# A robust combination technique

B. Harding[1] M. Hegland[2]

(Received 31 October 2012; revised 5 June 2013)

**Abstract**

One of the challenges for efficiently and effectively using petascale and exascale computers is the handling of run-time errors. Without such robustness, applications developed for these machines will have little chance of completing successfully. The sparse grid combination technique approximates the solution to a given problem by taking the linear combination of its solution on multiple grids. It is successful in many high performance computing applications due to its ability to tackle the curse of dimensionality. We present several approaches to fault tolerance using the combination technique. The first of these is implemented within the MapReduce model in order to utilise the existing fault tolerance of this framework. In addition, we present a method which utilises the redundancy shared by solutions on different grids. Finally, we describe a novel approach in which the solution is computed on additional grids which are used for alternative combina-

http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/6321 gives this article, © Austral. Mathematical Soc. 2013. Published August 16, 2013, as part of the Proceedings of the 16th Biennial Computational Techniques and Applications Conference. ISSN 1446-8735. (Print two pages per sheet of paper.) Copies of this article must not be made otherwise available on the internet; instead link directly to this URL for this article.

tions if other grids experience failure. We include some results based on the solution of the 2D scalar advection PDE.

## Contents

## 1 Introduction

Modern high performance computing architectures are extremely complex, having hundreds of thousands of components all working together to complete cumbersome tasks. As the computing power of machines in the Top500[1] increases, so too does the number of components. Given the limited lifetime of each component, the probability that any one of these will fail within a given period of time also increases. Numerous studies of faults indicated that software errors are as, if not more, significant [4]. This is a challenge that must be addressed if we are to achieve exascale computing.

Many traditional approaches to fault tolerance, like checkpoint restart based methods, are unfeasible at exascale because they are too memory intensive,

[1] http://www.top500.org

leading to an inefficient use of resources and high energy costs. Algorithm based fault tolerance (ABFT) was studied for a variety of problems and often provides a robust low cost solution [2, 9].

The sparse grid combination technique was introduced by Griebel et al. [7]. By solving a given problem on many regular anisotropic grids and taking a linear combination of the results one obtains the approximate sparse grid solution [3, 10]. In Section 2 we describe how this method is implemented within the MapReduce model which was recently popularised by Google [5]. In MapReduce a list of (key,value) pairs is distributed to a *map* function which, after some processing, returns new (key,value) pairs. This second list of pairs is then processed/sorted according to the key by a *reduce* function. For the combination technique one defines a *map* function to solve a problem on each grid and return (grid,solution) pairs which are later combined by the *reduce* function. Fault tolerance is achieved by recomputing pairs that are affected by faults as per the MapReduce model.

In many cases recomputation may be neither possible nor desirable. One issue is that unexpected recomputations may result in poor load balancing. This leads to inefficient use of resources and may even prevent computations from being completed in the allocated time. With this in mind, in Section 3 we present some approaches to ABFT in which (grid,solution) pairs experiencing a fault are not recomputed. The first of these approaches utilises redundancy within the computed grids to approximate lost (grid,solution) pairs from other computed pairs. The second approach involves solving the problem on additional grids which can be used to derive new combinations that avoid pairs which have experienced a fault.

Finally, in Section 4 we present numerical results in the context of the scalar advection PDE which demonstrate the extra errors produced using our methods. In particular, our latter approach appears to have a nice bound on the error even when multiple (grid,solution) pairs are affected by faults.

## 2 The combination technique

Let $\underline{i} \in \mathbb{N}^d$ be a multi-index and $\mathcal{I}_k = \{0, 2^{-k}, 2 \times 2^{-k}, 3 \times 2^{-k}, \ldots, 1\}$ be a discretisation of the unit interval. We define a grid on the unit $d$ cube $\Omega$ by

$$\Omega_{\underline{i}} := \mathcal{I}_{i_1} \times \mathcal{I}_{i_2} \times \cdots \times \mathcal{I}_{i_d} .$$

Given a grid $\Omega_{\underline{i}}$ we define an associated space of piecewise $d$ linear functions

$$V_{\underline{i}} := \operatorname{span}\{\phi_{\underline{i},\underline{j}} : j_k \in 2^{i_k}\mathcal{I}_{\underline{i}_k} , k = 1, \ldots, d\} ,$$

where $\phi_{\underline{i},\underline{j}}$ are the usual $d$ linear hat functions [6].

The sparse grid combination technique [6, 7] allows one to approximate the sparse grid solution [3] by taking linear combinations of the solution on multiple regular anisotropic grids. Suppose $f_{\underline{i}} \in V_{\underline{i}}$ denotes the solution to a given problem on the grid $\Omega_{\underline{i}}$ , then the combination solution is

$$f_n^c(\underline{x}) := \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\underline{i}|_1 = n-q} f_{\underline{i}}(\underline{x}) . \tag{1}$$

For $f \in H^2_{0,\text{mix}}(\Omega)$ , where $f$ satisfies $\|f\|^2_{H^2_{\text{mix}}} := \sum_{\underline{i}_1,\ldots,\underline{i}_d=0}^{2} \|D^{\underline{i}} f\|_2^2 < \infty$ and has zero boundary values, it is known that the combination interpolant is equal to the sparse grid interpolant [6] and has interpolation error

$$\|f - f_n^c\|_2 = \mathcal{O}\left[h_n^2 \cdot \log(h_n^{-1})^{d-1}\right] ,$$

where $h_n = 2^{-n}$ . The combination technique is generalizable in a way that makes it extremely flexible [8]. In this general setting

$$f_n^c(\underline{x}) = \sum_{\underline{i} \in I} c_{\underline{i}} f_{\underline{i}}(\underline{x}) . \tag{2}$$

The classical combination of (1) is written in this form by setting $I = I_n^d$ where

$$I_n^d := \{\underline{i} \in \mathbb{N}^d : n - d < |\underline{i}|_1 \leqslant n\} , \tag{3}$$

and deriving the coefficients from the index,

$$c_{\underline{i}} = (-1)^{n-|\underline{i}|_1} \binom{d-1}{n-|\underline{i}|_1} .$$

The combination technique is an ideal candidate for implementing within the MapReduce model. Consider the solution to a PDE problem. Starting with (key,value) pairs given by (grid index $\underline{i}$, initial/boundary conditions), these are processed independently by the *map* function which returns $(\underline{i}, f_{\underline{i}})$ pairs. The *reduce* function then combines the solutions by adding them together according to (2).

MapReduce was originally designed to process large data sets over distributed networks and therefore necessarily includes fault tolerant mechanisms. Google [5] uses MapReduce to process data relating to their web search service and implements the following approach to fault tolerance.

**Worker failure:** Workers are periodically pinged by the master. If no response is received, then it is assumed the worker has failed and the (key,value) pairs that were assigned to it are rescheduled to other workers.

**Master failure:** The master is periodically checkpointed. When a failure occurs the master is restarted from a previous checkpoint and all (key,value) pairs that were incomplete at the time of the checkpoint must be rescheduled.

Checkpointing the master process is straightforward and does not have the problems associated with global checkpoints since only a small amount of data relating to the job control needs to be duplicated.

Whilst this approach can be applied with the combination technique, rescheduling and recomputing can result in poor load balancing. Load balancing is generally not an issue with MapReduce because the number of *map* operations greatly exceeds the number of threads. With the combination technique this may not always be the case. For example, consider a 6D problem with level

$n = 16$ which consists of $66\,605$ grids. If one grid is distributed to each socket then, given the $\mathcal{O}(100\,000)$ sockets in current petascale machines, one sees that we cannot efficiently use the entire machine. Even when allocated only a portion of the machine, achieving good load balancing is tedious and having to recompute a grid may cause significant delays. This challenge motivates the next section.

## 3 Robust algorithms

We present two approaches to fault tolerance which trade-off some error in the solution for the need to recompute. The first of these utilises redundancy between the combination grids. Suppose that $\underline{i} \in I_n^d$, as defined in (3), and $|\underline{i}|_1 < n$, then there are at least $d$ elements $\underline{j} \in I_n^d$ such that $\underline{i} \leqslant \underline{j}$, that is $\underline{i}_k \leqslant \underline{j}_k$ for all $k = 1, \ldots, d$. In particular, one may take $\underline{j} = \underline{i} + \underline{e}^k$ where $\underline{e}^k$ is $1$ for the $k$th element and $0$ elsewhere. For each of these we have that $\Omega_{\underline{i}} \subset \Omega_{\underline{j}}$ and $V_{\underline{i}} \subset V_{\underline{j}}$ so it is therefore reasonable to expect that the interpolation of $f_{\underline{j}}(x)$ onto the grid $\Omega_{\underline{i}}$ will produce an acceptable approximation to $f_{\underline{i}}(x)$, that is,

$$f_{\underline{i}} \approx P_{\underline{i}} f_{\underline{j}} \,,$$

where $P_{\underline{i}}$ is the interpolation into the space $V_{\underline{i}}$. For example, if $f_{\underline{i}}$ and $f_{\underline{j}}$ were simply interpolations of $f$, then the interpolation of $f_{\underline{j}}$ onto $V_{\underline{i}}$ is exactly $f_{\underline{i}}$. Similarly, if $f_{\underline{i}}$ are Galerkin projections of $f$ and $P_{\underline{i}}$ is the Galerkin projection onto a coarser space, then $f_{\underline{i}} = P_{\underline{i}} f_{\underline{j}}$. This observation forms the basis of our first fault tolerant approach; faults affecting an $(\underline{i}, f_{\underline{i}})$ pair with $|\underline{i}| = n$ are recomputed whilst for all other pairs (with $|\underline{i}| < n$) the solution $f_{\underline{i}}$ is approximated by taking the interpolation of the solution from a finer grid $f_{\underline{j}}$.

Care must be taken with respect to the choice of grid from which to project the solution as the solution error is sensitive to this choice. This is seen if one assumes the error model

$$\epsilon_{\underline{i}} = \sum_{k=1}^{d} \gamma_{\underline{i}_k} h_{\underline{i}_k}^2 + \mathcal{O}(h^4)\,,$$

and then examines the resulting extrapolation error after the combination technique is applied.

Our second approach is motivated by the work of Hegland [8] where an algorithm for adaptive sparse grids was introduced. This algorithm involved finding new combination coefficients when the solution from new grids is added to the combination. In contrast, the occurrence of faults effectively takes grids away. Despite this we demonstrate that the coefficients can be updated in a similar way. We start with the linear projection operator

$$P_I = 1 - \prod_{\underline{i}\in I}(1 - P_{\underline{i}})\,, \tag{4}$$

where $I$ is an ordered downset containing the indices of nested function space lattices $V_{\underline{i}} = V_{i_1} \times \cdots \times V_{i_d}$, with $P_{\underline{i}} : V \to V_{\underline{i}}$ and $P_I : V \to V_I = \sum_{\underline{i}\in I} V_{\underline{i}}$. Given that $P_{\underline{i}}P_{\underline{j}} = P_{\underline{i}}$ when $\underline{i} \leqslant \underline{j}$ (that is $\underline{i}_k \leqslant \underline{j}_k$ for $k = 1, \ldots, d$) the operator reduces to

$$P_I = 1 - \prod_{\underline{i}\in \max I}(1 - P_{\underline{i}})\,, \tag{5}$$

where $\max I := \{\underline{i} \in I : \text{for all } \underline{j} \in I\,, \text{there exists } k \text{ such that } \underline{j}_k < \underline{i}_k\}$ are the maximal elements of $I$. Further, using the property $P_{\underline{i}}P_{\underline{j}} = P_{\underline{i}\wedge\underline{j}}$, this expands to

$$P_I = \sum_{\underline{i}\in I} c_{\underline{i}} P_{\underline{i}}\,, \tag{6}$$

where the $c_{\underline{i}}$ are the resulting combination coefficients for each of the indices in $I$ (note the similarity with (2)).

In order to apply this approach we compute the solutions on all of the grids with indicies in the smallest downset containing $I_n^d$ given by

$$J_n^d := \{\underline{i} \in \mathbb{N}^d : |\underline{i}|_1 \leqslant n\}\,. \tag{7}$$

If no faults occur during the course of the computation, then we combine solutions as in (1). The same also applies if faults only affect the grids with indicies $|\underline{i}|_1 \leqslant n - d$. On the other hand, if a fault affects a grid with index $\underline{i} \in J_n^d \cap I_n^d$, then our goal is to derive new coefficients such that $c_{\underline{i}} = 0$ and we utilise as many of the remaining grids as possible in a suitable way.

Consider a fault occuring on an $(\underline{i}, f_{\underline{i}})$ pair with grid index $\underline{j} \in \max J_n^d$, that is $|\underline{j}|_1 = n$. Observing that $J' = J_n^d \backslash \{\underline{j}\}$ is still a downset, we define the projection $P_{J'}$ as in (5) and expanded to

$$P_{J'} = \sum_{\underline{i} \in J'} c_{\underline{i}} P_{\underline{i}} \,,$$

giving us new combination coefficients which exclude the index $\underline{j}$.

*Remark* 1. For only one failure occurring on elements in $\max J_n^d$ we do not need all of the computed grids. Numerical experiments indicate that only the grids with $n - d \leqslant |\underline{i}|_1 \leqslant n$ are required.

For a fault occurring on a grid with index $n - d < |\underline{j}|_1 < n$ we can estimate the solution by taking the projection from a finer solution, as previously discussed. Alternatively, we could remove this element and all larger elements from $J_n^d$, then find the maximal elements and expand the projection operator as before to obtain new coefficients. However, this latter approach can give poor results, particularly in higher dimensions where it may mean removing tens or even hundreds of indices. Instead it is possible to remove just a few elements from $\max J_n^d$ such that the expanded projection operator has a coefficient of zero for the $\underline{j}$ which experienced a fault.

This zero coefficient method is most easily demonstrated in the 2D case. Suppose a fault occurs on a grid with index $(k, n-k-1) \in J_n^2$ for some $k \in \{0, \ldots, n-1\}$. If one defines $J' = J_n^2 \backslash \{(k+1, n-k-1)\}$ or $J' = J_n^2 \backslash \{(k, n-k)\}$ then upon expanding $P_{J'}$ one finds that $c_{(k,n-k-1)} = 0$. A similar condition arises in high dimensions where removing one or more appropriate maximal elements yields coefficients of zero for the desired grid indices.

*Remark* 2. Often it is possible to set $c_{\underline{i}} = 0$ for all $|\underline{i}|_1 < n - d$ which means we need only compute the grids with indices in $J_n^d \backslash J_{n-d-1}^d$. This seems to hold where faults affect only one grid.

We now give a result which provides an asymptotic estimate of the proportion of extra grid points in the computations of all the grids with indicies in $J_n^d$ compared to those in the grids with indicies in $I_n^d$.

**Proposition 3.** *Let $|\Omega_{\underline{i}}|$ denote the total number of grid points in $\Omega_{\underline{i}}$, $I_n^d := \{\underline{i} \in \mathbb{N}^d : n - d < |\underline{i}|_1 \leqslant n\}$ and $J_n^d := \{\underline{i} \in \mathbb{N}^d : |\underline{i}|_1 \leqslant n\}$. The sum of grid points in the collections of grids with indices in $I_n^d$, $J_n^d$ and $J_n^d \backslash J_{n-d-1}^d$ is*

$$L_n = \sum_{\underline{i} \in I_n^d} |\Omega_{\underline{i}}|\,, \quad M_n = \sum_{\underline{i} \in J_n^d} |\Omega_{\underline{i}}|\,, \quad N_n = \sum_{\underline{i} \in J_n^d \backslash J_{n-d-1}^d} |\Omega_{\underline{i}}|\,,$$

*respectively. In the limit of large levels n, the proportions*

$$\lim_{n \to \infty} \frac{M_n - L_n}{L_n} = \frac{1}{2^d - 1} \quad \textit{and} \quad \lim_{n \to \infty} \frac{N_n - L_n}{L_n} = \frac{1}{2(2^d - 1)}\,. \tag{8}$$

**Proof:** We define $a_{k,d}$ to be the sum of grid points in the collection of grids of level $k$, that is

$$a_{k,d} = \sum_{\underline{i} \in J_k^d \backslash J_{k-1}^d} |\Omega_{\underline{i}}|\,.$$

By simple combinatorical arguments one can show that the number of elements of $J_k^d \backslash J_{k-1}^d$ is $\binom{k+d-1}{d-1} = k^{d-1}/(d-1)! + \mathcal{O}(k^{d-2})$ [3]. If one has periodic boundaries then each grid of level $k$ has $2^k$ grid points and it follows that

$$\sum_{\underline{i} \in J_k^d \backslash J_{k-1}^d} |\Omega_{\underline{i}}| = \frac{k^{d-1} 2^k}{(d-1)!} + \mathcal{O}(k^{d-2} 2^k)\,.$$

For non-periodic boundaries the situation is more challenging. For $d = 1$ one has $a_{k,1} = 2^k + 1 = k^0 2^k / 0! + \mathcal{O}(k^{-1} 2^k)$. Then, by induction on $d \geqslant 2$ with the assumption $a_{k,d-1} = k^{d-2}2^k/(d-2)! + \mathcal{O}(k^{d-3}2^k)$,

$$\begin{aligned}
a_{k,d} &= \sum_{i=0}^{k}(2^i+1)a_{k-i,d-1} \\
&= \sum_{i=0}^{k}(2^i+1)\left(\frac{(k-i)^{d-2}}{(d-2)!}2^{k-i} + \mathcal{O}\left[(k-i)^{d-3}2^{k-i}\right]\right) \\
&= \sum_{i=0}^{k}\left(\frac{(k-i)^{d-2}}{(d-2)!}(2^k+2^{k-i}) + \mathcal{O}\left[(k-i)^{d-3}(2^k+2^{k-i})\right]\right) \qquad (9) \\
&= \frac{2^k}{(d-2)!}\sum_{i=0}^{k}(k-i)^{d-2} + \frac{1}{(d-2)!}\sum_{i=0}^{k}2^{k-i}(k-i)^{d-2} \\
&\quad + \mathcal{O}\left[2^k\sum_{i=0}^{k}(k-i)^{d-3} + \sum_{i=0}^{k}2^{k-i}(k-i)^{d-3}\right].
\end{aligned}$$

Estimates for the two sums are

$$\sum_{i=0}^{k}(k-i)^p = \sum_{i=0}^{k}i^p = \frac{k^{p+1}}{p+1} + \mathcal{O}(k^p)$$

and

$$\begin{aligned}
\sum_{i=0}^{k}2^{k-i}(k-i)^p &= 2^k\sum_{i=0}^{k}2^{-i}\left[k^p + \mathcal{O}(k^{p-1})\right] \\
&= 2^k\left[k^p + \mathcal{O}(k^{p-1})\right]\sum_{i=0}^{k}2^{-i} \\
&= 2^k\left[k^p + \mathcal{O}(k^{p-1})\right](2-2^{-k}) \qquad (10) \\
&= (2^{k+1}-1)\left[k^p + \mathcal{O}(k^{p-1})\right] \\
&= 2^{k+1}k^p + \mathcal{O}(2^k k^{p-1}).
\end{aligned}$$

Substituting these sums into (9) yields

$$\begin{aligned}
a_{k,d} &= \frac{2^k}{(d-2)!}\left[\frac{k^{d-1}}{d-1} + \mathcal{O}(k^{d-2})\right] + \frac{1}{(d-2)!}\mathcal{O}(2^k k^{d-2}) + \mathcal{O}(2^k k^{d-2} + 2^k k^{d-3}) \\
&= \frac{k^{d-1}}{(d-1)!}2^k + \mathcal{O}(k^{d-2}2^k)\,.
\end{aligned}$$

Hence the assumption holds for all $d \geqslant 1$. Now it remains to sum over all of the levels in the sets $I_n^d$, $J_n^d$ and $J_n^d \backslash J_{n-d-1}^d$. For $I_n^d$ one again applies the estimate of (10) to obtain

$$\begin{aligned}
L_n = \sum_{k=n-d+1}^{n} a_{k,d} &= \sum_{k=n-d+1}^{n}\left[\frac{k^{d-1}2^k}{(d-1)!} + \mathcal{O}(k^{d-2}2^k)\right] \\
&= \frac{1}{(d-1)!}\sum_{k=0}^{n}\left[k^{d-1}2^k + \mathcal{O}(k^{d-2}2^k)\right] \\
&\quad - \frac{1}{(d-1)!}\sum_{k=0}^{n-d}\left[k^{d-1}2^k + \mathcal{O}(k^{d-2}2^k)\right] \\
&= \frac{n^{d-1}}{(d-1)!}(2^{n+1} - 2^{n-d+1}) + \mathcal{O}(n^{d-2}2^n) \\
&= \frac{n^{d-1}2^{n-d+1}(2^d-1)}{(d-1)!} + \mathcal{O}(n^{d-2}2^n)\,.
\end{aligned}$$

Similarly, for $J_n^d$ and $J_n^d \backslash J_{n-d-1}^d$ it can be shown that

$$\begin{aligned}
M_n &= \sum_{k=0}^{n}\left[\frac{k^{d-1}2^k}{(d-1)!} + \mathcal{O}(k^{d-2}2^k)\right] = \frac{n^{d-1}2^{n+1}}{(d-1)!} + \mathcal{O}(n^{d-2}2^n)\,, \\
N_n &= \sum_{k=n-d}^{n}\left[\frac{k^{d-1}2^k}{(d-1)!} + \mathcal{O}(k^{d-2}2^k)\right] = \frac{n^{d-1}2^{n-d}(2^{d+1}-1)}{(d-1)!} + \mathcal{O}(n^{d-2}2^n)\,.
\end{aligned}$$

It is then straightforward to show that

$$\frac{M_n - L_n}{L_n} = \frac{n^{d-1}2^{n-d+1} + \mathcal{O}(n^{d-2}2^n)}{n^{d-1}2^{n-d+1}(2^d-1) + \mathcal{O}(n^{d-2}2^n)} \xrightarrow[n\to\infty]{} \frac{1}{2^d-1}\,,$$

and similarly,

$$\frac{N_n - L_n}{L_n} = \frac{n^{d-1}2^{n-d} + \mathcal{O}(n^{d-2}2^n)}{n^{d-1}2^{n-d+1}(2^d - 1) + \mathcal{O}(n^{d-2}2^n)} \xrightarrow[n\to\infty]{} \frac{1}{2(2^d - 1)}\,.$$

♠

Given that the computational complexity grows at least linearly with the number of grid points, Proposition 3 also gives us an estimate of the redundancy when computing the extra grids. For time dependant problems where the time stepping is bounded by the spatial resolution, one may expect the complexity to be superlinear leading to even smaller redundancies.

The second limit of (8) gives the proportion of extra grid points if the grids with indices in $J_n^d \backslash J_{n-d-1}^d$ are computed. Given the previous remarks, if we only expect at most one fault to occur, then computing these grids rather than those in the entire downset is sufficient to obtain a solution and significantly reduces the redundancy.

## 4 Numerical results

We present some numerical results for the solution of the scalar advection equation in 2D which, for $u := u(t, \underline{x})$, is

$$\frac{\partial u}{\partial t} + \boldsymbol{\nabla} \cdot (\underline{a} u) = 0\,. \tag{11}$$

For our results we chose $\underline{a} = (1, 1)$ and considered the solution for $\underline{x} = (x_1, x_2) \in [0, 1)^2$ with periodic boundaries and initial condition $u(0, \underline{x}) = \sin(2\pi x_1)\sin(2\pi x_2)$ . We solved up to $t = 0.5$ for which the exact solution is $u(0.5, \underline{x}) = u(0, \underline{x})$ . Our implementation uses PETSc [1] with centred finite difference discretisation of the advection term and Runge–Kutta for the time

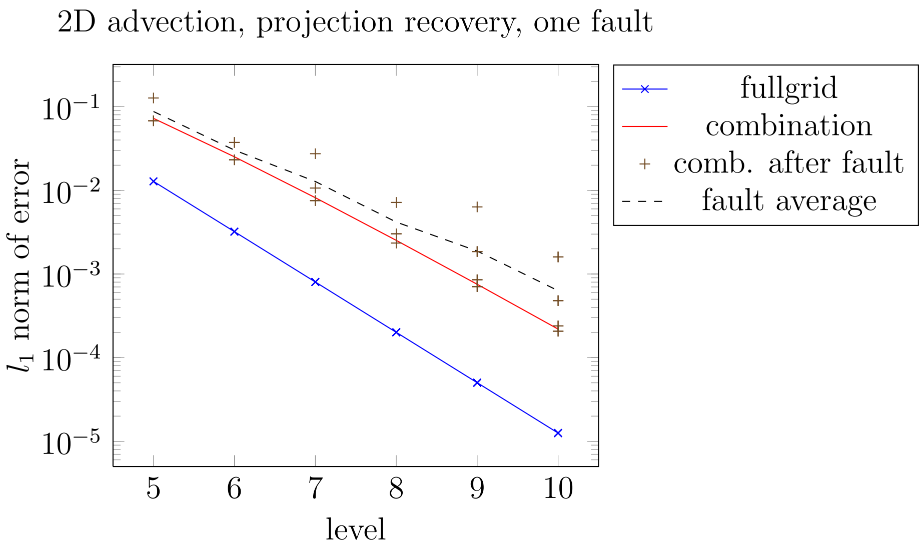

Figure 1: Comparison of errors after recovery when faults affect one of the coarser (grid,solution) pairs. Here we recover by approximating a faulty solution by projecting the solution from a finer grid.

stepping. The *map* and *reduce* functions of Python were used to compute the solution from multiple grids in serial.

In Figures 1 and 2 we demonstrate the additional error when a fault affects one of the coarser grids and is then recovered by projecting the solution from a finer grid. On the horizontal axis is the level $n$ which defines the set of indices $I_n^2$ computed. On the vertical axis is the $l_1$ average error of the solution when interpolated to the full grid $\Omega_{(n,n)}$ . We compare the full grid solution (on grid $\Omega_{(n,n)}$ with $n =$ level for each level) and the combination solution without faults to the combination solution when faults affect random grids. We also plot the average of the errors after recovery from faults. In Figure 1 we simulate recovery after a fault affects one of the (grid,solution) pairs whilst In Figure 2 we simulate the recovery of two pairs. In some cases the error after fault recovery is very close to the solution when no faults occur.

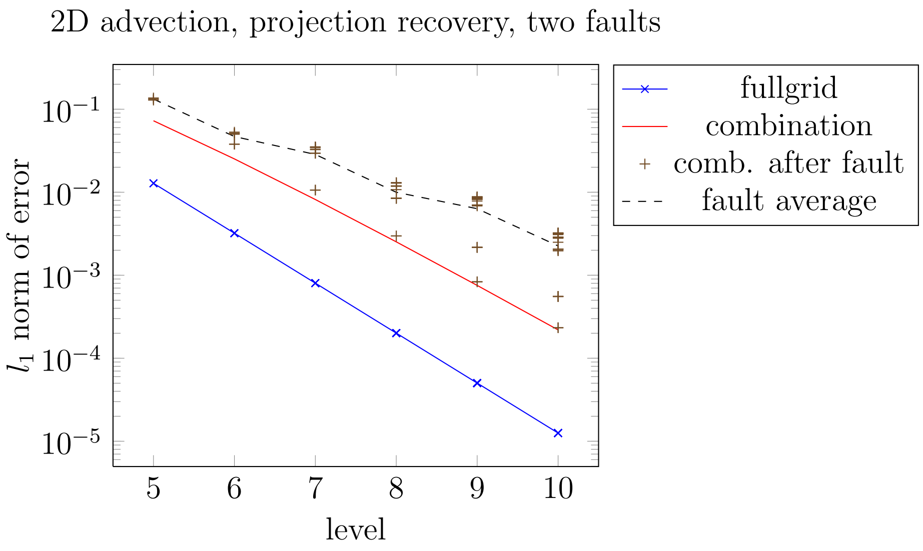

Figure 2: Comparison of errors after recovery when faults affect two of the coarser (grid,solution) pairs. Here we recover by approximating faulty solutions by projecting solutions on finer grids.

Also, the maximal error in the event of faults decreases as the level increases. Whilst the maximal error when faults affect two grids appears close to the case of a single grid failure the average error is much worse.

In Figures 3 and 4 we demonstrate the errors obtained when recovering from failures by obtaining new coefficients from the expansion of the projection operator. The grids with indices in $\mathrm{J}_{\mathrm{n}}^{\mathrm{d}} \backslash \mathrm{J}_{\mathrm{n-d-1}}^{\mathrm{d}}$ were computed and we again compare the full grid solution, combination solution, recovery solutions and average of the recovery solutions. We simulate one and two faults in Figures 3 and 4, respectively. For Figure 4 we assumed that only the grids in $\mathrm{I}_{\mathrm{n}}^{\mathrm{d}}$ are affected by faults since the smaller grids are quicker to compute and are hence much less likely to fail. The errors are much better than those of the previous method, shown in Figures 1 and 2, and the maximal error appears to decrease uniformly with increasing level. Again there are cases where the error after

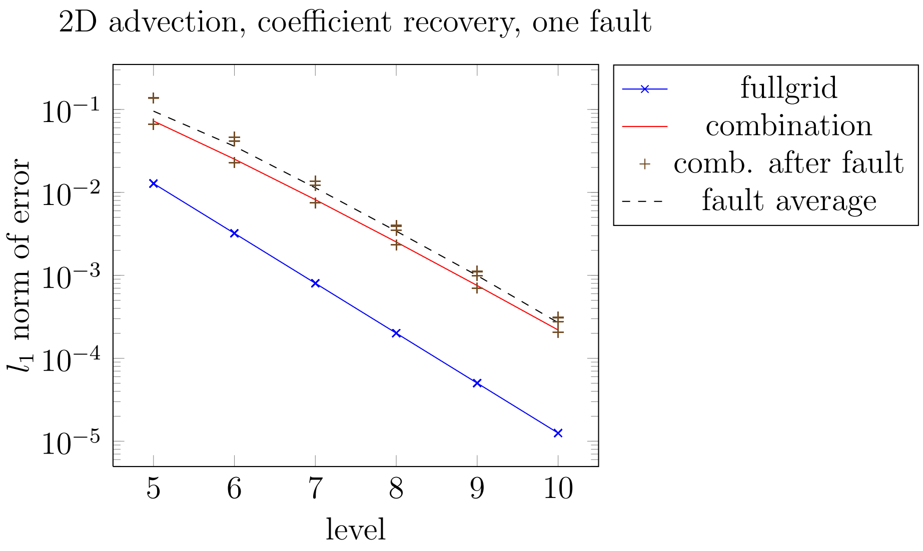

Figure 3: Comparison of errors after recovery when a fault affects one (grid,solution) pair. Here we recover by deriving new combination coefficients which avoid the faulty solution.

a fault occurs is very close to the error when no faults occur. Where faults affect two grids the average error is no more than 25% larger than the average for a single grid failure for levels more than five and the error still appears to decrease uniformly with increasing level.

## 5 Conclusion

It is clear that the first approach, approximating a faulty solution by projecting the solution from a finer grid, is not a suitable solution on its own since only coarser grids can be recovered. Additionally, it gives poor results when considering the error alone for our advection problem. The second approach of deriving new combination coefficients to avoid a faulty solution demon-

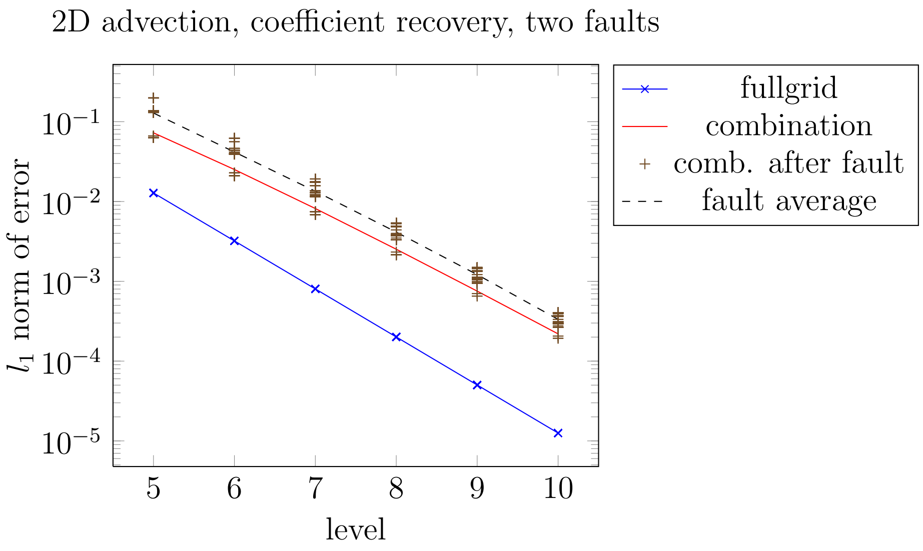

Figure 4: Comparison of errors after recovery when a fault affects two (grid,solution) pairs. Here we recover by deriving new combination coefficients which avoid faulty solutions.

strates much more potential. Despite requiring some additional redundancy the errors after recovery are much better, appearing to have comparable convergence rates to when no faults occur. Future work will include testing these approaches with more problems in up to six spatial dimensions and a detailed analysis of the convergence rates observed in Figures 3 and 4.

**Acknowledgements** This research was supported under the Australian Research Council's *Linkage Projects* funding scheme (project number LP110200410). We are grateful to Fujitsu Laboratories of Europe for providing funding as the collaborative partner in this project.

## References

[1] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page (2012). `http://www.mcs.anl.gov/petsc`. C405

[2] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou. Algorithm-based fault tolerance applied to high performance computing. *Journal of Parallel and Distributed Computing*, 69(4):410–416 (2009). doi:10.1016/j.jpdc.2008.12.002. C396

[3] H. J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269 (2004). doi:10.1017/S0962492904000182. C396, C397, C402

[4] F. Cappello. Fault Tolerance in Petascale/Exascale Systems: Current Knowledge, Challenges and Research Opportunities. *International Journal of High Performance Computing Applications*, 23(3):212–226, (2009). doi:10.1177/1094342009106189. C395

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113 (2008). doi:10.1145/1327452.1327492. C396, C398

[6] J. Garcke. Sparse grids in a nutshell. In J. Garcke and M. Griebel, editors, *Sparse grids and applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 57–80. Springer (2013). doi:10.1007/978-3-642-31703-3_3. C397

[7] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, North Holland (1992). Zbl 0785.65101. C396, C397

[8] M. Hegland. Adaptive sparse grids. *ANZIAM Journal*, 44:C335–C353 (2003). `http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/685`. C397, C400

[9] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, C-33(6):518–528 (1984). doi:10.1109/TC.1984.1676475. C396

[10] C. Zenger. Sparse Grids. In W.Hackbusch, editor, *Parrallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, 1990,* volume 31 of *Notes on Num. Fluid Mech.*, Vieweg–Verlag, 31:241–251 (1991). Zbl 0763.65091. C396

## Author addresses

1. **B. Harding**, Mathematical Sciences Institute, Australian National University, ACT 0200, Australia.
   mailto:brendan.harding@anu.edu.au
2. **M. Hegland**, Mathematical Sciences Institute, Australian National University, ACT 0200, Australia.
   mailto:markus.hegland@anu.edu.au