Soma-workflow: A unified and simple interface to parallel computing resources

S. Laguitton[1], D. Rivière[1,2], T. Vincent[1], C. Fischer[1], D. Geffroy[2], N. Souedet[2,3], I. Denghien[2], Y. Cointepas[1,2].
[1]CEA, I2BM, Neurospin, Gif-sur-Yvette, France/[2]IFR 49, Gif-sur-Yvette, France/[3]CEA, I2BM, MIRCEN, Fontenay-aux-Roses, France

Introduction

Neuroimaging research involves many computational processes applied on high dimensional data and could thus greatly take advantage of the increased availability of parallel computing resources: from multiple core machines to clusters or grids. Coarse-grained parallelism is well suited in most of the cases: it involves many long independent processes (from one minute to several days or weeks). Extensive work to introduce fine grain parallelism in algorithms is indeed worthless when the algorithm is intended to be applied on large sets of data. These coarse-grained parallelized processes can be described by workflows which are sequences of parallel and serial sub-tasks.

Soma-workflow is a Python application which aims at making easier the execution, control and monitoring of tasks and workflows on a wide range of parallel computing resources, through a simple and homogeneous Python application programming interface (API) and/or a graphical user interface (GUI). Soma-workflow was created for the purpose of being plugged to external software or being used directly by non expert users.

Soma-workflow and its interactions with parallel resources

The workflows are described by direct acyclic graphs: the nodes correspond to the sub-tasks and the arrows to the execution dependencies between sub-tasks. Each sub-task, also called job, wraps a program command line call. The workflows are created and then submitted to a computing resource using soma-workflow Python API. For each submitted workflow, soma-workflow deals with the execution of each job at the right time considering its dependencies. The status of workflows and jobs, as well as job standard outputs or exit information, are displayed in the GUI or can be recovered using the API. The workflows can be stopped at any time or/and restarted when needed.

For more flexibility, soma-workflow can be used as a simple single process application or as a client-server application. Compared to the former, the client-server application additionally offers a transparent access to remote computing resources and the possibility for the user to close soma-workflow at any time without impacting the execution of workflows. Tools are also available to handle file transfers and file path matchings in case the user's machine and the remote computing resource do not have a shared file system.

For the purpose of interacting with a wide range of parallel computing resources, soma-workflow uses only very basic parallel resource functionalities: job submission, job suppression and request for job status and exit information. Thereby, a new parallel resource can be plugged to soma-workflow by implementing only four Python methods. Furthermore, the interaction with many computing resources is already implemented in soma-workflow. In the case of a single multiple core machine, soma-workflow can be used directly: the maximum number of jobs which can run in parallel is adjusted by the user. In the case of distributed computing resources, soma-workflow interacts with DRMAA implementations (Distributed Resource Management Application API) [1] which can interface with a wide range of distributed resource management system. Using DRMAA, soma-workflow was used with success on clusters managed by Torque, LSF, Oracle Grid Engine and Condor.

Concrete use cases in neuroimaging

This paragraph illustrates the use of soma-workflow to take advantage of a 192 core shared cluster in three concrete use cases in neuroimaging. In the first use case, soma-workflow was used to execute a coarse-grained parallelized analysis: the joint detection-estimation of within-subject fMRI data [2]. The already existing code was wrapped in a workflow which splits the data, processes it and merges the results afterward. The analysis of a whole brain goes down from 10 hours on a single CPU to 15 mins on the cluster. In the second use case, soma-workflow was used to run regression tests in the process of reducing the sensitivity of the Morphologist pipeline of BrainVISA [3] to variability. The regression test workflow was generated from BrainVISA and consists in the processing of a sample database containing 80 T1 MR images. It runs in about an hour on the cluster against 23 hours on a single processor. In the third use case, soma-workflow was used to perform an extensive cross-validation of the older cortical sulci identification models of BrainVISA [4] . The complete validation represented about 5500 hours of computing (more than 7 months), and used about 70000 individual jobs. It ran in about 3 days using approximately 100 cores of the cluster. The validation brought a precise idea of the performance of the model and allows comparisons with other models such as the methods by Perrot [5].

Conclusion

Soma-workflow aims at bridging the gap between parallel computing resources and software or non expert end users. The three neuroimaging use cases presented here demonstrate its relevance in the research process. Indeed, coarse-grained parallelism can exist at the level of the analysis, as in the first use case. However, even if the method cannot be parallelized, the two other use cases show the relevance of using parallel resources to process large datasets, optimize, test and validate methods.

Acknowledgments. This work was funded by the ITEA2 HiPiP project. Developments relative to the implementation of JDE with soma-workflow were supported by the CL1-38093-007 Servier contract.

References

1. Troger, P., Rajic, H., Haas, A., Domagalski, P.: Standardization of an API for Distributed Resource Management Systems. In: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007). pp. 619–626 (2007).
2. Vincent, T., Risser, L., Ciuciu, P.: Spatially adaptive mixture modeling for analysis of within-subject fMRI time series. IEEE Trans. Med. Imag. 29, 1059–1074 (2010)
3. Cointepas, Y.: The BrainVISA project: a shared software development infrastructure for biomedical imaging research. In: Proceedings 16th HBM (2010)
4. Rivière, D., Mangin, J.-F., Papadopoulos-Orfanos, D., Martinez, J.-M, Frouin, V., Régis, J.: Automatic recognition of cortical sulci of the Human Brain using a congregation of neural networks. In:Medical Image Analysis. vol. 6, no. 2, pp. 77–92 (2002)
5. Perrot, M., Rivière, D., Mangin, J.-F.: Cortical sulci recognition and spatial normalization. In: Medical Image Analysis. In press (2011)