**Motivation**

- Lots of large-scale HPC systems are now adapting GPUs to achieve better performance
- Multi-GPU applications such as MPI-CUDA ones are being developed to exploit GPUs' highly parallel computationalism
- Fault tolerance obviously needs to be supported
- ECC alone cannot tolerate hard failures
- Checkpoint/restart is necessary

Current GPU's fault-tolerant tools do not support checkpointing multi-GPU applications

**Methodology**

**Save states in MPI CUDA applications**

- States of processes on GPUs
- States of interaction amongst processes
- States of GPUs
- States of communication between CPUs and GPUs

**MPI CUDA applications checkpointing scheme**

- Checkpoint signal arrival
- Wait until communication of GPU-GPU and kernel execution finish
- CPU process
- GPU
- Running kernel
- Evacuate states on GPU into CPU
- Open MPI+BLCR checkpoint
- Recover states on GPU

**Implementation**

**CUDA checkpointer**

Contracted followed by 3 steps:

- **Pre-processing**
  - Waiting until GPU's kernel execution and communication between CPU-GPU
  - Copy all the user data in the device memory to the host memory and destroy CUDA context

- **BLCR**
  - Have BLCR do checkpointing the CPU state

- **Post-processing**
  - Copy copied data on CPU back to GPU and restore CUDA context

**Manage data objects on GPU**

- Record GPU memory chunk sizes and addresses
- Use a custom memory allocator to allocate GPU memory regions in appropriate positions
- Since GPU memory regions allocated by cudaMalloc may change at restarting, we manage to keep them unchanged during checkpoint/restart

**Manage code objects on GPU**

- Keep track of data registered in the kernel generated by cudaBinRegisterFatBinary, cudaRegisterFunction
- These data need to be registered again at restarting

**CUDA checkpointer microbenchmark**

Target: a simple CUDA program which allocates raw data with size varying from 100 MB to 1000 MB

Checkpoint test is performed on one machine in Racoon

Overhead of pre-processing & post-processing is very small

**Evaluation**

**MPI CUDA checkpointer evaluation**

- Target: 3D Stencil MPI CUDA application
- Conduct a weak-scaling experiment with
  - # of procs: 10-60
  - Z-axis value: 100-600
  - X,Y values are fixed at 256
- Unknown = (App runtime with ckpt) – (App runtime w/o ckpt)
- (pre-processing + BLCR + post-processing)

Some costs depending on data size and # of procs are probably included in the unknown overhead

```
100/10proc 200/20proc 300/30proc 400/40proc 500/50proc 600/60proc
Z size / # of procs

The unknown overhead occupies 60%-89% of the overall checkpoint overheads
```

**Future Work**

- Analyze details of the unknown overhead
- Use diskless checkpointing to guarantee scalability
- Improve CUDA checkpointer to support more CUDA APIs