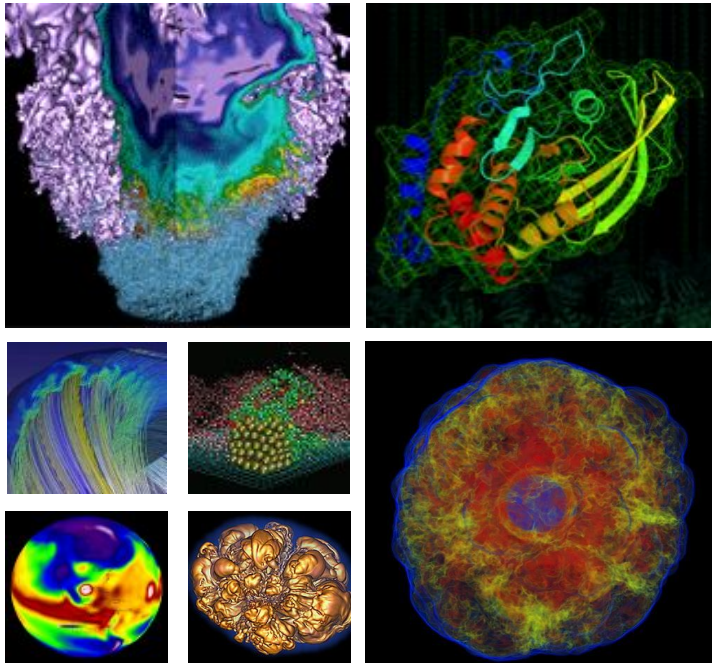
Optimizing Checkpoint-Restart Mechanism for HPC with DMTCP in Containers at NERSC



CUG Conference
May 5-9 2024
Perth, Australia

Madan Timalsina
NERSC/NESAP Postdoc
Data & AI Services

Coauthors (NERSC): W. Arndt, J. P. Blaschke, L. Gerhardt, and N. Tyler

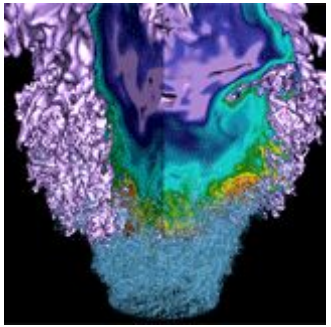
U.S. DEPARTMENT OF ENERGY | Office of Science

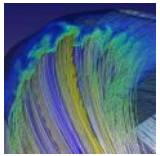
Outline

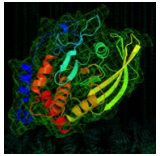
- National Energy Research Scientific Computing Center (NERSC)
- Checkpoint-Restart Mechanism
- DMTCP (Distributed MultiThreaded CheckPointing) Overview
- Checkpointing and Restarting Jobs using DMTCP
  - At NERSC Perlmutter
  - At NRSC Perlmutter inside the Containers
- Results
- Conclusion

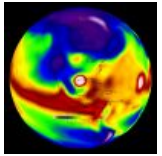
NERSC:

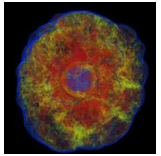
National Energy Research Scientific Computing Center

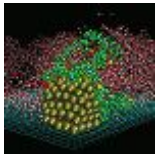












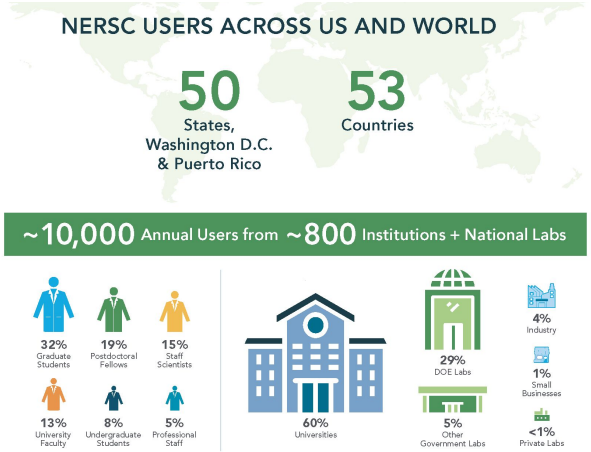
NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NERSC

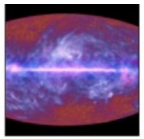
- NERSC (at LBNL), a state-of-the-art supercomputer, is the mission High Performance Computing and Data facility for the DOE Office of Science
- Our mission involves deploying supercomputer systems designed for pioneering simulations and large-scale data analytics
- NERSC Science Acceleration Program (NESAP) fosters collaboration with partners to optimize scientific research for next-generation computational architectures and systems



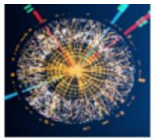
Acknowledged in ~ 5,800 refereed scientific publications & high profile journals since 2020

U.S. DEPARTMENT OF ENERGY | Office of Science

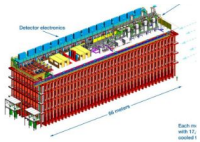

Palomar Transient Factory Supernova

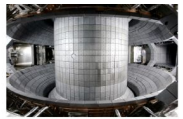

Planck Satellite Cosmic Microwave Background Radiation

Star Particle Physics


Atlas Large Hadron Collider

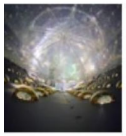

APS

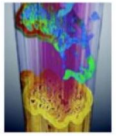

Dune


KStar


ARM


Katrin


Dayabay Neutrinos


ALS Light Source

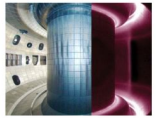

LCLS Light Source

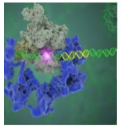

Joint Genome Institute Bioinformatics

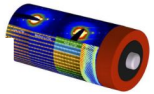

NSLS-II

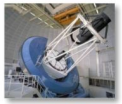

HSX


Majorana

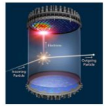

DIII-D


Cryo-EM


NCEM


DESI


LSST-DESC


LZ


IceCube


EXO

JBEI Joint BioEnergy Institute

NERSC Center Architecture

TOP 500 The List. #12, 113PF Peak

>5 TB/s

35 PB All-Flash Scratch

HPE Slingshot 11 ethernet-compatible interconnect
4 NICs/GPU node,
1 NIC/CPU node

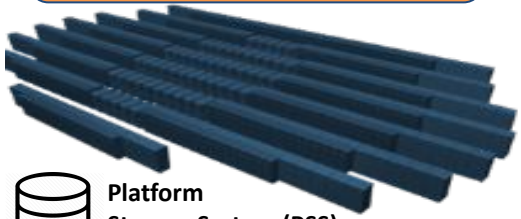
1,792 GPU-accelerated nodes
4 NVIDIA A100 GPUs+1 AMD "Milan" CPU
448 TB (CPU) + 320 TB (GPU) memory

3,072 CPU-only nodes
2 AMD "Milan" CPUs
1,536 TB CPU memory

1.6 TB/s

(2026)

NERSC-10



Platform Storage System (PSS)

Quality of Service Storage System (QSS)

Workflow Environment
Management Environment

Spin container services

Off Platform Storage

- HPSS Tape Archive >1 EB — 200 GB/s
- Community File System 240 PB — > 800 GB/s
- /home 450 TB — > 10 GB/s
- DTNs, Gateways

Ethernet

Science Friendly Security
Production Monitoring
Power Efficiency
LAN

3.25 TB/s (26 Tbps)

ESnet ENERGY SCIENCES NETWORK
2 x 400 Gb/s
2 x 100 Gb/s

Experimental Facility | ASCR Facility | Home Institution | Cloud | Edge

Containers at NERSC

Containers are valuable to our scientific computing users

- Encapsulation, isolation, reproducibility, portability, and even scalability

NERSC supports user container workloads via Shifter

- Developed at NERSC to address security concerns of docker
- Enables scalability on HPC systems
- Users can build their images with docker, then easily convert to shifter with a simple pull command

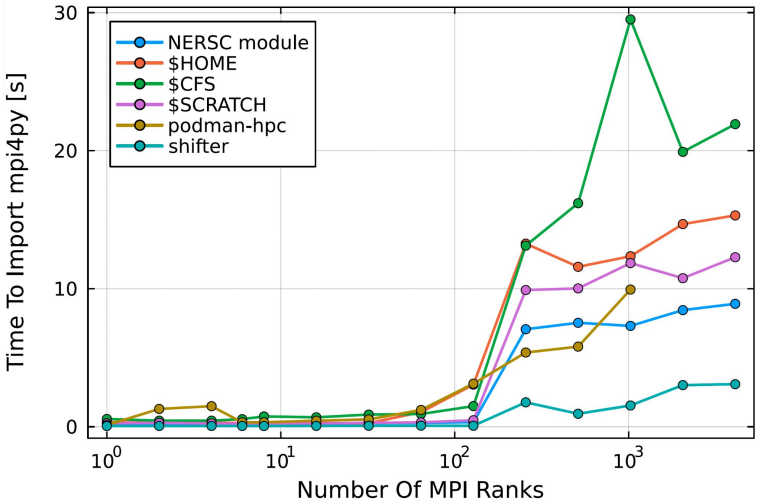
NERSC also supports podman-hpc

- NERSC built wrapper for podman (open source tool)
- All the benefits of shifter, but using OCI (Open Container Initiative) standard runtime
- A rootless containers enhances security, users can build images at NERSC

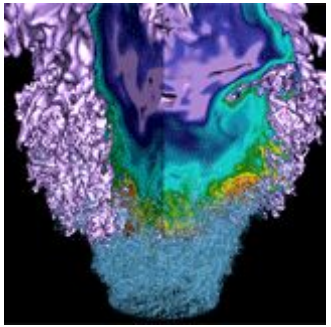
Performance Benchmark of Containers at NERSC

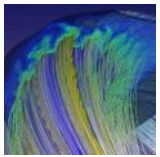
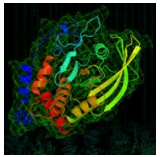
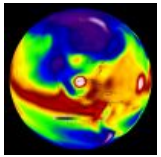
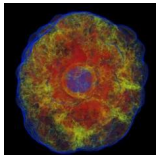
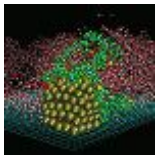
Scaling efficiency in scientific workflows with NERSC containers

- Enhances scaling efficiency of scientific workflows (reduced load times, even compared to optimized parallel file systems)
- Through encapsulation, containers introduce resilience in managing complex tasks
- At NERSC, Shifter and Podman-HPC offer scalable solutions supporting a wide range of research activities
- Containerized Checkpoint-Restart (C/R) mechanisms lead to faster and more reliable data processing, accelerating scientific discovery



Checkpoint-Restart (C/R)



NERSC

Checkpointing and Restarting (C/R)

- ***Checkpointing*** involves preserving the current state of a running process (jobs) by creating a checkpoint image file.
  - This includes capturing the memory, executing instructions, I/O status, and related data of the running process into a file
- ***Restarting*** the process is possible using the checkpoint file.
  - This enables the process to resume its execution from where it was saved (rather than from the beginning), either on the same or a different computer, seamlessly continuing its operation

It's a crucial capability in High-Performance Computing (HPC) due to complex and time-consuming computations. It can reduce startup times in applications and facilitates batch scheduler optimizations, including preemption

C/R: Benefits

HPC/NERSC Perspective

- **Enhanced Job Prioritization:** Potential preempting of less critical jobs for more urgent or time-sensitive tasks
- **Optimized Node Utilization:** Efficient backfilling, maximizing node usage, especially for large reservations
- **Uninterrupted Operations:** Run checkpointing jobs until system maintenance, ensuring minimal disruption
- **Enhanced Reliability:** Potentially checkpointing all jobs before unexpected power outages for system stability and job recovery

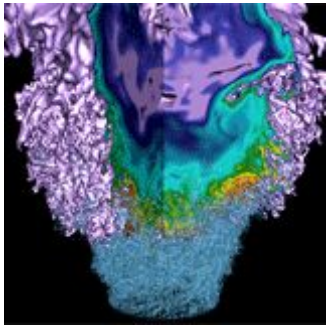
User Perspective

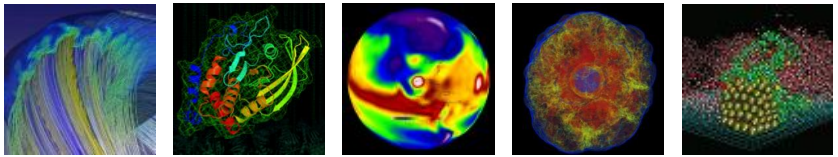
- **Extended Runtime:** Allow jobs to exceed walltime limits by resuming from checkpoints
- **Increased Throughput:** Leveraging gaps in the Slurm schedule to optimize job processing
- **Extended Interactivity:** Save and resume interactive sessions seamlessly (if it's time to go home to dinner, then checkpoint and restart the next day!)
- **Efficient Debugging**: Pause, identify errors, and restart jobs from specific checkpoints for iterative debugging

Challenges in C/R

- ***Complexity for User Transparency:*** Requires extensive effort to create a seamless experience for users during checkpointing and restarting processes
- ***MPI Support Challenges:*** Particularly intricate due to the combination of various MPI implementations (e.g., MPICH, OpenMPI) and networks (e.g., Slingshot, Infiniband), resulting in the need for multiple versions (MxN problem)
- DMTCP serves as a solution for overcoming these challenges
- For more details, refer to the NERSC documentation

DMTCP: Distributed MultiThreaded CheckPointing





NERSC documentation, DMTCP website, DMTCP github

DMTCP: Simplifying Checkpoint-Restart

An open-source tool offering seamless checkpoint and restart functionalities for distributed applications across clusters, grids, cloud environments etc

Preserves Application State Seamlessly

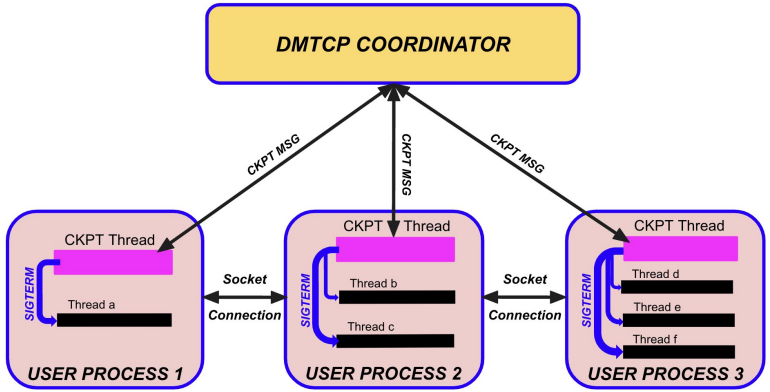
- **No Code or Kernel Modifications:** Stores complex threaded or distributed applications without altering their code or the Linux kernel
- **Accessible to Users:** Doesn't require special system privileges, allowing operation without root access

User-Friendly Checkpointing

- **Seamless User-Space Operation:** Performs checkpoints without changing user code or system settings
- **Versatile Application Support:** Works with diverse applications like MPI, OpenMP, Python, C/C++, Fortran, shell scripts, and resource managers (e.g., Slurm)

How does DMTCP Work?

DMTCP Architecture: Coordinated Checkpointing



DMTCP Coordinator to Computation Ratio: One DMTCP coordinator manages one checkpointable DMTCP computation

Multiple Checkpointable Computations: Multiple coordinators can handle separate computations, each independently checkpointtable

Checkpoint Thread vs. User Thread: Only one of the DMTCP checkpoint thread or user thread can be active at any given time, not both concurrently

Fault Tolerance without Single Point of Failure: No single point of failure if checkpoint image files are backed up. Even if the coordinator fails, the system can restart from the last checkpoint

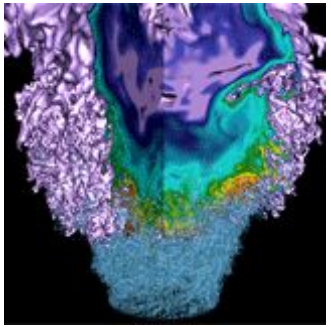
Preservation of Runtime Libraries: Runtime libraries are saved as part of the memory image. Applications continue using the same library API

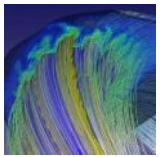
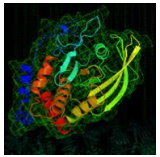
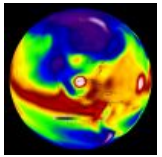
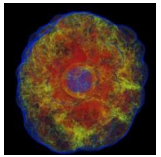
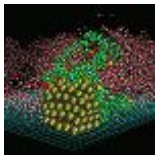
Inclusion of Linux Environment Variables: Linux environment variables are part of the memory image. Special DMTCP plugin needed to modify saved environment variables during checkpoint

User-Space Functionality: Entire process operates in user-space; no need for administrative privileges for its functioning

RESTART: same as ckpt, but in opposite order

Checkpoint/Restart (C/R) Jobs with DMTCP at Perlmutter



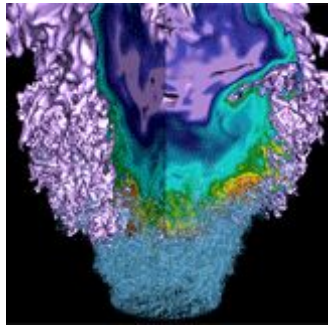






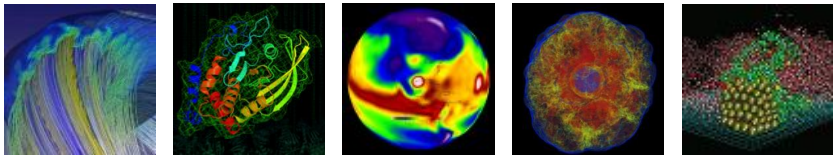
NERSC documentation

How does it work?

- NERSC CR Module (nersc_cr) manages Checkpoint/Restart (C/R) jobs
- Users can set the checkpoint interval with the -i option and submit their job either manual or automated way
- The batch system initiates job execution by allocating the requested nodes within available time frames, prioritizing higher-priority jobs
- As a part of automatic resubmission, the job runs until it receives signal USR1 (*--signal=B:USR1@60*) 60 seconds before it hits the allocated time limit
- Upon receiving the signal, the *func_trap* function gets executed, which in turn executes
  - *ckpt_command* if specified
  - Requeues the job and then updates remaining walltime for requeued job
- Steps 2-4 are repeated until the job completes or reaches the desired duration
- User checks the job results upon completion
- More details, NERSC documentation

Checkpoint/Restart (C/R) Jobs inside Container using DMTCP: Perlmutter





SHIFTER

NERSC

podman

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Requirements

- DMTCP cannot be checkpointed from outside the containers. It must be included within the container when it is build
- The simulation package can be built in many ways:
  - During the container's build process
  - After the container has been built, by linking the source code from elsewhere
  - Extend the functionality by building on top of an existing container, enabling quick experimentation with minimal modifications

  All methods have been tested and verified

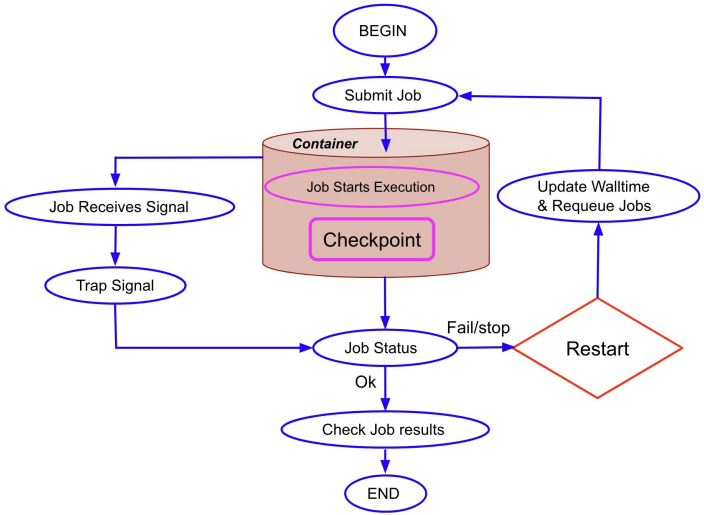
```
FROM my_application_container:latest

RUN git clone
https://github.com/dmtcp/dmtcp.git  \
   && cd dmtcp \
   && ./configure  && make \
   && make install
```

- In the context of Geant4, various versions can be directly sourced from the CernVM File System (CVMFS), facilitating easy access to multiple versions for testing and deployment

Automated C/R Strategies

- Users submit their job scripts, with the checkpoint interval (-i), incorporating DMTCP within containers, along with necessary software packages like Geant4, CP2K
- Custom batch scripts manage checkpoint-restart tasks, which isn't directly feasible within the container environment
- The script initiates checkpointing via *restart_job* function including a *start_coordinator* to initiate jobs and executes using *dmtcp_launch*, ensuring efficient job lifecycle management
- Upon receiving termination signals *(SIGTERM)*, the setup facilitates checkpointing, ensuring continuous job execution and effective resource utilization
- This method ensures efficient handling of Checkpoint/Restart processes, aligning with the specific needs of HPC environments, leading to the successful completion of jobs



Manual C/R Strategies

- Initial job submissions include checkpointing to set a baseline for potential restarts
- Users actively monitor job progress through output and error logs to detect interruptions
- Checkpoint files act as job snapshots, enabling precise recovery from disruptions
- Manual intervention allows for restarting jobs using these checkpoints, ensuring progress continuity
- The manual C/R process is a cycle of submission, monitoring, checkpointing, and restarting as needed
- This approach gives users direct control to address specific computational challenges within the job lifecycle

Results

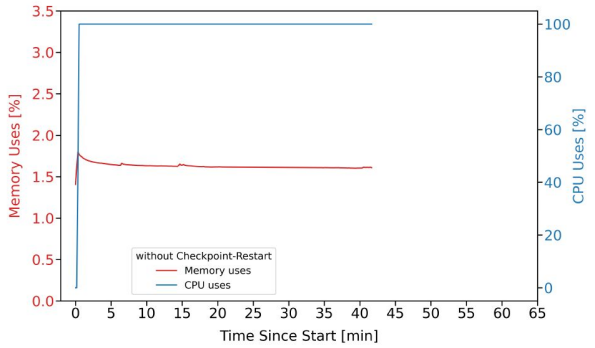
- Geant4 is a crucial tool for High Energy Physics (HEP) research, has been thoroughly tested and has passed the assessments
- Tested across multiple Geant4 versions (10.5, 10.7, 11.0) covering diverse simulation environments and particle interactions
- Performed tests using Shifter and Podman-HPC container images
- Each job, regardless of complexity, was preempted and then successfully resumed, highlighting the C/R mechanism's robustness
- Planning to extend our research into additional fields such as material science, with ongoing tests using CP2K

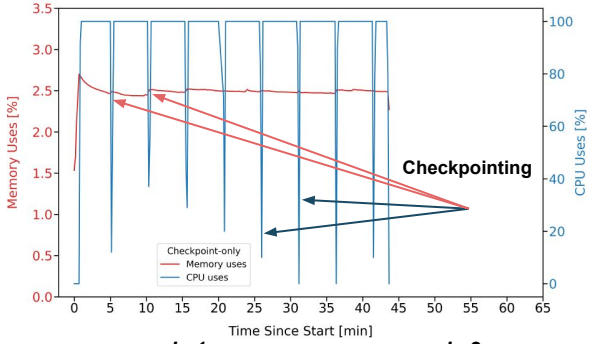
Results

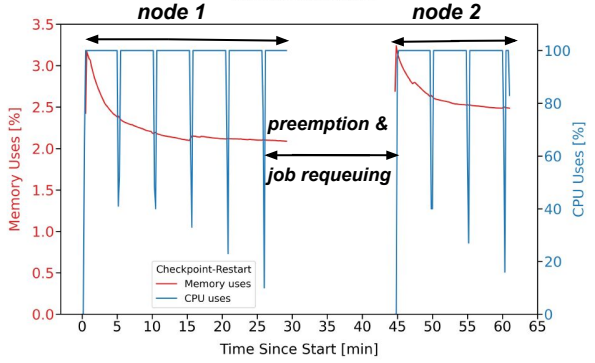
Impact of C/R on resource utilization

- ***Without C/R:*** The normal operational regime shows consistent CPU use and effective memory management
- ***Checkpoint-Only:*** Regular peaks in memory usage at checkpoints, with corresponding declines in CPU utilization
- ***Checkpoint-Restart:*** Spikes in memory use during checkpoints followed by corresponding declines in CPU utilization. A gaps in memory and CPU utilization due to preemption and job requeuing. We can see job has restarted in the different node afterward

C/R techniques exhibit a slight increase in computation time and memory usage (< 1%) because of DMTCP and associated file loading; however, this approach greatly reduces time and resource use by resuming the task from the last checkpoint state, enhancing efficiency







Future Directions

- Extend HEP-based simulation strategies to material science, enhancing research applications
- Plan to broaden testing with various material science software, including CP2K, VASP, BerkeleyGW, and LAMMPS
- Explore the use of MANA (MPI-Agnostic Network-Agnostic) for checkpointing to improve efficiency in MPI applications
- Leverage MANA's split-process approach for more streamlined and robust computational workflow management

Conclusion

- The study showcases the effectiveness of checkpoint-restart techniques using DMTCP in High-Performance Computing environments
- Demonstrated utility across HPC platforms including container technologies like Shifter and Podman-HPC
- This method is particularly valuable in complex, lengthy HPC computations, significantly reducing time and cost associated with process restarts
- Implementation in diverse simulations including HEP, medical science, and material science (test ongoing), showcasing versatility
- Highlights a critical advancement in efficient and reliable computational methodologies
- Confirms the effectiveness of the technique and opens new opportunities in computational science

Thank You

Some DMTCP Commands

dmtcp_coordinator -- coordinates checkpoints between multiple processes.

Example: `-i, --interval`: Time interval between automatic checkpoints (sec)

`--exit-on-last`: Auto-exits when the last client disconnects

dmtcp_launch -- Start a process under DMTCP control

Example: `-i, --interval`: Time interval between automatic checkpoints (sec)

`-j, --join-coordinator`: Join an existing coordinator, raise error if one doesn't already exist

dmtcp_restart -- Restart processes from a checkpoint image.

Example: `-h, --coord-host`: Specifies the hostname where `dmtcp_coordinator` is running

`-i, --interval`: Time interval between automatic checkpoints (sec)

dmtcp_command -- Send a command to the dmtcp_coordinator remotely

Example: `-s --status`: Prints status message

`-k --kill`: Kills all nodes

`-q --quit`: Kills all nodes and quits

For more details, refer to the DMTCP website, NERSC documentation

C/R Jobs with DMTCP CR Module: Perlmutter

Sample Job Script: Manual submission and resubmission

Original script

```
#!/bin/bash
#SBATCH -J test_cr
#SBATCH -q debug
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 00:07:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err

#user settings
export OMP_PROC_BIND=spread
export OMP_PLACES=threads
export OMP_NUM_THREADS=2


./g4.sh
```

run.sh

```
#!/bin/bash
#SBATCH -J test_cr
#SBATCH -q debug
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 00:07:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --time-min=00:05:30

#user settings
export OMP_PROC_BIND=spread
export OMP_PLACES=threads
export OMP_NUM_THREADS=2

#for c/r with dmtcp
module load dmtcp nersc_cr

#Checkpointing once every 5 min (change
interval as needed)
start_coordinator -i 300

#running under dmtcp control
dmtcp_launch -j ./g4.sh
```

restart.sh

```
#!/bin/bash
#SBATCH -J test_cr
#SBATCH -q debug
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 00:07:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --time-min=00:05:30

#user settings
export OMP_PROC_BIND=spread
export OMP_PLACES=threads
export OMP_NUM_THREADS=2

#for c/r with dmtcp
module load dmtcp nersc_cr

#Checkpointing once every 5 min (change
interval as needed)
start_coordinator -i 300

#restarting from dmtcp checkpoint files
./dmtcp_restart_script.sh
```

To run:

```
sbatch run.sh
```

To restart:

```
sbatch restart.sh
```

C/R Jobs with DMTCP CR Module: Perlmutter

Sample Job Script: Auto resubmission

```
#!/bin/bash

# SLURM Job Submission Directives
#SBATCH -J test_cr       # Job name: test_cr
#SBATCH -q debug         # Queue name: debug
#SBATCH -N 1             # Number of nodes: 1
#SBATCH -C cpu           # Node type: CPU
#SBATCH -t 00:07:00      # Time limit: 7 minutes
#SBATCH -e %x-%j.err     # Standard error file format
#SBATCH -o %x-%j.out     # Standard output file format
#SBATCH --time-min=00:05:30 # Minimum time: 5 min 30 sec

#SBATCH --signal=B:USR1@60   # Send USR1 signal 60s before time limit
#SBATCH --requeue            # Enable job requeuing
#SBATCH --open-mode=append   # Open output files in append mode

# Load necessary modules for checkpoint/restart
module load dmtcp nersc_cr

# Start DMTCP coordinator for checkpointing every 5 minutes
start_coordinator -i 300
```

New for C/R jobs with DMTCP Automatic resubmission

```
# Checkpoint/Restart Logic
if [[ $(restart_count) == 0 ]]; then
   # First run of the job

   # Set OpenMP environment variables
   export OMP_PROC_BIND=spread
   export OMP_PLACES=threads
   export OMP_NUM_THREADS=2

   # Launch the user's application under DMTCP control
   dmtcp_launch -j ./my_g4.sh &
elif [[ $(restart_count) > 0 ]] && [[ -e dmtcp_restart_script.sh ]];
then
   # Restarting the job

   # Execute the restart script
   ./dmtcp_restart_script.sh &
else
   # If unable to restart, print error message and exit
   echo "Failed to restart the job, exit" ; exit
fi

# Requeuing Logic
ckpt_command=ckpt_dmtcp     # Command for additional checkpointing
requeue_job func_trap USR1 # Requeue job upon receiving USR1 signal

# Wait for all background processes to finish
wait
```

To run: `sbatch run.sh`

- The *requeue_job* function captures the specified signal (e.g., USR1) and then executes the *func_trap* function upon its reception
- The *func_trap* function initiates checkpointing, prepares inputs for the next job, requeues the job, and updates remaining walltime

C/R Jobs with DMTCP within Container: Perlmutter

Sample Job Script: Manual submission and resubmission

Original script

```
#!/bin/bash
#SBATCH -J G4_test_cont
#SBATCH -q debug
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 01:00:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --time-min=00:06:00

## Additional directives...
#SBATCH --module=cvmfs
#SBATCH
--image=mtimalsina/geant4_dmtcp:9Nov2023

# Set the DMTCP_COORD_HOST variable
#to identify hosts or manage checkpoints
export DMTCP_COORD_HOST =$(hostname)

# Launch the job within the Shifter container
shifter --module=cvmfs
--image=mtimalsina/geant4_dmtcp:9Nov2023
/bin/bash ./my_g4.sh
```

run.sh

```
#!/bin/bash
#SBATCH -J G4_test_cont
#SBATCH -q debug
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 00:30:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --time-min=00:06:00

## Additional directives...
#SBATCH --module=cvmfs
#SBATCH --image=mtimalsina/geant4_dmtcp:9Nov2023

# Set the DMTCP_COORD_HOST variable
#to identify hosts or manage checkpoints
export DMTCP_COORD_HOST =$(hostname)

# Launch the job within the Shifter container
shifter --module=cvmfs
--image=mtimalsina/geant4_dmtcp:9Nov2023
/bin/bash ./test-checkpoint.sh
```

test-checkpoint.sh

```
#!/bin/bash
dmtcp_launch --interval 300 ./my_g4.sh
```

restart.sh

```
#!/bin/bash
#SBATCH -J G4_test_cont
#SBATCH -q debug
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 00:30:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --time-min=00:06:00

## Additional directives...
#SBATCH --module=cvmfs
#SBATCH --image=mtimalsina/geant4_dmtcp:9Nov2023

# Set the DMTCP_COORD_HOST variable
#to identify hosts or manage checkpoints
export DMTCP_COORD_HOST =$(hostname)

# Launch the job within the Shifter container
shifter --module=cvmfs
--image=mtimalsina/geant4_dmtcp:9Nov2023
/bin/bash ./dmtcp_restart_script.sh
```

Checkpoint image file

To run:

```
sbatch run.sh
```

To restart:

```
sbatch restart.sh
```

C/R Jobs with DMTCP within Container: Perlmutter

To run:

```
sbatch run.sh
```

```
#!/bin/bash

# Slurm directives for job properties
#SBATCH -J test-g4-cr       # Job name
#SBATCH -q regular          # Queue
#SBATCH -N 1                # Number of nodes
#SBATCH -C cpu              # CPU architecture
#SBATCH -t 01:00:00         # Wall clock time
#SBATCH -e %x-%j.err        # Error file
#SBATCH -o %x-%j.out        # Output file

#SBATCH --time-min=00:45:00     # Minimum time allocation
#SBATCH --comment=01:05:00      # Comment
#SBATCH --signal=SIGTERM@60     # Signal handling for termination
#SBATCH --requeue               # Requeue job if terminated
#SBATCH --open-mode=append      # Append mode for output files

## Additional directives...
#SBATCH --module=cvmfs                                  # Load module
#SBATCH --image=mtimalsina/geant4_dmtcp:Dec2023   # Container image

# Set the DMTCP_COORD_HOST variable
export DMTCP_COORD_HOST =$(hostname)

# Requeue function to resubmit the job on SIGTERM
function requeue () {
   echo "Got Signal. Going to requeue"
   scontrol requeue ${SLURM_JOB_ID}
}

# Trap SIGTERM to trigger requeue function
trap requeue SIGTERM

# Launch the job within the Shifter container
shifter --module=cvmfs --image=mtimalsina/geant4_dmtcp:Dec2023
/bin/bash ./test-auto.sh &

wait
```

Basic slurm directives

New for C/R jobs with DMTCP automatic resubmission

--comment sbatch flag is used to specify the desired walltime and to track the remaining walltime for the job after pre-termination

Export hostname to restart the job

Requeue function to resubmit the job

Trap signal (SIGTERM) to trigger requeue function

Launch the job within the Shifter container

C/R Jobs with DMTCP within Container: Perlmutter

test-auto.sh

```
#!/bin/bash

export DMTCP_COORD_HOST=$(hostname)
source my_env_setup.sh

# Function to restart or initiate the job
function restart_job() {
    start_coordinator -i 300

    if [[ $(restart_count) == 0 ]]; then
        # Initial job launch
        dmtcp_launch --join-coordinator -i 300 ./my_g4.sh
        echo "Initial launch successful."
    elif [[ $(restart_count) > 0 ]] && [[ -e $PWD/dmtcp_restart_script.sh ]];then
        # Restart the job
        echo "Restarting the job..."
        echo "Executing: $PWD/dmtcp_restart_script.sh"
        $PWD/dmtcp_restart_script.sh &
        echo "Restart initiated."

    else
        echo "Failed to restart the job, exiting." exit
    fi

    # Set up trap for checkpointing on termination signal
    trap ckpt_dmtcp SIGTERM
}

# Execute the function to restart the job
restart_job

# Wait for the job to complete or terminate
wait
```

This script provides functions for managing and monitoring SLURM jobs, including time tracking, signal trapping, job requeuing, and integration with DMTCP for checkpoint/restart functionality. It converts time to human-readable format, calculates remaining time for job scheduling, updates job comments accordingly, and manages job requeuing based on the remaining time

This function sets up and manages a job using DMTCP for checkpointing. It starts the job if it's the initial run. Or restarts it from a checkpoint if it's a subsequent run. Additionally, it configures a trap to automatically checkpoint the job when a termination signal is received

Your simulation code

Users can choose the checkpoint interval with the -i option.

C/R Jobs with DMTCP within Container: Perlmutter

```bash
#!/bin/bash

# Slurm directives for job properties
#SBATCH -J test-g4-cr-podman            # Job name
#SBATCH -q regular                      # Queue
#SBATCH -N 1                            # Number of nodes
#SBATCH -C cpu                          # CPU architecture
#SBATCH -t 01:00:00                     # Wall clock time
#SBATCH -e %x-%j.err                    # Error file
#SBATCH -o %x-%j.out                    # Output file
#SBATCH --time-min=00:45:00             # Minimum time allocation
#SBATCH --comment=01:05:00              # Comment
##SBATCH --signal=B:USR1@60              # Signal (previously used)
#SBATCH --signal=SIGTERM@60             # Signal handling for termination
#SBATCH --requeue                       # Requeue job if terminated
#SBATCH --open-mode=append              # Append mode for output files
## Additional directives...
#SBATCH --module=cvmfs                          # Load module
#SBATCH --image=mtimalsina/geant4_dmtcp:Dec2023   # Container image

# Set the DMTCP_COORD_HOST variable
export DMTCP_COORD_HOST=$(hostname)

# Requeue function to resubmit the job on SIGTERM
function requeue () {
   echo "Got Signal. Going to requeue"
   scontrol requeue ${SLURM_JOB_ID}
}
# Trap SIGTERM to trigger requeue function
trap requeue SIGTERM
#requeue_job func_trap USR1

# Launch the job within the Shifter container
podman-hpc run --userns keep-id --rm -it --mpi \
   -e SLURM_JOBID=${SLURM_JOB_ID} \
   -v /cvmfs:/cvmfs \
   -v $(pwd):/podman-hpc \
   -w /podman-hpc \
   mtimalsina/geant4_dmtcp:Dec2023 \
   /bin/bash ./test-auto.sh &

wait
```

```bash
#!/bin/bash

# Ensure the checkpoint directory exists and has the correct permissions
chmod 755 /podman-hpc
export DMTCP_COORD_HOST=$(hostname)
source my_env_setup.sh

# Function to restart or initiate the job
function restart_job() {
   start_coordinator -i 300

   if [[ $(restart_count) == 0 ]]; then
       # Initial job launch
       dmtcp_launch --join-coordinator --i 300 ./my_g4.sh
       echo "Initial launch successful."
   elif [[ $(restart_count) > 0 ]] && [[ -e $PWD/dmtcp_restart_script.sh ]]; then
       # Restart the job
       echo "Restarting the job..."
       echo "Executing: $PWD/dmtcp_restart_script.sh"
       $PWD/dmtcp_restart_script.sh &
       echo "Restart initiated."

   else
       echo "Failed to restart the job, exiting."; exit
   fi

   # Set up trap for checkpointing on termination signal
   trap ckpt_dmtcp SIGTERM
}

# Execute the function to restart the job
restart_job

# Wait for the job to complete or terminate
wait
```

Significant modifications have been implemented in the ***shifter*** image script to ensure compatibility with ***podman-hpc***