Rohan Jhaveri

A Checkpoint/Restart Scheme for CUDA Programs with Complex Computation States

Hai Jiang 1 , Yulu Zhang 1 , Jeff Jenness 1 , Kuan-Ching Li 2
1 Department of Computer Science,
Arkansas State University, USA
E-mail: {hjiang, yulu.zhang, jeffj}@cs.astate.edu
2 Department of Computer Science and Information Engr.,
Providence University, Taiwan
E-mail: kuancli@pu.edu.tw

Fault Resilience technique in High Performance Computers (HPC) in one of the major research field. Since past decade Checkpoint/restart has been an effective mechanism to achieve fault tolerance for many long running scientific applications in HPC. The most general implementation of the checkpointing technique operates by stopping the system's execution at regular intervals to save the state of all executing applications to a permanent storage device, typically a parallel file system. But recent HPC are built using GPU, and there is no sophisticated checkpoint/restart scheme yet due to the parallel state handling mechanism of the GPU. This paper proposes an application-level checkpoint/restart scheme to save and restore GPU computation states. To implement checkpoint/restart on a GPU a pre-compiler and run-time support module are developed to dynamically construct and save states in CPU system memory. Whereas, for scalability and long-term fault tolerance secondary storage can be utilized.

To implement checkpoint/restart on a GPU application, the computation state is collected/constructed at a checkpointing event and restored at a later restarting event. These states are represented by variables declared in the CUDA program. But due to the complex memory hierarchy of the GPU, those variables are spread in different memory locations like register, local memory, shared memory and global memory. Also, due to limited space in GPU global memory, the GPU application state are saved in the host system memory first. Variables in registers and local memory are copied into one buffer and variables in shared memory are grouped into another buffer, and these buffers are called as page-locked host memory. Variables in global memory are saved back in a user declared buffer. All three buffers can be moved to other machines for computation migration or copied on hard disks for long-term fault tolerance purposes. The checkpoint/restart of a GPU computation is accomplished through a pre-compiler at compile time with a run-time support module. The pre-compiler transforms the application source code into a format where the run-time support calls are inserted for constructing the computation state precisely. The task of pre-compiler is to insert checkpoint which is done during device side code transformation. This is followed by thread synchronization, state variable collection, checkpoint counter management, library call management and file storage management. The functions of run-time support module are to register checkpoint states, save the states on stack or heap and restore the states from the buffers during restart. Some of the tasks are carried on host side while some of them are carried on device side and the kernel is responsible for host side and device side code transformation.

Although, this method is very efficient technique for fault resilience in GPU, but it can only carry out application level checkpoint. It fails to provide information on thread level checkpoint and kernel level checkpoint. Also, based on current HPC trends, this resilience technique would consume lot of energy. There are many new HPC resilience techniques like multilevel checkpoint, partial redundancy or parallel recovery would give better performance in terms efficiency and energy.