NESUS

cost IC1305

Network for Sustainable Ultrascale Computing

# 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

# Resilience of Parallel Applications

Nuria Losada, María J. Martín, Patricia González

Universidade da Coruña, Spain

{nuria.losada, mariam, pglez}@udc.es

**Abstract**

*Future exascale systems are predicted to be formed by millions of cores. This is a great opportunity for HPC applications, however, it is also a hazard for the completion of their execution. Even if one computation node presents a failure every one century, a machine with 100.000 nodes will encounter a failure every 9 hours. Thus, HPC applications need to make use of fault tolerance techniques to ensure they successfully finish their execution. This PhD thesis is focused on fault tolerance solutions for generic parallel applications, more specifically in checkpointing solutions. We have extended CPPC, an MPI application-level portable checkpointing tool developed in our research group, to work with OpenMP applications, and hybrid MPI-OpenMP applications. Currently, we are working on transparently obtaining resilient MPI applications, that is, applications that are able to recover themselves from failures without stopping their execution.*

***Keywords*** Fault Tolerance, Checkpointing, Resilience, MPI, OpenMP

## I. Motivation

Current petascale systems are formed by hundreds of thousands of cores. Schroeder and Gibson [16] have analysed failure data collected at two large high-performance computing sites, showing failure rates from 20 to more than 1,000 failures per year, depending mostly on system size. That can be translated in a failure every 8.7 hours. Future exascale systems will be formed by several millions of cores, and they will be hit by error/faults much more frequently than petascale systems due to their scale and complexity [5]. Therefore, long-running HPC applications in these systems will need to use fault tolerance techniques to ensure the successful execution completion.

The MPI (Message Passing Interface) standard is the most popular parallel programming model in petascale systems. Moreover, current HPC systems are clusters of multicore nodes that can benefit from the use of a hybrid programming model, in which MPI is used for the inter-node communications while a shared memory programming model, such as OpenMP, is used intra-node [20, 8]. However, these programming models lack fault tolerance support. In this scenario, checkpointing is a widely used fault tolerance technique, in which the computation state is saved periodically to disk into checkpoint files, allowing the recovery of the application when a failure occurs.

This PhD. thesis is focused on the study of efficient fault tolerance solutions for those parallel programming models that will likely be the most used in the exascale era. For this purpose, new strategies and protocols will be implemented in CPPC (ComPiler for Portable Checkpointing) [14], a portable and transparent checkpointing infrastructure for MPI parallel applications, to adequate it for the exascale era.

## II. CPPC Overview

CPPC is an open-source checkpointing tool for MPI applications available at `http://cppc.des.udc.es` under GNU general public license (GPL). CPPC is made up of a compiler tool and a runtime library, and its main characteristics are:

- It constitutes a transparent solution for the final user, since at compile time the CPPC source-to-source compiler automatically transforms a paral-

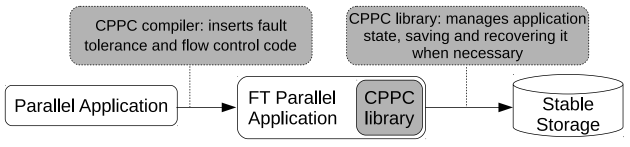
CPPC compiler: inserts fault tolerance and flow control code
CPPC library: manages application state, saving and recovering it when necessary
Parallel Application
FT Parallel Application
CPPC library
Stable Storage

*Figure 1: CPPC global flow*

lel code into an equivalent fault-tolerant version instrumented with calls to the CPPC library, as exemplified in Figure 1.

- It applies a spatially coordinaded checkpointing. The CPPC compiler identifies safe points, that is, code locations in which it is guaranteed that no inconsistencies due to messages may occur. The usage of safe points guarantees data consistency and no inter-process communications or runtime synchronization are necessary when checkpointing. Thus, reducing the checkpointing protocol overhead.
- It uses an application-level checkpointing, including in the checkpoint files only those application variables indispensable for the successful recovery. The CPPC compiler automatically performs a liveness analysis to identify the relevant variables, minimizing the checkpoint file size and, thus, reducing the checkpointing overhead.
- It results in a portable solution, thanks to the use of portable storage formats and the exclusion of architecture-dependent state from checkpoint files, allowing the recovery on machines with different architectures and/or operating systems than those in which the checkpoint files were generated.

## III. Thesis Work

In the literature, there exists some works focused on fault tolerance for shared memory systems, in which OpenMP is the de-facto standard for parallel programming on this systems. Some of these proposals are based on redundancy [7, 18], however, they can not tolerate multiple failures. On the other hand, the available checkpointing proposals for shared memory applications lack portability, whether code portability [13, 17] (allowing its use on different architectures) or checkpoint files portability [2, 4] (allowing to restart on different machines). In this context, we have extended CPPC to cope with OpenMP applications using a coordinated checkpointing protocol for data consistency [12], and applied different optimization techniques to minimize the overhead introduced during its operation [11]. Afterwards, we have extended that solution to cope with hybrid MPI-OpenMP applications using a hybrid protocol: coordinated checkpointing across OpenMP threads and uncoordinated across MPI processes (thanks to the use of safe points). We have evaluated the performance of this hybrid MPI-OpenMP solution on applications from the ASC Sequoia Benchmark Codes and the NERSC-8/Trinity benchmarks on over 6144 cores, obtaining overheads below 1.1% when checkpointing 50 GB of data. Additionally, the choice of an application-level approach and the portability of the checkpoint files allow building adaptable applications, that is, applications that are able to be restarted in a different resource architecture and/or number of cores, varying the number of OpenMP threads used by the application. This feature will be specially useful on heterogeneous clusters, allowing the adaptation of the application to the available resources.

Whether using the MPI or the hybrid MPI-OpenMP model, upon a single process/thread failure the entire application is aborted. This is the default behaviour because the state of MPI is undefined upon failure and, thus, there are no guarantees that the program can successfully continue its execution. Therefore, traditional fault tolerant solutions for these applications rely on stop&restart checkpointing: the application state is periodically saved into checkpoint files, so that, upon failure, a new job can be relaunched for restarting the application using the state files. However, a complete restart is unnecessary since, after a failure, most of the computation nodes used by a job will still be alive. Moreover, a complete restart introduces overheads both for re-queuing the job and for moving the checkpointed data across the cluster to the new granted resources. Thus, in the last years, new methods have emerged to provide fault tolerance support to MPI applications, such as failure avoidance approaches [6, 21] that preemptively migrate processes from processors that are about to fail. Unfortunately, these solutions are not able to cope with already happened failures.

Recently, the Fault Tolerance Working Group within the MPI forum proposed the ULFM (User Level Failure Mitigation) interface [3] to integrate resilience capabilities in the future MPI 4.0. It includes new semantics for process failure detection, and communicator revocation and reconfiguration. Thus, it enables the implementation of resilient MPI and hybrid MPI-OpenMP applications, that is, applications that are able to recover themselves from failures. Nevertheless, incorporating the ULFM capabilities in already existing codes is not a simple task. Different approaches for resilience using the new ULFM functionalities have emerged. Some of these solutions are Algorithm-Based Fault Tolerance (ABFT) techniques, which means that they are specific to one or a set of applications and they can not be generally applied [9, 1]. Other proposals, such as [15, 19] present a more general scope, however they rely on the developers to instrument their MPI applications in order to obtain fault tolerance support, which is, in general, a complex and time-consuming task.

In this scenario, we have exploit the ULFM new functionalities using CPPC to transparently obtain resilient MPI applications from generic MPI SPMD (Single Program Multiple Data) programs [10]. By means of the CPPC instrumentation of the original application code, failures in one or several MPI processes are tolerated using a non-shrinking backwards recovery based on checkpointing. In this solution, after a failure, the failed processes are re-spawned and all the processes rolled back to the last checkpoint available, so that the application can continue its execution with the same number of MPI processes.

## IV. Future Work

Our MPI resilience proposal combining CPPC and ULFM avois the overheads both for requeuing the job and for moving all the checkpointed data across the cluster. However, upon a failure, all the MPI processes roll back to a previous saved state to recover the application. In this situation, not only some computation done by the failed processes is lost, but also some computation performed by the survivor processes, as all of them roll back to the last checkpoint available and continue the execution from that point. Therefore, to adequate this proposal to the exascale era, we plan on designing and implementing a local recovery strategy, so that, only the failed processes have to roll back to a previous state, while the survivors can continue their computation. Apart from improving the scalability of the proposal, this strategy can reduced the energy consumption, as survivor processes do not repeat any part of their computation.

## Acknowledgment

This research was supported by the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU (Project TIN2013-42148-P, and the predoctoral grant of Nuria Losada ref. BES-2014-068066) and by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

## References

[1] M. M. Ali, J. Southern, P. Strazdins, and B. Harding. Application Level Fault Recovery: Using Fault-Tolerant Open MPI in a PDE Solver. In *IEEE International Parallel Distributed Processing Symposium Workshops*, pages 1169–1178, 2014.

[2] J. Ansel, K. Arya, and G. Cooperman. DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. In *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2009.

[3] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. An evaluation of User-Level Failure Mitigation support in MPI. *Computing*, 95(12):1171–1184, 2013.

[4] G. Bronevetsky, K. Pingali, and P. Stodghill. Experimental evaluation of application-level checkpointing for OpenMP programs. In *Proceedings of the 20th Annual International Conference on Supercomputing*, pages 2–13, 2006.

[5] F. Cappello. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and re-

search opportunities. *International Journal of High Performance Computing Applications*, 23(3):212–226, 2009.

[6] I. Cores, G. Rodríguez, P. González, and M. J. Martín. Failure avoidance in MPI applications using an application-level approach. *The Computer Journal*, 57(1):100–114, 2014.

[7] H. Fu and Y. Ding. Using Redundant Threads for Fault Tolerance of OpenMP Programs. In *Proceedings of the 2010 International Conference on Information Science and Applications*, pages 1–8, 2010.

[8] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman. High Performance Computing using MPI and OpenMP on Multi-core Parallel Systems. *Parallel Computing*, 37(9):562 – 575, 2011.

[9] I. Laguna, D.F. Richards, T. Gamblin, M. Schulz, and B.R. de Supinski. Evaluating User-Level Fault Tolerance for MPI Applications. In *European MPI Users' Group Meeting*, pages 57–62, 2014.

[10] N. Losada, I. Cores, M. J. Martín, and P. González. Resilient MPI applications using an application-level checkpointing framework and ULFM. In *Journal of Supercomputing. [In Press]*, 2016.

[11] N. Losada, M. J. Martín, G. Rodríguez, and P. González. I/O Optimization in the Checkpointing of OpenMP Parallel Applications. In *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 222–229, 2015.

[12] N. Losada, M.J. Martín, G. Rodríguez, and P. González. Extending an Application-Level Checkpointing Tool to Provide Fault Tolerance Support to OpenMP Applications. *Journal of Universal Computer Science*, 20(9):1352–1372, 2014.

[13] M. Prvulovic, Z. Zhang, and J. Torrellas. ReVive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *Proceedings of the 29th Annual International Symposium of Computer Architecture*, pages 111–122, 2002.

[14] G. Rodríguez, M.J. Martín, P. González, J. Touriño, and R. Doallo. CPPC: a compiler-assisted tool for portable checkpointing of message-passing applications. *Concurrency and Computation: Practice and Experience*, 22(6):749–766, 2010.

[15] K. Sato, A. Moody, K. Mohror, T. Gamblin, B.R. De Supinski, N. Maruyama, and S. Matsuoka. FMI: Fault Tolerant Messaging Interface for Fast and Transparent Recovery. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1225–1234, 2014.

[16] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, 2010.

[17] D.J. Sorin, M.M.K. Martin, M.D. Hill, and D.A. Wood. SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 123–134, 2002.

[18] O. Tahan and M. Shawky. Using dynamic task level redundancy for OpenMP fault tolerance. In *Proceedings of the 25th International Conference on Architecture of Computing Systems*, pages 25–36, 2012.

[19] K. Teranishi and M.A. Heroux. Toward Local Failure Local Recovery Resilience Model Using MPI-ULFM. In *European MPI Users' Group Meeting*, pages 51–56, 2014.

[20] R. Thakur, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, and J. L. Träff. MPI at Exascale. *Proceedings of Scientific Discovery through Advanced Computing*, 2, 2010.

[21] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive process-level live migration in HPC environments. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 43, 2008.