Proceedings of the First PhD Symposium on Sustainable Ultrascale Computing Systems (NESUS PhD 2016)
Timisoara, Romania

Jesus Carretero, Javier Garcia Blas
Dana Petcu
(Editors)

February 8-11, 2016

Dynamic Management of Resource Allocation for OmpSs Jobs

Sergio Iserte* Antonio J. Peña† Rafael Mayo* Enrique S. Quintana-Ortí*
Vicenç Beltran†

*Universitat Jaume I (UJI), Spain
†Barcelona Supercomputing Center (BSC-CNS), Spain

Abstract

The main purpose of this thesis is to research in the relation between task-based programming models and resource management systems in order to provide a smart autonomous load-balancing and fault-tolerant system. Thus, taking advantage of MPI malleable applications and execution models such as SMPD and MPMD we will dig in the principle of the dynamical reconfiguration. Apart from providing an overview of the thesis idea, this paper explains our initial motivation and reviews briefly the most remarkable work done in this field.

Keywords Exascale, heterogeneous systems, dynamic reconfiguration, OmpSs, resource management

I. Introduction and motivation

It is consensually believed that Exascale performance will only be achieved by adopting specialized hardware, what inevitably will turn systems into heterogeneous facilities. Dealing with heterogeneous hardware not only involves a tougher management of the cluster, but also a rise in the complexity of the applications which wanted to use all the resources available.

The vast majority of scientific applications have been developed using the Message Passing Interface (MPI) [7], in order to distribute the work among the nodes of a cluster. Two execution flows can be followed in this programming model:

- **Single Program Multiple Data (SPMD)** is the traditional and most extended approach. In this mode, all the processes will execute the same code working on different parts of the data.

- **Multiple Program Multiple Data (MPMD)**. This more recent mode does not restrict all processes to execute the same code. Usually, MPI applications are composed of several computational stages. If these stages can be executed independently and can be accelerated in specific hardware, we could refer to that as an offloading of the code in a device. This model fits better in heterogeneous environments.

The vast majority of MPI applications are moldable; they can be launched with different numbers of resources, which remain constant during all the application execution time. On the contrary, malleable applications can vary the amount of resources used in their execution, what means that applications are able to adapt themselves to changes in the environment.

Dynamic reconfiguration of MPI applications has been an important issue for many years. Its importance resides in the necessity of maximizing the utilization rate of the resources in an HPC cluster. Furthermore, it can reduce waiting times in queues by sizing jobs to the available resources or distributing sets of nodes among jobs. Hence, considerable effort made in the field of reconfiguration has been focused on the ability of malleability. This reconfiguration can be triggered by

the application itself or by the Resource Management System (RMS)—in the literature we can find defined this last set as *evolving applications*.

Nevertheless, dynamic reconfiguration is still a hot topic due to the blooming of new programming models which try to exploit heterogeneous HPC systems. One of the most extended modes is OmpSs [8] (developed by The Barcelona Supercomputing Center) which extends OpenMP with new directives to support asynchronous parallelism and heterogeneity. OmpSs enables asynchronous parallelism by using data dependencies among the tasks of the program. Offloading the MPI kernels dynamically using the OmpSs programming model could foster the adoption of the recenly emerged MPMD execution model [5] .

Moreover, the execution of these applications are generally handled by a RMS conscious of the status of all the hardware available in the facilities. If an application decided to change its allocated resources for different others, the RMS should be noticed in order to grant the operation at a given time.

II. Related Work

On the one hand, we find many contributions in the field of process malleability, having as a result excellent reconfiguration techniques or tools. For instance, authors in [3] explored the integration of malleability extensions in the process checkpointing and migration library (PCM) [4]. They take advantage of moldability to make the applications malleable by finishing and restarting them again. Also, there are contributions that make easier the adoption of malleability in applications with mechanisms of dynamic load-balancing [10], as well as reconfiguration techniques that are able to redistribute the workload and change the number of processes of a running application to obtain a certain performance [6].

On the other hand, projects that go further than just malleability techniques have been paving the road to exascale performance. One of the most remarkable is the DEEP Project [2]. DEEP is an innovative response to the exascale challenge, where a new organization is proposed: instead of providing the nodes with accelerators, the devices are put aside in an acceleration cluster, called "booster". In this scheme, both sides are interconnected by a high performance network. Applications offload their tasks to the "boosters" by using the OmpSs programming model.

[5] presents an extension of OmpSs to support dynamic offload of tasks among MPI processes. This provides flexibility, performance and scalability. However, the integration of that extension in a RMS is not addressed.

[1] presents a study of how to interact with an OmpSs job and the RMS that manages the facility. This work addresses the following limitations:

- The resources have to be requested on submission time, and the request is invariable. Hence, regardless of whether the application is using the "booster" or not, the resources are allocated.
- Queue and resource management. DEEP does not know the status of the nodes and its resources, making scheduling virtually impossible.

The work is concluded with a series of scripts to communicate the job and the RMS in order to perform the reconfiguration. However, an intelligent system with capacity of decision is left for future work.

III. Thesis Idea

The main objective of this thesis is to provide a user-friendly methodology to manage the resources assigned to a running job. Following partly the work in [1] (see Section II), our idea is still based on the fact that heterogeneous systems are paving the road to the exascale era, and that taking advantage of a programming model that supports asynchronous parallelism is crucial. Hence, combining the OmpSs multi-task (internally handled by threads) support with the capabilities of MPI to make the most of the distributed programming, the two most common programming models will be explored:

- SPMD: MPI + OmpSs (OpenMP). The user code should be adapted to provide a malleable MPI application (similar to application-based checkpoint/restart). Here, the application actively asks for a change of its assigned resources on response to a resource change request from the RMS.

- MPMD: MPI + OmpSs offload + OmpSs (OpenMP). In this scenario, the offloaded stage could be assigned with more or less resources depending on the decisions automatically taken by both the OmpSs runtime and the logic of the resource manager. However, having a malleable kernel like the one described in the previous point could boost the benefits.

Technically speaking, the OmpSs application should count with synchronization points where a reassignation of resources could be performed (whether a variation in the quantity or only a replacement). The synchronization points will be managed by a series of directives. Thus, the OmpSs runtime will be the responsible for moving data among tasks in different machines.

In addition, another interesting study case is that related to the *states*. Occasionally, servers save their own states as a guarantee of recovery in the case of a physical failure. This state could be loaded in another server and the execution of its jobs could be resumed. In this scenario the runtime of OmpSs should take additional care and provide more information about the states of its jobs in the different servers in order to let the RMS decide an appropriate strategy to reschedule the jobs and the resources.

In order to take reallocation decisions, four situations may happen:

1. An OmpSs job requests more resources: if the RMS has available resources, the job will be provided with them; otherwise, the request will be ignored or postponed.
2. An OmpSs job finishes a computational stage giving as a result a release of part of the allocated resources: the OmpSs runtime will notify the RMS about which resources are made available.
3. The RMS decides to assign more resources to an OmpSs job: at a given time, the RMS realizes that there are unused resources. Hence, if a job that previously had requested an expansion is still running, Slurm will assign more resources to it.
4. The RMS notices a stress situation (the queue is growing dramatically and the wait times have increased sharply) or the priority of other jobs is higher than that of the running job. If any running OmpSs job in the queue has been provided with the capability of reducing its allocation, the RMS could remove resources from the job. Of course, the OmpSs runtime will be aware of the location of the job data in order to redistribute it appropriately.

On the side of the RMS, we have decided to make use of Slurm [9]. Having an open source tool which provides a complete API and has proven that can reassign resources during the execution of a job [1] will increase the adoption of this project. Slurm is aware of the status of all the hardware under its control and ultimately the responsible for granting any reallocation operation.

To summarize, the main contributions that we expect from this work are:

- Integration of process malleability features in the OmpSs programming model, with the following actions:
  - We will propose extensions to the current application programming interface which will be considered for the OpenMP programming model.
  - We will develop the required functionality into the current OmpSs runtime and compiler.
  - We will define two APIs to face the triggered actions from both the RMS and the OmpSs application:
    - The first API will allocate/release resources.
    - While the second will check if there is a need for changing the resources currently assigned. In this case, once the RMS informs the application about a resource change, the application should use the first API to reallocate new resources.
- Novel dynamic reallocation scheduling policies with the enough intelligence to perform smart reallocation actions.

- Extensive performance evaluations in order to demonstrate the viability of using this new approach.

IV. Conclusion and Future Work

So far, the project is in an embryonic stage where we are still pursing an MPI malleable application. The application at issue will be used to measure the performance among versions.

Apart from the immediate appealing of having a process-malleable user-friendly environment, we strongly believe that this work can be directly applied on the resilience field, due to the capacity of adaptation to the environment that it presents. Exascale performance will involve a massive number of nodes working together. Such quantity of hardware increases the likelihood of experiencing a malfunction. Working at that scale a failure that entailed the re-execution of a job would represent a large waste of money and time. Having a system capable of reallocating efficiently resources in execution time, would transparently be highly beneficial.

Acknowledgment

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS); and the Project TIN2014-53495-R from MINECO and FEDER.

References

[1] Marco D'Amico. Extending deep offload programming model. Master's thesis, 2015.

[2] DEEP Project. http://www.deep-project.eu.

[3] Kaoutar El Maghraoui, Travis J. Desell, Boleslaw K. Szymanski, and Carlos A. Varela. Dynamic malleability in iterative MPI applications. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pages 591–598. IEEE, May 2007.

[4] Kaoutar El Maghraoui, Boleslaw K. Szymanski, and Carlos Varela. An architecture for reconfigurable iterative MPI applications in dynamic environments. In *Parallel Processing and Applied Mathematics*, pages 258–27. 2006.

[5] V. Beltran F. Sainz and J. Labarta. Collective offload for heterogeneous cluster. *2nd IEEE International Conference on High Performance Computing (HiPC)*, Dec 2015.

[6] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero. Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration. *Parallel Computing*, 46:60–77, Jul 2015.

[7] MPI Standard 3.1. http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[8] OmpSs. https://pm.bsc.es/ompss.

[9] SLURM Workload Manager. http://slurm.schedmd.com.

[10] Masha Sosonkina, Layne T. Watson, Nicholas R. Radcliffe, Rafael T. Haftka, and Michael W. Trosset. Adjusting process count on demand for petascale global optimization. *Parallel Computing*, 39(1):21–35, Jan 2013.