

# Fault Tolerance Techniques for High-End Computing Systems

Darius Buntinas

Mathematics and Computer Science Division

Argonne National Laboratory

# Hardware Resilience

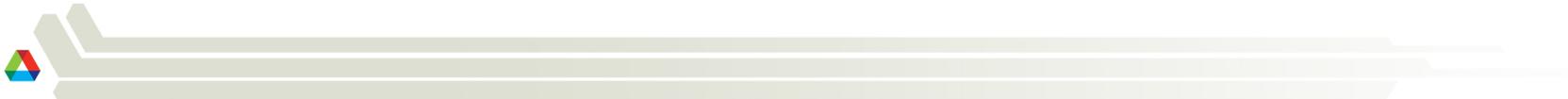
- Processor
  - Add ECC to cache and registers
  - Parity checks to ALUs
  - Reissue instructions on error
  - Redundant execution
- Memory
  - Parity and ECC
- Network
  - CRC
    - End-to-end or per-hop
    - Memory-to-memory
  - Automatic Path Migration
- Storage
  - RAID





# Software Resilience

- System-level checkpointing
- Application-based fault-tolerance
- Fault-tolerance in MPI



# System-Level Checkpointing

- Transparent to application
- Checkpointer takes snapshot of process state
  - OS-based or user-level
- Communication library must ensure global state is consistent
  - Flush channels or log messages
- For scalability we need to reduce size of rollback set
  - Detect clustering in application communication
  - Use coordinated checkpointing within cluster
  - Message logging between clusters



# Application-Based Fault-Tolerance

- Store and compute data redundantly
  - $D_1 + D_2 + \dots + D_n = E$  (n processes have  $D_1 \dots D_n$  + redundant process has E)
- If process i fails, recompute lost data
  - $D_i = E - (D_1 + \dots + D_{i-1} + D_{i+1} + \dots + D_n)$
  - But that requires stopping to collect all of the data
- What if we continued by substituting E for  $D_i$ 
  - Compensated for after computation



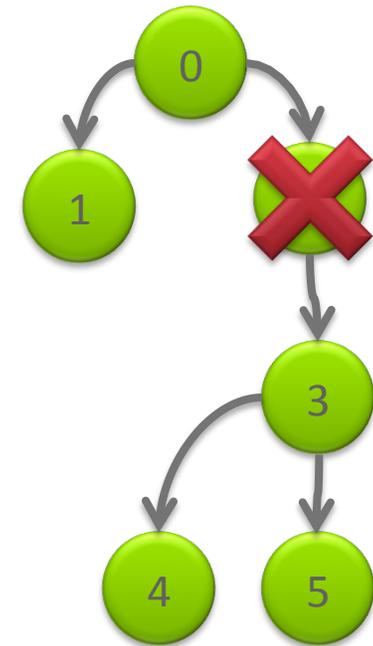
# Fault-Tolerance in MPI

- Consider only failed *processes*
  - Not things like network partitioning
- Issues to consider after a process fails
  - Collectives
    - Must not hang as a result of a failure
    - How to perform a collective with failed processes
  - Wildcard receives: MPI\_ANY\_SOURCE
    - What if the failed process was the one from which you're expecting the message?
  - RMA
    - Ops need to be low latency
  - I/O
    - What is the state of the file when a process dies?
  - Libraries
    - Is it safe to call into a library after a failure?
    - If some (live) process doesn't or can't call into library, you might deadlock
  - Replacing a failed process



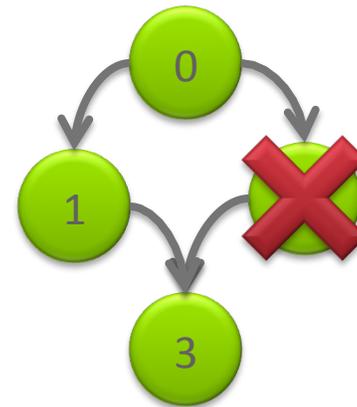
# Collectives - Can't hang after failure

- Immediate neighbors of failed process can detect a communication failure
  - Other processes may detect that a proc has failed, but may not know that the proc was part of the collective
- E.g., MPI\_Bcast
  - If Process 2 fails, and 3 returns with an error
    - Procs 4 and 5 will hang waiting for messages from 3
- A solution
  - Continue communication pattern, but indicate that failure occurred upstream
    - How to indicate a failure in a message?



# Collectives - Repair after failure

- After a failure, communication pattern may need to be changed
- Needs to be done in a coordinated manner
  - What if Proc 1 changes pattern but not 3?
- Use distributed agreement to make sure all processes agree on the same set of live processes



## Collectives in a Loop

- Can result in a hang
  - Not every process will detect the failure
- Option 1: Keep going through the loop after an error
  - Waste of cycles
  - What data do you send?
- Option 2: Check whether everyone succeeded after every bcast
  - Use `MPI_Comm_agreement()`
  - High overhead for FT agreement
  - Check every X iterations? Still adds overhead in error-free case

```
for (...) {  
    err = MPI_Bcast(...)  
    if (err) break;  
}
```



## Revoking a Communicator

- Option 3: “Kill” the communicator after detecting a failure

- MPI\_Comm\_revoke() is not collective
- (Eventually) any operation by any process on that comm results in an error

- Well, almost any operation

- MPI\_Comm\_shrink(oldcomm, newcomm)

- Creates a new communicator from the old one omitting failed procs

```
for (...) {  
    err = MPI_Bcast(...)  
    if (err) {  
        MPI_Comm_revoke();  
        break;  
    }  
}
```



# MPI\_ANY\_SOURCE Receives

- Example
  - Multiple server processes and multiple client processes
  - Servers do blocking ANY\_SOURCE receives waiting for request from any client
  - If every client fails, all servers could hang in blocking receive
- Possible solutions
  - Cancel every ANY\_SOURCE when a process fails
    - Processes will need to repost receives
    - May change order of posted ANY\_SOURCE and “named” receives
  - Cancel only blocking AS receives and return from MPI\_Wait\* calls but leave nonblocking AS receives as “pending”
    - User can cancel nonblocking AS receives if appropriate
    - But now we treat blocking differently from nonblocking
- Race condition
  - (1) Check that clients are alive; (2) client fails; (3) post ANY\_SOURCE receive
  - Need a mechanism to “acknowledge” failure



# RMA

- RMA operations need to be low latency
- Failure detection/reporting should be handled at the end of an epoch
  - Not during puts, gets, etc.
- Locks may need to be restored after failure
- What is the state of a window after a failure?
  - Can we be sure that data not targeted by failed processes is uncorrupted?



# MPI-I/O

- Collective file operations
  - Similar issues as collective communications
  - How much of the operation completed before the failure?
    - Operations are not atomic
- What can be known about the state of the file after a failure?
  - Currently, after a failure, the state of the file pointer is undefined



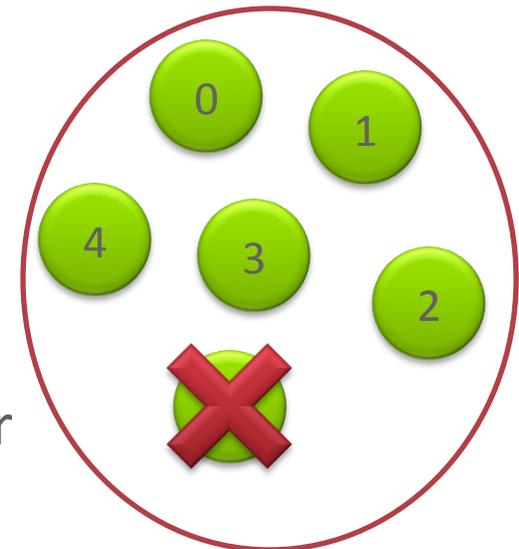
# Process Failures and Libraries

- If a process detects a failure, is it safe to jump out of the normal flow of execution?
  - E.g., call `MPI_Comm_revoke()` and break the loop
- What if that loop calls other libraries?
  - What if those libraries call collectives?
  - Can't revoke another library's communicator
- Should we call `MPI_Comm_agreement()` before every library call?



# Respawn Failed Process

- E.g., process 3 fails; spawn a new process, and give it rank of 3
  - Not restarting from a checkpoint
  - New process starts by calling `MPI_Init()`
- This works great for `MPI_COMM_WORLD`
  - But failed process was probably part of other communicators
- How to tell the process it was part of other communicators?
- How to distinguish one communicator from another?
  - Communicators are initially distinguished by the order in which they were created
  - Libraries might have created their own communicators



# Writing Fault Tolerant Applications

- How should people write FT applications?
  - ABFT
  - Master-worker
  - Ad-hoc
- How can we compose FT libraries?
  - BSP-style: Periodically sync and check for failures
    - Requires synchronization
    - Processes must continue to execute until sync point
  - Is there a missing feature?
    - No (very low) non-fault overhead
    - Easy for app writers to understand and use
    - Like C++ exceptions
      - Don't need to check for error return from every function



# Questions

