Fault Tolerance Interface (FTI)

State-of-the-art multi-level checkpointing library

Leonardo Bautista-Gomez
Senior Researcher
Barcelona Supercomputing Center

Why FTI

- Supercomputers grow exponentially
- Failures grow with the number of components
- Mean Time Between Failures (MTBF) decreasing
- Parallel File Systems (PFS) grow linearly
- Time to checkpoint increasing

Why FTI

- Supercomputers grow exponentially
- Failures grow with the number of components
- Mean Time Between Failures (MTBF) decreasing
- Parallel File Systems (PFS) grow linearly
- Time to checkpoint increasing
- MTBF about 6 hours (ckpt. Interval about 2 hours)
- Checkpoint 1000 nodes (100GB/node)
- PFS about 100GB/s => 1000 seconds (~17 minutes)
- About **15%** of the time spent in checkpointing

Why FTI

- Supercomputers grow exponentially
- Failures grow with the number of components
- Mean Time Between Failures (MTBF) decreasing
- Parallel File Systems (PFS) grow linearly
- Time to checkpoint increasing
- MTBF about 6 hours (ckpt. Interval about 2 hours)
- Checkpoint 1000 nodes (100GB/node)
- PFS about 100GB/s => 1000 seconds (~17 minutes)
- About **15%** of the time spent in checkpointing

We need multi-level checkpointing!!!

Why FTI

- Supercomputers grow exponentially
- Failures grow with the number of components
- Mean Time Between Failures (MTBF) decreasing
- Parallel File Systems (PFS) grow linearly
- Time to checkpoint increasing
- MTBF about 6 hours (ckpt. Interval about 2 hours)
- Checkpoint 1000 nodes (100GB/node)
- PFS about 100GB/s => 1000 seconds (~17 minutes)
- About **15%** of the time spent in checkpointing

https://github.com/leobago/fti

How to use FTI

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

    MPI_Init(&argc, &argv);

    MPI_Comm comm = MPI_COMM_WORLD;




    for(step=0; step<NB_STEPS; step++) {

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }


    MPI_Finalize();
    return 0;
}
```

How to use FTI

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

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




    for(step=0; step<NB_STEPS; step++) {

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

How to use FTI

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

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

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

How to use FTI

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

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

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

How to use FTI

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

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

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 120 #minutes
```

How to use FTI

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

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

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 360
```

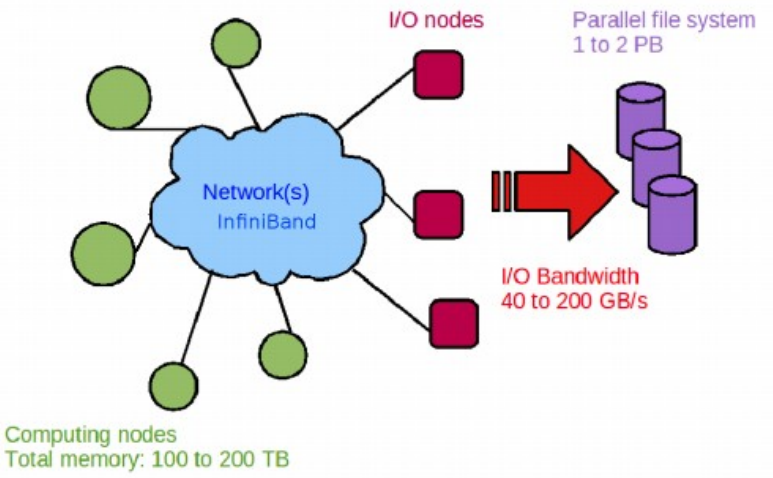
How to use FTI

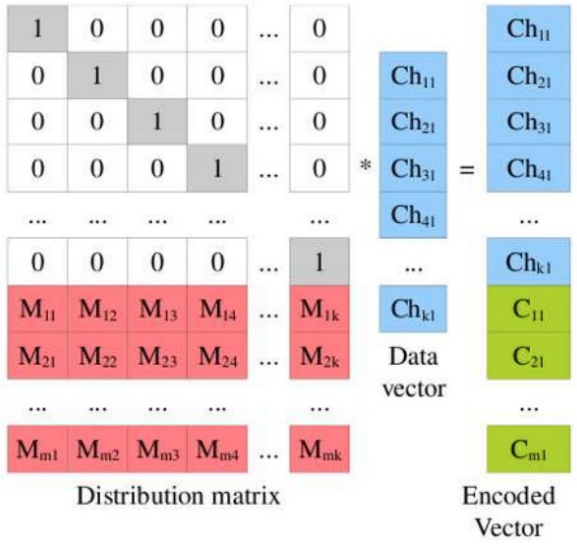
```
int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = MPI_COMM_WORLD;

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

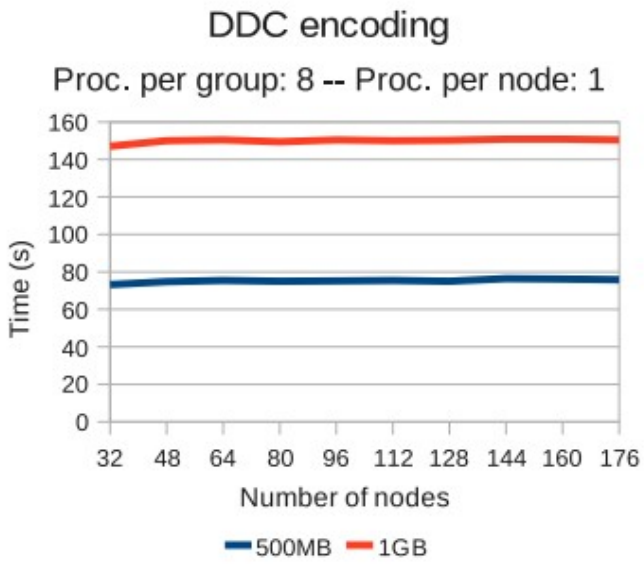
```
Node_size = 4

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 60
Ckpt_L3 = 0
Ckpt_L4 = 600
```

Reed-Solomon Encoding

- Data replication is not space efficient
- Erasure codes to recover data lost upon failures
- Same space as replication but much higher resilience
- Scalable Reed-Solomon implementation by groups
- Evaluation on hundreds of nodes







* *"Distributed diskless checkpoint for large scale systems"* - Leonardo Bautista-Gomez, Naoya Maruyama, Franck Cappello, Satoshi Matsuoka - 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (**CCGrid'10**)

Reed-Solomon Encoding

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = MPI_COMM_WORLD;

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);




    for(step=0; step<NB_STEPS; step++) {

        FTI_Snapshot();

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 60
Ckpt_L3 = 90
Ckpt_L4 = 0
```

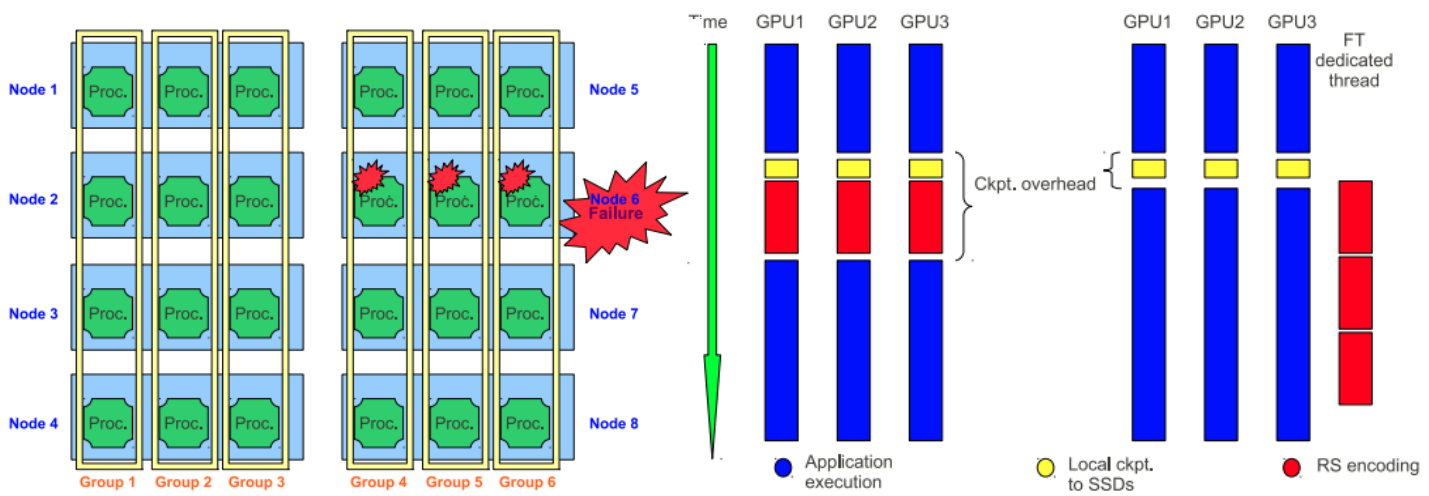
Fine Tuning Time to Checkpoint

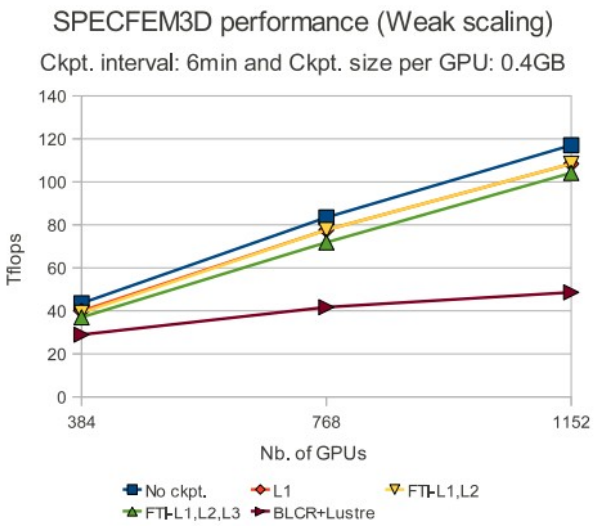
```
int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = MPI_COMM_WORLD;

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);
    state = FTI_Status();
    if (state == RESTART || state == KEEP) {
            result = FTI_Recover();
    }
    for(step=1; step=<NB_STEPS; step++) {
        if (step % 100 == 0) {
            FTI_Checkpoint(step, FTI_L3);
        }
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 0
```

Asynchronous Checkpointing

- Virtual ring domain decomposition for RS-encoding
- Dedicated threads for asynchronous checkpointing
- Multilevel ckpt: 4 levels of resilience and performance
- Under 10% overhead while executing over 1K GPUs
- Available: https://github.com/leobago/fti





* *"FTI: high performance fault tolerance interface for hybrid systems"* - Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, Satoshi Matsuoka - 2011 international conference for high performance computing, networking, storage and analysis (**SC'11**)

Asynchronous Checkpointing

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = MPI_COMM_WORLD;

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4
Head = 0

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 60
Ckpt_L3 = 0
Ckpt_L4 = 600
```

Asynchronous Checkpointing

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 60
Ckpt_L3 = 90
Ckpt_L4 = 0
```


* *"Optimization of a multilevel checkpoint model with uncertain execution scales"* - Sheng Di, Leonardo Bautista-Gomez, Franck Cappello - 2014 international conference for high performance computing, networking, storage and analysis (**SC'14**)

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 1 #POSIX
```

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 2 #MPI-IO
```

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 3 #FFF
```

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 4 #SIONlib
```

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 5 #HDF5
```

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 6 #IME native
```

I/O Features

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 5 #HDF5
Keep_Last_ckpt = 1
```

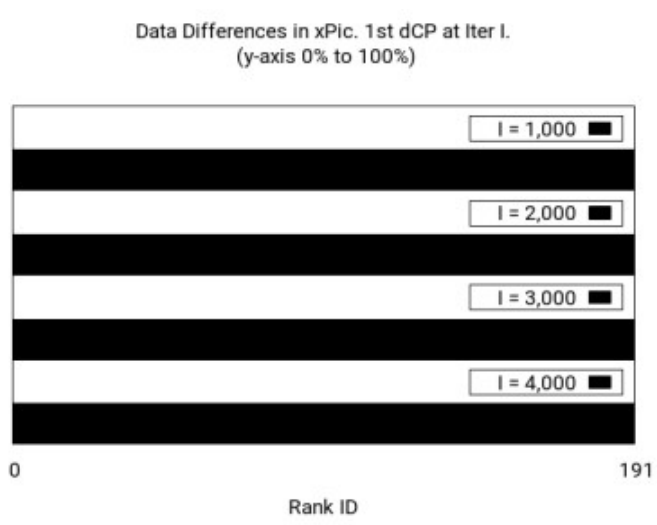
I/O Features

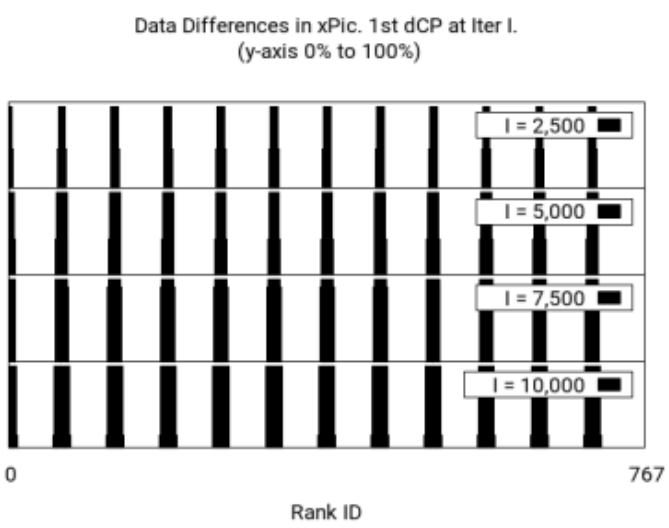
```
int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

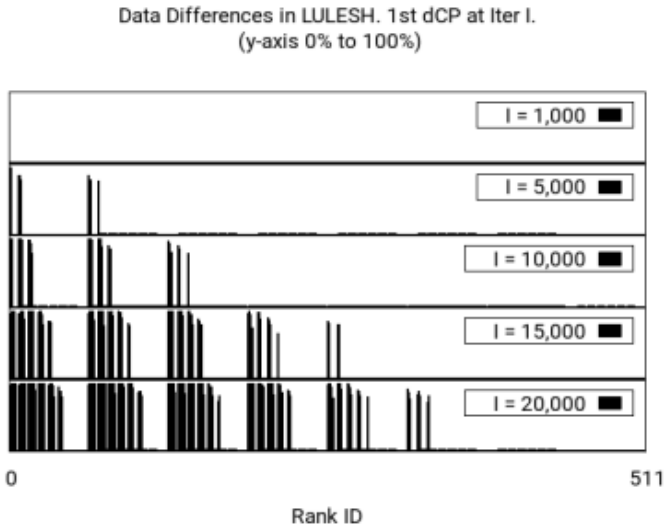
```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600

Ckpt_IO = 5 #HDF5
Keep_L4_ckpt = 1
```

Differential Checkpointing

- Checkpoint only the diff between last ckpt and this one
- Dirty page technology is not enough to detect changes
- Checksum based diff calculations (MD5)
- Dynamics Dataset sizes increases complexity
- Application data evolves differently: Xpic, Heat, Lulesh







* *"Application-Level Differential Checkpointing for HPC Applications with Dynamic Datasets"* - Kai Keller and Leonardo Bautista-Gomez - 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (**CCGRID'19**)

Differential Checkpointing

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;



    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Dcp_L4  = 0
Ckpt_L4 = 60

Enable_dcp = 0
Dcp_mode = 0
Dcp_block_size = 16384
```

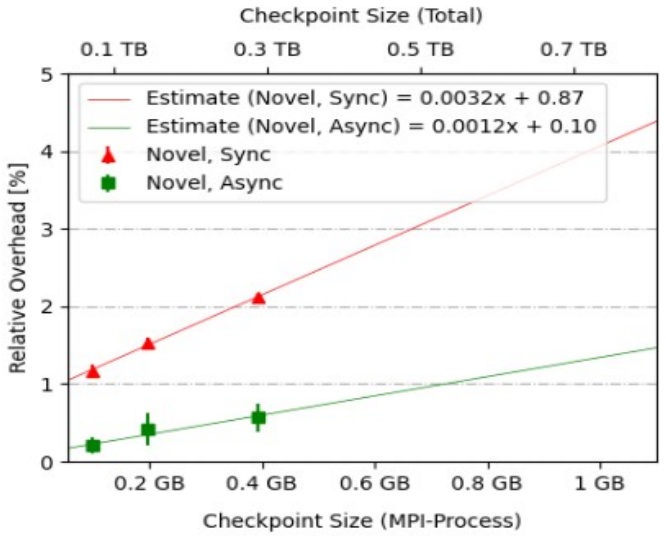
Differential Checkpointing

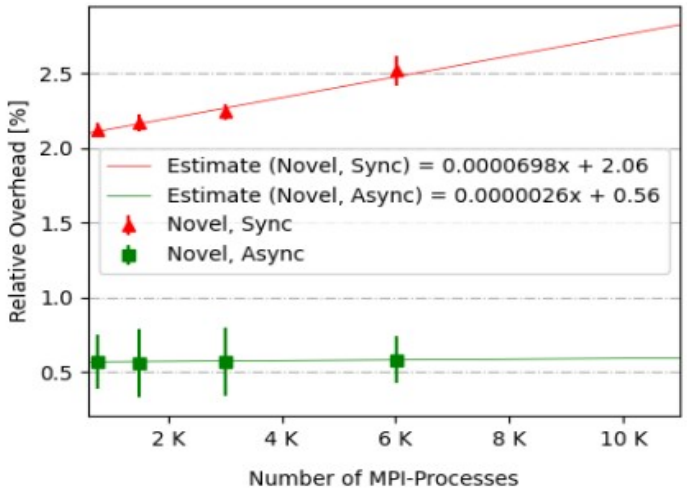
```
int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;



    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

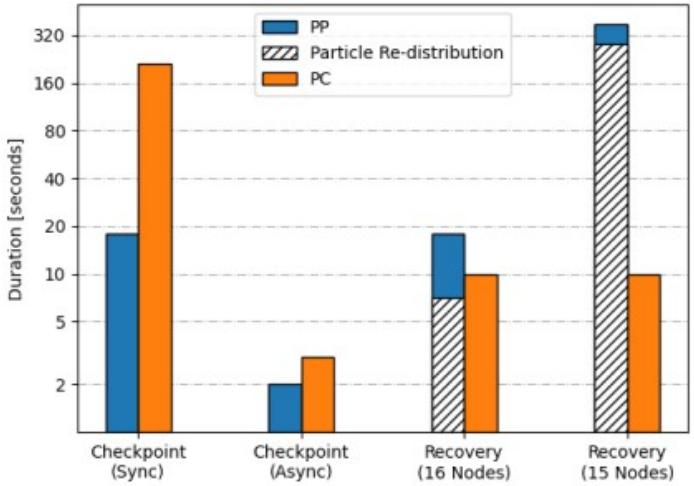
```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Dcp_L4  = 60
Ckpt_L4 = 240

Enable_dcp = 1
Dcp_mode = 1
Dcp_block_size = 16384
```

Elastic Recovery in HPC Applications

- Restart with a different number of processes
- Use HDF5 format with a N-1 ckpt. (instead of N-N)
- Two stages: First local N-N, then asynchronous N-1
- Over 5X speedup in comparison to ADIOS
- Tested on regular(heat) and irregular (xPic) applications







* *"Design and Study of Elastic Recovery in HPC Applications"* - Kai Keller, Konstantinos Parasyris, Leonardo Bautista Gomez - 2020 International Conference on High Performance Computing (**HiPC'20**)

Elastic Recovery in HPC Applications

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;


    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);


    for(step=0; step<NB_STEPS; step++) {

        FTI_Snapshot();

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4
Head = 0

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 180
```

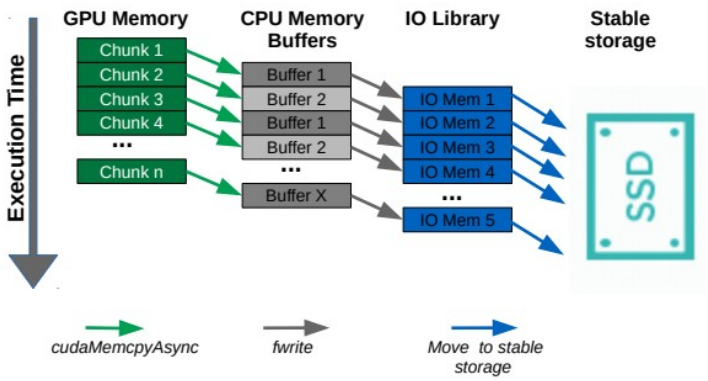
Elastic Recovery in HPC Applications

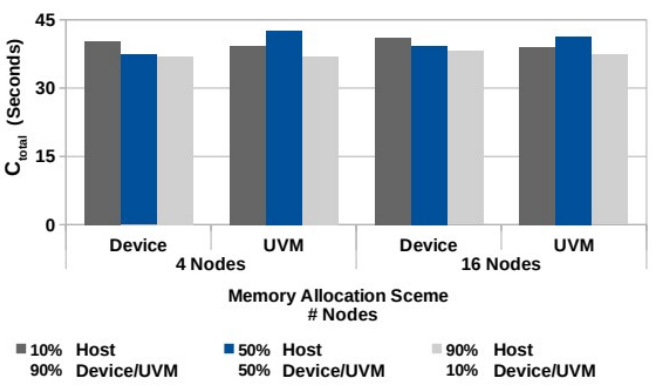
```
int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    FTI_DefineGlobalDataset(GRID_DATASET_ID, 2,
    {Mglobal,Nglobal}, "GRID", NULL, FTI_DBLE)
    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);
    FTI_AddSubset(1 (GRID_VARIABLE_ID), 2, {nrank*M,
    0}, {M, N}, GRID_DATASET_ID)
    for(step=0; step<NB_STEPS; step++) {

        FTI_Snapshot();

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

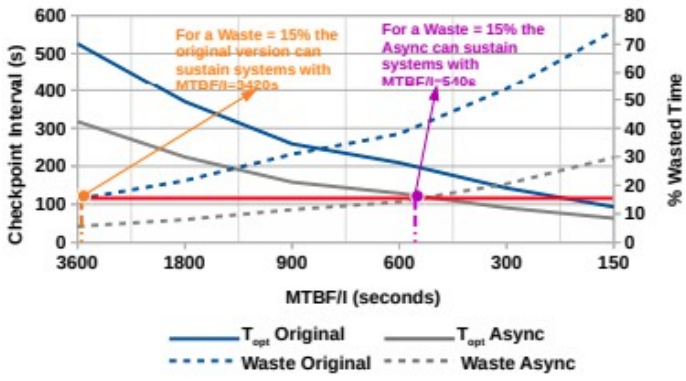
```
Node_size = 4
Head = 0

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 180
```

C/P support for Heterogeneous HPC

- Out of the top 10 supercomputers, 8 use GPUs
- Heterogeneity in memory as well (host, device, UVM)
- Streams and pipelines to optimize data movements
- Automatic and transparent data handling
- GPU-Accelerated MD5 computation
- Reduction of 15.23X/5.21X on checkpointing/recovery







* "*Checkpoint Restart Support for Heterogeneous HPC Applications*" - Konstantinos Parasyris, Kai Keller, Leonardo Bautista-Gomez, Osman Unsal - 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (**CCGRID'20**)

C/P support for Heterogeneous HPC

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    grid=(*double)malloc(M*N * sizeof(double));
    ghost=(*double)malloc(G *  sizeof(double));

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600
```

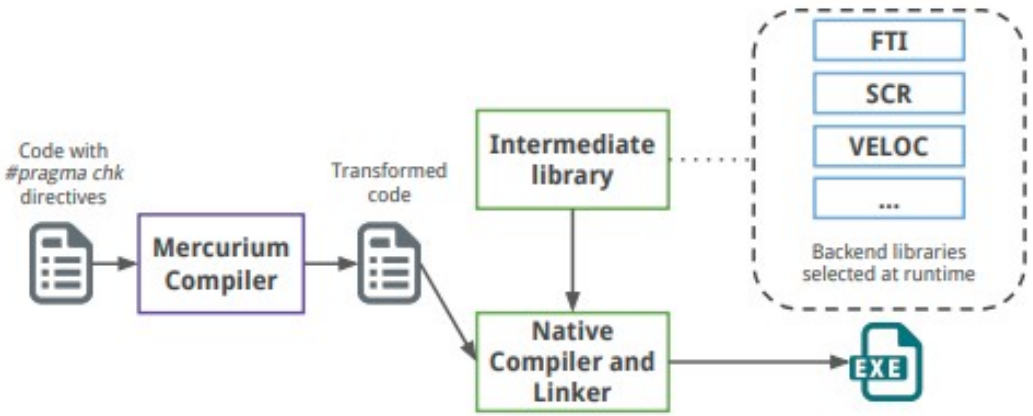
C/P support for Heterogeneous HPC

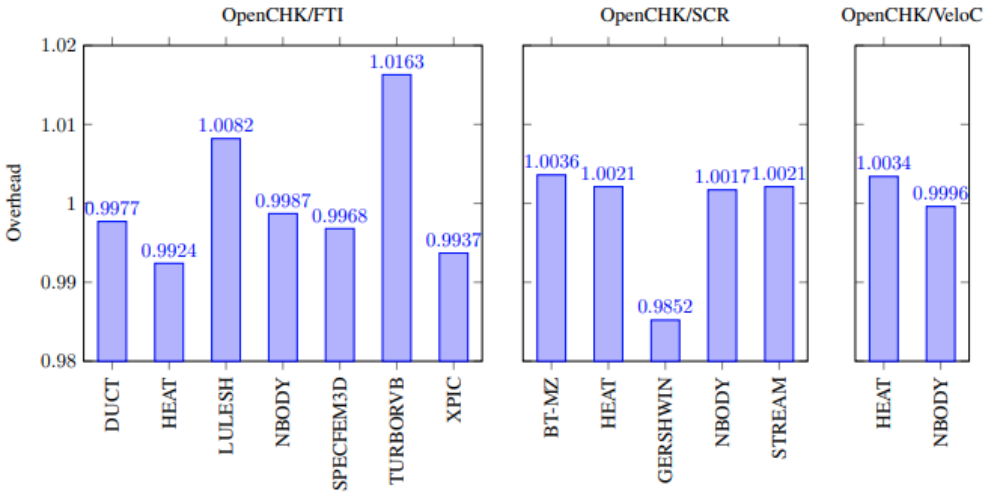
```
int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;
    cudaMalloc(&grid, M*N * sizeof(double);
    cudamalloc(&ghost, G *  sizeof(double);

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {
        FTI_Snapshot();
        accu = cudaWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 5
Head = 1

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 0
Ckpt_L2 = 6
Ckpt_L3 = 60
Ckpt_L4 = 600
```

OpenCHK: pragma interface

- Multilevel checkpointing libraries (FTI, SCR, VeloC)
- Similar tasks, different interfaces (file vs data)
- Pragma interface linked with checkpoint runtime
- #pragma chk (init, load, store, shutdown)
- Reduced LoC by 71% FTI, 94% SCR and 64% VeloC
- No overhead





* "*Extending the OpenCHK Model with advanced checkpoint features*" - Marcos Maroñas, Sergi Mateo, Kai Keller, Leonardo Bautista-Gomez, Eduard Ayguadé, Vicenç Beltran - **2020 Journal Future Generation Computer Systems**

OpenCHK: pragma interface

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

    MPI_Init(&argc, &argv);
    FTI_Init(argv[2], MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;

    FTI_Protect(0, &step, 1,   FTI_INTG);
    FTI_Protect(1, grid,  M*N, FTI_DBLE);
    FTI_Protect(2, ghost, G,   FTI_DBLE);

    for(step=0; step<NB_STEPS; step++) {

        FTI_Snapshot();

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

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

```
Node_size = 4
Head = 0

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 180
```

OpenCHK: pragma interface

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

    MPI_Init(&argc, &argv);
    #pragma chk init comm(MPI_COMM_WORLD);
    MPI_Comm comm = FTI_COMM_WORLD;

    #pragma chk load(step, grid[M*N], ghost[G])



    for(step=0; step<NB_STEPS; step++) {
        #pragma chk store(step, grid[M*N], ghost[G],
                          id(id), level(level))

        accu = doWork(grid, ghost, comm);
        if (accu < TARGET_ACCU) break;
    }

    #pragma chk shutdown;
    MPI_Finalize();
    return 0;
}
```

```
Node_size = 4
Head = 0

Ckpt_dir = /scratch/
Glbl_dir = /gpfs/myProject
Meta_dir = /home/user/.fti

Ckpt_L1 = 30
Ckpt_L2 = 0
Ckpt_L3 = 0
Ckpt_L4 = 180
```

Impact

- European projects: MontBlanc2, MontBlanc2020, Deepest, Legato, EoCoE, EoCoE2, eProcessor.
- ATOS integrates FTI as part of its software stack
- ASTRON integrated FTI for its Image Domain Gridding
- ALYA code ported with FTI at BSC
- Huawei testing FTI on its new hardware
- VeloC library from DoE uses same interface as FTI
- Integration with MPI_Re-Init (Collaboration with LLNL)

* “Co-Designing Multi-Level Checkpoint Restart for MPI Applications” - Konstantinos Parasyris, Giorgis Georgakoudis, Leonardo Bautista-Gomez, Ignacio Laguna - 2021 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (**CCGRID’21**) (To appear)

Summary

- Multi-level checkpointing, for reliability and performance
- Asynchronous checkpointing with dedicated processes
- Transparent GPU checkpointing for heterogeneous system
- Multiple I/O output formats supported
- Checkpointing for other than fault tolerance
- Differential checkpointing to reduce storage
- Fine tuning checkpoint frequency
- Elastic recovery for malleable applications

References

- *Distributed diskless checkpoint for large scale systems* - Leonardo Bautista-Gomez, Naoya Maruyama, Franck Cappello, Satoshi Matsuoka - 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (**CCGrid'10**)
- *Low-overhead diskless checkpoint for hybrid computing systems* - Leonardo Bautista-Gomez, Akira Nukada, Naoya Maruyama, Franck Cappello, Satoshi Matsuoka - 2010 International Conference on High Performance Computing (**HiPC'10**)
- *FTI: high performance fault tolerance interface for hybrid systems* - Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, Satoshi Matsuoka - 2011 international conference for high performance computing, networking, storage and analysis (**SC'11**)
- *Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing* - Mohamed Slim Bouguerra, Ana Gainaru, Leonardo Bautista Gomez, Franck Cappello, Satoshi Matsuoka, Naoya Maruyama - 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (**IPDPS'13**)
- *Optimization of multi-level checkpoint model for large scale HPC applications* - Sheng Di, Mohamed Slim Bouguerra, Leonardo Bautista-Gomez, Franck Cappello - 2014 IEEE 28th International Parallel and Distributed Processing Symposium (**IPDPS'14**)
- *Analysis of the tradeoffs between energy and run time for multilevel checkpointing* - Prasanna Balaprakash, Leonardo Bautista Gomez, Mohamed-Slim Bouguerra, Stefan M Wild, Franck Cappello, Paul D Hovland - 2015 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (**PMBS'15**)
- *Reducing waste in extreme scale systems through introspective analysis* - Leonardo Bautista-Gomez, Ana Gainaru, Swann Perarnau, Devesh Tiwari, Saurabh Gupta, Christian Engelmann, Franck Cappello, Marc Snir - 2016 IEEE International Parallel and Distributed Processing Symposium (**IPDPS'16**)
- *Performance study of non-volatile memories on a high-end supercomputer* - Leonardo Bautista Gomez, Kai Keller, Osman Unsal - 2018 International Conference on High Performance Computing (**WOPSS'18**)
- *Towards Ad Hoc Recovery for Soft Errors* - Nuria Losada, Leonardo Bautista-Gomez, Kai Keller, Osman Unsal - 2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (**FTXS'18**)
- *Checkpoint/restart approaches for a thread-based MPI runtime* - Julien Adam, Maxime Kermarquer, Jean-Baptiste Besnard, Leonardo Bautista-Gomez, Marc Pérache, Patrick Carribault, Julien Jaeger, Allen D Malony, Sameer Shende - (**2019 Journal Parallel Computing**)
- *Application-Level Differential Checkpointing for HPC Applications with Dynamic Datasets* - Kai Keller and Leonardo Bautista-Gomez - 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (**CCGRID'19**)
- *Checkpointing Kernel Executions of MPI+ CUDA Applications* - Max Baird, Sven-Bodo Scholz, Artjoms Šinkarovs, Leonardo Bautista-Gomez - 2019 European Conference on Parallel Processing (**EuroPar'19**)
- *Checkpoint Restart Support for Heterogeneous HPC Applications* - Konstantinos Parasyris, Kai Keller, Leonardo Bautista-Gomez, Osman Unsal - 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (**CCGRID'20**)
- *Extending the OpenCHK Model with advanced checkpoint features* - Marcos Maroñas, Sergi Mateo, Kai Keller, Leonardo Bautista-Gomez, Eduard Ayguadé, Vicenç Beltran - (**2020 Journal Future Generation Computer Systems**)
- *Design and Study of Elastic Recovery in HPC Applications* - Kai Keller, Konstantinos Parasyris, Leonardo Bautista Gomez - 2020 International Conference on High Performance Computing (**HiPC'20**)
- * "Co-Designing Multi-Level Checkpoint Restart for MPI Applications" - Konstantinos Parasyris, Giorgis Georgakoudis, Leonardo Bautista-Gomez, Ignacio Laguna - 2021 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (**CCGRID'21**) (To appear)

Questions?

Thank you!

leobago.com

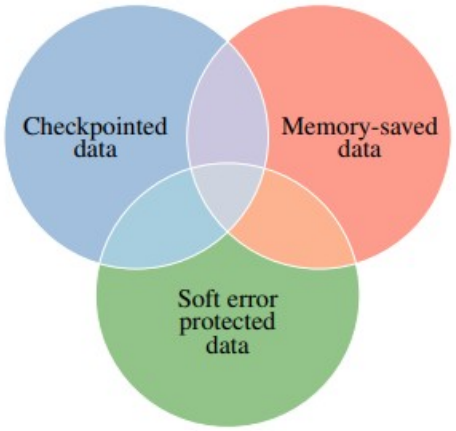
leonardo.bautista@bsc.es

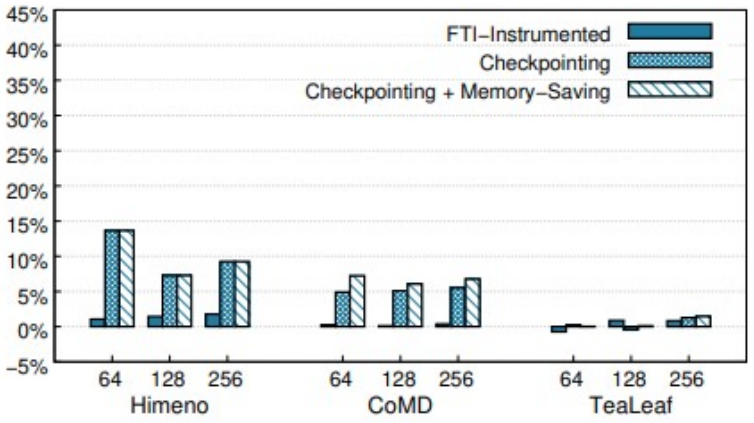
https://github.com/leobago/fti

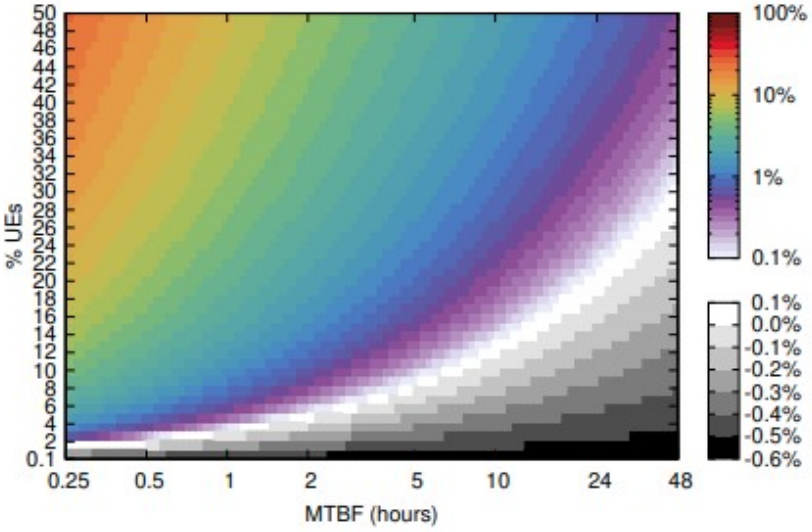
Acknowledgements: Kai Keller, Alexandre de Limas Santana, Konstantinos Parasyris, Karol Sierociński, Tomasz Paluszkiewicz, Sawsane Ouchtal, Julien Bigot, Nuria Losada, Pak Markthub, Max Baird, Mohamed Gaalich, Adele Villiermet, Slawomir Zdanowski

Ad Hoc Recovery for Soft Errors

- Detected Uncorrected Errors (DUEs) are common
- Silent Data Corruption (SDC) can occur in new devices
- Handling soft errors differs from handling hard errors
- Several extensions added to FTI to handle soft errors
- MemSave, MemLoad, RecoverLocalVars
- Evaluated with 3 applications







* "*Towards Ad Hoc Recovery for Soft Errors*" - Nuria Losada, Leonardo Bautista-Gomez, Kai Keller, Osman Unsal - 2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (**FTXS'18**)