News

Press Releases
Event Announcements
CCT Weekly
Grants and Funding
Student News
Archived News

Is ParalleX This Years Model?

(Source: IT Business Net)

Scientific application developers have masses of computing power at their disposal with today's crop of high-end machines and clusters. The trick, however, is harnessing that power effectively. Earlier this year, Louisiana State University's Center for Computation & Technology (CCT) released its approach to the problem: an open-source runtime system implementation of the ParalleX execution model. ParalleX aims to replace, at least for some types of applications, the Communicating Sequential Processes (CSP) model and the well-established Message Passing Interface (MPI), a programming model for high-performance computing. The runtime system, dubbed High Performance ParalleX (HPX) is a library of C++ functions that targets parallel computing architectures. Hartmut Kaiser -- lead of CTT's Systems Technology, Emergent Parallelism, and Algorithm Research (STE||AR) group and adjunct associate research professor of the Department of Computer Science at LSU -- recently discussed ParalleX with Intelligence in Software.

Q: The HPX announcement says that HPX seeks to address scalability for "dynamic adaptive and irregular computational problems." What are some examples of those problems?

Hartmut Kaiser: If you look around today, you see that there's a whole class of parallel applications -- big simulations running on supercomputers -- which are what I call "scaling-impaired." Those applications can scale up to a couple of thousand nodes, but the scientists who wrote those applications usually need much more compute power. The simulations they have today have to run for months in order to have the proper results.

One very prominent example is the analysis of gamma ray bursts, an astrophysics problem. Physicists try to examine what happens when two neutron stars collide or two black holes collide. During the collision, they merge. During that merge process, a huge energy eruption happens, which is a particle beam sent out along the axis of rotation of the resulting star or, most often, a black hole. These gamma ray beams are the brightest energy source we have in the universe, and physicists are very interested in analyzing them. The types of applications physicists have today only cover a small part of the physics they want to see, and the simulations have to run for weeks or months.

And the reason for that is those applications don't scale. You can throw more compute resources at them, but they can't run faster. If you compare the number of nodes these applications can use efficiently -- an order of a thousand -- and compare that with the available compute power on high-end machines today -- nodes numbering in the hundreds of thousands, you can see the frustration of the physicists. At the end of this decade, we expect to have machines providing millions of cores and billion-way parallelism.
The problem is an imbalance of the data distributed over the computer. Some parts of a simulation work on a little data and other parts work on a huge amount of data.

Another example: graph-related applications where certain government agencies are very interested in analyzing graph data based on social networks. They want to analyze certain behavioral patterns expressed in the social networks and in the interdependencies of the nodes in the graph. The graph is so huge it doesn't fit in the memory of a single node anymore. They are imbalanced: Some regions of the graph are highly connected, and some graph regions are almost disconnected between each other. The irregularly distributed graph data structure creates an imbalance. A lot of simulation programs are facing that problem.

Q: So where specifically does CSP and MPI run into problems?

H.K.: Let's try to do an analogy as to why these applications are scaling-impaired. What are the reasons for them to not be able to scale out? The reason, I believe, can be found in the "four horsemen": Starvation, Latency, Overhead, and Waiting for contention resolution -- slow. Those four factors are the ones that limit the scalability of our applications today.

If you look at classical MPI applications, they are written for timestep-based simulation. You repeat the timestep evolution over and over again until you are close to the solution you are looking for. It's an iterative method for solving differential equations. When you distribute the data onto several nodes, you cut the data apart into small chunks, and each node works on part of the data. After each timestep, you have to exchange information on the boundary between the neighboring data chunks -- as distributed over the nodes -- to make the solution stable.

The code that is running on the different nodes is kind of in lockstep. All the nodes do the timestep computation at the same time, and then the data exchange between the nodes happens at the same time. And then it goes to computation and back to communication again. You create an implicit barrier after each timestep, when each node has to wait for all other nodes to join the communication phase. That works fairly well if all the nodes have roughly the same amount of work to do. If certain nodes in your system have a lot more work to do than the others -- 10 times or 100 times more work -- what happens is 90 percent of the nodes have to wait for 10 percent of the nodes that have to do more work. That is exactly where these imbalances play their role. The heavier the imbalance in data distribution, the more wait time you insert in the simulation.

That is the reason that MPI usually doesn't work well with very irregular programs, more concretely -- you will have to invest a lot more effort into the development of those programs -- a task not seldom beyond the abilities of the domain scientists and outside the constraints of a particular project. You are very seldom able to evenly distribute data over the system so that each node has the same amount of work, or it is just not practical to do so because you have dynamic, structural changes in your simulation.

I don't want to convey the idea that MPI is bad or something not useful. It has been used for more than 15 years now, with high success for a certain class of simulations and a certain class of applications. And it will be used in 10 years for a certain class of applications. But it is not well-fitted for the type of irregular problems we are looking at.

ParalleX and its implementation in HPX rely on a couple of very old ideas, some of them published in the 1970s, in addition to some new ideas which, in combination, allow us to address the challenges we have to address to utilize today's and tomorrow's high-end computing systems: energy, resiliency, efficiency and -- certainly -- application scalability. ParalleX is defining a new model of execution, a new approach to how our programs function. ParalleX improves efficiency by exposing new forms of -- preferably fine-grain -- parallelism, by reducing average synchronization and scheduling overhead, by increasing system utilization through full asynchrony of workflow, and employing adaptive scheduling and routing to mitigate contention. It relies on data-directed, message-driven computation, and it exploits the implicit parallelism of dynamic graphs as encoded in their intrinsic metadata. ParalleX prefers methods that allow it to hide latencies -- not methods for latency avoidance. It prefers "moving work to the data" over "moving data to the work," and it eliminates global barriers, replacing them with constraint-based, fine-grain synchronization techniques.

Q: How did you get involved with ParalleX?

H.K.: The initial conceptual ideas and a lot of the theoretical work have been done by Thomas Sterling. He is the intellectual spearhead behind ParalleX. He was at LSU for five or six years, and he left only last summer for Indiana University. While he was at LSU, I just got interested in what he was doing and we started to collaborate on developing HPX.

Now that he's left for Indiana, Sterling is building his own group there. But we still tightly collaborate on projects and on the ideas of ParalleX, and he is still very interested in our implementation of it.

Q: I realize HPX is still quite new, but what kind of reception has it had thus far? Have people started developing applications with it?

H.K.: What we are doing with HPX is clearly experimental. The implementation of the runtime system itself is very much a moving target. It is still evolving.

ParalleX -- and the runtime system -- is something completely new, which means it's not the first-choice target for application developers. On the other hand, we have at least three groups that are very interested in the work we are doing. Indiana University is working on the development of certain physics and astrophysics community applications. And we are collaborating with our astrophysicists here at LSU. They face the same problem: They have to run simulations for months, and they want to find a way out of that dilemma. And there's a group in Paris that works on providing tools for people who write code in MATLAB, a high-level toolkit widely used by physicists to write simulations. But it's not very fast, so the Paris group is writing a tool to covert MATLAB to C++, so the same simulations can run a lot faster. They want to integrate HPX in their tool.

ParalleX and HPX don't have the visibility of the MPI community yet, but the interest is clearly increasing. We have some national funding from DARPA and NSF. We hope to get funding from the Department of Energy in the future; we just submitted a proposal. We expect many more people will gain interest once we can present more results in the future.

Publish Date:
03-14-2012

Center for Computation & Technology
2003 Digital Media Center • Telephone: +1 225/578-5890 • Fax: +1 225/578-8957