Asynchronous Checkpointing by Dedicated Checkpoint Threads

Faisal Shahzad[1], Markus Wittmann[1], Thomas Zeiser[1], and Gerhard Wellein[2]

[1] Erlangen Regional Computing Center, University of Erlangen-Nuremberg, Germany
[2] Department of Computer Science, University of Erlangen-Nuremberg, Germany
{faisal.shahzad,markus.wittmann,thomas.zeiser,gerhard.wellein}@rrze.fau.de

Abstract. Checkpoint/restart (C/R) is a classical approach to introduce fault tolerance in large HPC applications. Although it is relatively easy as compared to other fault tolerance approaches, its overhead hinders its wide usage. We present an application-level checkpointing technique that significantly reduces the checkpoint overhead. The checkpoint I/O is overlapped with the computation of the application by following a two-stage checkpointing mechanism with dedicated threads for doing I/O.

1 Algorithm and Implementation

With each step closer towards the exascale barrier, the mean time between failure (MTBF) of these futuristic systems reduces. This raises the importance of checkpoint/restart techniques [1]. As IO bandwidths cannot be increased arbitrarily, it is important to investigate approaches which can hide IO time of checkpointing. One of these approaches is to utilize non-blocking asynchronous MPI-IO for creating checkpoints. However, neither asynchronous non-blocking point-to-point communication [2] nor asynchronous non-blocking MPI-IO is supported by most of the MPI implementations. Therefore, we implement asynchronous checkpointing manually by creating a two-stage checkpointing mechanism and a dedicated checkpoint thread (CP-thread) as shown in Fig. 1. Each MPI process is divided into two threads, a worker thread and a CP-thread. The CP-thread of each MPI process is pinned to a simultaneous multi-threaded (SMT) core for the present Intel processor architectures, while the worker threads are pinned to the physical cores. If SMT is not available, physical cores may be oversubscribed. When a checkpoint is triggered, an in-memory checkpoint is made first by the worker thread. The second stage of checkpointing involves the copying of the in-memory checkpoint to the external file system and is carried out by the CP-thread. For benchmarking, we have utilized an MPI application based on a stencil type algorithm with toggle grids. Thus, it is obvious to introduce an additional checkpointing grid (CP-grid) which is responsible for temporarily storing the in-memory checkpoint. By switching the grid pointers, the extra in-memory copy of the CP-grid from the most updated grid is completely avoided.

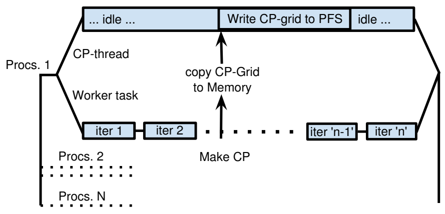

Fig. 1. Diagram of the program flow. Each MPI process is divided into worker and checkpoint threads. At checkpoint iteration, the worker thread signals the checkpoint thread to write the checkpoint.

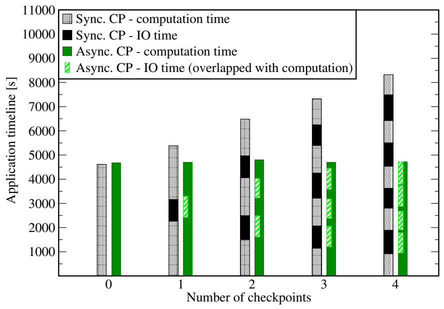

Fig. 2. Checkpoint overhead for 128 LiMa nodes (1536 MPI-processes, 1 CP-th./SMT-core) with aggregated checkpoint size of 800 GB/checkpoint

2 Results

Benchmarks were performed on an Intel based Infiniband cluster (LiMa)[1], which is connected to a Lustre parallel file system. In a naïve synchronous checkpointing technique, each MPI process interrupts its computation for the duration of writing checkpoints, i.e., the complete IO time is added as overhead to the total runtime. Figure 2 shows the checkpoint overhead comparison between a naïve synchronous checkpointing and our presented asynchronous checkpointing technique for our application on 128 LiMa nodes with an aggregated checkpoint size of 800 GB. Each synchronous checkpoint adds ≈22% overhead to the application, whereas each asynchronous checkpoint costs ≈0.6% overhead, i.e., almost all the IO time is effectively hidden. This significantly reduces the checkpoint overhead. The maximum number of low overhead asynchronous checkpoints can be calculated as: $\frac{\text{application runtime without checkpoints}}{\text{IO time for a single checkpoint}}$.

Acknowledgements. This work was supported by BMBF grant No. 01IH11011C (FETOL).

References

1. Hursey, J.: Coordinated Checkpoint/Restart Process Fault Tolerance for MPI Applications on HPC Systems. PhD thesis, Indiana University, Bloomington, IN, USA (July 2010)
2. Hager, G., Schubert, G., Schoenemeyer, T., Wellein, G.: Prospects for Truly Asynchronous Communication with Pure MPI and Hybrid MPI/OpenMP on Current Supercomputing Platforms. In: Cray Users Group Conference 2011, Fairbanks, AK, USA (2011)

[1] `http://www.hpc.rrze.fau.de/systeme/lima-cluster.shtml`