www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

FTI: Fault Tolerance Interface

Presented by: Kai Keller

Developed by Dr. Leonardo Bautista Gomez

Nesus Winter School, Vibo Valentina, Italy
February 21th 2017

Motivation

Motivations

- Fault tolerance is critical at extreme scale.
- More components, more failures.
- Dense architectures, correlated failures.
- Multiple different types of failures (hard, soft, ect).
- Power limits might impact reliability.
- Current techniques will not scale.

Basic informations about FTI

FTI

- Download at http://www.github.com/leobago/fti
- Documentation at http://leobago.github.io/fti
- Library in c/c++ with Fortran bindings
- More than 8000 lines of code
- Applications ported:

  - HACC
  - Nek5K
  - CESM (ice module)
  - LAMMPS
  - GYSELA 5D
  - SPECFEM3D (CUDA version)
  - HYDRO
  - Other miniApps

Why FTI?

Multilevel Checkpointing

Local Storage: SSD, PCM, NVM.
Fastest checkpoint level.
Low reliability, transient failures.

Partner Copy: Ckpt. Replication.
Fast copy to neighbor node.
It tolerates single node crashes.

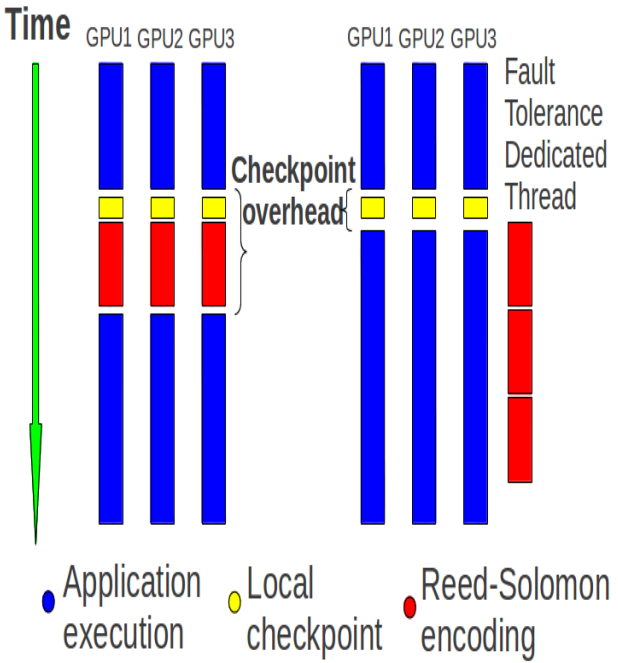
RS Encoding: Ckpt. Encoding.
Slow for large checkppoints.
Very reliable, multiple node crashes.

File System: Classic Ckpt.
Slowest of all levels.
The most reliable. Power outage.

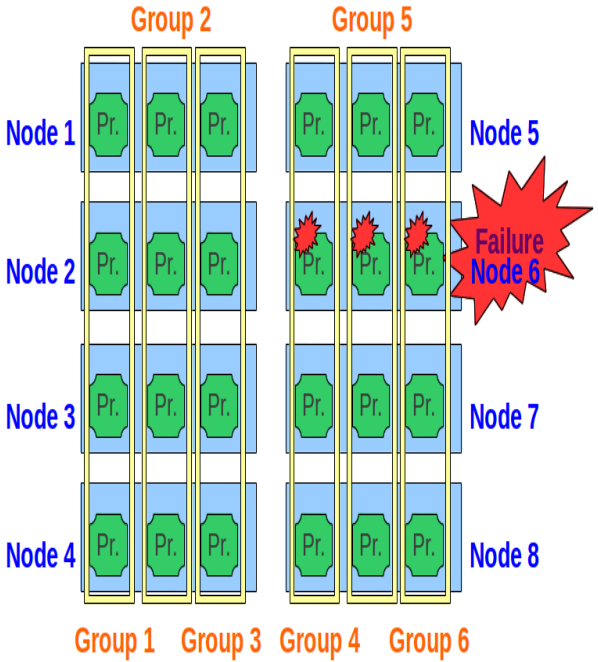
- **Multiple:**
  - Resiliency levels
  - Checkpoint overheads
  - Checkpoint intervals
  - Power consumptions

Topology aware clustering

- FTI dedicated threads
- Asynchronous data transfer
- Reduced ckpt. Overhead
- Fine-grained control
- Disjoint communicators



Topology aware clustering



- Automatic process location recognition
- Intelligent clustering
- Enhanced reliability for node crashes
- Automatic repositioning after failure

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

How to use?

Easy-to-use API

- **Functions:**
  - FTI_Init()
  - FTI_Protect()
  - FTI_Snapshot()
  - FTI_Finalize()
- **Communicator:**
  - FTI_COMM_WORLD

```
int main(int argc, char **argv) {

   MPI_Init(&argc, &argv);
   FTI_Init("conf.fti", MPI_COMM_WORLD);

   double *grid;
   int i, steps=500, size=10000;
   initialize(grid);
   FTI_Protect(0, &i,   1,   FTI_INTG);
   FTI_Protect(1, grid, size,FTI_DFLT);

   for (i=0; i<steps; i++) {
      FTI_Snapshot();
      kernel1(grid);
      kernel2(grid);
      comms(FTI_COMM_WORLD);
   }

   FTI_Finalize();
   MPI_Finalize();
   return 0;
}
```

FTI_Init()

- **FTI_Init(confFile, communicator)**:
  - Read/parse configuration file
  - Recognizes whether is a restart or not
  - Creates checkpoint directories
  - Detect topology of the system
  - Regenerates/moves data upon recovery
  - Splits the communicator (optional)

FTI_Protect()

- **FTI_Protect(ID,pointer,size,type):**
  - Stores metadata of the protected variable
  - FTI can predict size of checkpoints
  - Useful for data compression/aggregation
  - Can be reseted during the execution
  - User can create new FTI types
  - Required in order to write/read ckpt. data

FTI_Snapshot()

- **FTI_Snapshot():**
  - Measures (global average) iteration length
  - Exponential decay for global agreement
  - Translates from minutes to iterations
  - Test if it is time for a checkpoint
  - If it is, it checks which level of ckpt.
  - It saves the checkpoint as requested
  - It loads the checkpoint upon recovery
  - Planning to integrate notifications

Beyond FTI_Snapshot

- **FTI_Checkpoint(ID, lvl):**
  - Takes a checkpoint with id *ID* and level *lvl*
- **FTI_Status()**
  - Returns the status (initial run or restart)
- **FTI_Recover():**
  - It recovers from last available checkpoint

FTI_Finalize()

- **FTI_Finalize():**
  - Frees the allocated memory
  - Informs it is over to dedicated threads
  - Clean checkpoints and metadata
  - Moves last ckpt to PFS if requested

Cofiguration file for (1/3)

```
[basic]
# Set to 1 for having 1 FTI dedicated process per node
Head                            = 1

# Number of processes per node (including FTI dedicated processes)
node_size                       = 2

# Path where local checkpoints will be stored
ckpt_dir                        = /path/to/local/storage/

# Path where global checkpoints will be stored
glbl_dir                        = /path/to/global/storage/

# Path where checkpoints metadata will be stored
meta_dir                        = /path/to/myhome/.fti/

# Checkpoint interval in minutes for level 1
ckpt_int                        = 1

# Checkpoint interval in minutes for level 2
ckpt_l2                         = 2

# Checkpoint interval in minutes for level 3
ckpt_l3                         = 4

# Checkpoint interval in minutes for level 4
ckpt_l4                         = 8
```

Cofiguration file for FTI (2/3)

```
[basic]

# Set to 0 to do L2 post-processing asynchronously by the
dedicated process
inline_l2                       = 0

# Set to 0 to do L3 post-processing asynchronously by the
dedicated process
inline_l3                       = 0

# Set to 0 to do L4 post-processing asynchronously by the
dedicated process
inline_l4                       = 0

# Set to 1 to keep the last checkpoint after Finalize
keep_last_ckpt                  = 0

# Size of the group for RS-encoding and Partner-copy ring
group_size                      = 4

# Set to 1 for verbose mode, 2 for moderate, 3 for silent
verbosity
```

Cofiguration file for (3/3)

```
[restart]

# This will be set to 1 automatically after FTI_Init
Failure                         = 0

# This will be set to 1 automatically after FTI_Init
exec_id                         = 2013-11-20_15-01-52


[advanced]

# Block size for communications
block_size                      = 1024

# MPI tag for FTI communications
mpi_tag                         = 2612

# Set to 1 for local tests in one single node
local_test                      = 0
```

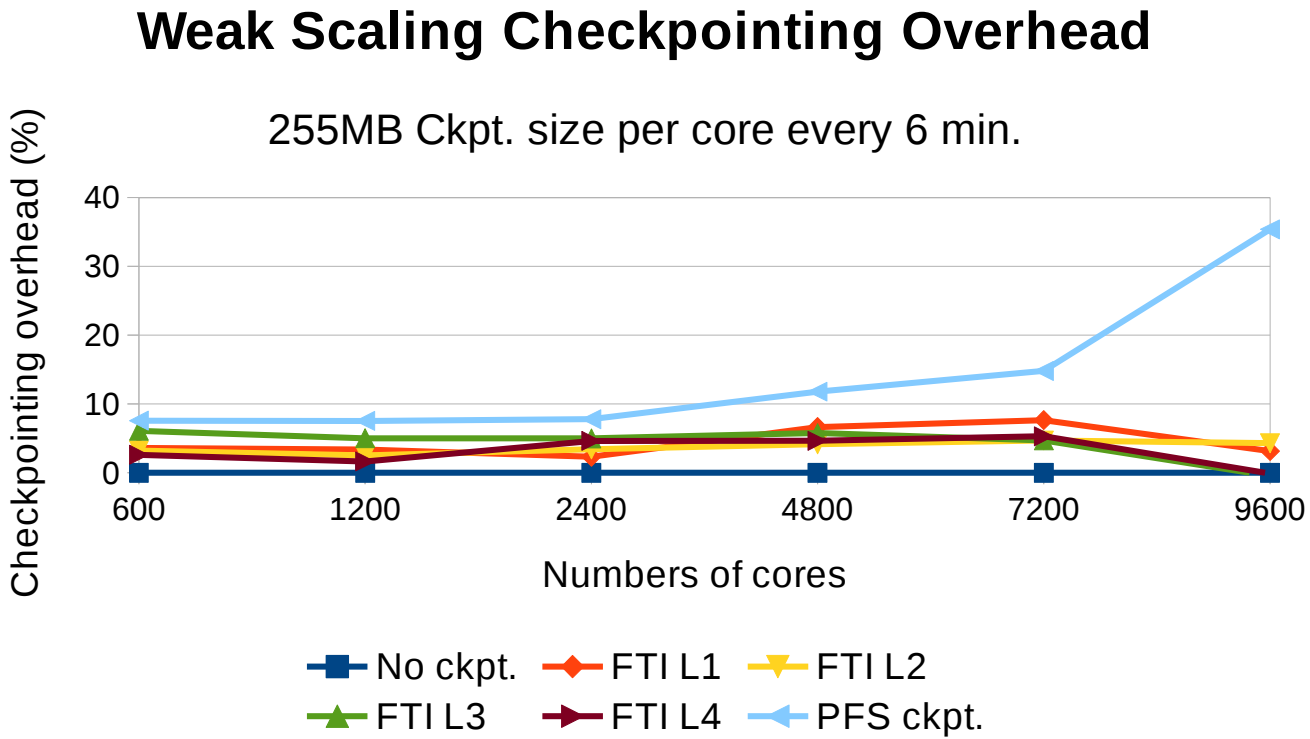
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance

Scaling to ~10K processes

- CURIE supercomputer in France
- SSD on the compute nodes (16 cores)
- HYDRO scientific application
- Using 1 FTI dedicated process per node
- Checkpointing every ~6 minutes
- Weak scaling to almost 10k processes

Scaling to ~10K processes



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Scaling to >32K processes

- MIRA supercomputer at ANL (BG\Q)
- Persistent memory compute nodes
- LAMMPS scientific application
- Lennard-Jones simulation of 1.3 billion atoms
- 512 nodes, 64 MPI processes per node (32,678pr.)
- Using 1 FTI dedicated process per node
- Power monitoring during the entire run
- Checkpointing every ~5 minutes
- Less than 5% overhead on time to completion

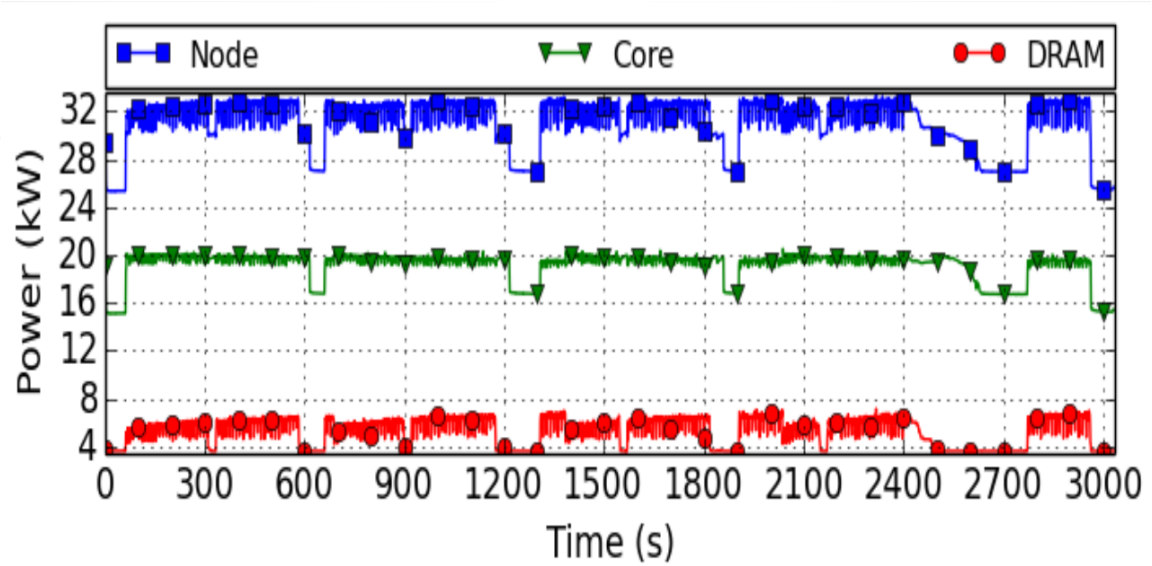
Scaling to >32K processes

Synchronous Checkpointing

Without FTI - dedicated process

Head = 0

Execution: ~ 3000s



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Scaling to >32K processes

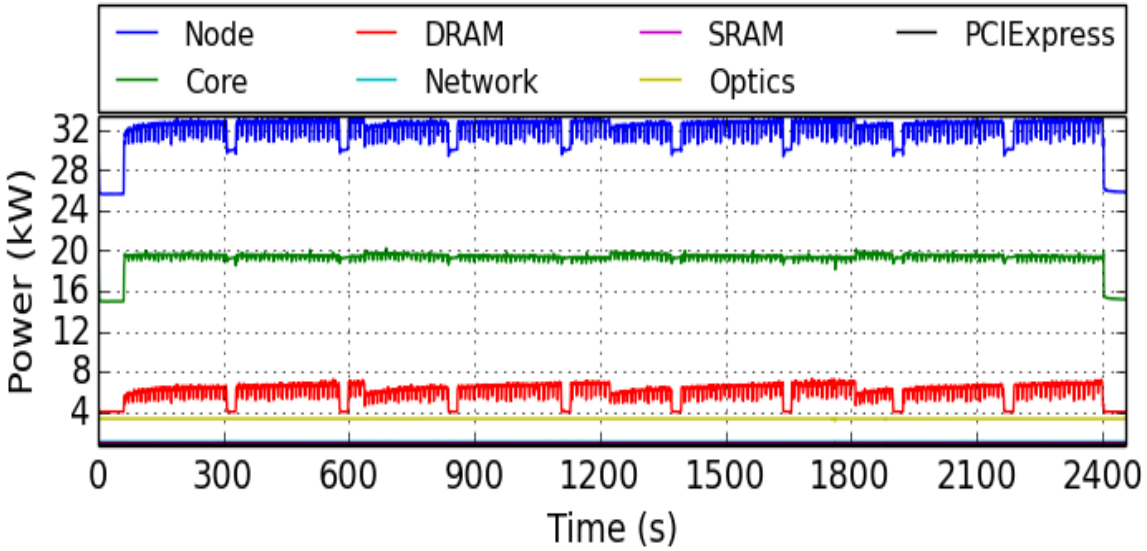
Asynchronous Checkpointing

With FTI-dedicated process

Head = 1

Execution: ~ 2400s

10 minutes faster!



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Features / Limitations

Interesting features

- FTI can predict time and size of next checkpoints
- Detailed knowledge of the datasets allows for transparent data compression/verification
- Transparent dedicated processes (Comm. Split)
- Topology reconstruction upon restart
- Dynamic checkpoint interval adaptation

Limitations

- FTI needs every rank in the given communicator to write a checkpoint file
- Application level checkpoint (code modification)
- Coordinated checkpoint, everybody restarts


Thank you!