# FTI: high performance Fault Tolerance Interface for hybrid systems

Leonardo Bautista-Gomez
Tokyo Institute of Technology
INRIA
leobago {at} matsulab.is.titech.ac.jp

Dimitri Komatitsch
Observatoire Midi-Pyrénées
University of Toulouse
dimitri.komatitsch {at} get.obs-mip.fr

Naoya Maruyama
Tokyo Institute of Technology
naoya {at} matsulab.is.titech.ac.jp

Seiji Tsuboi
JAMSTEC
tsuboi {at} jamstec.go.jp

Franck Cappello
INRIA
University of Illinois
fci {at} lri.fr

Satoshi Matsuoka
Tokyo Institute of Technology
National Institute of Informatics
matsu {at} is.titech.ac.jp

## ABSTRACT

*Large scientific applications deployed on current petascale systems expend a significant amount of their execution time dumping checkpoint files to remote storage. New fault tolerant techniques will be critical to efficiently exploit post-petascale systems. In this work, we propose a low-overhead high-frequency multi-level checkpoint technique in which we integrate a highly-reliable topology-aware Reed-Solomon encoding in a three-level checkpoint scheme. We efficiently hide the encoding time using one Fault-Tolerance dedicated thread per node. We implement our technique in the Fault Tolerance Interface FTI. We evaluate the correctness of our performance model and conduct a study of the reliability of our library. To demonstrate the performance of FTI, we present a case study of the Mw9.0 Tohoku Japan earthquake simulation with SPECFEM3D on TSUBAME2.0. We demonstrate a checkpoint overhead as low as 8% on sustained 0.1 Petaflops runs (1152 GPUs) while checkpointing at high frequency.*

## 1. INTRODUCTION

In high performance computing (HPC), systems are built from highly reliable components. However, the overall failures rate of supercomputers increases with the components count. Nowadays, petascale machines have a mean time between failures (MTBF) measured in hours or days[41] and fault tolerance (FT) is a well-known issue. Long running large applications rely in fault-tolerant (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 not expensive technique in comparison with the process-replication scheme that imposes over 100% of overhead.


*SC11* November 12-18, 2011, Seattle, Washington, USA


However, when a large application is checkpointed, tens of thousands of processes will write each 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 is not increasing at the same speed than the computational capabilities, large checkpoints can lead to an I/O bottleneck, which cause up to 25% of overhead in current petascale systems[38, 33]. Post-petascale systems will have a significantly larger number of components and an important amount of memory. This will have an impact in 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 patterns of the failures. Indeed, when moving from 90nm to 16nm technology, the soft error rate (SER) is likely to increase significantly, as showed in a recent study from Intel [40, 10]. 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) [2]. Moreover, some hard failures can be tolerated using solid-state-drives (SSD) [31] and cross-node redundancy schemes, such as checkpoint replication or XOR encoding [35] which allows to leverage multi-level checkpointing, as proposed by Moody et al. [1]. Furthermore, Cheng et al. demonstrated that more complex erasure codes such as Reed-Solomon (RS) encoding can be used to increase further the percentage of hard failures tolerated without stressing the PFS [3]. Our work goes in the same direction of these three studies and it partially leverage some of those results.

### 1.1 Contributions

In this section we list the contributions of our current work with respect to the cited works and our previous publications. The major contributions of this paper are:

- We propose a model of a highly reliable erasure code scheme based on our topology-aware RS encoding published in our previous work [5]. We extend our previous research by studying not only the scalability of the encoding algorithm but also the impact of the checkpoint size per node and the group size, on the encoding and decoding performance. We evaluate and prove the accuracy of our performance model and we show

that our topology-aware RS encoding scheme is several orders of magnitude more reliable than the XOR encoding.

- We apply our FT-dedicated thread scheme presented in our previous work [6] in order to decrease further the checkpoint overhead and we integrate it for the first time in a multi-level checkpoint technique that we implement in our FTI library. Our evaluation shows that by using FT-dedicated threads in the nodes, we can efficiently hide the encoding time making its overhead negligible in comparison with a simple local write checkpoint.
- We extend our evaluation with a functional test in a real case study by simulating the March $11^{th}$ Mw9.0 Tohoku, Japan earthquake and we present synthetic seismograms for the Hirono station in Fukushima prefecture.
- We perform a large scalability and overhead evaluation of our library with SPECFEM3D and we show that FTI can successfully scale to more than 1000 GPUs and reach over 100 TFlops while checkpointing at high frequency and causing only about 8% overhead in comparison with a no checkpointed execution.

The rest of this paper is organized as follows. Section 2 discuses the background and motivations for this work. Section 3 presents our low-overhead high-frequency multi-level checkpoint model and section 4 the implementation of our FTI library. In section 5 we present our evaluation. Finally, in section 6 we present some related work and section 7 concludes this paper.

# 2. BACKGROUND AND MOTIVATIONS

In this section we discuss the importance of resiliency for post-petascale systems, the limitations of PFS-based checkpointing and the requirements of a post-petascale FT library.

## 2.1 Importance of reliable post-petascale computing

In this work, we have chosen SPECFEM3D as a case study. SPECFEM3D [50] is an application that uses a Spectral Finite Element Method in three Dimensions to perform numerical calculation of synthetic seismograms in complex geological models and capable to treat the full complexity of the physics of seismic wave propagation in the full 3D Earth: anisotropy, attenuation, gravity, rotation and presence of the oceans. It is widely used in geophysics for instance to simulate the propagation of seismic waves following large earthquakes.

Geophysics is a science basically related to observations, and its rapid evolution in the last decades is mainly due to significant meteorological advances which improved the quality and quantity of the data collected. Obviously, progress achieved in instrumentation and data acquisition must be accompanied by a better theoretical understanding of underlying physical phenomena as well as better modeling of these phenomena by means of 3D numerical modeling techniques.

In the near future, the acoustic and mechanic scientific communities want to achieve simulations with increasingly higher resolutions, thus it is important to adapt current numerical methods to the architecture of modern large scale supercomputers. For example, ultrasonography uses frequencies in the order of several MHz, which leads to millimeter wavelengths over several tens of centimeters distances in a 3D model. Also, high-frequency sonars use frequencies around 3 kilohertz over distances of several kilometers, in 3D models as well.

Another example that is very difficult to perform is the calculation of the propagation of seismic waves through the entire Earth, including its solid inner core and fluid outer core. Indeed, for such computations one often needs to solve the elastic or viscoelastic wave equation for an anisotropic three-dimensional model of the Earth at very short seismic periods (around 2 seconds, that is 0.5 Hz, for the whole Earth), which remains a very difficult challenge from a computational point of view. Future post-petascale supercomputers will make such long simulations possible if the applications can successfully finish their execution even in the presence of failures. Hence the importance of reliable post-petascale systems, not only for the geophysics community, but for the global scientific community using numerical modeling techniques in general.

## 2.2 Post-petascale resiliency

First, the progress in transistors size will increase the SER several times [40, 10] in such way, that soft errors are expected to be the most common type of failures for post-petascale systems.

Second, post-petascale systems are expected to have tens of thousands of computing nodes and hundreds of thousands of sockets. While HPC systems are built with highly reliable components, such large machines are expected to experience failures on a regular basis [9, 33, 38]. If CR is used to deal with failures, then the checkpoint frequency should be high enough to decrease the recovery cost induced by the re-execution of the lost work.

Third, to provide the high level of concurrency needed, these machines will likely to be very dense, which in turn may increase the probability of correlated failures to occur.

For example, at the Lawrence Livermore National Laboratory (LLNL), on the Coastal system, two nodes share a single power supply[1], thus when a power supply fails two nodes will fail simultaneously. Sometimes, the patterns of such correlated failures may be predictable and then this information can be taken into account when deploying FT-techniques. However, other more complex correlated failures can occur. It has been observed some cases, in clusters where the cooling system is located at the lower part of the racks, when the cooling system does not work properly, the nodes located at the upper part of the racks will overheat and fail simultaneously. Also, when a part of the network fails, a subset of the nodes will be not accessible.

In consequence, post-petascale FT-techniques should use different schemes with different levels of reliability and different costs to improve the efficiency of the system (See section 3.4).

## 2.3 Remote-disk based checkpoint limitations

As explained in section 2.2, the failure rate will increase significantly in post-petascale systems thus the checkpoint frequency is likely to be higher than for current systems, in addition, the I/O bandwidth is not scaling as fast as the computing capabilities creating an I/O bottleneck when the applications try to write a large amount of checkpoint data in the PFS [39, 41, 43]. Moreover, as explained in section 2.2, an important percentage of failures are due to soft errors and can be recovered with other faster techniques (see section 3.4). Overall, although several PFS, such as Panasas [11], GPFS [12] and Lustre[13], have made important progress in performance, PFS based checkpoint is a time expensive approach, that is only required for a small percentage of failures[1, 2].

# 3. LOW-OVERHEAD HIGH-FREQUENCY MULTI-LEVEL CHECKPOINT MODEL

The scalability issue of current PFS based CR is tightly linked to the scaling difference between computing capacities and I/O bandwidth. When the compute nodes have local storage to save the

checkpoint image, the checkpoint writing bandwidth will increase linearly with the number of nodes avoiding network congestions and I/O bottlenecks.

This is the case of Tokyo Institute of Technology new supercomputer, TSUBAME2.0, ranking top 4th in the Top500 list [7] and 2nd in the Green500 list [8] as at the time of this submission (April 2011). Indeed, TSUBAME2.0 has two Server-grade SCL SSDs in each compute node, with a per node storage capacity of twice the memory size per node. Other systems at LLNL are also using SSDs on the compute nodes to speed-up the checkpoint performance [1]. In this work, we will focus on systems with local storage on the compute nodes as an incentive to motivate co-design for future post-petascale machines. For diskless systems we refer the reader to a study done by Moody at al. on how to use these techniques using extra nodes and in-memory checkpoint[1].

While storing the checkpoint files in local disks speeds up the checkpoint writing performance, this simple scheme has a low reliability rate. Indeed, this technique is able to restart an application only when all the checkpoint files are still available, such as soft errors or transient failures.

Dong et al. proposed an hybrid model [2] where local checkpoint in PCMs is mixed with less frequent global checkpoint, in such way that soft errors can be recovered with the low-overhead local checkpoints and the less frequent hard errors can be recovered with global checkpoint. In this work, we modify and extend that approach using erasure codes as an intermediate level between basic local checkpoint and global checkpoint. As a result, our model is also tightly related with the multi-level checkpoint model proposed by Moody et al. but with a more reliable and highly optimized for hybrid systems L2 checkpoint (See section 3.4).

## 3.1 Topology-aware RS encoding

Hard failures can be recovered with local checkpoint if the checkpoint availability is guaranteed after the failure. To guarantee this, one can use checkpoint replication or erasure codes. Checkpoint replication in a partner node is simple to implement and it only requires an extra transfer after the local checkpoint is done. Checkpoint replication can withstand failures as far as two partner nodes do not fail simultaneously.

Erasure codes are a storage efficient way to tolerate hard failures. For example, using exclusive-or (XOR) encoding, a group of N nodes can tolerate one node failure with a lower storage cost than the checkpoint replication approach. In the worst case, both approaches will fail to tolerate two simultaneous node failures but they will always succeed to tolerate one single node failure. Assuming the checkpoint files have a size S, the XOR technique will generate S extra parity data when the replication approach will generate N*S extra replicated data, which is an significant difference. Furthermore, this difference grows rapidly while trying to tolerate several simultaneous failures.

In information theory, erasure codes transform a message of K symbols into a longer message with K+M symbols in order to recover the original message after losing one or several symbols. RS is an optimal erasure code since it has the property that any K symbols out of the K+M symbols are sufficient to recover the original message. We chose RS encoding because we believe it is important to maximize the reliability of the system, particularly when dealing with correlated failures (See section 2.2). When using RS encoding for FT in HPC, the system will be partitioned in groups of K processes (K checkpoint files), thus the groups can encode the checkpoint files in parallel. Each group will generate M encoded checkpoint files in order to tolerate M erasures[5].

In order to tolerate hard failures, the partitioning into groups must take into account the topology of the machine. If M processes of one group belong to the same node, a crash of such node will produce an unrecoverable failure. To reach the optimal RS efficiency, we detect the processes belonging to the same node and build an sketch of the cluster's topology. Then, as shown in figure 1, the system is partitioned in such way that all the processes in a group belong to different nodes, similar to the chipkill technology used in computer memories.

The main drawback of the RS encoding is its complexity. In particular, for a group of K processes generating M encoded checkpoint files, the encoding time will increase linearly with M. For our model we will assume that the encoding is overlapped with the application execution (See section 3.3), thus we can choose the highest level of reliability and generate M=K encoded checkpoints per group.

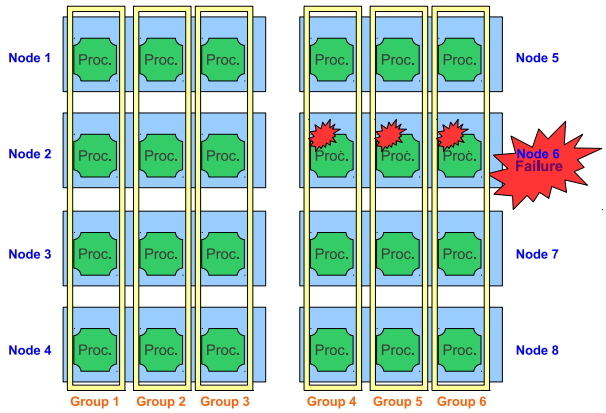



**Figure 1: Topology-aware Reed-Solomon encoding**

Since the encoded checkpoint files will be stored in the same devices than the checkpoint files, one failure will generate two erasures, one checkpoint file and one encoded checkpoint file, thus each group can tolerate M/2 process failures. Also, when a node crashes, our topology-aware partitioning technique *distributes* the failure among different groups, as presented in figure 1.

## 3.2 Performance model

In our previous work [5], we proposed an encoding algorithm and we focused on the scalability study. The complexity of the proposed algorithm is presented in formula 1.

$$T_{RSenc.} = t * (r * z + (m * (a + b * z + e * z)) + w * z) \qquad (1)$$

Where each group of k processes will generate m=k encoded checkpoint files. The checkpoint files are divided in t blocks of size z. For each block, the algorithm will read the block, perform m communications of z bytes and m encoding computations and finally write the encoded block. We assume that it takes a + b*z to transfer a message of size z between two processors regardless of their location, where a is the latency of the network and 1/b is the bandwidth of the network, the encoding rate is e seconds per byte, the reading rate is r seconds per byte and the writing rate is w seconds per byte.

The local storage in the compute nodes guarantee very fast reading and writing performance, thus the reading and writing time is negligible in comparison with the encoding computation (See section 4.2). Furthermore, we optimized our algorithm by overlapping communications and computation and by choosing the right block size we can completely hide the communications (See section 4.2),

hence our formula is simplified as follows.

$$T_{RSenc.} = t * m * e * z \quad (2)$$

As we can see, the encoding complexity will depend on the checkpoint size (s=t*z) and the group size (k=m), hence the encoding time should increase linearly with these two factors.

The decoding computation will generally last twice longer than the encoding because it is composed by two steps; the regeneration of the the lost checkpoint files (using the parity data generated by the RS encoding), and the re-encoding of the checkpoint files to regenerate the lost encoded checkpoint files, so the system will be able to tolerate a failure that occurs just after the recovery.

## 3.3 GPU computing and FT-dedicated resources

Roadrunner [42] was in 2008 the first supercomputer to break the petaflop barrier in Linkpack [18] performance and it had an hybrid architecture composed by Opteron processors and PowerXCell accelerators. Hybrid systems have become an important trend in the HPC community [32] during the last few years. As an example, three of the first five supercomputers in the Top500 list [7] are hybrid systems composed of CPUs and GPUs. While GPUs were initially specific to graphics, they are now widely used as general purpose devices in HPC and a large number of scientific applications have been ported to GPU clusters achieving significant speedups in performance. GPUs are now used, not only for the compute nodes, but also in storage systems [17, 34].

On the other hand, for many scientific applications it is very complex to efficiently divide the same workload among the GPU and CPUs in parallel. Indeed, since GPUs have a larger throughput than CPUs, the load balancing between these two devices is a difficult issue and moreover, exploitation of substantially lower-performing CPUs would not add much speedup for the effort. Thus, most of developers will rather map one GPU with one CPU core using one MPI process. Hence the role of CPUs are to serve as communication and service processors while the GPUs execute the dominant weak scaling portion of the computation, and only compute the minor strong scaling, less parallelizable portion where their greater single thread performance matters.

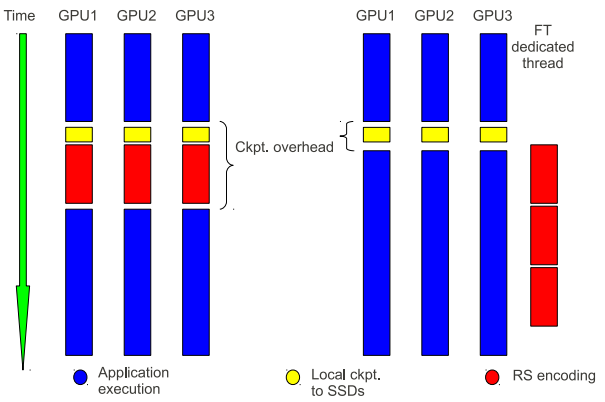



**Figure 2: Reed-Solomon encoding hiding with one Fault Tolerance dedicated thread per node**

Since those hybrid systems tend to have excess CPU cores per node when they execute such highly scaling applications as they also have to cater to CPU-only applications for various reasons, such GPU/CPU mapping will leave some CPU cores unused, thus they can be used for resiliency. Generally, for important scientific application long-runs (that are likely to be checkpointed), researchers will reserve a large number of exclusive nodes to guarantee the highest memory and network bandwidth.

For this kind of GPUs applications, it is possible to spawn one extra FT-dedicated thread per node using the idle resources to improve the checkpoint performance. Particularly in our technique, we can delegate the RS encoding of all the local checkpoint files to the FT-dedicated thread and overlap it with the application execution. By doing so, each FT-dedicated thread will encode several checkpoint files in a serial fashion. Serializing the encoding tasks will take longer than having each process encoding its own checkpoint file. However, as presented in figure 2, in the former, the encoding work can be done in parallel with the application execution, thus the FT-dedicated thread and serialized encoding mode will impose a significantly lower overhead. It is important to notice that the checkpoint interval has to be larger or equal than the serialized encoding time.

Moreover, this model can be extended to every kind of system (hybrid or not) by sacrificing a small loss in performance when launching a FT-dedicated thread per node. In current systems with about 12 CPU cores per node, such technique will immediately be translated as an 8.3% loss in performance. Furthermore, since the concurrency level per node will further increase in the next years, our model will impose a significantly lower overhead for non-hybrid systems. For example, the coming Blue Waters system at UIUC-NCSA will have 32 cores per quad-chip module (four IBM Power-7 processors) and can spawn 4 threads per core, for a total of 128 threads per node, thus the loss in performance while dedicating one thread per node for resiliency should be about 0.7%.

## 3.4 Multi-level checkpoint

For the first time in this work, we integrate our topology-aware RS encoding and our FT-dedicated thread technique in a multi-level checkpoint scheme. Multi-level checkpoint where different levels guarantee different reliability levels at different costs was well modeled by Moody et al. [1] using a Markov Model. In that work, a three-level checkpoint scheme was proposed. In such scheme, the first level (L1) is a Local checkpoint on local SSDs, the second level (L2) could be a Partner or a XOR technique, depending on the user's choice, and the third level (L3) is an standard checkpoint to the PFS. Our model is based on this work but we modify it and extend it using the techniques presented in the previous subsections.

We focus on the L2 of such three-level scheme. In the worst case scenario, the Partner and XOR techniques will fail to tolerate two simultaneous failures (See section 3.1). Future large and dense machines will probably suffer from correlated failures with a higher frequency than current systems (See section 2.2), thus we motivate the use of more reliable erasure codes, such our topology-aware RS encoding technique.

Moreover, in addition to the significant increase in reliability, we improve the performance of the L2 checkpoint by efficiently hiding the encoding work using one extra FT-dedicated thread per node. In this way, we build a highly-reliable low-overhead L2 checkpoint. The other two levels are Local checkpoint on SSDs for L1 and PFS-based checkpoint for L3. Also, we improve the L3 checkpoint by writing the checkpoint in local SSDs and then flushing the checkpoint files to the PFS in parallel with the application execution, similar to the OpenMPI staging option [44].

It is important to notice that we should be careful while using the standard formula proposed by Young et al. [45] for the checkpoint interval calculation for our L2 checkpoint because the L2 check-

point time and the L2 checkpoint cost are different. Indeed, the L2 checkpoint cost is the short time the application is stopped to store the checkpoint in the local SSDs and the L2 checkpoint time includes the encoding time that is hidden. Since the writing into SSDs time is very short in comparison with the encoding time, using the standard formula may lead us to a L2 checkpoint interval shorter than the encoding time. For this reason, we should always verify that the L2 checkpoint interval is larger than the encoding time, but short enough to keep the FT-dedicated threads almost constantly working to guarantee a high L2 checkpoint frequency. In addition, between two L2 checkpoints the library can perform one or several L1 checkpoint in order to deal with the increasing SER and other transient failures (See section 2.2). The L2 checkpoint can deal with one or multiple hard node failures while the L3 checkpoint is expected to be used in very rare cases since the L2 checkpoint should already guarantee a very high reliability level.

## 3.5 Reliability study

In our multi-level checkpoint model, the L1 checkpoint is able to tolerate every transient failure where the checkpoint data is not lost. In addition, the L2 checkpoint is designed to tolerate one or multiple node hard failures where part of the checkpoint data needs to be regenerated. In order to calculate the L3 checkpoint frequency we will determine the reliability of our L2 checkpoint technique using a probabilistic method. As explained in section 3.1, the system is partitioned in groups of size k and each group can tolerate $k/2$ node hard failures. For simplicity we will assume in this section that the total number of nodes n in the system is multiple of the group size k. We also assume for this study a random distribution of the failures.

Ideally, the probability for a system to experience a multiple node failure is low. When such failure occur, the system will lose part of the checkpoint data and it may be unable to recover it. However, the capacity to re-construct the lost data will depend on the distribution of the failures among the groups. This is true for XOR encoding as well, if at least one of the groups experience two failures simultaneously, the XOR technique will be unable to recover the lost data. In our model, if at least one group experiences $(k/2)+1$ failures simultaneously, the RS encoding will be unable to recover the lost data. More generally, when at least one group experiences more failures than its tolerance rate t, the L2 technique will fail to restart the execution, we call such failure a catastrophic failure.

$$\Pr(x \cap x_{Ct.}) = \Pr(x) * \Pr(x_{Ct.} \mid x) \tag{3}$$

The probability for a system to experience a catastrophic failure of x nodes depends on the probability of expiriencing a failure of x nodes and the probability of such failure to be catastrophic, as expressed in formula 3.

| Parameter | Description |
|---|---|
| n | Total number of nodes in the system ($n = k * g$) |
| k | Size of the encoding groups |
| g | Number of encoding groups |
| t | Number of node failures tolerated per group |
| x | Number of failed nodes for a given failure |

**Table 1: Resiliency study parameters**

It is difficult to predict $\Pr(x)$ for future post-petascale systems, for our evaluation we propose several scenarios using an exponential distribution based on a study of the failures of TSUBAME1 on the last 4 years (See section 5.2). In contrast, $\Pr(x_{Ct.} \mid x)$ can be calculated using combinatorial theory.

We can express $\Pr(x_{Ct.} \mid x)$ as the probability for a failure to be catastrophic, given x nodes hard failures. We compute the probability of those x nodes to be distributed in such way that at least one group of the system contains $t+1$ of those x failed nodes. We call such distribution of the failed nodes a catastrophic distribution. Then, $\Pr(x_{Ct.} \mid x)$ can be expressed as the ratio between the number catastrophic distributions and the total number of possible distributions, as presented in formula 4 using the parameters presented in table 1.

$$\Pr(x_{Ct.} \mid x) = \frac{\binom{g}{1} * \binom{k}{t+1} * \binom{n-(t+1)}{x-(t+1)}}{\binom{n}{x}} \tag{4}$$

Using formula 3 and 4 together, we can model the reliability of a L2 checkpoint that uses erasure codes, in large scale systems for a given failure rate. This failure rate can be estimated analyzing previous failure records. In our evaluation we will use these formulas to compare the reliability of a system using XOR encoding and RS encoding, for several failures rate scenarios (See section 5.2).

# 4. FTI IMPLEMENTATION

In this section we explain how FTI works, we will give some details about its implementation and optimizations.

## 4.1 FTI mechanisms

FTI is Fault Tolerance Interface that aims to add a highly reliable layer between the operating system (OS) and the application. It was implemented with C/MPI and Python in an effort to stay portable to a wide variety of different platforms. The applications can easily benefit from FTI functions by simply linking the library.

Since FTI spawns one extra MPI process per node it is important to guarantee that the library will not cause any damage to the application communication channels. Therefore, FTI has an initialization call, *FTI_Init()*, that will do all the necessary actions before the application starts the real execution. *FTI_Init()* will start by reading the configuration file that should be correctly filled by the user before the execution. Once the configuration has been checked, FTI will detect in which node is each process and will write this topology in a special file. Then, it will delegate one process per node as FT-manager and will create two MPI communicators, one for the FT-managers and another for the application processes. Then, FTI will create the encoding groups and create one MPI communicator per group, so the encoding groups are kept independent. Each group will then generate the RS encoding matrix and the FT-managers will then wait, ready to encode the checkpoint files.

The MPI communicator created by FTI for the application processes is called *FTI_COMM_WORLD* and it will replace the global communicator used in the application (*MPI_COMM_WORLD*). By simply replacing the global communicator in such way, FTI guarantees that no message will ever be exchanged between application processes and FT-managers within the application. Messages exchanged between application processes and FT-managers will be generated only by FTI calls.

In the case of a failure, the user just needs to modify one parameter in the configuration file and launch again the execution. *FTI_Init()* will then notice that the application needs to restart from the last checkpoint. It will start by recreating the same topology as previously saved and it will check if there has been lost of data due to a node hard failure. In such case, the RS decoding will be

launched to regenerate the lost data. In the exceptional case of losing more data than tolerated by the L2 scheme, FTI will then search for the last checkpoint saved in the PFS.

Finally, *FTI_Finalize()* check that all the FT-managers have finished their job and free all the resources. For the evaluation done in section 5.4 we added FTI support to a version of SPECFEM3D capable to do a basic application-level checkpoint to PFS. The process took less than one hour and only a few tens of extra code lines.

## 4.2 Toward auto-tuning

Our algorithm has been optimized by overlapping computation and communications (See section 3.2). However, when the number of communications is large, the overhead caused by the network latency will be high. Thus, it is important too limit the number of communications by selecting an appropiate block size. Since every machine is different and the network performance change significantly from an architecture to another, it is difficult to fix in advance a block size that match perfectly the network performance of every supercomputer. One will need to try several different block sizes to get best performance.

FTI has a set of commands to automate this process. The library tries a wide range of block sizes to find the block size with the fastest encoding. Figure 3 shows an example of this auto-tuning process on TSUBAME2.0. As we can see, the small blocks produced a slow encoding due to the overhead generated by the network latency. From 2KB blocks, the encoding time starts to stabilize, which means that the communications are totally hidden by the encoding computation despite the high network bandwidth of TSUBAME2.0 at 6.5GB/s effective measured.

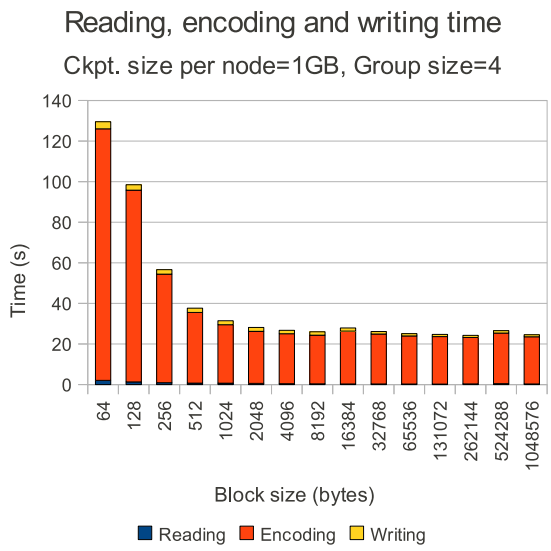



**Figure 3: Reed-Solomon encoding time for different block sizes**

Moreover, FTI also measures the local read and write performance of the machine. Since we have integrated our model in the core of FTI, the library can use the measured data to propose an appropriate checkpoint frequency for a given checkpoint size per node and group size. We are currently planning to add to the library a dynamic reconfiguration during the execution for applications that have different workloads at different stages of the execution.

# 5. EVALUATION

Our evaluation is divided in three parts. In the first one, we demonstrate the correctness and accuracy of the encoding and decoding performance of our model. The second part is related with our reliability study (See section 3.5) and we present a short study of the failure records of TSUBAME1 during the last 4 years. Finally, we evaluate the performance, scalability and efficiency of our library FTI using SPECFEM3D. All our experiments were done in TSUBAME2.0 with the configuration given in table 2.

| | |
|---|---|
| Nodes | 1408 High BW Compute Nodes |
| CPU | 2 Intel Westmere-EP 2.93GHz 12Cores/node |
| Mem | 55.8GB or 103GB (Total: 80.55TB) |
| GPU | NVIDIA M2050 515GFlops, 3GPUs/node<br>(Total: 4224 NVIDIA Fermi GPUs) |
| SSD | 60GB x 2 = 120GB (55.8GB node)<br>120GB x 2 = 240GB (103GB node)<br>(Total : 173.88TB)<br>Write speed : 360MB/s (RAID0) |
| Network | Dual rail QDR IB (4GB/s x 2) |
| File system | 5 DDN DFA10000 units (3 Lustre and 2 GPFS)<br>with 600 2TB HDDs each<br>Measured Lustre write troughput (10GB/s) |
| OS | Suse Linux Enterprise + Windows HPC |

**Table 2: TSUBAME2.0 architecture**

## 5.1 Performance model evaluation

The first step in our evaluation will be to demonstrate the correctness and accuracy of our performance model. The accuracy of our performance model is important because it allow us to predict the encoding performance, which in turn is neccesary to compute an approrpiate L2 checkpoint interval and the $L1/L2$ ratio. To prove this, we will try to predict the checkpointing and encoding time using the theoretical performance of the machine and our performance model. Particularly, we will study how the encoding time evolves in relation with the parameters given in the formula 2 (See section 3.2).

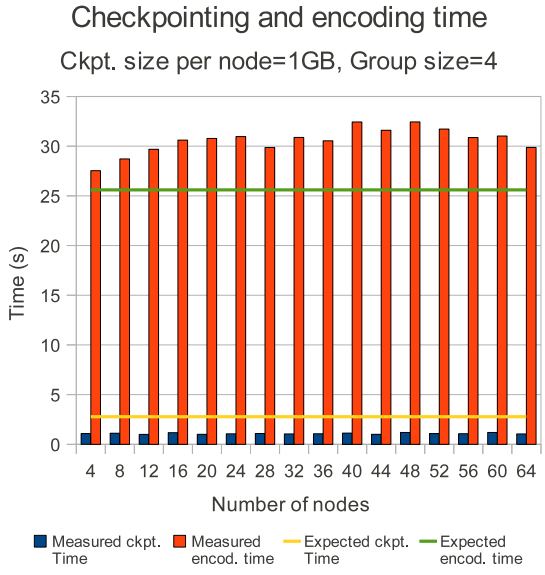
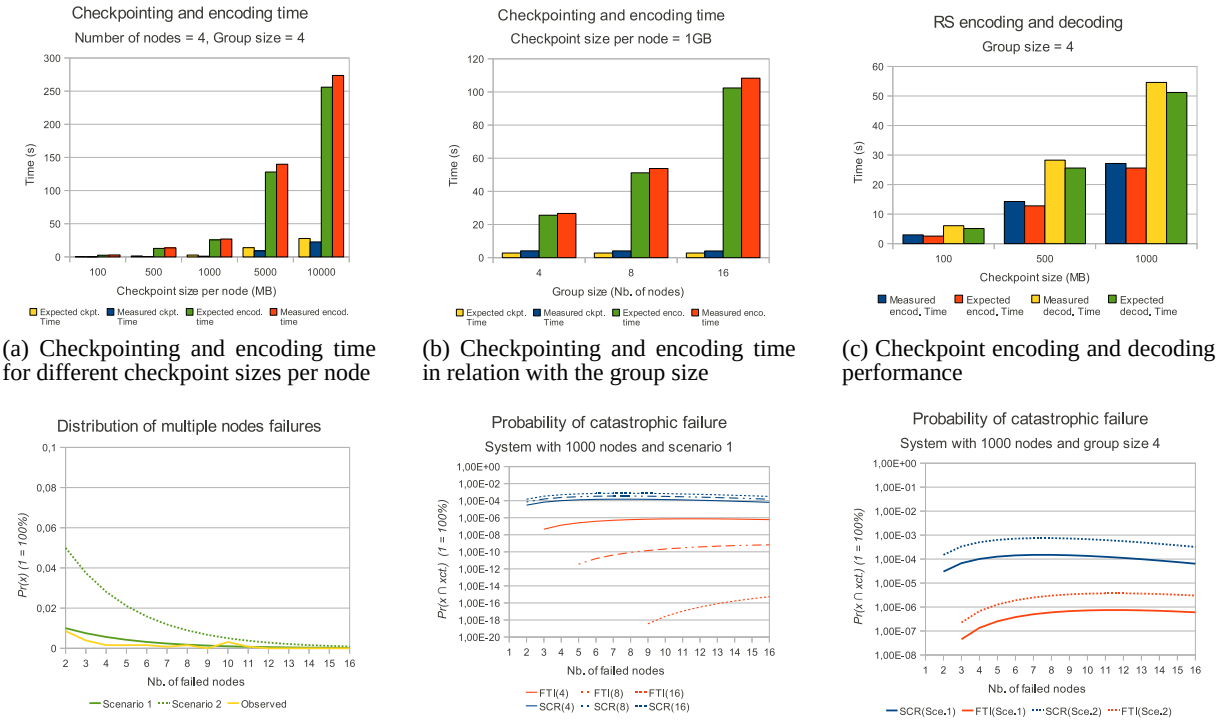



**Figure 4: Checkpointing and encoding time in relation with the number of nodes**




(a) Checkpointing and encoding time for different checkpoint sizes per node

(b) Checkpointing and encoding time in relation with the group size

(c) Checkpoint encoding and decoding performance

(d) Distribution of multiple nodes failures in TSUBAME1 and possible future scenarios

(e) Reliability comparison between FTI and SCR for different group sizes

(f) Reliability comparison between FTI and SCR for two different scenarios

**Figure 5: Performance model and reliability evaluation**

In previous work [5], we showed the scalability of our encoding algorithm in number of nodes, thus we start with this evaluation in the new machine. As explained in section 4.2, the library can determine by itself the best block size and the encoding speed of the machine. For this evaluation, we use a block size of 4KB and the encoding speed is around 25$\mu$s per block ($e * z = 25\mu s$). We fix the checkpoint size per node to 1GB and the group size is 4 nodes. The write speed on local SSDs is given in table 2.

Using formula 2 and these specifications, we can expect a checkpoint time of about 3s and an encoding time of about 25s, independently of the number of nodes. Figure 4 shows that the measured encoding work takes 30s in average with some slight variations. It is normal for the measured encoding time to be slightly longer than predicted by the model because of the traffic on the network. In our model we do not take into account the slight fluctuations of the network due to the traffic generated by the application. On the other hand, the measured time to write the checkpoint on local SSDs is shorter than the hardware specifications. This is normal too since the checkpoint size is not big enough to erase the buffering effect.

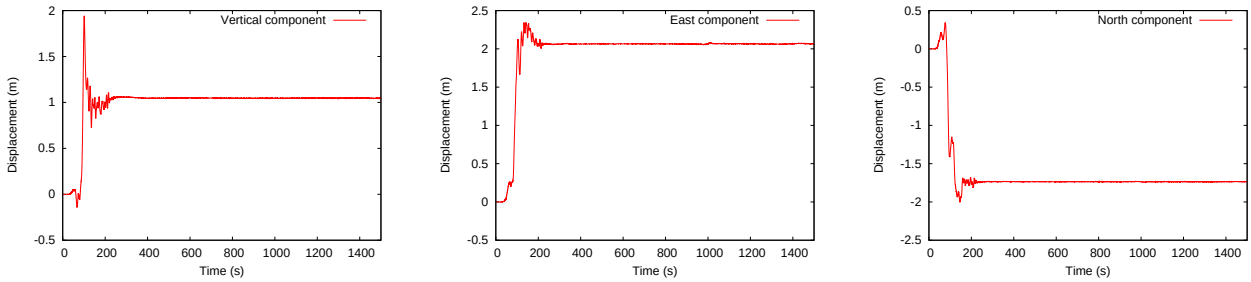
Once we have verified the scalability in number of nodes, we will stress our model to evaluate how the checkpoint and the encoding time evolves in relation with the checkpoint size per node. We use the same block size and group size than in the previous test. By changing the checkpoint size per node we are changing the parameter h in formula 2, thus the encoding time should increase linearly with respect to the checkpoint size per node. Also, the checkpoint writing time should evolve according to the write bandwidth of the SSDs. Similarly to the previous test, the encoding time is slightly superior than predicted by the model and the writing time slightly shorter, as presented in figure 5a. However, we can see that the measured performance match pretty well the linear evolution expected. The writing time in particular, pass from less than 1s for 100MB to more than 20s for 10GB. We can see that the buffering effect decrease rapidly when the checkpoint size increase, therefore approaching the hardware specifications.

Since the reliability of the system increase with the group size, it is important to evaluates how the encoding performance evolves with respect to the group size. Hence, we continue stressing our model by measuring the checkpointing and encoding time for different group sizes. We fix the checkpoint size per node to 1GB and we vary the group size from 4 to 16 nodes. As presented in figure 5b, the checkpoint writing time will remain constant since the checkpoint size per node remains constant. In contrast, the encoding time will increase linearly with the group size as predicted on our model. This was expected because the parity data generated by our topology-aware RS encoding is directly related with the size of the group (See section 3.1).

Finally, after stressing all the parameters of formula 2, we will study the decoding time. As explained in section 3.2, the decoding time should be twice longer than the encoding time. Figure 5c, shows the measured and expected decoding time in comparison with the encoding time for three different checkpoint sizes per node (100MB, 500MB and 1000MB). As we can see, the measured decoding performance match very well the expected times.

## 5.2 Reliability comparison

In this section we study the reliability of our proposed topology-aware RS technique (L2 of FTI) and compare it with the XOR technique used in L2 of SCR. For this purpose, we will use formulas 3 and 4 and the parameters of table 1 (See section 3.5). The number




(a) (b) (c)

**Figure 6: Synthetic seismograms of the three components of the displacement vector measured at the surface of the Earth at the Hirono station in the Fukushima prefecture. The fact that some components of the displacement vector do not go back to zero after the main wave fronts corresponds to the so-called 'static offset': the earthquake was so large that it permanently tilted the surface of Japan.**

of failures tolerated per group $t$ is 1 for SCR and $k/2$ for FTI.

First, we compute the probability of a failure of $x$ nodes ($\Pr(x)$) striking the system. The best way to compute it is studying failure traces of supercomputers. We analyzed a total of 1280 failures that occurred in TSUBAME during the last 4 years. This list of failures is available online [46]. From this list we noticed that less than 5% of failures affected more than one node. The failures affecting several nodes simultaneously could affect from 2 to sometimes more than 10 nodes, following the distribution *Observed* in figure 5d.

In order to establish a plausible scenario for future machines, we propose *Scenario 1* as an optimistic case, in which the average of multiple node failures will remain the same as in previous clusters. However, the failure records we studied are related to a machine that was, in this context, small at that time (from 2006 to 2010). Indeed, TSUBAME1 was composed by 655 nodes, which is almost one half of the current TSUBAME2.0 size and even more in comparison with future post-petascale systems. Such large and dense supercomputers may suffer correlated failures in a more common pattern (See section 2.2). Thus, we propose a second scenario (*Scenario 2*) where almost 20% of failures will affect several nodes simultaneously. The failure distributions of both scenarios are plotted in figure 5d.

In figure 5e, we studied the influence of the group size in relation with the probability of catastrophic failure for a system similar to TSUBAME2.0 (about 1000 nodes) assuming the optimistic case of *Scenario 1*. The first thing that we notice is that both libraries are very reliable and they have a very low probability of catastrophic failure for a low rate of multiple node failures. Nonetheless, there are some important discrepancies between both libraries. Not surprisingly, the reliability of FTI increases by several orders of magnitude by only increasing the group size. This is normal because in our model the reliability is directly linked to the group size. In contrast, while using XOR, the reliability decreases when the group size increases because the probability of two or morefailures occurring in the same XOR set will increase. Then we can see, how the inherent encoding technique can impose some limitations. Even for the smallest group size (4 nodes), the topology-aware RS encoding of FTI guarantees about two orders of magnitude more reliability than the XOR encoding.

Although this is an important difference, both libraries seem to guarantee a low enough probability of catastrophic failure. Then, we decided to stress the comparison by studying how both libraries evolve in relation with the multiple node failure rate. The results are shown in figure 5f. As presented, an increase of 15% in the multiple node failures rate will increase the probability of catastrophic failure of both libraries L2 technique, by almost one order of magnitude. This proves that both techniques are sensitive to the failures rate and failures patterns, but FTI can benefit from its improved reliability.

Moreover, FTI not only proposes an FT-enhanced L2 technique but also a low cost encoding thanks to the FT-dedicated threads. FTI guarantees a high reliability with a low overhead, for large scale HPC as presented in section 5.3.

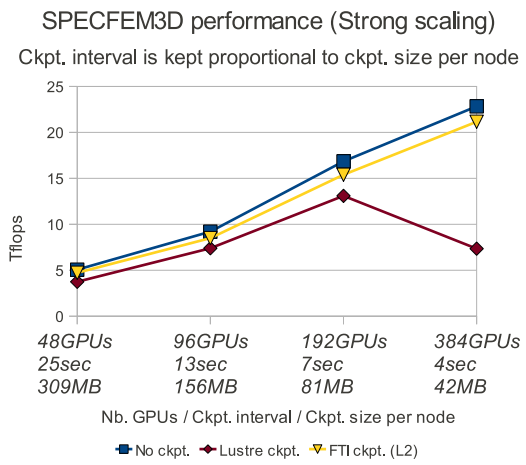
## 5.3 Simulating the March $11^{th}$ Mw9.0 Tohoku Japan earthquake

In an effort to extend our evaluation with a functional test of FTI in a real case simulation with a production level application, we decided to simulate the devastating Mw9.0 Tohoku Japan earthquake that struck the northeast part of the island on March $11^{th}$, 2011. The simulation is done with SPECFEM3D on TSUBAME2.0 using an input model that describes the source fault.

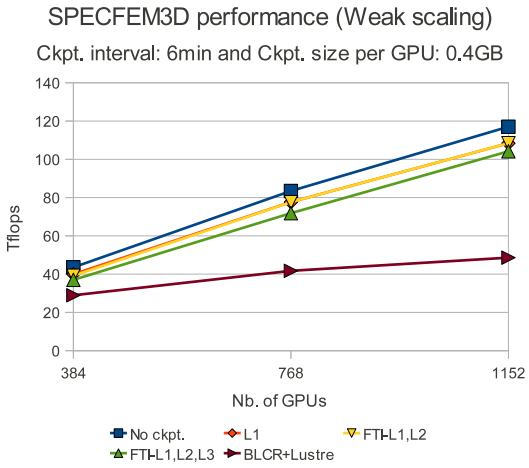
SPECFEM3D is used by more than 300 research groups in the world for a large number of applications, for example to model the propagation of seismic waves resulting from earthquakes, seismic acquisition experiments carried out in the oil industry, or laboratory experiments with ultrasounds in crystals. Also, this application won the prestigious Gordon Bell SuperComputing award for Best Performance [37] for a calculation of seismograms in the whole 3D Earth down to periods of approximately 5 seconds, carried out at 5 teraflops (sustained) on 1944 processors using 14.6 billion degrees of freedom stored in 2.5 terabytes of memory on the Earth Simulator, the fastest computer in the world at that time.

For the source model, we apply the waveform inversion [52,53] to obtain slip distribution in the source fault at the 2011 Tohoku, Japan earthquake in the same manner as Nakamura et al. [56]. We use broadband seismograms of IRIS GSN and IFREE OHP seismic stations with epicentral distance between 30 and 100 degrees. The broadband original data are integrated into ground displacement and band-pass filtered in the frequency band 0.002-1 Hz. We use the velocity structure model IASP91 [51] to calculate the wavefield near source and stations. We assume that the strike of the fault plane is 201 degree and the dip angle is 9 degree, based on Global Centroid Moment Tensor solution. The length of a subfault is 20 km along strike. The assumed fault length is totally 440 km consistent with the aftershock distribution.

The nonnegative least-squares method [55] is employed for constraining the rake angle in the waveform inversion. The results of




(a) Strong scalability




(b) Weak scalability

**Figure 7: FTI Scalability**

the inversion show the bilateral rupture to the northeast and the southwest with two main asperities along the fault; the maximum slip of around 40 m with the reverse fault mechanism at approximately 100 km northeast of the epicenter and another large slip with reverse fault mechanism at 100 km southeast of the epicenter. The total amount of the released seismic moment corresponds to moment magnitude $Mw = 9.1$. We calculate synthetic seismograms with this source propagation model for a realistic 3D Earth model using the spectral-element method [54, 57]. At this moment, it is not possible to compare the synthetics with the observed seismograms because the seismograms from Japanese stations are not still available due to the network trouble after the earthquake. However, as we can observe in figure 6b, the synthetic seismogram for the Hirono station located in the Fukushima prefecture, the E component shows about 2m static displacement to the East, which seems to be consistent to the observed crustal deformation caused by this earthquake. Figures 6c and 6a show the other two components at the same station.

Because of the complexity to determine a source input model and the rolling blackouts in Japan, we only could launch several high resolution simulations to demonstrate the source model accuracy. During the following months we plan to make more experiments using the new source model which should not have any impact on the results presented in the next section. In addition, we plan to launch a large simulation on the full machine to evaluate the performance of FTI at larger scale, but also to create a high resolution movie of the seismic waves generated by the earthquake in order to better understand what happened in the Tohoku region of Japan on March $11^{th}$, 2011.

## 5.4 FTI scalability study with SPECFEM3D

In order to demonstrate the efficiency and scalability of FTI we decided to evaluate it at large scale with SPECFEM3D. Recently, SPECFEM3D was ported to GPU clusters using CUDA [47, 48], so it can be used in hybrid systems such as TSUBAME2.0. It is important to notice that such seismic simulations generally do not need double precision and will perform most of their runs in single precision [47]. Also, we want to highlight that SPECFEM3D is a memory-bound application, as any finite difference or finite element code; this is intrinsically related to the fact that in such numerical methods few operations are performed per grid point, and thus the cost comes mostly from memory accesses [47, 48].

First, we start with a strong scalability test in which we evaluate the performance of SPECFEM3D in 3 cases: no checkpoint is done, checkpointing with FTI (L2) and checkpointing on Lustre [13]. Since the problem size is fixed, the memory used (and therefore the checkpoint size) per GPU decreases when the number of GPUs increases. In this experiment, for the FTI tests all the checkpoints were done with the L2 of FTI, thus we do not take advantage of the multi-level scheme of FTI at this point. Since the checkpoint size decreases, we also decrease the checkpoint interval in order to decrease the recovery cost in case of failure. All the checkpoints are done at the application level and we checkpoint only the strictly necessary data in order to restart the execution; this corresponds to about 20% of the memory used by the application.

As we can see in figure 7a, SPECFEM3D strong-scales well from 5 TFlops on 48GPUs to almost 23 TFlops on 384GPUs without checkpointing. FTI follows closely this progression by causing only about 4% of overhead for 384GPUs. In contrast, checkpointing to Lustre becomes prohibitively costly at high frequencies. The performance was measured using PAPI to measure the floating point operations [49] and each dot in the figure is the average of 5 runs.

To evaluate the overhead of FTI at large scale, we stressed even more our library by weak-scaling to more than 1000 GPUs. In this second experiment, we populate the GPUs memory with 2.1GBs of data, out of 2.6GBs available for the user (12.5% is used for ECC in Fermi GPUs [20]) and we kept the checkpoint interval fixed to 6 minutes, which is the Young's optimal checkpoint interval for a MTBF of 12 hours and a L1 checkpoint of 2 seconds. Then, we run SPECFEM3D for several configurations: The first one is without checkpoint (No ckpt.); the second one is checkpointing to the local SSDs without any encoding (L1); the third one is using FTI, thus in addition to the local checkpoint, every 2 checkpoints FTI will use the RS encoding proposed in our model (FTI-L1,L2); the fourth one is similar to the precedent one but in addition every 6 checkpoints the latest checkpoint files are flushed to Lustre (FTI-L1,L2,L3); and finally checkpointing with BLCR on Lustre (BLCR+Lustre). Although there are some works [36] going on to make it possible, BLCR cannot checkpoint GPU-accelerated sys-

tems. Hence, we emulate it by writing 2.1GBs of data per process (therefore per GPU) to Lustre in the same way BLCR would do it. It is important to highlight that BLCR, as any other kernel-level checkpoint, will save the complete memory of every process, creating a 5 times larger checkpoint.

In figure 7b, we can see that SPECFEM3D has an almost perfect weak scaling, from 43TFlops to 117TFlops on 1152GPUs for the No ckpt. test. Also, in the figure the L1 results are actually hidden by the FTI-L1,L2 results. Indeed, both scenarios achieve almost identical results causing about 8% overhead in comparison with the No ckpt. case. This means, that the RS encoding done at L2 is completely hidden thanks to the FT-dedicated threads. The L1 checkpoints, capable to tolerate transient failures, are done between two L2 checkpoints while the FT-threads are still encoding the previous, more reliable, checkpoint. The FTI-L1,L2,L3 scheme adds an extra 3% overhead due to the Lustre writing performance. Finally, the BLCR+Lustre scheme imposes an always larger and prohibitive overhead as the size of the problem increases. For each run we let the application run between 30 and 40 minutes and every point in the figure is the average of 3 runs.

At this point, we have achieved over 100TFlops of sustained performance with a production-level scientific application such as SPECFEM3D, on an hybrid supercomputer such as TSUBAME2.0 and checkpointing with our library FTI every 6 minutes (high frequency checkpointing). In the coming months, we expect to launch a larger evaluation in the full machine.

# 6. RELATED WORK

Some applications can benefit from algorithm-based checkpoint-free techniques [27]. Such techniques are very efficient and should be applied when possible. However, many applications would need other resiliency schemes. CR is the most popular technique in HPC and can be implemented at kernel level[21, 22, 23, 24] or at user level[25]; and they can include optimizations such as incremental checkpoint[25] or speculative checkpoint[26]. Unfortunately, such optimizations will have different results depending on the application. Diskless checkpoint [14, 15, 19, 28, 29, 30] was proposed by Plank et al. as a solution to avoid the I/O bottleneck; however several challenges, such as how to optimize the encoding algorithms and how to improve the time and memory efficiency of that technique, were important points against its adoption.

In this paper, we extend our previous work by proposing a low-overhead high-frequency multi-level checkpoint scheme and its performance model, and we evaluate its correctness. Also, we conduct a reliability study and we compare the reliability of two different erasure codes used for fault tolerance, in two different scenarios. In addition, we implement a Fault Tolerance Interface FTI, and we evaluate its performance using a real case scenario with a production level application such as SPECFEM3D on TSUBAME2.0. Furthermore, we stress our evaluation by scaling, for the first time, to over 1000 GPUs, achieving over 100TFlops of sustained performance while checkpointing with FTI at a high frequency.

PLFS[4] is a parallel log structured file system that enhances the underlying PFS performance for some access patterns by remapping the application data layout. This is an important optimization for those applications with special data access patterns. However, the applications still limited by the upper bound PFS performance. We believe this work can be complemented with our library. Indeed. PLFS is capable to transform a N-1 write access pattern into a N-N write access pattern by using a container structure implemented as hierarchical directory tree. This feature could be adapted to systems with non-volatile storage on the compute nodes where each process file will be stored in local, increasing significantly the writing performance. By coupling this with FTI, one can guarantee the files availability even in case of multiple hard failures.

Dong et al. proposed an interesting hybrid checkpoint model using PCRAM [2]. While this work seems very promising, we believe some of the assumptions and requirements may be relaxed if that technology is coupled with our model. For instance, one could still use hard disk drives (HDDs) in the PFS (i.e. 3DPCRAM + HDD) by using our topology-aware RS encoding technique. Since the encoding time is efficiently hidden by the FT-dedicated threads, the erasure codes will not be an obstacle for performance.

Finally, Moody et al. proposed a multi-level checkpoint and a probabilistic Markov model [1] that describes the performance of such multi-level scheme. That multi-level technique is probably the closest one to our work. In fact, the techniques proposed in our model are somehow based on several results presented in that work, such as the Markov model and the adaptation for disk-less systems proposed. Nonetheless, there are several important differences. We propose a reliability-enhanced L2 that can match the performance of a L1 checkpoint. In addition, we present this work in the context of hybrid systems and we exploit some of the characteristics of GPU computing to improve the performance of our library. We present an evaluation with SPECFEM3D on TSUBAME2.0 (over 1000 GPUs) in which we achieve over 100TFlops of sustained performance while checkpointing at high frequency. These two works are also complementaries, SCR could implement our FT-dedicated thread scheme as an optional feature to enhance the L2 checkpoint performance. Also, we plan to add to FTI, the XOR encoding used in SCR, as an intermediary level between the L1 checkpoint and the topology-aware RS encoding, giving as result a 4-level checkpoint library. We believe that by complementing XOR L2 with RS L3 checkpointing we can further improve the performance while reducing the need of checkpointing to the PFS.

# 7. CONCLUSION

In this work we have proposed a highly reliable technique based on a topology-aware RS encoding. Also, we have exploited some characteristics of GPU computing trough which many GPU applications are capable to spawn one extra FT-dedicated thread per node in order to improve the checkpoint performance. We have integrated both techniques in a multi-level checkpoint model, that we have implemented in our FTI library and we have conduct an exhaustive study of correctness of our preformance model and the reliability of our library.

Moreover, we have conducted for the first time a large scale evaluation of such multi-level technique with a production level scientific application in an hybrid platform. Our evaluation with SPECFEM3D on TSUBAME2.0, shows that FTI imposes only 8% of checkpoint overhead while running at over 0.1 petaflops and checkpointing every 6 minutes.

As future work, we plan to add the XOR encoding to our library and continue to develop more auto-tuning strategies, such as dynamic behavior for applications with different workloads at different stages of the execution.

# 8. ACKNOWLEDGMENTS

This work was supported in part by the JSPS, the ANR/JST FP3C project, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

# 9. ADDITIONAL AUTHORS

Additional author: Takeshi Nakamura, Affiliation: JAMSTEC, email: t_nakamura {at} jamstec.go.jp

## 10. REFERENCES


[1] A. Moody, G. Bronevetsky, K. Mohror, B. R. de Supinski, Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System, sc, pp.1-11, 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010

[2] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, Y. Xie. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In Super Computing Conference, Portland 2009.

[3] Z. Cheng, J. Dongarra, A scalable Checkpoint Encoding Algorithm for Diskless Checkpointing, Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium, 2008.

[4] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "Plfs: A checkpoint filesystem for parallel applications," SC Conference, vol. 0, 2009.

[5] L. Bautista-Gomez, N. Maruyama, A. Nukada, F. Cappello, S. Matsuoka, "Low-overhead diskless checkpoint for hybrid computing systems", International Conference on High Performance Computing, Goa, India, December 2010.

[6] L. Bautista-Gomez, N. Maruyama, F. Cappello, S. Matsuoka, "Distributed Diskless Checkpoint for large scale systems", IEEE/ACM International Symposium on Cluster, Cloud and Grid computing (CCGrid2010), Melbourne, Australia, May 2010.

[7] The Top 500 http://www.top500.org/

[8] The Green 500 http://www.green500.org/

[9] F. Cappello, Fault tolerance in Petascale/Exascale systems: current knowledge, challenges and research opportunities International Journal on High Performance Computing Applications, SAGE, Volume 23, Issue 3, 2009.

[10] B. Schroeder, E. Pinheiro, W. Weber. DRAM errors in the wild: A Large-Scale Field Study. SIGMETRICS, Seattle, June 2009.

[11] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies, pages 1–17, Berkeley, CA, USA, 2008. USENIX Association.

[12] F. Schmuck , R. Haskin, GPFS: A Shared-Disk File System for Large Computing Clusters, Proceedings of the Conference on File and Storage Technologies, p.231-244, January 28-30, 2002

[13] S. Microsystems. Lustre file system, October 2008

[14] J. S. Plank, Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications, Technical Report CS-07-603, University of Tennessee, September, 2007.

[15] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, Z. Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage. In Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST), San Francisco, CA, 2009.

[16] S. Matsuoka, The Road to TSUBAME and beyond, Petascale Computing: Algorithms and Applications, Chapman & Hall Crc Computational Science Series, 2008, pp. 289-310.

[17] A GPU Accelerated Storage System, Abdullah Gharaibeh, Samer Al-Kiswany, Sathish Gopalakrishnan, Matei Ripeanu, IEEE/ACM International Symposium on High Performance Distributed Computing (HPDC 2010), Chicago, IL, June 2010.

[18] A. Petitet, R. Whaley, J. Dongarra and A. Cleary. HPL – a portable implementation of the high performance Linpack benchmark for distributed computers. http://www.netlib.org/benchmark/hpl

[19] NA Kofahi, S Al-Bokhitan, A Al-Nazer, On Disk-based and Diskless Checkpointing for Parallel and Distributed Systems: An Empirical Analysis - Information Technology Journal, 2005.

[20] http://www.nvidia.com/object/fermi_architecture.html

[21] J. Duell, P. Hargrove and E. Roman, Requirements for Linux Checkpoint/Restart Lawrence Berkeley National Laboratory Technical Report LBNL-49659, 2002.

[22] E. Roman, A Survey of Checkpoint/Restart Implementations Lawrence Berkeley National Laboratory Technical Report LBNL-54942, 2003.

[23] J. Duell, P. Hargrove and E. Roman, The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart Lawrence Berkeley National Laboratory Technical Report LBNL – 54941, 2002.

[24] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove and E. Roman, The LAM/MPI checkpoint/restart framework: system-initiated checkpointing Proc. Los Alamos Computer Science Institute (LACSI) Symp. Santa Fe, New Mexico, USA, October 2003.

[25] J. S. Plank, M. Beck, G. Kingsley and K. Li, Libckpt: Transparent checkpointing under UNIX. In Proceedings of the USENIX, Technical Conference, 213—223, 1995.

[26] S. Matsuoka, I. Yamagata, H. Jitsumoto, H. Nakada, Speculative Checkpointing: Exploiting Temporal Affinity of Memory Operations, HPC Asia 2009, pp. 390–396, 2009.

[27] Z. Chen and J. J. Dongarra. Algorithm-Based Checkpoint-Free Fault Tolerance for Parallel Matrix Computations on Volatile Resources. Rhodes Island, Greece, april 2006.

[28] J. Plank, K. Li, M. A. Puening, Diskless Checkpointing, IEEE Transactions on Parallel and Distributed Systems, v.9 n.10, p.972-986, October 1998.

[29] J. S. Plank and L. Xu, Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications, NCA-06: 5th IEEE International Symposium on Network Computing Applications, Cambridge, MA, July, 2006.

[30] C. Lu, Scalable diskless checkpointing for large parallel systems, PhD. Thesis, University of Illinois at Urbana-Champaign, IL, 2005.

[31] A. Moody, G. Bronevetsky, Scalable I/O Systems via Node-Local Storage: Approaching 1 TB/sec File I/O LLNL, TeraGrid Fault tolerance for Extreme-Scale Computing, 2009.

[32] S. Matsuoka, T. Aoki, T. Endo, A. Nukada, T. Kato, A. Hasegawa, GPU-accelerated computing—from hype to mainstream, the rebirth of vector computing. J Phys Conf Ser 180, 2009.

[33] B. Schroeder and G. A. Gibson, Understanding failures in petascale computers, SciDAC 2007 J. Phys.: Conf. Ser., vol. 78, no. 012022, 2007.

[34] M. Curry, L. Ward, T. Skjellum, and R. Brightwell. Accelerating reed-solomon coding in raid systems with gpus. In International Parallel and Distributed Processing Symposium, April 2008.

[35] W. D. Gropp, R. Ross, and N. Miller. Providing efficient I/O redundancy in MPI environments. Lecture Notes in Computer Science, 3241:7786, September 2004.

[36] A. Nukada, S. Matsuoka, NVCR : A Transparent Checkpoint-Restart Library for NVIDIA CUDA in Proceedings at the International Heterogeneity in Computing Workshop, Alaska, 2011. (To appear)

[37] D. Komatitsch, S. Tsuboi, C. Ji and J. Tromp, A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator, Proceedings of the ACM / IEEE Supercomputing SC'2003 conference, November 2003.

[38] G. Grider, J. Loncaric, and D. Limpart, Roadrunner System Management Report, Los Alamos National Laboratory, Tech. Rep. LA-UR-07-7405, 2007.

[39] R. A. Oldfield, S. Arunagiri, P. J. Teller et al., Modeling the Impact of Checkpoints on Next-Generation Systems, in MSST'07. Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies, 2007, pp. 30-46.

[40] S. Y. Borkar, Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation, IEEE Micro, vol. 25, no. 6, pp. 10-16, 2005.

[41] D. Reed, High-End Computing: The Challenge of Scale, Director's Colloquium, May 2004.

[42] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, J. Sancho, Entering the petaflop era: the architecture and performance of Roadrunner, Proceedings of the 2008 ACM/IEEE conference on Supercomputing, November 15-21, 2008, Austin, Texas.

[43] B. Schroeder, G. A. Gibson, A large-scale study of failures in high-performance computing systems, Proceedings of the International Conference on Dependable Systems and Networks (DSN'06), p.249-258, June 25-28, 2006.

[44] http://www.open-mpi.org/

[45] John W. Young. 1974. A first order approximation to the optimum checkpoint interval. Commun. ACM 17, 9 (September 1974), 530-531. DOI=10.1145/361147.361115 http://doi.acm.org/10.1145/361147.361115

[46] http://www.gsic.titech.ac.jp/ ccwww/index.php?www&&&/tgc/trouble_list.html

[47] D. Komatitsch, D. Michéa, G. Erlebacher, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, Journal of Parallel and Distributed Computing, vol. 69(5), p. 451-460, doi: 10.1016/j.jpdc.2009.01.006, 2009.

[48] D. Komatitsch, G. Erlebacher, D. Göddeke, D. Michéa, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, Journal of Computational Physics, vol. 229(20), p. 7692-7714, doi: 10.1016/j.jcp.2010.06.024, 2010.

[49] http://icl.cs.utk.edu/papi/

[50] http://www.geodynamics.org/cig/software/specfem3d-globe

[51] B. Kennet, E. Engdahl, Traveltimes for global earthquake location and phase identification. Geophys. J. Int., 105, 429-465, 1991.

[52] M. Kikuchi, H. Kanamori, Inversion of complex body waves. III, Bull. Seismol. Soc. Am., 81, 2335-2350, 1991.

[53] M. Kikuchi, H. Kanamori, Note on Teleseismic Body-Wave Inversion Program, 2003. http://www.eri.u-tokyo.ac.jp/ETAL/KIKUCHI/

[54] D. Komatitsch, J. Ritsema, J. Tromp, The spectral-element method, Beowulf computing, and global seismology, Science 298, 1737-1742, 2002.

[55] C. Lawson, R. Hanson, Solving Least Squares Problems, Prentice-Hall, New Jersey, 340 pp, 1974.

[56] T. Nakamura, S. Tsuboi, Y. Kaneda, Y. Yamanaka, Rupture process of the 2008 Wenchuan, China earthquake inferred from teleseismic waveform inversion and forward modeling of broadband seismic waves, Tectonophysics, vol. 491, 72-84, 2010.

[57] S. Tsuboi, D. Komatitsch, C. Ji, J. Tromp, Broadband modelling of the 2002 Denali fault earthquake on the Earth Simulator, Phys. Earth Planet. Inter. 139, 305-312, 2003.