Transparency Checkpointing Software Test Benches to Improve Productivity of SOC Verification in an Emulation Environment

{Ankit Garg, K. Suresh, Jeff Evans}, Mentor, A Siemens Business,
{Gene Cooperman, Rohan Garg}, Northeastern University
Emulator as Enterprise Platform

- Emulation moved to Virtualization
  - Virtual Data center friendliness
  - Enterprise level usage
  - Moved to Application age
Efficient Emulator Utilization Desired

• Emulators allow jobs sized as integral multiple of minimum partition size
  – Complexities for job scheduling as jobs can be of varying sizes
• Non-preemptive nature of Virtual Emulation jobs results in inefficiencies and under utilization of resources
• Underutilization of expensive resource like Emulator directly affects cost of ownership
Making Virtual Emulation Jobs Preemptive

Solution:
• Checkpoint-Restore can solve problem of non preemption

Challenges:
• Virtual Emulation Jobs requires checkpoint-restore of two parts
  – Hardware Checkpoint-Restore: available for many years
  – Software Testbench Checkpoint-Restore: an elusive problem to solve
    • Virtual testbench environments can be non-deterministic, multi-threaded/process
    • Involves multiple languages like C/SystemC, SystemVerilog, etc.
Software Checkpoint-Restore

• Application-level checkpoint-restore
  – Difficult to maintain, complicated with multiple languages involved
  – Requires changes in user code; third-party libraries add more complexities

• Checkpoint-Restore supported by kernel modules
  – Requires root privileges
  – Difficult to maintain: closely coupled with kernel, which has frequent changes

• Replay Software testbench using stimulus from the hardware
  • Time consuming; capturing all stimulus may result in large database size
  • Transparent checkpoint-restore without change to user code or kernel
Background - DMTCP

• DMTCP: Distributed MultiThreaded Checkpointing
  – Transparently checkpoint-restores state of running application
  – Operates directly on user binary executable
    • No root privileges needed
    • No loading of kernel modules
    • No application source code change needed
    • No re-linking/re-compilation needed
  – Mature: 11 years in development
  – Robust user base with more than 11 thousand downloads
DMTCP Generic Application Checkpoint/Restart

dmtpc_launch
--with-plugin=libUsrPlugin.so
./userJob.exe

libUserPlugin.so

userJob.exe

DMTCP
User Plugin

Main Process

User Job

Threads

ckpt command

dmtpc_command --checkpoint

Restarted (Main) Process

User Job

./dmtpc_restart_script.sh

DMTCP
User Plugin

DMTCP
User Plugin

User space

Hardware

Host-1

Host-2
DMTCP Emulation Application
Checkpoint/Restart

```
dmtcp_launch \n--with-plugin=libEmuPlugin.so \n./emuJob.exe
```

```
emuJob.exe

libEmuPlugin.so

libEmuDriver.so

DMTCP

Emulation Job

Emulator libs.
libc

Main Process

;; = Threads

```

if (sim.cycles == 1000000)
{
  dmtcp_checkpoint();
}
```

```
/dmtcp_restart_script.sh

DMTCP

Emulator Job

Emulator libs.
libc

Restarted (Main) Process

Emulator-1 <-> Host-1

Emulator-2

Host-2

User space

Hardware

Emulator-1

Host-1

3/2/2022
Motivating Use Cases (I)

• Skipping repeated initial sequence
  – E.g. hardware reset phase or boot-up
  – Large runtime time after which actual test start
  – Can save lot of emulation runtime by taking checkpoint right after this repeated sequence
Motivating Use Cases (II)

- Better Job Management Policies
  - Enable a Pre-emptive scheduling policy
  - Job migration possible leading to more efficient Emulator utilization
  - Large capacity jobs having low priority will get fair chance to execute
Motivating Use Cases (III)

• Debugging from past simulation time
  – Full system Checkpoint capability let engineers debug just prior to first appearance of issue
  – Save lot of time in case issue occurs only after a run of long duration
  – Users can take periodic checkpoint and restart from the corresponding window
DMTCP Emulation Integrated Checkpoint flow

- Flowchart describing steps taken for checkpointing two parts
  - Hardware checkpoint is done by Emulator
  - Software checkpoint is done by DMTCP
DMTCP Emulation Integrated Restart/Restore flow

• Flowchart describing steps for restoring two part
  – Process tree is restored by DMTCP
  – Hardware State is restored by Emulator
Case Study: Skipping The OS Boot In An OEM Company's SOC Validation Environment

- SOC consists of a CPU, Memory subsystem, switching fabric and peripherals
- Hybrid environment
  - Part of SOC modeled on workstation
  - Part of SOC modeled in emulator
- SOC validation required booting an OS to run applications being validated
  - Boot of the OS takes on the order of hours to days
- DMTCP eliminated OS boot time allowing more applications to run per user per day
Case Study: Skipping The OS Boot In An OEM Company's SOC Validation Environment

- Regression suite consists of 20 jobs
- Each job similar profile around 1 hour OS boot and 1 hour of execution
- DMTCP based checkpointing was used to checkpoint just after boot
- Restoration takes around 5 minutes
Case Study: Results

- Regression suite emulation time comparison
Case Study: Challenges and Solutions

• Large files were getting checkpointed increasing checkpoint time and database size
  – DMTCP emulation plugin created a skip list for read only files

• Checkpoint database was not portable to another site
  – File paths were preserved during checkpointing
  – This was solved using file path virtualization plugin of DMTCP
  – This allowed file paths to be changed at restore time
Next Steps

• Work on developing pre-emptive capabilities for emulation jobs

• Integrate job migration capability to make job scheduling more flexible

• Discover practical challenges in deployment

• Some use cases have a remote process on Windows machine, additional work is required on this front
Questions?