Software-level Fault Tolerant Framework for Task-based Applications

Joy Yeh, Grzegorz Pawelczak, James Sewart,
James Price, Amaurys Avila Ibarra, Simon McIntosh-Smith*
Department of Computer Science
University of Bristol
Bristol, UK
Email: *cssnmis@bristol.ac.uk

Ferad Zyulkyarov,
Leonardo Bautista-Gomez,
Osman Unsal
Department of Computer Science
Barcelona Supercomputing Center
Barcelona, SPAIN

***Abstract*—Fault tolerance has been identified as one of the major challenges for exascale computing. In addition to fail-stop errors, silent data corruptions (SDCs) can perturb applications and produce incorrect results. Software-based fault tolerance mechanisms have the advantage of being capable of leveraging some of the properties of the applications to improve their reliability. In this poster, we present a fault tolerance framework that implements multiple resiliency schemes to cope with both fail-stop errors and data corruption. Our techniques are tested with two real scientific applications: BUDE, a molecular docking engine, and TeaLeaf, a heat conduction code. Using this framework we have successfully detected and recovered from real data corruptions. We have also performed error injection experiments, which clearly demonstrated the efficacy of our framework.**

I. Introduction

Reliability is one of the main challenges to achieve exascale computing, fail-stop errors and silent data corruption (SDC) can cause important damage to scientific simulations. As the operating voltage of main memory decreases in tandem with rapid growth in the data sizes of simulations, the probability of data corruption increases. Hardware mechanisms to detect data corruption are efficient but not flawless. Error correcting codes (ECC) offer single error correction and double error detection (SECDED) in most high-end systems. When an uncorrectable error occurs, generally the application is terminated by the system. This is largely inefficient for task-based applications, that could simply re-execute the task related to the affected memory address.

In addition, SDC can be tackled at the software-level by leveraging several application characteristics. When data-integrity mechanisms are in place, they can be coupled with the task-level resiliency schemes to re-execute the corrupted task. In this research we explore several resilience techniques and we couple them together to offer a reliable task-based framework to execute scientific applications.

II. Methodology

We propose a fault-tolerant, task-based runtime that can re-execute failed tasks in a transparent fashion for the user. We implement this inside Nanos [1], the runtime of the OmpSs [2] framework. The fault tolerant Nanos runtime is capable of handling signal events triggered by the system when uncorrectable errors occur [3], [4]. In such a scenario the system would trigger a signal that will force the protection of the faulty page. Then, when the related task tries to access the protected page it will fail to load the data and the task will fail. Nanos will then re-execute the task using another memory region.

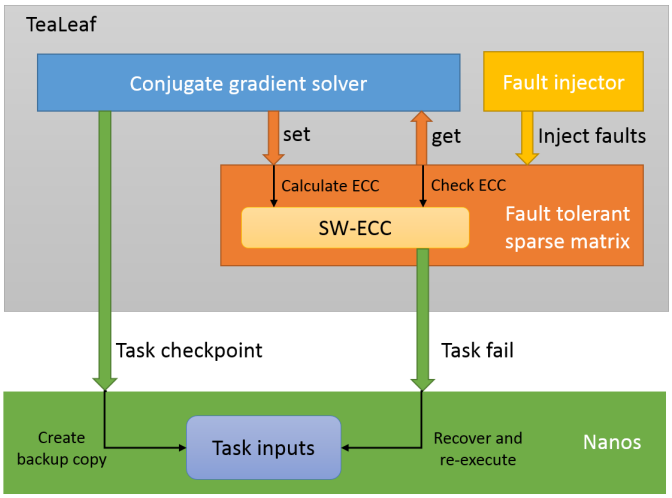

Fig. 1. Fault Tolerant TeaLeaf

To further leverage the resilient scheme of Nanos, we couple this resilient task-based runtime with several algorithm-based fault tolerance (ABFT) strategies that we implemented in two different applications.

A. TeaLeaf

TeaLeaf [5] is a heat conduction mini-app implemented in C. The code can model 2D and 3D domains that are partitioned and distributed in multiple MPI processes. The processes share ghost cells to interact with neighbours. The application makes use of a classic conjugate gradient (CG) solver, which has been improved with software-level ECC and CRC to detect SDCs [6]. In addition, we parallelize local computation using OmpSs and the fault tolerant Nanos runtime. Therefore, when the ABFT-protected CG solver detects data corruption, it can either try to correct it or simply fail and let Nanos re-execute the corrupted task, with a much lower overhead than traditional checkpoint/restart. Figure 1 shows the fault tolerance runtime and the interactions with the TeaLeaf application.

B. BUDE

To further test our framework, we experimented with a second application with different fault tolerance properties.

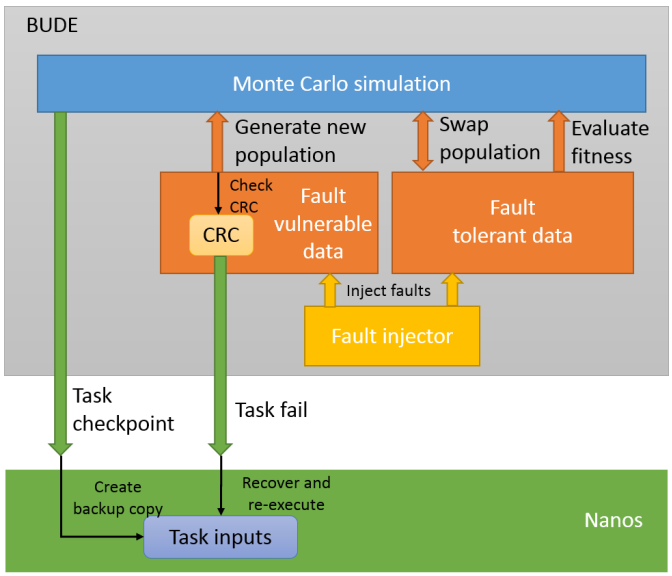

Fig. 2. Fault Tolerant BUDE

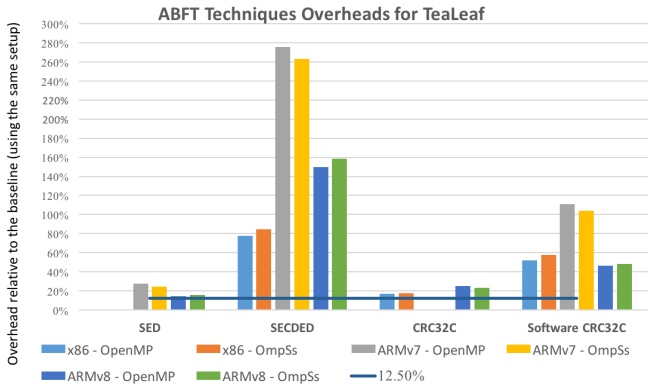

Fig. 3. Single thread overheads of SDC detection schemes for sparse matrices in TeaLeaf.

BUDE [7] is a molecular docking application written in C/C++ that performs molecule docking, ligand binding site identification on protein surfaces, and protein to protein docking. The application exhibits some natural robustness to SDCs because it follows a Monte Carlo simulation, whereby its evolving population is randomly generated or mutated from previous best parents. The rest of the program memory is considered critical by default and is protected using 32-bit CRC codes, NaN/Inf floating-point filters, and by double checking the final solution. When errors are detected, the Nanos runtime re-executes the failed task with a lower overhead than traditional, global checkpoint/restart. Figure 2 shows the different components of the fault tolerant framework and their interactions with BUDE.

III. Results and Conclusions

In BUDE, the overhead of the error checking schemes was measured to be negligible ($<<1\%$). Protected memories include 16KB of program code segment, 19KB of critical data, and 2KB of read-only pages. In comparison to no checkpointing, the Nanos task checkpointing scheme saw overheads of 7% for ARMv7 on the Mont Blanc prototype, $<1\%$ for ARMv8 on the Merlin and Thunder cluster, with variability averaging about 20% amongst x86 machines. The checkpointing overhead also did not increase with population size since we do not checkpoint the evolving population, which is naturally fault tolerant. OmpSs tasks were observed to recover successfully and consistently when BUDE was subjected to memory error injections. Test population sizes were at a minimum of 50 thousand poses, increasing up to 6 million poses.

For TeaLeaf, we ran a CG solve on a grid measuring 2048x2048 and with 5 time steps and compared it against the baseline measurements where no software ECC/CRC has been used.

The TeaLeaf results in Figure 3 clearly show that the SED scheme can be efficiently implemented in software as the overheads are relatively small, while the SECDED approach is currently not a viable option due to the large overheads of around 72% on x86 and 150% on ARMv8. We have also implemented CRC, where a single 32-bit value can protect the whole row of data. In this approach, the first four elements from the matrix row each store 8-bits of the 32-bit checksum. The CRC is then checked by either using CRC via intrinsics (x86 and ARMv8) or by calculating it in software (ARMv7). The hardware supported implementation of CRC showed very promising results, especially considering it can detect multibit SDC, which otherwise might have not been detected by ECC.

The ARMv7 architecture used in the Mont Blanc ARM-based prototype machine does not provide hardware protection for memory, and so our ABFT-protection schemes provide an efficient alternative to detect and correct any SDCs. We can inject bitflips into the critical data and successfully detect and correct these, with the recovery adding negligible overhead. Using TeaLeaf, which stores much more critical data than BUDE, we have been able to run it at scale and detect **real** SDC and successfully recovered from them using Task Recovery.

References

[1] X. Teruel, X. Martorell, A. Duran, R. Ferrer, and E. Ayguadé, "Support for OpenMP tasks in Nanos v4," in *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*. IBM Corp., 2007, pp. 256–259.

[2] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "OmpSs: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[3] O. Subasi, J. Arias, O. Unsal, J. Labarta, and A. Cristal, "Nanocheckpoints: A task-based asynchronous dataflow framework for efficient and scalable checkpoint/restart," in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2015, pp. 99–102.

[4] ——, "Programmer-directed partial redundancy for resilient HPC," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*. ACM, 2015, p. 47.

[5] S. McIntosh-Smith, M. Martineau, M. Boulton, W. Gaudin, P. Garrett, W. Liu, and R. Smedley-Stevenson, "The TeaLeaf heat diffusion benchmark from Mantevo," https://github.com/UoB-HPC/TeaLeaf, 2016 (accessed Oct 7, 2016).

[6] S. McIntosh-Smith, R. Hunt, J. Price, and A. Vesztrocy, "Application-Based Fault Tolerance Techniques for Sparse Matrix Solvers," *International Journal of High Performance Computing Applications*, 2016.

[7] S. McIntosh-Smith, J. Price, R. Sessions, and A. Avila Ibarra, "High performance in silico virtual drug screening on many-core processors," *International Journal of High Performance Computing Applications*, vol. 29, no. 2, pp. 119–134, 5 2015.