ULFM Update
(March to June 2014)

FTWG@MPI Forum meeting
Chicago, June 2014
Application Recovery Patterns

User Level Failure Mitigation: a set of MPI interface extensions to enable MPI programs to restore MPI communication capabilities disabled by failures

Coordinated Checkpoint/Restart, Automatic, Compiler Assisted, User-driven Checkpointing, etc.
In-place restart (i.e., without disposing of non-failed processes) accelerates recovery, permits in-memory checkpoint

Naturally Fault Tolerant Applications, Master-Worker, Domain Decomposition, etc.
Application continues a simple communication pattern, ignoring failures

Uncoordinated Checkpoint/Restart, Transactional FT, Migration, Replication, etc.
ULFM makes these approaches portable across MPI implementations

ULFM allows for the deployment of ultra-scalable, algorithm specific FT techniques.
Minimal Feature Set for FT

• Failure Notification (MPI exceptions)
• Error Propagation (Revoke)
• Error Recovery (Shrink, Agree, Spawn)

Not all recovery strategies require all of these features, that’s why the interface splits notification, propagation and recovery
Modified

- Ticket 0 (Bill’s comments)
- Finalize completes (removed “successfully”, due to attributes destructors)
- RMA rewrite
  - Relaxed memory consistency after failure (the entire window exposed memory may become undefined)
  - Relaxed error raising requirement (previous text overspecified our intention, now only synchronization function must raise exceptions)
  - Added advice on win_free (after raising PF, it becomes non-synchronizing, users should be careful).
  - Added advice to implementors “please, do not continue to deliver RMA operations from dead processes after win_free”
- Agree is now a bitwise AND (on integer)
- Examples use error classes and codes properly
Considered, but discarded

- **MPI_Comm_ishrink**
  - Performance benefit unclear at this point
  - Postponed until proven to serve a purpose (that is a better implementation than doing it all in wait is possible)

- **MPI_Win_free synchronizes even with failures**
  - Considered too costly
  - There is a way out for users, it can then be deployed only when FT is necessary
Coming next

• Upgrade from F77 to F08 interfaces
• Query of FT support
  • Predefined attribute on MPI_COMM_WORLD or info key in MPI_INFO_ENV
  • Alternative: using MPI_Init_with_info (future ticket from Hybrid group)
• Query status of revoked handles
  • MPI_Comm_is_revoked(comm)
• MPIT keys and better interaction with tools
Implementation progress

• Open MPI implementation
  • Failure free performance is satisfactory
  • Poor agreement algorithm in the current implementation results in poor post-failure performance
  • Work ongoing to provide better Agreement (ERA early august)

• MPICH implementation also well advanced
Some more applications

• Large number of papers at EuroMPI about ULFM (5+ submissions)

• IPDPS: ANU presented a sparse PDE code (deployed in GENE application)
Getting more info

• Ticket wiki with standard text, links
  • https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/323

• Users and applications using ULFM
  • http://fault-tolerance.org/2014/05/27/anu-presents-pde-solver-with-ulfm-at-ipdps/20140527-ulfm-users/

• Example codes and modular snippets
  • http://fault-tolerance.org/2014/02/04/slides-with-ulfm-examples/ulfm-mpi-dec13forum/

• Implementations
  • Open MPI: https://bitbucket.org/icldistcomp/ulfm
  • MPICH

• Publications
Implementation in Open MPI

• It works! Performance is good!

Sequoia AMG is an unstructured physics mesh application with a complex communication pattern that employs both point-to-point and collective operations. Its failure free performance is unchanged whether it is deployed with ULFM or normal Open MPI.

The failure of rank 3 is detected and managed by rank 2 during the 512 bytes message test. The connectivity and bandwidth between rank 0 and rank 1 are unaffected by failure handling activities at rank 2.

Sequoia AMG Performance with Fault Tolerance

ULFM Fault Tolerant MPI Performance with failures
IMB Ping-pong between ranks 0 and 1 (IB20G)

Thanks for CREST, Riken support
More performance: synthetic benchmarks

1-byte Latency (microseconds) (cache hot)

<table>
<thead>
<tr>
<th>Interconnect</th>
<th>Vanilla</th>
<th>Std. Dev.</th>
<th>Enabled</th>
<th>Std. Dev.</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Memory</td>
<td>0.8008</td>
<td>0.0093</td>
<td>0.8016</td>
<td>0.0161</td>
<td>0.0008</td>
</tr>
<tr>
<td>TCP</td>
<td>10.2564</td>
<td>0.0946</td>
<td>10.2776</td>
<td>0.1065</td>
<td>0.0212</td>
</tr>
<tr>
<td>OpenIB</td>
<td>4.9637</td>
<td>0.0018</td>
<td>4.9650</td>
<td>0.0022</td>
<td>0.0013</td>
</tr>
</tbody>
</table>

Bandwidth (Mbps) (cache hot)

<table>
<thead>
<tr>
<th>Interconnect</th>
<th>Vanilla</th>
<th>Std. Dev.</th>
<th>Enabled</th>
<th>Std. Dev.</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Memory</td>
<td>10,625.92</td>
<td>23.46</td>
<td>10,602.68</td>
<td>30.73</td>
<td>-23.24</td>
</tr>
<tr>
<td>TCP</td>
<td>6,311.38</td>
<td>14.42</td>
<td>6,302.75</td>
<td>10.72</td>
<td>-8.63</td>
</tr>
<tr>
<td>OpenIB</td>
<td>9,688.85</td>
<td>3.29</td>
<td>9,689.13</td>
<td>3.77</td>
<td>0.28</td>
</tr>
</tbody>
</table>

Collective communications: 48 core shared memory (very stressful)
Performance difference is less than std-deviation
Failure Notification

• Notification of failures is **local only**
  • New error MPI_ERR_PROC_FAILED Raised when a communication with a **targeted** process fails

• In an operation (collective), **some process may succeed** while **other raise an error**
  • Bcast might succeed for the top of the tree, but fail for some subtree rooted on a failed process

• **ANY_SOURCE** must raise an exception
  • the dead could be the expected sender
  • Raise error MPI_ERR_PROC_FAILED_PENDING, preserve matching order
  • The application can complete the recv later (MPI_COMM_FAILURE_ACK())

• Exceptions indicate an operation failed
  • To know what process failed, apps call MPI_COMM_FAILURE_ACK(), MPI_COMM_FAILURE_GET_ACKED()
App using notification only

- Error notifications do not break MPI
  - App can continue to communicate on the communicator
  - More errors may be raised if the op cannot complete (typically, most collective ops are expected to fail), but p2p between non-failed processes works

- In this Master-Worker example, we can continue w/o recovery!
  - Master sees a worker failed
  - Resubmit the lost work unit onto another worker
  - Quietly continue
App using propagation only

- Application does only p2p communications
- P1 fails, P2 raises an error and wants to change comm pattern to do application recovery
- but P3..Pn are stuck in their posted recv
- P2 unlocks them with Revoke
- P3..Pn join P2 in the new recovery p2p communication pattern
Error Recovery

- Restores full communication capability (all collective ops, etc).
- MPI_COMM_SHRINK(comm, newcomm)
  - Creates a new communicator excluding failed processes
  - New failures are absorbed during the operation
  - The communicator can be restored to full size with MPI_COMM_SPAWN
Error Agreement

• When in need to decide if there is a failure and if the condition is recoverable (collectively)
  • MPI_COMM_AGREE(comm, flag)
    • Fault tolerant agreement over boolean flag
    • Unexpected failures (not acknowledged before the call) raise MPI_ERR_PROC_FAILED
    • The flag can be used to compute a user condition, even when there are failures in comm

• Can be used as a global failure detector
4 Two-dimension PDE Solver: Recovery Methods

- replication/re-sampling:
  recover grids 0–3 from duplicate grids 7–10;
  recover grids 4–6 via resampling from grid 0–3

- alternate combination:
  lost grid $g \in \{0..6\}$ is ignored; final result (sparse grid) is constructed via a subset of \{0..6, 11..13\} – \{g\}
5 Recovery Methods: Alternate Combination Formula

- uses extra set of smaller sub-grids on a 3rd (next lower) diagonal (modest amount of extra overhead)
- for a single failure on a fine sub-grid, can find a new combination with an inclusion/exclusion principle avoiding the failed sub-grid
- also works for many (but not all) cases of multiple failures

- if the failure is on 2nd diagonal, can similarly use a 4th (lower) diagonal to avoid this
Fault Recovery Procedure: Detect Failed Process

- can detect failed processes as follows:
  - attach an error handler ensuring failures get acknowledged on (original) communicator `comm`
  - call `MPI_Barrier(comm); if fails:`
  - revoke it via `MPI_Comm_revoke(comm)`
  - and create shrunken communicator via `OMPI_Comm_shrink(comm, &scomm)`
  - use `MPI_Group_difference(...) &fg)` to make a globally consistent list of failed processes

A communicator with global size 7

```
0 1 2 3 4 5 6
```

Process 3 and 5 on parent fail

```
0 1 2 x 4 x 6
```

Shrink the communicator and spawn failed processes as child with rank 0 and 1

```
0 1 2 4 6 1 1
```

Use intercommunicator merge to assign the two highest ranks to the newly created processes on child part

```
0 1 2 3 4 5 6
```

Sending failed ranks from parent to the two highest ranks on child and split the communicator with the same color to assign rank 3 and 5 to the child processes to order the ranks as it was before the failure

```
0 1 2 4 6 3 5
```

Changing child to parent

```
0 1 2 4 6 3 5
```
Results: Scalability

- Results on OPL cluster, max. resolution of $2^{13}$
- In terms of absolute time, CR is always more longer (however, uses fewer processes)
- RC and AC also show best scalability
- Plots for 2 failures erratic due to high overheads in $\beta$ version of ULFM MPI

RC = Replication/resampling
AC = Alternate recombination
CR = Checkpoint/Restart

OPL cluster node: 2x6 cores Xeon5670, QDR IB
13 Fault Recovery of a Real Application - GENE

- GENE: Gyrokinetic Electromagnetic Numerical Experiment
  - plasma microturbulence code
  - multidimensional solver of Vlasov equation
  - fixed grid in five-dimensional phase space \((x_\parallel, x_\perp, x_r, v_\parallel, v_\perp)\)
- computes gyroradius-scale fluctuations and transport coefficients
- these fields are the main output of GENE
- hybrid MPI/OpenMP parallelization – high scalability to 2K cores

<table>
<thead>
<tr>
<th>cores</th>
<th>(t_g)</th>
<th>(t_c)</th>
<th>(\Delta t_f)</th>
<th>(t_G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>48.9</td>
<td>3.4</td>
<td>1.0</td>
<td>107.6</td>
</tr>
<tr>
<td>98</td>
<td>36.8</td>
<td>3.8</td>
<td>7.4</td>
<td>65.3</td>
</tr>
<tr>
<td>196</td>
<td>63.2</td>
<td>11.5</td>
<td>19.9</td>
<td>98.7</td>
</tr>
</tbody>
</table>

times: 
- \(t_g\) for GENE instance
- \(t_c\) for comb. alg.
- \(\Delta t_f\) extra for one failure
- \(t_G\) for full-grid GENE instance