

# Towards High-Level Languages for Peta-Scale Computing

*Hans P. Zima*

Institute of Scientific Computing, University of Vienna, Austria

*and*

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

## Abstract

High performance computing has become the third pillar of science and technology, providing the superior computational capability required for dramatic advances in fields such as DNA analysis, drug design and astrophysics. However, during the past decade, progress has been impeded by a growing lack of adequate language and tool support. In today's dominating programming paradigm, users are forced to adopt a low-level programming style similar to assembly language if they want to fully exploit the capabilities of parallel machines. This leads to high-cost software production and error-prone programs that are difficult to write, reuse, and maintain. Emerging peta-scale architectures with hundreds of thousands of processors, and applications of growing size and complexity will further aggravate this problem.

This talk will discuss the state-of-the-art in programming paradigms for high performance computing and identify the challenges posed by future architectures and their applications. We will discuss the requirements for high-productivity programming languages that represent a viable compromise between the dual goals of human productivity and target code efficiency, and outline the related research challenges for effective programming environments supporting high-level language abstractions. Specifically, we will provide an overview of the new programming language *Chapel* developed in the High-Productivity-Computing-Systems (HPCS) project *Cascade*. Chapel is a modern object-oriented language providing support for generic programming, collections and iterators, and explicit parallelism in conjunction with user-defined data distributions.

The final part of the presentation will focus on the definition, use, and implementation of data distributions in Chapel. The main challenge is to expose the user to enough level of detail regarding parallel execution to grant effective communication of problem-specific knowledge, while concealing the unproductive details related to low-level parallel programming such as communication, synchronization, and the explicit distinction between local and remote memory accesses. Rather than offering a fixed set of built-in distributions, Chapel provides a distribution class interface which allows the explicit specification of the mapping of elements in a collection to units of uniform memory access, the control of the arrangement of elements within such units, the definition of sequential and parallel iteration over collections, and the specification of allocation policies. The result is a concise high-productivity programming model that separates algorithms from data representation and enables reuse of distributions, allocation policies, and specialized data structures.