SIAM J. SCI. COMPUT.
Vol. 37, No. 3, pp. C331–C353

FAULT TOLERANT COMPUTATION WITH THE SPARSE GRID COMBINATION TECHNIQUE*

BRENDAN HARDING†, MARKUS HEGLAND†, JAY LARSON†, AND JAMES SOUTHERN‡

Abstract. This paper continues to develop a fault tolerant extension of the sparse grid combination technique recently proposed in [B. Harding and M. Hegland, *ANZIAM J. Electron. Suppl.*, 54 (2013), pp. C394–C411]. This approach to fault tolerance is novel for two reasons: First, the combination technique adds an additional level of parallelism, and second, it provides algorithm-based fault tolerance so that solutions can still be recovered if failures occur during computation. Previous work indicates how the combination technique may be adapted for a low number of faults. In this paper we develop a generalization of the combination technique for which arbitrary collections of coarse approximations may be combined to obtain an accurate approximation. A general fault tolerant combination technique for large numbers of faults is a natural consequence of this work. Using a renewal model for the time between faults on each node of a high performance computer, we also provide bounds on the expected error for interpolation with this algorithm in the presence of faults. Numerical experiments solving the scalar advection PDE demonstrate that the algorithm is resilient to faults on a real application. It is observed that the time to solution is not significantly affected by the presence of (simulated) faults. Additionally the expected error increases with the number of faults but is relatively small even for high fault rates. A comparison with traditional checkpoint-restart methods applied to the combination technique shows that our approach is highly scalable with respect to the number of faults.

Key words. exascale computing, algorithm-based fault tolerance, sparse grid combination technique, parallel algorithms

AMS subject classifications. 65Y05, 68W10

DOI. 10.1137/140964448

1. Introduction.

Many recent survey articles on the challenges of achieving exascale computing identify three issues to be overcome: exploiting massive parallelism, reducing energy usage, and, in particular, coping with run-time failures [2, 6, 3]. Faults are an issue at petascale/exascale due to the increasing number of components in such systems. Traditional checkpoint-restart–based solutions become infeasible at this scale as the decreasing mean time between failures approaches the time required to checkpoint and restart an application. Algorithm-based fault tolerance has been studied as a promising solution to this issue for many problems [14, 1].

Sparse grids were introduced in the study of high-dimensional problems to reduce the *curse of dimensionality*. They are based on the observation that when a solution on a regular grid is decomposed into its hierarchical bases, the highest frequency components have the most unknowns but contribute the least to sufficiently smooth solutions. Thus removing some of these high frequency components has a

*Submitted to the journal's Software and High-Performance Computing section April 10, 2014; accepted for publication (in revised form) March 3, 2015; published electronically May 12, 2015. This research was supported by the Australian Research Council's *Linkage Projects* funding scheme (project LP110200410). We are grateful to Fujitsu Laboratories of Europe for providing funding as the collaborative partner in this project.

http://www.siam.org/journals/sisc/37-3/96444.html

†Mathematical Sciences Institute, The Australian National University, Acton, Australian Capital Territory, 2601, Australia (brendan.harding@anu.edu.au, markus.hegland@anu.edu.au, jay.larson@anu.edu.au).

‡Fujitsu Laboratories of Europe, Hayes Park Central, Hayes, Middlesex, UB4 8FE, UK (james.southern@gmail.com).

small impact on the accuracy of the solution but significantly reduces the computational complexity [8, 7]. The combination technique was introduced to approximate sparse grid solutions without the complications of computing with a hierarchical basis. In recent years these approaches have been applied to a wide variety of applications from real-time visualization of complex datasets to solving high-dimensional problems that were previously cumbersome [18, 8].

Previous works [9, 17, 10] have described how the combination technique can be implemented within a MapReduce [4] framework. MapReduce is a functional programming pattern by which an application is factored into successive "Map" and "Reduce" stages; in the combination technique these correspond to the computation of component grid solutions and their subsequent combination to arrive at a solution, respectively. This approach endows an extra layer of parallelism at relatively low implementation cost. Furthermore, fault tolerance is a natural outgrowth of MapReduce and can be achieved by recomputing failed Map tasks. Also proposed was an alternative fault tolerant approach in which recomputation of failed tasks can be avoided by computing a solution that excludes the failed component solutions with a small trade-off in solution error. In [11] we demonstrated this approach for a simple two-dimensional problem showing that the average solution error after simulated faults was generally close to that without faults.

Here, we extend our previous work in the following ways: We generalize the combination technique beyond the classic combination; we apply this generalized formulation to define a procedure for computing alternative solutions in response to process failures; we provide expected error bounds for sparse grid interpolation under scenarios in which the classic combination suffers faults in one or more of its component grids; and we apply this approach to a three-dimensional benchmark problem. This generalized combination technique formulation is applicable to an arbitrary collection of grids that belong to a lattice of nested grids. Motivated by inclusion/exclusion principles and error bounds for sparse grid interpolation, we have formulated the computation of combination coefficients as a constrained binary integer programming problem (BIP), resulting in a general fault tolerant combination technique. This approach forms the core of a general framework for computations using the fault tolerant combination technique (FTCT) and operates as follows: Consider an application that at the outset aims to compute component solutions on a collection of grids for which the combination formula is known. If one or more of these component solutions are lost due to faults, a new set of coefficients is calculated for the combination by solving the corresponding BIP. This new combination formula is applied to compute an alternative approximation to the solution. We evaluate the accuracy, scalability with respect to number of faults, and parallel performance of this strategy for the three-dimensional scalar advection equation.

The remainder of the paper is organized as follows. In section 2 we review the combination technique and some well-known results which are relevant to our analysis of the FTCT. We then develop the notion of a general combination technique and show that the derivation of combination coefficients can be framed as a BIP.

In section 3 we describe how the general combination technique gives rise to a general fault tolerant combination technique able to handle large numbers of faults. As the corresponding BIP may be cumbersome to solve, we discuss some specialized scenarios for which combination coefficients may be found quickly. Using a simple model for faults on each node of a supercomputer, we are able to model the failure of component grids in the combination technique and apply this to the simulation of faults in our code. We derive bounds on the expected error of sparse grid interpolation

and briefly demonstrate how faults affect the scalability of the algorithm as a whole for the specialized cases discussed.

In section 4 we describe the details of our implementation. In particular we discuss the two layers of parallelism and how these interact throughout the computation.

Finally, in section 5 we present numerical results obtained by running our implementation with simulated faults on a PDE solver. It is demonstrated that the implementation is algorithmically scalable; that is, the time to solution is not significantly affected by large numbers of faults and has a relatively small impact on the solution error.

2. The combination technique and a generalization. We introduce the combination technique and a classical result which will be used in our analysis of the FTCT. For a complete introduction of the combination technique one should refer to [5, 8, 7]. We then extend this to a more general notion of a combination technique building on existing work on adaptive sparse grids [12].

2.1. The combination technique. Let $i \in \mathbb{N}$; then we define $\Omega_i := \{k2^{-i} : k = 0, \ldots, 2^i\}$ to be a discretization of the unit interval. Similarly, for $i \in \mathbb{N}^d$ we define $\Omega_i := \Omega_{i_1} \times \cdots \times \Omega_{i_d}$ as a grid on the unit d-cube. Throughout the rest of this paper we treat the variables i, j as multi-indices in $\mathbb{N}^d$. We say $i \leq j$ iff $i_k \leq j_k$ for all $k \in \{1, \ldots, d\}$, and similarly, $i < j$ iff $i \leq j$ and $i \neq j$.

Now suppose we have a problem with solution $u \in V \subset C([0,1]^d)$; then we use $V_i \subset V$ to denote the function space consisting of piecewise linear functions uniquely determined by their values on the grid Ω_i. Further, we denote an approximation of u in the space V_i by u_i. The sparse grid space of level n is defined to be $V_n^s := \sum_{\|i\|_1 \leq n} V_i$. A sparse grid solution is a $u_n^s \in V_n^s$ which closely approximates $u \in V$. The combination technique approximates sparse grid solutions by taking the sum of several solutions from many different anisotropic grids. The classical combination technique with level n is given by the equation

$$u_n^c := \sum_{k=0}^{d-1} (-1)^k \binom{d-1}{k} \sum_{\|i\|_1 = n-k} u_i \,. \tag{2.1}$$

Fundamentally, this is an application of the inclusion/exclusion principle. This can be seen if the function spaces are viewed as a lattice [12]. For example, if one wishes to add the functions $u_i \in V_i$ and $u_j \in V_j$, then the result will have two contributions from the intersection space $V_{i \wedge j} = V_i \cap V_j$, with $i \wedge j = (\min\{i_1, j_1\}, \ldots, \min\{i_d, j_d\})$. To avoid this we simply take $u_i + u_j - u_{i \wedge j}$. This can be seen in Figure 1, which shows a level 4 combination in two dimensions.

An important concept in the development of sparse grids is that of the hierarchical surplus. A simple definition of the hierarchical surplus W_i is the space of all functions $f_i \in V_i$ such that f_i is zero when sampled on all grid points in the set $\bigcup_{j<i} \Omega_j$. Equivalently, we have $V_i = W_i \oplus \sum_{j<i} V_j$. Noting that $V_i = \bigoplus_{j \leq i} W_j$, a hierarchical decomposition of $u_i \in V_i$ is the computation of the unique components $h_j \in W_j$ for $j \leq i$ such that $u_i = \sum_{j \leq i} h_j$. The sparse grid space can also be written in terms of hierarchical surpluses as $V_n^s = \bigoplus_{\|i\|_1 \leq n} W_i$.

Let $H^2_{0,\text{mix}}$ be the Sobolev space having zero boundaries and dominating mixed derivatives with norm

$$\|u\|^2_{H^2_{0,\text{mix}}} = \sum_{\|i\|_\infty \leq 2} \left\| \frac{\partial^{\|i\|_1}}{\partial x^i} u \right\|_2^2 .$$

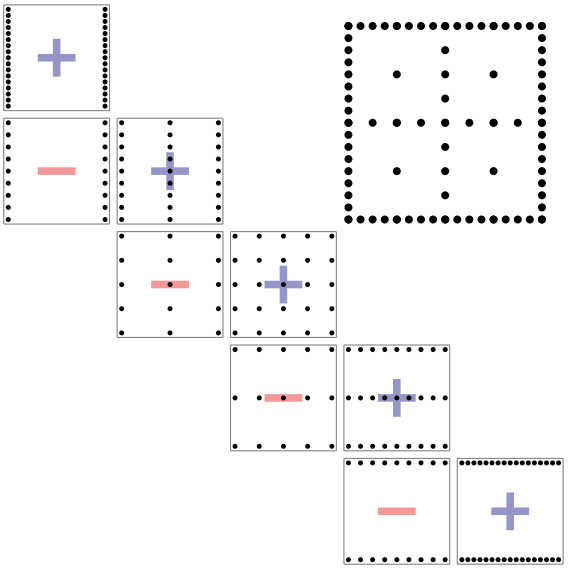

FIG. 1. *A level 4 combination in two dimensions. The nine component grids are arranged according to the frequency of data points in each dimension. A plus or minus denotes a combination coefficient of +1 or −1, respectively. In the top right is the (enlarged) sparse grid corresponding to the union of the component grids.*

If $u \in H^2_{0,\text{mix}}$, then we have the estimate $\|h_j\|_2 \leq 3^{-d} 2^{-2\|i\|_1} |u|_{H^2_{0,\text{mix}}}$ for each of the hierarchical spaces where $|u|_{H^2_{0,\text{mix}}} := \|\frac{\partial^{2d}}{\partial x_1^2 \cdots \partial x_d^2} u\|_2$ is a seminorm [5]. In the classical theory of the combination technique this estimate is used to prove the error bound

$$\|u - u_n^c\|_2 \leq \sum_{\|j\|_1 > n} \|h_j\|_2 \leq 3^{-d} |u|_{H^2_{0,\text{mix}}} \sum_{k=n+1}^{\infty} 2^{-2k} \binom{k+d-1}{d-1} \tag{2.2}$$

$$= \frac{1}{3} \cdot 3^{-d} 2^{-2n} |u|_{H^2_{0,\text{mix}}} \sum_{k=0}^{d-1} \binom{n+d}{k} \left(\frac{1}{3}\right)^{d-1-k} \tag{2.3}$$

$$= \frac{1}{3} \cdot 3^{-d} 2^{-2n} |u|_{H^2_{0,\text{mix}}} \left(\frac{n^{d-1}}{(d-1)!} + \mathcal{O}(n^{d-2}) \right)$$

for sparse grid interpolation. Similar bounds can be shown for the ∞ and energy norms [7].

In practice, strongly anisotropic grids, i.e., those with $\|i\|_\infty \approx \|i\|_1$, can be problematic. Not only are they difficult to compute in some circumstances, but one also finds that they give poor approximations that do not cancel out in the combination as expected. Therefore it is often beneficial to implement a *truncated* combination. In this paper we define a truncated combination as

$$u^c_{n,\tau} := \sum_{k=0}^{d-1} (-1)^k \binom{d-1}{k} \sum_{\substack{\|i\|_1 = n-k \\ \min(i) \geq \tau}} u_i \,, \tag{2.4}$$

where τ is referred to as the truncation parameter. Such combinations will be used for the numerical results presented in section 5.

2.2. A general combination technique. The combination technique can be generalized arbitrary sums of solutions computed on different grids. Given a (finite) set of multi-indices $I \subset \mathbb{N}^d$, one can write

$$u_I^c = \sum_{i \in I} c_i u_i \,, \tag{2.5}$$

where the c_i are referred to as the combination coefficients. It is straightforward to show that $u_I^c \in V_I^s := \sum_{i \in I} V_i = \bigoplus_{i \in I\downarrow} W_i$, where $I\downarrow = \{i \in \mathbb{N}^d : \exists j \in I \text{ with } i \leq j\}$ is the smallest downset containing I. Not every choice of the c_i will produce a reasonable approximation to u. The question now is which combination coefficients produce the best approximation to u? One could attempt to solve the optimization problem of minimizing, for example, $\|u - \sum_{i \in I} c_i u_i\|_2$ which is known as *opticom* [13]. However, this may be cumbersome to solve in a massively parallel implementation and also requires an approximation of the residual. In this paper we consider combinations which we know a priori will give a good approximation in some sense. We define a *sensible* combination to be one in which each hierarchical space contributes either once to the solution or not at all. In particular we would like the coefficients to follow an inclusion/exclusion similar to that of the classical combination technique as described in section 2.1. We observe that $W_i \subset V_j$ for all $j \geq i$; therefore one can easily determine how many times a given W_i contributes to the solution by summing all of the coefficients c_j for which $W_i \subset V_j$. Further, it is also desirable that the i for which W_i contributes to the solution forms a downset; see [12]. In light of this we have the following definition.

DEFINITION 2.1. *A set of combination coefficients $\{c_i\}_{i \in I}$ is said to be* valid *if for each $i \in I\downarrow$ it satisfies the property*

$$\sum_{j \in I,\, j \geq i} c_j \in \{0, 1\} \,.$$

Further, if $\sum_{j \in I,\, j \geq i} c_j = 1$ for some $i \in I\downarrow$, then $\sum_{j \in I,\, j \geq i'} c_j = 1$ for all $i' \leq i$. We refer to a combination (of solutions) as valid if the corresponding combination coefficients are valid.

The motivation for this definition is that these properties are satisfied for dimension adaptive sparse grids [12]. Let $P_i : V \mapsto V_i$ be a lattice of projection operators associated with the tensor product space V which satisfy $P_i P_j = P_{i \wedge j}$, $P_i P_j = P_j P_i$, and $P_i P_i = P_i$. Given a downset I and $P_I : V \mapsto V_I^s$, it follows from [12, p. C344] that for $i \in I$

$$P_i P_I = P_i \left(1 - \prod_{j \in I} (1 - P_j)\right) = P_i - P_i(1 - P_i) \prod_{j \in I \setminus \{i\}} (1 - P_j) = P_i \,.$$

Letting $P_I = \sum_{i \in I} c_i P_i$ it follows that $\sum_{\{j \in I \text{ s.t. } j \wedge i = i\}} c_i = \sum_{j \in I,\, j \geq i} c_i = 1$.

For a given I there are many sets of valid combination coefficients that one might take. To determine which of these a priori will give the best approximation of u, we use error estimates for sparse grid interpolation. It is reasonable to expect that such combinations will work well in a more general setting given the underlying inclusion/exclusion principle. The error estimate (2.2) for sparse grid interpolation is extended to the general combination technique by

$$\|u - u_I^c\|_2 \leq 3^{-d} |u|_{H^2_{0,\text{mix}}} \sum_{i \in \mathbb{N}^d} \left(4^{-\|i\|_1} \left| 1 - \sum_{j \in I,\, j \geq i} c_j \right| \right) .$$

Rather than searching for coefficients which minimize $\|u-u_I^c\|_2$ directly, we can search for coefficients which minimize the bound; for valid coefficients this is equivalent to maximizing

$$Q(\{c_i\}_{i\in I}) := \sum_{i\in I\downarrow} 4^{-\|i\|_1} \sum_{j\in I,\, j\geq i} c_j. \tag{2.6}$$

Therefore, the general problem of finding the best combination of solutions u_i for $i \in I$ can be formulated as an optimization problem, in particular the maximization of $Q(\{c_i\}_{i\in I})$ subject to the constraints $\sum_{j\in I,\, j\geq i} c_j \in \{0,1\}$ for each $i \in I$ and $\sum_{j\in I,\, j\geq i'} c_j \geq \sum_{j\in I,\, j\geq i} c_j$ for all $i' \leq i$. Since the c_i must be integers this would be a simple integer linear programming problem if not for the nontrivial constraints. (To see why the c_i must be integers, we note that the nonzero c_i for which $c_j = 0$ for all $j > i$ must be 1 in order for the set of coefficients to be valid. The remaining coefficients are now obtained from the application of the inclusion/exclusion principle which can only result in integer coefficients.) Fortunately we can simplify this by introducing the hierarchical coefficient.

DEFINITION 2.2. *Let I be a set of multi-indices; then for $i \in I\downarrow$ we define the* hierarchical coefficients

$$w_i := \sum_{j\in I,\, j\geq i} c_j. \tag{2.7}$$

Suppose we expand our list of coefficients to the set $\{c_i\}_{i\in I\downarrow}$ with the assumption $c_i = 0$ for $i \notin I$. Now let c, w be vectors for the sets $\{c_i\}_{i\in I\downarrow}$, $\{w_i\}_{i\in I\downarrow}$, respectively (both having the same ordering with respect to $i \in I\downarrow$). Using (2.7) we can write $w = Mc$, where M is an $|I\downarrow| \times |I\downarrow|$ matrix. Further, we note that if the elements of c, w are ordered according to ascending or descending values of $\|i\|_1$, then M is an upper or lower triangular matrix, respectively, with 1's on the diagonal. Therefore M is invertible, and we can write $c = M^{-1}w$. Additionally, the restriction that $c_i = 0$ for $i \notin I$ can be written as $(M^{-1}w)_i = 0$.

Since $w_i \in \{0,1\}$ for any set of valid coefficients we can formulate the *general coefficient problem* (GCP) as the BIP

$$\begin{aligned} \underset{w}{\text{argmin}} \quad & Q'(w) := -\sum_{i\in I\downarrow} 4^{-\|i\|_1} w_i \\ \text{subject to} \quad & (M^{-1}w)_i = 0 \text{ for } i \notin I, \\ & w_i \geq w_j \text{ for all } i < j. \end{aligned} \tag{2.8}$$

Many of the inequality constraints are redundant and as such can be reduced to $w_i \geq w_{i+e_k}$ for all i and $k = 1, \ldots, d$ with e_k being the multi-index with $(e_k)_l = \delta_{k,l}$. This is more manageable in practice and can be solved using a variety of algorithms that are typically based on branch-and-bound and/or cutting plane techniques. However, this formulation also reveals that the general coefficient problem is NP-complete [15]. If I is a downset, then we can solve this rather quickly, but in general there exist cases which take an incredibly long time to solve. Another problem one runs into is that there is often not a unique solution. In such circumstances we will simply pick any one of the optimal solutions as they cannot be further distinguished without additional information about u.

The only way to guarantee that a solution can be found quickly is to carefully choose the index set I. One particular class of index sets of interest are those which

are closed under the $\wedge$ operator, that is, if $i, j \in I$, then $i \wedge j \in I$. In the theory of partially ordered sets, $(I, \leq)$ with this property is referred to as a lower semilattice. For such I there is a unique solution to the GCP, namely $w_i = 1$ for all $i \in I\downarrow$, which clearly minimizes $Q'(w)$. The fact that the constraints are satisfied is readily checked. Computationally the coefficients can be found quickly by first finding $\max I := \{i \in I : \nexists j \in I \text{ s.t. } j > i\}$, setting $c_i = 1$ for $i \in \max I$, and then using the inclusion/exclusion principle to find the remaining coefficients in the order of descending $\|i\|_1$. This can also be viewed as an application of the lattice theory of projections on function spaces [12].

While the GCP as developed here is based upon estimates for $u \in H^2_{0,\text{mix}}$, we anticipate that the resulting combinations will still yield reasonable results for larger function spaces. This is based on the observation that the classical combination technique has been successfully applied to a wide variety of problems with $u \notin H^2_{0,\text{mix}}$.

3. Fault tolerant combination technique and fault simulation.

3.1. Fault tolerant combination technique (FTCT). In [9, 10] a fault tolerant combination technique was introduced. The most difficult aspect of generalizing this work is the updating of coefficients. While some theory and a few simple cases have been investigated, no general algorithm has been presented. Given the development of the general combination technique in section 2.2, we are now able to consider a more complete theory of the FTCT.

Suppose we have a set I of multi-indices for which we intend to compute each of the solutions u_i and combine as in (2.5). As each of the u_i can be computed independently, the computation of these is easily distributed across different nodes of a high performance computer. Suppose that one or more of these nodes experience a fault, hardware or software in nature. As a result, the computation of some of the u_i may not complete. We denote $J \subset I$ to be the set of indices for which the u_i did not complete. A lossless approach to fault tolerance would be to recompute u_i for $i \in J$. However, since recomputation is often costly, we propose a lossy approach to fault tolerance in which the failed solutions are not recomputed. In this approach, rather than solving the GCP for I, we instead solve it for $I\backslash J$. As $(I\backslash J)\downarrow \subseteq I\downarrow$, we expect this solution to have a larger error than the solution obtained if no faults had occurred. However, if $|J|$ is relatively small, we would also expect the loss of accuracy to be small because of the redundancy within the coarse approximations $\{u_i\}_{i\in I}$.

It is important to note at this point that our main consideration is faults which are fatal to a process/node and hence any computations on it. We suggest that our algorithm may also be used in the context of silent errors if a suitable error detection algorithm is implemented. Given the many coarse solutions, one might compare the result between each and reject coarse solutions which are significantly different from the others. The development and evaluation of such silent error detection schemes will not be explored in this paper.

As discussed in section 2.2, the GCP can be costly to solve in its most general form. While it can be solved rather quickly if the poset $(I, \leq)$ is a lower semilattice, this is no longer any help in the FTCT since the random nature of faults means we cannot guarantee that $(I\backslash J, \leq)$ is always a lower semilattice. The only way we can ensure this is to restrict which elements of I can be in J. A simple way to achieve this is to recompute missing u_i if $(I\backslash\{i\}, \leq)$ is not a lower semilattice. In particular, this is achieved if all u_i with $i \notin \max I$ are recomputed upon failure. Since elements in $\max I$ correspond to the solutions on the largest of the grids, we are avoiding the recomputation of the solutions which take the longest to compute. As these are

also the most prone to failure, one already has a significant impact on the expected time spent on recomputations. Additionally, this also means only the largest of the hierarchical spaces are ever omitted as a result of a failure. As these contribute the least to the solution, we expect the resulting error to be relatively close to that of the solution obtained if no faults had occurred. Finally, since $(I\backslash J, \leq)$ is still a lower semilattice, the resulting GCP for $I\backslash J$ has a unique maximal solution which is easily computed.

We now illustrate this approach as it is applied to the classical combination technique. We define $I_n = \{i \in \mathbb{N}^d : \|i\|_1 \leq n\}$. It was shown in [9] that the proportion of additional unknowns in computing the solutions u_i for all $i \in I_n$ compared to $n - d < \|i\|_1 \leq n$ is at most $\frac{1}{2^d-1}$. If no faults occur, then the combination is exactly the classical combination technique with $c_i = (-1)^{n-\|i\|_1}\binom{d-1}{n-\|i\|_1}$ if $n - d < \|i\| \leq n$ and $c_i = 0$ otherwise. If faults do occur, then we recompute any u_i with $\|i\|_1 < n$ that was not successfully computed. If no faults occurred for any u_i with $\|i\| = n$, then we can again proceed with the classical combination. If faults affect any u_i with $\|i\|_1 = n$, then we add such i to the set J and then solve the GCP for $I_n\backslash J$. The solution to (2.8) is trivially obtained with hierarchical coefficients $w_i = 1$ for all $i \in I_n\backslash J$.

The largest solutions (in terms of unknowns) which may have to be recomputed are those with $\|i\|_1 = n - 1$, which would be expected to take at most half the time of those solutions with $\|i\|_1 = n$. Since they take less time to compute they are also less likely to be lost due to failure. Additionally, there are $\binom{n-1+d-1}{d-1}$ solutions with $\|i\|_1 = n - 1$, which is less than the $\binom{n+d-1}{d-1}$ with $\|i\|_1 = n$. Combining these observations, we would expect to see far fewer disruptions caused by recomputations when using this approach compared to a lossless approach where all failed solutions are recomputed.

The worst-case scenario with this approach is that all u_i with $\|i\|_1 = n$ are not successfully computed due to faults. In this case the resulting combination is simply a classical combination of level $n - 1$. This requires only the solutions u_i with $n - d \leq \|i\| \leq n - 1$. Likewise, all solutions to the GCP in this approach result in zero coefficients for all c_i with $i < n - d$. We can therefore reduce the overhead of the FTCT by computing only the solutions u_i for $n - d \leq \|i\|_1 \leq n$. It is known that the proportion of additional unknowns compared to the classical combination technique in this case is at most $\frac{1}{2(2^d-1)}$ [9].

The solutions u_i with $\|i\|_1 = n - 1$ are only half the size of the largest u_i, and hence recomputation of these may still be somewhat disruptive and thus undesirable. We could therefore consider recomputing only solutions with $\|i\|_1 \leq n - 2$. By doing this the recomputations are even more manageable, having at most one-quarter the unknowns of the largest u_i. The worst case now is that all solutions with $i \geq n - 1$ fail, and one obtains a classical combination of level $n - 2$. Again it turns out one does not require the entire downset I_n; in this case one requires solutions u_i with $n - d - 1 \leq \|i\|_1 \leq n$. Using arguments similar to those in [9], it is easily shown that the overhead in this case is at most $\frac{3}{4(2^d-1)}$. The trade-off now is that the update of coefficients takes a little more work. We are back in the situation where we cannot guarantee that $(I_n\backslash J, \leq)$ is a lower semilattice.

To solve the GCP in this case we start with all w_i equal to 1. If failures affected any u_i with $\|i\|_1 = n$, we set the corresponding constraints $c_i = w_i = 0$. For failures occurring on u_i with $\|i\|_1 = n - 1$ we have the constraints $w_i - \sum_{k=1}^{d} w_{i+e_k} = 0$. We note that (since the w_i are binary variables) this can only be satisfied if at most one of

the w_{i+e_j} is equal to 1. Further, if $\sum_{k=1}^{d} w_{i+e_k} = 0$, we must also have $w_i = 0$. This gives us a total of $d+1$ feasible solutions to check for each such constraint. Given g failures on solutions with $\|i\|_1 = n-1$, we have at most $(d+1)^g$ feasible solutions to the GCP to check. We note that typically all these need not be checked to find a minimum. Where some of the failures on the second layer are sufficiently far apart on the lattice, it is possible to significantly reduce the number of cases to check as constraints can be optimized independently. Further, this can be kept manageable if solutions are combined frequently enough that the number of failures g that are likely occur between combinations is small.

We could continue and describe an algorithm for only recomputing the fourth layer and below; however, the coefficient updates here become much more complex such that one effectively solves the full GCP. Our experience indicates that the recomputation of the third layer and below is a good trade-off between the need to recompute and the complexity of updating the coefficients. Our numerical results in section 5 are obtained using this approach.

3.2. Probability of failure for computations. To analyze the expected outcome of the fault tolerant combination technique described in section 3.1, we need to know the probability of each u_i failing. In particular, the availability of u_i will be modelled as a simple Bernoulli process U_i which is 0 if u_i was computed successfully and is 1 otherwise. It is assumed that each u_i is computed on a single computational node. Therefore we are interested in the probability that a failure occurs on this node before the computation of u_i is complete, that is, $\Pr(U_i = 1)$. Suppose T is a random variable denoting the time to failure on a given node and the time required to compute u_i is given by t_i; then one has $\Pr(U_i = 1) = \Pr(T \leq t_i)$. One therefore needs to know something about the distribution of T.

Schroeder and Gibson analyzed the occurrence of faults on 22 high performance machines at Los Alamos National Laboratory from 1996–2005 [19]. They found that the distribution of time between failures for a typical node in the systems studied was best fit by the Weibull distribution with a shape parameter of 0.7. Based upon this study we will consider a model of faults on each node based upon the Weibull renewal process, which is a renewal process where interarrival times are Weibull distributed, with shape parameter $0 < \kappa \leq 1$.

There are several reasons for considering a renewal process for modelling faults. First, renewal theory is commonly used in availability analysis, and there are many extensions such as alternating renewal processes in which one can also consider repair times. Second, we expect a fault tolerant implementation of message passing interface (MPI) to enable the substitution of a failed node with another available node in which case computation can continue from some recovered state. This will be further discussed in section 3.3. We now derive the value of $\Pr(U_i = 1)$.

Let $\{X_k\}_{k=1}^{\infty}$ be random variables for the successive times between failures on a node. We assume that the X_k are positive and independent and identically distributed with cumulative distribution

$$F(t) := \Pr(X_k \leq t) = 1 - e^{-(t/\lambda)^{\kappa}} \tag{3.1}$$

for some $0 < \lambda < \infty$ and $0 < \kappa \leq 1$. Let $S_k = \sum_{m=1}^{k} X_m$ (for $k \geq 1$) be the waiting time to the kth failure. Let $N(t)$ count the number of failures that have occurred up to (and including) time t, that is, $N(t) = \max\{k : S_k \leq t\}$). By the elementary

renewal theorem one has

$$\lim_{t\to\infty}\frac{1}{t}\mathbb{E}[N(t)] = \frac{1}{\mathbb{E}[X_1]} = \frac{1}{\lambda\Gamma(1+\frac{1}{\kappa})}\,. \tag{3.2}$$

We thus get an expression for the (long term) average rate of faults.

Now, while we have a distribution for the time between failures, when a computation starts it is generally unknown how much time has elapsed since the last failure occurred. Hence our random variable T is what is referred to as the random incidence (or residual lifetime). Noting that one is more likely to intercept longer intervals of the renewal process than shorter ones and that the probability distribution of the starting time is uniform over the interval, it is straightforward to show [20] that T has density

$$g(t) = \frac{1-F(t)}{\mathbb{E}[X_1]} = \frac{e^{-(t/\lambda)^\kappa}}{\lambda\Gamma(1+\frac{1}{\kappa})}\,.$$

It follows that the cumulative probability distribution is given by

$$G(t_i) := \Pr(T \le t_i) = \frac{1}{\lambda\Gamma(1+\frac{1}{\kappa})}\int_0^{t_i} e^{-(x/\lambda)^\kappa}dx\,.$$

The resulting distribution has properties similar to those of the original Weibull distribution. In fact, when $\kappa = 1$ we note that X_k and T are identically distributed: i.e., they are exponential with mean λ. Further, for $0 < \kappa \le 1$ we have the following bound.

LEMMA 3.1. *For $0 < \kappa \le 1$ one has*

$$G(t) \le F(t). \tag{3.3}$$

Proof. We note that via a change of variables

$$\Gamma\left(1+\frac{1}{\kappa}\right) = \int_0^\infty y^{\frac{1}{\kappa}}e^{-y}dy = \int_0^\infty \frac{\kappa}{\lambda}\left(\frac{x}{\lambda}\right)^\kappa e^{-(x/\lambda)^\kappa}dx,$$

and therefore

$$\begin{aligned}
\Gamma\left(1+\frac{1}{\kappa}\right)e^{-(t/\lambda)^\kappa} &= \int_0^\infty \frac{\kappa}{\lambda}\left(\frac{x}{\lambda}\right)^\kappa e^{-(x/\lambda)^\kappa}e^{-(t/\lambda)^\kappa}dx \\
&= \int_0^\infty \frac{x}{\lambda}\frac{\kappa}{\lambda}\left(\frac{x}{\lambda}\right)^{\kappa-1} e^{-(x/\lambda)^\kappa}e^{-(t/\lambda)^\kappa}dx \\
&= \left[-\frac{x}{\lambda}e^{-(x/\lambda)^\kappa}e^{-(t/\lambda)^\kappa}\right]_0^\infty - \int_0^\infty -\frac{1}{\lambda}e^{-(x/\lambda)^\kappa-(t/\lambda)^\kappa}dx \\
&= \int_0^\infty \frac{1}{\lambda}e^{-(x/\lambda)^\kappa-(t/\lambda)^\kappa}dx.
\end{aligned}$$

Since $0 < \kappa \leq 1$ and $t, x \geq 0$, one has $x^\kappa + t^\kappa \geq (x+t)^\kappa$ and hence

$$\begin{aligned}\Gamma\left(1+\frac{1}{\kappa}\right)e^{-(t/\lambda)^\kappa} &\leq \int_0^\infty \frac{1}{\lambda}e^{-((x+t)/\lambda)^\kappa}dx \\ &= \int_t^\infty \frac{1}{\lambda}e^{-(x/\lambda)^\kappa}dx \\ &= \int_0^\infty \frac{1}{\lambda}e^{-(x/\lambda)^\kappa}dx - \int_0^t \frac{1}{\lambda}e^{-(x/\lambda)^\kappa}dx \\ &= \Gamma\left(1+\frac{1}{\kappa}\right) - \frac{1}{\lambda}\int_0^t e^{-(x/\lambda)^\kappa}dx.\end{aligned}$$

Rearranging gives

$$\frac{1}{\lambda\Gamma(1+\frac{1}{\kappa})}\int_0^t e^{-(x/\lambda)^\kappa}dx \leq 1 - e^{-(t/\lambda)^\kappa},$$

which is the desired inequality. ∎

As a result of Lemma 3.1 one has that the probability of u_i failing to compute successfully is bounded above by

$$\Pr(U_i = 1) = G(t_i) \leq F(t_i).$$

Remark 1. The result of Lemma 3.1 is essentially a consequence of the property

$$\Pr(X \leq s + t \mid X > s) \leq \Pr(X \leq t), \tag{3.4}$$

where X is Weibull distributed with shape parameter $0 < \kappa \leq 1$ and $s, t \geq 0$. This property can be extended to the fact that if $s_2 \geq s_1 \geq 0$, then

$$\Pr(X \leq s_2 + t \mid X \geq s_2) \leq \Pr(X \leq s_1 + t \mid X \geq s_1).$$

This has important implications for the order in which we compute successive solutions on a single node. Solutions one is least concerned about not completing due to a fault should be computed first, and solutions for which we would like to minimize the chance of failure should be computed later.

Remark 2. We note that for $\kappa \geq 1$ the inequalities of (3.3) and (3.4) are reversed; thus $G(t) \geq F(t)$ and $\Pr(X \leq s + t \mid X > s) \geq \Pr(X \leq t)$ for $s, t \geq 0$.

Remark 3. We have considered here the computation of each u_i restricted to a node. If the u_i are distributed across many nodes, say M, then u_i will fail if a fault occurs on any one of the M nodes during the computation. It follows that $\Pr(U_i = 1) = 1 - (1 - G(t_i))^M$. As a rough approximation one has $\Pr(U_i = 1) \approx MG(t_i)$ for small $G(t_i)$. Another rough estimate is to simply divide the mean time between failures (MTBF) by M, that is, to use $G(t_i)$ with $\lambda = \text{MTBF}/(M\Gamma(1+\frac{1}{\kappa}))$. This is exact if $\kappa = 1$.

3.3. Fault simulation in the FTCT algorithm. We first describe the parallel FTCT algorithm:

1. Given a (finite) set of multi-indices I, distribute the computation of component solutions u_i for $i \in I$ among the available nodes such that each node has a comparable amount of work.
2. Each node begins to compute the u_i which have been assigned to it. For time evolving PDEs, the solvers are evolved for some fixed simulation time t_s.

3. On each node, once a u_i is computed a (local) checkpoint of the result is saved. If the node experiences a fault during the computation of a u_i, a fault tolerant implementation of MPI is used to replace this node with another (or continue once the interrupted node is rebooted). On the new node, checkpoints of already computed u_i are loaded, and we then assess whether the interrupted computation should be recomputed or discarded. If it is to be recomputed, this is done before computing any of the remaining u_i allocated to the node.
4. Once all nodes have completed their computations they communicate which u_i have been successfully computed via an MPI_ALLREDUCE. All nodes now have a list of multi-indices $I' \subseteq I$ and can solve the GCP to obtain combination coefficients.
5. All nodes compute a partial sum $c_i u_i$ for the u_i that they have computed, and then the sum is completed globally via MPI_ALLREDUCE calls on process subsets such that all nodes now have a copy of $u_{I'}^{\text{gcp}}$.
6. In the case of a time evolving PDE, the u_i can be sampled from $u_{I'}^{\text{gcp}}$ and the computation further evolved by repeating from step 2.

This approach differs from traditional checkpoint restart in that interrupted computations are typically not restarted. Further, rather than restart the entire application, one need only restart the failed node. The optional step 6 for time evolution problems has many advantages. First, by combining component solutions several times throughout the computation one can improve the approximation to the true solution. Second, each combination can act like a checkpoint such that when a u_i fails it can be easily restarted from the last combination (rather than from the very beginning). Third, there are potential opportunities to reassess the load balancing after each combination and potentially redistribute the u_i to improve performance in the next iteration.

For the numerical results in section 5 we do not currently use a fault tolerant implementation of MPI and instead simulate faults by modifying the following steps:

2. Before each node begins computing the u_i assigned to it, a process on the node computes a (simulated) time of failure by sampling a distribution for time to failure and adding it to the current time. In our results we sample the Weibull distribution for some mean $\lambda > 0$ and shape $0 < \kappa \leq 1$.
3. Immediately after a u_i has been computed on a node we check to see whether the current time has exceeded the (simulated) time of failure. If this is the case, the most recent computation is discarded. We then pretend that the faulty node has been instantly replaced and continue with step 3 as described.

Note from section 3.2 that sampling the Weibull distribution produces faults at least as often as the random incidence $G(t)$. Thus, by sampling the Weibull distribution in our simulation, the number of faults in our simulation is slightly worse than what might occur in reality.

The assumption in step 3 of replacing a failed node with another is based upon what one might expect from a fault tolerant MPI. In fact, both Heterogeneous Adaptive Reconfigurable Networked SyStem FT-MPI[1] and the relatively new User Level Fault Mitigation (ULFM) specification[2] allow this, although it certainly does not occur in an instant as is assumed in our simulation. Due to limited data at the current time we are unable to predict what recovery times one might expect. We also note that (simulated) failures are checked for at the completion of the computation of

[1] See http://icl.cs.utk.edu/ftmpi/.
[2] See http://fault-tolerance.org/.

each u_i. Since a failure is most likely to occur some time before the computation completes, time is wasted in the simulation from the sampled time of failure to the completion of the affected computation. Improving these aspects of the simulation and implementation with a fault tolerant MPI will be the subject of future work.

3.4. Expected error of the FTCT. In this section, we bound the expected interpolation error for the FTCT as applied to the classical combination technique as described in section 3.1. In particular we look at the case where all solutions with $\|i\|_1 < n$ are recomputed, and the case where all solutions with $\|i\|_1 < n-1$ are recomputed as described in section 3.1.

Given $u \in H^2_{\text{mix}}$, let

$$\epsilon_n := \frac{1}{3} \cdot 3^{-d} 2^{-2n} |u|_{H^2_{\text{mix}}} \sum_{k=0}^{d-1} \binom{n+d}{k} \left(\frac{1}{3}\right)^{d-1-k}$$

such that $\|u - u_n^c\|_2 \leq \epsilon_n$; see (2.3). Now, given a (finite) set of multi-indices I, we denote u_I^{gcp} to be a combination $\sum_{i \in I} c_i u_i$ which is a solution to the GCP described in section 2.2. When faults prevent successful computation of some of the u_i we must find $u_{I'}^{\text{gcp}}$ for some $I' \subset I$. Consider the Bernoulli process $\{U_i\}_{i \in I}$ for which each $U_i = 0$ if u_i is computed successfully and is 1 otherwise as described in section 3.2. Additionally it is assumed that the computation of each u_i is done within one node; that is, many u_i can be computed simultaneously on different nodes, but each individual u_i is computed within one hardware node. Let t_i be the time required to compute u_i for each $i \in I$. We assume that the time between failures on each node is Weibull distributed. As demonstrated in section 3.2, the probability of each u_i being lost as the result of a fault is given by the random incidence distribution

$$\Pr(U_i = 1) = G(t_i) \leq F(t_i).$$

Given that u_i with the same $\|i\|_1$ have a similar number of unknowns, we assume they will take roughly the same amount of time to compute. We therefore define $t_k := \max_{\|i\|_1 = k} t_i$, that is, the maximal time to compute any u_i with level k.

As a result, for each $i \in I$ the probability of each u_i not completing due to failures is bounded by

$$\Pr(U_i = 1) \leq G(t_{\|i\|_1}) \leq F(t_{\|i\|_1}).$$

With this we can now give the main result.

PROPOSITION 3.2. *Given $d, n > 0$ and $I_n := \{i \in \mathbb{N}^d : \|i\|_1 \leq n\}$, let u_i be the interpolant of $u \in H^2_{mix}$ for $i \in I_n$. Let each u_i be computed on a different node of a parallel computer for which the time between failures on every node is independent and identically Weibull distributed with mean $\lambda > 0$ and shape parameter $0 < \kappa \leq 1$. Let t_i be the (wall) time required to compute each u_i and let $t_n = \max_{\|i\|_1 = n} t_i$. Suppose we recompute any u_i with $\|i\|_1 < n$ which is interrupted by a fault; then let $\mathcal{I}$ be the set of all possible $I' \subseteq I_n$ for which u_i was successfully computed (eventually) iff $i \in I'$. Let $u_{\mathcal{I}}^{\text{gcp}}$ be the function-valued random variable corresponding to the result of the FTCT (i.e., $u_{I'}^{\text{gcp}}$ for some random $I' \in \mathcal{I}$); then the expected error is bounded above by*

$$\mathbb{E}\left[\|u - u_{\mathcal{I}}^{\text{gcp}}\|_2\right] \leq \epsilon_n \left(1 + 3\left(1 - e^{-(t_n/\lambda)^\kappa}\right)\right).$$

Proof. Since u_i with $\|i\|_1 < n$ are recomputed, we have that $U_i = 0$ for all $\|i\|_1 < n$, and therefore $\Pr(\mathcal{I} = I') = 0$ for $I_{n-1} \not\subseteq I'$. Note that I_k is a downset for

$k \geq 0$, that is, $I_k = I_k \downarrow$. Since the i with $\|i\|_1 = n$ are covering elements for I_{n-1}, it follows that all of the I' for which $\Pr(\mathcal{I} = I') > 0$ are also downsets. It follows that there is a unique solution to the GCP for such I', namely $w_i = 1$ for all $i \in I'$ and $w_i = 0$ otherwise. That is, $w_i = 0$ iff $U_i = 1$, and hence $w_i = 1 - U_i$. It follows that the error is bounded by

$$\|u - u^{\text{gcp}}_{I'}\|_2 \leq \|u - u^c_n\|_2 + \sum_{\|i\|_1 = n} |1 - w_i| \, \|h_i\|_2 = \|u - u^c_n\|_2 + \sum_{\|i\|_1 = n} U_i \|h_i\|_2.$$

From Lemma 3.1, the probability of a fault occurring during the computation of any u_i with $\|i\|_1 = n$ is bounded by $G(t_n)$, and therefore for $\|i\|_1 = n$ one has

$$\mathbb{E}[U_i] = 0 \cdot \Pr(U_i = 0) + 1 \cdot \Pr(U_i = 1) = \Pr(U_i = 1) = G(t_i) \leq G(t_n).$$

It follows that

$$\begin{aligned} \mathbb{E}\left[\|u - u^{\text{gcp}}_{\mathcal{I}}\|_2\right] &\leq \mathbb{E}\left[\|u - u^c_n\|_2 + \sum_{\|i\|_1 = n} U_i \|h_i\|_2\right] \\ &= \|u - u^c_n\|_2 + \sum_{\|i\|_1 = n} \mathbb{E}[U_i] \|h_i\|_2 \\ &\leq \|u - u^c_n\|_2 + \sum_{\|i\|_1 = n} G(t_i) \|h_i\|_2, \end{aligned} \tag{3.5}$$

and substituting the estimate $\|h_i\|_2 \leq 3^{-d} 2^{-2\|i\|_1} |u|_{H^2_{\text{mix}}}$ yields

$$\begin{aligned} \mathbb{E}\left[\|u - u^{\text{gcp}}_{\mathcal{I}}\|_2\right] &\leq \epsilon_n + \sum_{\|i\|_1 = n} G(t_n) 3^{-d} 2^{-2n} |u|_{H^2_{\text{mix}}} \\ &\leq \epsilon_n + F(t_n) \sum_{\|i\|_1 = n} 3^{-d} 2^{-2n} |u|_{H^2_{\text{mix}}} \\ &= \epsilon_n + \left(1 - e^{-(t_n/\lambda)^\kappa}\right) \binom{n+d-1}{d-1} 3^{-d} 2^{-2n} |u|_{H^2_{\text{mix}}}. \end{aligned}$$

Now, noting that $\binom{n+d-1}{d-1} \leq \sum_{k=0}^{d-1} \binom{n+d}{k} (1/3)^{d-1-k}$, one has

$$\binom{n+d-1}{d-1} 3^{-d} 2^{-2n} |u|_{H^2_{\text{mix}}} \leq 3\epsilon_n$$

and therefore,

$$\mathbb{E}\left[\|u - u^{\text{gcp}}_{\mathcal{I}}\|_2\right] \leq \epsilon_n \left(1 + 3\left(1 - e^{-(t_n/\lambda)^\kappa}\right)\right). \tag{3.6}$$

Note that as $t_n/\lambda \to \infty$, we have $\mathbb{E}\left[\|u - u^{\text{gcp}}_{\mathcal{I}}\|_2\right] \leq 4\epsilon_n$. However, the worst-case scenario is when $I' = I_{n-1}$, resulting in a classical combination of level $n-1$, which has the error bound

$$\begin{aligned} \|u - u^c_{n-1}\|_2 \leq \epsilon_{n-1} &= \frac{1}{3} \cdot 3^{-d} 2^{-2(n-1)} |u|_{H^2_{\text{mix}}} \sum_{k=0}^{d-1} \binom{n-1+d}{k} \left(\frac{1}{3}\right)^{d-1-k} \\ &\leq \frac{4}{3} \cdot 3^{-d} 2^{-2n} |u|_{H^2_{\text{mix}}} \sum_{k=0}^{d-1} \binom{n+d}{k} \left(\frac{1}{3}\right)^{d-1-k} \\ &= 4 \cdot \epsilon_n. \end{aligned}$$

This is consistent with the upper bound (3.6), which is the desired result. ∎

Note the assumption that each u_i is computed on a different node is not necessary as we have bounded the probability of a failure during the computation of u_i to be independent of the starting time. As a result our bound is independent of the number of nodes that are used during the computation and how the u_i are distributed among them as long as each individual u_i is not distributed across multiple nodes.

The nice thing about this result is that the bound on the expected error is simply a multiple of the error bound for u_n^c, i.e., the result in the absence of faults. If the bound on $\|u - u_n^c\|_2$ were tight, then one might expect

$$\mathbb{E}\left[\|u - u_{\mathcal{I}}^{\mathrm{gcp}}\|_2\right] \lessapprox \|u - u_n^c\|_2 \left(1 + 3\left(1 - e^{-(t_n/\lambda)^\kappa}\right)\right).$$

Also note that (3.5) can be expressed as

$$\mathbb{E}\left[\|u - u_{\mathcal{I}}^{\mathrm{gcp}}\|_2\right] \leq \|u - u_n^c\|_2 + \Pr(T \leq t_n)\|u_{n-1}^c - u_n^c\|_2 .$$

If the combination technique converges for u, then $\|u_{n-1}^c - u_n^c\|_2 \to 0$ as $n \to \infty$. Since $\Pr(T \leq t_n) \leq 1$, the error due to faults diminishes as $n \to \infty$. We now prove an analogous result for the case where only solutions with $\|i\|_1 < n-1$ are recomputed.

PROPOSITION 3.3. *Given $d, n > 0$ and $I_n := \{i \in \mathbb{N}^d : \|i\|_1 \leq n\}$, let u_i, t_i, and t_n be as described in Proposition* 3.2 *with each u_i computed on different nodes for which time between failures is independent and identically distributed having Weibull distribution with $\lambda > 0$ and $0 < \kappa \leq 1$. Additionally, let $t_{n-1} = \max_{\|i\|_1 = n-1} t_i$. Suppose we recompute any u_i with $\|i\|_1 < n-1$ which is interrupted by a fault; then let $\mathcal{I}$ be the set of all possible $I' \subseteq I_n$ for which u_i was successfully computed (eventually) iff $i \in I'$. Let $u_{\mathcal{I}}^{\mathrm{gcp}}$ be the function-valued random variable corresponding to the result of the FTCT; then*

$$\mathbb{E}\left[\|u - u_{\mathcal{I}}^{\mathrm{gcp}}\|_2\right] \leq \epsilon_n \cdot \min\left\{16, 1 + 3\left(d + 5 - e^{-\left(\frac{t_n}{\lambda}\right)^\kappa} - (d+4)e^{-\left(\frac{t_{n-1}}{\lambda}\right)^\kappa}\right)\right\}.$$

Proof. This is much the same as the proof of Proposition 3.2. The solution to the GCP for I_n satisfies the property that if $w_i = 0$ for $\|i\|_1 = n-1$, then u_i was not computed successfully; that is, $U_i = 1$. However, the converse does not hold in general. Regardless, if $U_i = 1$ for $\|i\|_1 = n-1$, then the worst case is that $w_i = 0$ and $w_j = 0$ for the d possible $\|j\|_1 = n$ satisfying $j > i$. We therefore note that the error generated by faults affecting u_i with $\|i\|_1 = n-1$ is bounded by

$$\sum_{j \in I_n,\, j \geq i} \|h_j\|_2 \leq (d+4)3^{-d}2^{-2n}|u|_{H^2_{\mathrm{mix}}}. \tag{3.7}$$

Therefore we have

$$\begin{aligned}
\mathbb{E}\left[\|u - u_{\mathcal{I}}^{\mathrm{gcp}}\|_2\right] &\leq \|u - u_n^c\|_2 + \sum_{\|i\|_1 = n} G(t_n)\|h_i\|_2 \\
&\quad + \sum_{\|i\|_1 = n-1} G(t_{n-1}) \sum_{j \in I,\, j \geq i} \|h_j\|_2 \\
&\leq \epsilon_n \left(1 + 3\left(1 - e^{-(t_n/\lambda)^\kappa}\right)\right) \\
&\quad + \binom{n-1+d-1}{d-1}\left(1 - e^{-(t_{n-1}/\lambda)^\kappa}\right)(d+4)3^{-d}2^{-2n}|u|_{H^2_{\mathrm{mix}}} \\
&\leq \epsilon_n \left(1 + 3\left(1 - e^{-(t_n/\lambda)^\kappa}\right) + 3(d+4)\left(1 - e^{-(t_{n-1}/\lambda)^\kappa}\right)\right).
\end{aligned}$$

Now the expected error should be no more than the worse case, which is $I' = I_{n-2}$, for which we have $\|u - u^c_{n-2}\|_2 \leq 16\epsilon_n$. Taking the minimum of the two estimates yields the desired result. ∎

To illustrate how this result may be used in practice, suppose we compute a level 12 interpolation in three dimensions on a machine whose mean time to failure can be modelled by the Weibull distribution with a mean of 100 seconds and shape parameter 0.7. Further, suppose u_i with $\|i\|_1 > 10$ are not recomputed if lost as a result of a fault and that t_{12} is estimated to be 1.0 second and t_{11} is at most 0.5 second. The expected error for our computation is bounded above by 1.63 times the error bound if no faults were to occur.

While this provides some theoretical validation of our approach, in practice we can numerically compute an improved estimate by enumerating all possible outcomes and the probability of each occurring. The reason for this is that (3.7) is an overestimate in general, particularly for relatively small d. In practice, a fault on u_i with $\|i\|_1 = n-1$ will generally result in the loss of $d-1$ of the largest hierarchical spaces, in which case (3.7) overestimates by a factor of $\frac{d+4}{d-1}$.

3.5. Expected computation time. We now repeat the above analysis, this time focusing on the mean time required for recomputations. The first issue to consider is that a failure may occur during a recomputation which will trigger another recomputation. Given a solution u_i with $\|i\|_1 = m$, the probability of having to recompute r times is bounded by $G(t_m)^r \leq F(t_m)^r$. Hence the expected number of recomputations for such a u_i is bounded by

$$\sum_{r=1}^{\infty} rF(t_m)^r = \frac{F(t_m)}{(1-F(t_m))^2} = e^{(t_m/\lambda)^\kappa}(e^{(t_m/\lambda)^\kappa} - 1).$$

Let the time required to compute each u_i be bounded by $t_i \leq c2^{\|i\|_1}$ for some fixed $c > 0$ and all $\|i\|_1 \leq n$. For a given $m \leq n$, suppose we intend to recompute all u_i with $\|i\|_1 \leq m$ upon failure; then the expected time required for recomputations is bounded by

$$R_m \leq \sum_{\|i\|_1 \leq m} t_{\|i\|_1} \sum_{r=1}^{\infty} rF(t_{\|i\|_1})^r \leq \sum_{k=0}^{m} \binom{k+d-1}{d-1} c2^k e^{(c2^k/\lambda)^\kappa} \left(e^{(c2^k/\lambda)^\kappa} - 1\right).$$

Further, by bounding components of the sum with the case $k = m$ one obtains

$$\begin{aligned} R_m &\leq \binom{m+d-1}{d-1} e^{(c2^m/\lambda)^\kappa} \left(e^{(c2^m/\lambda)^\kappa} - 1\right) \sum_{k=0}^{m} c2^k \\ &\leq \binom{m+d-1}{d-1} e^{(c2^m/\lambda)^\kappa} \left(e^{(c2^m/\lambda)^\kappa} - 1\right) c2^{m+1}. \end{aligned}$$

The time required to compute all u_i with $\|i\|_1 \leq n$ once is similarly bounded by

$$C_n = \sum_{\|i\|_1 \leq n} t_i \leq \sum_{k=0}^{n} \binom{k+d-1}{d-1} t_k \leq \binom{n+d-1}{d-1} c2^{n+1},$$

and hence $R_m/C_n \approx (m/n)^{d-1} 2^{m-n} e^{(c2^m/\lambda)^\kappa}(e^{(c2^m/\lambda)^\kappa} - 1)$ estimates the expected proportion of extra time spent on recomputations. We would generally expect that

$c2^m/\lambda \ll 1$ (we assume the time to compute level m grids is much less than the mean time to failure), and therefore this quantity is small. As an example, if we again consider a level 12 computation in three dimensions for which $t_m \leq 2^{-(m-12)}$ and the time to failure is Weibull distributed with mean 100 seconds with shape parameter 0.7, the expected proportion of time spent recomputing solutions level 10 or smaller is $(10/12)^2 2^{-2} e^{(0.25/100)^{0.7}}(e^{(0.25/100)^{0.7}} - 1) \approx 2.68 \times 10^{-3}$. In comparison, if any of the u_i which fail were to be recomputed, then a proportion of 4.23×10^{-2} additional time would be expected for recomputations, almost 16 times more. While this is a somewhat crude estimate, it clearly demonstrates that our approach will scale better than a traditional checkpoint restart when faults are relatively frequent.

4. Implementation details. At the heart of our implementation is a very simple procedure: Solve the problem on different grids, combine the solutions, and repeat. We distinguish between two layers, the top layer responsible for general management of the computation including global communications and coarse level parallelism, and the bottom layer consisting of solvers and processing tools utilizing fine grain parallelism.

4.1. Top layer: Load balancing and combination. The top layer is written in Python. As in many other applications, we use Python to glue together the different components of our implementation as well as provide some high level functions for performing the combination. This layer can be further broken down into four main parts. The first is the loading of all dependencies including various Python and `numpy` modules as well as any shared libraries that will be used to solve the given problem. In particular, bottom layer components which have been compiled into shared libraries from various languages (primarily C++ with C wrappers in our case) are loaded into Python using `ctypes`. The second part is the initialization of data structures and construction/allocation of arrays which will hold the relevant data. This is achieved using PyGraFT [16, 17], which is a general class of grids and fields that allows us to handle data from the various components in a generic way. The third part consists of solving the given problem. This is broken into several combination steps. A *combination step* consists of a series of time steps of the underlying solver for each component solution, followed by a combination of the component solutions into a sparse grid solution, and finally a sampling of the component solutions from the sparse grid solution before repeating the procedure. The fourth and final part of the code involves checking the error of the computed solutions, reporting of various log data, and finalizing/cleanup.

The top layer is primarily responsible for the coarsest grain parallelism, that is, distributing the computation of different component solutions across different processes and communication between them. This is achieved through MPI using `mpi4py`.[3] An appropriate load balancing of the different component solutions across the MPI processes is determined on startup and used throughout the computation. We found that simple models for load with respect to grid dimensions do not typically perform well with large numbers of grids. Therefore, after first making an initial allocation using a simple model, we then time the solver over each grid for a few iterations and then refine the allocation. In future work we intend to reassess this allocation after each combination step to further improve the load balancing and adjust for changes in system performance. Communication between MPI processes occurs during the combination step. The first step is to perform an ALL_REDUCE call to as-

[3]See http://mpi4py.scipy.org/.

sess on which grids a solution was successfully computed. Following this all processes are able to compute consistent combination coefficients that avoid solutions which did not successfully compute. The second step is to perform communications so that the combination may be computed on each process. Each grid is first processed from a nodal basis representation to a hierarchical basis representation. For each hierarchical surplus, a list of processes containing this surplus is generated. Hierarchical surpluses with common process groups are concatenated and then combined via an ALL_REDUCE call over the process subset. Once all hierarchical surpluses have been combined, each component grid is processed back to a nodal representation.

4.2. Bottom layer: Solver and sparse grid algorithms. The bottom layer is made up of several different components, many of which are specific to the problem that is to be solved. When solving our advection problem we have two main components: One is responsible for functions relating to the sparse grid (e.g., processing of nodal basis to hierarchical basis and vice versa), and the other component is the advection solver itself. Both the sparse grid and advection solver components use OpenMP to achieve a fine grain level of parallelism. In future work we intend the solver to be a hybrid MPI/OpenMP which is spawned from the top layer. This allows additional parallelism over each grid via domain decompositions and enables us to compute the solution for each grid across several nodes. However, the overheads associated with repeatedly spawning processors will also need to be investigated.

5. Numerical results. In this section, we present some numerical results which test our algorithm on solutions of the scalar advection equation

$$\frac{\partial u}{\partial t} + \nabla \cdot (au) = 0$$

on the domain $[0,1]^3 \subset \mathbb{R}^3$ for a divergence-free velocity field $a = a(x) \in \mathbb{R}^3$ (and thus $\nabla \cdot (au) = a \cdot \nabla u$). In particular, we set $a(x) = (\sin(\pi x)\cos(\pi y), -\cos(\pi x)\sin(\pi y), 1)$ for $x \in [0,1]^3$. Notice that the flow field is such that x and y boundary conditions are not required. In the z direction we have the periodic boundary condition $u(t,x,y,1) = u(t,x,y,0)$. We start the solver from the Gaussian-like initial condition

$$u_0(x) = \exp\left(-\frac{\pi^2}{2}\left(x - \frac{3}{8}\right)^2 - 2\pi^2\left(y - \frac{3}{8}\right)^2 - 2(1 + \cos(2\pi z))\right)$$

centered around $(\frac{3}{8}, \frac{3}{8}, \frac{1}{2})$. This peak flows in a helix-like pattern around the line $x = y = 1/2$. We evolve from $t = 0$ up to $t = 0.25$ in our experiments. The PDE is solved using second order central difference discretization of spatial derivatives and fourth order Runge–Kutta (RK4) time integration. We compare numerical solutions against the exact solution obtained by the method of characteristics computed with an RK4 solver.

Our experiments are performed with a truncated combination technique as in (2.4). In order to apply the FTCT effectively we compute some additional grids. We define

$$I_{n,\tau} := \{i \in \mathbb{N}^d : \min(i) \geq \tau \text{ and } n - d - 1 \leq \|i\|_1 \leq n\}$$

with n, τ denoting the level and truncation, respectively. For this choice of $I_{n,\tau}$ the computation of coefficients is simplified when only the top two levels are not recomputed in the event of faults as discussed in section 3.1.

TABLE 1

Numerical results for $r = 50$ runs with a level $n = 22$ and truncation $\tau = 5$ combination which consists of 110 grids. Faults were simulated with different MTBF on each MPI process (in seconds) to study the effect on the error and time to solution. The simulated faults were Weibull distributed with shape parameter of 0.7 in each case. The component solutions were combined four times throughout the computation. The computations were performed on 64 processors using eight MPI processes with eight OpenMP threads each. The times reported are for the inner computation loop and exclude overheads like the dynamic load balance calculation and error estimation.

MTBF	f_{ave}	ϵ_{ave}	ϵ_{min}	ϵ_{max}	w_{ave}	w_{min}	w_{max}
3600	1.22	1.132e-5	1.057e-5	1.815e-5	152.1	151.4	153.2
1800	1.78	1.159e-5	1.057e-5	1.848e-5	153.0	151.9	155.7
900	3.48	1.293e-5	1.057e-5	2.044e-5	150.9	149.0	153.5
450	5.66	1.412e-5	1.058e-5	2.048e-5	152.3	151.0	153.8
225	9.42	1.641e-5	1.059e-5	3.199e-5	153.0	151.4	155.0
112.5	16.00	1.803e-5	1.064e-5	3.452e-5	152.9	151.0	155.4
56.25	26.76	2.170e-5	1.159e-5	3.593e-5	154.2	152.0	157.9

Note that as the grid sizes vary between the u_i so too does the maximum stable time step size as determined by the CFL condition. We choose the same time step size for all component solutions to avoid instability that may otherwise arise from the extrapolation of time stepping errors during the combination. In particular, our time steps are such that the CFL condition is satisfied for all component grids. All of our computations were performed using the Raijin cluster at the National Computational Infrastructure.[4] Raijin is a Fujitsu PRIMERGY cluster consisting of 3592 compute nodes, each with two Intel Xeon Sandy Bridge CPUs (8 core, 2.6GHz) with Infiniband FDR interconnect.

5.1. Solution error. We first studied the effect of simulated faults on the error of the computed solution. Given level n and truncation parameter τ, the code was executed for some number of runs r on a fixed number of nodes using the same number of threads. Component solutions are combined four times in each run at equally spaced intervals. For each run we recorded the number of faults f_k that occurred, the l_1 error of the solution ϵ_k, and the elapsed wall time w_k throughout the main computation. Over r runs we then calculated the average number of faults $f_{ave} = \frac{1}{r}\sum_{k=1}^{r} f_k$; the average, minimal, and maximal observed errors $\epsilon_{ave} = \frac{1}{r}\sum_{k=1}^{r} \epsilon_k$, $\epsilon_{min} = \min\{\epsilon_1, \ldots, \epsilon_r\}$, $\epsilon_{max} = \max\{\epsilon_1, \ldots, \epsilon_r\}$; and the average, minimal, and maximal observed wall times $w_{ave} = \frac{1}{r}\sum_{k=1}^{r} w_k$, $w_{min} = \min\{w_1, \ldots, w_r\}$, $w_{max} = \max\{w_1, \ldots, w_r\}$.

Table 1 shows results for $r = 50$ runs of the FTCT with fault simulation on a problem with $n = 22$, $\tau = 5$ consisting of 110 grids and computed on eight MPI processes with eight OpenMP threads each. Faults were simulated as described in section 3.3 using the Weibull distribution with varying MTBF per MPI process and shape parameter 0.7. Decreasing the MTBF leads to an increase in the average number of faults that occur per run, as one would expect. The time to solution is not significantly affected by the number of faults, as seen in t_{ave}, which varies relatively little over a 10-fold increase in the average number of faults. It is clear that the error ϵ_{ave} increases with f_{ave}, and from ϵ_{min} and ϵ_{max} one has some indication that the variance in error also increases. For MTBF of 450 or more seconds, the effect is relatively small. For higher frequencies of failure, there is a noticeable effect, but even at 27 faults per run the average error is approximately twice that without faults.

[4] See http://nci.org.au/.

TABLE 2

Numerical results for $r = 50$ runs with a level $n = 22$ and truncation $\tau = 6$ combination which consists of 35 grids. Faults were simulated with different MTBF on each MPI process (in seconds) to study the effect on the error and time to solution. The simulated faults were Weibull distributed with shape parameter of 0.7 in each case. The component solutions were combined four times throughout the computation. The computations were performed on 32 processors using four MPI processes with eight OpenMP threads each. The times reported are for the inner computation loop and exclude overheads like the dynamic load balance calculation and error estimation.

MTBF	f_{ave}	ϵ_{ave}	ϵ_{min}	ϵ_{max}	w_{ave}	w_{min}	w_{max}
128	3.08	1.671e-4	1.305e-4	3.727e-4	32.99	32.56	33.43
64	4.32	1.693e-4	1.305e-4	2.938e-4	33.04	32.63	33.60
32	7.02	1.791e-4	1.305e-4	2.681e-4	33.07	32.63	33.77
16	12.02	2.191e-4	1.340e-4	5.722e-4	33.01	32.61	33.68
8	19.04	2.700e-4	1.393e-4	4.785e-4	33.22	32.64	33.79
4	31.58	3.382e-4	1.830e-4	6.305e-4	33.40	32.80	35.28

Table 2 again shows results for $r = 50$ runs of the FTCT with fault simulation, this time with an $n = 22$, $\tau = 6$ truncated combination consisting of only 35 grids. Having fewer grids and therefore less redundancy, this problem is more sensitive to faults. Faults were again simulated as Weibull distributed with varying means and shape parameter 0.7. The MTBF values are significantly smaller here to investigate what happens to this problem for a large number of faults. The same observations can be made as in Table 1, namely, that the time to solution is not significantly affected by the failure rate and that the mean and variance of the error increase with f_{ave}. While growing faster for this problem at almost 32 faults on average, the error is less than three times the error when no faults occur.

While not shown here, we have observed that if the number of grids is large enough, then the expected error initially decreases as the number of faults increases. We suspect this occurs because such a large number of grids becomes a suboptimal use of resources and the solution error becomes dominated by terms which are additive when the combination technique is applied. As more grids fail, there are fewer such terms in the resulting solution, leading to smaller error. Eventually, with enough failures, the error again increases as the dominating terms become those which cancel, and similar trends as above are observed from this point. Determining the optimal number of grids in terms of error versus time to solution is problem dependent but can be estimated by careful study of error bounds based on error splittings [8].

5.2. Parallel and algorithmic scalability. In Figure 2 we demonstrate the parallel scalability and efficiency of our implementation both with and without fault simulation for a reasonably high failure rate. The advection problem was solved using an $n = 22$, $\tau = 5$ truncated combination having 110 grids and an $n = 22$, $\tau = 4$ truncated combination having 230 grids. The component solutions were combined four times throughout each computation. The times reported here include the timing of the core of the code, which is the repeated computation, combination, and communication of the solution. Start-up and completion overheads, including Python imports, the dynamic load balancing procedure, and the error calculation, are excluded. It is clear that the implementation scales very well as far as the distribution of grids to nodes will allow. In particular, it is apparent that adding fault resilience has had negligible impact on the speedup of the application. Therefore, for an application that is otherwise capable of scaling to an exascale system, it is not anticipated that adding fault resilience via this method would be a barrier to deployment on such a system. In our test case, by further increasing the number of grids and computing

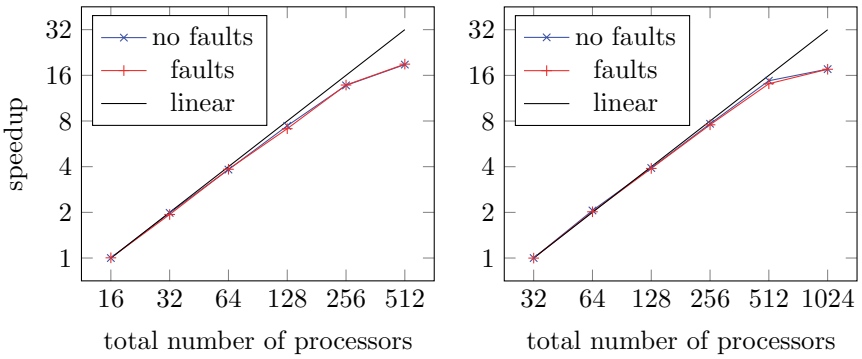

FIG. 2. *The plots demonstrate the parallel scaling of our implementation with an* $l = 22$, $\tau = 5$ *truncated combination on the left and an* $l = 22$, $\tau = 4$ *truncated combination on the right. Each problem has* 110 *and* 230 *coarse grids, respectively. Given a fixed number of grids, one observes that the scalability eventually drops off, which is due to the distribution of grids to nodes. However, also notice that additional grids lead to increased parallelism. By increasing the number of grids or computing each grid over several nodes we would expect good scalability for many more processors. The plots include scalability both with and without the fault simulation and recovery. It is clear that our approach to fault recovery has no observable effect on the scalability. The MTBF per node used for fault simulation on the* $\tau = 5$ *problem was* 128 *seconds, which led to* 9–36 *faults occurring in each of the computations. For the* $\tau = 4$ *problem, the MTBF per node was* 900 *seconds, which led to* 12–41 *faults occurring in each of the computations.*

each grid across several nodes, we expect the implementation to scale much further. We note, however, that increasing the number of grids does not necessarily lead to improved error.

In Figure 3 we compare the computation time required for our approach to reach a solution compared to more traditional checkpointing approaches, in particular, with local and global checkpointing approaches. With global checkpointing we keep a copy of the last combined solution. If a failure affects any of the component grids, it is assumed that the entire application is killed, and computations must be restarted from the most recent combined solution. We emulate this by checking for faults at each combination step and restart from the last combination step if any faults have occurred. With local checkpointing each MPI process saves a copy of each component solution it computes. In this case when a fault affects a component solution we recompute the affected component solution from its last saved state. In both checkpointing methods the extra component solutions used in our approach are not required and are hence not computed. As a result these approaches are slightly faster when no faults occur. However, as the number of faults increases, it can be seen from Figure 3 that the computation time for the local and global checkpointing methods begins to grow. A line of best fit has been added to the figure which makes it clear that the time for recovery with global checkpointing increases rapidly with the number of faults. Local checkpointing is a significant improvement on this but still shows an increase in recovery time. On the other hand, our approach is barely affected by the number of faults and beats both the local and global checkpointing approaches after only a few faults.

Conclusion. A generalization of the sparse grid combination technique has been presented. From this a fault tolerant combination technique able to handle large numbers of failures has been developed. Through some calculations and numerical experiments it has been demonstrated that the approach reduces recovery time at

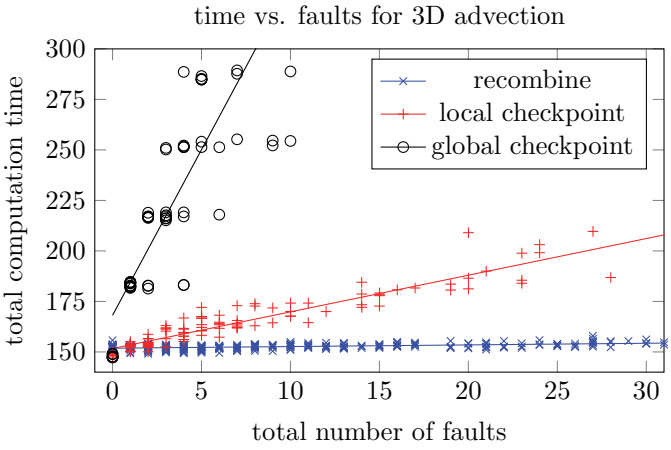

FIG. 3. *We compare the time taken to compute the solution to the three-dimensional advection problem using three different approaches to fault tolerance. The problem size is fixed at level* 22 *with truncation parameter* 5. *All computations used eight MPI processes with eight OpenMP threads each. Component solutions are combined four times throughout the computation, and it is during the combination that we check for faults. The* recombine *method is our approach described in section* 3.1. *The* local checkpoint *method involves each MPI process checkpointing component solutions. The global checkpoint method involves all MPI processes checkpointing the last combined solution. For each method the problem was run several times with many different MTBF values per MPI process to study the effect of the number of faults on the run time.*

the expense of reduced solution accuracy. Theoretical bounds on the expected error and numerical experiments show that the reduction in solution accuracy is typically small. Comparison with traditional checkpoint-restart techniques shows that the run time of our algorithm is not significantly affected by the number of faults. There are many opportunities for further development. We intend to expand on error bounds presented here and, in particular, extend the error splitting estimates used in the analysis of the classical combination technique. Detailed investigation into the performance where the u_i are distributed on many nodes is also the subject of future work; our implementation continues to develop as we seek to support this. As the ULFM specification continues to develop, the validation of the FTCT on a system with real faults is also being investigated.

Acknowledgment. The authors would like to thank the anonymous referees for their valuable feedback which improved the quality of this paper.

REFERENCES

[1] G. BOSILCA, R. DELMAS, J. DONGARRA, AND J. LANGOU, *Algorithm-based fault tolerance applied to high performance computing*, J. Parallel Distrib. Comput., 69 (2009), pp. 410–416.

[2] F. CAPPELLO, *Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities*, Int. J. High Perform. Comput. Appl., 23 (2009), pp. 212–226.

[3] F. CAPPELLO, A. GEIST, B. GROPP, L. KALE, B. KRAMER, AND M. SNIR, *Toward exascale resilience*, Int. J. High Perform. Comput. Appl., 23 (2009), pp. 374–388.

[4] J. DEAN AND S. GHEMAWAT, *MapReduce: Simplified data processing on large clusters*, Comm. ACM, 51 (2008), pp. 107–113.

[5] J. GARCKE, *Sparse grids in a nutshell*, in Sparse Grids and Applications, Lect. Notes Comput. Sci. Eng. 88, J. Garcke and M. Griebel, eds., Springer-Verlag, Berlin, Heidelberg, 2013, pp. 57–80.

[6] G. Gibson, B. Schroeder, and J. Digney, *Failure tolerance in petascale computers*, CTWatch Quart., 3 (2007), pp. 4–10.
[7] M. Griebel and H. J. Bungartz, *Sparse grids*, Acta Numer., 13 (2004), pp. 147–269.
[8] M. Griebel, M. Schneider, and C. Zenger, *A combination technique for the solution of sparse grid problems*, in Iterative Methods in Linear Algebra (Brussels, 1991), North-Holland, Amsterdam, 1992, pp. 263–281.
[9] B. Harding and M. Hegland, *A robust combination technique*, ANZIAM J. Electron. Suppl., 54 (2013), pp. C394–C411.
[10] B. Harding and M. Hegland, *Robust solutions to PDEs with multiple grids*, in Sparse Grids and Applications - Munich 2012, Lect. Notes Comput. Sci. Eng. 97, J. Garcke and D. Pflüger, eds., 2014, Springer International, Cham, Switzerland, pp. 171–193.
[11] B. Harding and M. Hegland, *A parallel fault tolerant combination technique*, in Parallel Computing: Accelerating Computational Science and Engineering (CSE), Adv. Parallel Comput. 25, M. Bader, A. Bode, H.-J. Bungartz, M. Gerndt, G. R. Joubert, and F. Peters, eds., IOS Press, Amsterdam, 2014, pp. 584–592.
[12] M. Hegland, *Adaptive sparse grids*, ANZIAM J., 44 (2003), pp. C335–C353.
[13] M. Hegland, J. Garcke, and V. Challis, *The combination technique and some generalisations*, Linear Algebra Appl., 420 (2007), pp. 249–275.
[14] K. Huang and J. Abraham, *Algorithm-based fault tolerance for matrix operations*, IEEE Trans. Comput., 33 (1984), pp. 518–528.
[15] R. M. Karp, *Reducibility among combinatorial problems*, in Proceedings of a Symposium on the Complexity of Computer Computations (Yorktown Heights, 1972), IMB Res. Symposia Ser., R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[16] J. Larson, M. Hegland, B. Harding, S. Roberts, L. Stals, A. Rendell, P. Strazdins, 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 Comput. Sci., 18 (2013), pp. 130–139.
[17] J. Larson, M. Hegland, B. Harding, S. Roberts, L. Stals, A. Rendell, P. Strazdins, M. Ali, and J. Southern, *Managing complexity in the parallel sparse grid combination technique*, in Parallel Computing: Accelerating Computational Science and Engineering (CSE), Adv. Parallel Comput. 25, M. Bader, A. Bode, H.-J. Bungartz, M. Gerndt, G. R. Joubert, and F. Peters, eds., IOS Press, Amsterdam, 2014, pp. 593–602.
[18] A. Murarasu, J. Weidendorfer, G. Buse, D. Butnaru, and D. Pflüger, *Compact data structure and scalable algorithms for the sparse grid technique*, ACM SIGPLAN Not., 46 (2011), pp. 25–34.
[19] B. Schroeder and G. Gibson, *A large-scale study of failures in high-performance computing systems*, in Proceedings of the International Conference on Dependable Systems and Networks, IEEE Computer Society, Washington, DC, 2006, pp. 249–258.
[20] K. Trivedi, *Probability and Statistics with Reliability Queuing and Computer Science Applications*, John Wiley & Sons, New York, 2002.