Programming a Parallel Future

Joseph M. Hellerstein

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-144
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-144.html

November 7, 2008

Programming a Parallel Future

Joe Hellerstein
UC Berkeley Computer Science

Things change fast in computer science, but odds are good that they will change especially fast in the next few years. Much of this change centers on the shift toward parallel computing. In the short term, parallelism will thrive in settings with massive datasets and analytics. Longer term, the shift to parallelism will impact all software. In this note, I'll outline some key changes that have recently taken place in the computer industry, show why existing software systems are ill-equipped to handle this new reality, and point toward some bright spots on the horizon.

Technology Trends: A Divergence in Moore's Law

Like many changes in computer science, the rapid drive toward parallel computing is a function of technology trends in hardware. Most technology watchers are familiar with Moore's Law, and the general notion that computing performance per dollar has grown at an exponential rate — doubling about every 18-24 months — over the last 50 years. But recently, Moore's Law has been pushing different aspects of computing in different directions, and this divergence is substantially changing the rules for computer technology.

Moore's Law and exponential growth may be familiar, but they are so phenomenal they deserve a pause for consideration. In common parlance, the term "exponential" is often used as slang for "very fast." But when computer scientists talk about something exponentiating, they tend to use more colorful phrases: *blowing up* or *exploding*. The storage industry has been a continual example of this, delivering exponentially increasing storage density per dollar over time. In concrete terms, consumers today might have a few Terabyte disks in the basement for home movies and laptop backups — after all, they only cost about $150 at the local office supply store[1]. But only 10 years ago, when Google was starting out, that was roughly the storage capacity of the *largest databases on the planet*.

If all aspects of computer hardware followed Moore's Law together, software developers could simply enjoy increasing performance without changing their behavior much. This has been true, more or less, for decades now. The reason things are changing so radically now is that *not all of the computing substrate is*

[1] Given exponentially decreasing prices per byte, that comment nearly instantly makes this note obsolete!

exponentiating in the same way. And when things "blow up" in different directions, the changes are swift and significant.

To explain, recall that what Moore's Law actually predicts is the number of transistors that can be placed on an integrated circuit. Until recently, these extra transistors had been used to increase CPU speed. But, in recent years, limits on heat and power dissipation have prevented computer architects from continuing this trend. So although Moore's Law continues in its strict definition, *CPUs are not getting much faster,* even though many other computing trends – RAM sizes, disk density, network bandwidth – continue at their usual pace.

Rather than providing speed improvements, the extra transistors from Moore's Law are being used to pack more CPUs into each chip. Most computers being sold today have a single chip containing between 2 and 8 processor "cores." In the short term, this still seems to make our existing software go faster: one core can run operating systems utilities, another can run the currently active application, another can drive the display, and so on. But remember, Moore's Law continues doubling every 18 months. That means your laptop in nine years will have 128 processors, and a typical corporate rack of 40-odd computers will have something in the neighborhood of 20,000 cores. Parallel software should, in principle, take advantage not only of the hundreds of processors per machine, but of the entire rack — even an entire datacenter of machines.

What does this mean for software? Since individual cores will not get appreciably faster, *we need massively parallel programs that can scale up with the increasing number of cores, or we will effectively drop off of the exponential growth curve of Moore's Law***.** Unfortunately, the large majority of today's software is written for a single processor, and there is no technique known to "auto-parallelize" these programs. Worse yet, this is not just a "legacy software" problem. Programmers still find it notoriously difficult to reason about multiple, simultaneous tasks in the parallel model, which is much harder for the human brain to grasp than writing "plain old" algorithms. So this is a problem that threatens to plague even new, green-field software projects.

This is the radically new environment facing software technologists over the coming years. Things will have to change substantially for progress to continue.

Bright Spots: SQL, Big Data and MapReduce

To look for constructive ways forward, it helps to revisit past successes and failures. Back in the 1980's and 1990's, there were big research efforts targeting what was then termed "massively parallel" computing, on dozens and sometimes hundreds of processors. Much of that research targeted computationally intensive scientific applications, and spawned startup companies like Thinking Machines, Kendall Square and Sequent. The challenge at that time was to provide programming languages and models that could take a complex computational task, and tease apart separate pieces that could run on different processors. In rough terms, that burst of

energy did not have much short-term success: very few programmers were able to reason in the programming models that were developed, and the various startup companies mostly failed or were quietly absorbed.

Parallel Databases and SQL

But one branch of parallel research from that time did meet with success: parallel databases. Companies like Teradata – and research projects like Gamma at Wisconsin and Bubba at MCC – demonstrated that the Relational data model and SQL lend themselves quite naturally to what is now called "data parallelism." Rather than requiring the programmer to unravel an algorithm into separate threads to be run on separate cores, parallel databases let them chop up the input data tables into pieces, and pump each piece through *the same single-machine program* on each processor. This "parallel dataflow" model makes programming a parallel machine as easy as programming a single machine. It also works on "shared-nothing" clusters of computers in a datacenter: the machines involved can communicate via simple streams of data messages, without a need for an expensive shared RAM or disk infrastructure.

The parallel database revolution was enormously successful, but at the time it only touched a modest segment of the computing industry: a few high-end enterprise data warehousing customers who had unusually large-scale data entry and data capture efforts. But this niche has grown radically in recent years, and the importance of these ideas is being rethought.

Big Data

We're now entering what I call the "Industrial Revolution of Data," where the vast majority of data will be *stamped out by machines*: software logs, cameras, microphones, RFID readers, wireless sensor networks, and so on. These machines generate data a lot faster than people can, and their production rates will grow exponentially with Moore's Law. Storing this data is cheap, and it can be mined for valuable information.

This trend has convinced nearly everyone in computing that the next leaps forward will revolve around what many computer scientists have dubbed "Big Data" problems. In previous years this focus was largely confined to the database field, and SQL was a popular language to tackle the problem. But a broader array of developers are now interested, and they want to wrangle their data in typical programming languages, rather than use SQL, which they often find unfamiliar and restrictive.

MapReduce

The MapReduce programming model has turned a new page in the parallelism story. In the late 1990s, the pioneering web search companies built new parallel software infrastructure to manage web crawls and indexes. As part of this effort, they were forced to reinvent a number of ideas from parallel databases — in part because the commercial database products at the time did not handle their workload well. Google developed a business model that depended not on crawling and indexing per

se, but instead on running massive analytics tasks over their data, e.g., to ensure that advertisements were relevant to searches so that ads could be sold for measurable financial benefit. To facilitate this kind of task, they implemented a programming framework called MapReduce, based on two simple operations from functional programming languages.

The Google MapReduce framework is a parallel dataflow system that works by partitioning data across machines, each of which runs the same single-node logic. But unlike SQL, MapReduce largely asks programmers to write traditional code, in languages like C, Java, Python, and Perl, whereas SQL provides a higher-level language that is more flexible and optimizable, but less familiar to many programmers. In addition to its familiar syntax, MapReduce allows programs to be written to and read from traditional files in a filesystem, rather than requiring database schema definitions.

Q: SQL or MapReduce? A: Yes.

Technically speaking, SQL has some advantages over MapReduce, including easy ways to combine multiple data sets, and the opportunity for deeper code analysis and just-in-time query optimizations. In this context, one of the most exciting developments on the scene is the emergence of platforms that provide both SQL and MapReduce interfaces within a single runtime environment. These are especially useful when they support parallel access to both database tables and filesystem files from *either* language. Examples of these frameworks include the commercial Greenplum system (which provides all of the above), the commercial Aster Data system (which provides SQL and MapReduce over database tables), and the open-source Hive framework from Facebook (which provides an SQL-like language over files, layered on the open-source Hadoop MapReduce engine.) DryadLINQ from Microsoft Research is another interesting design point in this space that merges a SQL-like syntax with a MapReduce-style parallel dataflow.

MapReduce has brought a new wave of excited, bright developers to the challenge of writing parallel programs against Big Data. This is critical: a revolution in parallel software development can only be achieved by a broad base of enthusiastic, productive programmers. SQL also has legions of experienced programmers, and integration with a wide variety of software tools and frameworks. The new combined platforms for data parallelism expand the options for these programmers, and should foster synergies between the SQL and MapReduce communities. There are interesting days ahead for massively parallel programs against Big Data – which is good news for the future of Moore's Law and computing in general.

On the Commercial Front

In the context of the previous discussion, I should note that I am an advisor at Greenplum, and strongly encouraged their development of a MapReduce interface for many of the reasons listed above. I was pleasantly surprised to learn of Aster's entry into that space at the same time Greenplum made their first public

announcements. Meanwhile, I am in close touch with friends and colleagues in the Hadoop world, and the various companies extending that work.

For the moment, these platforms have various distinguishing features that differentiate them in the marketplace. But these are systems in quick evolution, and there will undoubtedly be a lot of change in this realm in upcoming months and years. In addition to evolution from the first set of players, we can expect the large enterprise players like Oracle, IBM and Microsoft to make moves in this space if it can be shown to be lucrative. The role of the web search and cloud hosting companies will be interesting to watch as well, since they have the expertise and infrastructure to host large MapReduce jobs.

If the startups grow and the large players follow, this will only fuel the fire for data-centric parallel programming. That in turn will drive the training of more and more programmers to think about parallel programs on Big Data. And once the programmers are on board in large numbers, bigger and more unexpected changes may emerge.

What Is Computer Science Doing About It?

The computing industry is actively pursuing parallel data management with both SQL and MapReduce frameworks. And the pipeline is filling from the academic front as well.

In terms of education, MapReduce is such a compelling entryway into parallel programming, it is being used to nurture a new generation of parallel programmers. Every Berkeley CS freshman now learns MapReduce. Other schools have undertaken similar programs, and a consortium of companies is eagerly supporting these efforts with shared computing platforms, curriculum development, and support of the Hadoop open-source backend. This is a great example of an academic/industrial collaboration working toward a common goal.

But it's the news on the research front that I find most intriguing. The focus on data-parallelism today surrounds Big Data, and that is an important application domain. But what about the rest of the software industry? How will parallel computing transcend Big Data problems, so that broad classes of software can leverage multicore and cluster parallelism?

This is a hard problem, but based on research in the last 5-10 years, I am optimistic that academic computer science can play a leadership role here. In recent years, the data-centric approach to programming exemplified by SQL and MapReduce has been gaining footholds well outside of batch-oriented data parallelism. There has been a groundswell of work on "declarative", data-centric languages for a variety of domain-specific tasks, mostly using extensions of Datalog, a formal language popular with theoreticians. These new data-centric languages have been popping up in networking and distributed systems, natural language processing, compiler analysis, modular robotics, security, machine learning, and video games, among

other applications. And they are being proposed for tasks that are not embarrassingly parallel. It turns out that focusing on the data can make a broad class of programs simpler — much simpler! — to express.

As one example from my research group at Berkeley, our version of the Chord Distributed Hash Table (DHT) is 47 lines of our Overlog language; the reference implementation is over 10,000 lines of C++. (DHTs are a key component of cloud services like Amazon's Dynamo). That is the kind of scenario where the quantitative difference is best captured qualitatively. You can print out our Chord implementation on one sheet of paper, take it down to the coffee shop, and figure it out. Doing that with 10,000 lines of C++ would be a superhuman feat of Programmer-Fu, and a big waste of paper. Now, Overlog is a fairly academic language, but we are following it up with a much more complete language called Lincoln that is targeted at a much wider range of programmers. Other such languages are under development in various groups.

Things to Watch

It is easy to see the need for new parallel programming approaches, but hard to envision where and how the next generation of big ideas will play out. Here are some things I am keeping an eye on:

- **MapReduce Extensions and Integration.** I am not a big fan of the specifics of Google's MapReduce, nor of the cloning of that model in the open-source Hadoop framework. Its biggest drawback is the need to stage data to disk over and over, which prevents it from providing real-time feedback, and dooms it to poor overall performance. (It is not only slower than it should be, it is an enormous energy hog, which should even bother resource-rich companies like Google and Yahoo). But I do not view MapReduce as a fixed target. Most of the companies I talk to using Hadoop have modified it significantly, and there are a number of languages layered on top including Yahoo!'s Pig, IBM's JAQL, and Facebook's Hive. New MapReduce implementations like those of Greenplum and Aster Data will presumably remove some of these design roadblocks, and open up interesting integrations with database technology. MapReduce is a movement, not an artifact, and the landscape there is likely to change substantially in the next months and years.
- **The Next Programming Language(s).** The need for Parallelism opens the door for a new programming language. MapReduce is almost certainly too simple, and SQL too cumbersome to be a broadly useful language. If the next popular language has to provide parallelism, what will it look like? I am placing a bet on data-centric declarative languages like our new effort, Lincoln. But these have yet to have their day in the sun, and there are certainly plenty of fans of other paradigms. Given that languages succeed for both technical and non-technical reasons, how will this space get staked out?

- **Cloud Computing.** One of the major challenges in exploiting parallelism is to make improvements in legacy software, including operating systems, desktop applications, games etc. The emergence of software in the cloud is intriguing in its timing, because it provides an opportunity to rewrite everything from scratch. Shouldn't this new platform be programmed for parallelism, both at its core and in its applications? That's not really the case today. Are the programs being written today for early cloud platforms doomed to irrelevance because they predate the next parallel language? Is this an opportunity to leapfrog the early movers in the Cloud Computing space?
- **Data and Statistics.** Data volumes will continue to grow as never before; everybody will want to turn information into value. As part of that evolution, statisticians, machine learning experts, and other data analysts will play an increasingly important role in effective organizations, and will need to be skilled at handling very big data sets. The practice of "Data Mining" arose with roughly this premise, but the research has grown increasingly sophisticated, and I expect this sophistication to become *much* more widespread in industry over the coming years. Practitioners who can master this combination of skills will be highly valued, as will the tools that they embrace for their work.

About Joe Hellerstein

Joseph M. Hellerstein is a Professor of Computer Science at the University of California, Berkeley, whose research focuses on data management and networking. His work has been recognized via awards including an Alfred P. Sloan Research Fellowship, MIT Technology Review's inaugural TR100 list, and two ACM-SIGMOD "Test of Time" awards. Key ideas from his research have been incorporated into commercial and open-source database software released by IBM, Oracle, and PostgreSQL. He has also held industrial posts including Director of Intel Research Berkeley, and Chief Scientist of Cohera Corporation.