MPI-CUDA Applications Checkpointing

Nguyen Toan[†] Tatsuo Nomura[†] Hideyuki Jitsumoto[††]
Naoya Maruyama[†] Toshio Endo[†] Satoshi Matsuoka[†] [†††] [††††]
{nguyen, tatsuo, naoya, endo, matsu}@matsulab.is.titech.ac.jp,
jitumoto@cc.u-tokyo.ac.jp

[†] Tokyo Institute of Technology
[††] The University of Tokyo
[†††] National Institute of Informatics
[††††] Japan Science and Technology Agency

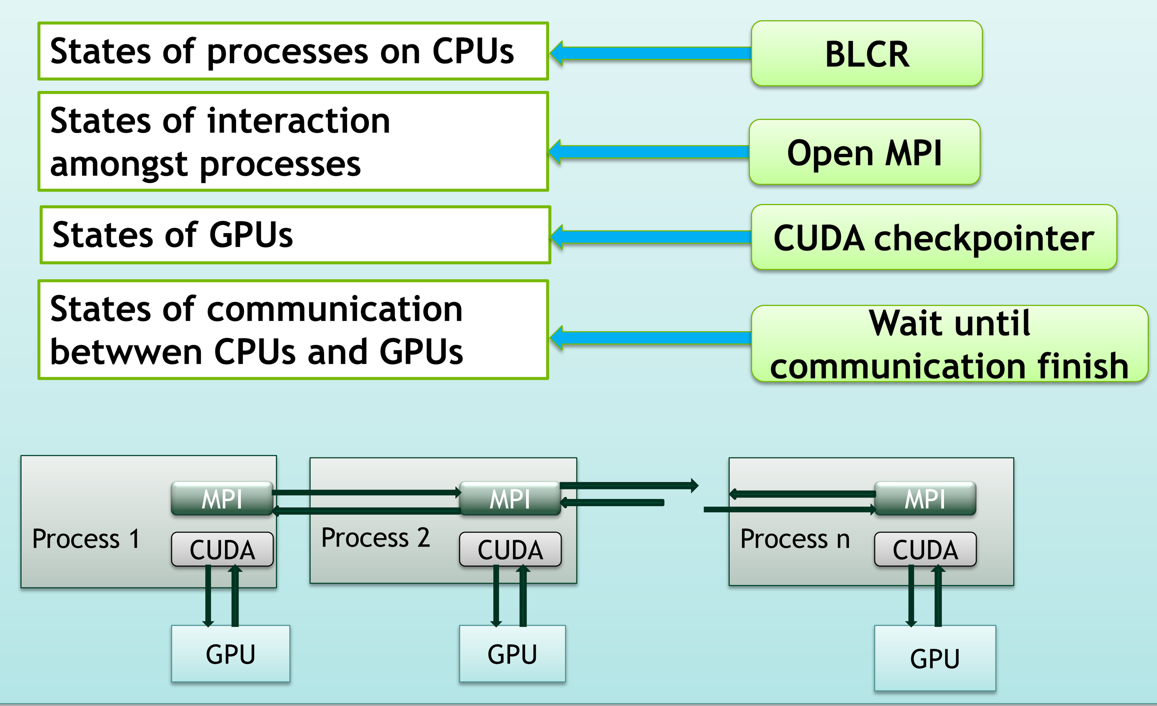
Motivation

- 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 computationality
- *Fault tolerance* obviously needs to be supported
  - + ECC alone cannot tolerate hard failures
  - + **Checkpoint/restart** is necessary

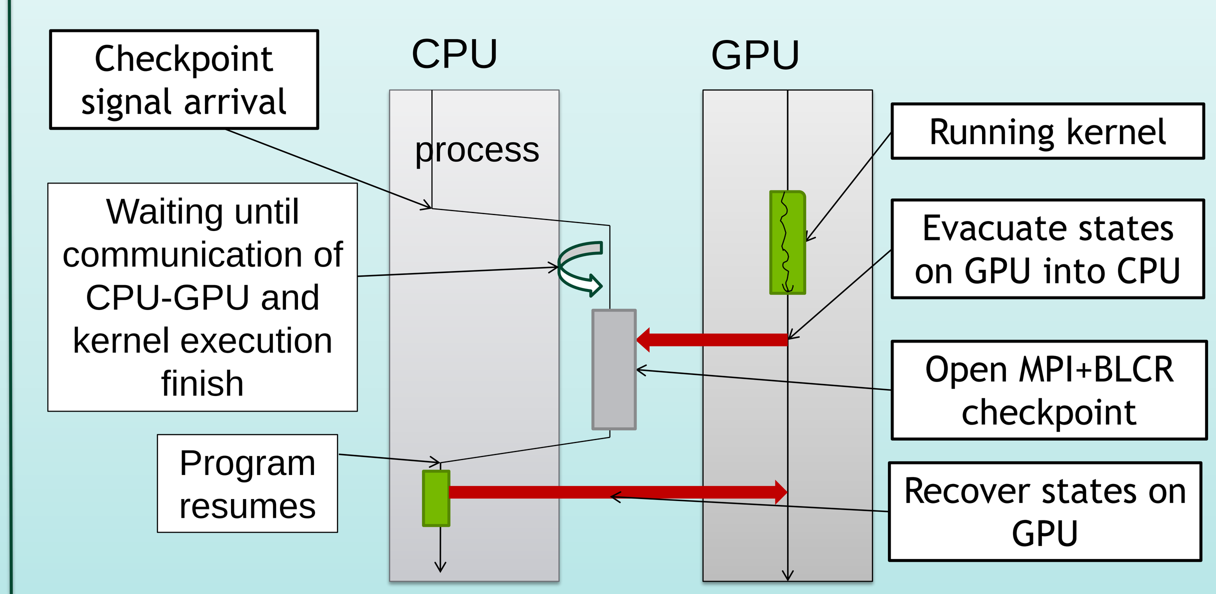
Current GPU's fault-tolerant tools do not support checkpointing multi-GPU applications

Methodology

Save states in MPI CUDA applications



MPI CUDA applications checkpointing scheme



Implementation

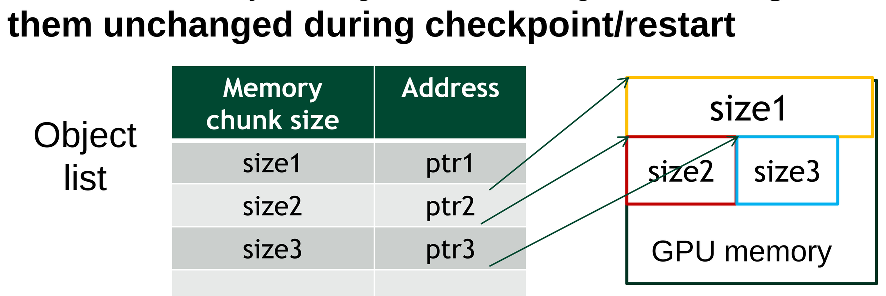
CUDA checkpointer

Contructed 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 addresses 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 + Open MPI + BLCR intergration

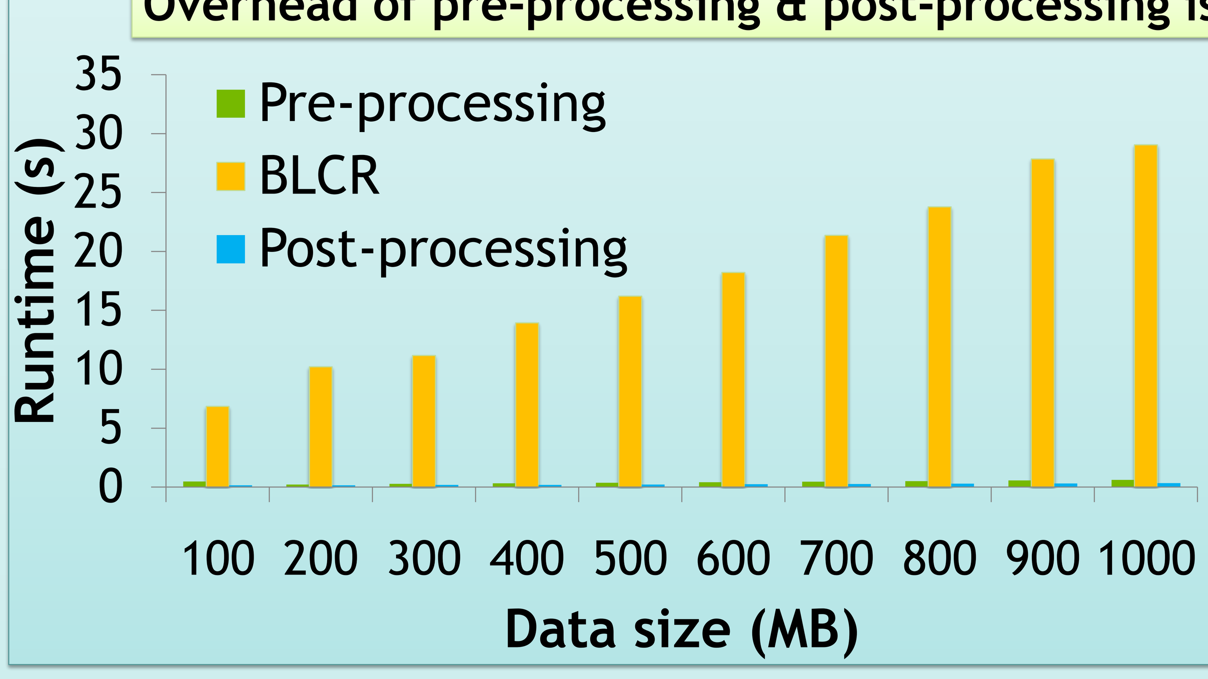
- Attach CUDA checkpointer to BLCR
  - + Make use of user callback function in BLCR
- Prevent signal interruption during CUDA API execution by applying signal masking
  - + Before each CUDA API execution, the signal handler is modified to perceive the arrival of the checkpoint signal
  - + After each CUDA API execution, the signal handler is returned back to the original one which handles checkpointing
  - + If the checkpoint signal arrives in the middle of signal masking, it will be sent to the current thread to perform checkpointing
- Guarantee CUDA API's proper execution in signal handler
  - + We conducted tests to verify that CUDA APIs which are used in our CUDA checkpointer such as *cudaMemcpy, cudaThreadSynchronize()* perform properly in the signal context

Evaluation

CUDA checkpointer microbenmark

- 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 Raccoon

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



BLCR's overhead increases linearly with data size

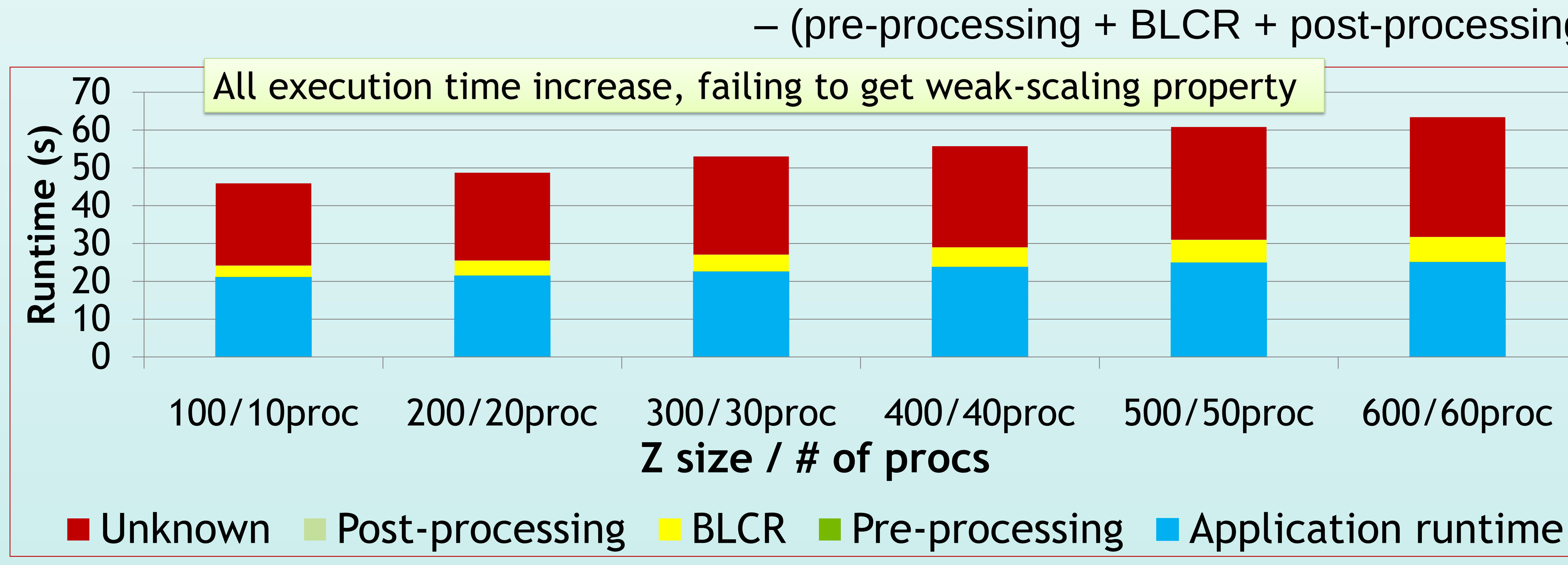
Experiment Environment

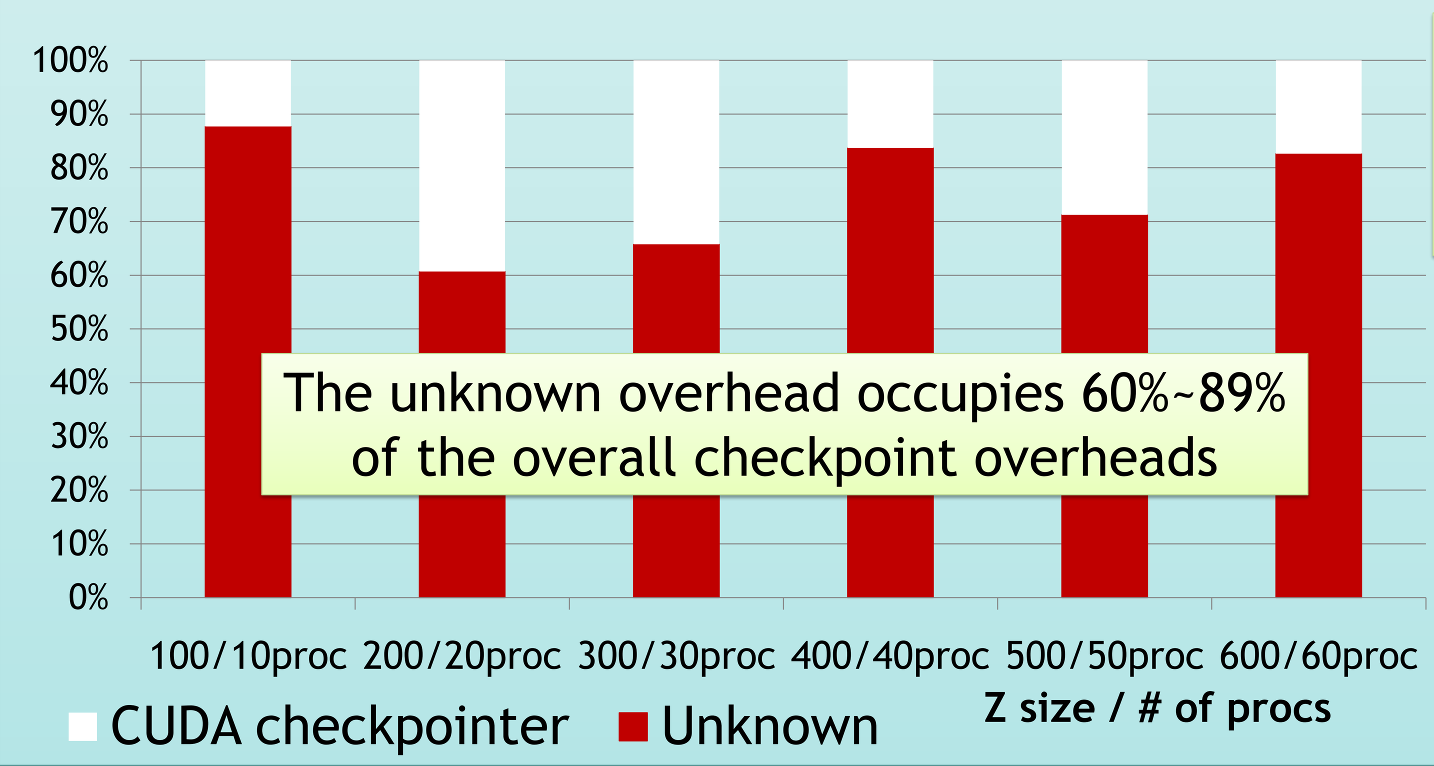
	Raccoon	TSUBAME
CPU/Memory	Intel i7 920 2.67GHz (4core/8thread) 12GB Memory	AMD Opteron 880 2.4GHz x 8(16core) 32GB Memory
GPU/Memory	Tesla C2050, 2.6GB Memory	Tesla S1070, 4GB Memory
OS	CENTOS 5.4	SUSE 10.3
BLCR	0.8.2	0.8.1
CUDA	3.0	2.3
Open MPI	1.4.2	1.4.2
Network	Infiniband 4x DDR	Infiniband 4x SDR
Local Disk	SSD	HDD

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)

All execution time increase, failing to get weak-scaling property





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

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

Problem size	# of procs	BLCR (s)	Unknown (s)
500x500 x500	50	25.1	307
500x500 x500	40	25.7	273
400x400 x400	40	14.5	87.9

Future Work

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