2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Application Fault Tolerance for Shrinking Resources via the Sparse Grid Combination Technique

Peter E. Strazdins
Research School of Computer Science
Australian National University
Canberra, Australia
Peter.Strazdins@cs.anu.edu.au

Md Mohsin Ali
Research School of Computer Science
Australian National University
Canberra, Australia
md.ali@anu.edu.au

Bert Debusschere
Combustion Research Facility
Sandia National Laboratories
Livermore, USA
bjdebus@sandia.gov

***Abstract*—The need to make large-scale scientific simulations resilient to the shrinking and growing of compute resources arises from exascale computing and adverse operating conditions (fault tolerance). It can also arise from the cloud computing context where the cost of these resources can fluctuate.**

In this paper, we describe how the Sparse Grid Combination Technique can make such applications resilient to shrinking compute resources. The solution of the non-trivial issues of dealing with data redistribution and on-the-fly malleability of process grid information and ULFM MPI communicators are described. Results on a 2D advection solver indicate that process recovery time is significantly reduced from the alternate strategy where failed resources are replaced, overall execution time is actually improved from this case and for checkpointing and the execution error remains small, even when multiple failures occur.

***Keywords*-algorithm-based fault tolerance; ULFM; process failure recovery; PDE solvers; sparse grid combination technique; parallel computing; elasticity; cloud computing**

I. Introduction

Large-scale scientific simulations typically arise from the solution of time-dependent Partial Differential Equations (PDEs), involving the evolution over time of multi-dimensional fields over a physical space. There is however an increasing need to engineer the applications, driving these simulations to be adaptable and resilient to changing computational resources. Various scenarios motivate this:

- very large scale computing: for a system with n components, the mean time between failures (MTBF) is proportional to $1/n$. Thus for a sufficiently large n, a long-running application will *never* finish! By 'failure' we usually mean a transient or permanent failure of a component (e.g. node), although it could be that of simply a process, failing due to temporary lack of resources.
  Under adverse operating condition scenarios, the MTBF can be much smaller, and hence this can be an issue on even moderate scale systems.
- cloud computing: resources (e.g. compute nodes) may have periods of scarcity / high costs. For a long-running application, we may wish to shrink and grow the nodes it is running on accordingly: this scenario is also known as *elasticity*.
  Elasticity is also desirable in the case of transient failures, where for example after reboot, a node becomes available again within the lifetime of the application.

Traditionally, large-scale parallel applications use the *Message Passing Interface* (MPI) [1], which is a widely used standard for parallel and distributed programming of HPC systems. However, the standard does not include methods to deal with one or more component failures at run-time.

Recently, the MPI Forum's Fault Tolerance Working Group began work on designing and implementing a standard for *User Level Failure Mitigation* (ULFM) by introducing a new set of tools to facilitate the creation of fault-tolerant applications and libraries. This draft standard (targeted for Open MPI 3.1) allows the application writers to design recovery methods and control them from the user level, rather than specifying an automatic form of fault tolerance managed by the operating system or communication library [2], [3].

PDEs are normally solved on a regular grid. With uniform discretization across all its dimensions, the number of grid points increases exponentially with the increase of dimensionality. This behavior makes the high-dimensional PDE solver computationally expensive. In order to address this issue, high-dimensional PDEs can be solved on a grid with substantially fewer grid points than the regular full grid. These grids are called *sparse grids* [4].

A numerical method called the *Sparse Grid Combination Technique* (SGCT) [5], [6] is employed to approximate the solutions to PDEs on the sparse grid. Instead of solving the PDEs on a full isotropic grid, it solves them on several *anisotropic* grids with substantially fewer grid points, called the *sub-grids* or *component grids*, as shown in Figure 1. Solutions on these sub-grids are then linearly combined to approximate the solution on the sparse grid. As well as making high-dimensional PDEs tractable, employing the SGCT can result in computational efficiencies even for lower-dimensional problems [5], [6], [7], [8].

/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.210

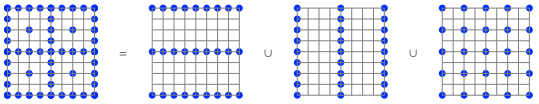


Figure 1. Sparse grid with its components.

The SGCT can also be extended to support algorithm-based fault tolerance [9], [10], [8]. Upon detection of a fault, the SGCT can be applied in such a way that the component grids associated with the failed nodes are avoided. With the development of a highly scalable SGCT algorithm [11], [12], recent work has been done to make various applications fault-tolerant via the SGCT using process or node recovery via ULFM-MPI [10], [8], [13], but these all make the assumption that there are *constant resources* available, i.e. the failed nodes can be replaced by spares. However, not in all situations is this realistic or desirable. For example, the costs of spawning new processes may offset the benefits of their potential extra processing power.

An alternative is to try to continue the computation with *shrinking resources*, accepting there will be some degradation in performance. In this paper, we extend the above previous work by showing how SGCT-enabled fault-tolerant PDE solvers, and the underlying SGCT infrastructure, can be designed to continue with shrinking resources.

This paper is organized as follows. Section II gives background information on the SGCT and its parallel implementation. Design and implementation considerations are detailed in Section III, with experiment results on fault recovery time, overall scaling and approximation error given in Section IV. Related work is discussed in Section V and conclusions are given in Section VI.

II. The Sparse Grid Combination Technique

Consider the SGCT for the 2D case. This will involve solving the PDE on each component sub-grid G_i, where $i = (i_x, i_y)$. G_i is assumed to have $(2^{i_x}+1) \times (2^{i_y}+1)$ grid points with a grid spacing of $h_1 = 2^{-i_x}$ and $h_2 = 2^{-i_y}$ in the x- and y-directions, respectively, where $i_x, i_y \geq 0$. If we consider a square domain, then the grid points of G_i are $\{(\frac{x'}{2^{i_x}}, \frac{y'}{2^{i_y}}) | x' = 0, 1, \cdots, 2^{i_x}, y' = 0, 1, \cdots, 2^{i_y}\}$.

If u_i denotes the approximate solution of a PDE on G_i, the combination solution u_I^c generally takes the form

$$u_I^c = \sum_{i \in I} c_i u_i, \tag{1}$$

where $c_i \in \mathbb{R}$ are the combination coefficients. For the 2D case, good choices of the coefficients are ± 1. For instance, in the classical SGCT of the 'level' l, we have for the set $I = \{(i_x, i_y) | i_x, i_y \geq 0, l-1 \leq i_x + i_y \leq l\}$ and the following combination formula:

$$u_I^c = \sum_{i_x+i_y=l} u_i - \sum_{i_x+i_y=l-1} u_i \tag{2}$$

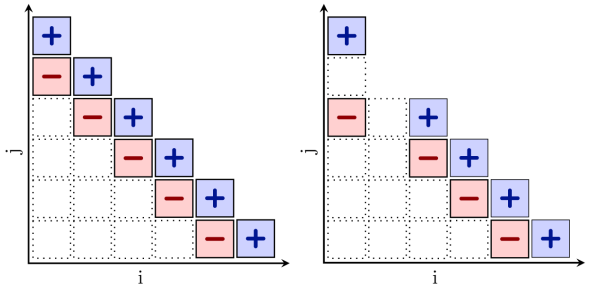


Figure 2. A depiction of the 2D SGCT combination coefficients $c_{i,j}$, where $+$, $-$ and blank represent values of $+1$, -1 and 0, respectively. On the left is a combination of the form (2). On the right is a modified combination of the form (1) which avoids 3 of the solutions required in the original combination instead making use of one smaller solution. Thus this combination serves as an alternative when a failure affects one or more of the 3 solutions that were avoided.

which is depicted in the left half of Figure 2.

A. Fault-tolerant SGCT

A fault-tolerant adaptation of the SGCT (FT-SGCT) has been studied in [9]. When a process failure affects one or more processes involved in the computation, we set $c_i = 0$ for each u_i which was not successfully computed. After a combination, all sub-grids may be restarted from the combined solution, including those which had previously failed.

For the 2D fault-tolerant SGCT computations in this paper, two extra layers (or diagonals) of sub-grid solutions u_i were computed satisfying $i_x + i_y = l-2$ and $i_x + i_y = l-3$. These two extra layers of grids have levels $l-2$ and $l-3$, respectively. During fault-free operation these extra sub-grids are not used in the combination (Equation 2). However, if any of the sub-grid solutions u_i with $i_x + i_y = l$ or $i_x + i_y = l-1$ do not complete due to a fault, these extra sub-grids are used.

B. Parallel SGCT Algorithm

Referring to Equation (1), each PDE instance whose solution is u_i will be run on a distinct set of processes denoted by P_i, arranged in a logical 2-dimensional grid. The SGCT algorithm consists of first a *gather* stage, where each process in P_i sends its portion of u_i to each of the corresponding (in terms of physical space) processes in a logical 2-dimensional grid P^c. For reasons of efficient resource utilization, P^c is made up of a (normally near-maximal) subset of all processes in $\cup_{i \in I} P_i$. Each process in P^c then gathers the $|I|$ versions of each point of the full grid (using interpolation where necessary), and performs the summation according to Equation (1) to get the sparse grid solution u_I^c, which can be used as an approximation to the full grid solution. The use of interpolation in turn requires that a 'halo' of neighbouring points (in the positive direction, for our implementation) have been filled by a halo exchange

operation by each process in each P_i and is also sent in the gather stage.

In the *scatter* stage, each process in P^c sends a down-sample of its portion of u_I^c to the corresponding process in P_i, iteratively, for each $i \in I$.

We assume that the MPI ranks of processes increase contiguously both within each process grid P_i and also between each P_i, when each $i \in I$ is arranged in some canonical order.

Full details of the algorithm may be found in [11]: we use the so-called 'direct' version of the algorithm, due to its superior speed and simplicity.

III. Design and Implementation

Compared to the recovery from faults via replacement processes or nodes, recovery with a shrinking number of processes or nodes appears to be a much more difficult problem when the fields are distributed across processes. This is especially the case in applications made fault-tolerant by the SGCT, because we now have multiple process grids (P_i), and the loss of a process in a grid can affect the MPI rank (among other things) of all process in grids ordered afterwards. However two factors mitigate this:

1) any existing fault-tolerant application must be capable of a restart from the middle of the computation. Depending on how the application is engineered, it should be possible to change parameters affected by shrinkage, i.e. local field sizes and process grid parameters, at this point.
2) the FT-SGCT provides an effectively cost and effort-free method for the required data redistribution.

In this section, Section III-A describes issues with shrinkage within the SGCT. Section III-B describes how process failure detection and recovery for shrinkage is performed within ULFM MPI. The support for the parallel SGCT algorithm for shrinkage, mainly an enhanced data structure for representing process grids (P_i), and how they may be shrunk, is described in Section III-C. Finally how the PDE solver itself has to be adapted for fault tolerant shrinkage is described in Section III-D.

A. Shrinking Based Recovery

In this technique, fault tolerance is achieved without spawning the replacement processes. After detecting the process or node failures, the faulty communicator is shrunk, containing only the alive processes to complete the rest of the computation. With an SGCT-based application, this is achieved by shrinking the process grid and updating the data structures of the sub-grids that are experiencing failures. The sub-grids whose processes are not lost continue their operations without disruption. However, after shrinking the communicator, a mapping of processes is required to access the appropriate data owned by each sub-grid.

An example of the overall approach is shown in Figure 3. The process grid for each sub-grid without any failures is shown in Figure 3a. Suppose, processes 8 and 38 fail. After this failure, the communicator is shrunk excluding processes 8 and 38. The process grids of sub-grids containing processes 8 and 38 are shrunk as shown in Figure 3b. The data structures of each process of these process grids are updated to adjust the whole range of grid points of the corresponding sub-grids. The remaining processes of the other sub-grids do not need to update their data structures. But a mapping of processes is required to point to the appropriate sub-grid data. As for example in Figure 3b, processes 38 to 41 of the shrunken communicator play the role of processes 40 to 43 of the un-shrunken communicator. Otherwise, a disruption in communication and/or unexpected data transfer will happen. The process grid for each sub-grid after shrinking the faulty communicator is shown in Figure 3c.

Comparing Figures 3a and 2 indicates the ordering we use to delineate the process grid index space I.

B. Communicator Recovery via ULFM MPI

The recovery method for process shrinkage is similar to that for process replacement, as is described in more detail in [10].

Firstly, an ULFM MPI error handler must be created and set as described in [10], in such a way that the address of `MPI_Comm ftComm`, initialized to the SGCT application's communicator world, is passed to the handler.

Each time before we invoke the parallel SGCT algorithm, we call `MPI_Barrier(ftComm)` to detect any failure.

If a failure is detected, we revoke the communicator via `OMPI_Comm_revoke(&ftComm)` and create a shrunken communicator via `OMPI_Comm_shrink(ftComm, &shrunkComm)`. Then we synchronize the system via `OMPI_Comm_agree(ftComm=shrunkComm, ...)`.

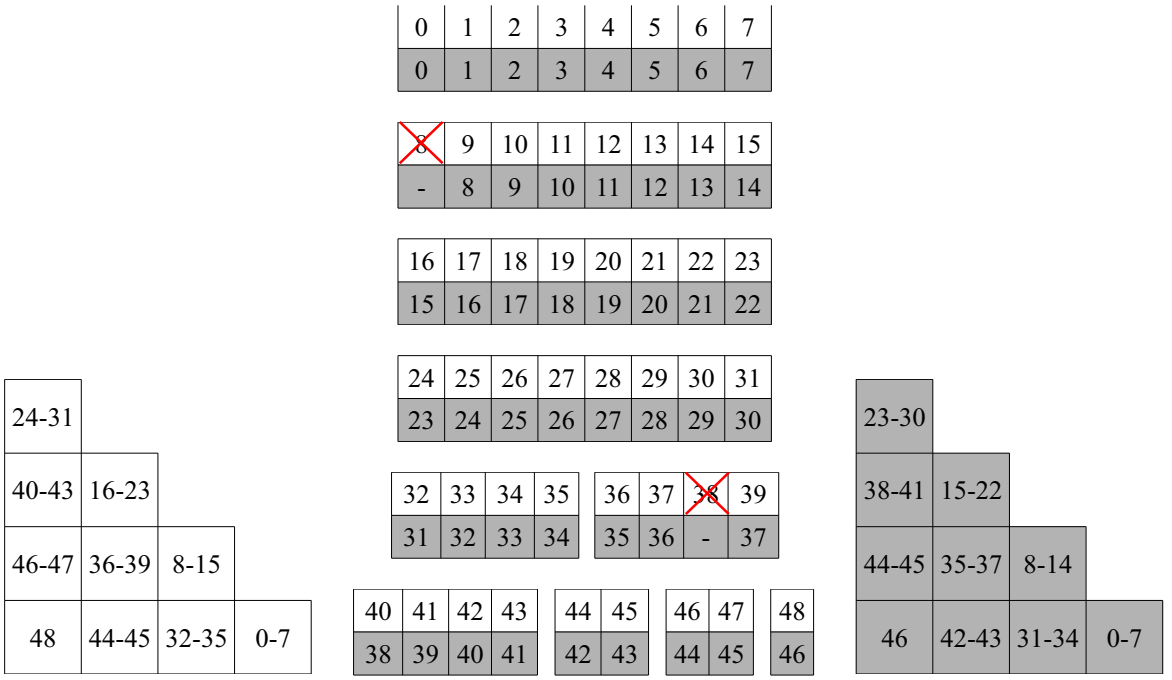
Finally, we reset any local variables holding the MPI rank via `MPI_Comm_rank(ftComm, myrank)` and the number of MPI processes via `MPI_Comm_size(ftComm, nprocs)`.

C. SGCT Algorithm Support

For a (2D) SGCT-enabled application, the computation of solutions on different sub-grids is embarrassingly parallel and each sub-grid is assigned to a different (2D) process grid. Each sub-grid then uses a domain decomposition according to the process grid. The solutions are combined in parallel using a *gather-scatter* approach. This is all handled by setting up process grids with corresponding process maps which indicate the MPI ranks associated with each logical process in the grid. This section details the data structure for these process grids and how it supports shrinkage.

The data structure contains the fields:

- n_p, the total number of processes available.



(a) process grids before shrinking the communicator

(b) detailed process grid shrinking and process mapping

(c) process grids after shrinking the communicator

Figure 3. Process grid configurations of different sub-grids of the FT-SGCT based 2D applications with level $l = 4$ when the faulty communicator is shrunk as a recovery action. Numbers with white and gray background cells represent the MPI processes before and after shrinking the communicator, respectively. The mark 'X' represents the failure of an MPI process.

- r_0, the lowest MPI rank of the processes associated with the grid. It is assumed that the MPI rank r for each process in the group satisfies $r_0 \leq r < r_0 + n_p$.
- $P = (P_x, P_y)$, the 2D logical process grid shape. The logical process id $p = (p_x, p_y)$ satisfies $(0, 0) \leq p < P$ and has rank given by $r_0 = p_y P_x + p_x$. Initially (before failures) we will also have $p_x p_y = n_p$; afterwards, we may have $p_x p_y \leq n_p$. Note that the $n_p - p_x p_y$ spare processes can be used to offset future process failures in this process grid.

Given $f < n_p$ process failures in P, assuming $P_x \geq P_y$, we resize the process grid to $P \leftarrow (P_x - \lceil f/P_y \rceil, P_y)$ and set $n_p \leftarrow n_p - f$. Finally, we set $r_0 \leftarrow r_0 - f_l$, where f_l is the number of failures in lower-ordered process grids. This is illustrated in Figure 3.

It can be observed that under true process shrinkage under ULFM, the process grid data structure (and any local variables and arrays depending on them) needs only to be changed if $f > 0$.

We also have implemented a testing mode running under normal MPI where we simulate failures and we shift MPI ranks across process grids in the same way as illustrated in Figure 3. In the example of Figure 3, with a total of 2 failures, the last 2 MPI ranks (47 and 48) no longer participate in the computation and we have to change process grids and any dependent local variables and data structure for any process of lower rank. In this case, the shrinkage of the process grids also must occur between the *gather* and *scatter* stages of the parallel SGCT algorithm.

D. Modifications to the PDE Solver

To make an application fault-tolerant under the regime of process replacement, the replaced processes need to be started from an arbitrary point (e.g. checkpoint). This in turn requires two different paths for initialization. To accommodate this, the initialization of all process grid dependent variables and arrays are put into a single function. For the regime of process shrinkage, this is also required.

Before calling the parallel SGCT, the program must check for failures, and then determine a list of the ranks of all failed processes (see [10] for details on how to determine this under ULFM MPI). The program must iterate though its array of process grid data structures P_i, for each $i \in I$, and shrink them, as described in Section III-C. The combined grid process grid P^c must similarly be shrunk.

If the current process is part of a process grid P_i which had a failure, the solution field u_i will need to be re-sized. In this case, the respective sub-grid for the field will be assigned a zero combination coefficient in the SGCT and this process plays no part in the *gather* stage. It does however

participate in the *scatter* stage, and the process receives its re-sized local solution field automatically. In this way, the SGCT effectively forms a re-distribution at zero cost.

IV. Experimental Results

We use a parallel solver for the scalar advection equation in two spatial dimensions. The problem is solved on regular grids using a McCormack scheme [14]. An exact analytical solution can be computed to determine the accuracy of the solver.

For the SGCT version of the solver, the computation of solutions on different sub-grids is embarrassingly parallel and each sub-grid is assigned to a different process grid. The number of unknowns (grid size) on the lower diagonal sub-grids (see Figure 2) is half that of the other. As we use a fixed simulation timestep (Δt) across all grids for stability purposes, our load balancing strategy is to use half of the number of processes on these grids than on the others.

All experiments were conducted on the Raijin cluster managed by the National Computational Infrastructure (NCI) with the support of the Australian Government. Raijin has a total of 57,472 cores distributed across 3,592 compute nodes each consisting of dual 8-core Intel Xeon (Sandy Bridge 2.6 GHz) processors (i.e., 16 cores) with Infiniband FDR interconnect, a total of 160 terabytes (approximately) of main memory, with an x86_64 GNU/Linux OS. It has $\approx$ 10 petabytes of usable LUSTRE filesystem, which has been shown to have very low write latency [10].

We used git revision icldistcomp-ulfm-46b781a8f170 of ULFM MPI under the development branch 1.7ft of Open MPI for our experiments. The parameters for the collective communications for `mpirun` were set to `coll tuned,ftbasic,basic,self`. We set the MCA parameter `coll_ftbasic_method=1` of ULFM MPI to choose the 'Two-Phase Commit' as an agreement algorithm for failure recovery. The 'Log Two-Phase Commit' option was more scalable than the former, but could not be used in our experiments due to its instability. All the source code (including ULFM MPI) was compiled with GNU-4.6.4 compilers using the flag `-O3`. Process failure was implemented by sending a `kill` signal to the required number of (randomly selected) processes.

Figure 4 compares the measured recovery overheads of the process replacement vs the process shrinkage regimes for fault recovery. We see an order of magnitude improvement for shrinkage. The plots scale roughly linearly with the number of cores, indicating the degree of scalability of the current ULFM MPI distributed agreement algorithm. We found that varying the number of failures had little effect on the recovery time.

The overall scalability of the two regimes, plus a checkpoint-restart version of our parallel advection SGCT solver (CR-SGCT) is shown in Figure 5. Details of the disk-based checkpoint-restart method are given in [10]: the

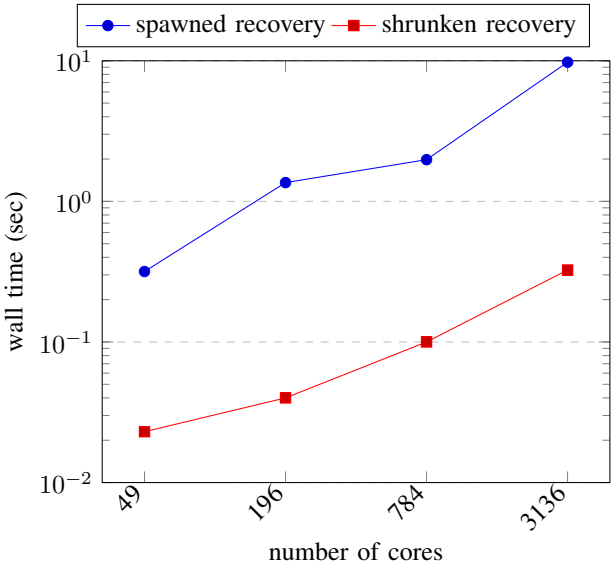


Figure 4. Comparison of communicator recovery overheads of the FT-SGCT based 2D general advection solver with a single combination, level $l = 4$, and full grid size $(2^{13}+1) \times (2^{13}+1)$. The overhead for 'spawned recovery' includes detection and identification of process failures, shrinking the broken communication, and spawning the replacement processes. But for 'shrunken recovery', no spawning of replacement processes is needed. Two randomly selected MPI processes are killed to simulate the process failure.

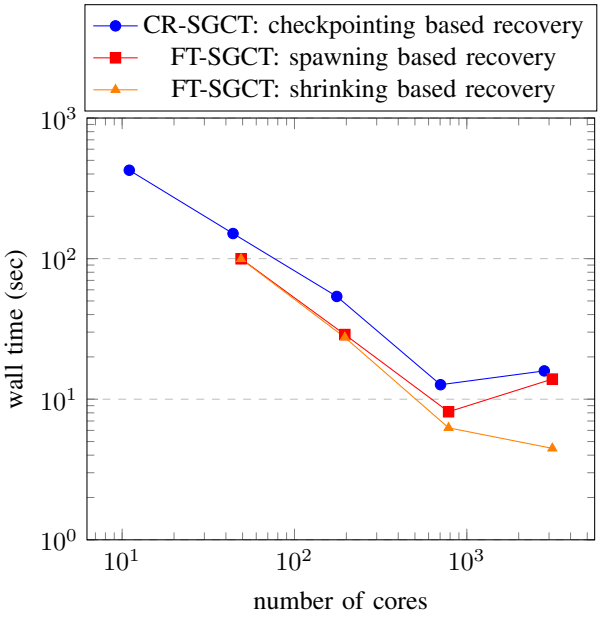


Figure 5. Overall parallel performance of the CR-SGCT and FT-SGCT based 2D general advection solver with a single combination, level $l = 4$, and full grid size $(2^{13}+1) \times (2^{13}+1)$. The execution time in each plot includes faulty communicator reconstruction time by spawning the replacement MPI processes or shrinking the communicator, data recovery time, and application running time. The results shown are an average of five experiments, with each time two randomly selected processes failing.

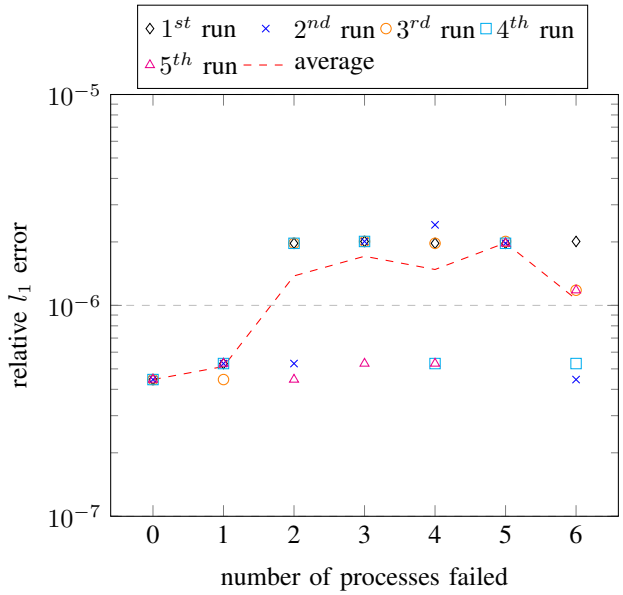


Figure 6. Approximation error of the FT-SGCT based 2D general advection solver with a single combination, level $l = 4$, and full grid size $(2^{13} + 1) \times (2^{13} + 1)$. Randomly selected MPI processes are killed to simulate process failure, out of an initial set of 49. After the failure, the rest of the computations and communications are performed with either the non-shrunken or shrunken communicator. The baseline error rate (no failures) is 4.45E-07.

actual runtime is augmented with the expected number of checkpoints writes with an optimal checkpointing interval. Two randomly selected process failures were used: this is sufficient to reveal interesting recovery behavior. Despite the loss of compute resources, the shrinkage regime is the fastest, and especially for $\approx$ 3000 cores, where the ULFM agreement algorithm impacts on application performance (c.f. Figure 4). It is also faster than the checkpoint-restart regime, by a factor of approximately 2.

Upon process failure in the SGCT, the respective solution field component grids, G_i, are discarded. This results in a loss of accuracy. Figure 6 indicates the 1-norm of the error under the SGCT as a function of the number of processes failed. Note that both the replacement and shrinkage regimes perform identically in this respect. The impact of the number of failures depends on which G_i are affected, and how many of them. The figure indicates instances of negligible impact (lower points) and a noticeable (but arguably tolerable) impact for more than 2 faults (by a factor of ≈ 5).

V. Related Work

A technique for replacing only a single failed process on the communicator and matrix data repair for a QR-Factorization problem is proposed in [15]. Process failure is handled by the ULFM standard, and data repair is accomplished by using a reduction operation on a checksum and remaining data values. The technique was not applied to a varying number of processes on other realistic parallel applications.

Algorithm-based fault tolerance techniques for creating robust PDE solvers based on the modified sparse grid combination technique are proposed in [16], [9]. These works however were implemented using simulated, rather than genuine, process failures. They also assumed the process replacement regime for fault recovery.

Fault-tolerant SGCT-based applications have been developed for the case of an advection solver [10], GENE [8], [13] and Taxilla LBM and SFI [13]. These also assumed the process replacement regime for fault recovery.

A fault-tolerant implementation of a multi-level Monte Carlo simulation that avoids checkpointing or recomputation of samples was proposed in [17]. It used the ULFM standard to recover the communicator by sacrificing its original size and employing a periodic reduction strategy with all the unaffected samples to generate final result. However, this work did not have to deal with resizing of local fields as does our work.

VI. Conclusions

We have demonstrated, via a parallel advection solver, that applications can be made fault tolerant via the SGCT under a regime of shrinking compute resources. The fault recovery under ULFM MPI is relatively simple and reliable. Process grid data structure shrinkage was the main addition needed in the parallel SGCT algorithm, and only modest modifications on an existing fault tolerant application was required. The SGCT simply and elegantly avoids the problem of redistribution, which is normally an issue when compute resources are shrunk.

Unsurprisingly, the recovery overhead for the shrinking regime was an order of magnitude less than for the established replacement regime, due to the fact that spawning new processes takes considerable overhead. Surprisingly, with a small number of failures, the shrinking regime led to better application performance, especially for high core counts. In all cases, it clearly out-performed checkpoint-restart based recovery. Depending on which component grids were excluded due to process failure, the loss of accuracy in the advection solver was either negligible, or within a factor of 5, even when 10% of the processes failed.

Note that the current (stable) ULFM MPI distributed agreement algorithm is still inefficient due to the recentness of this technology, and we expect the performance of our method relative to checkpointing will improve.

Future work includes extending our work to allow also for the growing of compute resources and thus achieve elasticity. It also includes extending our work to real applications, such as GENE, Taxilla LBM and SFI [13]. While there is no in-principle reason why this cannot be done, all these applications are very large ($\approx$ 10–100 KLOC), and any local or global variable dependent on changes in the (equivalent of

the) process grid data structure must be located and changed upon shrinkage or expansion.

VII. Acknowledgments

This research was supported under the Australian Research Council's Linkage Projects funding scheme (project number LP110200410). We thank the NCI National Facility for the use of the Raijin cluster. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation. We acknowledge support by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing under Award Number 13-016717.

References

[1] Message Passing Interface Forum, "MPI: A message passing interface," in *Proceedings of Supercomputing*. IEEE Computer Society Press, November 1993, pp. 878–883.

[2] Fault Tolerance Working Group, "Run-through stabilization interfaces and semantics," svn.mpi-forum.org/trac/mpi-forum-web/wiki/ft/run_through_stabilization.

[3] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra, "An evaluation of user-level failure mitigation support in MPI," in *Recent Advances in the Message Passing Interface*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7490, pp. 193–203.

[4] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta Numerica*, vol. 13, pp. 147–269, 2004.

[5] M. Griebel, "The combination technique for the sparse grid solution of PDE's on multiprocessor machines," in *Proceedings of Parallel Processing Letters*, 1992, pp. 61–70.

[6] M. Griebel, M. Schneider, and C. Zenger, "A combination technique for the solution of sparse grid problems," in *Proceedings of Iterative Methods in Linear Algebra*, P. de Groen and R. Beauwens, Eds. IMACS, Elsevier, North Holland, 1992, pp. 263–281.

[7] C. Kowitz, D. Pflüger, F. Jenko, and M. Hegland, "The combination technique for the initial value problem in linear gyrokinetics," in *Proceedings of Sparse Grids and Applications*, ser. Lecture Notes in Computational Science and Engineering, vol. 88. Heidelberg: Springer, October 2012, pp. 205–222.

[8] M. M. Ali, P. E. Strazdins, B. Harding, M. Hegland, and J. W. Larson, "A fault-tolerant gyrokinetic plasma application using the sparse grid combination technique," in *Proceedings of the 2015 International Conference on High Performance Computing & Simulation (HPCS 2015)*, Amsterdam, The Netherlands, July 2015, pp. 499–507.

[9] B. Harding and M. Hegland, "A parallel fault tolerant combination technique," in *Proceedings of the International Conference on Parallel Computing, (ParCo 2013)*, Garching, Germany, 2013, pp. 584–592.

[10] M. M. Ali, J. Southern, P. E. Strazdins, and B. Harding, "Application level fault recovery: Using fault-tolerant Open MPI in a PDE solver," in *Proceedings of the IEEE 28th International Parallel & Distributed Processing Symposium Workshops (IPDPSW 2014)*, Phoenix, USA, May 2014, pp. 1169–1178.

[11] P. E. Strazdins, M. M. Ali, and B. Harding, "Highly scalable algoritihms for the sparse grid combination technique," in *Proceedings of the IEEE 29th International Parallel & Distributed Processing Symposium Workshops (IPDPSW 2015)*, Hyderabad, India, May 2015, pp. 941–950.

[12] P. E. Strazdins, M. M. Ali, and B. Harding, "Design and analysis of two highly scalable sparse grid combination algorithms," *Journal of Computational Science (JCS 2016)*, 17 pages, http://hdl.handle.net/1885/95531, (Under Review).

[13] M. M. Ali, P. E. Strazdins, B. Harding, and M. Hegland, "Complex scientific applications made fault-tolerant with the sparse grid combination technique," *International Journal of High Performance Computing Applications (IJHPCA 2016)*, 2016, http://hpc.sagepub.com/content/early/2016/02/10/1094342015628056 (Published Online).

[14] P. Lax and B. Wendroff, "Systems of conservation laws," *Communications on Pure and Applied Mathematics*, vol. 13, no. 2, pp. 217–237, 1960.

[15] W. B. Bland, "Toward message passing failure management," Ph.D. dissertation, University of Tennessee, 2013.

[16] J. W. Larson, M. Hegland, B. Harding, S. G. Roberts, L. Stals, A. P. Rendell, P. E. Strazdins, M. M. Ali, C. Kowitz, R. Nobes, J. Southern, N. Wilson, M. Li, and Y. Oishi, "Fault-tolerant grid-based solvers: Combining concepts from sparse grids and mapreduce," *Procedia Computer Science*, vol. 18, no. 0, pp. 130–139, 2013, 2013 International Conference on Computational Science (ICCS 2013).

[17] S. Pauli, M. Kohler, and P. Arbenz, "A fault tolerant implementation of multi-level Monte Carlo methods," in *Proceedings of the International Conference on Parallel Computing, (ParCo 2013)*, Garching, Germany, 2013, pp. 471–480.