Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Contextual abstraction in a type system for component-based high performance computing platforms

Francisco Heron de Carvalho Junior [*], Cenez Araújo Rezende, Jefferson de Carvalho Silva, Wagner Guimarães Al-Alam, João Marcelo Uchoa de Alencar

Mestrado e Doutorado em Ciência da Computação, Universidade Federal do Ceará, Brazil

A R T I C L E I N F O

Article history:
Received 19 April 2014
Received in revised form 20 July 2016
Accepted 21 July 2016
Available online xxxx

Keywords:
Component-based software engineering
High performance computing
Component-based high performance computing
Parallel programming
Type systems

A B S T R A C T

HTS (Hash Type System) is a type system designed for component-based high performance computing (CBHPC) platforms, aimed at reconciling portability, modularity by separation of concerns, a high-level of abstraction and high performance. Portability and modularity are properties of component-based systems that have been extensively validated. For improving the performance of HPC applications, HTS introduces an automated approach for dynamically discovering, loading and binding parallel components tuned for the characteristics of the parallel computing platforms where the application will execute. To do so, it is based on contextual abstraction, where the performance of components that encapsulate parallel computations, communication patterns and data structures may be tuned according to the features of parallel computing platforms and the application requirements. In turn, for providing a higher level of abstraction in parallel programming, HTS supports an expressive approach for skeleton-based programming. A study of the safety properties of HTS using a calculus of component composition has provided solid foundations for the design of configuration languages for the safe specification and deployment of parallel components. The features of HTS are validated with three case studies that exercise the programming techniques behind contextual abstraction, including skeletons and performance tuning.

© 2016 Published by Elsevier B.V.

1. Introduction

Since the 2000s, the advent of grids [1] and clouds [2] has broadened the horizon of High Performance Computing (HPC) applications. They inaugurated the era of large-scale HPC platforms comprised of heterogeneous collections of computational *resources*. These platforms may attend to the high performance requirements of a number of applications that present outstanding impacts in scientific discovery and technological innovation, when they are orchestrated in an effective way. By using these platforms, scientists and engineers have proposed innovative environments for cooperative interdisciplinary work, moving forward the scale and complexity of modern HPC software [3]. With the consolidation of heterogeneous parallel computing, where different kinds of processors [4], accelerators [5–7] and multiple memory hierarchies may be orchestrated to solve computation problems, the effective use of HPC resources became even more challenging [8]. Such a

[*] Corresponding author.
E-mail addresses: heron@lia.ufc.br (F.H. de Carvalho Junior), cenezaraujo@lia.ufc.br (C.A. Rezende), jeffersoncarvalho@lia.ufc.br (J. de Carvalho Silva), joao.marcelo@lia.ufc.br (J.M. Uchoa de Alencar).

http://dx.doi.org/10.1016/j.scico.2016.07.005
0167-6423/© 2016 Published by Elsevier B.V.

context increases the demand for research about new programming models, techniques and abstractions for addressing both the increasing degree of scale and complexity of HPC software and the tuning of its performance according to the particular characteristics of target parallel computer architectures.

Components are independent units of software composition with well-defined interfaces, subject to independent deployment and third-party composition [9–11]. Component-based software engineering (CBSE) has been successfully applied in the software industry in dealing with the complexity and scale of business software. In turn, the research works on the design and implementation of component-based high performance computing (CBHPC) platforms have investigated how to use components in dealing with complexity and scale of software with HPC requirements. More recently, they have approached the challenge behind heterogeneity of parallel computing platforms [8]. CBHPC has been guided by the researchers involved in a number of initiatives, including CCA (Common Component Architecture) [12], Fractal [13], and GCM (Grid Component Model) [14]. Most of these researchers are computational scientists and engineers who have presented successful case studies of CBHPC platforms in the last few years, using real applications.

The design of CCA has been driven by the requirements of frameworks for computational science applications (*computational frameworks*) [15], aiming at minimizing overheads in component bindings and supporting typical data types of scientific applications through SIDL (Scientific Interface Definition Language). A number of computational frameworks and component platforms have adhered to CCA [16–19]. Other CCA frameworks have been developed from scratch for evaluating alternative framework designs focused on different requirements, such as parallelism support [20–22]. In turn, Fractal [23] is a hierarchical component model where components may be recursively composed to form new components. Julia, AOKell and ProActive/Fractal are implementations of Fractal. GCM, implemented by ProActive/GCM [24], extends Fractal with autonomic, adaptation and reconfiguration capabilities for grid computing [14].

Our main contribution to CBHPC is the Hash component model, which faces the lack of models that reconcile expressiveness and efficiency in building software with parallel components [25]. HPE (Hash Programming Environment) is the reference implementation of Hash. It manages the life-cycle of parallel components, called #-components, targeted at cluster computing platforms [26]. HPE complies with CCA, introducing support for parallelism, distribution and hierarchical composition into the design of CCA frameworks [27]. Experimental studies have reported that #-components may reconcile expressiveness and performance in the component-based development of parallel programs [19].

For the purposes of this paper, we define a *parallel computing system* as a view that integrates HPC software and the parallel computing platform (HPC hardware) where it will execute. This is a convenient way of viewing the design of parallel programs with critical performance requirements, since the choice of the best algorithms and parallelism strategies for implementing them is highly dependent on the architecture of their target parallel computing platforms [28]. For instance, consider a parallel computing system built from components of a CBHPC platform. However, instead of making direct reference to each component, the parallel computing system only specifies what it requires and what it provides to the component, delegating to a resolution system the selection of a component that may maximize its overall performance.

We report how we address, in the context of HPE, the problem of selecting the component that best satisfies the needs of a parallel computing system from a set of alternative implementations cataloged in a library. To that end, we introduce the abstraction of *context* for making assumptions, in a parallel computing system, about the requirements of the application and the features of the target parallel computing platform. This is so-called *contextual abstraction*. Contextual abstraction becomes more relevant as heterogeneous parallel computing platforms become more widespread. So, contextual abstraction is particularly relevant for the requirements of computational environments that either provide heterogeneous parallel computing platforms as resources (e.g. grids) or use them for enabling higher-level HPC services (e.g. clouds).

This paper introduces HTS (Hash Type System), the component type system of HPE for supporting *contextual abstraction*. It automates the performance tuning of parallel components according to the features of parallel computing platforms, by minimizing the end-user intervention and delegating performance tuning responsibilities to parallel programming specialists. It is based on a kind of contractual interface, a so-called *abstract component*. An abstract component represents a set of components that address a well-defined concern. The implementation of each component is tuned according to a set of context parameters. The contextual abstraction distinguishes HTS from other component adaptability approaches proposed by CBHPC platforms. We claim that HTS contributes to advancing the state-of-the-art of CBHPC platforms in addressing the problem of dealing with large-scale heterogeneous HPC systems.

HTS applies skeleton programming [29,30] to CBHPC platforms. In component-based skeleton-programming, components may encapsulate patterns of parallel computation, communication and synchronization that may be particularized through polymorphism and tuned for performance through contextual abstraction. Using this approach, component developers may tune the performance of the pattern implementation according to the particular architectural features of the target parallel computing platform. This is a key feature that must be supported by skeleton programming systems.

The three case studies presented in this paper demonstrate how a component-based approach of skeleton-programming may be implemented using contextual abstraction. Therefore, the first two case studies show how to use the well-known parallel programming skeleton so-called *Farm* for implementing numerical integration and Map-Reduce, a parallel computing pattern that has been widely adopted in large-scale data parallel processing. Map-Reduce may be also viewed as a skeleton [31]. In particular, we show how it may be implemented as a composition of farms. In turn, the third case study presents the *alternating direct implicit* (ADI), a solution method for sparse systems of equations, as a skeleton. These case studies show the role of HTS in the instantiation of skeletons to particular computations, in such a way that not only the performance of the skeleton is tuned for the target parallel computing platform, but also the performance of the components used to