Chapter 14
A Domain-Specific Language for High-Level Parallelization

Ritu Arora
Texas Advanced Computing Center, USA

Purushotham Bangalore
University of Alabama at Birmingham, USA

Marjan Mernik
University of Maribor, Slovenia

ABSTRACT

There are several ongoing research efforts in the High Performance Computing (HPC) domain that are employing Domain-Specific Languages (DSLs) as the means of augmenting end-user productivity. A discussion on some of the research efforts that can positively impact the end-user productivity without negatively impacting the application performance is presented in this chapter. An overview of the process of developing a DSL for specifying parallel computations, called High-Level Parallelization Language (Hi-PaL), is presented along with the metrics for measuring its impact. A discussion on the future directions in which the DSL-based approaches can be applied in the HPC domain is also included.

INTRODUCTION

The High Performance Computing (HPC) discipline has seen tremendous growth in the last decade in terms of the power and scale of computing platforms. By the next decade we might as well be into the exascale computing era. One of the key challenges on the path to exascale computing is to develop programming environments that would reduce the complexity associated with the process of developing HPC applications, especially in the light of heterogeneity in the computing platforms and multiple levels of memory hierarchies.

There are plenty of parallel programming models that are already in widespread usage or have the potential of being widely used - for example, MPI (MPI Forum, 2009), OpenMP (OpenMP, 2011), OpenCL (OpenCL, 2011), TBB (Threaded Building Blocks, 2011), Cilk Plus (Intel Cilk Plus, 2011), Coarray Fortran (Mellor-Crummey et al, 2009), and UPC (UPC, 2005). Each model is suitable for a particular architecture and has

DOI: 10.4018/978-1-4666-6042-7.ch014

a learning curve associated with it. Due to the rapidly evolving solution space for developing HPC applications, the programmers are caught in the "problem of plenty." Moreover, the exascale computing platforms are likely to use hybrid programming models – probably MPI for communication between different address spaces and a shared memory programming paradigm for communication within an address space. While programming in a single parallel paradigm is quite a challenge by itself (Arora et al, 2012), handling hybrid programming models could put additional burden on programmers.

With the increase in heterogeneity in the computing platforms, it is also a challenge to write portable applications that can run optimally on a variety of architectures. It is a tedious task to hand-tune the applications for every architecture that they are meant to run on. High-level parallel programming environments and abstractions that can assist in the rapid development of scalable and optimized HPC applications are, therefore, required.

Domain-Specific Languages (DSLs) that are written in a platform-independent manner can be helpful in capturing the description of algorithms at a high-level. The high-level description of an algorithm can then be used to generate low-level code for multiple HPC platforms (Sujeeth et al, 2011). The low-level code generated from the DSL code can be optimized automatically on the basis of the domain knowledge built in the compiler and run-time system. DSLs, therefore, not only have the potential of increasing the programmer productivity but can also help in squeezing maximum performance from the applications via domain-specific optimizations.

The key objectives of this chapter are to discuss some active research works that aim at increasing the productivity of HPC application developers through the usage of DSLs and abstractions, present a walk-through of the complete process of developing a DSL, and discuss ways to measure its impact.

BACKGROUND

DSLs are gradually gaining popularity in the HPC domain because DSL-based approaches have been shown to be helpful in developing parallel applications and algorithms at a high-level of abstraction. The low-level details related to the development and deployment of HPC applications (*viz.* managing the communication between processors, distribution of data on various processors, and load-balancing) can be hidden from the programmers. A key advantage of the DSL-based approaches is that the specifications of applications or algorithms need not be modified in the event of a change in the underlying architecture. Therefore, the application portability concerns are assuaged. DSLs are also helpful in the separation of concerns (Arora et al, 2011; Kiczales et al, 1997). Some of the DSLs that have the potential of reducing the adoption barriers to HPC are discussed in this section.

High-Level Parallelization Language (Hi-PaL) is an application-domain neutral, declarative and platform-independent DSL for expressing concurrency in existing sequential applications written in C/C++ (Arora et al, 2011). The existing sequential applications are analyzed for concurrency and data-dependencies by the programmers who then write Hi-PaL code to provide specifications about what to parallelize and where (in the sequential application). The Hi-PaL code is automatically translated into low-level C/C++/MPI code and woven into the existing applications with the help of a Source-to-Source Compiler (SSC). It is a multi-step process that involves generation of grammar-specific (C/C++ grammar-specific) rules so that the SSC can carry out automatic code transformation (Arora et al, 2011; Czarnecki et al, 2000). Without Hi-PaL, the end-users would have to write their own grammar-specific rules if they have to do automatic source-to-source transformation, or they would have to manually (and invasively) insert the C/C++/MPI code into the existing sequential application to make