TOWARDS RESILIENT OPENSHEM

Delafrouz Mirfendereski1, Tony Curtis1 and Dr. Barbara Chapman1,2
Department of Computer Science, Institute for Advanced Computer Science
1 Exascale Programming Model Laboratory, Stony Brook University (https://you.stonybrook.edu/exascallab)
2 Brookhaven National Lab

Motivation

- Grown interest in parallel programming models.
- More diverse programming languages.
- More complex applications, hardware and architecture.
- Higher amount of components in systems.
- With more complexity comes great error rate.
- Failures cost considerable amount of money, time, and energy, and reduce the system’s performance and speed.
- Errors delay reaching the goal of Brookhaven remotely Programming Model Laboratory, Stony Brook communication and computation overlap.

Fault Tolerance

- Mean Time Between Failures (MTBF) is expected to reduce to couple of minutes in exascale systems.
- Failures differ in the origin, failures they may cause, range of effects, life span, and visibility to other system parts.
- Technologies used for resilience include:
  - Avoidance, detection, containment, recovery, diagnosis, and repair.
  - Base system, or application level.
  - Three important stages of fault tolerance:
    - Detection.
    - Knowledge propagation.
    - Recovery.

Resilient OpenSHMEM

- The most used recovery plan is checkpoint/restart.
- Checkpoint/restart techniques differ in how and where to store data, what portion of data to preserve, being coordinated or not, global or partial recovery, etc.
- Some checkpoint/restart methods are:
  - Global checkpoint/restart (g-CPR).
  - Coordinated local checkpoint/restart (l-CPR).
  - Uncoordinated distributed checkpointing (uncoordinated l-CPR).
  - Scalable multi-level checkpoint/restart (SCK).
  - Hierarchical checkpointing (H-CPR).

- Requesting PE would send a checkpoint carrier to each spare PE’s checkpointing queue.
- Spare PEs would update the stored data by reading the checkpoint carriers.
- In case of PEs crashing, survivors would rollback to their local last checkpoint. Some spare PEs would replace the dead ones.

- Added API includes:
  - `shmem_cpr_init()`
  - `shmem_cpr_reserve()`
  - `shmem_cpr_checkpoint()`
  - `shmem_cpr_restore()`
  - `shmem_cpr_finalize()`

- We have implemented a checkpoint/restart:
  - Local operation.
  - Global recovery.
  - Some PEs are separated in the beginning as spares.
  - In case one or some PEs fail, spare ones (if any) will replace them.
  - There are 4 exclusive PE types:
    - Original PEs.
    - Dead PEs.
    - Spare PEs.
    - Substitute PEs.
  - Original or substitute PEs can checkpoint any parts of their private or remotely accessible memories.
  - Checkpoints are stored locally and remotely.
  - Reservation is needed before checkpointing.
  - A request for reservation is made through sending a carrier to a spare PE’s reservation queue.
  - Spare PEs would allocate the necessary memory.
  - If previously declared, a piece of memory can be checkpointed through checkpointing carriers.

Introduction to OpenSHMEM

- Partitioned Global Address Space (PGAS) Parallel Programming Library.
- SPMD (Single Program Multiple Data) programming model.
- Provides global view of memory in a distributed programming environment.
- One sided communication compared to traditional two sided communication (e.g. MPI).
- Work is divided into multiple processes known as processing elements (PEs).
- OpenSHMEM routines support remote data transfer, work-shared broadcast and reduction, synchronization, and atomic memory operations across PEs.
- For communication OpenSHMEM uses RDMA (Remote Direct Memory Access) one-sided putget instead of matched pairs (e.g. MPI send/rece).
- Less communication overhead.
- More communication and computation overlap.
- OpenSHMEM provides specification for a standardized API and an execution model.
- Exascale Programming Models Laboratory at Stony Brook University is responsible for the reference implementation.
- Many implementations available for different hardware platforms.

More About Our Project

- Cross-Layer Application-Aware Resilience at Extreme Scale (CARES) project.
- Collaborating with the University of Tennessee Knoxville and Rutgers University.
- Providing the basic resilience support for OpenSHMEM and MPI.
- Leveraging User Level Failure Mitigation (ULFM) and other resilience libraries.
- The goals of this project are:
  - Developing application-aware and collaborative resilience frameworks.
  - Studying the types and costs of failures in MPI and OpenSHMEM.
  - Providing failure detection and knowledge propagation techniques at process management layer.
  - Enhancing the resilience techniques using compilers.
  - Using a compiler to analyze the computation and communication patterns.
  - Suggesting automated resilience techniques.

Similar Works

- User Level Failure Mitigation (ULFM) is an effort to handle process and node failure and knowledge propagation for MPI.
- Fenix is an application-level checkpointing library built upon ULFM for MPI.
- GASPI is a parallel programming PGAS library with fault tolerant API.
- GVR is a parallel programming library with multi-version checkpointing capability.

Future Work

- Test checkpointing system on applications.
- Suggestions on making OpenSHMEM’s API more resilient.
- The implemented checkpointing for OpenSHMEM can be improved or customized in various ways:
  - The number of copies of checkpoints.
  - Versioned checkpoints to overcome latent errors.
  - Checkpoints can be stored on disk.
  - Message logging to help with uncoordinated and therefore faster recoveries.
- Compilers may detect changes to memory and suggest places to insert checkpoints, or avoid redundant checkpoints.

Sponsor

Collaborators

Interested? Contact
Tony Curtis (Project Lead)
Exascale Programming Models Laboratory
Email: anthony.curtis@stonybrook.edu