BalticLSC: A low-code HPC platform for small and medium research teams

Radosław Roszczyk, Marek Wdowiak, Michał Śmiałek
Kamil Rybiński and Krzysztof Marek
Warsaw University of Technology
Warsaw, Poland

***Abstract*—Research teams often face problems with insufficient resources to develop and execute their computation algorithms. This is due to limited access to professional programmers and High Performance Computing centres. This paper presents a platform that uses a new visual language to solve these problems. It uses high-level abstractions to define the flow of data and the execution of computation modules in a distributed computation environment. The runtime system follows the rules of low-code, where the development is on-line and highly simplifies the deployment of apps. This way, even small research teams can easily gain access to high-end computation resources and develop and share their applications.**

***Index Terms*—visual language, HPC, computation applications**

I. Introduction

Contemporary research teams often need to perform complex computations on large data sets. This poses two main barriers: insufficient resources to develop computation applications (skilled programmers) and insufficient resources to run these applications efficiently (High-Performance Computing – HPC – centres). This is especially evident in the case of smaller research teams (in academia and in the industry) that have limited access to such resources.

To overcome these barriers, we propose an approach that combines two areas of research: visual programming environments for HPC and low-code development environments. The first area is already quite mature [1], and aims at reducing the complexity of distributed and parallelised computation applications through proposing high-level visual languages. The second area is relatively new [2], and is oriented on the development of web applications using on-line development environments, based on visual modelling languages [3].

By combining these areas, we aim at reaching several goals:

- **effective development of computation apps** – usage of a visual language suitable for HPC computations, reduces code size and associated effort,
- **development of computation apps directly by the researchers** – high-level abstractions hide all low-level (HPC-specific) issues, thus increasing accessibility to “citizen programmers”,
- **facilitated reuse of algorithms** – composition of applications into uniform computation modules allows for easy exchange of algorithms on a common “market”,

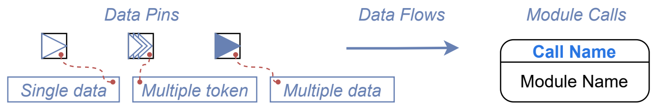


Fig. 1. CAL syntactic elements

- **instant deployment and execution of computations** – an integrated, on-line development and runtime environment removes the effort of installing and running computations in a complex distributed computing environment,
- **easy access to computation resources** – the computing infrastructure is hidden behind a comprehensible visual language and its on-line execution environment.

These goals lead to the construction of the Baltic Large Scale Computing (BalticLSC) platform, presented in this paper. The system is based on a visual language called the Computation Application Language (CAL) and its runtime environment.

II. Language design

Basic requirements for the new language include ease-of-use, modularization of computation algorithms, and representation of parallel processing through clear visualisation of data flows. This led us to develop the language syntax consisting of just three types of elements, as presented in Figure 1. Data Pins represent specific data items - input data, and output data. We distinguish between “single data” (e.g., single file or single data item or record in a database), “multiple data” (e.g., folders of files or database collections) and “multiple tokens” (sequences of e.g. files) pins. Data Pins are attached to Module Calls which represent instances of computation modules. Each module can perform specific computations on the data items referenced by the attached pins. Finally, Data Flows visualise the passing of “data tokens” between module instances. These tokens point to appropriate data items, as defined by the connected Data Pins.

An example CAL application is presented in Figure 2. This application solves a real-life problem of microscopic image processing. The input image is first split into several smaller images. Then, each of the small images is segmented and processed in parallel by two different instances of “cell analysers”. The results produced by these two analysers are then fed into the module that performs further calculations

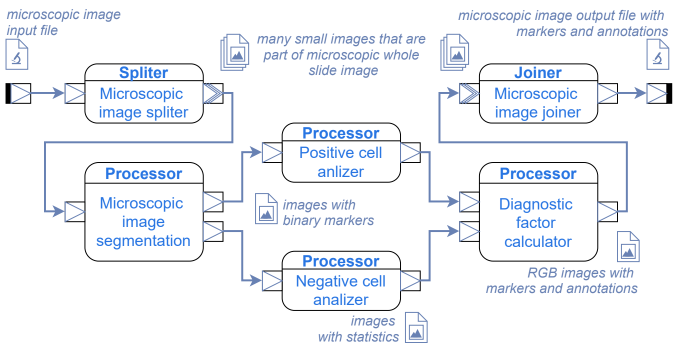

microscopic image input file
many small images that are part of microscopic whole slide image
microscopic image output file with markers and annotations
Spliter
Microscopic image spliter
Joiner
Microscopic image joiner
Processor
Positive cell anlizer
Processor
Microscopic image segmentation
images with binary markers
Processor
Diagnostic factor calculator
Processor
Negative cell analizer
RGB images with markers and annotations
images with statistics

Fig. 2. Example CAL application (slightly simplified)

and produces an output image adorned with markers and annotations. These images are then joined into a single image and sent as an output file.

Despite this simple syntax, the language semantics allows for sophisticated data processing that includes synchronisation based on data token arrival and parallelisation of processing based on the distribution of token sequences between module instances. For example, the application in Figure 2 allows for the split image fragments to be processed in parallel, and all Calculator instances are synchronised by waiting for the arrival of tokens from the respective two Analyser instances.

III. Language Implementation and System Architecture

The language syntax was defined through a MOF metamodel [4]. The language semantics was defined using a hybrid (operational and translational) approach [5]. We have defined an intermediate language (CALExec) that can be directly executed (interpreted) in an execution engine. Its semantics were defined through an abstract machine and an associated transition system. Finally, the CAL syntax was explained through supplying its translation rules into CalExec.

The metamodel was used to develop an online CAL editor. Figure 4 in the Appendix shows the editor's user interface, implemented as part of the BalticLSC system. Furthermore, we have developed a CAL-to-CalExec translator and an elaborate CALExec runtime engine. An overview of the BalticLSC Platform (see the demo version at www.balticlsc.eu) is presented in Figure 3. Its central element is the Main Server hosting the CAL Editor (the Frontend), the CAL Translator, and the CALExec Engine (the Backend). The CalExec Engine manages communication with many computation Nodes (clusters) that can run batches of module instances (CAL jobs). This communication includes token passing and distribution of CAL job batches to the nodes. Within the nodes, computation jobs are executed as encapsulated containers uploaded from a central container repository. These activities are managed by dedicated Batch Managers, on top of standard container management software. Figure 5 in the Appendix shows the appearance of the computation cockpit, which gathers information on the computation tasks. From here, the user can check the status of the executed tasks regardless of the node on which they are currently running. In addition, the user at this level can also specify data sets for processing or abort selected tasks. Figure 6 in the Appendix presents the user interface of the development shelf with applications developed by the given user and available to be executed.

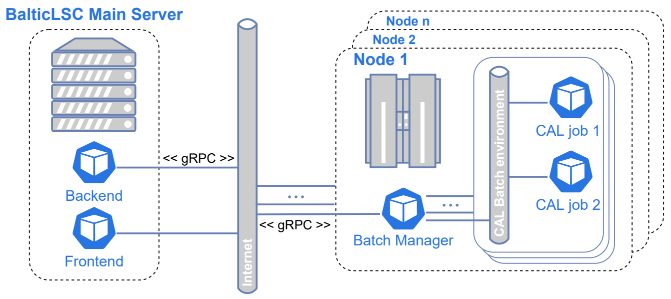


Fig. 3. Physical architecture of the BalticLSC Platform

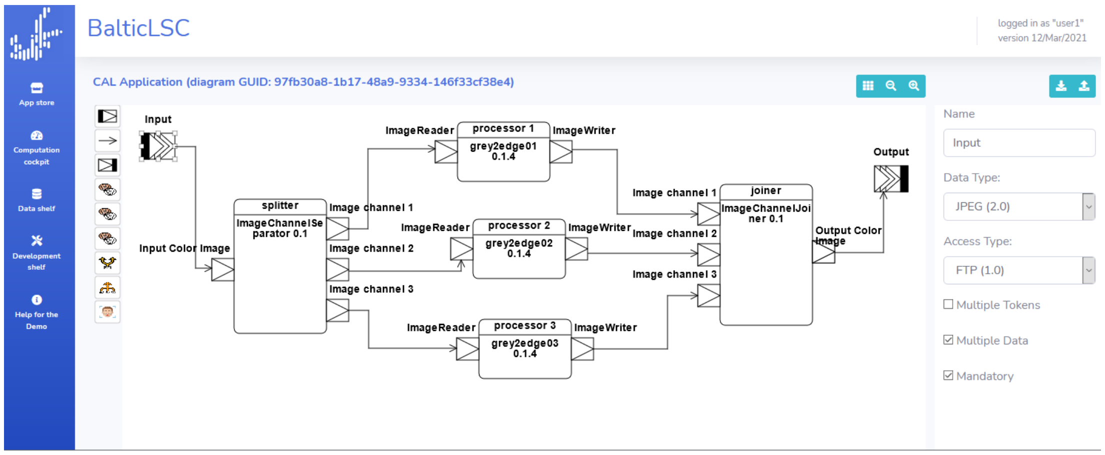
IV. Related work and contribution

As a recent study shows, only a small number (5%) of HPC users use a visual language [6]. This is despite that several relevant visual notations were proposed already since the beginning of the 1990's [7], [8]. More recent approaches are based on the model-driven paradigm [9] and offer cloud-based development environments [10]. Still, none of these approaches offer a comprehensive end-to-end solution, where HPC applications are developed using a visual language and instantly executed in a distributed computation environment.

The main contribution of BalticLSC is its coherent integration of a visual HPC application development language with a distributed network of HPC computation resources. BalticLSC serves as a high-level web portal for HPC [11]. Unlike for other such solutions, it does not necessitate any low-level constructs associated with application deployment and execution [12]. Being a low-code system, it hides all technical complexity behind the semantics of our visual language CAL. Moreover, by modularising the computation algorithms, the system facilitates their exchange and reuse.

V. Validation and future work

Currently, the BalticLSC system is being evaluated through case studies in research and industry settings. The initial results show that the visual language and the system are easy to use by research teams with limited programming experience. Still, in non-standard cases, traditional programming of computation modules (basic blocks for CAL applications) is needed. Thus, our future work will concentrate on facilitating the development and reuse of modules. This will include providing easy to follow, standard module templates for traditional programming and the introduction of an extension to CAL that would allow to develop module contents visually. In addition, we plan to extend the capabilities of the platform by providing an extended library of ready computation modules, a "market" for module exchange and improved facilities for connecting external computing centres.



CAL Application (diagram GUID: 97fb30a8-1b17-48a9-9334-146f33cf38e4)

Name: Input

Data Type: JPEG (2.0)

Access Type: FTP (1.0)

☐ Multiple Tokens

☑ Multiple Data

☑ Mandatory

Fig. 4. Screenshot from BalticLSC, showing the CAL Editor

Acknowledgement

This work is partially funded by the European Regional Development Fund, Interreg Baltic Sea Region programme, project BalticLSC #R075. The CAL editor was developed in collaboration with IMCS University of Latvia.

References

[1] R. Frost, “High-performance visual programming environments: Goals and considerations,” *ACM SIGGRAPH Computer Graphics*, vol. 29, no. 2, pp. 45–48, 1995.

[2] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, “Supporting the understanding and comparison of low-code development platforms,” in *46th Euromicro Conference on Software Engineering and Advanced Applications*, 2020, pp. 171–178.

[3] J. Cabot, “Positioning of the low-code movement within the field of model-driven engineering,” in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020.

[4] *OMG Meta Object Facility (MOF) Core Specification, version 2.4.1, formal/2013-06-01*, Object Management Group, 2013.

[5] K. Slonneger and B. L. Kurtz, *Formal Syntax and Semantics of Programming Languages*. Addison-Wesley, 1995.

[6] V. Amaral, B. Norberto, M. Goulão, M. Aldinucci, S. Benkner, A. Bracciali, P. Carreira, E. Celms, L. Correia, and C. Grelck, “Programming languages for data-intensive HPC applications: A systematic mapping study,” *Parallel Computing*, vol. 91, p. 102584, 2020.

[7] P. Newton and J. C. Browne, “The CODE 2.0 graphical parallel programming language,” in *Proceedings of the 6th international conference on Supercomputing*, 1992, pp. 167–177.

[8] J. C. Browne, S. I. Hyder, J. Dongarra, K. Moore, and P. Newton, “Visual programming and debugging for parallel computing,” *IEEE Parallel & Distributed Technology: Systems & Applications*, vol. 3, no. 1, pp. 75–83, 1995.

[9] M. Palyart, I. Ober, D. Lugato, and J.-M. Bruel, “HPCML: A Modeling Language Dedicated to High-Performance Scientific Computing,” in *Proc. 1st International Workshop on Model-Driven Engineering for High Performance and CLoud computing - MDHPCL12*, 2012, pp. 1–6.

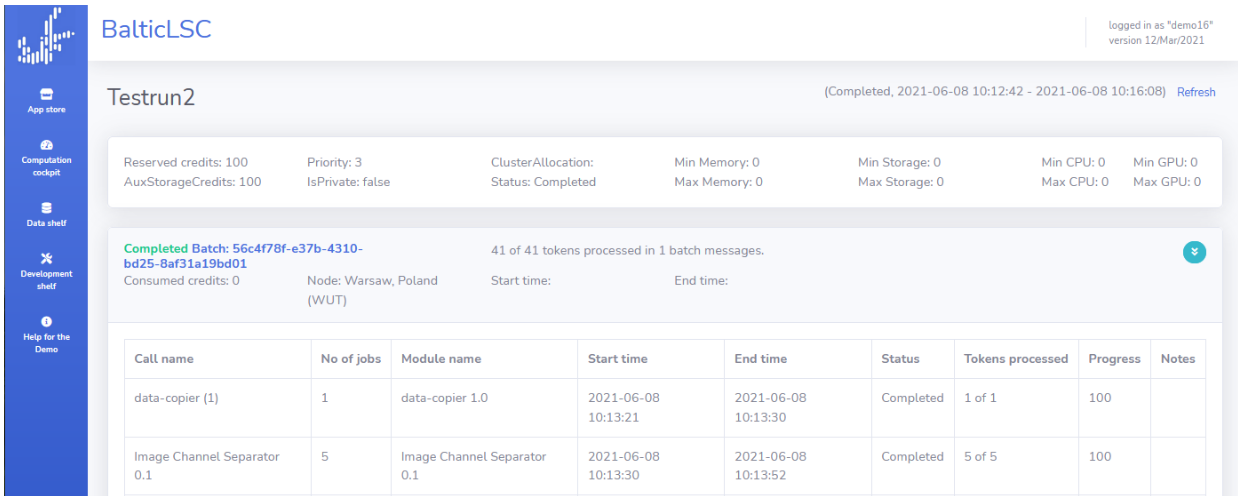
[10] J. L. Quiroz-Fabián, G. Román-Alonso, M. A. Castro-García, J. Buenabad-Chávez, A. Boukerche, and M. Aguilar-Cornejo, “VPPE: A novel visual parallel programming environment,” *International Journal of Parallel Programming*, vol. 47, no. 5, pp. 1117–1151, 2019.

[11] P. Calegari, M. Levrier, and P. Balczyński, “Web portals for high-performance computing: a survey,” *ACM Transactions on the Web (TWEB)*, vol. 13, no. 1, pp. 1–36, 2019.

[12] C. Bunch, N. Chohan, C. Krintz, and K. Shams, “Neptune: A domain specific language for deploying HPC software on cloud platforms,” in *ScienceCloud’11 - Proceedings of the 2nd International Workshop on Scientific Cloud Computing*, 2011, pp. 59–68.

Appendix

Sample screenshots from the running system are included in Figures 4–6. The working system is available at https://www.balticlsc.eu/.



logged in as "demo16"
version 12/Mar/2021

Testrun2

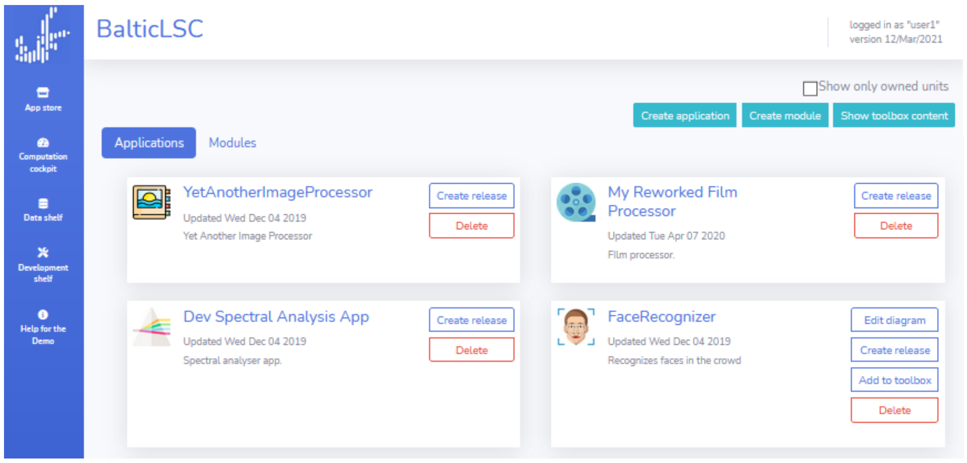
(Completed, 2021-06-08 10:12:42 - 2021-06-08 10:16:08)

Reserved credits: 100	Priority: 3	ClusterAllocation:	Min Memory: 0	Min Storage: 0	Min CPU: 0	Min GPU: 0
AuxStorageCredits: 100	IsPrivate: false	Status: Completed	Max Memory: 0	Max Storage: 0	Max CPU: 0	Max GPU: 0

Completed Batch: 56c4f78f-e37b-4310-bd25-8af31a19bd01
41 of 41 tokens processed in 1 batch messages.
Consumed credits: 0 Node: Warsaw, Poland (WUT) Start time: End time:

Call name	No of jobs	Module name	Start time	End time	Status	Tokens processed	Progress	Notes
data-copier (1)	1	data-copier 1.0	2021-06-08 10:13:21	2021-06-08 10:13:30	Completed	1 of 1	100	
Image Channel Separator 0.1	5	Image Channel Separator 0.1	2021-06-08 10:13:30	2021-06-08 10:13:52	Completed	5 of 5	100	

Fig. 5. Screenshot from BalticLSC, showing the Computation Cockpit



logged in as "user1"
version 12/Mar/2021

YetAnotherImageProcessor
Updated Wed Dec 04 2019
Yet Another Image Processor

My Reworked Film Processor
Updated Tue Apr 07 2020
Film processor.

Dev Spectral Analysis App
Updated Wed Dec 04 2019
Spectral analyser app.

FaceRecognizer
Updated Wed Dec 04 2019
Recognizes faces in the crowd

Fig. 6. Screenshot from BalticLSC, showing the Development Shelf