Pattern–based Object–Oriented Parallel Programming

Steve MacDonald, Jonathan Schaeffer, and Duane Szafron

University of Alberta, Edmonton, Alberta CANADA T6G 2H1
Email: {stevem,jonathan,duane}@cs.ualberta.ca

1 Introduction and Motivation

Over the past five years there have been several attempts to produce *template–based* (or, as they are now called, *pattern–based* [4]) parallel programming systems (PPS). By observing the progression of these systems, we can clearly see the evolution of pattern–based computing technology. Our first attempt, FrameWorks [9], allowed users to graphically specify the parallel structure of a procedural program in much the same way they would solve a puzzle, by piecing together different components. The programming model of FrameWorks was low–level and placed the burden of correctness on the user. This research led to Enterprise [7], a PPS that provided a limited number of templates that could be composed in a structured way. The programming model in Enterprise was at a much higher level, with many of the low–level details handled by a combination of compiler and run–time technology.

In this progression, we can see more emphasis placed on the *usability* of the tools rather than raw performance gains, as shown by Szafron and Schaeffer [12]. Each successive system reduced the probability of introducing programmer errors. However, since performance considerations cannot be ignored, each successive systems also supported incremental application tuning. A related performance issue identified by Singh *et al.* [10] is *openness*, where a user is able to access low–level features in the PPS and use them as necessary. This work led to a critical evaluation of pattern–based systems that provides the motivation for our new system, CO_2P_3S (Correct Object–Oriented Pattern–based Parallel Programming System, pronounced "cops").

In this paper, we present an architecture and model for CO_2P_3S in which we address some of the shortcomings of FrameWorks and Enterprise. Our continuing goal is to produce usable parallel programming tools. The first shortcoming we address is the loose relationship between the user's code and the graphical specification of the program structure. Enterprise improved on FrameWorks by verifying a correspondence between the parallel structure and the code at compile–time. However, we feel that forcing the user to write a program that conforms to an existing diagram is redundant. If the structure of the application is already known, then the basic framework can be generated automatically. This reduces the amount of effort required to write programs, while simultaneously reducing programmer errors even further. The CO_2P_3S architecture also supports improved incremental tuning. The architecture is novel in that it provides several user–accessible layers of abstraction. At any given time during performance tuning, a programmer can work at the appropriate level of abstraction, based on what is being tuned. This can range from modifying the basic parallel pattern at the highest level, to modifying synchronization techniques at the middle layer, to modifying which communication primitives are used at the lowest level. Our goal is an open system where the performance of an application is directly commensurate with programmer effort.

2 The Architecture of CO_2P_3S

The architecture of CO_2P_3S consists of three layers: `Patterns`, `Intermediate Code`, and `Native Code`. These layers represent different levels of abstraction, where the abstraction of each layer is implemented by the one underneath.

Each layer is transformed during compilation to the layer underneath. At the pattern layer, the developer selects a pattern using a graphical tool. The PPS then generates a template for the parallel application and the user is restricted to providing sequential application–specific code at particular locations in the generated template. At the pattern level, the PPS guarantees the correctness of the generated program by using conservative synchronization mechanisms that may not yield peak parallel performance.

Unlike Enterprise, CO_2P_3S provides programmer access to the second layer where the templates can be edited. From this layer down, the programmer is responsible for the correctness of the resulting program. To simplify this task, we provide a high–level parallel programming language that is an extension of existing OO languages (Java or C++). This superset includes keywords to denote parallel classes, specify concurrent activities and express necessary synchronization (using constructs such as asynchronous methods, threads, and futures).

Finally, the third layer is the native programming language augmented with a library that provides the services required by the first two layers. Users are given full access to all language features and library code.

We believe that providing intermediate levels of abstraction can provide several benefits. First, by generating correct template code at the first level, we can ensure that a user has a working parallel program before the tuning process begins. Second, it eases the tuning process by introducing the run–time system in smaller increments. These smaller increments provide better opportunities for novice or intermediate users to find a comfortable level of abstraction while still providing full access to the run–time system for experienced users. Lastly, it should always be possible to improve the performance of a program by using the abstraction of a lower level. If so, then the performance of an application should be more directly commensurate with programmer effort.

3 The Model

In this section, we more fully specify the model and demonstrate it using an example program. Our example shows some details of a generic mesh computation.