New Parallel Programming Abstractions and the Role of Compilers

Laxmikant V. Kalé

University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
kale@uiuc.edu

Abstract

Most of the parallel programming, especially in applications in Computational Science and Engineering (CSE), is done using MPI. OpenMP is used on some shared memory platforms. However, it is becoming increasingly evident that new higher level parallel programming abstractions are needed if we have to increase programming productivity further.

Here, I present my views on what kinds of high level languages and abstractions one should look for, what research is needed to develop them, what obstacles I see in their development and adoption, and what role compilers can and should play in their development. In particular, I argue that adaptive run-time systems to separate the issues of resource management and abstractions for supporting global (but disciplined) view of data and global view of control are needed. Further, the role of compiler research needs to be directed to supporting such models, even though that requires a paradigm shift (toward simpler problems!) for the compiler research community.

Most of the parallel programming, especially in applications in Computational Science and Engineering (CSE), is done using MPI. OpenMP is used on some shared memory platforms. However, it is becoming increasingly evident that new higher level parallel programming abstractions are needed if we have to increase programming productivity further. Such an increase in programming productivity is needed for two reasons.

- Broadening of the parallel programming community: First, even within the CSE community, parallel programming expertise is confined to a narrow segment of researchers, especially concentrated at DOE and other governmental laboratories. For the broad CSE community to adopt parallel programming, it needs to become simpler and more productive. An aerospace engineer, for example, will want to develop a parallel program only when it is made easy for them. One should not expect them to be an expert parallel programmer in addition to being an expert modeler. Of course, parallel programming is slated to grow beyond the CSE community. We can expect desktops embody to multi-core processors and the number of processors in a typical desktop to increase beyond 10 or even hundred, as projected by the industry. This places a further premium on parallel programming productivity.

- **Deepening of the program complexity** Secondly, even for the expert programmers of today, the applications of tomorrow pose new programming challenges. To harness the unprecedented power of new large parallel machines, scientists and engineers are coming up with new generations of algorithms that tend to be adaptive and increasingly model dynamic behavior. In order to increase the fidelity of their simulations, they are also increasingly using multi-physics simulations. These simulations involve integrating separate (and often independently developed) parallel modules into a single simulation.

Here, I present my views on what kinds of high level languages and abstractions one should look for, what research is needed to develop them, what obstacles I see in their development and adoption, and what role compilers can and should play in their development.

To summarize some of the points:

Resource management needs to be separated from parallel programming effort. This can be done via adaptive run-time systems, which need to be empowered by a programming paradigm that takes away the notion of physical processor from the programmer's purview. Systems like Charm++ and Adaptive MPI are examples of this, where the program is divided into a large number of objects or threads,

1-4244-0054-6/06/$20.00 ©2006 IEEE

which are assigned and reassigned to processors at run-time as needed.

Global view of data: many abstractions attempt to support a global view of data. It seems increasingly clear that such a view is needed, but the undisciplined and race-condition prone SAS model, where any process can modify any data any time is not the right model. Appropriately restrictive models are needed, those that interoperate with message-passing are especially desirable. This is reflected in development of models such as UPC, Titanium, and Co-array-Fortran, as well as libraries including global arrays, multi-phase shared arrays (MSA) etc. We need to examine which abstractions are useful in what contexts, and how to unify them if possible.

Global view of control: HPF (High Performance Fortran) and OpenMP are examples of languages with which the programmer deals with one apparent global thread of control. Parallel iterations of a loop may fork the control, but it joins into a single thread after the loop. This explains the attractiveness of these programming models. Of course, the specific systems are not general enough to support the needs of many parallel applications, and so have not been as successful as MPI. But the "hankering" for a global flow of control is real. Unfortunately, the complexity of parallel applications makes it less and less likely that an HPF/OpenMP style single flow of control will be adequate in high performance applications. For example, two loops that happen to be independent of each other should be allowed to run concurrently (possibly on overlapping sets of processors). This should be true even if the two loops, as long as they are independent, are nested in two separate function calls. If you express the computation in terms of object arrays, as in Charm, this problem can be seen clearly in its complex manifestation. A program may be made up of multiple arrays of data-driven objects. With message-driven objects, control may shift from one array to the other in a purely reactive manner. One can explain what happens by drawing a diagram; but the overall flow remains an "emergent property". You cannot easily point to the program text and by merely doing so understand and explain its control flow. We need a method that will provide the program writer and program reader with a single "apparent" thread of control, and still have the compiler and runtime system work together to expose all the natural concurrency across modules and object arrays.

Compiler Research and Compiler Support: Many of the ideas and abstractions for improving programmer productivity require compiler support. However, the research that this entails is very different than that of traditional compiler research in this area. The latter is dominated by dependency analysis and extracting loop parallelism. In fact, some of the techniques needed are sometimes much too simple to attract the attention of compiler researchers and this is a genuine problem! Without a reasonable pliable compiler infrastructure, one cannot develop, implement and test new abstractions. Some "directed" compiler research is necessary to do the kind of analysis needed for particular abstractions or functionalities. This calls for a grand collaboration between the abstraction designers and compiler researchers (or just compiler developers). There is a need for "compilation support service". Work on telescoping languages is an example of such a collaboration.