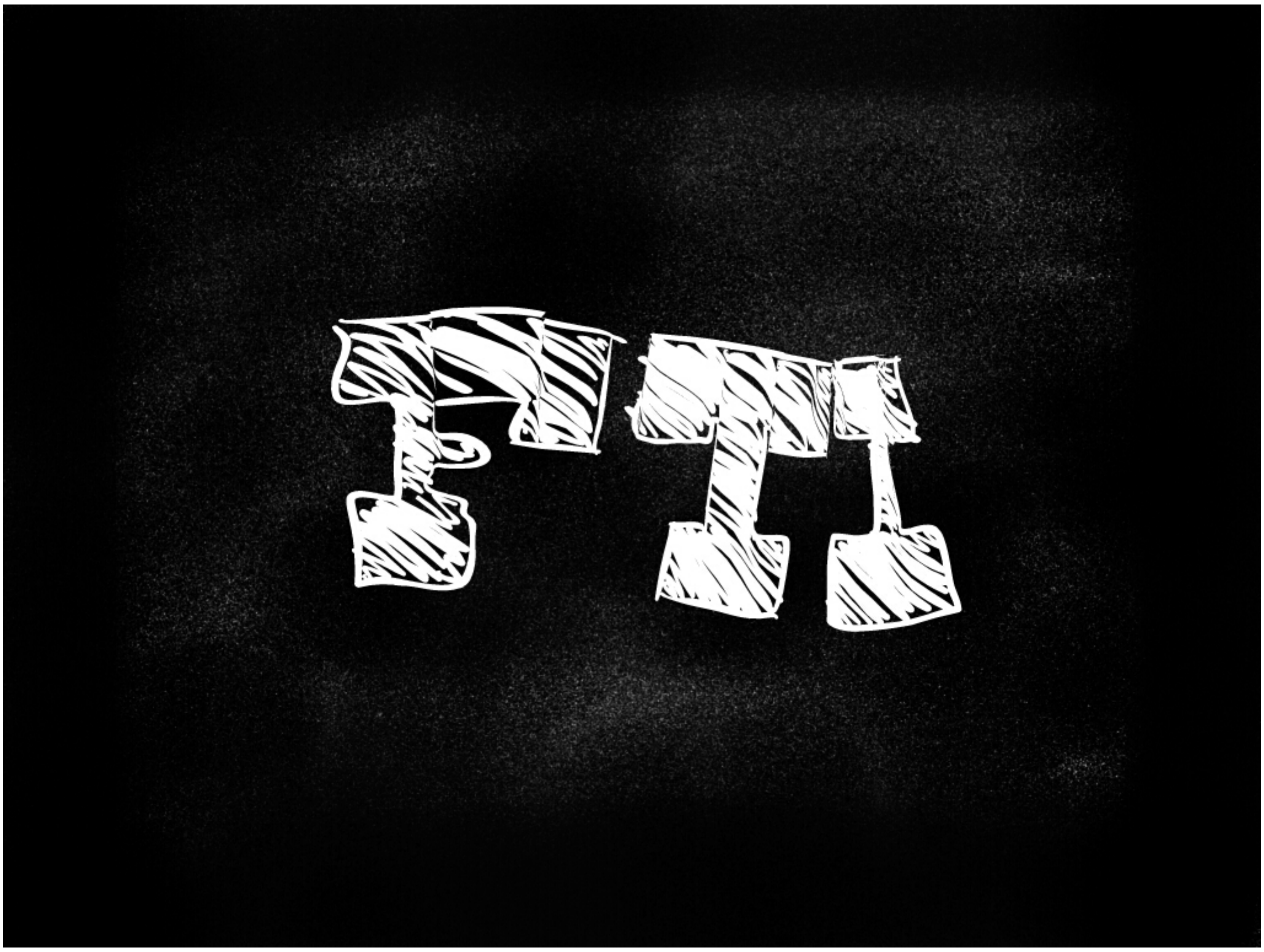
USER-MANUAL

FTI - Fault Tolerance Interface

Contact:
Dr. Leonardo Bautista-Gomez (Leo) - leonardo.bautista@bsc.es
Barcelona Supercomputing Center
Carrer de Jordi Girona, 29-31, 08034 Barcelona, SPAIN
Phone : +34 934 13 77 16



Authors: Kai Keller, Tomasz Paluszkiewicz, Karol Sierocinski

Release: 0.9.9

Introduction

In high performance computing (HPC), systems are built from highly reliable components. However, the overall failure rate of supercomputers increases with component count. Nowadays, petascale machines have a mean time between failures (MTBF) measured in hours or days and fault tolerance (FT) is a well-known issue. Long running large applications rely on FT techniques to successfully finish their long executions. Checkpoint/Restart (CR) is a popular technique in which the applications save their state in stable storage, frequently a parallel file system (PFS); upon a failure, the application restarts from the last saved checkpoint. CR is a relatively inexpensive technique in comparison with the process-replication scheme that imposes over 100% of overhead.

However, when a large application is checkpointed, tens of thousands of processes will each write several GBs of data and the total checkpoint size will be in the order of several tens of TBs. Since the I/O bandwidth of supercomputers does not increase at the same speed as computational capabilities, large checkpoints can lead to an I/O bottleneck, which causes up to 25% of overhead in current petascale systems. Post-petascale systems will have a significantly larger number of components and an important amount of memory. This will have an impact on the system's reliability. With a shorter MTBF, those systems may require a higher checkpoint frequency and at the same time they will have significantly larger amounts of data to save. Although the overall failure rate of future post-petascale systems is a common factor to study when designing FT-techniques, another important point to take into account is the pattern of the failures. Indeed, when moving from 90nm to 16nm technology, the soft error rate (SER) is likely to increase significantly, as shown in a recent study from Intel. A recent study by Dong et al. explains how this provides an opportunity for local/global hybrid checkpoint using new technologies such as phase change memories (PCM). Moreover, some hard failures can be tolerated using solid-state-drives (SSD) and cross-node redundancy schemes, such as checkpoint replication or XOR encoding which allows to leverage multi-level checkpointing, as proposed by Moody et al.. Furthermore, Cheng et al. demonstrated that more complex erasure codes such as Reed-Solomon (RS) encoding can be used to further increase the percentage of hard failures tolerated without stressing the PFS.

FTI is a multi-level checkpointing interface. It provides an api which is easy to apply and offers a flexible configuration to enable the user to select the checkpointing strategy which fits best to the problem.

L1

L1 denotes the first safety level in the multilevel checkpointing strategy of FTI. The checkpoint of each process is written on the local SSD of the respective node. This is fast but possesses the drawback, that in case of a data loss and corrupted checkpoint data even in only one node, the execution cannot successfully restarted.

L2

L2 denotes the second safety level of checkpointing. On initialisation, FTI creates a virtual ring for each group of nodes with user defined size (see group_size). The first step of L2 is just a L1 checkpoint. In the second step, the checkpoints are duplicated and the copies stored on the neighbouring node in the group.

That means, in case of a failure and data loss in the nodes, the execution still can be successfully restarted, as long as the data loss does not happen on two neighbouring nodes at the same time.

L3

L3 denotes the third safety level of checkpointing. In this level, the check- point data trunks from each node getting encoded via the Reed-Solomon (RS) erasure code. The implementation in FTI can tolerate the breakdown and data loss in half of the nodes.

In contrast to the safety level L2, in level L3 it is irrelevant which of nodes encounters the failure. The missing data can get reconstructed from the remaining RS-encoded data files.

L4

L4 denotes the fourth safety level of checkpointing. All the checkpoint files are flushed to the parallel file system (PFS).

Compilation

FTI uses Cmake to configure the installation. The recommended way to perform the installation is to create a build directory within the base directory of FTI and perform the cmake command in there. In the following you will find configuration examples. The commands are performed in the build directory within the FTI base directory.

Default The default configuration builds the FTI library with Fortran and MPI-IO support for GNU compilers:

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=/install/here/fti ..
make all install
```

> Notice: THE TWO DOTS AT THE END INVOKE CMAKE IN THE TOP LEVEL DIRECTORY.

Intel compilers Fortran and MPI-IO support for Intel compilers:

```
cmake -C ../intel.cmake -DCMAKE_INSTALL_PREFIX:PATH=/install/here/fti ..
make all install
```

Disable Fortran Only build FTI C library:

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=/install/here/fti -DENABLE_FORTRAN=OFF ..
make all install
```

Lustre For Lustre user who want to use MPI-IO, it is strongly recommended to configure with Lustre support:

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=/install/here/fti -DENABLE_LUSTRE=ON ..
make all install
```

Cray For Cray systems, make sure that the modules craype/* and PrgEnv* are loaded (if available). The configuration should be done as:

```
export CRAY_CPU_TARGET=x86-64
export CRAYPE_LINK_TYPE=dynamic
cmake -DCMAKE_INSTALL_PREFIX:PATH=/install/here/fti -DCMAKE_SYSTEM_NAME=CrayLinuxEnvironment ..
make all install
```

> Notice: MODIFY x86-64 IF YOU ARE USING A DIFFERENT ARCHITECTURE. ALSO, THE OPTION `CMAKE_SYSTEM_NAME=CrayLinuxEnvironment` IS AVAILABLE ONLY FOR CMAKE VERSIONS 3.5.2 AND ABOVE.

FTI Datatypes and Constants

FTI Datatypes

`FTI_CHAR` : FTI data type for chars.
`FTI_SHRT` : FTI data type for short integers.
`FTI_INTG` : FTI data type for integers.
`FTI_LONG` : FTI data type for long integers.
`FTI_UCHR` : FTI data type for unsigned chars.
`FTI_USHT` : FTI data type for unsigned short integers.
`FTI_UINT` : FTI data type for unsigned integers.
`FTI_ULNG` : FTI data type for unsigned long integers.
`FTI_SFLT` : FTI data type for single floating point.
`FTI_DBLE` : FTI data type for double floating point.
`FTI_LDBE` : FTI data type for long double floating point.

FTI Constants

`FTI_BUFS` : 256
`FTI_DONE` : 1
`FTI_SCES` : 0
`FTI_NSCS` : -1
`FTI_NREC` : -2

FTI_Init

> - Reads configuration file.
> - Creates checkpoint directories.
> - Detects topology of the system.
> - Regenerates data upon recovery.

DEFINITION

```
int FTI_Init ( char * configFile , MPI_Comm globalComm )
```

INPUT

Variable	What for?
`char * configFile`	Path to the config file
`MPI_Comm globalComm`	MPI communicator used for the execution

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No Success
`FTI_NREC`	FTI could not recover ckpt files

DESCRIPTION

`FTI_Init` initializes the FTI context. It must be called before any other FTI function and after `MPI_Init`.

EXAMPLE

```
int main ( int argc , char **argv ) {
    MPI_Init (&argc , &argv );
    char *path = "config.fti"; // config file path
    int res = FTI_Init ( path , MPI_COMM_WORLD );
    if (res == FTI_NREC) {
        printf("Recovery not possible, terminating...");
        FTI_Finalize();
        MPI_Finalize();
        return 1;
    }
.
.
.
    return 0;
}
```

FTI_InitType

> - Initializes a data type.

DEFINITION

```
int FTI_InitType ( FTIT_type *type , int size )
```

INPUT

Variable	What for?
`FTIT_type * type`	The data-type to be initialized
`int size`	The size of the data-type to be initialized

OUTPUT

Value	Reason
`FTI_SCES`	Success

DESCRIPTION

FTI_InitType initializes a FTI data-type. A data-type which is not defined by default by FTI (see:FTI Datatypes), must be defined using this function in order to protect variables of that type with FTI_Protect.

EXAMPLE

```
typedef struct A {
    int a;
    int b;
} A;
FTIT_type structAinfo ;
//sizeof sturct is safest due to padding
//in more complex structs
FTI_InitType (&structAinfo , sizeof(A));
```

FTI_Protect

> - Stores metadata concerning the variable to protect.

DEFINITION

```
int FTI_Protect ( int id, void *ptr, long count, FTIT_type type )
```

INPUT

Variable	What for?
`int id`	Unique ID of the variable to protect
`void * ptr`	Pointer to memory address of variable
`long count`	Number of elements at memory address
`FTIT_type type`	FTI data type of variable to protect

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No success

DESCRIPTION

`FTI_Protect` is used to add data fields to the list of protected variables. Data, protected by this function will be stored during a call to FTI_Checkpoint or FTI_Snapshot and restored during a call to FTI_Recover.

If the dimension of a protected variable changes during the execution, a subsequent call to `FTI_Protect` will update the meta-data whithin FTI in order to store the correct size during a successive call to FTI_Checkpoint or FTI_Snapshot.

EXAMPLE

```
int A;
float *B = malloc (sizeof(float) * 10) ;
FTI_Protect(1, &A, 1, FTI_INTG );
FTI_Protect(2, B, 10, FTI_SFLT );
// changing B size
B = realloc(B, sizeof(float) * 20) ;
// updating B size in protected list
FTI_Protect(2, B, 20, FTI_SFLT);
```

FTI_GetStoredSize

> - Returns size of protected variable saved in metadata

DEFINITION

```
long FTI_GetStoredSize ( int id )
```

INPUT

Variable	What for?
`int id`	ID of the protected variable

OUTPUT

Value	Reason
`long`	Size of a variable
`0`	No success

DESCRIPTION

`FTI_GetStoredSize` returns the size of a protected variable with `id` from the FTI metadata. The result may differ from the size of the variable known to the application at that moment. If the function is called on a restart, it returns the size stored in the metadata file. Called during the execution, it returns the value stored in the FTI runtime metadata, i.e. the size of the variable at the moment of the last checkpoint.

The function is needed to manually reallocate memory for protected variables with variable size on a recovery. Another possibility for the reallocation of memory is provided by FTI_Realloc.

EXAMPLE

```
...
long* array = calloc(arraySize, sizeof(long));
FTI_Protect(1, array, arraySize, FTI_LONG);
if (FTI_Status() != 0) {
    long arraySizeInBytes = FTI_GetStoredSize(1);
    if (arraySizeInBytes == 0) {
            printf("No stored size in metadata!\n");
            return GETSTOREDSIZE_FAILED;
    }
    array = realloc(array, arraySizeInBytes);
    int res = FTI_Recover();
    if (res != 0) {
        printf("Recovery failed!\n");
        return RECOVERY_FAILED;
    }
    //update arraySize
    arraySize = arraySizeInBytes / sizeof(long);
}
for (i = 0; i < max; i++) {
    if (i % CKTP_STEP) {
        //update FTI array size information
        FTI_Protect(1, array, arraySize, FTI_LONG);
        int res = FTI_Checkpoint((i % CKTP_STEP) + 1, 1);
        if (res != FTI_DONE) {
            printf("Checkpoint failed!.\n");
            return CHECKPOINT_FAILED;
        }
    }
    ...
    //add element to array
    arraySize += 1;
    array = realloc(array, arraySize * sizeof(long));
}
...
```

FTI_Realloc

> Reallocates dataset to last checkpoint size.

DEFINITION

```
void* FTI_Realloc ( int id, void* ptr )
```

INPUT

Variable	What for?
`int id`	ID of the protected variable
`void * ptr`	Pointer to memory address of variable

OUTPUT

Value	Reason
`void*`	Pointer to reallocated data
`NULL`	On failure

DESCRIPTION

`FTI_Realloc` is called for protected variables with dynamic size on recovery. It reallocates sufficient memory to store the checkpoint data to the pointed memory address. It must be called before FTI_Recover to prevent segmentation faults. If the reallocation must/is wanted to be done within the application, FTI provides the function FTI_GetStoredSize to request the variable size of the checkpoint to recover.

EXAMPLE

```
...
FTI_Protect(1, &arraySize, 1, FTI_INTG);
long* array = calloc(arraySize, sizeof(long));
FTI_Protect(2, array, arraySize, FTI_LONG);
if (FTI_Status() != 0) {
    array = FTI_Realloc(2, array);
    if (array == NULL) {
            printf("Reallocation failed!\n");
            return REALLOC_FAILED;
    }

    int res = FTI_Recover();
    if (res != 0) {
        printf("Recovery failed!\n");
        return RECOVERY_FAILED;
    }
}
for (i = 0; i < max; i++) {
    if (i % CKTP_STEP) {
        //update FTI array size information
        FTI_Protect(2, array, arraySize, FTI_LONG);
        int res = FTI_Checkpoint((i % CKTP_STEP) + 1, 1);
        if (res != FTI_DONE) {
            printf("Checkpoint failed!.\n");
            return CHECKPOINT_FAILED;
        }
    }
    ...
    //add element to array
    arraySize += 1;
    array = realloc(array, arraySize * sizeof(long));
}
...
```

FTI_Checkpoint

> - Stores protected variables in the checkpoint of a desired safety level.

DEFINITION

```
int FTI_Checkpoint( int id, int level )
```

INPUT

Variable	What for?
`int id`	Unique checkpoint ID
`int level`	Checkpoint level (1=L1, 2=L2, 3=L3, 4=L4)

OUTPUT

Value	Reason
`FTI_DONE`	Success
`FTI_NSCS`	Failure

DESCRIPTION

`FTI_Checkpoint` is used to store the current values of protected variables into a checkpoint of safety level `level` (see Multilevel-Checkpointing for descritions of the particular levels).

> **NOTICE:** The checkpoint id must be different from 0!

EXAMPLE

```
int i;
for (i = 0; i < 100; i ++) {
    if (i % 10 == 0) {
        FTI_Checkpoint ( i /10 + 1, 1) ;
    }
.
. // some computations
.
}
```

FTI_Status

> - Returns the current status of the recovery flag.

DEFINITION

```
int FTI_Status()
```

OUTPUT

Value	Reason
`int 0`	No checkpoints taken yet or recovered successfully
`int 1`	At least one checkpoint is taken. If execution fails, the next start will be a restart
`int 2`	The execution is a restart from checkpoint level L4 and `keep_last_checkpoint` was enabled during the last execution

DESCRIPTION

`FTI_Status` returns the current status of the recovery flag.

EXAMPLE

```
if ( FTI_Status () != 0) {
    .
    . // this section will be executed during restart
    .
}
```

FTI_Recover

> - Recovers the data of the protected variables from the checkpoint file.

DEFINITION

```
int FTI_Recover()
```

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	Failure

DESCRIPTION

`FTI_Recover` loads the data from the checkpoint file to the protected variables. It only recovers variables which are protected by a preceeding call to FTI_Protect. If a variable changes its size during execution, the proper amount of memory has to be allocated for that variable before the call to `FTI_Recover`. FTI provides the API functionsFTI_GetStoredSize and

EXAMPLE

Basic example:

```
if ( FTI_Status() == 1 ) {
    FTI_Recover() ;
}
```

FTI_Snapshot

> - Invokes the recovery of protected variables on a restart.
> - Writes multilevel checkpoints regarding their requested frequencies during execution.

DEFINITION

```
int FTI_Snapshot()
```

OUTPUT

Value	Reason
`FTI_SCES`	Successfull call (without checkpointing) or if recovery successful
`FTI_NSCS`	Failure of `FTI_Checkpoint`
`FTI_DONE`	Success of `FTI_Checkpoint`
`FTI_NREC`	Failure on recovery

DESCRIPTION

On a restart, `FTI_Snapshot` loads the data from the checkpoint file to the protected variables. During execution it performs checkpoints according to the checkpoint frequencies for the various safety levels. The frequencies may be set in the configuration file (see e.g.: ckpt_L1).

`FTI_Snapshot` can only take care of variables which are protected by a preceding call to FTI_Protect.

EXAMPLE

```
int res = FTI_Snapshot();
if ( res == FTI_SCES ) {
    .
    . // executed after successful recover
    . // or when checkpoint is not required
}
else { // res == FTI_DONE
    .
    . // executed after successful checkpointing
    .
}
```

FTI_GetStageDir

> - Returns the local staging directory.

DEFINITION

```
int FTI_GetStageDir ( char* stageDir, int maxLen )
```

INPUT

Variable	What for?
`int maxLen`	The length of the string buffer `stageDir`

OUTPUT

Value	Reason
`char * stageDir`	Path to the local staging directory

DESCRIPTION

`FTI_GetStageDir` initializes the string `stageDir` with the path to the local stage directory. This is a directory in the local ckpt path, set by the user in the configuration file.

EXAMPLE

see example for FTI_SendFile

FTI_GetStageStatus

> - Returns the status of the stage request.

DEFINITION

```
int FTI_GetStageStatus ( int ID )
```

INPUT

Variable	What for?
`int ID`	Request ID (returned from FTI_SendFile)

OUTPUT

Value	Reason
`int FTI_SI_PEND`	Head is occupied and request is pending
`int FTI_SI_ACTV`	Head is processing the request
`int FTI_SI_SCES`	Request was successfully processed
`int FTI_SI_FAIL`	Request failed
`int FTI_SI_NINI`	Request does not exist or was already processed

DESCRIPTION

`FTI_GetStageStatus` queries the status of the staging request. If the request was successful or failed, the function returns the respective status and resets the ID in order to allow the reassignment. I.e., after the function returns `FTI_SI_SCES` or `FTI_SI_FAIL`, a consecutive call with the same ID retuns `FTI_SI_NINI`.

EXAMPLE

see example for FTI_SendFile

FTI_SendFile

> - Triggers the asynchronous transfer of local file to the PFS.

DEFINITION

```
int FTI_SendFile( char* lpath, char *rpath )
```

OUTPUT

Value	Reason
`int ID`	On success, the request ID is returned. This ID may be used to query the status of the request whithin FTI_GetStageStatus
`FTI_NSCS`	On failure

DESCRIPTION

The user may store files local on the nodes to a fast storage layer (e.g. NVMe) and send these files to the PFS asynchronously to the execution. The transfer is performed by the FTI head process. Thus, in order to use this feature, the head feature must be enabled. Is the head feature disabled, the files are send by the calling process itself.

EXAMPLE

```
#include "fti.h"
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>


int main() {

    MPI_Init(NULL,NULL);
    FTI_Init("config.fti", MPI_COMM_WORLD);
```

```
    int rank;
    MPI_Comm_rank( FTI_COMM_WORLD, &rank );

    char local_dir[512];
    char remote_dir[] = "./";

    // get local stage directory
    if ( FTI_GetStageDir( local_dir, 512 ) != FTI_SCES ) {
        fprintf( stderr, "Failed to get the local directory.\n" );
        exit( EXIT_FAILURE );
    }


    char filename[512];
    snprintf( filename, 512, "testfile-%d", rank );

    char local_fn[512];
    char remote_fn[512];

    snprintf( local_fn, 512, "%s/%s", local_dir, filename );
    snprintf( remote_fn, 512, "%s/%s", remote_dir, filename );

    // crate local dummy file (1MB)
    FILE *fstream = fopen( local_fn, "wb+" );
    fsync(fileno(fstream));
    fclose( fstream );
    truncate( local_fn, 1024L*1024L );

    int reqID;
    // send local file to PFS
    if ( (reqID = FTI_SendFile( local_fn, remote_fn )) == FTI_NSCS ) {
        fprintf( stderr, "Failed to stage %s.", local_fn );
        exit( EXIT_FAILURE );
    }

    // check status of staging request
    int reqStatus = FTI_SI_NINI; // set status to not initialized (null)
    while( 1 ) {
        int request_final = 0;
        reqStatus = FTI_GetStageStatus( reqID );
        switch( reqStatus ) {
            case FTI_SI_ACTV:
                printf("Stage Status: ACTIVE\n");
                break;
            case FTI_SI_PEND:
                printf("Stage Status: PENDING\n");
                break;
            case FTI_SI_SCES:
                printf("Stage Status: SUCCESS\n");
                request_final = 1;
                break;
            case FTI_SI_FAIL:
                printf("Stage Status: FAILED\n");
                request_final = -1;
                break;
        }
        if ( request_final == -1) {
            fprintf( stderr, "Staging request with ID: %d failed!\n", reqID );
            break;
        }
        if ( request_final == 1) {
            printf( "Staging request with ID: %d succeed!\n", reqID );
            break;
        }
    }

    FTI_Finalize();
    MPI_Finalize();

    exit( EXIT_SUCCESS );

}
```

FTI_Finalize

> - Frees the allocated memory.
> - Communicates the end of the execution to dedicated threads.
> - Cleans checkpoints and metadata.

DEFINITION

```
int FTI_Finalize()
```

OUTPUT

Value	Reason
`FTI_SCES`	For application process
`exit(0)`	For FTI process

DESCRIPTION

`FTI_Finalize` notifies the FTI processes that the execution is over, frees FTI internal data structures and it performs a clean up of the checkpoint folders at a normal execution. If the setting `keep_last_ckpt` is set, it flushes local checkpoint files (if present) to the PFS. If the setting `head` is set to 1, it will also terminate the FTI processes. It should be called before `MPI_Finalize()`.

EXAMPLE

```
int main ( int argc , char ** argv ) {
    .
    .
    .
    FTI_Finalize () ;
    MPI_Finalize () ;
    return 0;
}
```

HDF5-Routines

FTI_InitType

DESCRIPTION

`FTI_InitType` initializes a FTI data-type that will be treated as binary data in hdf5 file (see:FTI_InitType for more information).

EXAMPLE

```
typedef struct A {
    int a;
    int b;
} A;
FTIT_type structAinfo ;
//sizeof sturct is safest due to padding
//in more complex structs
FTI_InitType(&structAinfo , sizeof(A));
```

FTI_InitComplexType

> - Initializes a complex data type with structure information for hdf5 file.

DEFINITION

```
int FTI_InitComplexType ( FTIT_type *newType, FTIT_complexType *typeDefinition, int length, size_t size, char
```

INPUT

Variable	What for?
`FTIT_type * newType`	The data-type to be initialized
`FTIT_complexType * typeDefinition`	The definition of the data-type to be initialized
`int length`	Number of fields in a structure
`size_t size`	The size of a structure
`char * name`	Name of the new datatype in hdf5file.

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No success

DESCRIPTION

`FTI_InitComplexType` initializes a complex FTI data-type with structure information for hdf5 file.
If `name` is set to `NULL` FTI will set default value (`"Type{id}"`).

EXAMPLE

```
typedef struct A {
    int a;
    int b;
} A;

typedef struct B {
    A structA;
    int c[5][4];
} B;

FTIT_complexType structAdef;
addSimpleField(&structAdef, &FTI_INTG, offsetof(A, a), 0, "int a");
addSimpleField(&structAdef, &FTI_INTG, offsetof(A, b), 1, "int b");

FTIT_type structAinfo;
FTI_InitComplexType(&structAinfo , &structAdef), 2, sizeof(A), "struct A", NULL);

FTIT_complexType structBdef;
addSimpleField(&structBdef, &structAinfo, offsetof(B, structA), 0, "A structA");
int dimLength[] = {5, 4};
addComplexField(&structBdef, &FTI_INTG, offsetof(B, c), 2, dimLength, 1, "int array c");

FTIT_type structBinfo;
FTI_InitComplexType(&structBinfo , &structBdef, 2, sizeof(B), "struct B", NULL);
```

FTI_AddSimpleField

> - Adds information about a field to a type definition.
> - Simple field is a field that isn't an array.

DEFINITION

```
void FTI_AddSimpleField ( FTIT_complexType* typeDefinition, FTIT_type* ftiType, size_t offset, int id, char*
```

INPUT

Variable	What for?
`FTIT_complexType * typeDefinition`	The definition of the data-type to add field
`FTIT_type * ftiType`	The definition of the data-type to be initialized
`size_t offset`	Offset of the field in the structure
`int id`	Id of the field
`char * name`	Name of the field in hdf5file.

DESCRIPTION

`FTI_AddSimpleField` adds information about a simple field in complex datatype to a type definition. If user pass`NULL` to `name` then default name is used. If given id was previously added, the field with that id is updated.

EXAMPLE

```
typedef struct A {
    int a;
    int b;
} A;

FTIT_complexType structAdef;

FTI_AddSimpleField(&structAdef, $FTI_INTG, offsetof(A, a), 0, "int a");
FTI_AddSimpleField(&structAdef, $FTI_INTG, offsetof(A, b), 1, "int b");

FTIT_type structAinfo;
FTI_InitComplexType(&structAinfo , &structAdef, 2, sizeof(A), "struct A", NULL);
```

FTI_AddComplexField

> - Adds information about a field to a type definition.
> - Complex field is a field that is an array.

DEFINITION

```
void FTI_AddComplexField ( FTIT_complexType* typeDefinition, FTIT_type* ftiType, size_t offset, int rank, int
```

INPUT

Variable	What for?
`FTIT_complexType * typeDefinition`	The definition of the data-type to add field
`FTIT_type * ftiType`	The definition of the data-type to be initialized
`size_t offset`	Offset of the field in the structure
`int rank`	Number of dimensions
`int * dimLength`	Length of each dimension
`int id`	Id of the field
`char * name`	Name of the field in hdf5file.

DESCRIPTION

`FTI_AddComplexField` adds information about a complex field in complex datatype to a type definition. Complex field is a field that is an array. If user pass `NULL` to `name` then default name is used. If given id was previously added, the field with that id is updated.

EXAMPLE

```
typedef struct A {
    int a;
    int b[5][4];
} A;

FTIT_complexType structAdef;

FTI_AddSimpleField(&structAdef, $FTI_INTG, offsetof(A, a), 0, "int a");
int dimLength[] = {5, 4};
FTI_AddComplexField(&structAdef, $FTI_INTG, offsetof(A, b), 2, &dimLength, 1, "int array b");

FTIT_type structAinfo;
FTI_InitComplexType(&structAinfo , &structAdef, 2, sizeof(A), "struct A", NULL);
```

FTI_InitGroup

> - Initializes information about a hdf5 group.
> - If parent is set to `NULL`, parent will be set to the root group.

DEFINITION

```
int FTI_InitGroup ( FTIT_H5Group* h5group, char* name, FTIT_H5Group* parent )
```

INPUT

Variable	What for?
`FTIT_H5Group * h5group`	The definition of the group to add field
`char * name`	Name of the group to be initialized
`FTIT_H5Group * parent`	The parent group of the group

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No success

DESCRIPTION

`FTI_InitGroup` inits a hdf5 group with given name and parent group. If parent group is set to`NULL`, then the root group is the parent of this group.

EXAMPLE

```
FTIT_H5Group group1;
FTIT_H5Group group2;
//root is the parent of group1
FTI_InitGroup(&group1, "Group 1", NULL);
//group1 is the parent of group2
FTI_InitGroup(&group2, "Group 2", &group1);
```

FTI_RenameGroup

> - Renames a group.

DEFINITION

```
int FTI_RenameGroup ( FTIT_H5Group* h5group, char* name )
```

INPUT

Variable	What for?
`FTIT_H5Group * h5group`	Group to change name
`char * name`	New name of the group

OUTPUT

Value	Reason
`FTI_SCES`	Success

DESCRIPTION

`FTI_RenameGroup` renames given group to the given name.

EXAMPLE

```
FTIT_H5Group group;
//group name is "Group"
FTI_InitGroup(&group, "Group", NULL);
//group name is now "New name"
FTI_RenameGroup(&group, "New name");
```

FTI_DefineDataset

> - Defines a dataset.

DEFINITION

```
int FTI_DefineDataset ( int id, int rank, int* dimLength, char* name, FTIT_H5Group* h5group )
```

INPUT

Variable	What for?
`int id`	Id of the dataset
`int rank`	Rank of the array
`int * dimLength`	Length of each dimension
`char * name`	New name of the dataset
`FTIT_H5Group * h5group`	Group of the dataset

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No success

DESCRIPTION

`FTI_DefineDataset` gives FTI all information needed by HDF5 to correctly save the dataset in the checkpoint file. If the h5group is set to `NULL`, dataset is assigned to the root group.

EXAMPLE

```
int x;
int y[5][5];
FTIT_H5Group group;
FTI_InitGroup(&group, "Group", NULL);

FTI_Protect(1, &x, 1, FTI_INTG);
FTI_DefineDataset(1, 0, NULL, "single int", NULL);

FTI_Protect(2, &y, 25, FTI_INTG);
int dimLength[] = {5, 5};
FTI_DefineDataset(2, 2, dimLength, "single int", &group);
```

FTI_Protect

> - Stores metadata concerning the variable to protect.
> - Name of a variable is set to default.

DESCRIPTION

`FTI_Protect` (see: FTI_Protect for more information) is used to add data fields to the list of protected variables.

EXAMPLE

```
typedef struct A {
    int a;
    int b;
} A;

FTIT_complexType structAdef;
FTIT_type structAinfo;
.
.
.
FTI_InitComplexType(&structAinfo , &structAdef, 2, sizeof(A), NULL, NULL);
A someVariable;
FTI_Protect(1, &someVariable, structAinfo);
```

FTI_ProtectWithName

> - Stores metadata concerning the variable to protect.
> - Name of a variable is given by user.

DEFINITION

```
int FTI_ProtectWithName ( int id, void *ptr, long count, FTIT_type type, char *name )
```

INPUT

Variable	What for?
`int id`	Unique ID of the variable to protect
`void * ptr`	Pointer to memory address of variable
`long count`	Number of elements at memory address
`FTIT_type type`	FTI data type of variable to protect
`char * name`	Name of the variable to write in hdf5 file

OUTPUT

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No success

DESCRIPTION

`FTI_ProtectWithName` is used the same way as `FTI_Protect`. The difference is that this function provides a name of variable to protect that will be stored in hdf5 checkpoint file.

EXAMPLE

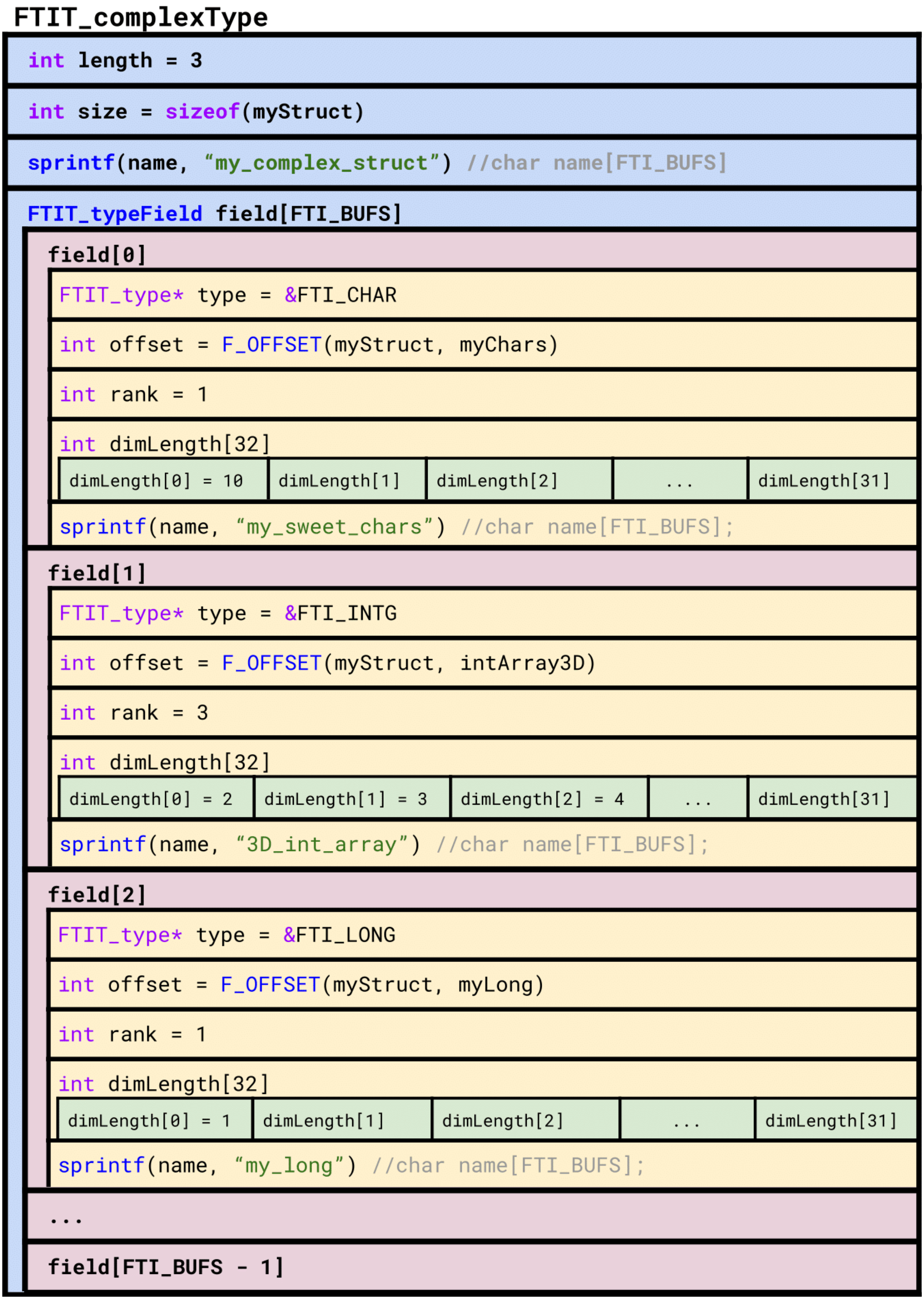
```
typedef struct A {
    int a;
    int b;
} A;

FTIT_complexType structAdef;
FTIT_type structAinfo;
.
.
.
FTI_InitComplexType(&structAinfo , &structAdef, 2, sizeof(A), "struct A", NULL);
A someVariable;
FTI_ProtectWithName(1, &someVariable, structAinfo, "someVariable");
```

HDF5 struct example

Code of the struct in example:

```
struct myStruct {
    char myChars[10];
    int intArray3D[2][3][4];
    long myLong;
};

FTIT_complexType myStructDef;
int dimLength[3];

dimLength[0] = 10;
addComplexField(&myStructDef, &FTI_CHAR, offsetof(struct myStruct, myChars), 1, dimLength, 0, "my_sweet_chars

dimLength[0] = 2;
dimLength[1] = 3;
dimLength[2] = 4;
addComplexField(&myStructDef, &FTI_INTG, offsetof(struct myStruct, intArray3D), 3, dimLength, 1, "3D_int_arra

addSimpleField(&myStructDef, &FTI_LONG, offsetof(struct myStruct, myLong), 2, "my_long");

FTIT_type myStructInfo;
InitComplexType(&myStructInfo, &myStructDef, 3, sizeof(struct myStruct), "my_complex_struct", NULL);
```

Graphical representation of FTIT_complexType myStructDef:



```
FTIT_complexType
int length = 3
int size = sizeof(myStruct)
sprintf(name, “my_complex_struct”) //char name[FTI_BUFS]
FTIT_typeField field[FTI_BUFS]
    field[0]
        FTIT_type* type = &FTI_CHAR
        int offset = F_OFFSET(myStruct, myChars)
        int rank = 1
        int dimLength[32]
            dimLength[0] = 10 | dimLength[1] | dimLength[2] | ... | dimLength[31]
        sprintf(name, “my_sweet_chars”) //char name[FTI_BUFS];
    field[1]
        FTIT_type* type = &FTI_INTG
        int offset = F_OFFSET(myStruct, intArray3D)
        int rank = 3
        int dimLength[32]
            dimLength[0] = 2 | dimLength[1] = 3 | dimLength[2] = 4 | ... | dimLength[31]
        sprintf(name, “3D_int_array”) //char name[FTI_BUFS];
    field[2]
        FTIT_type* type = &FTI_LONG
        int offset = F_OFFSET(myStruct, myLong)
        int rank = 1
        int dimLength[32]
            dimLength[0] = 1 | dimLength[1] | dimLength[2] | ... | dimLength[31]
        sprintf(name, “my_long”) //char name[FTI_BUFS];
    ...
    field[FTI_BUFS - 1]
```

Example:

```
struct myStruct example;
sprintf(example.myChars, "simplestr");
int i, j, k;
for (i = 0; i < 2; i++)
    for (j = 0; j < 3; j++)
        for(k = 0; k < 4; k++)
           example.intArray3D[i][j][k] = 100 * i  + 10 * j + k;
example.myLong = 1234560;
FTI_ProtectWithName(1, &example, 1, myStructInfo, "example");
```

OUTPUT:

```
HDF5 "Type2.h5" {
GROUP "/" {
   GROUP "dataset" {
      DATASET "example" {
         DATATYPE  "/datatype/my_complex_struct"
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): {
               [ 115, 105, 109, 112, 108, 101, 115, 116, 114, 0 ],
               [ 0, 1, 2, 3,
                  10, 11, 12, 13,
                  20, 21, 22, 23,
                  100, 101, 102, 103,
                  110, 111, 112, 113,
                  120, 121, 122, 123 ],
               1234560
            }
         }
      }
   }
   GROUP "datatype" {
      DATATYPE "my_complex_struct" H5T_COMPOUND {
         H5T_ARRAY { [10] H5T_STD_I8LE } "my_sweet_chars";
         H5T_ARRAY { [2][3][4] H5T_STD_I32LE } "3D_int_array";
         H5T_STD_I64LE "my_long";
      }
   }
}
}
```

Fortran-API-Reference

FTI Datatypes and Constants

FTI Datatypes

FTI datatypes are used in the C-API function `FTI_Protect`. With the `count` parameter and the datatype, FTI is able to determine the size of the allocated memory region at `ptr`.

The FTI Fortran interface defines a template of `FTI_Protect` for all intrinsic data types. Hence the datatype definitions are not necessary here and are not avalable for the Fortran interface.

FTI Constants

`FTI_BUFS` : 256
`FTI_DONE` : 1
`FTI_SCES` : 0
`FTI_NSCS` : -1
`FTI_NREC` : -2

FTI_Init

> - Reads configuration file.
> - Creates checkpoint directories.
> - Detects topology of the system.
> - Regenerates data upon recovery.

DEFINITION

```
subroutine FTI_Init ( config_file, global_comm, err )
```

ARGUMENTS

Variable		What for?
`character` `config_file`	IN	Path to the config file
`integer` `global_comm`	IN/OUT	MPI communicator used for the execution
`integer` `err`	OUT	Token for FTI error code.

ERROR HANDLING

ierr	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No Success
`FTI_NREC`	FTI could not recover ckpt files

DESCRIPTION

`FTI_Init` initializes the FTI context. It must be called before any other FTI function and after `MPI_Init`. The MPI communicator passed, must be declared as `integer, target`.

EXAMPLE

```
integer, target :: rank, nbProcs, err, FTI_comm_world

call MPI_Init(err)
FTI_comm_world = MPI_COMM_WORLD
call FTI_Init('config.fti', FTI_comm_world, err) ! modifies FTI_comm_world
call MPI_Comm_size(FTI_comm_world, nbProcs, err)
call MPI_Comm_rank(FTI_comm_world, rank, err)
```

FTI_InitType

> - Initializes a data type.

DEFINITION

```
subroutine FTI_InitType ( type_F, size_F, err )
```

ARGUMENTS

Variable		What for?
`type(FTI_type) type_F`	IN	The data type to be initialized
`integer size_F`	IN	The size of the data type to be initialized
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

err	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No Success

DESCRIPTION

FTI_InitType initializes a FTI data-type. A data-type which is not Fortran intrinsic, must be defined using this function in order to protect variables of that type with FTI_Protect.

EXAMPLE

```
!...

type polar
    real :: radius
    real :: phi
end type

type(FTI_Type)              :: FTI_Polar

type(polar), target         :: choord
type(polar), pointer        :: choord_ptr
type(c_ptr)                 :: choord_c_ptr

choord_ptr => choord
choord_c_ptr = c_loc(choord)

! ...

call FTI_InitType(FTI_Polar, int(sizeof(choord),4), ierr)

! ...
```

FTI_Protect

> - Stores metadata concerning the variable to protect.

In the Fortran interface, `FTI_Protect` comes with two different function headers. One may be used for intrinsic Fortran types and the other must be used for derived data-types.

DEFINITION

```
subroutine FTI_Protect ( id, data, err ) !> For intrinsic data-types
subroutine FTI_Protect ( id, data_ptr, count_F, type_F, err ) !> For derived data-types
```

ARGUMENTS (intrinsic types)

Variable		What for?
`integer id`	IN	Unique ID of the variable to protect
`Fortran type, pointer data`	IN	Pointer to memory address of variable
`integer err`	OUT	Token for FTI error code.

ARGUMENTS (derived types)

Variable		What for?
`integer id`	IN	Unique ID of the variable to protect
`type(c_ptr) data_ptr`	IN	Pointer to memory address of variable
`integer count_F`	IN	Number of elements.
`tape(FTI_Type) type_F`	IN	FTI_Type of Derived data-type.
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

err	Reason
`FTI_SCES`	Success
`FTI_NSCS`	Number of protected variables is > `FTI_BUFS`

DESCRIPTION

`FTI_Protect` is used to add data fields to the list of protected variables. Data, protected by this function will be stored during a call to FTI_Checkpoint or FTI_Snapshot and restored during a call to FTI_Recover.

If the dimension of a protected variable changes during the execution, a subsequent call to `FTI_Protect` will update the meta-data whithin FTI in order to store the correct size during a successive call to FTI_Checkpoint or FTI_Snapshot.

EXAMPLE

For Fortran intrinsic data-types:

```
! ...

integer, target :: nbProcs, iter, row, col, err, FTI_comm_world
integer, pointer  :: ptriter
real(8), pointer :: g(:,:)

call MPI_Init(err)
FTI_comm_world = MPI_COMM_WORLD
call FTI_Init('config.fti', FTI_comm_world, err) ! modifies FTI_comm_world
call MPI_Comm_size(FTI_comm_world, nbProcs, err)

row = sqrt((MEM_MB * 1024.0 * 512.0 * nbProcs)/8)

col = (row / nbProcs)+3

allocate( g(row, col) )
allocate( h(row, col) )

! INIT DATA ...

ptriter => iter
call FTI_Protect(0, ptriter, err)
call FTI_Protect(2, g, err)

! ...
```

For derived data-types

```
! ...

use iso_c_binding

type polar
    real :: radius
    real :: phi
end type

type(FTI_Type)              :: FTI_Polar
integer, parameter        :: N=128*1024*25  !> 25 MB / Process
integer, parameter        :: N1 = 128
integer, parameter        :: N2 = 1024
integer, parameter        :: N3 = 25
integer, target             :: FTI_COMM_WORLD
integer                     :: ierr, status

type(polar), dimension(:,:,:), pointer :: arr
type(c_ptr)                            :: arr_c_ptr

allocate(arr(N1,N2,N3))

shape = (/ N1, N2, N3 /)
arr_c_ptr = c_loc( arr( &
    lbound(arr,1), &
    lbound(arr,2), &
    lbound(arr,3)))

!> INITIALIZE MPI AND FTI
call MPI_Init(ierr)
FTI_COMM_WORLD = MPI_COMM_WORLD
call FTI_Init('config.fti', FTI_COMM_WORLD, ierr)

call FTI_InitType(FTI_Polar, int(2*sizeof(1.0),4), ierr)

!> PROTECT DATA AND ITS SHAPE
call FTI_Protect(0, arr_c_ptr, size(arr), FTI_Polar, ierr)

! ...
```

FTI_Checkpoint

> - Writes values of protected runtime variables to a checkpoint file of requested level.

DEFINITION

```
subroutine FTI_Checkpoint ( id_F, level, err )
```

ARGUMENTS

Variable		What for?
`integer id_F`	IN	Unique checkpoint ID
`integer level`	IN	Checkpoint level (1=L1, 2=L2, 3=L3, 4=L4)
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

err	Reason
`FTI_DONE`	Success
`FTI_NSCS`	Failure

DESCRIPTION

`FTI_Checkpoint` is used to store the current values of protected variables into a checkpoint of safety level `level` (see Multilevel-Checkpointing for descritions of the particular levels).

EXAMPLE

The handling is identical to the C case, except that in Fortran it is a*subroutine* and not a function, hence:

```
! ...

!> LEVEL 2 CHECKPOINT, ID = 1
call FTI_Checkpoint(1, 2, err)

! ...
```

FTI_GetStoredSize

> - Delivers the variable size in Bytes of a protected variable. The returned size is consistent to the FTI state, i.e. it might differ to the current variable size in the execution.

DEFINITION

```
subroutine FTI_GetStoredSize ( id_F, size_F )
```

ARGUMENTS

Variable		What for?
`integer id_F`	IN	Unique variable ID
`integer size_F`	OUT	Size of protected variable

ERROR HANDLING

size	Reason
`> 0`	Success
`0`	No size saved

DESCRIPTION

`FTI_GetStoredSize` returns the size of a protected variable with `id` from the FTI metadata. The result may differ from the size of the variable known to the application at that moment. If the function is called on a restart, it returns the size stored in the metadata file. Called during the execution, it returns the value stored in the FTI runtime metadata, i.e. the size of the variable at the moment of the last checkpoint.

The function is needed to manually reallocate memory for protected variables with variable size on a recovery. Another possibility for the reallocation of memory is provided by FTI_Realloc.

EXAMPLE

```
! ...

integer, parameter        :: N1=128*1024*25  !> 25 MB / Process
integer, parameter        :: N2=128*1024*50  !> 50 MB / Process
integer                   :: varSizeMeta
integer, target           :: FTI_COMM_WORLD
integer                   :: ierr, status

real(dp), dimension(:), pointer :: arr
real(dp), dimension(:), pointer :: tmp

allocate(arr(N1))

!> INITIALIZE MPI AND FTI
call MPI_Init(ierr)
FTI_COMM_WORLD = MPI_COMM_WORLD
call FTI_Init('config.fti', FTI_COMM_WORLD, ierr)

!> PROTECT DATA AND ITS SHAPE
call FTI_Protect(0, arr, ierr)

call FTI_Status(status)

!> EXECUTE ON RESTART
if ( status .eq. 1 ) then
    !> REALLOCATE TO SIZE AT CHECKPOINT
    call FTI_GetStoredSize(0, varSizeMeta)
    if(varSizeMeta .ne. sizeof(arr)) then
        deallocate(arr)
        allocate(arr(varSizeMeta))
        call FTI_Protect(0, arr, ierr) ! necessary to pass new address
    end if
    call MPI_Barrier(FTI_COMM_WORLD, ierr)
    call FTI_recover(ierr)

 ! ...

end if

! ...

!> FIRST CHECKPOINT
call FTI_Checkpoint(1, 1, ierr)

! ...

!> CHANGE ARRAY DIMENSION
allocate(tmp(N2))
tmp(1:N11) = arr
deallocate(arr)
arr => tmp

!> TELL FTI ABOUT THE NEW DIMENSION
call FTI_Protect(0, arr, ierr)

! ...

!> SECOND CHECKPOINT
call FTI_Checkpoint(2,1, ierr)

! ...
```

FTI_Realloc

> - Provides the reallocation of memory on FTI API side for protected variables upon a restart.

DEFINITION

```
subroutine FTI_Realloc ( id, data, err ) !> For intrinsic data-types
subroutine FTI_Realloc ( id, data_ptr, err ) !> For derived data-types
```

ARGUMENTS (intrinsic types)

Variable		What for?
`integer id`	IN	Unique ID of the variable to protect
`Fortran type, pointer data`	IN/OUT	Pointer to memory address of variable
`integer err`	OUT	Token for FTI error code.

ARGUMENTS (derived types)

Variable		What for?
`integer id`	IN	Unique ID of the variable to protect
`type(c_ptr) data_ptr`	IN/OUT	Pointer to memory address of variable
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

err	Reason
`FTI_SCES`	Success
`FTI_NSCS`	No success

DESCRIPTION

`FTI_Realloc` is called for protected variables with dynamic size on recovery. It reallocates sufficient memory to store the checkpoint data to the pointed memory address. It must be called before FTI_Recover to prevent segmentation faults. If the reallocation must/is wanted to be done within the application, FTI provides the function FTI_GetStoredSize to request the variable size of the checkpoint to recover.

EXAMPLE

For intrinsic data-types:

```
! ...

integer, parameter       :: N1=128*1024*25  !> 25 MB / Process
integer, parameter       :: N2=128*1024*50  !> 50 MB / Process
integer, parameter       :: N11 = 128
integer, parameter       :: N12 = 1024
integer, parameter       :: N13 = 25
integer, parameter       :: N21 = 128
integer, parameter       :: N22 = 1024
integer, parameter       :: N23 = 50
integer, target          :: FTI_COMM_WORLD
integer                  :: ierr, status

real(dp), dimension(:,:,:), pointer :: arr
type(c_ptr)                :: arr_c_ptr
real(dp), dimension(:,:,:), pointer :: tmp
integer, dimension(:), pointer      :: shape

allocate(arr(N11,N12,N13))
allocate(shape(3))

!> INITIALIZE MPI AND FTI
call MPI_Init(ierr)
FTI_COMM_WORLD = MPI_COMM_WORLD
call FTI_Init('config.fti', FTI_COMM_WORLD, ierr)

!> PROTECT DATA AND ITS SHAPE
call FTI_Protect(0, arr, ierr)
call FTI_Protect(1, shape, ierr)

call FTI_Status(status)

!> EXECUTE ON RESTART
if ( status .eq. 1 ) then
    !> REALLOCATE TO SIZE AT CHECKPOINT
    arr_c_ptr = c_loc(arr(1,1,1))
    call FTI_Realloc(0, arr_c_ptr, ierr)
    call FTI_recover(ierr)
    !> RESHAPE ARRAY
    call c_f_pointer(arr_c_ptr, arr, shape)
    call FTI_Realloc(0, arr, ierr)

 ! ...

end if

! ...

!> FIRST CHECKPOINT
call FTI_Checkpoint(1, 1, ierr)

! ...

!> CHANGE ARRAY DIMENSION
!> AND STORE IN SHAPE ARRAY
shape = [N21,N22,N23]
allocate(tmp(N21,N22,N23))
tmp(1:N11,1:N12,1:N13) = arr
deallocate(arr)
arr => tmp

!> TELL FTI ABOUT THE NEW DIMENSION
call FTI_Protect(0, arr, ierr)

! ...

!> SECOND CHECKPOINT
call FTI_Checkpoint(2,1, ierr)

! ...
```

For derived data-types:

```
! ...

use iso_c_binding

! ...

type polar
    real :: radius
    real :: phi
end type

type(FTI_Type)           :: FTI_Polar
integer, parameter       :: N1=128*102*25  !> 25 MB / Process
integer, parameter       :: N2=128*102*50  !> 50 MB / Process
integer, parameter       :: N11 = 128
integer, parameter       :: N12 = 102
integer, parameter       :: N13 = 25
integer, parameter       :: N21 = 128
integer, parameter       :: N22 = 102
integer, parameter       :: N23 = 50
integer, target          :: FTI_COMM_WORLD
integer                  :: ierr, status

type(polar), dimension(:,:,:), pointer :: arr
type(c_ptr)                            :: arr_c_ptr
type(polar), dimension(:,:,:), pointer :: tmp
integer, dimension(:), pointer         :: shape

allocate(arr(N11,N12,N13))
allocate(shape(3))

!> INITIALIZE C POINTER
arr_c_ptr = c_loc( arr( &
    lbound(arr,1), &
    lbound(arr,2), &
    lbound(arr,3)))

! ...

!> PROTECT DATA AND ITS SHAPE
call FTI_Protect(0, arr_c_ptr, size(arr), FTI_Polar, ierr)
call FTI_Protect(1, shape, ierr)

call FTI_Status(status)

!> EXECUTE ON RESTART
if ( status .eq. 1 ) then
    !> REALLOCATE TO SIZE AT CHECKPOINT
    call FTI_Realloc(0, arr_c_ptr, ierr)
    call FTI_recover(ierr)
    !> RESHAPE ARRAY
    call c_f_pointer(arr_c_ptr, arr, shape)

 ! ...

end if

! ...

!> FIRST CHECKPOINT
call FTI_Checkpoint(1, 1, ierr)

! ...

!> CHANGE ARRAY DIMENSION
!> AND STORE IN SHAPE ARRAY
shape = [N21,N22,N23]
allocate(tmp(N21,N22,N23))
tmp(1:N11,1:N12,1:N13) = arr
deallocate(arr)
arr => tmp

! ...
```

```
!> UPDATE C POINTER BEFORE CALL TO 'FTI_Protect'
arr_c_ptr = c_loc( arr( &
    lbound(arr,1), &
    lbound(arr,2), &
    lbound(arr,3)))

!> TELL FTI ABOUT THE NEW DIMENSION
call FTI_Protect(0, arr_c_ptr, size(arr), FTI_Polar, ierr)

! ...

!> SECOND CHECKPOINT
call FTI_Checkpoint(2,1, ierr)

! ...
```

FTI_Status

> - Returns the current status of the recovery flag.

DEFINITION

```
subroutine FTI_Status ( status )
```

ARGUMENTS

Variable		What for?
`integer status`	OUT	Token for status flag.

OUTPUT

Value	Reason
`0`	No checkpoints taken yet or recovered successfully
`1`	At least one checkpoint is taken. If execution fails, the next start will be a restart
`2`	The execution is a restart from checkpoint level L4 and `keep_last_checkpoint` was enabled during the last execution

DESCRIPTION

`FTI_Status` returns the current status of the recovery flag.

EXAMPLE

```
call FTI_Status(status)

!> EXECUTE ON RESTART
if ( status .eq. 1 ) then

 ! ...

 call FTI_recover(ierr)

 ! ...

end if
```

FTI_Recover

> - Recovers the data of the protected variables from the checkpoint file.

DEFINITION

```
subroutine FTI_Recover ( err )
```

ARGUMENTS

Variable		What for?
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

Value	Reason
`FTI_SCES`	Success
`FTI_NSCS`	Failure

DESCRIPTION

`FTI_Recover` loads the data from the checkpoint file to the protected variables. It only recovers variables which are protected by a preceeding call to FTI_Protect. If a variable changes its size during execution, the proper amount of memory has to be allocated for that variable before the call to `FTI_Recover`. FTI provides the API functionsFTI_GetStoredSize and

EXAMPLE

see example of FTI_Status.

FTI_Snapshot

> - Invokes the recovery of protected variables on a restart.
> - Writes multilevel checkpoints regarding their requested frequencies during execution.

DEFINITION

```
subroutine FTI_Snapshot ( err )
```

ARGUMENTS

Variable		What for?
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

Value	Reason
`FTI_SCES`	Successfull call (without checkpointing) or if recovery successful
`FTI_NSCS`	Failure of `FTI_Checkpoint`
`FTI_DONE`	Success of `FTI_Checkpoint`
`FTI_NREC`	Failure on recovery

DESCRIPTION

On a restart, `FTI_Snapshot` loads the data from the checkpoint file to the protected variables. During execution it performs checkpoints according to the checkpoint frequencies for the various safety levels. The frequencies may be set in the configuration file (see e.g.: ckpt_L1).

`FTI_Snapshot` can only take care of variables which are protected by a preceding call to FTI_Protect.

EXAMPLE

```
! ...

ptriter => iter
call FTI_Protect(0, ptriter, err)
call FTI_Protect(2, g, err)
call FTI_Protect(1, h, err)

do iter = 1, ITER_TIMES

  call FTI_Snapshot(err)

  call doWork(nbProcs, rank, g, h, localerror)

  ! ...

enddo

if ( rank == 0 ) then
  print '("Execution finished in ",F9.0," seconds.")', MPI_Wtime() - wtime
endif

! ...
```

FTI_Finalize

> - Frees the allocated memory.
> - Communicates the end of the execution to dedicated threads.
> - Cleans checkpoints and metadata.

DEFINITION

```
subroutine FTI_Finalize ( err )
```

ARGUMENTS

Variable		What for?
`integer err`	OUT	Token for FTI error code.

ERROR HANDLING

Value	Reason
`FTI_SCES`	For application process
`exit(0)`	For FTI process (only if head == 1)

DESCRIPTION

`FTI_Finalize` notifies the FTI processes that the execution is over, frees FTI internal data structures and it performs a clean up of the checkpoint folders at a normal execution. If the setting `keep_last_ckpt` is set, it flushes local checkpoint files (if present) to the PFS. If the setting `head` is set to 1, it will also terminate the FTI processes. It should be called before `MPI_Finalize()`.

EXAMPLE

```
! ...

deallocate(h)
deallocate(g)

call FTI_Finalize(err)
call MPI_Finalize(err)

! ...
```

FTI-File-Format

FTI File Format (FTI-FF)

Structure

The file format basic structure, consists of a meta block and a data block:

```
    +--------------+ +------------------------+
    |              | |                        |
    | FB           | | VB                     |
    |              | |                        |
    +--------------+ +------------------------+
```

The `FB` (file block) holds meta data related to the file whereas the `VB` (variable block) holds meta and actual data of the variables protected by FTI.

The `FB` has the following structure:

```
    FB {
        checksum         // Hash of the VDB block in hex representation (33 bytes)
        hash             // Hash of FB without 'hash' in unsigned char (16 bytes)
        ckptSize         // Size of actual data stored in file
        fs               // Size of FB + VB
        maxFs            // Maximum size of FB + VB in group
        ptFs             // Size of FB + VB of partner process
        timestamp        // Time in ns of FB block creation
    }
```

The `VB` block possesses the following sub structure:

```
    |<-------------------------------------------------- VB --------------------------------------------------
    #
    |<------------- VCB_1 --------------------------->|      |<------------- VCB_1 ---------------------------
    #                                                 #      #
    +-------------------------------------------------+      +------------------------------------------------
    | +--------++-------------+      +--------------+ |      | +--------++-------------+      +--------------+
    | |        ||             |      |              | |      | |        ||             |      |              |
    | | VMB_1  || VC_11       | ---- | VC_1k        | | ---- | | VMB_n  || VC_n1       | ---- | VC_nl        |
    | |        ||             |      |              | |      | |        ||             |      |              |
    | +--------++-------------+      +--------------+ |      | +--------++-------------+      +--------------+
    +-------------------------------------------------+      +------------------------------------------------
```

Where the `VMB_i` (variable meta data block) hold meta data related to the data chunk stored in `VDB_ij` (variable chunk). The number of data chunks (e.g. `k` and `l` in the scetch), generally may differ. We refer to the set `VMB_i, VC_i1, ..., VC_ik` as `VCB_i` (variable chunk block).

The `VMB_i` have the following sub structure:

```
    |<-------------- VMB_i ------------->|
    #                                    #
    +-------++---------+      +----------+
    |       ||         |      |          |
    | BMD_i || VMD_i1  | ---- | VMD_ij   |
    |       ||         |      |          |
    +-------++---------+      +----------+
```

Where the `BMD_i` (block meta data) have the following structure:

```
    BMD_i {
        numvars          // Number of variable chunks in data block
        dbsize           // Size of entire block VCB_i (meta + actual data)
    }
```

The `VMD_ij` have the following structure:

```
    VMD_ij {
        id               // Id of protected variable the data chunk belongs to
        idx              // Index of element in FTI_Data corresponding to protected variable with id='id'
        containerid      // Id of container variable chunk is stored in
        hascontent       // Boolean value indicating if container holds data or not
        dptr             // Position of chunk in runtime-data (FTI_Data[idx].ptr + dptr)
        fptr             // Position of chunk in file
        chunksize        // Size of chunk stored in container
        containersize    // Total space in container
        hash             // Hash of 'VC_ij'
    }
```

[Basic]

head

> The checkpointing safety levels L2, L3 and L4 produce additional overhead due to the necessary postprocessing work on the checkpoints. FTI offers the possibility to create an MPI process, called HEAD, in which this postprocessing will be accomplished. This allows it for the application processes to continue the execution immediately after the checkpointing.

Value	Meaning
`0`	The checkpoint postprocessing work is covered by the application processes
`1`	The HEAD process accomplishes the checkpoint postprocessing work (notice: In this case, the number of application processes will be (n-1)/node)

(*default = 0*)

node_size

> Lets FTI know, how many processes will run on each node (ppn). In most cases this will be the amount of processing units within the node (e.g. 2 CPU’s/node and 8 cores/CPU ! 16 processes/node).

Value	Meaning
`ppn (int > 0)`	Number of processing units within each node (notice: The total number of processes must be a multiple of `group_size*node_size`)

(*default = 2*)

ckpt_dir

> This entry defines the path to the local hard drive on the nodes.

Value	Meaning
`string`	Path to the local hard drive on the nodes

(*default = /scratch/username/*)

glbl_dir

> This entry defines the path to the checkpoint folder on the PFS (L4 checkpoints).

Value	Meaning
`string`	Path to the checkpoint directory on the PFS

(*default = /work/project/*)

meta_dir

> This entry defines the path to the meta files directory. The directory has to be accessible from each node. It keeps files with information about the topology of the execution.

Value	Meaning
`string`	Path to the meta files directory

(*default* = */home/user/.fti*)

ckpt_L1

> Here, the user sets the checkpoint frequency of L1 checkpoints when using `FTI_Snapshot()`.

Value	Meaning
`L1 intv. (int >= 0)`	L1 checkpointing interval in minutes
`0`	Disable L1 checkpointing

(*default* = *3*)

ckpt_L2

> Here, the user sets the checkpoint frequency of L2 checkpoints when using `FTI_Snapshot()`.

Value	Meaning
`L2 intv. (int >= 0)`	L2 checkpointing interval in minutes
`0`	Disable L2 checkpointing

(*default* = *5*)

ckpt_L3

> Here, the user sets the checkpoint frequency of L3 checkpoints when using `FTI_Snapshot()`.

Value	Meaning
`L3 intv. (int >= 0)`	L3 checkpointing interval in minutes
`0`	Disable L3 checkpointing

(*default* = *7*)

ckpt_L4

> Here, the user sets the checkpoint frequency of L4 checkpoints when using `FTI_Snapshot()`.

Value	Meaning
`L4 intv. (int >= 0)`	L4 checkpointing interval in minutes
`0`	Disable L4 checkpointing

(*default* = *11*)

dcp_L4

> Here, the user sets the checkpoint frequency of L4 differential checkpoints when using `FTI_Snapshot()`.

Value	Meaning
`L4 dCP intv. (int >= 0)`	L4 dCP checkpointing interval in minutes
`0`	Disable L4 dCP checkpointing

(*default = 0*)

inline_L2

> In this setting, the user chose whether the post-processing work on the L2 checkpoints is done by an FTI process or by the application process.

Value	Meaning
`0`	The post-processing work of the L2 checkpoints is done by an FTI process (notice: This setting is only alowed if head = 1)
`1`	The post-processing work of the L2 checkpoints is done by the application process

(*default = 1*)

inline_L3

> In this setting, the user chose whether the post-processing work on the L3 checkpoints is done by an FTI process or by the application process.

Value	Meaning
`0`	The post-processing work of the L3 checkpoints is done by an FTI process (notice: This setting is only alowed if head = 1)
`1`	The post-processing work of the L3 checkpoints is done by the application process

(*default = 1*)

inline_L4

> In this setting, the user chose whether the post-processing work on the L4 checkpoints is done by an FTI process or by the application process.

Value	Meaning
`0`	The post-processing work of the L4 checkpoints is done by an FTI process (notice: This setting is only alowed if head = 1)
`1`	The post-processing work of the L4 checkpoints is done by the application process

(*default = 1*)

keep_last_ckpt

> This setting tells FTI whether the last checkpoint taken during the execution will be kept in the case of a successful run or not.

Value	Meaning
`0`	During `FTI_Finalize()`, all checkpoints will be removed (except case 'keep_l4_ckpt=1')
`1`	After `FTI_Finalize()`, the last checkpoint will be kept and stored on the PFS as a L4 checkpoint (notice: Additionally, the setting failure in the configuration file is set to 2. This will lead to a restart from the last checkpoint if the application is executed again)

(*default = 0*)

keep_l4_ckpt

> This setting triggers FTI to keep all level 4 checkpoints taken during the execution. The checkpoint files will be saved in glbl_dir/l4_archive.

Value	Meaning
`0`	During `FTI_Finalize()`, all checkpoints will be removed (except case 'keep_last_ckpt=1')
`1`	All level 4 checkpoints taken during the execution, will be stored under `glbl_dir/l4_archive`. This folder will not be deleted during the `FTI_Finalize()` call.

(*default = 0*)

group_size

> The group size entry sets, how many nodes (members) forming a group.

Value	Meaning
`int i (2 <= i <= 32)`	Number of nodes contained in a group (notice: The total number of processes must be a multiple of `group_size*node_size`)

(*default = 4*)

max_sync_intv

> Sets the maximum number of iterations between synchronisations of the iteration length (used for `FTI_Snapshot()`). Internally the value will be rounded to the next lower value which is a power of 2.

Value	Meaning
`int i (0 <= i <= INT_MAX )`	maximum number of iterations between measurements of the global mean iteration time (`MPI_Allreduce` call)
`0`	Sets the value to 512, the default value for FTI

(*default = 0*)

ckpt_io

> Sets the I/O mode.

Value	Meaning
`1`	POSIX I/O mode
`2`	MPI-IO I/O mode
`3`	FTI-FF I/O mode
`4`	SIONLib I/O mode
`5`	HDF5 I/O mode

(*default = 1*)

enable_staging

> Enable the staging feature. This feature allows to stage files asynchronously from local (e.g. node local NVMe storage) to the PFS. FTI offers the API functions FTI_SendFile, FTI_GetStageDir and FTI_GetStageStatus for that.

Value	Meaning
`0`	Staging disabled
`1`	Stagin enabled (creation of the staging directory in folde 'ckpt_dir')

(*default* = *0*)

enable_dcp

> Enable differential checkpointing. In order to use this feature,ckpt_io has to be set to 3 (FTI-FF). To trigger differential checkpoints, use either level `FTI_L4_DCP` in FTI_Checkpoint or set the interval indcp_L4 for usage in FTI_Snapshot.

Value	Meaning
`0`	dCP disabled
`1`	dCP enabled

dcp_mode

> Set the hash algorithm used for differential checkpointing.

Value	Meaning
`0`	MD5
`1`	CRC32

(*default* = *0*)

dcp_block_size

> Set the desired partition block size for differential checkpointing in bytes. The block size must be within 512 .. `USHRT_MAX` (65535 on most systems).

Value	Meaning
`b (512 <= i <= USHRT_MAX)`	block size for dataset partition for dCP

(*default* = *16384*)

verbosity

> Sets the level of verbosity.

Value	Meaning
`1`	Debug sensitive. Beside warnings, errors and information, FTI debugging information will be printed
`2`	Information sensitive. FTI prints warnings, errors and information
`3`	FTI prints only warnings and errors
`4`	FTI prints only errors

(*default* = *2*)

[Restart]

failure

> This setting should mainly set by FTI itself. The behaviour within FTI is the following:
>
> - Within `FTI_Init()`, it remains on it initial value.
> - After the first checkpoint is taken, it is set to 1.
> - After `FTI_Finalize()` and `keep_last_ckpt` = 0, it is set to 0.
> - After `FTI_Finalize()` and `keep_last_ckpt` = 1, it is set to 2.

Value	Meaning
`0`	The application starts with its initial conditions (notice: In order to force a clean start, the value may be set to 0 manually. In this case the user has to take care about removing the checkpoint data from the last execution)
`1`	FTI is searching for checkpoints and starts from the highest checkpoint level (notice: If no readable checkpoints are found, the execution stops)
`2`	FTI is searching for the last L4 checkpoint and restarts the execution from there (notice: If checkpoint is not L4 or checkpoint is not readable, the execution stops)

(*default = 0*)

exec_id

> This setting should mainly set by FTI itself. During `FTI_Init()` the execution ID is set if the application starts for the first time (failure = 0) or the execution ID is used by FTI in order to find the checkpoint files for the case of a restart (`failure` = 1,2)

Value	Meaning
`yyyy-mm-dd_hh-mm-ss`	Execution ID (notice: If variate checkpoint data is available, the execution ID may set by the user to assign the desired starting point)

(*default = NULL*)

[Advanced]

The settings in this section, should **ONLY** be changed by advanced users.

block_size

> FTI temporarily copies small blocks of the L2 and L3 checkpoints to send them through MPI. The size of the data blocks can be set here.

Value	Meaning
`int`	Size in KB of the data blocks send by FTI through MPI for the checkpoint levels L2 and L3

(*default = 1024*)

transfer_size

> FTI transfers in chunks local checkpoint files to PFS. The size of the chunk can be set here.

Value	Meaning
`int`	Size in MB of the chunks send by FTI from local to PFS

(*default = 16*)

general_tag

> FTI uses a certain tags for the MPI messages. The tag for general messages can be set here.

Value	Meaning
`int`	Tag, used for general MPI messages within FTI

(*default = 2612*)

ckpt_tag

> FTI uses a certain tags for the MPI messages. The tag for messages related to checkpoint communication can be set here.

Value	Meaning
`int`	Tag, used for MPI messages related to a checkpoint context within FTI

(*default = 711*)

stage_tag

> FTI uses a certain tags for the MPI messages. The tag for messages related to staging communication can be set here.

Value	Meaning
`int`	Tag, used for MPI messages related to a staging context within FTI

(*default = 406*)

final_tag

> FTI uses a certain tags for the MPI messages. The tag for the message to the heads to trigger the end of the execution can be set here.

Value	Meaning
`int`	Tag, used for the MPI message that marks the end of the execution send from application processes to the heads within FTI

(*default = 3107*)

lustre_striping_unit

> This option only impacts if `-DENABLE_LUSTRE` was added to the Cmake command. It sets the striping unit for the MPI-IO file.

Value	Meaning
`int i (0 <= i <= INT_MAX )`	Striping size in Bytes. The default in Lustre systems is 1MB (1048576 Bytes), FTI uses 4MB (4194304 Bytes) as the dafault value
`0`	Assigns the Lustre default value

(*default = 4194304*)

lustre_striping_factor

> This option only impacts if `-DENABLE_LUSTRE` was added to the Cmake command. It sets the striping factor for the MPI-IO file.

Value	Meaning
`int i (0 <= i <= INT_MAX )`	Striping factor. The striping factor determines the number of OST's to use for striping.
`-1`	Stripe over all available OST's. This is the default in FTI.
`0`	Assigns the Lustre default value

(*default* = *-1*)

lustre_striping_offset

> This option only impacts if `-DENABLE_LUSTRE` was added to the Cmake command. It sets the striping offset for the MPI-IO file.

Value	Meaning
`int i (0 <= i <= INT_MAX )`	Striping offset. The striping offset selects a particular OST to begin striping at.
`-1`	Assigns the Lustre default value

(*default* = *-1*)

local_test

> FTI is building the topology of the execution, by determining the hostnames of the nodes on which each process runs. Depending on the settings for `group_size`, `node_size` and `head`, FTI assigns each particular process to a group and decides which process will be Head or Application dedicated. This is meant to be a local test. In certain situations (e.g. to run FTI on a local machine) it is necessary to disable this function.

Value	Meaning
`0`	Local test is disabled. FTI will simulate the situation set in the configuration
`1`	Local test is enabled (notice: FTI will check if the settings are correct on initialization and if necessary stop the execution)

(*default* = *1*)

Default Configuration

```
head                           = 0
node_size                      = 2
ckpt_dir                       = ./Local
glbl_dir                       = ./Global
meta_dir                       = ./Meta
ckpt_l1                        = 3
ckpt_l2                        = 5
ckpt_l3                        = 7
ckpt_l4                        = 11
dcp_l4                         = 0
inline_l2                      = 1
inline_l3                      = 1
inline_l4                      = 1
keep_last_ckpt                 = 0
keep_l4_ckpt                   = 0
group_size                     = 4
max_sync_intv                  = 0
ckpt_io                        = 1
enable_staging                 = 0
enable_dcp                     = 0
dcp_mode                       = 0
dcp_block_size                 = 16384
verbosity                      = 2

failure                        = 0
exec_id                        = 2018-09-17_09-50-30

rank                           = 0
number                         = 0
position                       = 0
frequency                      = 0

block_size                     = 1024
transfer_size                  = 16
general_tag                    = 2612
ckpt_tag                       = 711
stage_tag                      = 406
final_tag                      = 3107
local_test                     = 1
lustre_striping_unit           = 4194304
lustre_striping_factor         = -1
lustre_striping_offset         = -1
```

DESCRIPTION

> This configuration is made of default values (see: 5). FTI processes are not created (`head = 0`, notice: if there is no FTI processes, all post-checkpoints must be done by application processes, thus `inline_L2`, `inline_L3` and `inline_L4` are set to 1), last checkpoint won't be kept (`keep_last_ckpt = 0`), `FTI_Snapshot()` will take L1 checkpoint every 3 min,L2 - every 5 min, L3 - every 7 min and L4 - every 11 min, FTI will print errors and some few important information (`verbosity = 2`) and IO mode is set to POSIX (`ckpt_io = 1`). This is a normal launch of a job, because failure is set to 0 and `exec_id` is `NULL`. `local_test = 1` makes this a local test.

Using FTI Processes

```
head                         = 1
node_size                    = 2
ckpt_dir                     = /scratch/username/
glbl_dir                     = /work/project/
meta_dir                     = /home/username/.fti/
ckpt_L1                      = 3
ckpt_L2                      = 5
ckpt_L3                      = 7
ckpt_L4                      = 11
inline_L2                    = 0
inline_L3                    = 0
inline_L4                    = 0
keep_last_ckpt               = 0
group_size                   = 4
max_sync_intv                = 0
ckpt_io                      = 1
verbosity                    = 2
failure                      = 0
exec_id                      = NULL
block_size                   = 1024
transfer_size                = 16
mpi_tag                      = 2612
lustre_striping_unit         = 4194304
lustre_striping_factor       = -1
lustre_striping_offset       = -1
local_test                   = 1
```

DESCRIPTION

> FTI processes are created (`head = 1`) and all post-checkpointing is done by them, thus `inline_L2`, `inline_L3` and `inline_L4` are set to 0. Note that it is possible to select which checkpoint levels should be post-processed by heads and which by application processes (e.g. `inline_L2 = 1`, `inline_L3 = 0`, `inline_L4 = 0`). L1 post-checkpoint is always done by application processes, because it’s a local checkpoint. Be aware, when `head = 1`, and `inline_L2`, `inline_L3` and `inline_L4` are set to 1 all post-checkpoint is still made by application processes.

Using only selected ckpt level with FTI_Snapshot

```
head                         = 0
node_size                    = 2
ckpt_dir                     = /scratch/username/
glbl_dir                     = /work/project/
meta_dir                     = /home/username/.fti/
ckpt_L1                      = 0
ckpt_L2                      = 5
ckpt_L3                      = 0
ckpt_L4                      = 0
inline_L2                    = 1
inline_L3                    = 1
inline_L4                    = 1
keep_last_ckpt               = 0
group_size                   = 4
max_sync_intv                = 0
ckpt_io                      = 1
verbosity                    = 2
failure                      = 0
exec_id                      = NULL
block_size                   = 1024
transfer_size                = 16
mpi_tag                      = 2612
lustre_striping_unit         = 4194304
lustre_striping_factor       = -1
lustre_striping_offset       = -1
local_test                   = 1
```

DESCRIPTION

> `FTI_Snapshot()` will take only L2 checkpoint every 5 min Notice that other configurations are also possible (e.g. take L1 ckpt every 5 min and L4 ckpt every 30 min).

Keeping last checkpoint

```
head                        = 0
node_size                   = 2
ckpt_dir                    = /scratch/username/
glbl_dir                    = /work/project/
meta_dir                    = /home/username/.fti/
ckpt_L1                     = 3
ckpt_L2                     = 5
ckpt_L3                     = 7
ckpt_L4                     = 11
inline_L2                   = 1
inline_L3                   = 1
inline_L4                   = 1
keep_last_ckpt              = 1
group_size                  = 4
max_sync_intv               = 0
ckpt_io                     = 1
verbosity                   = 2
failure                     = 0
exec_id                     = NULL
block_size                  = 1024
transfer_size               = 16
mpi_tag                     = 2612
lustre_striping_unit        = 4194304
lustre_striping_factor      = -1
lustre_striping_offset      = -1
local_test                  = 1
```

DESCRIPTION

> FTI will keep last checkpoint (`Keep_last_ckpt = 1`), thus after finishing the job Failure will be set to 2.

Using different IO mode

For instance MPI-I/O:

```
head                        = 0
node_size                   = 2
ckpt_dir                    = /scratch/username/
glbl_dir                    = /work/project/
meta_dir                    = /home/username/.fti/
ckpt_L1                     = 3
ckpt_L2                     = 5
ckpt_L3                     = 7
ckpt_L4                     = 11
inline_L2                   = 1
inline_L3                   = 1
inline_L4                   = 1
keep_last_ckpt              = 0
group_size                  = 4
max_sync_intv               = 0
ckpt_io                     = 2
verbosity                   = 2
failure                     = 0
exec_id                     = NULL
block_size                  = 1024
transfer_size               = 16
mpi_tag                     = 2612
lustre_striping_unit        = 4194304
lustre_striping_factor      = -1
lustre_striping_offset      = -1
local_test                  = 1
```

DESCRIPTION

> FTI IO mode is set to MPI IO (`ckpt_io = 2`). Third option is SIONlib IO mode (`ckpt_io = 3`).

Restart after a failure

```
head                            = 0
node_size                       = 2
ckpt_dir                        = /scratch/username/
glbl_dir                        = /work/project/
meta_dir                        = /home/username/.fti/
ckpt_L1                         = 3
ckpt_L2                         = 5
ckpt_L3                         = 7
ckpt_L4                         = 11
inline_L2                       = 1
inline_L3                       = 1
inline_L4                       = 1
keep_last_ckpt                  = 0
group_size                      = 4
max_sync_intv                   = 0
ckpt_io                         = 1
verbosity                       = 2
failure                         = 1
exec_id                         = 2017-07-26_13-22-11
block_size                      = 1024
transfer_size                   = 16
mpi_tag                         = 2612
lustre_striping_unit            = 4194304
lustre_striping_factor          = -1
lustre_striping_offset          = -1
local_test                      = 1
```

DESCRIPTION

> This config tells FTI that this job is a restart after a failure (`failure` set to 1 and `exec_id` is some date in a format `YYYY-MM-DD_HH-mm-ss`, where `YYYY` - year, `MM` - month, `DD` - day, `HH` - hours, `mm` - minutes, `ss` - seconds). When recovery is not possible, FTI will abort the job (when using `FTI_Snapshot()`) and/or signal failed recovery by `FTI_Status()`.

Examples

Using FTI_Snapshot

```c
#include <stdlib.h>
#include <fti.h>

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);
    for (; i < 100; i++) {
        FTI_Snapshot();
        MPI_Allgather(&number, 1, MPI_INT, array,
        1, MPI_INT, FTI_COMM_WORLD);
 number += 1;
    }
    free(array);
    FTI_Finalize();
    MPI_Finalize();
    return 0;
}
```

DESCRIPTION

> `FTI_Snapshot()` makes a checkpoint by given time and also recovers data after a failure, thus makes the code shorter. Checkpoints intervals can be set in configuration file (see: ckpt_L1 - ckpt_L4).

Using FTI_Checkpoint

```
#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;
}
```

DESCRIPTION

> `FTI_Checkpoint()` allows to checkpoint at precise application intervals. Note that when using `FTI_Checkpoint()`, `ckpt_L1`, `ckpt_L2`, `ckpt_L3` and `ckpt_L4` are not taken into account.

Using FTI_Realloc with Fortran and Intrinsic Types

```
program test_fti_realloc
    use fti
    use iso_c_binding
    implicit none
    include 'mpif.h'

    integer, parameter          :: dp=kind(1.0d0)
    integer, parameter          :: N1=128*1024*25  !> 25 MB / Process
    integer, parameter          :: N2=128*1024*50  !> 50 MB / Process
    integer, parameter          :: N11 = 128
    integer, parameter          :: N12 = 1024
    integer, parameter          :: N13 = 25
    integer, parameter          :: N21 = 128
    integer, parameter          :: N22 = 1024
    integer, parameter          :: N23 = 50
    integer, target             :: FTI_COMM_WORLD
    integer                     :: ierr, status

    real(dp), dimension(:,:,:), pointer :: arr
    type(c_ptr)    :: arr_c_ptr
    real(dp), dimension(:,:,:), pointer :: tmp
    integer(4), dimension(:), pointer   :: shape

    allocate(arr(N11,N12,N13))
    allocate(shape(3))

    !> INITIALIZE MPI AND FTI
    call MPI_Init(ierr)
    FTI_COMM_WORLD = MPI_COMM_WORLD
    call FTI_Init('config.fti', FTI_COMM_WORLD, ierr)

    !> PROTECT DATA AND ITS SHAPE
    call FTI_Protect(0, arr, ierr)
    call FTI_Protect(1, shape, ierr)

    call FTI_Status(status)

    !> EXECUTE ON RESTART
    if ( status .eq. 1 ) then
        !> REALLOCATE TO SIZE AT CHECKPOINT
        arr_c_ptr = c_loc(arr(1,1,1))
        call FTI_Realloc(0, arr_c_ptr, ierr)
        call FTI_recover(ierr)
        !> RESHAPE ARRAY
        call c_f_pointer(arr_c_ptr, arr, shape)
        call FTI_Finalize(ierr)
        call MPI_Finalize(ierr)
        STOP
    end if

    !> FIRST CHECKPOINT
    call FTI_Checkpoint(1, 1, ierr)

    !> CHANGE ARRAY DIMENSION
    !> AND STORE IN SHAPE ARRAY
    shape = [N21,N22,N23]
    allocate(tmp(N21,N22,N23))
    tmp(1:N11,1:N12,1:N13) = arr
    deallocate(arr)
    arr => tmp

    !> TELL FTI ABOUT THE NEW DIMENSION
    call FTI_Protect(0, arr, ierr)

    !> SECOND CHECKPOINT
    call FTI_Checkpoint(2,1, ierr)

    !> SIMULATE CRASH
    call MPI_Abort(MPI_COMM_WORLD,-1,ierr)
end program
```

Using FTI_Realloc with Fortran and Derived Types

```fortran
program test_fti_realloc
    use fti
    use iso_c_binding
    implicit none
    include 'mpif.h'

    !> DEFINE DERIVED TYPE
    type :: polar
        real :: radius
        real :: phi
    end type

    integer, parameter          :: dp=kind(1.0d0)
    integer, parameter          :: N1=128*1024*25  !> 25 MB / Process
    integer, parameter          :: N2=128*1024*50  !> 50 MB / Process
    integer, parameter          :: N11 = 128
    integer, parameter          :: N12 = 1024
    integer, parameter          :: N13 = 25
    integer, parameter          :: N21 = 128
    integer, parameter          :: N22 = 1024
    integer, parameter          :: N23 = 50
    integer, target             :: FTI_COMM_WORLD
    integer                     :: ierr, status
    type(FTI_type)              :: FTI_Polar

    type(c_ptr)                            :: cPtr
    type(polar), dimension(:,:,:), pointer :: arr
    type(polar), dimension(:,:,:), pointer :: tmp
    integer(4), dimension(:), pointer      :: shape

    !> INITIALIZE FTI TYPE 'FTI_POLAR'
    call FTI_InitType(FTI_Polar, 2*4, ierr)

    allocate(arr(N11,N12,N13))
    allocate(shape(3))

    !> INITIALIZE MPI AND FTI
    call MPI_Init(ierr)
    FTI_COMM_WORLD = MPI_COMM_WORLD
    call FTI_Init('config.fti', FTI_COMM_WORLD, ierr)

    !> PROTECT DATA AND ITS SHAPE
    call FTI_Protect(0, c_loc(arr), size(arr),FTI_Polar, ierr)
    call FTI_Protect(1, shape, ierr)

    call FTI_Status(status)

    !> EXECUTE ON RESTART
    if ( status .eq. 1 ) then
        !> REALLOCATE TO DIMENSION AT LAST CHECKPOINT
        cPtr = c_loc(arr)
        call FTI_Realloc(0, cPtr, ierr) !> PASS DATA AS C-POINTER
        call FTI_recover(ierr)
        call c_f_pointer(cPtr, arr, shape) !> CAST BACK TO F-POINTER
        call FTI_Finalize(ierr)
        call MPI_Finalize(ierr)
        STOP
    end if

    !> FIRST CHECKPOINT
    call FTI_Checkpoint(1, 1, ierr)

    !> CHANGE ARRAY DIMENSION
    !> AND STORE IN SHAPE ARRAY
    shape = [N21,N22,N23]
    allocate(tmp(N21,N22,N23))
    tmp(1:N11,1:N12,1:N13) = arr
    deallocate(arr)
    arr => tmp
```

```
    !> TELL FTI ABOUT THE NEW DIMENSION
    call FTI_Protect(0, c_loc(arr), size(arr), FTI_Polar, ierr)

    !> SECOND CHECKPOINT
    call FTI_Checkpoint(2,1, ierr)

    !> SIMULATE CRASH
    call MPI_Abort(MPI_COMM_WORLD,-1,ierr)
end program
```

Applications-integrated-with-FTI

List of applications integrated with FTI

CoMD

Classical molecular dynamics proxy application.

https://github.com/exmatex/CoMD

File changes

Integrating FTI in CoMD took only addition of ~30 lines of code in 2 files. All occurrences of`MPI_COMM_WORLD` changed to `FTI_COMM_WORLD` except `FTI_Init("config.fti", MPI_COMM_WORLD);`

```
File: src-mpi/CoMD.c
102:    int i = 1;
103:    FTI_Protect(i++, sim->boxes->nAtoms, sim->boxes->nTotalBoxes, FTI_INTG);
104:
105:    FTIT_type RealTInfo;
106:    FTI_InitType(&RealTInfo, sizeof(real_t));
107:    FTIT_type Real3Info;
108:    FTI_InitType(&Real3Info, sizeof(real3));
109:    int maxTotalAtoms = MAXATOMS * (sim->boxes->nTotalBoxes);
110:
111:    FTI_Protect(i++, sim->atoms->gid, maxTotalAtoms, FTI_INTG);
112:    FTI_Protect(i++, sim->atoms->iSpecies, maxTotalAtoms, FTI_INTG);
113:    FTI_Protect(i++, sim->atoms->r, maxTotalAtoms, Real3Info);
114:    FTI_Protect(i++, sim->atoms->p, maxTotalAtoms, Real3Info);
115:    FTI_Protect(i++, sim->atoms->f, maxTotalAtoms, Real3Info);
116:
117:    int iStep = 0;
118:    FTI_Protect(i++, &iStep, 1, FTI_INTG);
119:
120:    if (FTI_Status() != 0) {
121:        int res = FTI_Recover();
122:        if (res != 0) {
123:            printf("\tRecovery failed! FTI_Recover returned %d.\n", res);
124:        }
125:    }
---
139:    profileStart(loopTimer);
140:    for (; iStep<nSteps;)
141:    {
142:       startTimer(commReduceTimer);
143:       sumAtoms(sim);
144:       stopTimer(commReduceTimer);
145:
146:       printThings(sim, iStep, getElapsedTime(timestepTimer));
147:
148:       startTimer(timestepTimer);
149:       timestep(sim, printRate, sim->dt);
150:       stopTimer(timestepTimer);
151:
152:       iStep += printRate;
153:       int res = FTI_Checkpoint(iStep, 1);
154:       if (res != FTI_DONE) {
155:             printf("\tCheckpoint failed! FTI_Checkpoint returned %d.\n", res);
156:       }
157:    }
158:    profileStop(loopTimer);
```

```
File: src-mpi/parallel.c
void initParallel(int* argc, char*** argv)
{
#ifdef DO_MPI
   MPI_Init(argc, argv);
   FTI_Init("config.fti", MPI_COMM_WORLD);
   MPI_Comm_rank(FTI_COMM_WORLD, &myRank);
   MPI_Comm_size(FTI_COMM_WORLD, &nRanks);
#endif
}

void destroyParallel()
{
#ifdef DO_MPI
   FTI_Finalize();
   MPI_Finalize();
#endif
}
```

Results

Log of run without FTI integrated.

```
===============================================================================
                   Poznan Supercomputing and Networking Center
                               eagle.man.poznan.pl
===============================================================================
-------------------------------------------------------------------------------
Start of calculations [pon, 9 paź 2017, 12:57:49 CEST]
-------------------------------------------------------------------------------
Support:        support-hpc@man.poznan.pl
-------------------------------------------------------------------------------
Mon Oct  9 12:57:53 2017: Starting Initialization
Mini-Application Name    : CoMD-mpi
Mini-Application Version : 1.1
Platform:
  hostname: e0026
  kernel name: 'Linux'
  kernel release: '3.10.105-1.el6.elrepo.x86_64'
  processor: 'x86_64'
Build:
  CC: '/opt/exp_soft/local/generic/openmpi/1.10.2-1_gcc482/bin/mpicc'
  compiler version: 'gcc (GCC) 4.8.2 20140120 (Red Hat 4.8.2-14)'
  CFLAGS: '-std=c99 -DDOUBLE -DDO_MPI -g -O5  -I/home/users/ksiero1/fti/include/ '
  LDFLAGS: '-L/home/users/ksiero1/fti/lib/ -lm -lcrypto'
  using MPI: true
  Threading: none
  Double Precision: true
Run Date/Time: 2017-10-09, 12:57:53
Command Line Parameters:
  doeam: 0
  potDir: pots
  potName: Cu_u6.eam
  potType: funcfl
  nx: 800
  ny: 800
  nz: 800
  xproc: 8
  yproc: 8
  zproc: 8
  Lattice constant: -1 Angstroms
  nSteps: 100
  printRate: 10
  Time step: 1 fs
  Initial Temperature: 600 K
  Initial Delta: 0 Angstroms
Simulation data:
  Total atoms        : 2048000000
  Min global bounds  : [   0.0000000000,   0.0000000000,   0.0000000000 ]
  Max global bounds  : [ 2892.0000000000, 2892.0000000000, 2892.0000000000 ]
Decomposition data:
```

```
  Processors         :      8,     8,     8
  Local boxes        :     62,    62,    62 =   238328
  Box size           : [   5.8306451613,   5.8306451613,   5.8306451613 ]
  Box factor         : [   1.0074548875,   1.0074548875,   1.0074548875 ]
  Max Link Cell Occupancy: 32 of 64
Potential data:
  Potential type   : Lennard-Jones
  Species name     : Cu
  Atomic number    : 29
  Mass             : 63.55 amu
  Lattice Type     : FCC
  Lattice spacing  : 3.615 Angstroms
  Cutoff           : 5.7875 Angstroms
  Epsilon          : 0.167 eV
  Sigma            : 2.315 Angstroms
Memory data:
  Intrinsic atom footprint =   88 B/atom
  Total atom footprint     = -157.000 MB (335.69 MB/node)
  Link cell atom footprint = 1280.082 MB/node
  Link cell atom footprint = 1408.000 MB/node (including halo cell data
Initial energy : -1.166063303598, atom count : 2048000000
Mon Oct  9 12:58:06 2017: Initialization Finished
Mon Oct  9 12:58:06 2017: Starting simulation
#                                                                                         Performance
#  Loop   Time(fs)       Total Energy   Potential Energy     Kinetic Energy  Temperature   (us/atom)     # At
      0       0.00    -1.166063303598    -1.243619295198     0.077555991600     600.0000      0.0000  2048000
     10      10.00    -1.166059649733    -1.233151964368     0.067092314635     519.0494      2.3384  2048000
     20      20.00    -1.166048425247    -1.208164731096     0.042116305849     325.8263      2.4122  2048000
     30      30.00    -1.166037572103    -1.186566075400     0.020528503297     158.8156      2.4182  2048000
     40      40.00    -1.166042088520    -1.183621872290     0.017579783770     136.0033      2.4197  2048000
     50      50.00    -1.166051685771    -1.193725983586     0.027674297815     214.0979      2.4213  2048000
     60      60.00    -1.166054644001    -1.202677534791     0.036622890790     283.3274      2.4201  2048000
     70      70.00    -1.166052134038    -1.204922829363     0.038870695326     300.7172      2.4207  2048000
     80      80.00    -1.166048793793    -1.203643980438     0.037595186645     290.8494      2.4198  2048000
     90      90.00    -1.166048002607    -1.203830919192     0.037782916585     292.3017      2.4193  2048000
    100     100.00    -1.166049790544    -1.206871500823     0.040821710279     315.8109      2.4176  2048000
Mon Oct  9 13:14:11 2017: Ending simulation
Simulation Validation:
  Initial energy  : -1.166063303598
  Final energy    : -1.166049790544
  eFinal/eInitial : 0.999988
  Final atom count : 2048000000, no atoms lost
Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1     977.2662      977.2662      101.34
loop                       1     964.3040      964.3040      100.00
timestep                  10      96.4285      964.2854      100.00
  position               100       0.1001       10.0087        1.04
  velocity               200       0.1025       20.5055        2.13
  redistribute           101       1.2731      128.5869       13.33
    atomHalo             101       0.9613       97.0902       10.07
  force                  101       7.8922      797.1134       82.66
commHalo                 303       0.3041       92.1548        9.56
commReduce                39       0.4388       17.1143        1.77
Timing Statistics Across 512 Ranks:
        Timer        Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total               51:  977.2630     140:  977.2672    977.2650      0.0012
loop                67:  964.3023     104:  964.3043    964.3033      0.0008
timestep            51:  964.2738     463:  964.2999    964.2841      0.0061
  position          49:    4.5466     373:   16.3177     11.2964      3.3782
  velocity           3:    7.8438     329:   29.0049     20.1119      6.3239
  redistribute      51:   53.6754     481:  168.3860    127.0497     17.6334
    atomHalo        51:   24.1044     481:  142.1157     94.3223     21.2231
  force            323:  775.9471      51:  905.1509    795.6917      9.6434
commHalo            51:   19.3264     481:  137.4904     89.3663     21.3808
commReduce          51:    2.8272     339:   27.7147     19.1658      4.1010
---------------------------------------------------
 Average atom update rate:       2.41 us/atom/task
---------------------------------------------------
---------------------------------------------------
 Average all atom update rate:   0.00 us/atom
---------------------------------------------------
```

```
---------------------------------------------------
 Average atom rate:            212.39 atoms/us
---------------------------------------------------
Mon Oct  9 13:14:11 2017: CoMD Ending
-------------------------------------------------------------------------------
End of calculations [pon, 9 paź 2017, 13:14:11 CEST].
-------------------------------------------------------------------------------
```

Log of run with FTI integrated.

```
===============================================================================
                   Poznan Supercomputing and Networking Center
                               eagle.man.poznan.pl
===============================================================================
-------------------------------------------------------------------------------
Start of calculations [pon, 9 paź 2017, 12:14:02 CEST]
-------------------------------------------------------------------------------
Support:        support-hpc@man.poznan.pl
-------------------------------------------------------------------------------
Mon Oct  9 12:14:08 2017: Starting Initialization
Mini-Application Name    : CoMD-mpi
Mini-Application Version : 1.1
Platform:
  hostname: e0026
  kernel name: 'Linux'
  kernel release: '3.10.105-1.el6.elrepo.x86_64'
  processor: 'x86_64'
Build:
  CC: '/opt/exp_soft/local/generic/openmpi/1.10.2-1_gcc482/bin/mpicc'
  compiler version: 'gcc (GCC) 4.8.2 20140120 (Red Hat 4.8.2-14)'
  CFLAGS: '-std=c99 -DDOUBLE -DDO_MPI -g -O5  -I/home/users/ksiero1/fti/include/ '
  LDFLAGS: '-L/home/users/ksiero1/fti/lib/ -lm -lcrypto'
  using MPI: true
  Threading: none
  Double Precision: true
Run Date/Time: 2017-10-09, 12:14:08
Command Line Parameters:
  doeam: 0
  potDir: pots
  potName: Cu_u6.eam
  potType: funcfl
  nx: 800
  ny: 800
  nz: 800
  xproc: 8
  yproc: 8
  zproc: 8
  Lattice constant: -1 Angstroms
  nSteps: 100
  printRate: 10
  Time step: 1 fs
  Initial Temperature: 600 K
  Initial Delta: 0 Angstroms
Simulation data:
  Total atoms        : 2048000000
  Min global bounds  : [   0.0000000000,   0.0000000000,   0.0000000000 ]
  Max global bounds  : [ 2892.0000000000, 2892.0000000000, 2892.0000000000 ]
Decomposition data:
  Processors         :      8,      8,      8
  Local boxes        :     62,    62,    62 =   238328
  Box size           : [   5.8306451613,   5.8306451613,   5.8306451613 ]
  Box factor         : [   1.0074548875,   1.0074548875,   1.0074548875 ]
  Max Link Cell Occupancy: 32 of 64
Potential data:
  Potential type   : Lennard-Jones
  Species name     : Cu
  Atomic number    : 29
  Mass             : 63.55 amu
  Lattice Type     : FCC
  Lattice spacing  : 3.615 Angstroms
  Cutoff           : 5.7875 Angstroms
```

```
  Epsilon          : 0.167 eV
  Sigma            : 2.315 Angstroms
Memory data:
  Intrinsic atom footprint =   88 B/atom
  Total atom footprint     = -157.000 MB (335.69 MB/node)
  Link cell atom footprint = 1280.082 MB/node
  Link cell atom footprint = 1408.000 MB/node (including halo cell data
Initial energy : -1.166063303598, atom count : 2048000000
Mon Oct  9 12:14:21 2017: Initialization Finished
Mon Oct  9 12:14:21 2017: Starting simulation
#                                                                                         Performance
#  Loop   Time(fs)       Total Energy   Potential Energy     Kinetic Energy  Temperature   (us/atom)     # At
      0       0.00    -1.166063303598    -1.243619295198     0.077555991600     600.0000      0.0000   2048000
     10      10.00    -1.166059649733    -1.233151964368     0.067092314635     519.0494      2.4020   2048000
     20      20.00    -1.166048425247    -1.208164731096     0.042116305849     325.8263      2.4509   2048000
     30      30.00    -1.166037572103    -1.186566075400     0.020528503297     158.8156      2.5215   2048000
     40      40.00    -1.166042088520    -1.183621872290     0.017579783770     136.0033      2.2746   2048000
     50      50.00    -1.166051685771    -1.193725983586     0.027674297815     214.0979      2.2883   2048000
     60      60.00    -1.166054644001    -1.202677534791     0.036622890790     283.3274      2.3185   2048000
--------------------------------------------------------------------------
mpirun has exited due to process rank 416 with PID 0 on
node e0769 exiting improperly. There are three reasons this could occur:
1. this process did not call "init" before exiting, but others in
the job did. This can cause a job to hang indefinitely while it waits
for all processes to call "init". By rule, if one process calls "init",
then ALL processes must call "init" prior to termination.
2. this process called "init", but exited without calling "finalize".
By rule, all processes that call "init" MUST call "finalize" prior to
exiting or it will be considered an "abnormal termination"
3. this process called "MPI_Abort" or "orte_abort" and the mca parameter
orte_create_session_dirs is set to false. In this case, the run-time cannot
detect that the abort call was an abnormal termination. Hence, the only
error message you will receive is this one.
This may have caused other processes in the application to be
terminated by signals sent by mpirun (as reported here).
You can avoid this message by specifying -quiet on the mpirun command line.
--------------------------------------------------------------------------
Mon Oct  9 12:30:09 2017: Starting Initialization
Mini-Application Name    : CoMD-mpi
Mini-Application Version : 1.1
Platform:
  hostname: e0026
  kernel name: 'Linux'
  kernel release: '3.10.105-1.el6.elrepo.x86_64'
  processor: 'x86_64'
Build:
  CC: '/opt/exp_soft/local/generic/openmpi/1.10.2-1_gcc482/bin/mpicc'
  compiler version: 'gcc (GCC) 4.8.2 20140120 (Red Hat 4.8.2-14)'
  CFLAGS: '-std=c99 -DDOUBLE -DDO_MPI -g -O5  -I/home/users/ksiero1/fti/include/ '
  LDFLAGS: '-L/home/users/ksiero1/fti/lib/ -lm -lcrypto'
  using MPI: true
  Threading: none
  Double Precision: true
Run Date/Time: 2017-10-09, 12:30:09
Command Line Parameters:
  doeam: 0
  potDir: pots
  potName: Cu_u6.eam
  potType: funcfl
  nx: 800
  ny: 800
  nz: 800
  xproc: 8
  yproc: 8
  zproc: 8
  Lattice constant: -1 Angstroms
  nSteps: 100
  printRate: 10
  Time step: 1 fs
  Initial Temperature: 600 K
  Initial Delta: 0 Angstroms
Simulation data:
  Total atoms        : 2048000000
  Min global bounds  : [   0.0000000000,   0.0000000000,   0.0000000000 ]
```

```
  Max global bounds  : [ 2892.0000000000, 2892.0000000000, 2892.0000000000 ]
Decomposition data:
  Processors         :      8,     8,     8
  Local boxes        :     62,    62,    62 =   238328
  Box size           : [   5.8306451613,   5.8306451613,   5.8306451613 ]
  Box factor         : [   1.0074548875,   1.0074548875,   1.0074548875 ]
  Max Link Cell Occupancy: 32 of 64
Potential data:
  Potential type   : Lennard-Jones
  Species name     : Cu
  Atomic number    : 29
  Mass             : 63.55 amu
  Lattice Type     : FCC
  Lattice spacing  : 3.615 Angstroms
  Cutoff           : 5.7875 Angstroms
  Epsilon          : 0.167 eV
  Sigma            : 2.315 Angstroms
Memory data:
  Intrinsic atom footprint =   88 B/atom
  Total atom footprint     = -157.000 MB (335.69 MB/node)
  Link cell atom footprint = 1280.082 MB/node
  Link cell atom footprint = 1408.000 MB/node (including halo cell data
Initial energy : -1.166063303598, atom count : 2048000000
Mon Oct  9 12:30:22 2017: Initialization Finished
Mon Oct  9 12:30:22 2017: Starting simulation
#                                                                                         Performance
#  Loop   Time(fs)       Total Energy   Potential Energy     Kinetic Energy  Temperature   (us/atom)     # At
     70      70.00    -1.166063303598    -1.243619295198     0.077555991600     600.0000      0.0000   2048000
     80      80.00    -1.166048793793    -1.203643980438     0.037595186645     290.8494      2.2586   2048000
     90      90.00    -1.166048002607    -1.203830919192     0.037782916585     292.3017      2.3377   2048000
    100     100.00    -1.166049790544    -1.206871500823     0.040821710279     315.8109      2.3146   2048000
Mon Oct  9 12:36:51 2017: Ending simulation
Simulation Validation:
  Initial energy  : -1.166063303598
  Final energy    : -1.166049790544
  eFinal/eInitial : 0.999988
  Final atom count : 2048000000, no atoms lost
Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1     401.7819      401.7819      103.47
loop                       1     388.3197      388.3197      100.00
timestep                   3      92.1447      276.4340       71.19
  position                30       0.1194        3.5818        0.92
  velocity                60       0.1042        6.2535        1.61
  redistribute            31       0.8148       25.2584        6.50
    atomHalo              31       0.4576       14.1847        3.65
  force                   31       8.0059      248.1816       63.91
commHalo                  93       0.1349       12.5416        3.23
commReduce                18       0.1819        3.2744        0.84
Timing Statistics Across 512 Ranks:
        Timer        Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total               37:  401.7036      42:  401.9202    401.7762      0.0357
loop                34:  388.3196     370:  388.4222    388.3436      0.0197
timestep            79:  276.2655     235:  276.5081    276.4445      0.0336
  position         147:    1.3705     221:    6.0085      3.6971      0.9796
  velocity         147:    2.3765     206:    9.8759      6.5194      1.7920
  redistribute     206:   18.4708     417:   37.1847     25.1751      4.2695
    atomHalo       415:    5.8228     417:   29.1367     13.9548      5.2873
  force            481:  241.1729      10:  261.1229    247.7969      2.5561
commHalo           415:    4.1255     417:   27.6639     12.3329      5.3435
commReduce         415:    1.4558     193:    6.4059      3.3989      0.9957
---------------------------------------------------
 Average atom update rate:       2.30 us/atom/task
---------------------------------------------------
---------------------------------------------------
 Average all atom update rate:   0.00 us/atom
---------------------------------------------------
---------------------------------------------------
 Average atom rate:            222.25 atoms/us
---------------------------------------------------
Mon Oct  9 12:36:52 2017: CoMD Ending
-------------------------------------------------------------------------------
```

```
End of calculations [pon, 9 paź 2017, 12:36:53 CEST].
-------------------------------------------------------------------------------
```

CoSP2

Linear algebra algorithms and workloads for a quantum molecular dynamics (QMD) electronic structure code.

https://github.com/exmatex/CoSP2

File changes

Integrating FTI in CoSP2 took only addition of ~30 lines of code in 2 files. All occurrences of`MPI_COMM_WORLD` changed to `FTI_COMM_WORLD` except `FTI_Init("config.fti", MPI_COMM_WORLD);`

```
File: src-mpi/sp2Loop.c
Function: sp2Loop()
    FTIT_type RealTInfo;
    FTI_InitType(&RealTInfo, sizeof(real_t));
    int i = 1;
    FTI_Protect(i++, &iter, 1, FTI_INTG);
    FTI_Protect(i++, xmatrix->iia, xmatrix->hsize, FTI_INTG);
    FTI_Protect(i++, xmatrix->jjcontig, xmatrix->hsize * xmatrix->msize , FTI_INTG);
    FTI_Protect(i++, xmatrix->valcontig, xmatrix->hsize * xmatrix->msize, RealTInfo);

    if (FTI_Status() != 0) {
      int res = FTI_Recover();
      if (res != 0) {
         printf("\tRecovery failed! FTI_Recover returned %d.\n", res);
      }
    }

...
    if (iter % 10 == 0) {
        int res = FTI_Checkpoint(iter, 1);
        if (res != FTI_DONE) {
              printf("\tCheckpoint failed! FTI_Checkpoint returned %d.\n", res);
        }
    }
```

```
File: src-mpi/parallel.c
void initParallel(int* argc, char*** argv)
{
#ifdef DO_MPI
   MPI_Init(argc, argv);
   FTI_Init("config.fti", MPI_COMM_WORLD);
   MPI_Comm_rank(FTI_COMM_WORLD, &myRank);
   MPI_Comm_size(FTI_COMM_WORLD, &nRanks);

   requestList = (MPI_Request*) malloc(nRanks*sizeof(MPI_Request));
   rUsed = (int*) malloc(nRanks*sizeof(int));
   for (int i = 0; i < nRanks; i++) { rUsed[i] = 0; }
#endif
}

void destroyParallel()
{
#ifdef DO_MPI
   free(requestList);
   FTI_Finalize();
   MPI_Finalize();
#endif
}
```

Results

Log of run without FTI integrated.

```
===============================================================================
                   Poznan Supercomputing and Networking Center

                               eagle.man.poznan.pl
===============================================================================
-------------------------------------------------------------------------------
Support:        support-hpc@man.poznan.pl
-------------------------------------------------------------------------------
CoSP2: SP2 Loop

Parameters:
msparse = 80  hDim = 98304  debug = 1
hmatName =
eps = 1e-05  hEps = 1e-16
idemTol = 1e-14

hDim = 98304 M = 80
Adjusted M = 96
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 3 local row min = 18432  row max = 24576  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 1 local row min = 6144  row max = 12288  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 11 local row min = 67584  row max = 73728  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 13 local row min = 79872  row max = 86016  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 8 local row min = 49152  row max = 55296  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 7 local row min = 43008  row max = 49152  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 6 local row min = 36864  row max = 43008  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 4 local row min = 24576  row max = 30720  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 9 local row min = 55296  row max = 61440  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 10 local row min = 61440  row max = 67584  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 14 local row min = 86016  row max = 92160  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 12 local row min = 73728  row max = 79872  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 2 local row min = 12288  row max = 18432  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
total procs = 16  total rows = 98304  total cols = 96
global row min = 0  row max = 98304  row extent = 98304
rank = 0 local row min = 0  row max = 6144  row extent = 6144

Sparsity:
Initial sparsity = 672042, fraction = 6.258879e-04, Avg per row = 6.836365
Max per row = 7
I = 4, count = 2, fraction = 0.000020
I = 5, count = 621, fraction = 0.006317
I = 6, count = 14838, fraction = 0.150940
I = 7, count = 82843, fraction = 0.842723
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 15 local row min = 92160  row max = 98304  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 5 local row min = 30720  row max = 36864  row extent = 6144

Gershgorin:
New  eMax, eMin = 1.745500e+00, -7.356212e-01
bufferSize = 9437184
Initial sparsity normalized = 672042, fraction = 6.258879e-04,  avg = 6.83636, max = 7

SP2Loop:
iter = 0  trX = 4.935743e+04  trX2 = 2.720037e+04
iter = 1  trX = 2.720037e+04  trX2 = 9.994787e+03
iter = 2  trX = 4.440595e+04  trX2 = 2.485384e+04
iter = 3  trX = 6.395806e+04  trX2 = 4.735425e+04
iter = 4  trX = 4.735425e+04  trX2 = 3.149323e+04
iter = 5  trX = 6.321528e+04  trX2 = 5.026180e+04
```

```
iter = 6  trX = 5.026180e+04  trX2 = 3.881328e+04
iter = 7  trX = 3.881328e+04  trX2 = 2.922713e+04
iter = 8  trX = 4.839943e+04  trX2 = 4.062611e+04
iter = 9  trX = 5.617275e+04  trX2 = 4.981154e+04
iter = 10  trX = 4.981154e+04  trX2 = 4.464542e+04
iter = 11  trX = 4.464542e+04  trX2 = 4.032639e+04
iter = 12  trX = 4.896445e+04  trX2 = 4.554145e+04
iter = 13  trX = 5.238745e+04  trX2 = 4.956883e+04
iter = 14  trX = 4.956883e+04  trX2 = 4.731790e+04
iter = 15  trX = 4.731790e+04  trX2 = 4.544718e+04
iter = 16  trX = 4.918861e+04  trX2 = 4.771064e+04
iter = 17  trX = 4.771064e+04  trX2 = 4.649398e+04
iter = 18  trX = 4.892731e+04  trX2 = 4.795556e+04
iter = 19  trX = 4.989906e+04  trX2 = 4.910173e+04
iter = 20  trX = 4.910173e+04  trX2 = 4.855031e+04
iter = 21  trX = 4.965316e+04  trX2 = 5.060054e+04
iter = 22  trX = 4.870578e+04  trX2 = -9.750371e+05
iter = 23  trX = 1.072449e+06  trX2 = -5.136388e+12
iter = 24  trX = -5.136388e+12  trX2 = 7.295617e+24

Results:
X2 Sparsity CCN = 2906510, fraction = 2.706898e-03 avg = 29.5665, max = 89
D Sparsity AAN = 2906464, fraction = 2.706856e-03 avg = 29.5661, max = 89
Number of iterations = 25


Counters for Rank 0
       Counter          Calls    Avg/Call(MB)         Total(MB)
_________________________________________________________________
reduce                    29            0.0000            0.0004
send                      39            2.2910           89.3504
recv                      39            2.2772           88.8095

Counter Statistics Across 16 Ranks:
        Counter      Rank: Min(MB)        Rank: Max(MB)       Avg(MB)      Stdev(MB)
_______________________________________________________________________________________
reduce               0:      0.0004       0:      0.0004        0.0004        0.0000
send                15:     87.4100       7:    138.5495      129.9097       15.7564
recv                15:     88.4093       6:    137.0340      129.9097       15.6236


Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1       3.4711        3.4711      100.00
loop                       1       3.4711        3.4711      100.00
  pre                      1       0.5444        0.5444       15.68
  sp2Loop                  1       2.7193        2.7193       78.34
    norm                   1       0.0417        0.0417        1.20
    x2                    25       0.0473        1.1820       34.05
    xadd                  13       0.0454        0.5899       16.99
    xset                  12       0.0383        0.4591       13.23
    exchange              50       0.0032        0.1576        4.54
    reduceComm            29       0.0070        0.2034        5.86

Timing Statistics Across 16 Ranks:
        Timer        Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                1:    3.4591      15:    3.5566      3.5160      0.0296
loop                 1:    3.4591      15:    3.5566      3.5160      0.0296
  pre                3:    0.4203       5:    0.5927      0.5180      0.0440
  sp2Loop           15:    2.7191      12:    2.7256      2.7229      0.0019
    norm             3:    0.0082       7:    0.0450      0.0376      0.0112
    x2               1:    0.2678      15:    1.1916      1.0701      0.3027
    xadd             1:    0.0548       0:    0.5899      0.5167      0.1744
    xset             1:    0.0408      15:    0.4638      0.4071      0.1383
    exchange         0:    0.1576       1:    1.1532      0.3589      0.2991
    reduceComm       5:    0.0513       3:    1.4170      0.3006      0.4217
-------------------------------------------------------------------------------

End of calculations [pon, 16 paź 2017, 12:17:13 CEST].

-------------------------------------------------------------------------------
```

Log of run with FTI integrated.

```
-------------------------------------------------------------------------------
Support:        support-hpc@man.poznan.pl
-------------------------------------------------------------------------------
CoSP2: SP2 Loop

Parameters:
msparse = 80  hDim = 98304  debug = 1
hmatName =
eps = 1e-05  hEps = 1e-16
idemTol = 1e-14

hDim = 98304 M = 80
Adjusted M = 96
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 1 local row min = 6144  row max = 12288  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 3 local row min = 18432  row max = 24576  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 10 local row min = 61440  row max = 67584  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
total procs = 16  total rows = 98304  total cols = 96
global row min = 0  row max = 98304  row extent = 98304
rank = 0 local row min = 0  row max = 6144  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 5 local row min = 30720  row max = 36864  row extent = 6144

Sparsity:
Initial sparsity = 672042, fraction = 6.258879e-04, Avg per row = 6.836365
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 6 local row min = 36864  row max = 43008  row extent = 6144
Max per row = 7
I = 4, count = 2, fraction = 0.000020
I = 5, count = 621, fraction = 0.006317
I = 6, count = 14838, fraction = 0.150940
I = 7, count = 82843, fraction = 0.842723
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 13 local row min = 79872  row max = 86016  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 7 local row min = 43008  row max = 49152  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 12 local row min = 73728  row max = 79872  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 15 local row min = 92160  row max = 98304  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 8 local row min = 49152  row max = 55296  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 11 local row min = 67584  row max = 73728  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 14 local row min = 86016  row max = 92160  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 4 local row min = 24576  row max = 30720  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 9 local row min = 55296  row max = 61440  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 2 local row min = 12288  row max = 18432  row extent = 6144

Gershgorin:
New  eMax, eMin = 1.745500e+00, -7.356212e-01
bufferSize = 9437184
Initial sparsity normalized = 672042, fraction = 6.258879e-04,  avg = 6.83636, max = 7

SP2Loop:
iter = 0  trX = 4.935743e+04  trX2 = 2.720037e+04
iter = 1  trX = 2.720037e+04  trX2 = 9.994787e+03
iter = 2  trX = 4.440595e+04  trX2 = 2.485384e+04
iter = 3  trX = 6.395806e+04  trX2 = 4.735425e+04
iter = 4  trX = 4.735425e+04  trX2 = 3.149323e+04
iter = 5  trX = 6.321528e+04  trX2 = 5.026180e+04
iter = 6  trX = 5.026180e+04  trX2 = 3.881328e+04
iter = 7  trX = 3.881328e+04  trX2 = 2.922713e+04
iter = 8  trX = 4.839943e+04  trX2 = 4.062611e+04
```

```
iter = 9  trX = 5.617275e+04  trX2 = 4.981154e+04
iter = 10  trX = 4.981154e+04  trX2 = 4.464542e+04
iter = 11  trX = 4.464542e+04  trX2 = 4.032639e+04
iter = 12  trX = 4.896445e+04  trX2 = 4.554145e+04
iter = 13  trX = 5.238745e+04  trX2 = 4.956883e+04
iter = 14  trX = 4.956883e+04  trX2 = 4.731790e+04
--------------------------------------------------------------------------
mpirun has exited due to process rank 3 with PID 12638 on
node e0700 exiting improperly. There are three reasons this could occur:

1. this process did not call "init" before exiting, but others in
the job did. This can cause a job to hang indefinitely while it waits
for all processes to call "init". By rule, if one process calls "init",
then ALL processes must call "init" prior to termination.

2. this process called "init", but exited without calling "finalize".
By rule, all processes that call "init" MUST call "finalize" prior to
exiting or it will be considered an "abnormal termination"

3. this process called "MPI_Abort" or "orte_abort" and the mca parameter
orte_create_session_dirs is set to false. In this case, the run-time cannot
detect that the abort call was an abnormal termination. Hence, the only
error message you will receive is this one.

This may have caused other processes in the application to be
terminated by signals sent by mpirun (as reported here).

You can avoid this message by specifying -quiet on the mpirun command line.

--------------------------------------------------------------------------
CoSP2: SP2 Loop

Parameters:
msparse = 80  hDim = 98304  debug = 1
hmatName =
eps = 1e-05  hEps = 1e-16
idemTol = 1e-14

hDim = 98304 M = 80
Adjusted M = 96
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 1 local row min = 6144  row max = 12288  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 3 local row min = 18432  row max = 24576  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 15 local row min = 92160  row max = 98304  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 4 local row min = 24576  row max = 30720  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 9 local row min = 55296  row max = 61440  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 13 local row min = 79872  row max = 86016  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 5 local row min = 30720  row max = 36864  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 11 local row min = 67584  row max = 73728  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 12 local row min = 73728  row max = 79872  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 8 local row min = 49152  row max = 55296  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 10 local row min = 61440  row max = 67584  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 6 local row min = 36864  row max = 43008  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 7 local row min = 43008  row max = 49152  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 14 local row min = 86016  row max = 92160  row extent = 6144
Generated H Matrix nnz = 672042 avg nnz/row = 6
total procs = 16  total rows = 98304  total cols = 96
global row min = 0  row max = 98304  row extent = 98304
rank = 0 local row min = 0  row max = 6144  row extent = 6144

Sparsity:
```

```
Initial sparsity = 672042, fraction = 6.258879e-04, Avg per row = 6.836365
Max per row = 7
I = 4, count = 2, fraction = 0.000020
I = 5, count = 621, fraction = 0.006317
I = 6, count = 14838, fraction = 0.150940
I = 7, count = 82843, fraction = 0.842723
Generated H Matrix nnz = 672042 avg nnz/row = 6
rank = 2 local row min = 12288  row max = 18432  row extent = 6144

Gershgorin:
New  eMax, eMin = 1.745500e+00, -7.356212e-01
bufferSize = 9437184
Initial sparsity normalized = 672042, fraction = 6.258879e-04,  avg = 6.83636, max = 7

SP2Loop:
iter = 10  trX = 4.981154e+04  trX2 = 4.464542e+04
iter = 11  trX = 4.464542e+04  trX2 = 4.032639e+04
iter = 12  trX = 4.896445e+04  trX2 = 4.554145e+04
iter = 13  trX = 5.238745e+04  trX2 = 4.956883e+04
iter = 14  trX = 4.956883e+04  trX2 = 4.731790e+04
iter = 15  trX = 4.731790e+04  trX2 = 4.544718e+04
iter = 16  trX = 4.918861e+04  trX2 = 4.771064e+04
iter = 17  trX = 4.771064e+04  trX2 = 4.649398e+04
iter = 18  trX = 4.892731e+04  trX2 = 4.795556e+04
iter = 19  trX = 4.989906e+04  trX2 = 4.910173e+04
iter = 20  trX = 4.910173e+04  trX2 = 4.855031e+04
iter = 21  trX = 4.965316e+04  trX2 = 5.060054e+04
iter = 22  trX = 4.870578e+04  trX2 = -9.750371e+05
iter = 23  trX = 1.072449e+06  trX2 = -5.136388e+12
iter = 24  trX = -5.136388e+12  trX2 = 7.295617e+24

Results:
X2 Sparsity CCN = 2906510, fraction = 2.706898e-03 avg = 29.5665, max = 89
D Sparsity AAN = 2906464, fraction = 2.706856e-03 avg = 29.5661, max = 89
Number of iterations = 25


Counters for Rank 0
       Counter          Calls    Avg/Call(MB)         Total(MB)
_________________________________________________________________
reduce                    19            0.0000            0.0003
send                      29            2.6508           76.8721
recv                      29            2.6315           76.3141

Counter Statistics Across 16 Ranks:
        Counter      Rank: Min(MB)        Rank: Max(MB)       Avg(MB)      Stdev(MB)
_______________________________________________________________________________________
reduce               0:      0.0003       0:      0.0003        0.0003        0.0000
send                15:     74.9711       7:    113.5838      106.5789       11.6620
recv                15:     75.9751       6:    112.0425      106.5789       11.5183


Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1       4.4862        4.4862      100.00
loop                       1       4.4862        4.4862      100.00
  pre                      1       0.5449        0.5449       12.15
  sp2Loop                  1       3.7464        3.7464       83.51
    norm                   1       0.0439        0.0439        0.98
    x2                    15       0.0423        0.6340       14.13
    xadd                   8       0.1030        0.8236       18.36
    xset                   7       0.0369        0.2582        5.76
    exchange              30       0.0033        0.0982        2.19
    reduceComm            19       0.0210        0.3999        8.91

Timing Statistics Across 16 Ranks:
        Timer        Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                1:    4.4668      10:    4.5643      4.5171      0.0276
loop                 1:    4.4668      10:    4.5643      4.5171      0.0276
  pre                1:    0.4197       2:    0.5793      0.5138      0.0399
  sp2Loop            9:    3.7438       8:    3.7513      3.7490      0.0019
    norm             1:    0.0081       6:    0.0463      0.0350      0.0112
```

```
    x2               3:    0.1789     13:    0.7067      0.5970      0.1598
    xadd             1:    0.0360     10:    0.8244      0.6514      0.2632
    xset             3:    0.0240      7:    0.2859      0.2339      0.0798
    exchange         0:    0.0982      3:    1.1791      0.4241      0.3105
    reduceComm       7:    0.1875      1:    1.2948      0.4136      0.3341
-------------------------------------------------------------------------------

End of calculations [pon, 16 paź 2017, 12:03:18 CEST].

-------------------------------------------------------------------------------
```

LULESH

Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)

File changes

In order to perform the cast from a C++ object to a char buffer, BOOST serialization was used. Three files were modified to port FTI: `lulesh.cc`, `lulesh.h` and `lulesh-comm.cc`. The modifications to the first two files are shown here. The modifications to the third file were barely the replacements of `MPI_COMM_WORLD` by `FTI_COMM_WORLD` and are not listed here.

```
diff --git a/LULESH/lulesh.cc b/FTI_LULESH/lulesh.cc
index a141611..d5572f8 100644
--- a/LULESH/lulesh.cc
+++ b/FTI_LULESH/lulesh.cc
@@ -162,6 +162,22 @@ Additional BSD Notice

 #include "lulesh.h"

+//********************
+// Boost Serialization
+//********************
+#include <boost/archive/text_oarchive.hpp>
+#include <boost/archive/text_iarchive.hpp>
+
+#include <sstream>
+// --- File version ---
+#include <fstream>
+std::stringstream locDom_ser;
+
+//*************************
+// FTI Checkpoint - Restart
+//*************************
+#include <fti.h>
+#define ITER_CKPT 500

 /*********************************/
 /* Data structure implementation */
@@ -213,7 +229,7 @@ void TimeIncrement(Domain& domain)
 #if USE_MPI
       MPI_Allreduce(&gnewdt, &newdt, 1,
                     ((sizeof(Real_t) == 4) ? MPI_FLOAT : MPI_DOUBLE),
-                    MPI_MIN, MPI_COMM_WORLD) ;
+                    MPI_MIN, FTI_COMM_WORLD) ;
 #else
       newdt = gnewdt;
 #endif
@@ -1061,7 +1077,7 @@ void CalcHourglassControlForElems(Domain& domain,
       /* Do a check for negative volumes */
       if ( domain.v(i) <= Real_t(0.0) ) {
 #if USE_MPI
-         MPI_Abort(MPI_COMM_WORLD, VolumeError) ;
+         MPI_Abort(FTI_COMM_WORLD, VolumeError) ;
 #else
          exit(VolumeError);
 #endif
@@ -1111,7 +1127,7 @@ void CalcVolumeForceForElems(Domain& domain)
       for ( Index_t k=0 ; k<numElem ; ++k ) {
          if (determ[k] <= Real_t(0.0)) {
 #if USE_MPI
```

```
-            MPI_Abort(MPI_COMM_WORLD, VolumeError) ;
+            MPI_Abort(FTI_COMM_WORLD, VolumeError) ;
 #else
             exit(VolumeError);
 #endif
@@ -1626,7 +1642,7 @@ void CalcLagrangeElements(Domain& domain, Real_t* vnew)
          if (vnew[k] <= Real_t(0.0))
         {
 #if USE_MPI
-           MPI_Abort(MPI_COMM_WORLD, VolumeError) ;
+           MPI_Abort(FTI_COMM_WORLD, VolumeError) ;
 #else
            exit(VolumeError);
 #endif
@@ -2030,7 +2046,7 @@ void CalcQForElems(Domain& domain, Real_t vnew[])

      if(idx >= 0) {
 #if USE_MPI
-         MPI_Abort(MPI_COMM_WORLD, QStopError) ;
+         MPI_Abort(FTI_COMM_WORLD, QStopError) ;
 #else
         exit(QStopError);
 #endif
@@ -2399,7 +2415,7 @@ void ApplyMaterialPropertiesForElems(Domain& domain, Real_t vnew[])
          }
          if (vc <= 0.) {
 #if USE_MPI
-             MPI_Abort(MPI_COMM_WORLD, VolumeError) ;
+             MPI_Abort(FTI_COMM_WORLD, VolumeError) ;
 #else
              exit(VolumeError);
 #endif
@@ -2683,6 +2699,19 @@ void LagrangeLeapFrog(Domain& domain)
 #endif
 }

+//Serialization
+void save (Domain *dom_saved){
+  boost::archive::text_oarchive oa(locDom_ser);
+  oa << dom_saved;
+}
+
+//Deserialization
+Domain* load (){
+  Domain *dom_loaded;
+  boost::archive::text_iarchive ia(locDom_ser);
+  ia >> dom_loaded;
+  return dom_loaded;
+}

 /******************************************/

@@ -2697,8 +2726,10 @@ int main(int argc, char *argv[])
    Domain_member fieldData ;

    MPI_Init(&argc, &argv) ;
-   MPI_Comm_size(MPI_COMM_WORLD, &numRanks) ;
-   MPI_Comm_rank(MPI_COMM_WORLD, &myRank) ;
+   char config_fti[] = "config.fti";
+   FTI_Init(config_fti, MPI_COMM_WORLD);
+   MPI_Comm_size(FTI_COMM_WORLD, &numRanks) ;
+   MPI_Comm_rank(FTI_COMM_WORLD, &myRank) ;
 #else
    numRanks = 1;
    myRank = 0;
@@ -2755,7 +2786,7 @@ int main(int argc, char *argv[])
    CommSBN(*locDom, 1, &fieldData) ;

    // End initialization
-   MPI_Barrier(MPI_COMM_WORLD);
+   MPI_Barrier(FTI_COMM_WORLD);
 #endif

    // BEGIN timestep to solution */
```

```
@@ -2766,10 +2797,68 @@ int main(int argc, char *argv[])
    gettimeofday(&start, NULL) ;
 #endif
 //debug to see region sizes
-//   for(Int_t i = 0; i < locDom->numReg(); i++)
-//      std::cout << "region" << i + 1<< "size" << locDom->regElemSize(i) <<std::endl;
-   while((locDom->time() < locDom->stoptime()) && (locDom->cycle() < opts.its)) {
+  // for(Int_t i = 0; i < locDom->numReg(); i++)
+  //    std::cout << "region" << i + 1<< "size" << locDom->regElemSize(i) <<std::endl;
+
+
+//--------------------------------------------------------------------------------------------------------
+
+  //First serialization to get a buffer size
+  save(locDom);

+  //Cast std::stringstream -> char*
+  int buffer_size = 0;
+  char* buffer_locDom_ser;
+  std::string tmp = locDom_ser.str();
+  buffer_size = tmp.size();
+  buffer_size += 1000000; //Add this to handle the dynamic change size of the buffer
+  buffer_locDom_ser = new char [buffer_size];
+  strcpy(buffer_locDom_ser, tmp.c_str());
+
+  //Checkpoint informations
+  int id = 1;
+  int level = 1;
+  int res;
+
+  FTI_Protect(0, &id, 1, FTI_INTG);
+  FTI_Protect(1, &level, 1, FTI_INTG);
+  FTI_Protect(2, buffer_locDom_ser, buffer_size, FTI_CHAR);
+
+
+  //Restart
+  if(FTI_Status() != 0){
+    if(!myRank)
+      std::cout << "---- Restart ----\n";
+
+    res = FTI_Recover();
+
+    //Update checkpoint information
+    if (res != 0) {
+        exit(1);
+    }
+    else { // Update ckpt. id & level
+        level = (level+1)%5;
+        id++;
+    }
+
+    //Cast char* to stringstream
+    locDom_ser.str(""); //reset the stringstream
+    locDom_ser.str(buffer_locDom_ser);
+
+    //Deserialization
+    Domain *tmp;
+    tmp = load();
+
+    //Set the used by simulation object
+    delete locDom;
+    locDom = NULL;
+    locDom = tmp;
+  }
+
+//--------------------------------------------------------------------------------------------------------
+   if (!myRank)
+     std::cout << "-- Start of the main loop --\n";
+   while((locDom->time() < locDom->stoptime()) && (locDom->cycle() < opts.its)) {
       TimeIncrement(*locDom) ;
       LagrangeLeapFrog(*locDom) ;

@@ -2777,6 +2866,26 @@ int main(int argc, char *argv[])
          printf("cycle = %d, time = %e, dt=%e\n",
```

```
                 locDom->cycle(), double(locDom->time()), double(locDom->deltatime()) ) ;
       }
+
+      //Checkpoint at ITER_CKPT
+      if((locDom->cycle()%ITER_CKPT) == 0 && locDom->cycle() != opts.its){
+
+        //Serialization of locDom in std::stringstream
+        locDom_ser.str("");
+        save(locDom);
+
+        //Cast std::stringstream -> char*
+        std::string tmp = locDom_ser.str();
+        buffer_locDom_ser[0] = '\0'; //reset the buffer
+        strcpy(buffer_locDom_ser, tmp.c_str());
+
+        res = FTI_Checkpoint(id, level);
+        // sleep(3); //for the tests
+        if(res != 0){
+          id++;
+          level= (level%4)+1;
+        }
+      }
    }

    // Use reduced max elapsed time
@@ -2791,7 +2900,7 @@ int main(int argc, char *argv[])
    double elapsed_timeG;
 #if USE_MPI
    MPI_Reduce(&elapsed_time, &elapsed_timeG, 1, MPI_DOUBLE,
-              MPI_MAX, 0, MPI_COMM_WORLD);
+              MPI_MAX, 0, FTI_COMM_WORLD);
 #else
    elapsed_timeG = elapsed_time;
 #endif
@@ -2806,6 +2915,7 @@ int main(int argc, char *argv[])
    }

 #if USE_MPI
+   FTI_Finalize();
    MPI_Finalize() ;
 #endif
```

```
diff --git a/LULESH/lulesh.h b/FTI_LULESH/lulesh.h
index b6afd5c..1ca6a59 100644
--- a/LULESH/lulesh.h
+++ b/FTI_LULESH/lulesh.h
@@ -24,6 +24,16 @@
 #include <math.h>
 #include <vector>

+//********************
+// Boost Serialization
+//********************
+#include <boost/serialization/vector.hpp>
+#include <iostream>
+#include <fstream>
+#if _OPENMP
+#include <omp.h>
+#endif
+
 //**************************************************
 // Allow flexibility for arithmetic representations
 //**************************************************
@@ -133,6 +143,27 @@ class Domain {
           Index_t rowLoc, Index_t planeLoc,
           Index_t nx, Int_t tp, Int_t nr, Int_t balance, Int_t cost);

+   Domain ()   :
+   m_e_cut(Real_t(1.0e-7)),
+   m_p_cut(Real_t(1.0e-7)),
+   m_q_cut(Real_t(1.0e-7)),
+   m_v_cut(Real_t(1.0e-10)),
```

```
+   m_v_cut(Real_t(1.0e-10)),
+   m_u_cut(Real_t(1.0e-7)),
+   m_hgcoef(Real_t(3.0)),
+   m_ss4o3(Real_t(4.0)/Real_t(3.0)),
+   m_qstop(Real_t(1.0e+12)),
+   m_monoq_max_slope(Real_t(1.0)),
+   m_monoq_limiter_mult(Real_t(2.0)),
+   m_qlc_monoq(Real_t(0.5)),
+   m_qqc_monoq(Real_t(2.0)/Real_t(3.0)),
+   m_qqc(Real_t(2.0)),
+   m_eosvmax(Real_t(1.0e+9)),
+   m_eosvmin(Real_t(1.0e-9)),
+   m_pmin(Real_t(0.)),
+   m_emin(Real_t(-1.0e+15)),
+   m_dvovmax(Real_t(0.1)),
+   m_refdens(Real_t(1.0)) {};
+
    //
    // ALLOCATION
    //
@@ -423,6 +454,243 @@ class Domain {
    void SetupElementConnectivities(Int_t edgeElems);
    void SetupBoundaryConditions(Int_t edgeElems);

+   friend class boost::serialization::access;
+   template <typename Archive>
+   void serialize(Archive &ar, const unsigned int version){
+
+
+      //Check de/serialization
+      // if(Archive::is_loading::value){
+      //    std::cout << "-------------------------\n";
+      //    std::cout << "Start of deserialization.\n";
+      //    std::cout << "-------------------------\n";
+      // }
+      // else {
+      //    std::cout << "-------------------------\n";
+      //    std::cout << "Start of serialization.\n";
+      //    std::cout << "-------------------------\n";
+      // }
+
+      ar & m_x ;  /* coordinates */
+      ar & m_y;
+      ar & m_z;
+
+      ar & m_xd ; /* velocities */
+      ar & m_yd ;
+      ar & m_zd ;
+
+      ar & m_xdd ; /* accelerations */
+      ar & m_ydd ;
+      ar & m_zdd ;
+
+      ar & m_fx ;  /* forces */
+      ar & m_fy ;
+      ar & m_fz ;
+
+      ar & m_nodalMass ;  /* mass */
+
+      ar & m_symmX ;  /* symmetry plane nodesets */
+      ar & m_symmY ;
+      ar & m_symmZ ;
+
+      // Element-centered
+
+      ar & m_numRanks ;
+      ar & m_colLoc ;
+      ar & m_rowLoc ;
+      ar & m_planeLoc ;
+      ar & m_tp ;
+
+      ar & m_sizeX ;
+      ar & m_sizeY ;
+      ar & m_sizeZ ;
```

```
+      ar & m_numElem ;
+      ar & m_numNode ;
+
+      ar & m_maxPlaneSize ;
+      ar & m_maxEdgeSize ;
+
+      // Region information
+      ar &  m_numReg ;
+      ar &  m_cost; //imbalance cost
+
+      if(Archive::is_loading::value){
+         m_regElemSize = new Index_t[m_numReg];
+      }
+      ar & boost::serialization::make_array <Index_t> (m_regElemSize, m_numReg); // Size of region sets
+
+      if(Archive::is_loading::value){
+         m_regNumList = new Index_t[m_numElem];
+      }
+      ar & boost::serialization::make_array <Index_t> (m_regNumList, m_numElem); // Region number per domain
+
+      if(Archive::is_loading::value){
+         m_regElemlist = new Index_t*[m_numReg];
+         for (int i = 0; i < m_numReg; i++){
+            m_regElemlist[i] = new Index_t[m_regElemSize[i]];
+         }
+      }
+
+      for (int i = 0; i < m_numReg; i++){
+         ar & boost::serialization::make_array <Index_t> (m_regElemlist[i], m_regElemSize[i]);
+      }
+
+      ar &  m_nodelist ;     /* elemToNode connectivity */
+
+      ar & m_lxim ;  /* element connectivity across each face */
+      ar & m_lxip ;
+      ar & m_letam ;
+      ar & m_letap ;
+      ar & m_lzetam ;
+      ar & m_lzetap ;
+
+      ar & m_elemBC ;  /* symmetry/free-surface flags for each elem face */
+
+      ar & m_dxx ;  /* principal strains -- temporary */
+      ar & m_dyy ;
+      ar & m_dzz ;
+
+      ar & m_delv_xi ;    /* velocity gradient -- temporary */
+      ar & m_delv_eta ;
+      ar & m_delv_zeta ;
+
+      ar & m_delx_xi ;    /* coordinate gradient -- temporary */
+      ar & m_delx_eta ;
+      ar & m_delx_zeta ;
+
+      ar & m_e ;   /* energy */
+
+      ar & m_p ;   /* pressure */
+      ar & m_q ;   /* q */
+      ar & m_ql ;  /* linear term for q */
+      ar & m_qq ;  /* quadratic term for q */
+
+      ar & m_v ;     /* relative volume */
+      ar & m_volo ;  /* reference volume */
+      ar & m_vnew ;  /* new relative volume -- temporary */
+      ar & m_delv ;  /* m_vnew - m_v */
+      ar & m_vdov ;  /* volume derivative over volume */
+
+      ar & m_arealg ;  /* characteristic length of an element */
+
+      ar & m_ss ;      /* "sound speed" */
+
+      ar & m_elemMass ;  /* mass */
+
+      // Cutoffs (treat as constants)
```

```
+      ar & const_cast<Real_t &>(m_e_cut);
+      ar & const_cast<Real_t &>(m_p_cut);
+      ar & const_cast<Real_t &>(m_q_cut);
+      ar & const_cast<Real_t &>(m_v_cut);
+      ar & const_cast<Real_t &>(m_u_cut);
+
+      // Other constants (usually setable, but hardcoded in this proxy app)
+      ar & const_cast<Real_t &>(m_hgcoef);
+      ar & const_cast<Real_t &>(m_ss4o3);
+      ar & const_cast<Real_t &>(m_qstop);
+      ar & const_cast<Real_t &>(m_monoq_max_slope);
+      ar & const_cast<Real_t &>(m_monoq_limiter_mult);
+      ar & const_cast<Real_t &>(m_qlc_monoq);
+      ar & const_cast<Real_t &>(m_qqc_monoq);
+      ar & const_cast<Real_t &>(m_qqc);
+      ar & const_cast<Real_t &>(m_eosvmax);
+      ar & const_cast<Real_t &>(m_eosvmin);
+      ar & const_cast<Real_t &>(m_pmin);
+      ar & const_cast<Real_t &>(m_emin);
+      ar & const_cast<Real_t &>(m_dvovmax);
+      ar & const_cast<Real_t &>(m_refdens);
+
+      // Variables to keep track of timestep, simulation time, and cycle
+      ar &  m_dtcourant ;         // courant constraint
+      ar &  m_dthydro ;           // volume change constraint
+      ar &   m_cycle ;             // iteration count for simulation
+      ar &  m_dtfixed ;           // fixed time increment
+      ar &  m_time ;              // current time
+      ar &  m_deltatime ;         // variable time increment
+      ar &  m_deltatimemultlb ;
+      ar &  m_deltatimemultub ;
+      ar &  m_dtmax ;             // maximum allowable time increment
+      ar &  m_stoptime ;          // end time for simulation
+
+      // OMP hack
+      #if _OPENMP
+         Index_t numthreads = omp_get_max_threads();
+      #else
+         Index_t numthreads = 1;
+      #endif
+
+      if (numthreads > 1) {
+         if(Archive::is_loading::value){
+            m_nodeElemStart = new Index_t[m_numNode+1];
+         }
+         ar & boost::serialization::make_array <Index_t> (m_nodeElemStart, m_numNode+1);
+
+         if(Archive::is_loading::value){
+            m_nodeElemCornerList = new Index_t[m_nodeElemStart[m_numNode]];
+         }
+         ar & boost::serialization::make_array <Index_t> (m_nodeElemCornerList, m_nodeElemStart[m_numNode]);
+      } else {
+         m_nodeElemStart = NULL;
+         m_nodeElemCornerList = NULL;
+      }
+
+      // Used in setup
+      ar & m_rowMin;
+      ar & m_rowMax;
+      ar & m_colMin;
+      ar & m_colMax;
+      ar & m_planeMin;
+      ar & m_planeMax;
+
+     #if USE_MPI
+     // account for face communication
+     Index_t comBufSize =
+       (m_rowMin + m_rowMax + m_colMin + m_colMax + m_planeMin + m_planeMax) *
+       m_maxPlaneSize * MAX_FIELDS_PER_MPI_COMM ;
+
+     // account for edge communication
+     comBufSize +=
+       ((m_rowMin & m_colMin) + (m_rowMin & m_planeMin) + (m_colMin & m_planeMin) +
+        (m_rowMax & m_colMax) + (m_rowMax & m_planeMax) + (m_colMax & m_planeMax) +
```

```
+        (m_rowMax & m_colMin) + (m_rowMin & m_planeMax) + (m_colMin & m_planeMax) +
+        (m_rowMin & m_colMax) + (m_rowMax & m_planeMin) + (m_colMax & m_planeMin)) *
+       m_maxEdgeSize * MAX_FIELDS_PER_MPI_COMM ;
+
+     // account for corner communication
+     // factor of 16 is so each buffer has its own cache line
+     comBufSize += ((m_rowMin & m_colMin & m_planeMin) +
+          (m_rowMin & m_colMin & m_planeMax) +
+          (m_rowMin & m_colMax & m_planeMin) +
+          (m_rowMin & m_colMax & m_planeMax) +
+          (m_rowMax & m_colMin & m_planeMin) +
+          (m_rowMax & m_colMin & m_planeMax) +
+          (m_rowMax & m_colMax & m_planeMin) +
+          (m_rowMax & m_colMax & m_planeMax)) * CACHE_COHERENCE_PAD_REAL ;
+
+
+      // Communication Work space
+      if(Archive::is_loading::value){
+         commDataSend = new Real_t[comBufSize];
+         commDataRecv = new Real_t[comBufSize];
+      }
+      ar & boost::serialization::make_array <Real_t> (commDataRecv,comBufSize);
+      ar & boost::serialization::make_array <Real_t> (commDataSend,comBufSize);
+
+      #endif
+
+      //Check de/serialization
+      // if(Archive::is_loading::value){
+      //    std::cout << "-------------------------\n";
+      //    std::cout << "Deserialization finished.\n";
+      //    std::cout << "-------------------------\n";
+      // }
+      // else {
+      //    std::cout << "-------------------------\n";
+      //    std::cout << "Serialization finished.\n";
+      //    std::cout << "-------------------------\n";
+      // }
+   }
+
    //
    // IMPLEMENTATION
    //
```