REFINING FORTRAN FAILED IMAGES

Nathan Weeks, Glenn Luecke, Gurpur Prabhu

Fifth International IEEE Workshop on Extreme Scale Programming Models and Middleware (ESPM2 2020)
November 11, 2020
PRESENTATION TOPICS

- Fortran 2018 features for fault-tolerant parallel programming
- Prototype implementation (“OpenCoarrays-ft”)
- Proposed changes to Fortran 2018 standard
MOTIVATION
For Tianhe-2, the MTBF is about 2h on 8192 nodes.

*Chen et al., Toward fault-tolerant hybrid programming over large-scale heterogeneous clusters via checkpointing/restart optimization* (2019)
On average [over 261 days], an application failure caused by a system-related issue [on Blue Waters] occurs every 15 min…

Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)
1.53% of applications failed due to system problems, these... account for about 9% of total production node hours.

Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)
...an increase of 20x in the application failure probability... when scaling XE applications from 10,000 to 22,000 nodes.

Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)
…failed applications that are not recovered through checkpoint/restart add potentially $421,878 to the Blue Waters energy bill…

Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)
Therefore, the impact of system errors on applications and costs of ownership is substantial and destined to grow for larger machines.

DI MARTINO ET AL., MEASURING THE RESILIENCY OF EXTREME-SCALE COMPUTING ENVIRONMENTS (2016)
STATE OF THE PRACTICE
STATE OF THE PRACTICE
COORDINATED CHECKPOINT/RESTART

• Processes periodically save state off node
• *Global rollback (backward) recovery* if a process fails:
  • All (non-failed) processes terminate
  • Newly-launched processes read state and resume execution
EXAMPLES OF OTHER RECOVERY MODELS

- Local non-shrinking backward recovery
  - Can use in-memory checkpoint from peer/buddy process
- Forward shrinking recovery
# EXASCALE COMPUTING PROJECT (ECP) 2017 SURVEY

<table>
<thead>
<tr>
<th>Applications</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use MPI</td>
<td>28</td>
</tr>
<tr>
<td>Call MPI from Fortran</td>
<td>13</td>
</tr>
</tbody>
</table>
MPI itself provides no mechanisms for handling processor failures.

MPI 3.1 (2015)

- MPI Extensions:
  - Reinit
  - User-Level Failure Mitigation (ULFM)
FORTTRAN 2018
FORTRAN 2018

FAILED IMAGES
FORTRAN PROCESSORS SUPPORTING FAILED IMAGES

- OpenCoarrays
  - partial support, to allow an application to detect & exit
- Intel Fortran 19.1
- NAG Fortran Compiler 7.0
  - Single node
INTRODUCING OPENCOARRAYS-FT

• Prototype extensions to OpenCoarrays
• Adds support for:
  ■ FORM TEAM
    ○ New teams excludes failed images
    ○ NEW_INDEX=
  ■ STAT= and ERRMSG= in:
    ○ FORM TEAM
    ○ CHANGE TEAM / END TEAM
    ○ SYNC TEAM
    ○ CRITICAL
  ■ ULFM2 (Open-MPI)
OVERVIEW OF FORTRAN 2018 FEATURES FOR FAULT-TOLERANT PARALLEL PROGRAMMING
<table>
<thead>
<tr>
<th>Fortran</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>process</td>
</tr>
<tr>
<td>image index</td>
<td>rank</td>
</tr>
<tr>
<td>team</td>
<td>communicator</td>
</tr>
</tbody>
</table>
An image can be in one of 3 states:

Failed
  Fail-stop/crash failure

Stopped
  • Reached end of program or STOP statement
  • Coarray data can still be accessed by active images

Active
  An image that has neither stopped nor failed
integer :: X[*]
...
X = X[1]
use intrinsic :: iso_forTRAN_env, only: STAT_FAILED_IMAGE
integer :: s, X[*]
...
X = X[1, STAT=s]
if (s == STAT_FAILED_IMAGE) then
  ... handle image failure ...

An *image control statement* “affects the execution ordering between images” ( synchORIZATION)
SYNC TEAM(STAT=s)
if (s == STAT_FAILED_IMAGE) then
    failed = FAILED_IMAGES()
    ... handle image failure ...

SYNC TEAM
! error termination of image failure detected
If the STAT= specifier appears in… a CHANGE TEAM, END TEAM, EVENT POST, FORM TEAM, SYNC ALL, SYNC IMAGES, or SYNC TEAM statement… [and] one of the images involved has failed… the intended action is performed on the active images involved and stat-variable is assigned the value STAT FAILED IMAGE…

If Fortran 2018 requires many image control statements to perform their intended action in the presence of failed images
If the STAT= specifier appears in… a CHANGE TEAM, END TEAM, EVENT-POST, FORM TEAM, SYNC ALL, SYNC IMAGES, or SYNC TEAM statement… [and] one of the images involved has failed… the intended action is performed on the active images involved and stat-variable is assigned the value STAT_FAILED_IMAGE…

Some of these image control statements involve synchronization among all images in a team.
// OpenCoarray-ft implementation
MPI_Comm team_comm;
...
int rc, flag = 1;
do {
  MPIX_Comm_failure_ack(team_comm);
  rc = MPIX_Comm_agree(team_comm, &flag);
} while (rc != MPI_SUCCESS);
MPIX_Comm_failure_get_acked(team_comm, &failed_group);
MPI_Group_size(failed_group, &num_failed_in_group);
if (num_failed_in_group > 0) {
  *stat = STAT_FAILED_IMAGE;
  ... translate ranks to **MPI_COMM_WORLD** (initial team)
  ... **and** add to MPI group of known process failures

See ULFM spec: “Fault-Tolerant Consistent Group of Failures Example (Agree variant)”
TEAM SYNCHRONIZATION WITH STAT=

MPI_COMM_FAILURE_ACK acknowledges process failures in `team_comm` detected by the caller.
TEAM SYNCHRONIZATION WITH STAT=

1. MPI_ALLREDUCE w/ MPI_BAND on flag (unused)
2. Synchronizes acknowledged failed processes
TEAM SYNCHRONIZATION WITH STAT=

```c
1 // OpenCoarray-ft implementation
2 MPI_Comm team_comm;
3 ...
4 int rc, flag = 1;
5 do {
6   MPIXComm_failure_ack(team_comm);
7   rc = MPIXComm_agree(team_comm, &flag);
8 } while (rc != MPI_SUCCESS);
9 MPIXComm_failure_get_acked(team_comm, &failed_group);
10 MPI_Group_size(failed_group, &num_failed_in_group);
11 if (num_failed_in_group > 0) {
12   *stat = STAT_FAILED_IMAGE;
13   ... translate ranks to MPI_COMM_WORLD (initial team)
14   ... and add to MPI group of known process failures
```
TEAM SYNCHRONIZATION WITH STAT=

- Propagates a consistent knowledge of failed images in team
  - FAILED_IMAGES( ) returns list of images (in current team) known by caller to have failed
  - Fortran 2018 requires only at least 1 failed image
Rationale:

1. OpenCoarrays-ft doesn’t reliably support detecting new image failures in coarray operations, e.g.
   \[ X = X[1, \text{STAT}=s] \]
   - ULFM2 lacks explicit support for detecting process failure in MPI one-sided operations

2. Consistent knowledge of image failure can aid recovery
   - And not that costly, as we’ll see later...
COLLECTIVE SUBROUTINES

defn:
intrinsic subroutine that performs a calculation on a team of images *without requiring synchronization*

- CO_BROADCAST, CO_MAX, CO_MIN, CO_REDUCE, CO_SUM
COLECTIVE SUBROUTINES

```call co_sum(A, STAT=s)`
```

- $s \equiv \text{STAT\_FAILED\_IMAGE}$:
  - May be true for subset of images in current team\(^1\)
  - Result ($A$) is undefined
  - Current team cannot be used for collectives
    - Would need to form a new team w/o failed images

---

1. Fortran 2018: implies all images see the same STAT= value; will change in future standard
use, intrinsic: iso_fortran_env, only: team_type
type(team_type) :: team_variable
integer :: team_number
...
FORM TEAM (team_number, new_team)

Similar to:

call MPI_Comm_split(comm = current_team, &
                     color = team_number, &
                     key = 0, &
                     newcomm = new_team)
All operations in CHANGE TEAM construct refer to new team

CHANGE/END TEAM cause synchronization among images in new_team
Adding FORM TEAM STAT= allows shrinking recovery

- Failed images removed from new_team
- Image indices in new_team processor-dependent
EXAMPLE
EXAMPLE
PARALLEL MONTE CARLO PI
call random_init(repeatable=.false.,
  image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*), 4.0d0*n/SS/NUM.Images()
PARALLEL MONTE CARLO PI

1 call random_init(repeatable=.false.,
   image_distinct=.true.)
2
3 do sample = 1, SS
4   call random_number(x); call random_number(y)
5   if (hypot(x, y) <= 1) n = n + 1
6 end do
7
8 call co_sum(n, result_image=1)
9
10 if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM IMAGES()

Seed the random number generator on each image.
Randomly sample $SS$ ordered pairs $(x, y) \in [0, 1]$
PARALLEL MONTE CARLO PI

Count the number that are within the unit circle.

Note \( \text{hypot}(x, y) = \sqrt{x^2 + y^2} \)
PARALLEL MONTE CARLO PI

1 call random_init(repeatable=.false.,
   image_distinct=.true.)
2
do sample = 1, SS
4   call random_number(x); call random_number(y)
5     if (hypot(x, y) <= 1) n = n + 1
6 end do
7
8 call co_sum(n, result_image=1)
9
10 if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()

Sum counts from each image.
Save the results on image 1.
PARALLEL MONTE CARLO PI

1  call
   random_init(repeateable=.false.,image_distinct=.true.)
2
3  do  sample = 1, SS
4    call random_number(x); call random_number(y)
5    if (hypot(x, y) <= 1) n = n + 1
6  end do
7
8  call co_sum(n, result_image=1)
9
10 if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()

\[ \pi \approx \frac{4 \times \frac{n_{\text{sum}}}{SS}}{\text{NUM IMAGES}} \]
PARALLEL MONTE CARLO PI (RESILIENT)

1. call random_init(repeatable=.false.,
   image_distinct=.true.)

2. do sample = 1, SS
3.   call random_number(x); call random_number(y)
4.   if (hypot(x, y) <= 1) n = n + 1
5. end do

6. n_copy = n

7. do
8.   form team(1, team_active_images, stat=status)
9.   change team (team_active_images, stat=status)
10.  image_in_team = this_image()
11.  call co_sum(n, result_image=1, stat=status)

Forward, shrinking recovery from failure of any image\(^1\)

1. Except image_in_team == 1 after END TEAM
On image failure, restore $n$ from a copy...
PARALLEL MONTE CARLO PI (RESILIENT)

```fortran
1   call random_init(repeatable=.false.,
   image_distinct=.true.)
2
3   do sample = 1, SS
4     call random_number(x); call random_number(y)
5    if (hypot(x, y) <= 1) n = n + 1
6   end do
7
8   n_copy = n
9
10  do
11     form team(1, team_active_images, stat=status)
12     change team (team_active_images, stat=status)
13     image_in_team = this_image()
14     call co_sum(n, result_image=1, stat=status)
```

...form a new team (excluding failed images)...
PARALLEL MONTE CARLO PI (RESILIENT)

1. call random_init(repeatable=.false.,
   image_distinct=.true.)

2. do sample = 1, SS
3.   call random_number(x); call random_number(y)
4.   if (hypot(x, y) <= 1) n = n + 1
5. end do

6. n_copy = n

7. do
8.   form team(1, team_active_images, stat=status)
9.   change team (team_active_images, stat=status)
10.  image_in_team = this_image()
11. call co_sum(n, result_image=1, stat=status)

Use the new team for CO_SUM
PARALLEL MONTE CARLO PI (RESILIENT)

1 call random_init(repeatable=.false.,
               image_distinct=.true.)
2
3 do sample = 1, SS
4    call random_number(x); call random_number(y)
5    if (hypot(x, y) <= 1) n = n + 1
6 end do
7
8 n_copy = n
9
10 do
11    form team(1, team_active_images, stat=status)
12    change team (team_active_images, stat=status)
13       image_in_team = this_image()
14    call co_sum(n, result_image=1, stat=status)

If no (further) image failure is detected after CO_SUM and END TEAM, then EXIT the DO loop...
Output result (adjusting by # of active images) from image that was image 1 in team_active_images.

1. Fortran disallows THIS_IMAGE(TEAM=child), NUM_IMAGES(TEAM=child)
OpenCoarrays-ft implementation (approximate)\(^{1}\)

1. Based on original OpenCoarrays MPI error handler.
Attempt to create an MPI communicator for the new team
Fault-tolerant consensus on success of split operation (bitwise-AND among non-failed processes)
If MPI_COMM_SPLIT failed at any process (due to locally-detected process failure), or any (locally-unacknowledged) process failure is detected during MPI_COMM_AGREE...
MPI_COMM_SHRINK (ULFM): create new comm w/o failed processes
Acknowledge failed processes in `current_team`

Allows subsequent use without

`MPIX_ERR_PROC_FAILED`
Get group of failed processes in `current_team`
Retry, creating a new team from the current team – failed images
FORM TEAM

Non-shrinking recovery

team_num = 1
if (this_image >= 100) team_num = 2 ! spare image
...
FORM TEAM(team_num, new_team, NEW_INDEX=new_team_idx, STAT=s)

- Initial team contains "spare" images
- Image index ordering preserved with NEW_INDEX=
  - Added to OpenCoarrays-ft (and custom GFortran)
- Example C.6.8 (Fortran 2018 standard)
  - Issues; See paper for enhanced version
1  \textbf{FORM} \textbf{TEAM}((\texttt{THIS\_IMAGE()}-1)/(\texttt{NUM\_IMAGES()}/2)+1), \&
2  \quad \text{new\_team}, \&
3  \quad \text{NEW\_INDEX=MOD(THIS\_IMAGE()-1,NUM\_IMAGES()/2)+1}, \&
4  \quad \text{STAT=stat})
FORM TEAM NUM IMAGES =

Proposed semantics

<table>
<thead>
<tr>
<th>current_team</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>FAIL</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_team</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\text{team\_number == 1} \quad \text{ VALID} \quad \text{ team\_number == 2} \quad \text{ VALID}
\]

! Similar to MPI_COMM_SPLIT, with non-consecutive key values

```
call MPI_Comm_split(comm = current_team, &
    color = team_number, &
    key = new_index, &
    newcomm = new_team, &
    ierror = stat)
```
BENCHMARKS
SOFTWARE ENVIRONMENT

- OpenCoarrays-ft
- ULFM2
- GFortran 9.3.0 (modified)
  - STAT= and ERRMSG= in additional image control statements
  - FORM TEAM NEW_INDEX= specifier
SOFTWARE ENVIRONMENT

Available as a software container image (Alpine Linux)

Docker

```
$ alias dcaf='docker run -it --rm -v $PWD:/mnt -w $PWD:/mnt
ghcr.io/nathanweeks/espm2-2020:latest'
$ dcaf caf prog.f90
$ dcaf cafrun -np 8 ./a.out
```

Singularity

```
$ singularity pull docker://ghcr.io/nathanweeks/espm2-2020:latest
$ singularity exec espm2-2020_latest.sif caf prog.f90
$ singularity exec espm2-2020_latest.sif cafrun -np 8 ./a.out
```

Caveat: expect bugs!
SOFTWARE ENVIRONMENT

- NERSC Cori (KNL)
- Shifter
  - TCP BTL
TEAM SYNCHRONIZATION

MPI_BARRIER-based synchronization

1. do i = 1, 1000
2. call system_clock(...)
3. form team(1, active_images)
4. call system_clock(...)
5. change team(active_images)
6. call system_clock(...)
7. end team
8. end do
9. call system_clock(...)


Fault-tolerant synchronization (MPI_COMM_AGREE)

```plaintext
1 do i = 1, 1000
2   call system_clock(...)
3   form team(1, active_images, stat=status)
4   call system_clock(...)
5   change team(active_images, stat=status)
6     call system_clock(...)  
7   end team(stat=status)
8 end do
9 call system_clock(...)  
```
FORM TEAM

Synchronization
- **MPI_BARRIER (ft disabled)**
- **MPI_BARRIER**
- **MPI_COMM_AGREE**

Time (milliseconds)

Number of images
CHANGE TEAM / END TEAM

Synchronization
- MPI_BARRIER (ft disabled)
- MPI_BARRIER
- MPI_COMM_AGREE

Time (milliseconds)

Number of images

2 4 8 16 32 64
SOLE SURVIVOR

Time FORM / CHANGE / END team in the presence of image failures.

```plaintext
1   sync all
2   do i = 1, num_images()-1
3      call system_clock(...) 
4   form team(1, active_images, stat=status)
5      call system_clock(...) 
6   change team(active_images, stat=status)
7      call system_clock(...) 
8      if (this_image() == num_images()) fail image
9      end team(stat=status)
10  end do
11  call system_clock(...) 
```
SOLE SURVIVOR

```c
1  sync all
2  do i = 1, num_images()-1
3      call system_clock(...)
4  form team(1, active_images, stat=status)
5      call system_clock(...)
6  change team(active_images, stat=status)
7      call system_clock(...)
8  if (this_image() == num_images()) fail image
9      end team(stat=status)
10  end do
11  call system_clock(...)
```

One image fails per iteration, until only one remains.
SUMMARY

• Fortran 2018 defines abstractions for fault tolerance
• Prototype Fortran failed images + teams
• Debug/validate Fortran 2018
  ▪ Fortran standard changes needed to facilitate portable resilient applications
ACKNOWLEDGEMENT

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231, ROR ID https://ror.org/05v3mvq14.
QUESTIONS?