Enablement of multi-scale simulation, analytics and visualization workflows

Marc Casas, Miquel Moreto, Rosa M Badia, Raul Sirvent, Eduard Ayguadé, Jesús Labarta, Mateo Valero

During the last decades, simulation has been a very useful tool for the enablement of progress of science. Simulation allows to model a large number of phenomena in science and engineering, enormously reducing the number of hours and cost required in contrast with real experimentation or by being able to perform experiments that would not be possible in the real world. However, the path to simulate large and complex systems is still a challenge and one of the clear applications that will require exascale computing.

One of the aspects that add complexity to simulations is the so-called multi-scale simulation. Multi-scale simulators compose simulators at different levels of granularity (detail), from coarser to finer grains, switching between them whenever necessary in order to attain the required accuracy. For instance, in the framework of the Human Brain Project (HBP), BSC is proposing the extension of the PyCOMPSs programming and execution environment [1] to enable workflows that implement multi-scale simulation systems for neuroscience.

PyCOMPSs, the Python binding of COMPSs [4], is a coarse grain, task-based programming model aiming at the implementation of dynamic computational workflows. In these workflows, annotations for those methods that will become tasks and the directionality of their arguments are used to dynamically build a task graph; this task graph is used to automatically detect parallelism among tasks and properly offload them to the available and appropriate resources. COMPSs supports the execution of workflows in clusters, private, public and hybrid clouds, including specific platforms such as the Chameleon cloud or the EGI Federated cloud.

Implementing a multi-scale simulator workflow with PyCOMPSs is relatively easy, being each node of the workflow an instance of one of the required simulators. PyCOMPSs enables the coupling of different simulators, each of them possibly parallelized with MPI or MPI+X, being X your favorite node-level programming model for multicores and accelerators. The PyCOMPSs runtime will orchestrate the execution of the multiscale simulation, deciding when each simulator should be invoked and when/how the data that should be exchanged between different simulators. Each simulator will advance a number of iterations (timesteps for example) during each invocation and then stop until it is invoked again.

Under this scenario, one of the aspects that is key for the performance of the overall workflow is the ability to keep the state of a given level in the simulation when invoking another one. Keeping the state between simulation invocations would avoid the need to upload in memory all initialization data at the beginning of the task, which may represent an important amount of time. The web-service enabled version of COMPSs is able to keep the state of a task between different invocations. However,

we do not consider that using the web-service paradigm is the best option for HPC systems. To implement this idea, the current persistent workers of the COMPSs runtime will be extended with the possibility of keeping in memory the data-structures between different invocations.

To address this challenge, we are considering the use of new storage solutions, such as Hecuba [1] or dataClay [5], already integrated with PyCOMPSs. These solutions can largely benefit from the capabilities offered by new storage devices such as NVRAMs or other byte addressable devices, which are foreseen to be used in a near future.

In addition to high-demanding computationally intensive simulations, the neuroscience community would also like to steer them by interleaving analytics or visualization jobs while the main multi-scale simulation is running. As a result of executing these jobs, the course of the simulation could be reconfigured. Here we will pursue an approach where scheduling of applications will go from a single decision to book a set of resources, to a layered architecture of schedulers at different levels. We will consider four levels: kernel, application, node and job scheduler levels, that will be able to talk to each other and apply complex scheduling policies at different and several of these levels at the same time in order to achieve different objectives. For example, to increase the performance of one or several applications, maximize the system utilization per node, consolidate the usage of resources, exploit data locality, etc.

With this objective, the Dynamic Load Balancing (DLB [2]) library developed at BSC will coordinate with the corresponding job scheduler (SLURM or LSF) in order to give priority to visualization or analysis jobs. DLB is able to share CPUs between different jobs in a system, with the objective of balancing the load between them. Under this scenario, once an urgent visualization job is submitted, a request will be issued to DLB to lend CPUs from larger (and longer) simulation jobs. The visualization (or analytic) job will run for a while and when finished the CPUs will be given back to the simulation.

One of the key challenges to meet the requirements of this multi-scale simulations is the integration of storage and computation to handle workloads with very large memory storage needs. Indeed, a memory subsystem with 100PB of memory capacity and able to sustain 100PB/s of memory bandwidth is required to satisfy them. If we are constrained to a power budget of 20MW with 30% budgeted for the memory subsystem, designs exclusively based on the DRAM approach are not an option due to their large energy consumption. Instead, Non-Volatile Memories (NVM) offer a large memory capacity at low power consumption rates under the cost of short write endurance compared to DRAM. NVM memories can be integrated into a hybrid DRAM/NVM main memory or as a global NVM storage composed of servers globally accessible from all the nodes in the system over the interconnection network.

Besides the potential of these technological innovations, the question on how to orchestrate very complex parallel runs involving massively parallel and heterogeneous hardware remain open, although research proposals aimed at solving this problem are

appearing. Indeed, hybrid memory hierarchies composed of scratchpad and cache storage partially or totally managed by the runtime system have been proposed [3]. These designs aim at fully exploiting the potential of hybrid memories in terms of enhanced performance and reduced power consumption without increasing the programmability burden. In these proposals, the runtime system is in charge of mapping data specified by the programmer via task-based annotations to the scratchpad devices, so memory accesses to this data are served in a power-efficient way and without generating coherence traffic, while the rest of memory accesses are served by the L1 cache. That same approach can be taken to the next level and be used to handle multi-scale simulation workloads in machines with hybrid memory subsystems combining DRAM and NVM devices. Since PyCOMPSs assembles different MPI+X simulators, the runtime system handling parallelism expressed by the X programming model can operate in a similar way as in [3] to manage hybrid DRAM+NVM memory systems.

[1] Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M Badia, Jordi Torres, Toni Cortes, and Jesús Labarta, PyCOMPSs: Parallel computational workflows in Python, International Journal of High Performance Computing Applications, first published on August 19, 2015 as doi:10.1177/1094342015594678

[2] Marta Garcia, Jesús Labarta, Julita Corbalán, Hints to improve automatic load balancing with LeWI for hybrid applications. J. Parallel Distrib. Comput. 74(9): 2781-2794 (2014)

[3] L. Alvarez, M. Moreto, M. Casas, E. Castillo, X. Martorell, J. Labarta, E. Ayguade, and M. Valero. “Runtime-guided management of scratchpad memories in multicore architectures”. International Conference on Parallel Architecture and Compilation (PACT), pages 379–391, Oct 2015.

[4] Lordan F, Tejedor E, Ejarque J, Rafanell R, Álvarez J, Marozzo F, Lezzi D, Sirvent R, Talia D, Badia RM. **ServiceSs: an interoperable programming framework for the Cloud**. Journal of Grid Computing. March 2014, Volume 12, issue 1, pages 67-91.

[5] Jonathan Martí, Anna Queralt, Daniel Gasull, Toni Cortes: Towards DaaS 2.0: Enriching Data Models. SERVICES 2013: 349-355.