Fault Tolerant Runtime Research @ ANL

Wesley Bland
Joint Lab for Petascale Computing Workshop
November 26, 2013
Brief History of FT

- Checkpoint/Restart (C/R) has been around for quite a while
  - Guards against process failure
  - Won’t talk too much about this
- Algorithm Based Fault Tolerance (ABFT)
  - Spawned a new class of “naturally fault-tolerant” applications (linear algebra, Monte Carlo, etc.)
  - Requires libraries to be fault tolerant as well
    - FT-MPI
    - MPI/RT
- Soft Errors cause silent incorrect answers (cosmic rays, overheating, etc.)
  - Sometimes they’re caught by ECC checking, other times not.
  - Don’t cause process failures, but may require handling anyway
- Some FT methods are trying to handle everything at once
  - Containment domains
Overview

- Process Fault Tolerance Work
  - MPI-3
  - MPI-<next>
- Soft Errors
  - GVR / LRDS
  - VOCL-FT
- Future Work
Process Fault Tolerance
Pre-MPI-3

- Process failure -> Application Failure
- Few implementations support MPI_Errhandlers
  - Previous Work: Checkpoint-on-Failure

STOP
Process Fault Tolerance
MPI-3

- Still no explicit FT
- New non-collective communicator creation
  - MPI_COMM_CREATE_GROUP
  - Provides a functionally complete FT model
  - Expensive to use
Proof of concept for FT in MPI 3.0

- Virtualize MPI objects
- Transparently map "virtual" user objects to "physical" objects
MPIXFT

- Add transparent metadata to MPI objects and cache when used
- Because (almost) everything has a NB equivalent, we can do notification
- Communicators are rebuilt via MPI_COMM_CREATE_GROUP

Wesley Bland, Argonne National Laboratory
**MPIXFT Early Results**

- **MCCK Mini-app**
  - Domain decomposition communication kernel
  - Each process has 4 outstanding requests at a time
- **Up to 256 nodes**

**MCCK Mini-app**

- Non-blocking ring test
  - More tokens taxes request caching system
  - Extreme case with lots of outstanding requests
- **10 nodes**

Wesley Bland, Argonne National Laboratory
Process Fault Tolerance
MPI-<next> (User Level Failure Mitigation)

- Enable application-level recovery by providing minimal FT API to prevent deadlock and enable recovery
- Don’t do recovery for the application, but let the application (or a library) do what is best.
- Only handling process failures currently
ULFM Overview

- Failure Notification
  - Error codes
  - New API for getting group of failed processes

- Failure Propagation
  - Local notification
  - New API for notifying other processes

- Failure Recovery
  - Point-to-point
    - Nothing required
  - Wildcard
    - New API to re-enable MPI_ANY_SOURCE
  - Communicator
    - New API to create communicator without failed processes
ULFM Mechanisms

- Minimal API
  - 5 main functions
- Encourages FT libraries to sit on top of MPI and provide high level recovery abstractions

Wesley Bland, Argonne National Laboratory
ULFM Continued

- RMA Windows & Files must be recreated after failures
- Minimal additions to encourage FT libraries rather than direct usage
  - Doesn’t restrict use cases
- Reference implementation complete
  - MPICH implementation in progress
- Standardization in progress
Soft Errors

GVR (Global View Resilience)

- Multi-versioned, distributed memory
  - Application commits “versions” which are stored by a backend
  - Versions are coordinated across entire system
- Different from C/R
  - Don’t roll back full application stack, just the specific data.
Soft Errors
LRDS (Local Reliable Data Storage)

- Backend data store for GVR
- Provides versioning across all kinds of storage
  - In-memory
  - NVRAM
  - Disk
- Uses dirty-bit tracking to create deltas between versions

Wesley Bland, Argonne National Laboratory
Soft Errors

VOCL: Transparent Remote GPU Computing

- Transparent utilization of remote GPUs
- Efficient GPU resource management:
  - Migration (GPU / server)
  - Power Management: pVOCL

Traditional Model

- Compute Node
- Application
- OpenCL API
- Native OpenCL Library
- Physical GPU

VOCL Model

- Compute Node
- Application
- OpenCL API
- VOCL Library
- Virtual GPU

Wesley Bland, Argonne National Laboratory
Efficient Virtualization of Accelerators with VOCL

Performance Overhead of Remote GPU Acceleration

- native OpenCL (local)
- VOCL (remote)
- Percentage of slowdown

Migration Overhead

- Without migration
- With migration
- Overhead caused by migration

Power Usage for Various Node Configurations Using Two GPUs

- h/w-pm
- h/w-s/w static pm
- pVOCL
- improv. vs. h/w-s/w static-pm

Energy usage (Kilo Joules)

- Energy efficiency improvement by pVOCL

- Power budget (Watts)

• P Lama, Y Li, AM Aji, P Balaji, JS Dinan, S Xiao, Y Zhang, W Feng, RS Thakur, and X Zhou. “pVOCL: power-aware dynamic placement and migration in virtualized GPU environments”. In ICDCS 2013.

• S Xiao, P Balaji, JS Dinan, Q Zhu, RS Thakur, S Coghlan, H Lin, G Wen, J Hong, and W Feng. “Transparent accelerator migration in a virtualized GPU environment”. In CCGrid 2012.

• S Xiao, P Balaji, Q Zhu, RS Thakur, S Coghlan, H Lin, G Wen, JH Hong, and W Feng, “VOCL: an optimized environment for transparent virtualization of graphics processing units”. In InPar 2012.
Soft Errors

VOCL-FT (Fault Tolerant Virtual OpenCL)

Synchronous Detecting Model

User App.
- bufferWrite
- launchKernel
- bufferRead
- sync

VOCL FT Functionality
- ECC Query
- Checkpointing

Asynchronous Detecting Model

User App.
- bufferWrite
- launchKernel
- bufferRead
- sync

VOCL FT Thread
- ECC Query
- ECC Query
- ECC Query
- ECC Query
- ECC Query

Double- (uncorrected) and single-bit (corrected) error counters may be queried in both models

Minimum overhead, but double-bit errors will trash whole executions

Wesley Bland, Argonne National Laboratory
VOCL-FT: Single and Double Bit Error Detection Overhead

**Matrix Transpose**
- **vocl**: Blue bars
- **async**: Yellow bars
- **sync**: Red bars

- **#threads**:
  - 512*512
  - 1024*1024
  - 2048*2048
  - 4096*4096

- **Execution Time (ms)**

**Smith-Waterman**
- **vocl**: Blue bars
- **async**: Yellow bars
- **sync**: Red bars

- **query size**:
  - 1K
  - 2K
  - 3K
  - 4K
  - 5K
  - 6K

- **Execution Time (ms)**

**Nbody**
- **vocl**: Blue bars
- **async**: Yellow bars
- **sync**: Red bars

- **#of bodies**:
  - 15360
  - 23040
  - 30720
  - 38400

- **Execution Time (ms)**

**DGEMM**
- **vocl**: Blue bars
- **async**: Yellow bars
- **sync**: Red bars

- **matrix block**:
  - 512*512
  - 1024*1024
  - 2048*2048
  - 4096*4096

- **Execution Time (ms)**
Future Work
Performance Faults

- What is the effect of a non-catastrophic error on performance?
  - Link failure
  - Corrected memory failures

- Is it better to mask these faults or perform a recovery action?