Workflow Editor for Definition and Execution of Parallelized Earth Data Processing Algorithms

Constantin Nandra
Technical University of Cluj-Napoca
constantin.nandra@cs.utcluj.ro

Dorian Gorgan
Technical University of Cluj-Napoca
dorian.gorgan@cs.utcluj.ro

ABSTRACT
The aim of this paper is to present a workflow editor application meant to facilitate the description of parallelizable Earth Data processing algorithms. The application was developed as part of the BigEarth project, whose overall aim was to improve the execution time of large Earth Observation data sets by spreading the processing effort over a distributed, high-performance computing network. To achieve a simple and flexible manner of partitioning the parallel parts of the algorithms between different processing nodes, we chose to represent them using a simple, workflow-based model. Since one of the main purposes of the project was to provide a simple and intuitive way of creating algorithms descriptions, we chose to implement a Domain Specific Language coupled with a visual workflow representation used for providing feedback. Throughout this paper, we will be demonstrating the use of the workflow editor together with the description language in order to define and execute distributed data processing algorithms. We will be highlighting the effectiveness of the application as a means of providing processing algorithms by analyzing its behavior under various stress tests.

Author Keywords
Interactive processing; workflow description; earth data

ACM Classification Keywords
Interactive applications

INTRODUCTION
Society's ability of acquiring new data has always been a challenge to its capabilities for processing them. This is especially true nowadays, in the information age. New trends, such as the internet of things, continually push the limits of data volumes and acquisition rates. These attributes are indeed some of the challenges of the Big Data phenomenon.

Increasing data volumes and acquisition rates can prove ever more challenging to small and medium level organizations, lacking the resources to properly store and manage them. This is especially true in the context of the newly developing technologies being able to derive increasing value from said data.

While increasing data volumes and acquisition rates are seen as generalized problems, some domains, in particular, could prove especially affected, due to the nature of the data they deal with. One such domain is that of the Earth Observation (EO) sciences. Scientists working in EO-related fields have to deal mostly with large resolution, multi-layered satellite and aerial photographic data. Processing these types of data in an effective manner is often beyond the capabilities of most standalone workstations – with respect to available memory, processing power and sometimes even raw storage. To address this problem, the BigEarth project [1] proposes the employment of a distributed computing infrastructure meant to take advantage of the task and data parallelism identified within the processing algorithms. One of the main challenges lies in defining the tasks in such a way as to be able to leverage said parallelism. At the same time, the task description methodology should prove accessible enough to allow the system to effectively cater to the needs of as large a user pool as possible. This is where the application described within the present paper comes in. It aims to provide the common user, lacking in programming expertise, with a workflow-based algorithm description methodology. Its key attributes should allow for the easy and intuitive definition of processing algorithms and, at the same time, allow the distributed platform at the other end to effectively leverage possible parallelism opportunities.

RELATED WORKS
The early versions of Geographical Information Systems (GIS), such as GRASS GIS [2] were little more than collections of software packages and executable programs developed for very specific tasks. GRASS, for example was initially designed to serve as a land management tool for the US military. Their very nature implied some expertise from the user's part, likely involving at least a basic level of programming knowledge, seeing as they usually offered sets of Unix shell commands as primary means of user interfacing. This practice has carried on to modern times and is still encountered in newer systems, such as ArcGIS [3] and QGIS [4].

By its very nature, the work done by the GIS solutions has a visual characteristic. Indeed, virtually all modern solutions offer some kind of visual-based processing methodology, be it for the definition and execution of processing tasks, the selection and manipulation of data or, sometimes, for

both types of scenarios. The benefits of employing a Graphics User Interface (GUI) are fairly obvious, since this practice tends to enlarge the system's user's pool, by eliminating the requirement for more technical skills from the user's part. It should be noted, however, that the GUI is not really the silver bullet when it comes to defining processing tasks. The best argument to support this claim is the lingering and still strong support for scripting languages offered by most GIS solutions out there, both commercial and free or open-source. Scripting support tends to cater to more experienced users, allowing for quicker access to the desired functionality. All the while, GUI-based approaches can have their own shortcomings, particularly complex ones, presenting cluttered interfaces and requiring steep user leaning curves.

While preserving scripting support as a way of user interaction, the majority of the newer GIS solutions have since made the transition from the traditional shell scripting environment to a more user friendly approach, based on Python scripting. There are certain benefits to be reaped, especially since Python is a much higher-level programming/scripting alternative when compared to the Unix shell environment. Among these benefits, one could count, lower user training effort and increased productivity. This is the case with all the aforementioned systems.

Google's Earth Engine [5] is another major GIS solution available nowadays. True to the practice of offering script-based process description capabilities, it offers both Python and JavaScript APIs (Application Programming Interfaces).

Continuing to develop the potential of language-based process descriptions, recent research efforts have started looking into the employment of dedicated, Domain Specific Languages (DSL) for defining processing tasks. These efforts span a variety of research fields, while having in common the processing of large data sets. One such example is described within [6]. The authors present the characteristics of a distributed processing solution which relies on a specialized DSL in order to define the data processing tasks. Much like in the case of our proposed solution, it aims to remove the necessity for parallel programming expertise from the user's part. This would normally be required in order to take advantage of a distributed, parallel processing setup. Instead, the proposed solution employs the DSL description like an abstract model, relying on the system to generate distributable processing tasks.

Another interesting use of a DSL is described within [7]. The language is called Vivaldi and is employed to describe processing tasks dealing with the analysis of medical tissue imagery.

In [8], the authors present another DSL, employed for the purposes of defining processing algorithms for dealing with high-throughput telescope and microscope image data. The language is called Diderot and it offers a C-based syntax. It was designed specifically to be able to break the processing algorithm into smaller, parallelizable sub-processes having as end goal the overall shortening of the processing time.

WORDEL EDITOR APPLICATION

The BigEarth platform [1] was designed and built with the express purpose of handling large amounts of data processing tasks by taking advantage of the capabilities of a distributed, multi-processor computing network. To receive its processing tasks, it relies on a standalone user application, henceforth referred to as the WorDeL Editor Application (WEA). This acts as the user's interface with the system, providing the means by which said user can:

- Define processing algorithms in the form of workflows
- Launch processes into execution
- Retrieve the processing results

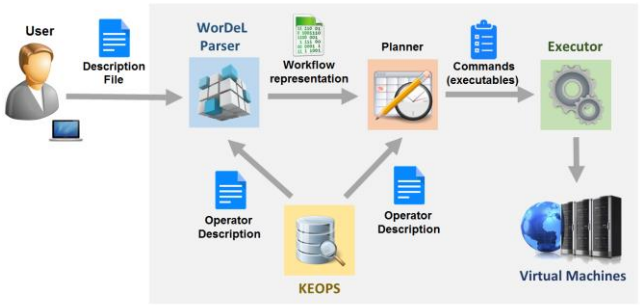

Figure 1: The architecture of the BigEarth platform

BigEarth is meant to provide an easily-accessible big data processing solution, catering to users who do not necessarily have much in the way of programming experience. Therefore, the need for a simple and intuitive manner of defining the processing algorithms constituted one of the most important requirements [9]. This is the main reason why we have opted for the employment of a Domain Specific Language (DSL) with a simplified structure, focused on the definition of workflow models. Compared to a standard programming language, this DSL is far more simplified, as it has no need for many of the features that the average programming language offers to its users. Such features might include constructs for working with memory, support for complex data structures and object-oriented programming or other advanced, high-level constructs such as lambda expressions.

In order to make the interface more intuitive and simpler to employ by new, untrained users, the workflow description language is meant to be coupled with a visual representation of the processing workflow. The general aim is to offer an interactive experience, showing operator nodes and connections as defined and modified by the user, thus reinforcing the degree of intuition when working with the model. The overall goal is to increase the application's usability, as this would directly correlate to an increase in the effectiveness of the processing platform.

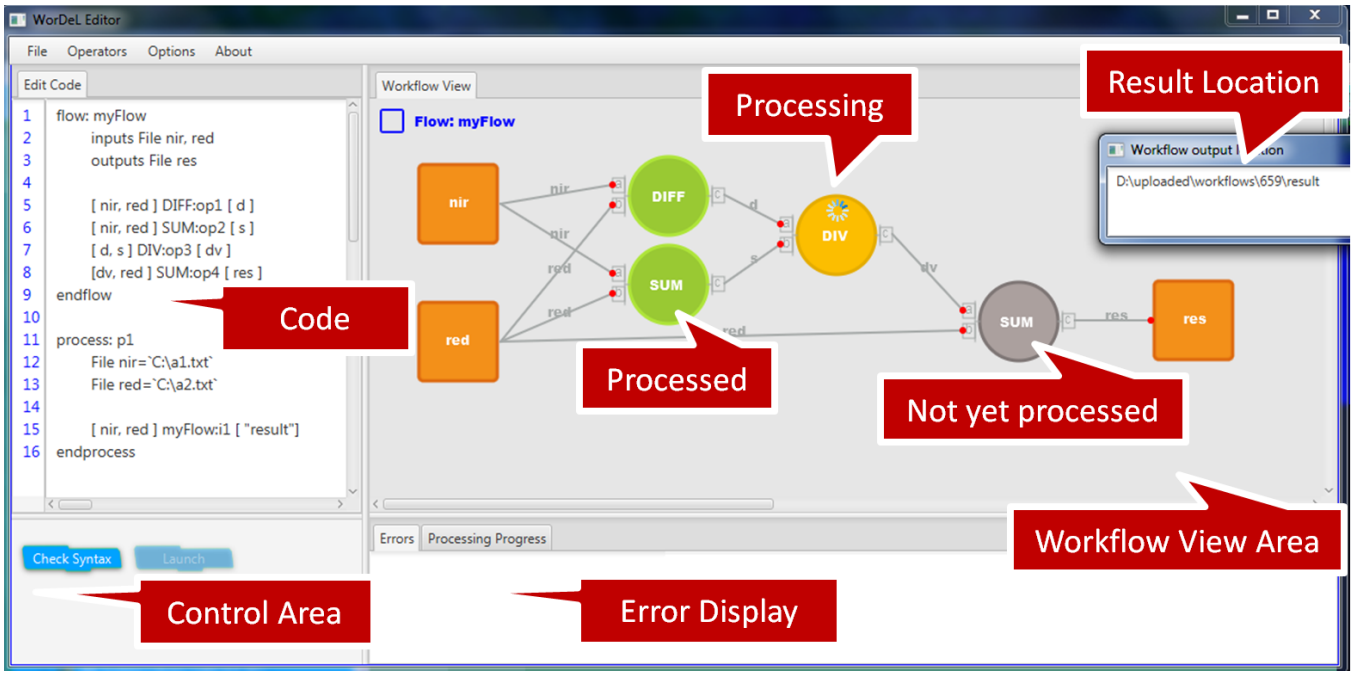

Figure 2: Screenshot of the WorDeL Editor application.

Figure 2 shows a screenshot of the editor application. Its two most important features correspond to its design purposes: constructing workflow descriptions and monitoring their execution. Workflows are created through the use of the WorDeL (Workflow Description Language) DSL, a compact and visually intuitive means of representation, especially when coupled with a visual rendition of the result. At its core, WorDeL is not as much a programming language, but more of an all-round scripting language, providing a way of linking together existing functional elements offered in the form of operators. The operators can either be pre-established, shipped together with the application, or they can be created directly by the user to suit domain-specific needs [10]. This design decision adds flexibility to the description methodology, as it is no longer bound to a pre-defined set of existing operators and can be used to tackle processes from multiples fields of study.

The center piece of the application is a basic code editor (Figure 2, left) allowing for the writing and processing of WorDeL code. This is supported by three main elements. The first, and most obvious of these, is the workflow visualization area. It provides a simple and intuitive visual representation of the workflow defined within the code editing area. This is meant to help the user better grasp the structure of the workflow being defined, by clearly laying out the sub-processes of the algorithm and the data flow between them. Its second, equally important, role is that of monitoring process execution. After the user loads the input data and launches the process into execution, the workflow representation area will monitor and update, in real time, the status of each processing node and data connection. The status is indicated by different color themes applied to the visual elements (as shown in Figure 2 – right). In addition to these roles, this element also allows the user to quickly access the files resulting from the workflow's execution with a single click on the relevant connection or workflow output port.

The second element of note is the error display area. The editor application interfaces directly with the language parser component and receives feedback from it pertaining to the description analysis process. The error display area compiles the feedback data into error and warning reports, allowing the user to recognize and correct any syntax and semantic errors that might appear within the description code.

The third auxiliary element has to do with the management of the operator collections. Without some kind of mechanism for indexing, searching and sorting through existing operator collection, their employment within user-defined workflows would be impractical, if not next to impossible. The operator management mechanism is not directly visible in the figure. Its functionality can be accessed through the *Operators* menu. In the current version, it involves a basic operator navigation panel allowing for operations like keyword searching, category filtering and operator data analysis (Figure 3).

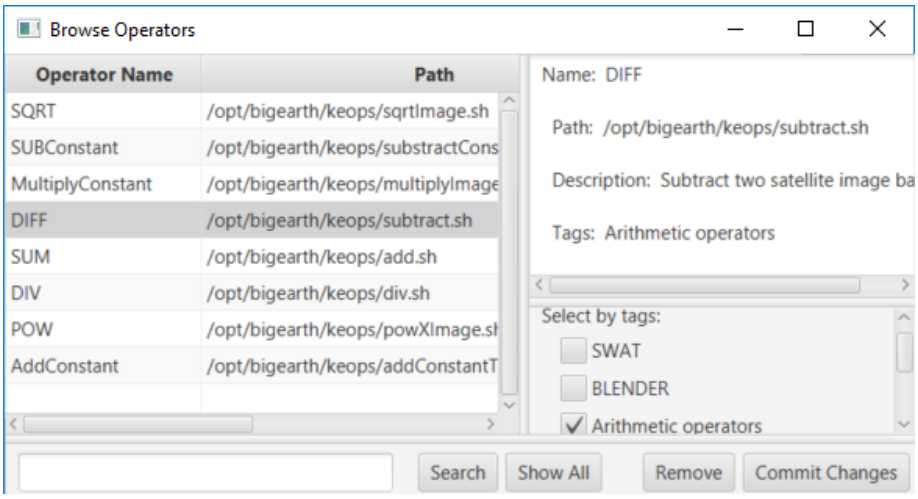

Figure 3: Operator browsing and sorting functionality

WORKFLOW EXAMPLES

In this section we will showcase the capabilities of the WorDeL Editor application by following its functionality in the definition of a set of workflow processes.

The first workflow example represents a simple formula meant to compute the NDVI (Normalized Vegetation Index) (1) value for a pair of input image bands. This has been the subject of our experiments before, within [11] and [12] while demonstrating WorDeL's capabilities for the definition and execution of batch-processing tasks exploiting data parallelism. Within this paper, the NDVI workflow will be the baseline example upon which we will build in complexity in order to showcase the functionality of our application.

$$NDVI = \frac{NIR - RED}{NIR + RED} \quad (1)$$

The NDVI is implemented in our example using three arithmetic operators, working at the pixel level (as shown in Figure 4).

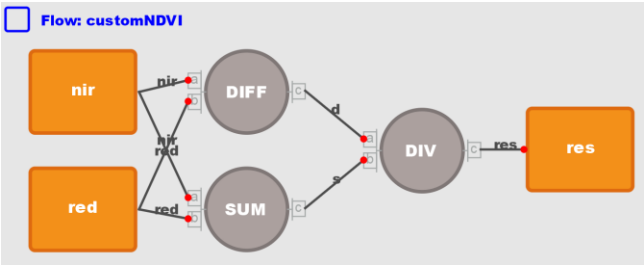

Figure 4: WorDeL Editor – NDVI workflow

One of the most important features of WorDeL is that it allows for the referencing and reuse of externally defined workflows into new designs. This is illustrated in the workflow shown in Figure 5. In this case, the current workflow (*NDVIApp*) incorporates the functionality of the previously defined NDVI workflow to compute the index and then takes the resulting image and feeds it into a pseudo-coloring operator. This operator takes a color palette and applies it to cover certain interval values of the NDVI result. The final result of this workflow will be a colored NDVI image, easier to interpret by the human eye.

When dealing with externally referenced workflows, such as this, WEA makes use of its integrated parser component to read and process the contents of any and all referenced files, creating a temporary database which indexes all workflows defined within those files. This database is then used to be able to recognize any external workflows employed by the user as processing nodes and map them correctly within the current workflow. A welcome side-effect of this indexing mechanism is that it allows access to the internal workings of the referenced external workflows. WEA exploits this, allowing the user to display the contents of any such workflow by selecting it within the display area. This is beneficial to the user, since it allows for easier exploration of embedded workflows in an interactive manner.

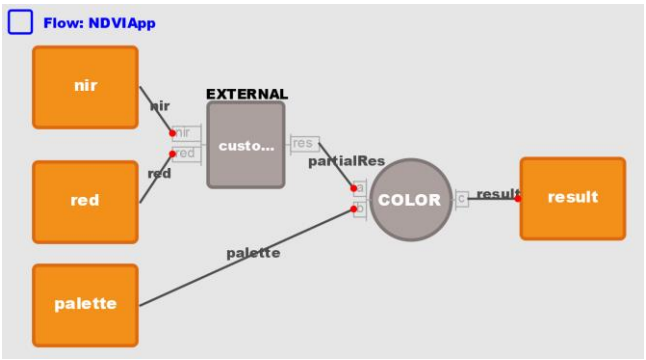

Figure 5: WorDeL – externally referenced workflow example

The mechanism for workflow inclusion and visualization works on multiple recursive levels. The number of levels is only limited by the capabilities of the machine and the available resources. Ultimately, it is also a matter related to the effectiveness of the application in complex usage scenarios, as highlighted towards the end of this paper.

In order to maximize its processing effectiveness, the BigEarth platform allows for the batch processing of massive data loads with little to no extra effort required from the user's part. For this reason, we envisaged the introduction of a repetitive statement meant to facilitate the running of a given process with different data sets. This is the role of the *for-each* statement. Its employment and usefulness have been demonstrated within [11] and [12]. In our example, we make use of this construct to apply the previously-defined NDVI workflow on a series of input images. As seen in Figure 6, the input is given in the form of two sets, representing the near-infrared and red band images.

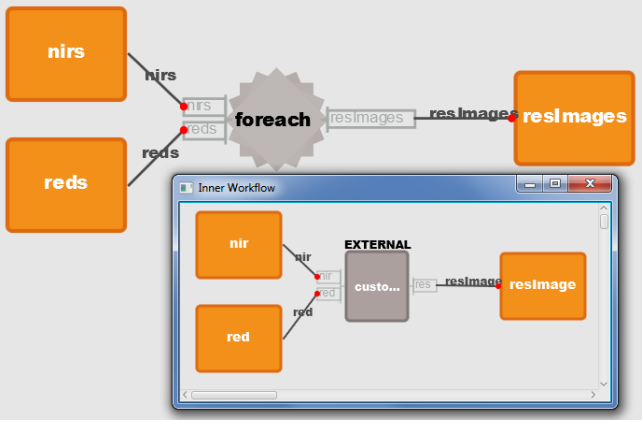

Figure 6: WorDeL – FOR-EACH repetitive workflow

To accommodate the need for repetitive processing, WorDeL offers the List data type, allowing for the aggregation of similar data items into collections. These are then and then fed as input to the *for-each* element. WEA allows its users to specify the data lists explicitly, within the WorDeL code (Figure 7) or, if needed, it can read the list of files from a user-provided indexing file.

The *for-each* node repeatedly applies a given workflow or workflow portion on different data sets. In our example, the repeated functionality is represented by the NDVI workflow. This construct then iterates over the two input lists, creating input pairs with members from each list. It then applies said input pairs as inputs to the replicated workflow, creating multiple, distinct processing instances. Once all the processes are completed, the application will take care of collating the results into a result list, making its location available to the user. This is done in a completely transparent manner from the user's point of view. It has the obvious benefit of automating the definition and execution of batch processing tasks, generated from large data sets with high refresh rates, such as in the case of EO data. This kind of functionality goes to show the potential to be achieved by combining a simplified workflow modelling language with a dedicated support application like the workflow editor.

```
1   #include "NDVI.wdl"
2
3   flow: forFlow
4         inputs List<File> nirs, reds
5         outputs List<File> resImages
6
7         [ nirs:nir, reds:red ] for-each [ resImages: resImage]
8               [ nir, red ] customNDVI:o1 [ resImage ]
9         endfor
10
11  endflow
12
13  process: p2
14        List<File> nirs={"C:\a1.txt","C:\a2.txt"}
15        List<File> reds={"C:\b1.txt", "C:\b2.txt"}
16
17        [ nirs, reds ] forFlow:i1 [ "resss.tiff"]
18  endprocess
```

Figure 7: FOR-EACH workflow - WorDeL syntax

EFFICIENCY AND USABILITY EVALUATION

As a measure of the WEA's usability we have decided to prove its viability as a process description tool by subjecting it to a series of stress tests involving the development of increasingly complex workflow scenarios. These tests are concerned with the application's ability to perform its tasks in conditions involving high-level, multi-layer processing workflows. They closely follow the application's usage of the CPU and memory resources. Given that the application was developed in Java, we have employed the *VisualVM* tool in order to monitor the resource utilization within the java virtual machine (JVM). The tests have been run on a single, mid-tier, Windows-based desktop workstation, with a four-core 3.20GHz CPU, 8GB of RAM and a maximum JVM heap size of 2GB.

These tests have been performed as part of the BigEarth project in order to ascertain the viability of the platform at the user interface level, under various stress conditions dictated by the complexity of the workflow models. The following sections will present our results.

Workflow Operator Capacity

This is the most basic test, and it attempts to determine the maximum number of operators within a single workflow that the application can manage to process while maintaining interface responsiveness. We henceforth define the application's behavior as meeting the responsiveness criterion if the application manages to process the workflow description within a five seconds time limit.

For this test, we have procedurally generated WorDeL workflow descriptions housing increasing numbers of operators. The following figures present the relationship between the number of nodes within a given workflow and the resources required by the editor application in order to process them. Figure 8 and Figure 9 show the memory and CPU loads observed while monitoring the resource consumption of the java virtual machine.

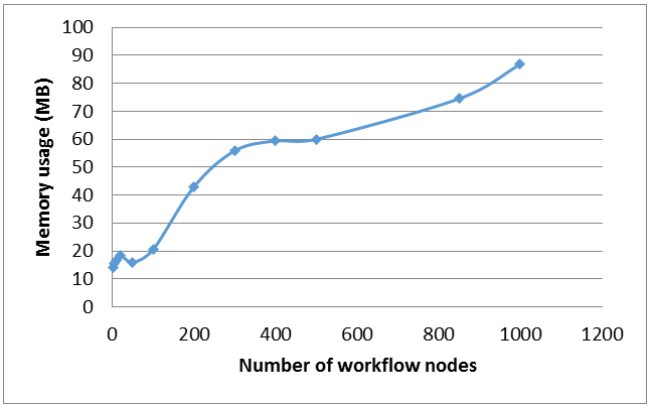

Figure 8: Operator capacity – memory load

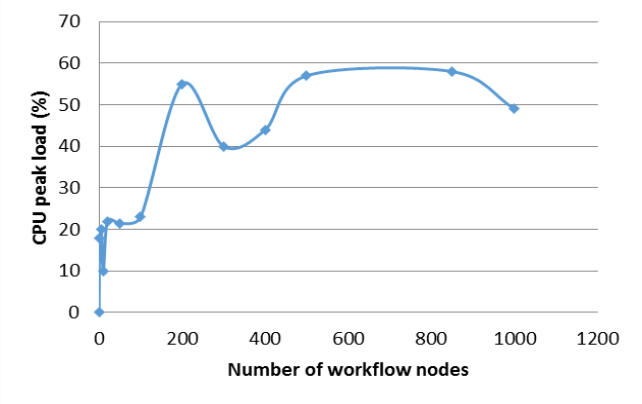

Figure 9: Operator capacity – CPU load

This test revealed promising results, with the application proving able to handle up to 1000 operator instances within a single workflow description file. As seen in the charts, the CPU utilization evened out at about 60%. After a threshold of around 1050 operators, the application itself would become blocked and cease to respond. Following an analysis of the program, we found that the limiting factor was with the applications display area, which was unable to generate visual representations for workflows with more than 1050 nodes. A number of 1000 operators within a workflow is however, more than reasonable, since such a workflow would really be too complex for any user to grasp. This number of operators comes with pretty modest memory requirements (roughly 90MB). Looking at the memory usage chart, one can notice a steep increase after about 200 nodes per workflow (circa 43MB of JVM memory). However, in terms of workflow size, even this is quite a large number, as in real world scenarios users would break up such complex designs to make them more manageable.

Included Workflows – Horizontal Test

The second test starts from the reasonable assumption that users would tend to fragment larger, complex workflows into smaller pieces and make use of the include mechanism in order to make their designs more modular and easier to understand and employ. Therefore, the test was intended to determine the upper limit of the number of include statements within a single workflow description file. In this case, each include statement represented one external file, containing one workflow description.

The results of this test proved particularly successful, as seen at first glance at the figures 10 and 11. We managed to process workflows with upwards of 2200 included files, before reaching the responsiveness limit of 5 seconds for the main workflow. In the case of this test set we got quite significant memory hogging, since the application needed to open, parse and store the contents of each description file - and it does that for upwards of 2200 files.

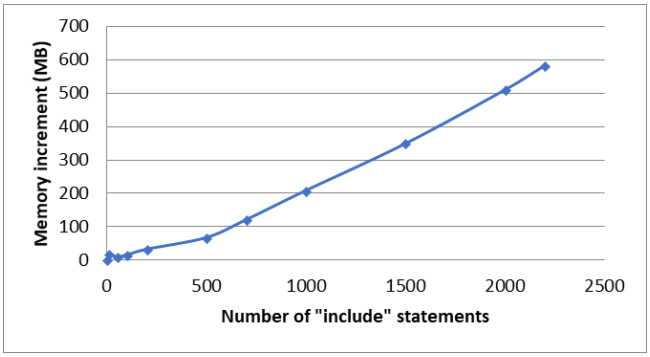

Figure 10: Horizontal include test – memory load

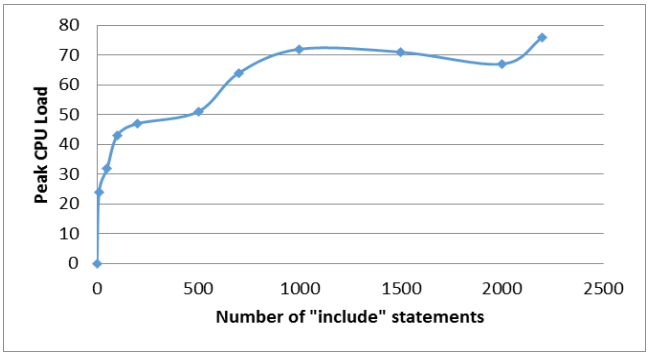

Figure 11: Horizontal include test – CPU load

While the resources required to reach the 2200 included workflows limit are not negligible, the fact that the application can achieve this has relevance for the description method as a whole. This is because, to build workflows of even moderate complexity, a user needs to be able to partition the design and rely on externally referenced workflows to even stand a chance of grasping its

functionality. Thus, the development of ever more complex included workflow networks is bound to put a strain on the application's capabilities to process them. In light of the obtained results, we ascertain that a number of 2200 included workflows is a reasonable achievement which could allow for the definition of medium to high-complexity designs without much of an effect on the application's responsiveness.

Included Workflows – Vertical Test

The previous test was meant to ascertain the number of maximum allowable included workflows within a single description file – a horizontal, or breadth-wise test. To test the capabilities of the include mechanism, we decided to also perform a vertical, or depth extension test. To this end, we have generated workflows with increasing numbers of include levels. To illustrate an example, we've defined a workflow description file containing one reference to an external file as being a two-level include workflow. Following our tests, we managed to process workflows of up to a depth of 2000 include levels.

As seen in figures 12 and 13, we managed to get similar results to the horizontal tests, both in terms of included workflow numbers and resource requirements.

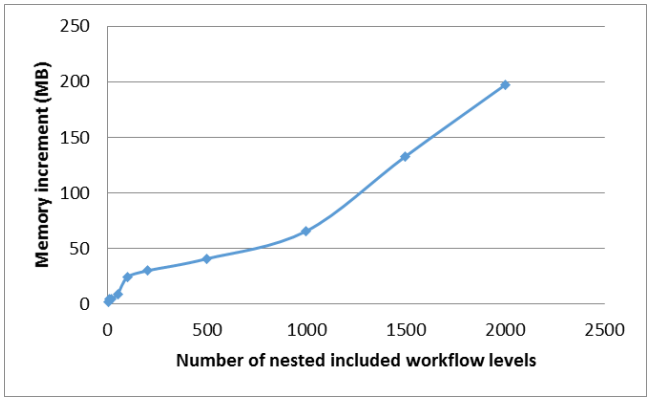

Figure 12: Vertical include test – memory load

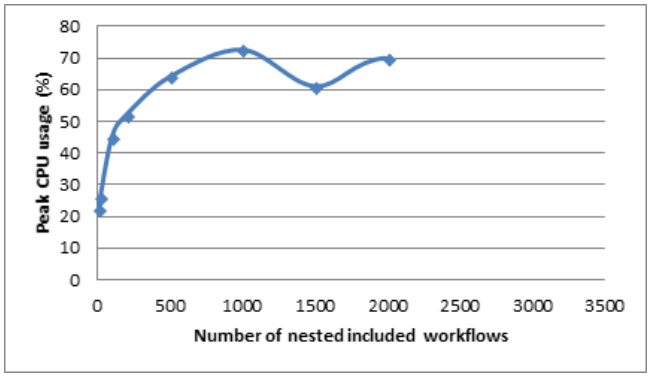

Figure 13: Vertical include test – CPU load

Nested Workflow Repetitions

The fourth and final test set was aimed at gauging the ability of the application to service the needs of the user in terms of the for-each loop employment (as mentioned within the last section). Specifically, we wanted to determine the application's ability to deal with multiple, nested for-each loops. The *for-each* example showcased previously was a one-level loop. A two-level nested loop would involve the processing of a *for-each* workflow inside another *for-each* element. This raises a number of issues, the first of which would be the required inputs. A two-level for-each loop would work on a list containing lists of elements. Therefore, even such a nester loop could prove challenging to understand and effectively employ, with three and four level loops probably being the maximum actual useful depth.

Even though the usable depth level of a *for-each* loop would probably be situated at about two or three, we wanted to prove the capabilities of the application and the language itself. Therefore, we started by dynamically generating input file lists, and incrementally building multi-level lists, which we employed as inputs for multi-level *for-each* loops.

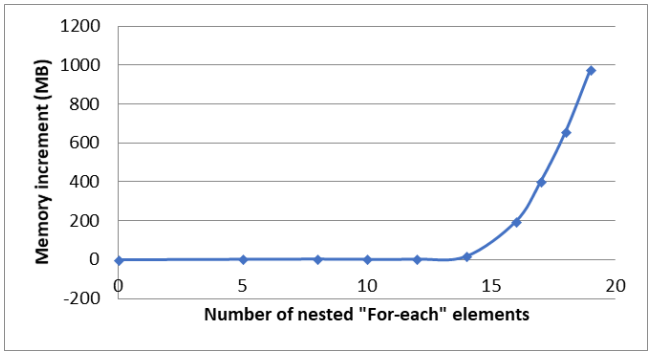

Figure 14: Nested for-each loop test – memory load

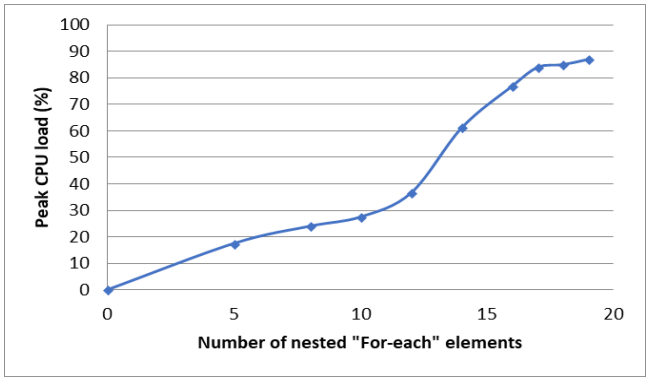

Figure 15: Nested for-each loop test – CPU load

As seen in figures 14 and 15, we managed to process workflows of up to 19 nested loops, by which point we got to the limits of our available resources. In this case, the memory load was the first to reach its limit, with CPU loads

following closely behind. Considering the previously mentioned facts, this is a pretty impressive result, as its shows the potential of the WorDeL language and its application in helping to define high-level process descriptions.

CONCLUSIONS

To increase the effectiveness of the workflow modelling methodology employed within the BigEarth platform, we have developed a support editor application, designed to help the user visualize and modify the workflow model in a simplistic and intuitive manner. The application provides close support for visualizing the workflow model, allowing for in-depth, multi-layered workflow analysis. This is meant to offer a simple, interactive and intuitive manner of defining processing algorithms, without requiring any programming experience from the user's part.

Throughout the paper we have touched upon the editor application's design and purpose and have shown its use in helping the user develop workflows of increasing levels of complexity. While simple workflows certainly have their merits in that they are quick and intuitive to set up, sometimes there are scenarios in which more complex setups might prove more effective. Such is the case of the workflow include and for-each mechanisms. In these situations, relying on the specialized model visualization tools offered by our application can greatly help the user in grasping the inner workings of a given design.

The stress test that we ran on the editor application show that it can easily provide support for the definition and processing of workflows consisting of considerable numbers of elements and levels of complexity. With these results, one ca conclude that our application can help provide a suitable interface between the user and the high-performance processing network constituting the BigEarth platform. Ultimately, the platform as a whole can provide users lacking programming skills with a simple and effective way to build and run parallelizable tasks over large data volumes while taking advantage of the capabilities offered by a network of processing nodes.

ACKNOWLEDGMENTS

The application employed for the experiments presented within this paper was developed as part of a research project supported by ROSA (Romanian Space Agency) through Contract CDI-STAR 106/2013, BIGEARTH – Flexible Processing of Big Earth Data over High Performance Computing Architectures. The scientific consultancy and technology transfer has been supported by MEN-UEFISCDI through Contract no. 344/2014, PECSA – Experimental High Performance Computing Platform for Scientific Research and Entrepreneurial Development.

REFERENCES

1. BIGEARTH Project Home [Online] - http://cgis.utcluj.ro/projects/bigearth/
2. GRASS GIS [Online] - http://grass.osgeo.org/documentation/general-overview/
3. ArcGIS [Online] - https://www.esri.com/en-us/arcgis/products/index
4. QGIS [Online] - https://qgis.org/en/site
5. Google Earth Engine - https://earthengine.google.com/faq
6. M. Krämer, I. Senner, "A modular software architecture for processing of big geospatial data in the cloud", in *Computers and Graphics*, vol. 49, pp. 69-81, 2015
7. H. Choi, W. Choi, T.M. Quan, D. G. Hildebrand, H. Pfister, W. K. Jeong, "Vivaldi: A Domain-Specific Language for Volume Processing and Visualization on Distributed Heterogeneous Systems", in *IEEE transactions on visualization and computer graphics*, pp. 2407-2416, 2014
8. Chiw, C., Kindlmann, G., Reppy, J., Samuels, L. and Seltzer,N., 2012. Diderot: A parallel dsl for image analysis and visualization. Proceedings of the *33rd ACM SIGPLAN Conference on Programming Language Design and Implementation* 47(6), pp. 111–120.
9. V. Bacu, T.. Stefanut, D. Gorgan, "Adaptive Processing of Earth Observation Data on Cloud Infrastructures based on Workflow Description", in *Proceedings of the Intelligent Computer Communication and Processing* (ICCP), pp.449-454, 2015.
10. D. Mihon, V. Bacu, V.D. Colceriu, D. Gorgan, "Modeling of Earth Observation Use Cases through the KEOPS System", *in Proceedings of the Intelligent Computer Communication and Processing* (ICCP), pp.455-460, 2015.
11. C. Nandra, D. Gorgan, "Defining earth data batch processing tasks by means of a flexible workflow description language" in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-4, pp. 59-66, July 2016.
12. C. Nandra, V. Bacu, D. Gorgan, "Parallel Earth Data Processing on a Distributed, Cloud Based Computing Architecture", in *Proceedings of the 21st International Conference on Control Systems and Computer Science* (CSCS), pp. 677-684, 2016