Support for Fault Tolerance (Dynamic Process Control)

Rich Graham
Oak Ridge National Laboratory

- Problem : Some component affecting a running MPI job fails (H/W, S/W)
- Issue: How does one deal with this within the context of a running MPI job ? (not implementation questions)
  - Does the job have to abort ?
  - If not, can the job continue to communicate ?
  - Can there be a change in resources available to the job

Some Failure Modes

- Network
- Process
- File system
- H/W vs. S/W (does MPI even “care” about this aspect ?)
- Permanent vs. Transient

Failures that should be considered at the API level

- Process
- Network failures can be dealt with w/o changing the standard
- File System ?

Approches that have been tried

- Checkpoint restart (CPR)
- CPR + message logging
- Application notification, with getting the MPI communications to a well defined state
- Master/Slave type of algorithms, discarding failed Slave communicators

- What, if any of these, need help from the standard ?

Potential Areas Affected in the Standard

- Communications
- Communicators (==> collective optimizations)
- Dynamic Process Control

Important to Consider

- What are the “common cases” the standard aims to support well ?
- What are potential impacts of the proposed changes ?
  - Collective communications
  - Point-to-point communications
- What additional synchronizations may be imposed ?
- What additional work may need to be done in the critiacal path of an implementation ?
- Can we live w/o this, and still expect to have the standard used ?

Basic Principals

- MPI does NOT provide fault-tolerance
- MPI enables (more than makes possible) possible to implement fault-tolerant algorithms
- From a Checkpoint/Restart point of view, MPI is just one more program, that happens to be parallel

Proposal to handle Process Failure

- Have the library notify the application of process failure
- Have the application decide how to proceed
  - Abort
  - Continue w/o restoring a new process
  - Restore as many failed process as possible
- Have the library reset communications to a well defined state, if not aborting

Implications

- Communicators may change in the middle of a job
- Synchronization is needed, in some cases, to achieve consistent MPI internal state
- Internal library state information may need to change to reflect the new communicator
  - Communications addressing
  - Collective communications patterns
  - ?

Discussion

- Is support for fault tolerance in MPI sufficient

Discussion

- Is support for fault tolerance in MPI sufficient
- If so, Should MPI be an enabler of fault-tolerant algorithms, or provide these algorithms ?

Discussion

- Is support for fault tolerance in MPI sufficient
- If so, Should MPI be an enabler of fault-tolerant algorithms, or provide these algorithms ?
- If so, What additional support is needed ?

Discussion

- Is support for fault tolerance in MPI sufficient
- If so, Should MPI be an enabler of fault-tolerant algorithms, or provide these algorithms ?
- If so, What additional support is needed ?
- What else ?

Next steps

- Is there sufficient interest in proceeding ?
- Setup weekly telecon to continue discussing these issues

Next steps

Assuming this is a go,

1. Support for FT, or provide FT algorithms ?
2. What sort of FT are we talking about
3. Specific use-case scenarios
4. How would this translate into API additions ?
5. What requirements do these additions impose ?
6. What are the effects of these additions ?
7. Can we live with these effects ?
8. API changes proposed
9. API prototyped
10. Assess the prototype/s
11. Decision with respect to inclusion in the standard