BY PAT HANRAHAN, PhD, PROFESSOR OF COMPUTER SCIENCE AT STANFORD UNIVERSITY

Using Domain Specific Languages to Access Parallel Computing in All Its Forms

LINDA A. CICERO | STANFORD NEWS SERVICE

Increasingly, parallel computing is becoming pervasive. The processors found in laptop and desktop machines now have two to four cores. Cheap and widely available graphical processing units (GPUs) contain hundreds, and soon thousands, of cores. And with the advent of cloud and grid computing infrastructure, every scientist can access thousands of processors and petaflops of computing.

For scientists who want to do simulation or data analysis, this is great news. Unfortunately, writing parallel programs is hard. The processor on your desktop uses a different programming model (e.g. threads and locks) than a GPU (that uses CUDA or OpenCL) or a cluster (that uses MPI). Because hardware designers want their computers to be efficient, they specialize them for different types of workloads. A GPU, for example, is ten times more power efficient than a CPU at floating point intensive, high-throughput calculations. This need for power efficiency will lead to even more heterogenous machines in the future. Learning the low-level details of how to program many different types of machines efficiently takes a long time. And computational biologists have other fish to fry—solving biological research problems.

I believe that to take advantage of emerging parallel computers without learning how to program different machines, computational biologists will need to write software at a higher-level: They will need to use domain-specific languages and libraries (DSLs). A DSL is an environment that is tailored to a particular domain or task. Most scientists already use DSLs such as matlab, R, and latex routinely in their work. DSLs take the grunge out of programming, letting the computational scientist focus on the science, not on the computer hardware.

Unfortunately, DSLs have a reputation for being slow (often because they are interpreted). This need not be the case. Two widely used DSLs are OpenGL and SQL.

OpenGL is a graphics library that can be thought of as a DSL for graphics. OpenGL drivers implement graphics commands on highly parallel GPUs in an efficient way.

SQL is the *lingua franca* of databases. Because queries are expressed at a high-level, it is possible to implement SQL on large datacenters. These two DSLs are both portable and efficient.

Because DSLs operate at a higher-level of abstraction, they can often be automatically parallelized. Take for example molecular dynamics or n-body algorithms. There are very efficient n-body algorithms that are tailored to different types of parallel machines. A DSL for molecular dynamics can build these algorithms into the system, and thus a program written in the molecular dynamics DSL will run portably on different types of parallel computers. By contrast, a general-purpose parallelizing compiler could never automatically discover these algorithms, if the molecular dynamics application is written in a lower-level programming language like C++.

As computational biology evolves and new software is developed, we need to do two things to ensure that this software will run on the parallel computers of the future. First, we need to assemble teams of computational biologists and computer scientists to develop DSLs for the major areas of computational biology. One example of successful collaborations of this type is the OpenMM Project developed by Simbios at Stanford. DSLs could be developed for many other areas including finite element calculations, fluid flow, machine learning, and data analysis, to name a few. Second, computer scientists need to build tools that make it easier to build DSLs. Currently, programs like Matlab are implemented from the ground up. DSL writers essentially "roll-their-own" systems. We need a general infrastructure so that DSLs can be easily built and extended by small groups of people. If we succeed, future computational biologists will have access to extraordinary computing capabilities, which in turn will enable many scientific discoveries. □

I believe that to take advantage of emerging parallel computers without learning how to program different machines, computational biologists will need to write software at a higher-level: They will need to use domain-specific languages and libraries (DSLs).