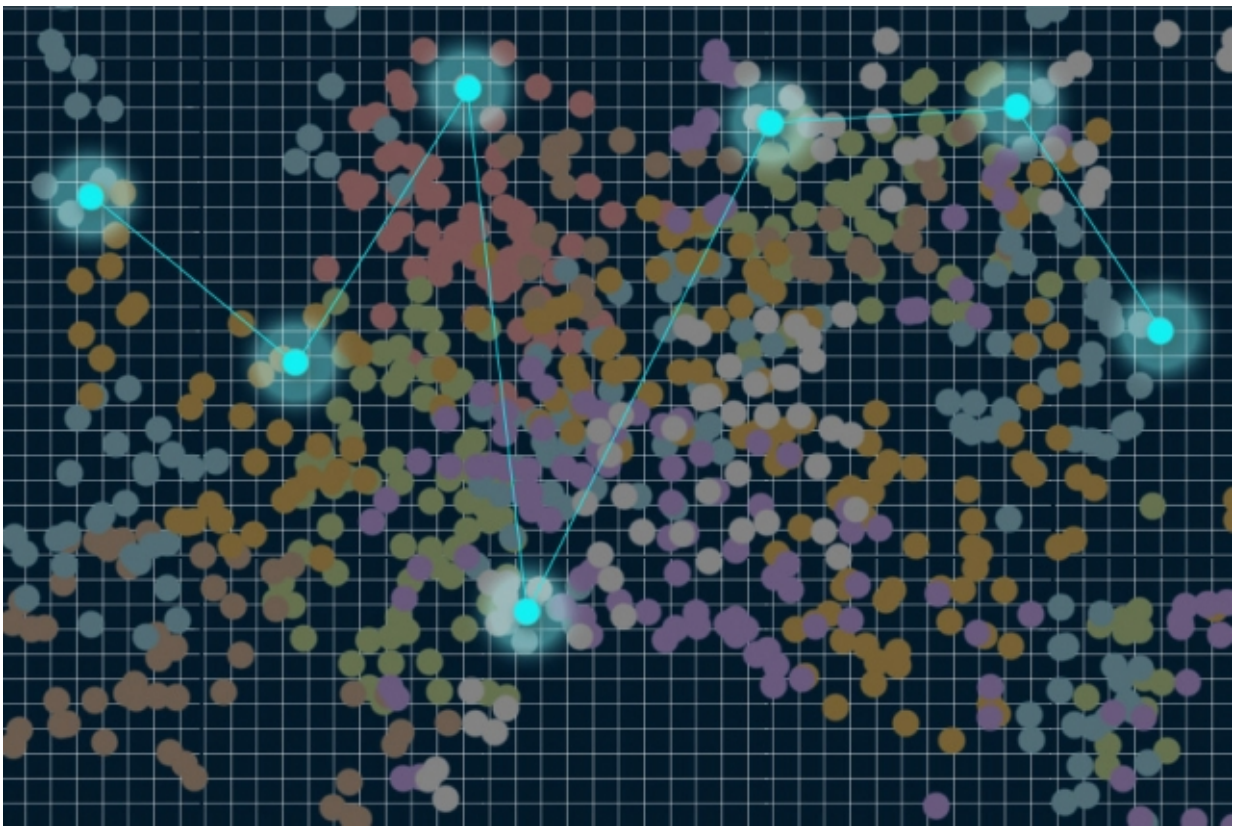
New programming language delivers fourfold speedups on problems common in the age of big data

September 13 2016, by Larry Hardesty



Researchers have designed a new programming language that lets application developers manage memory more efficiently in programs that deal with scattered data points in large data sets. In tests on several common algorithms, programs written in the new language were four times as fast as those written in existing languages. Credit: Christine Daniloff/MIT

In today's computer chips, memory management is based on what computer scientists call the principle of locality: If a program needs a chunk of data stored at some memory location, it probably needs the neighboring chunks as well.

But that assumption breaks down in the age of big data, now that computer programs more frequently act on just a few data items scattered arbitrarily across huge data sets. Since fetching data from their main memory banks is the major performance bottleneck in today's chips, having to fetch it more frequently can dramatically slow program execution.

This week, at the International Conference on Parallel Architectures and Compilation Techniques, researchers from MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) are presenting a new programming language, called Milk, that lets application developers manage memory more efficiently in programs that deal with scattered data points in large data sets.

In tests on several common algorithms, programs written in the new language were four times as fast as those written in existing languages. But the researchers believe that further work will yield even larger gains.

The reason that today's big data sets pose problems for existing memory management techniques, explains Saman Amarasinghe, a professor of electrical engineering and computer science, is not so much that they are large as that they are what computer scientists call "sparse." That is, with big data, the scale of the solution does not necessarily increase proportionally with the scale of the problem.

"In social settings, we used to look at smaller problems," Amarasinghe says. "If you look at the people in this [CSAIL] building, we're all connected. But if you look at the planet scale, I don't scale my number of

friends. The planet has billions of people, but I still have only hundreds of friends. Suddenly you have a very sparse problem."

Similarly, Amarasinghe says, an online bookseller with, say, 1,000 customers might like to provide its visitors with a list of its 20 most popular books. It doesn't follow, however, that an online bookseller with a million customers would want to provide its visitors with a list of its 20,000 most popular books.

Thinking locally

Today's computer chips are not optimized for sparse data—in fact, the reverse is true. Because fetching data from the chip's main memory bank is slow, every core, or processor, in a modern chip has its own "cache," a relatively small, local, high-speed memory bank. Rather than fetching a single data item at a time from main memory, a core will fetch an entire block of data. And that block is selected according to the principle of locality.

It's easy to see how the principle of locality works with, say, image processing. If the purpose of a program is to apply a visual filter to an image, and it works on one block of the image at a time, then when a core requests a block, it should receive all the adjacent blocks its cache can hold, so that it can grind away on block after block without fetching any more data.

But that approach doesn't work if the algorithm is interested in only 20 books out of the 2 million in an online retailer's database. If it requests the data associated with one book, it's likely that the data associated with the 100 adjacent books will be irrelevant.

Going to main memory for a single data item at a time is woefully inefficient. "It's as if, every time you want a spoonful of cereal, you open

the fridge, open the milk carton, pour a spoonful of milk, close the carton, and put it back in the fridge," says Vladimir Kiriansky, a PhD student in electrical engineering and computer science and first author on the new paper. He's joined by Amarasinghe and Yunming Zhang, also a PhD student in electrical engineering and computer science.

Batch processing

Milk simply adds a few commands to OpenMP, an extension of languages such as C and Fortran that makes it easier to write code for multicore processors. With Milk, a programmer inserts a couple additional lines of code around any instruction that iterates through a large data collection looking for a comparatively small number of items. Milk's compiler—the program that converts high-level code into low-level instructions—then figures out how to manage memory accordingly.

With a Milk program, when a core discovers that it needs a piece of data, it doesn't request it—and a cacheful of adjacent data—from main memory. Instead, it adds the data item's address to a list of locally stored addresses. When the list is long enough, all the chip's cores pool their lists, group together those addresses that are near each other, and redistribute them to the cores. That way, each core requests only data items that it knows it needs and that can be retrieved efficiently.

That's the high-level description, but the details get more complicated. In fact, most modern computer chips have several different levels of caches, each one larger but also slightly less efficient than the last. The Milk compiler has to keep track of not only a list of memory addresses but also the data stored at those addresses, and it regularly shuffles both around between cache levels. It also has to decide which addresses should be retained because they might be accessed again, and which to discard. Improving the algorithm that choreographs this intricate data ballet is where the researchers see hope for further performance gains.

"Many important applications today are data-intensive, but unfortunately, the growing gap in performance between memory and CPU means they do not fully utilize current hardware," says Matei Zaharia, an assistant professor of computer science at Stanford University. "Milk helps to address this gap by optimizing memory access in common programming constructs. The work combines detailed knowledge about the design of memory controllers with knowledge about compilers to implement good optimizations for current hardware."

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology