Chapel

Making Large-Scale Parallel Programming Productive

Overview

Chapel is an emerging programming language designed to improve the programmability of large-scale parallel computer systems. An open-source effort led by Cray Inc. and supported by a rich community of collaborators, Chapel's primary goal is to increase the productivity of high-end computer users by making it easier to write, read, modify, port, tune and maintain parallel programs.

Background

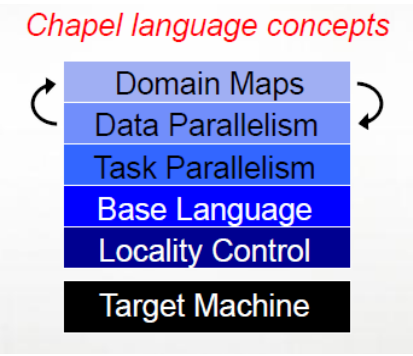
The concept for a new, high productivity programming language arose out of the DARPA[1]-led High Productivity Computing Systems (HPCS) program. In 2002, HPCS called for the development of a new generation of economically viable, productivity-boosting computing systems for national security and industry. That call included a vision for easier-to-program petascale computer systems. In response, Cray proposed its next-generation "Cascade" supercomputer in combination with "Chapel" (Cascade High Productivity Language) – a programming language designed from first principles and aimed at addressing the limitations of existing languages.

Chapel development began in 2003 with an in-depth study of current programming languages, followed by several years of experimentation and feedback. By 2006, Chapel – the implementation and language – had taken form.

Work continues on Chapel to evolve it into a production-grade programming language. Such efforts include improving its interoperability with existing code in other languages as well as adding support for developer tools. Simultaneously, the Chapel team is working to address key research questions for next-generation systems such as making effective use of increasingly hierarchical/heterogeneous processor architectures while also improving resiliency and fault tolerance.

Features

Chapel's overall goal is to increase programmer productivity by: improving the programmability of parallel computers; matching or beating the performance of current programming models; providing better portability than current programming models; and improving the robustness of parallel codes.



In pursuit of this goal, Chapel focuses on five key areas of language technology: multiresolution design; general parallel programming; global-view abstractions; control of locality; and closing the gap between mainstream and HPC languages.

Multiresolution design. Chapel's multiresolution design permits users to write very abstract code and then incrementally add more detail. This means users can write at a high level for the 90 percent of their code that is non-performance critical and then dive down as close to the machine as they need for the crucial10 percent.

General parallel programming. Chapel supports a multithreaded execution model via high-level abstractions for data parallelism, task parallelism, concurrency and nested parallelism. This feature means users can express any parallel algorithm they can conceive of and target it to any parallel hardware.

Global-view abstractions. Chapel supports global-view data aggregates with user-defined implementations, permitting operations on distributed data structures. This feature means users can program distributed memory systems using abstractions that support more of a desktop programming model look-and-feel.

Control of locality. Chapel's locale type enables users to specify and reason about the placement of data and tasks on a target architecture in order to tune for locality. This means that users can write code that will scale to large machine sizes by controlling their data placement and task affinity.

Reducing gap between mainstream and parallel languages. Chapel is an imperative block-structured language with a lightweight, familiar syntax designed to be easy to learn for users of C, C++, Fortran, Java, Perl, Matlab and other popular languages. This feature leverages advances in modern language design and makes it easier for users to capitalize on the skills of the entry-level workforce.

Innovations

Chapel is designed from first principles rather than by extending an existing language. However, it does build on concepts and syntax from previous languages. Its parallel features are most directly influenced by ZPL, High-Performance Fortran (HPF), and the Cray MTA™/Cray XMT™ extensions to C and Fortran. Within that context, Chapel has introduced several key innovations:

Ability to define arrays. Chapel gives the user the ability to create their own parallel array implementations by controlling issues like memory layout, data decomposition and iterator patterns. This feature gives users full control over how their high-level data structures are implemented.

Specification of parallel iterators. Chapel allows users to specify how parallel loops are decomposed into tasks and how the loop's iterations are divided between those tasks. Iterators are written within Chapel, resulting in more seamless integration between data- and task-parallel styles of computation than any previous parallel language has supported.

Distinct concepts for parallelism and locality. In most high performance computing programming models, programmers either cannot express parallelism distinctly from locality or they cannot control locality at all. Chapel uses distinct concepts for parallel computation ("tasks") and system organization ("locales") which makes it well-suited for emerging exascale node architectures.

More information can be found online at http://chapel.cray.com

---

[1] DARPA – Defense Advanced Research Projects Agency, an agency of the U.S. Department of Defense

Corporate Headquarters
901 Fifth Ave, Suite 1000
Seattle, WA 98164
Phone: 206.701.2000
Fax: 206.701.2500
www.cray.com