bit of news considering performance was still not there for some of their earlier users when compared to more traditional approaches. This is no surprise as with any layer of abstraction's performance overhead, but the real question is what HPC developers are wiling to live with.

"We have historically been geared toward HPC programmers because as Cray that is who we most interacted with—the big DoE labs and such. But as parallel computing has gone mainstream the number of people showing an interest in Chapel, especially those who are not already Cray customers, is striking. They are the subset who are not satisfied with the status quo in HPC," says Robbins.

Chamberlain adds that the real challenge as more people from outside traditional large-scale HPC see Chapel as a way to get high performance scalability on bigger clusters is to keep Chapel as feature rich as the other languages but with all that parallelism, scalability and locality built in. This is harder than it sounds he says but they are finally at the point where their benchmark results are showing capabilities that can rival MPI—a significant achievement.

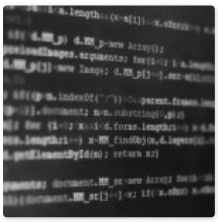
Pardon the play on words, but for any chapel to serve its purpose, it needs plenty of followers to keep it full and vibrant. HPC is still committed to its lower level tools and that will remain the case with domain scientists dabbling in Python until it fails to scale. This seems to clear the way for either Julia or Chapel, but for HPC centers, the "P" tends to always stand for performance and not productivity—something that both parallel programming upstarts need to keep tweaking to be the answer to HPC programmer pleas.

SIMILAR VEIN

Dirt Simple HPC: Making the Case for Julia

Julia Language Delivers Petascale HPC Performance

OpenMP Has More in Store for GPU Supercomputing



Programming Toward Exascale: A Look Ahead at OpenACC in 2016

OpenMP: From Parallel Loops To Exaflops

Turning OpenMP Programs into Parallel Hardware

Categories: Code, HPC

Tags: Chapel, Julia, OpenMP

Riding the AI Cycle Instead of Building It

Talking Up the Expanding Markets for GPU Compute

ONE THOUGHT ON "Will Chapel Mark Next Great Awakening for Parallel Programmers?"

Cristian Vasile says: April 10, 2018 at 1:12 pm

Other approach to HPC programming could use some strange named projects like Graal, Truffle, Sulong, Substrate VM etc. The main idea is not to re-create the new Python or Java for HPC, but reuse what has been developed and is battle tested.

In near future JavaScript, Julia, R, Fortan, C & C++ code will happily run on thin Java VM, inside Singularity's containers.

Reply

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

PAGES

About
Contact
Contributors
Newsletter

RECENT POSTS

Feeding The Insatiable Bandwidth Beast
Playing Dominoes In Data Science
Sluggish Moore's Law Doesn't Impede Intel One Bit
The Slow But Sure Return Of AMD In The Datacenter
AI Software Writing AI Software For Healthcare?