Checkpointing a Subsystem Remotely

Gene Cooperman and Rohan Garg
{gene,rohgarg}@ccs.neu.edu

College of Computer and Information Science
Northeastern University, Boston, USA
and Université Fédérale Toulouse Midi-Pyrénées

August 7, 2018

*Partially supported by NSF Grant ACI-1440788 and OAC-1740218, by a grant from Intel Corporation, and by an IDEX Chaire d'Attractivité (Université Fédérale Toulouse Midi-Pyrénées) under Grant 2014-345.

Table of Contents

Outline

DMTCP History

The DMTCP project and antecedents began almost 15 years ago, and is widely used today: `http://dmtcp.sourceforge.net`

Typical use case for HPC: 12-hour batch time slot, an application is expected to finish in 18 hours.
(NOTE: A resource manager such as SLURM can send a signal one hour before the end of the time slot, giving the application time to checkpoint; DMTCP can then transparently checkpoint the state.)

Ease of use (unprivileged and transparent):

```
dmtcp_launch a.out arg1 arg2 ...
dmtcp_command --checkpoint # from other terminal or window
dmtcp_restart ckpt_a.out_*.dmtcp
```

(DMTCP also works for programs that are multi-threaded, distributed, MPI-based, ...)

Fundamental Research Question for the DMTCP Team

"What are the limits of checkpointing applications in the real-world?"

Previous work:

PROBLEM: An application may store a session id, tty, network peer address, TMPDIR for temporary files, process id, thread id, etc. On restart, some or all of these are likely to change.

SOLUTION: Process virtualization: interpose on all calls to such ids; replace actual id by DMTCP-defined virtual id.
PRINCIPLE: "Never let the application see a real id!" (See thesis of Kapil Arya for details; see thesis of Jiajun Cao for virtualization of the highly complex Verbs API of InfiniBand.)

- See ICPADS'16 (Jiajun Cao et al.) for checkpointing MPI applications using over 32,000 CPU cores. To the best of our knowledge, this is ***100 times larger*** than the previously largest transparent checkpointing study in the literature.

Outline

Goal of this Talk

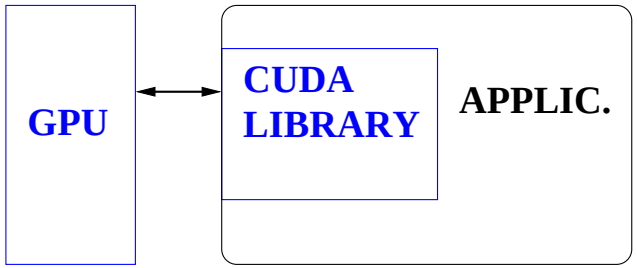
GOAL: ***To convince you of a general proxy paradigm for handling the "hard" checkpointing challenges that remain.***

We take as testbeds two such "hard" examples:

1. Checkpoint a CUDA application running on a GPU (Problem: how to save and restore the state of GPU hardware)
   (joint with Rohan Garg, Apoorve Mohan, Michael Sullivan)
   To appear, IEEE Cluster'18; see, also, technical report:
   `https://arxiv.org/abs/1808.00117`
2. Checkpoint MPI on NERSC/Cori (GNI network; no checkpoint-restart service to support it)
   (joint with Twinkle Jain, Rohan Garg, Gregory Price)
   In progress; see poster by Twinkle Jain at this MUG meeting.

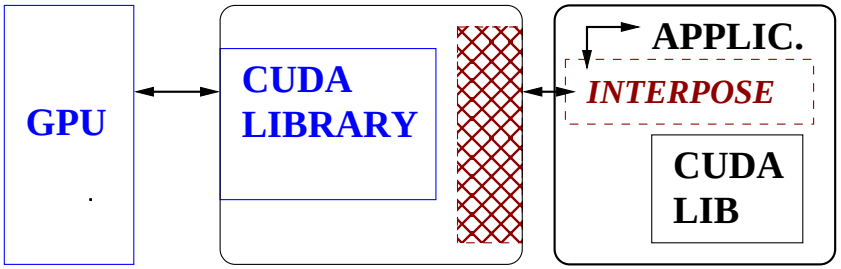
Proxies: The Secret Sauce

BEFORE:



Application

AFTER:



Proxy Process

Application Process

Basic Idea of a Proxy for System Services

PROBLEM: Often, system services are provided with the help of an auxiliary system that cannot be checkpointed.

EXAMPLES: GPU device hardware; MPI auxiliary process or MPI coordinator; sshd (ssh daemon); VNC server; etc.

SOLUTION:

A. Split user process into two: an application process with all of the application state; and a proxy process communicating with hardware or software for system services.

B. System service requests are passed from application process to proxy process through inter-process communication, and pass result back.

C. At checkpoint time, the proxy process is temporarily disconnected, and the application process is checkpointed as an isolated "vanilla" Linux process. (Note: the proxy process must be in a quiescent state (no unfinished system service tasks) at checkpoint time.)

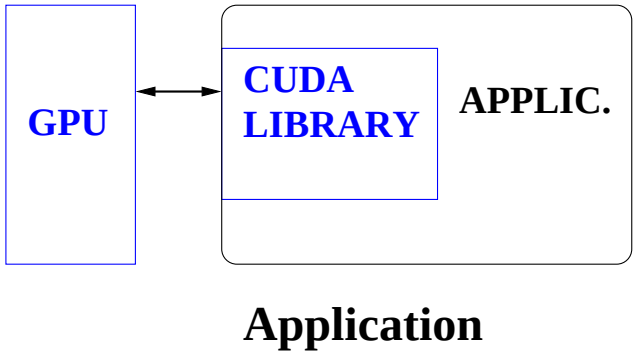
D. At restart time, a *new* proxy process re-connects with the system service and with the restarted application. The application process then replays some old system service requests, restoring system to a state that equivalent to pre-checkpoint time.

For more on this, see the thesis of Rohan Garg (in progress).

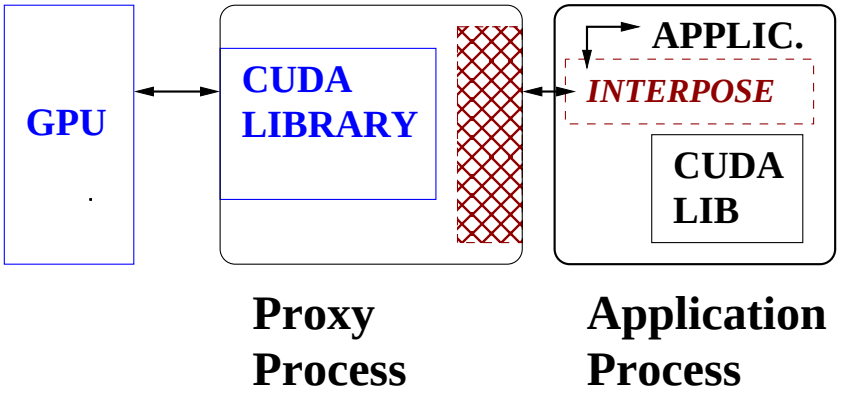
(NOTE: Proxies have been used before. For example, this is the basis of an old trick for checkpointing VNC sessions. We propose it here as a general paradigm for checkpoint-restart.)

Proxies: The Secret Sauce (again)

BEFORE:



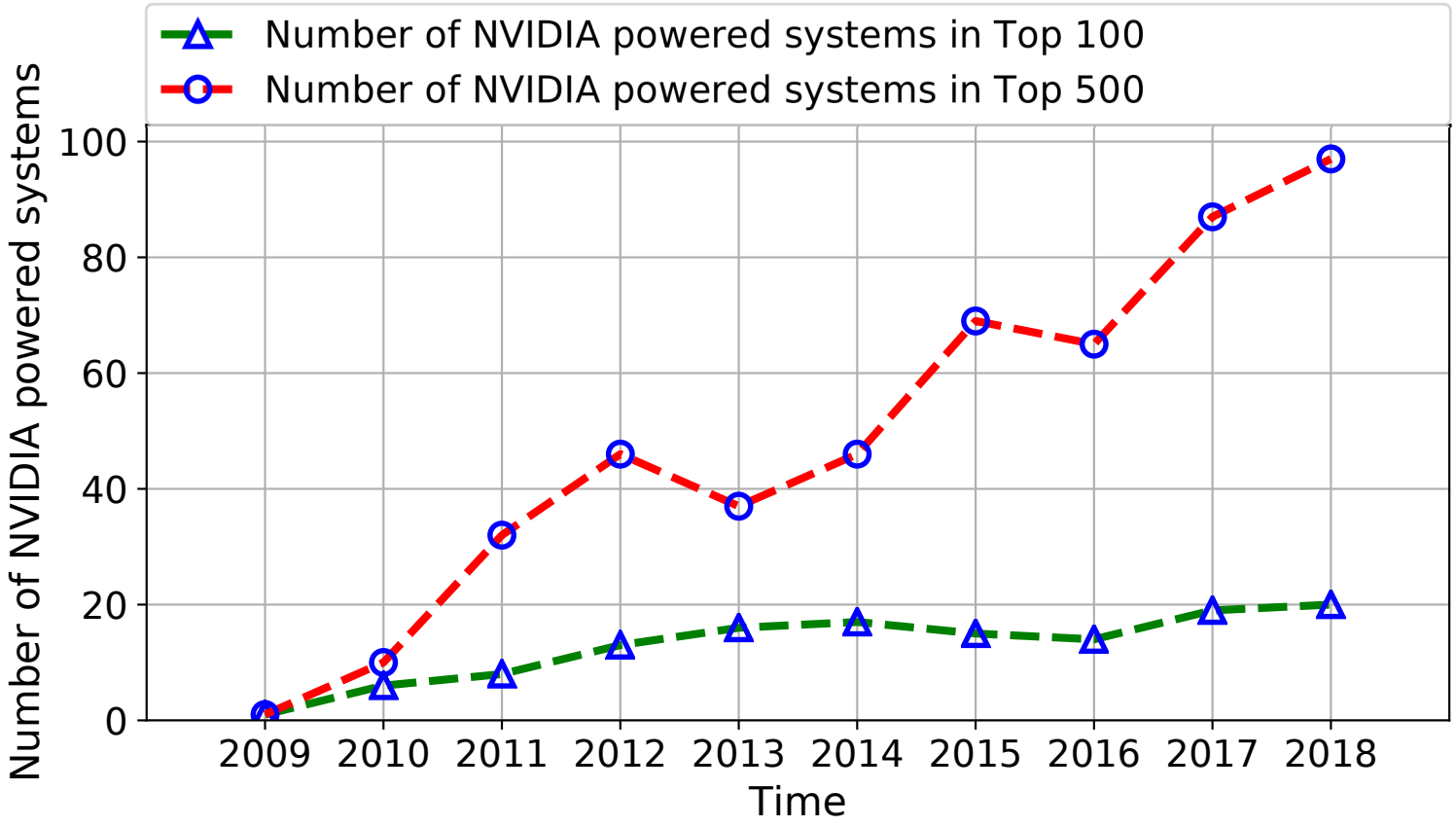
AFTER:



*First tell them what you're going to say; then say it; and then tell them what you said.

Outline

GPUs are a Growing Proportion of the TOP 500



Why is Checkpointing CUDA/GPU Difficult?

The system service in this case is the set of CUDA operations performed jointly by the CUDA library and the GPU device.

Modern CUDA programs almost always call many, short CUDA kernels. This takes advantage of the strong scheduling abilities for NVIDIA kernels.

Most checkpointing approaches take advantage of this to "drain" any active GPU kernels, and then checkpoint when no more GPU kernels are active.

Ideally, a checkpoint-restart package would "drain" all state from device to host, and then checkpoint.

PROBLEM: Since CUDA 4, CUDA introduced zero-copy transfers through CUDA managed memory. The CUDA API does not allow a user to "sync" the host application memory by copying all CUDA managed memory from device to host!

RESULT: ***Progress in checkpointing CUDA stalled in 2011, when CUDA 4 was introduced.***

Timeline of Progress in Checkpointing CUDA/GPU

2009: CheCUDA

2009: vCUDA

2011: NVCR

2011: CheCL (but for OpenCL *(not CUDA)*)

2011: **Cuda 4** introduces CUDA managed memory NB: CUDA library can no longer be re-initialized on restart; *all previous checkpointing results no longer work.*

2013: **Cuda 6** introduces CUDA UVM-Lite

2016: **Cuda 8** introduces CUDA UVM-Full

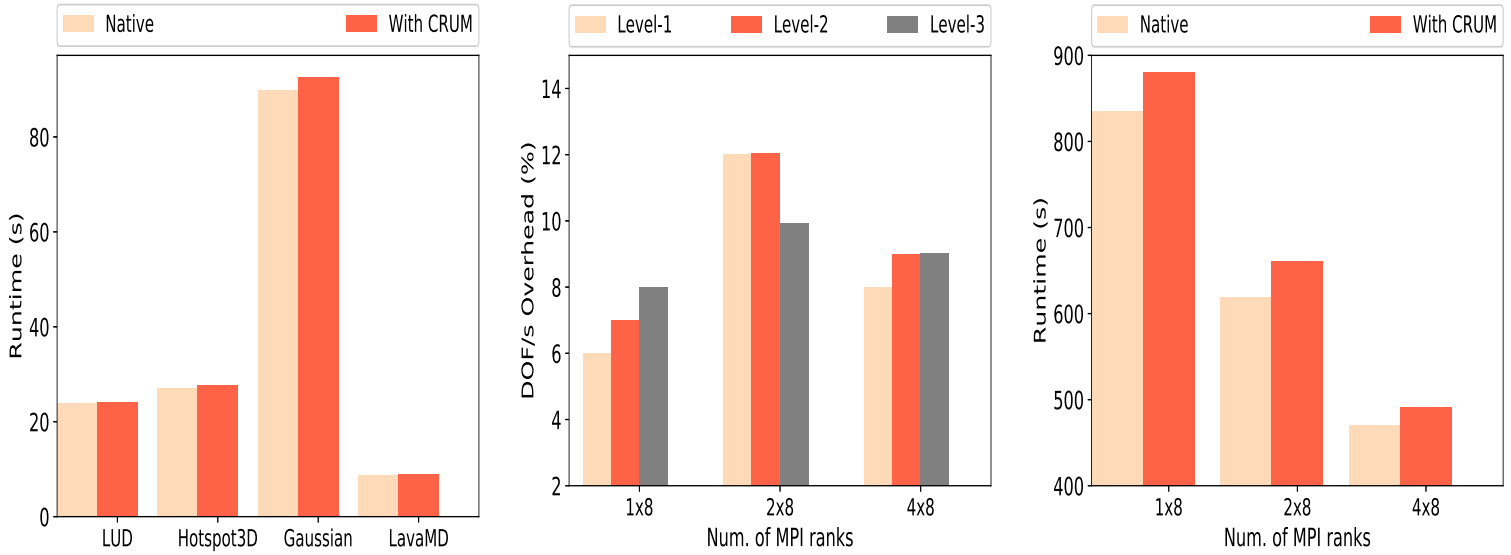
2016: CRCUDA – works for CUDA 7, but still no support for CUDA 4's managed memory.

2018: **CRUM** is first checkpoint-restart system to support CUDA managed memory, including CUDA UVM (Unified Virtual Memory) of the latest CUDA releases:
Uses proxy-based shadow paging

CRUM Novelty: Proxy-based Shadow Paging

- *Shadow UVM page synchronization:* Catches memory transfers between proxy and application through memory permissions and segfault detection. The difficulty for transparent checkpointing with CUDA-managed memory: **How to make this efficient?**
  See the CRUM paper (Algorithm 1, shadow-page synchronization) for details.
- Enables fast *forked checkpointing model* for UVM memory that overlaps writing a checkpoint image to stable storage, while the application continues. *Almost free benefit of our approach!* (This was difficult in the past due to the need to share memory among the GPU device, and the UVM-based host that was split among parent and forked child processes.)

CRUM (Checkpoint-Restart for CUDA Unified Memory): Runtime



a Rodinia Benchmark.

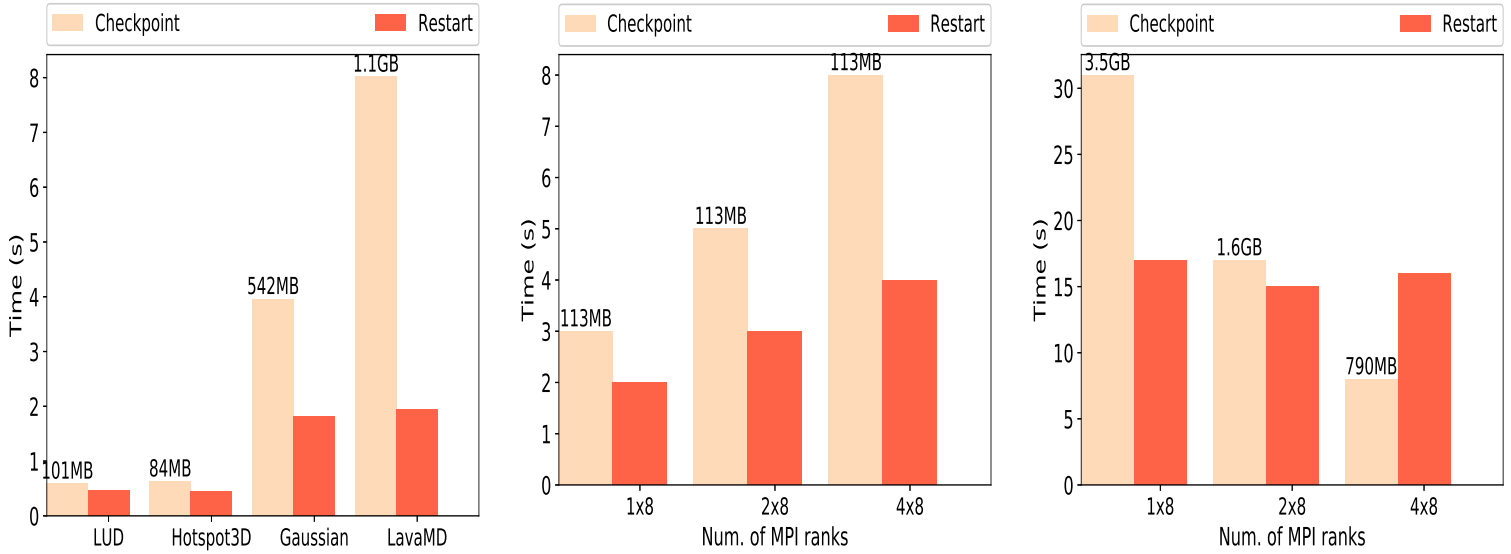
b HPGMG-FV Benchmark.

c HYPRE Benchmark.

Figure: Runtime overheads for different benchmarks under CRUM.

Operating environment: Four NVIDIA Tesla P100's per node; CUDA 8 (max config: four nodes with 4 GPUs per node and 8 MPI ranks per node)

CRUM (Checkpoint-Restart for CUDA Unified Memory): Checkpoint-Restart Time and Image Sizes



a Rodinia Benchmark.

b HPGMG-FV Benchmark.

c HYPRE Benchmark.

Figure: Checkpoint-restart times and checkpoint image sizes for different benchmarks under CRUM.

Operating environment: Four NVIDIA Tesla P100's per node; CUDA 8 (max config: four nodes with 4 GPUs per node and 8 MPI ranks per node)

Outline

NERSC: Use Proxies to Support Cray MPI over GNI Network on Cori

GOAL: "If you have a hammer, make MPI look like a nail!"

The system service provided by the MPI proxy is now:

the set of MPI ranks, communicators, groups, topologies, etc.,

along with any communication requests. Checkpoint can only occur when all ongoing communication requests are complete.

Using Proxies for Cray MPI/GNI on Cori

With proxies, the GNI network is hidden inside the MPI system service. No need to directly support the GNI network!

By isolating the application process from the MPI service, one even gains the ability to checkpoint a Cray MPI/GNI computation on Cori, and to restart over MVAPICH MPI/InfiniBand at the Ohio Supercomputer Center! 

(Thank you to the organizers of the MVAPICH User's Group meeting at the Ohio Supercomputer Center.)

(See poster of Twinkle Jain (this meeting) for further details.)

Questions?

THANKS TO THE MANY STUDENTS AND OTHERS WHO HAVE CONTRIBUTED TO DMTCP OVER THE YEARS:

Jason Ansel, Kapil Arya, Alex Brick, Jiajun Cao, Tyler Denniston, Xin Dong, William Enright, Rohan Garg, Twinkle Jain, Samaneh Kazemi, Jay Kim, Gregory Kerr, Apoorve Mohan, Mark Mossberg, Gregory Price, Manuel Rodríguez Pascual, Artem Y. Polyakov, Michael Rieker, Praveen S. Solanki, Ana-Maria Visan

QUESTIONS?