A Graphical Environment for Development of MPI Applications

J. L. Quiroz-Fabián
Universidad Autónoma Metropolitana, México City
jlqf@xanum.uam.mx

G. Román-Alonso
Universidad Autónoma Metropolitana, México City
grac@xanum.uam.mx

M. A. Castro-García
Universidad Autónoma Metropolitana, México City
mcas@xanum.uam.mx

J. Buenabad-Chávez
Centro de Investigación y de Estudios Avanzados-IPN, México City
jbuenabad@cs.cinvestav.mx

M. Aguilar-Cornejo
Universidad Autónoma Metropolitana, México City
mac@xanum.uam.mx

ABSTRACT

This paper presents GD-MPI: a Graphical environment for Development of parallel MPI applications. GD-MPI offers users a web browser-based GUI to graphically specify both: workflows that represent a set of Java-MPI processes and communication between these processes including group creation, point-to-point and collective communications. GD-MPI also runs such processes remotely.

Keywords

Parallel Applications, WorkFlow, Graph Grammar, Hyperedge Replacement Grammar, Java-MPI.

1. INTRODUCTION

Parallel computers are common today, making it possible to reduce computational time substantially and to run memory demanding applications. However, parallel programming is not simple, especially for non-experts in parallel computing even if they are experienced sequential programmers.

A possible solution is the use of tools that offer a high-level abstraction that hides the details of parallel programming and is more intuitive to use [2] [4] [6]. In this work we propose such a tool named GD-MPI, a development environment and language for developing Java-MPI applications. GD-MPI differentiates itself from other tools in that it generates, from sequential code, a Java-MPI program that is run in parallel. GD-MPI was derived from the original project called GDEC (Graphical Development Environment in the Cloud) [5]. GD-MPI applications are specified as workflows, which correspond to a set of Java-MPI processes where group creation, point-to-point and collective communications are graphically expressed by developers. The workflows thus described are translated into lower-level Java-MPI code organised into various programming language constructions.

The paper continues as follows. In Section 2 we describe the GD-MPI development environment, its architecture and language. Section 3 presents an example using GD-MPI. We conclude and present future work in Section 4.

EuroMPI/ASIA '14, September 9-12 2014, Kyoto, Japan

http://dx.doi.org/10.1145/2642769.2642793.

2. GD-MPI

GD-MPI runs on different web browsers as a Rich Internet Application[1]. It is composed of the development and execution environment and a graphical language for the implementation of workflows. The following sections describe in detail these elements.

2.1 GD-MPI architecture

The GD-MPI architecture consists of five components. A component that runs in web browsers is used to draw workflows by dragging, dropping and connecting icons together (Figure 1 a). A component that stores in memory a workflow like a data structure (Figure 1 b). Each item in this data structure represents an icon or a list of workflows. A component that saves or opens a workflow (Figure 1 c). Two components that generate MPI code and run that code (Figure 1 d and e). It is worth mentioning that the execution environment has been designed in order to be extended to use other target programming languages such as C or Fortran-MPI.

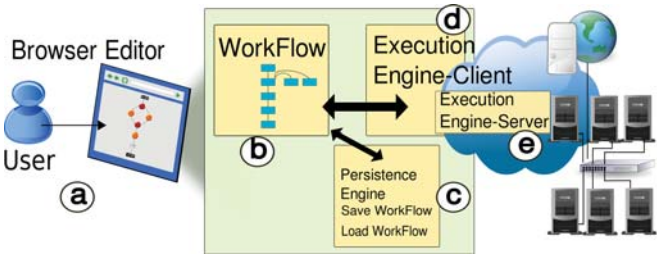

Figure 1: GD-MPI architecture.

[1]A Rich Internet Application is a Web application that has many of the characteristics of desktop applications.

2.2 GD-MPI language

The GD-MPI language supports an icon-based composition model for the development of workflows. GD-MPI icons are classified into three main types: *Processing*, *Output Data*, and *Collective Communications*. Processing icons (Figure 2a) correspond to sequential processes. Two classes of these processes are currently used: orange and red. An orange sequential process can send/receive data to/from other orange or red process. A red sequential process indicates collective operations, i.e., it is the root process. Output data icons (Figure 2b) are used to specify how the final results of a GD-MPI application should be managed. Collective icons (Figure 2c) define broadcast (B), scatter (S) or gather (G) communication operations.

Figure 2: The main icons in GD-MPI.

The set of programs that can be developed with the GD-MPI language are specified by the formalism of Hyperedge Replacement Grammars [3]. The grammar of the GD-MPI language is shown in Figure 3.

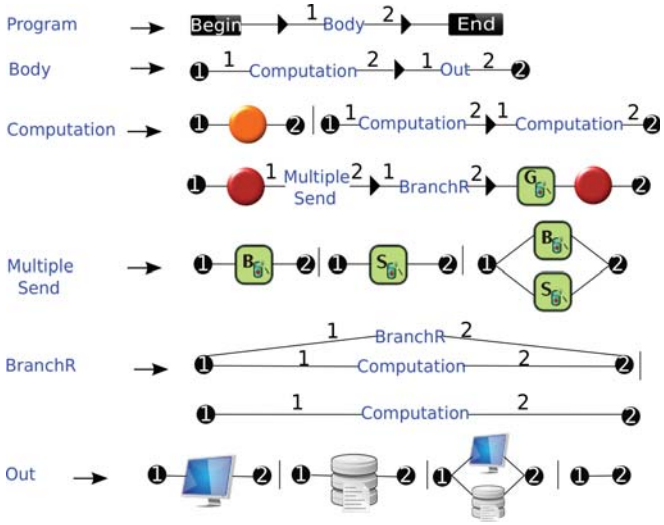

Figure 3: GD-MPI grammar.

The blue words (non-terminal symbols) represent production rules, that specify what symbols may be replaced with other symbols. These rules are used to generate workflows. Each such rule has a body (right-hand side), which consists of a combination of GD-MPI icons (terminal symbols) and non-terminal symbols that may replace it. The *Program* and *Body* rules generate all icons of a workflow. The *Computation* rule defines the types of processing icons that can be used in GD-MPI. This rule connects (concatenates) sequences of computations. Also, this rule enables the creation of groups of processes transparently. The *Multiple Send* rule defines how data will be sent, and how many processes receive such data. The *BranchR* rule allows creating a process group (computation components). Finally, the *Out* rule defines the different ways to manage the results in GD-MPI.

3. PROGRAMMING WITH GD-MPI

Our experimental application is an algorithm for determining numbers in the Mandelbrot set [1]. The blue points in Figure 4 point out the main steps in the corresponding workflow.

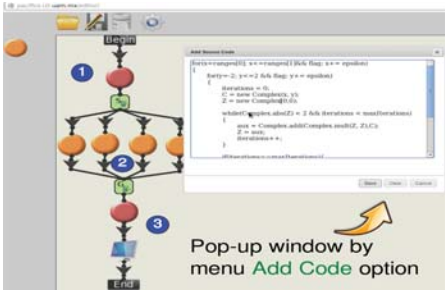

Figure 4: Maldebrot Set Workflow Porgram.

The first red process creates an array where the numbers of the Mandelbrot set will be searched (Step 1). This process distributes its array in four parts, one per each orange process. Each orange process calculates 100,000 numbers in the Mandelbrot set (Step 2). Another red process gathers all numbers, and displays them (Step 3). The developer only has to write sequential code, and GD-MPI creates and runs the generated MPI program. In order to add a code in a processing icon we provide a simple menu by just clicking on an option called "*Add Code*" (the pop-up window in Figure 4).

4. CONCLUSION AND FUTURE WORK

We have presented GD-MPI, an environment for developing workflows that are translated into Java-MPI and run. Our environment allows creating and editing a workflow by dragging, dropping and connecting icons together. The GD-MPI language was designed using Hyperedge Replacement Grammar theory, a formalism that facilitates the visualisation of graphical representations. GD-MPI runs through a web browser without having to install any software. We are working on extending our language and execution engine to support both loops within workflows and SPMD processing on a large number of computing elements as found in GPUs.

5. REFERENCES

[1] Mandelbrot set, accessed: 2014-04-01. http://en.wikipedia.org/wiki/Mandelbrot_set.

[2] Z. Farkas et al. P-grade portal: A generic workflow system to support user communities. *Future Generation Comp. Syst.*, 27(5):454–465, May 2011.

[3] A. Habel. *Hyperedge Replacement: Grammars and Languages*. Springer Verlag, USA, 1992.

[4] D. P. Pazel et al. Intentional mpi programming in a visual development environment. SoftVis '06, pages 169–170, New York, NY, USA, 2006. ACM.

[5] J. L. Quiroz-Fabián et al. A graphical language for development of parallel applications. PDPTA'13, pages 672–678, Las Vegas, Nevada, USA, 2013. WorldComp 2013 Proceedings.

[6] B. Stanislav et al. Kaira: Modelling and generation tool based on petri nets for parallel applications. UKSIM '11, pages 403–408, Washington, DC, USA, 2011. IEEE.