ULFM Process Fault Tolerance reading

Aurelien Bouteiller
FT WG

MPI Forum, June 2016
Bellevue, WA
Info, resources, participate

• Issue Ticket (w/ links to PRs)
  • https://github.com/mpi-forum/mpi-issues/issues/20

• Implementation available
  • Version 1.1 based on Open MPI 1.6 released early November 2015
    https://bitbucket.org/icldistcomp/ulfm
  • Full communicator-based (point-to-point and all flavors of collectives) support
  • Network support IB, uGNI, TCP, SM
  • Runs with ALPS, PBS, etc...
  • RMA, I/O in progress

http://fault-tolerance.org/
Use cases: Fenix+S3D

- Fenix is a framework to provide scoped user-level checkpoint/restart
  - Provides some of the same services provided by the “MPI_Reinit” idea floated around by T. Gamblin and I. Laguna
  - Recover failed processes with revoke-shrink-spawn-reorder sequence
  - Recovered and surviving processes jump back to the start (longjump in Fenix_init)
  - Fenix has helpers to perform user directed “in-memory” or “buddy” checkpointing (and reload)
  - Injection of FT layer: PMPI based

- **Fenix_Checkpoint_Allocate** mark a memory segment (baseptr,size) as part of the checkpoint.

- **Fenix_Init** Initialize Fenix, and restart point after a recovery, status contains info about the restart mode

- **Fenix_Comm_Add** can be used to notify Fenix about the creation of user communicators

- **Fenix_Checkpoint** performs a checkpoint of marked segments

```c
allocate(yspc(nx,ny,nz,nslvs))
allocate(other_arrays)
call MPI_Init()
[...] ! Initialize non-conflicting modules
call Fenix_Checkpoint_Allocate(C_LOC(yspc),
    sizeof(yspc),ckpt_yspc)
call Fenix_Init(Fenix_Neighbors,PEER_NODE_SIZE,
    Fenix_resume_to_init, status, C_LOC(world))
if(status.eq.Fenix_st_survivor) then
    [...] ! Finalize conflicting modules
endif
[...] ! Initialize conflicting modules
if(status.eq.Fenix_st_new) 
call initialize_yspc()
endif
do ! Main loop
    [...] ! Iterate and update yspc array
    if(mod(step-1,CHECKPOINT_PERIOD).eq.0) then
        call Fenix_Checkpoint(ckpt_yspc);
    endif
endo
call Fenix_Finalize()
call MPI_Finalize()
```
Use cases: Fenix+S3D

- S3D is a production, highly parallel method-of-lines solver for PDEs
  - used to perform first-principles-based direct numerical simulations of turbulent combustion
- S3D rendered fault tolerant using Fenix/ULFM
- 35 lines of code modified in S3D in total!
- Order of magnitude performance improvement in failure scenarios
  - thanks to online recovery and in-memory checkpoint advantage over I/O based checkpointing
- Injection of FT layer: addition of a couple of Fenix calls

---

```
1 call MPI_Comm_split(gcomm, py+1000*pz, r, xcomm)
2 call MPI_Comm_split(gcomm, px+1000*pz, r, ycomm)
3 call MPI_Comm_split(gcomm, px+1000*py, r, zcomm)
4 call Fenix_Comm_Add(xcomm);
5 call Fenix_Comm_Add(ycomm);
6 call Fenix_Comm_Add(zcomm);
7 [...] 
8 call MPI_Comm_split(gcomm, xid, r, yz_comm)
9 call MPI_Comm_split(gcomm, yid, r, xz_comm)
10 call MPI_Comm_split(gcomm, mid, r, xy_comm)
11 call Fenix_Comm_Add(yz_comm);
12 call Fenix_Comm_Add(xz_comm);
13 call Fenix_Comm_Add(xy_comm);
```

S3D Code snippet to declare to Fenix the communicators to recover

---

Fig. 3. Checkpoint time for different core counts (8.6 MB/core). The numbers above each test show the aggregated bandwidth (the total checkpoint size over the average checkpoint time).
Use cases: Resilient X10

- X10 is a PGAS programming language
  - Legacy resilient X10 TCP based

```java
try{ /*Task A*/
    at (p) { /*Task B*/
        finish { at (q) async { /*Task C*/ } }
    }
} catch(dpe:DeadPlaceException){ /*recovery steps*/}

```

- **MPI operations in resilient X10 runtime**
  - Progress loop does MPI_Iprobe, post needed recv according to probes
  - Asynchronous background collective operations (on multiple different comm's to form 2d grids, etc).

- Recovery
  - Upon failure, all communicators recreated (from shrinking a large communicator with spares, or using MPI_COMM_SPAWN to get new ones)
  - Ranks reassigned identically to rebuild the same X10 “teams”

- **Injection of FT layer**
  - Unnecessary, x10 has a runtime that hides all MPI from the application and handles failures internally

- The performance improvement due to using ULFM v1.0 for running the LULESH proxy application [3] (a shock hydrodynamics stencil based simulation) running on 64 processes on 16 nodes with

Source: Sara Hamouda, Benjamin Herta, Josh Milthorpe, David Grove, Olivier Tardieu. Resilient X10 over Fault Tolerant MPI. In: poster session SC’15, Austin, TX, 2015.
Use cases: CoArrays “failed images”
WIP to support Fortran TS 18508

if (num_images(failed=.true.) > 0 ) then
    form subteam(1, recover)
    sync all (stat=st)! Will return statFailed_image
    change team (recover)
        :! Execute as a subteam
    end team
end if

Additional Coarray Features in Fortran, John Reid (JKR Associates), 7th international conference on PGAS programming models, 2013

• Implementation effort in progress using ULFM
  • Failure detection/propagation is communicator-based (service communicator)
  • RMA based communications (win_revoke interrupted)
  • Team repair based on comm_shrink – team windows recreated from service communicator
Use cases: Monte-Carlo PDE solver

• ALSVID-UQ algorithm solving the 2-dimensional stochastic Euler equations of gas dynamics.
  • Multi-level Monte-carlo expressed as a telescopic sum

\[ E[X_{hL}] = E_{M0}[X_{h0}] + \sum_{\ell=1}^{L} E_{M\ell}[X_{h\ell} - X_{h\ell-1}] \]

• Communication pattern:
  • P2p Halo exchange between decomposed domains
  • Collective allreduce inside levels (between domains)
  • Collective aggregation between levels

• FT pattern:
  • Fine levels domain decomposed, with halo exchange between domains and in-memory checkpoints on neighbors processes, work redistributed after failure
  • Coarse domains replicated (failure ignored)
  • Failure of all processes holding a domain looses the results for that domain
  • Massive failure will degrade the solution

![Figure 2](image_url) The idea of MLMC is illustrated on the left and compared to the MC method on the right.

![Figure 4](image_url) Parallel distribution of work in FT-MLMC with improved failure resilience.

![Figure 5](image_url) Results of the FT-MLMC implementation for three different failure scenarios.

Stefan Pauli, Manuel Kohler, Peter Arbenz: A fault tolerant implementation of Multi-Level Monte Carlo methods. PARCO 2013: 471-480
Use cases: Hadoop over MPI

- Non-HPC workflow usually do not consider MPI because it lacks FT


- ULFM permits high performance exchange in non-HPC runtimes (like Hadoop)

Figure 2: The architecture of FT-MRMPI.

Figure 8: Normalized job completion time of failed and recovery run.

• Some applications can continue w/o recovery

• Some applications are maleable
  • Shrink creates a new, smaller communicator on which collectives work

• Some applications are *not* maleable
  • Spawn can recreate a “same size” communicator
  • It is easy to reorder the ranks according to the original ordering
  • Pre-made code snippets available
**Minimal Feature Set for a Resilient MPI**

1. Failure Notification
2. Error Propagation
3. Error Recovery

Not all recovery strategies require all of these features, that's why the interface splits notification, propagation and recovery*.

ULFM is not a recovery strategy, but a minimalistic set of building blocks for more complex recovery strategies.

*: some machines are stable, supporting post-failure semantic is optional
Notification integrating with existing error handling features

- Use existing error handlers
  - MPI_COMM_SET_ERRHANDLER
  - conveniently capture and manage the new survivable error codes

- New error codes to deal with failures
  - MPI_ERROR_PROC_FAILED: report that the operation discovered a newly dead process. Returned from all blocking function, and all completion functions.
  - MPI_ERROR_PROC_FAILED_PENDING: report that a non-blocking MPI_ANY_SOURCE potential sender has been discovered dead.
  - MPI_ERROR_REVOKED: a communicator has been declared improper for further communications. All future communications on this communicator will raise the same error code, with the exception of a handful of recovery functions

- Operations that **can’t complete** return ERR_PROC_FAILED
  - State of MPI objects unchanged (communicators, etc)
  - Repeating the same operation has the same outcome

- Operations that **can be completed** return MPI_SUCCESS
  - Pt-2-pt operations between non failed ranks can continue

Example: only rank4 should report the failure of rank 5
Summary of new functions

- **MPI_Comm_failure_ack** (comm)
  - Resumes matching for MPI_ANY_SOURCE

- **MPI_Comm_failure_get_acked** (comm, &group)
  - Returns to the user the group of processes acknowledged to have failed

- **MPI_Comm_revoke** (comm)
  - Non-collective, interrupts all operations on comm (future or active, at all ranks) by raising MPI_ERR_REVOKED

- **MPI_Comm_shrink** (comm, &newcomm)
  - Collective, creates a new communicator without failed processes (identical at all ranks)

- **MPI_Comm_agree** (comm, &mask)
  - Collective, agrees on the AND value on binary mask, ignoring failed processes (reliable AllReduce), and the return code
Resolving transitive dependencies

• 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 can unlock them with Revoke
  • P3..Pn join P2 in the recovery

```c
proc_failed_err_handler(MPI_Comm comm, int err, ...) {
  if(err == MPI_ERR_PROC_FAILED ||
      err == MPI_ERR_REVOKED) {
    if(err == MPI_ERR_PROC_FAILED) MPI_Comm_revoke(comm);
    recovery(comm);
  }
}

ft_transitive_deps(void) {
  for(i=0; i<nbrecv; i++) {
    if(myrank>0) MPI_Irecv(buff, count, datatype,
                          myrank-1, tag, comm, &req);
    if(myrank<n) MPI_Send(buff2, count, datatype,
                          myrank+1, tag, comm, &req);
  }
}```
Scalable Agreement/Shrink

- Novel Early Returning Agreement algorithm*
- Logarithmic topology & logarithmic computation: scalable
- 2x the Cray AllReduce latency at 6k processors!

Summary

• Issue Ticket (w/ links to PRs)
  • [https://github.com/mpi-forum/mpi-issues/issues/20](https://github.com/mpi-forum/mpi-issues/issues/20)

• Implementation available
  • It is actually fast, now.  [http://fault-tolerance.org/](http://fault-tolerance.org/)

• User base has grown quickly
  • Filling a need
  • outlined best practice
  • Varied use cases exercise all capabilities

• FAQ
  • *I don’t care, my machines are stable*
    • Fair enough, your implementation does not have to support FT (just provide stub interfaces so that FT programs compile and run w/o faults)
  • *I want to do only Checkpoint/Rerstart*
    • ULFM opens up faster, better C/R than before (that can use NVRAM effectively, etc)
  • *This is too complicated*
    • It doesn’t have to be: high level frameworks and code snippets for common tasks are available and help tremendously for quick prototyping
Scalable Resilient Constructs: Revoke

- BMG* Revoke propagation in less than 100µs
- First post-Revoke collective operation sustains some performance degradation resulting from the network jitter associated with the circulation of revoke tokens
- After the fifth Barrier (approximately 700µs), the Revoke reliable broadcast has completely terminated, therefore leaving the application free from observable jitter.

These works use ULFM

- ST PAULI, P. Arbenz et SCHWAB, Ch. Intrinsic fault tolerance of multi level Monte Carlo methods. ETH Zurich, Computer Science Department, Tech. Rep, 2012.
- ENGELMANN, Christian et NAUGHTON, Thomas. A NETWORK CONTENTION MODEL FOR THE EXTREME-SCALE SIMULATOR.

**Credits:** ETH Zurich

**Figure 5.** Results of the FT-MLMC implementation for three different failure scenarios.

The performance improvement due to using ULFM v1.0 for running the LULESH proxy application (3)
(a shock hydrodynamics stencil based simulation) running on 64 processes on 16 nodes with...