Leveraging a Task-based Asynchronous Dataflow Substrate for Efficient and Scalable Resiliency

Omer Subasi Javier Arias Jesus Labarta Osman Unsal Adrian Cristal
Barcelona Supercomputing Center

1. INTRODUCTION

Reliability has been identified as a major challenge for high performance computing (HPC) and multi-core systems. It is widely believed that in the near future hardware-based fault mitigation schemes will not suffice by themselves to keep error rates below design targets for such systems. Consequently this has increased the importance of software focused resilience solutions at the runtime, application and algorithm levels. In particular, the runtime has been identified as a key component for resilience by the International Exascale Software Project Roadmap [1]. Moreover, the Roadmap recommends that the runtime be responsible for fine-grain resilience solutions such as preserving correct state, and reissuing failed tasks. However, to be feasible these solutions must be efficient, scalable, flexible and easy to implement.

In our work, we aim to demonstrate that a dataflow runtime coupled with a task-based programming model provides an ideal substrate to develop such solutions. We leverage OmpSs, a task-based OpenMP derivative programming model and the Nanos asynchronous data-flow runtime to develop two fault-tolerant designs for compute node resilience. In the first design, we use the task directionality information to develop an efficient incremental checkpoint/restart scheme that only checkpoints minimal data to recover from errors that are detected but not corrected by hardware. Moreover, due to the dataflow semantics of the runtime, both checkpointing and error recovery are asynchronous. Our results indicate that the design is scalable even for high error rates. In one variant of this design, we develop an efficient memory mapping scheme that decreases the checkpointing memory overheads significantly. In the second design, we explore task replication to detect and recover from Silent Data Corruption. Our baseline design seamlessly and transparently duplicates tasks for resilience. In our baseline design, the runtime duplicates every task for very high error coverage. Although asynchronous execution helps to limit overheads, the cost could be high. We therefore develop a simple, yet effective runtime heuristic that selectively replicates the most reliability critical tasks for a reasonable tradeoff between performance and error coverage. Finally, we provide support in runtime for the programmer to express his insight in terms of which tasks should be protected by runtime.

2. BACKGROUND AND MOTIVATION

In this section we provide a short background on our substrate and its advantages in terms of reliability. OmpSs is task-based and it extends OpenMP with new data directionality clauses. These clauses specify scopes of execution for driving the construction of an asynchronous data-flow execution graph on its runtime, Nanos. The programmer only specifies the task input and outputs. These data directionalities then drive the construction of data dependencies among the tasks at runtime. Reader is advised to consult [2] and Nanos [3] for further information.

OmpSs model and its runtime Nanos offer several advantages for resiliency. As application-level checkpoints are data-aware, only the data that is utilized in the task need to be saved. Application-level checkpointing using Nanos runtime can exploit data locality and data directionality providing efficient and scalable memory utilization. Our fault detection and recovery mechanisms rely on these precise checkpoints, thus minimizing overheads. In addition, Nanos tasks are executed asynchronously and in parallel which makes it inherently easy to exploit these properties for fault tolerance features. Crucially, failure recovery is very efficient: Nanos tasks that do not have dependencies to a task that has failed can continue to execute, in parallel and asynchronously with the recovery of the failed task. This makes fault-tolerance hardened dataflow runtimes such as the Nanos design presented here very efficient for HPC and multicore systems which are predicted to suffer from high fault rates. In contrast, in coordinated checkpointing implementations the fault recovery process acts as a barrier both in the physical sense (even non-faulty tasks block waiting for the resolution of the fault recovery) and in the scalability sense. Moreover, for dataflow runtimes such as Nanos, task dependencies are known at runtime which enables heuristics to determine reliability-critical task for partial redundancy. In terms of error propagation, since the only state that propagates out of the task is through the outputs, it is straightforward to limit error propagation, and to determine the source of an error. In the next section, we present our mechanisms together with results of our experiments.

3. OUR MECHANISMS AND EXPERIMENTAL RESULTS

We implemented our mechanisms on the Nanos runtime. In the first design, we implemented a checkpoint/restart mechanism as a recovery mechanism for hardware detected but uncorrected errors. Then, we implement an optimized version of it in which we share data information among threads to maintain a single copy of task data inputs, which we call smart backup mechanism. In second design, we use task duplication for fault detection to target silent data corruption. Finally, we have partial redundant schemes for the two designs. In the automated version, we define the risk of a task as a function of its input size, number of inputs and successors and compare it to the global risk value of all tasks. If it is bigger than the global value, then we checkpoint and duplicate the task. Otherwise we do not protect the task. Finally we update the global value by incorporating the task's risk value. The risk definition and the update of the global value is experimentally determined. The second design of the partial framework enables to the programmer to express his insights in terms of the which tasks should be protected. The programmer specifies them and runtime checkpoints those tasks and duplicates them.

We run our experiments in MareNostrum supercomputer hosted at Barcelona Supercomputing Center. We implemented fault inject functionalities in our runtime that can inject faults to tasks' outputs with a fixed fault probability per task or with rates that are proportional to the average execution time of the benchmark as well as the size of application/benchmark arguments. We use sparse LU factorization, Cholesky decomposition and Fast Fourier Transform (FFT) as our benchmarks.

Next we provide some results for our mechanisms. Figure 1 demonstrates that our smart checkpoint/restart mechanism is strongly scalable even with unrealistically high per task fault probabilities (rates).

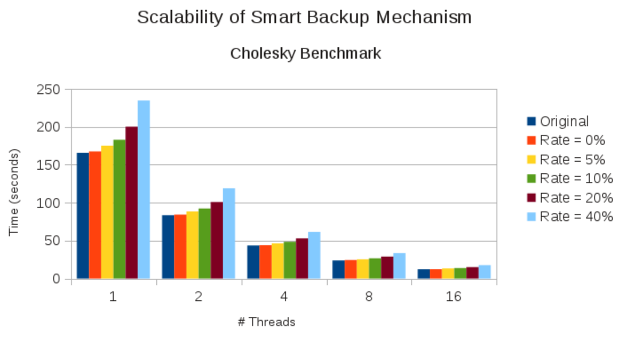

Figure 1: Scalability of Smart Backup Mechanism: Cholesky

Moreover, we find that the overhead of checkpointing is significantly low with respect to fault-free executions when our smart checkpoint/restart mechanism is utilized. For sparse LU, the overhead is 0.2%, for Cholesky it is 1% and for FFT it is 7%.

Figure 2 shows how our runtime heuristic and user insights enabled by our implementation provides better tradeoff between amount of protection and that of corruption in the final program output for Cholesky decomposition. The single red point shows the protection percentage and the corruption in the program output when our heuristic is used. The single green point shows the case for programmer (user) specifies his insight by our support in runtime. The blue line is experimentally obtained where we randomly select tasks to checkpoint and replicate.

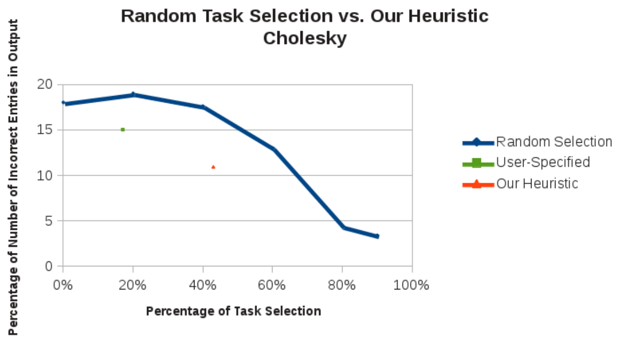

Figure 2: Comparison to Random Task Selection: Cholesky Benchmark

We do not include all results for brevity but they show that our checkpoint/restart and task replication mechanisms are scalable and incur low overhead for all benchmarks and our two partial frameworks provide good tradeoff between reliability and cost of redundancy.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007 - 2013] under the Mont-blanc 2 Project (www.montblanc-project.eu), grant agreement no. 610402.

4. REFERENCES

[1] Jack Dongarra, Pete Beckman, and Terry Moore et al. The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, 2011.

[2] Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. Ompss: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(2):173–193, 2011.

[3] Xavier Teruel, Xavier Martorell, Alejandro Duran, Roger Ferrer, and Eduard Ayguadé. Support for openmp tasks in nanos v4. In *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, CASCON '07, pages 256–259, Riverton, NJ, USA, 2007. IBM Corp.