What is FTI?

FTI stands for Fault Tolerance Interface and is a library that aims to give computational scientists the means to perform fast and efficient multilevel checkpointing in large scale supercomputers. FTI leverages local storage plus data replication and erasure codes to provide several levels of reliability and performance. FTI is application-level checkpointing and allows users to select which datasets needs to be protected, in order to improve efficiency and avoid wasting space, time and energy. In addition, it offers a direct data interface so that users do not need to deal with files and/or directory names. All metadata is managed by FTI in a transparent fashion for the user. If desired, users can dedicate one process per node to overlap fault tolerance workload and scientific computation, so that post-checkpoint tasks are executed asynchronously.

Loading the FTI module

module load Intel/2018.2.199-GCC-5.5.0 ParaStationMPI/5.2.1-1 FTI

Documentation and User guide

Doxygen: ?http://leobago.github.io/fti/

Manual: ?https://github.com/leobago/fti/wiki

Quick Start Guide

Before the Execution

The user needs to provide an FTI configuration file (see attachments: config.fti)

FTI Calls inside Application

- Before any other FTI API call, **FTI_Init** has to be called.
- **FTI_Protect** informs FTI about a buffer that needs to be checkpointed.
- **FTI_Checkpoint** writes the checkpoint file, containing all protected buffers.
- **FTI_Status** returns 1 if the execution has been failed and was recovered.
- **FTI_Recover** updates all protected buffer by loading the data from the checkpoint file.
- **FTI_Finalize** finalizes FTI.

Example

```
#include <stdlib.h>
#include <fti.h>
#define ITER_CHECK 10

int main(int argc, char** argv){
   MPI_Init(&argc, &argv);
   char* path = "config.fti"; //config file path
   FTI_Init(path, MPI_COMM_WORLD);
   int world_rank, world_size; //FTI_COMM rank & size
   MPI_Comm_rank(FTI_COMM_WORLD, &world_rank);
   MPI_Comm_size(FTI_COMM_WORLD, &world_size);

   int *array = malloc(sizeof(int) * world_size);
   int number = world_rank;
   int i = 0;
   //adding variables to protect
   FTI_Protect(1, &i, 1, FTI_INTG);
   FTI_Protect(2, &number, 1, FTI_INTG);
   if (FTI_Status() != 0) {
       FTI_Recover();
   }
```

```
    for (; i < 100; i++) {
        if (i % ITER_CHECK == 0) {
            FTI_Checkpoint(i / ITER_CHECK + 1, 2);
        }
        MPI_Allgather(&number, 1, MPI_INT, array,
                      1, MPI_INT, FTI_COMM_WORLD);
        number += 1;
    }
    free(array);
    FTI_Finalize();
    MPI_Finalize();
    return 0;
}
```

Feature List

- Multi level checkpointing
  - Node local
    - single checkpoint per node (level 1)
    - Buddy checkpoints (level 2)
    - Encoding checkpoints tolerating failures of half of the nodes/processes (level 3)
  - checkpoints on parallel file system (level 4)
- Differential checkpointing (available for posix/fti-ff level 4). Differential checkpointing means the differential update of a CP file. Unchanged application data compared to the last checkpoint will not be updated and thus saves I/O time.
- Incremental checkpointing. Incremental checkpointing means the incremental completion of a CP file. This technique serves primarily to avoid overhead caused by oversaturated network channels. It may also be used to overlap computation and checkpoint I/O
- Single file checkpoint on the PFS using HDF5 or MPI-IO
- Asynchronous post-processing of higher level checkpoints