# Sparse Grid Algorithms to Recover from Hard and Soft Faults

Alfredo Parra Hinojosa
and Hans-Joachim Bungart
Chair of Scientific Computing
Technische Universität München
Munich, Germany
Email: {hinojosa,bungartz}@in.tum.de

Mario Heene
and Dirk Pflüger
Institute of Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
Email: {mario.heene,dirk.pflueger}@ipvs.uni-stuttgart.de

## I. Introduction

Current supercomputers are already vulnerable to various types of hardware errors, which can occur in any part of the system: at node level, in the I/O, disk, or compute network, or even in the service host [1]. Some errors trigger alert signals and the user can react accordingly (*hard errors*). Others are more elusive and give no indication that something went wrong (*soft errors*). This is the case of SDC, which are errors in floating point data that do not cause the running application to crash [2]. Any algorithm that is meant to be run on massively parallel systems should have mechanisms to deal with at least the most common types of errors.

Several classes of algorithms have properties that make them fault tolerant. This means that they can give good numerical solutions even in the presence of system faults. One such algorithm is the *Sparse Grid Combination Technique* [3] (SGCT), an extrapolation scheme particulary useful for high-dimensional problems. The fact that the SGCT has some inherent data redundancy can be exploited to make it tolerant to both hard and soft faults. In other words, the SGCT can recover from process failures as well as data corruption without the need for checkpointing, process replication or any of the typical system-level approaches. In this work we describe two main results: first, that our parallel implementation of the *Fault Tolerant Combination Technique* (FTCT) [4] scales well with simulated hard faults on a large parallel system (*Hazel Hen*). And second, that the FTCT can be extended to deal with *Silent Data Corruption* (SDC), a type of soft fault that is becoming more common as supercomputers grow in size. These and other properties of the SGCT make it a promising algorithm for future exascale systems. Proving this is the goal of our project *EXAHD*, which is part of the German Priority Program *SPPEXA*.

## II. The Sparse Grid Combination Technique

The classical SGCT is shown in Fig. I in two dimensions for a simple linear advection equation. The PDE solution on a full grid with $2^n + 1$ discretization points per dimension (here, $n = 4$) is approximated by solving the PDE on many different coarse, anisotropic grids, which are then combined together according to the formula

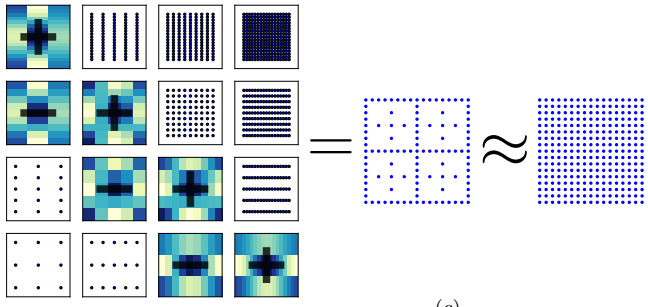


Fig. 1. SGCT in 2D for a linear advection equation.

$$u_{\mathbf{n}} \approx u^{(c)}_{\mathbf{n}} = \sum_{\|\mathbf{i}\|_1 = n+1} u_{\mathbf{i}} - \sum_{\|\mathbf{i}\|_1 = n} u_{\mathbf{i}}, \tag{1}$$

where each $u_{\mathbf{i}}$ is a solution of the PDE on a grid with $(2^{i_1}+1) \times (2^{i_2}+1)$ grid points. We call each $u_{\mathbf{i}}$ a *component solution*. These solutions are considerably less expensive than the full grid solution and they can be computed independently of each other. This offers a second level of parallelism that allows the SGCT to scale on large HPC systems. For functions $u$ whose mixed second derivatives are bounded, the interpolation error of the combination technique is $\mathcal{O}(h_n^2(\log h_n^{-1})^{d-1})$, with $h_n = 2^{-n}$, only slightly larger than on a full grid, $\mathcal{O}(h_n^2)$ [5].

## III. Hard Faults

In practice we want to solve PDEs in 3 to 10 dimensions, which usually means combining hundreds of solutions $u_{\mathbf{i}}$. Our parallelization strategy (Fig. 2) consists in arranging our available processing elements in groups. There is a central *manager process* that assigns to each group a subset of all the component solutions (or *tasks*) to be solved. It also coordinates the stages at which the different solutions should be combined. Each group has a *master process*, which coordinates the work within a group and communicates with the manager process.

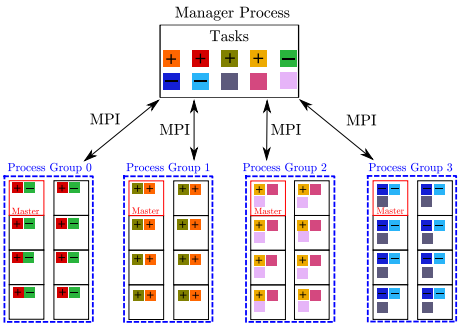


Fig. 2. Parallelization concept for the SGCT

We then simulate processes or nodes crashing using a self-implemented FT-MPI-like layer, since ULFM is not natively installed on the supercomputers we use. We specify at runtime which processes will die, making them go idle and unresponsive in collective operations. We then mark the whole process group as dead, which means that the component solutions assigned to that group go lost. Instead of recomputing them, we combine the surviving solutions with *alternative* weights and exclude from our communicators the process group(s) where the faults occurred. In our example (Fig. I) suppose solutions $u_{(2,3)}$ and $u_{(2,2)}$ go lost. An alternative combination of component solutions could be $u_{(1,4)}+u_{(3,2)}+u_{(4,1)}-u_{(1,2)}-u_{(3,1)}$. This new combination involves only five solutions instead of the original seven: it is not as good as the original, but it is still quite good.

Our experiments on *Hazel Hen* show that our approach has a low overhead (Fig. 3, left). Recovering from faults (redistributing tasks to living groups and, in some cases, recomputing single tasks) costs less than a timestep of the PDE solver. Additionally, the combined solution is only slightly worse when faults occur than without faults - even when a large number of processes fail (Fig. 3, right) [6].

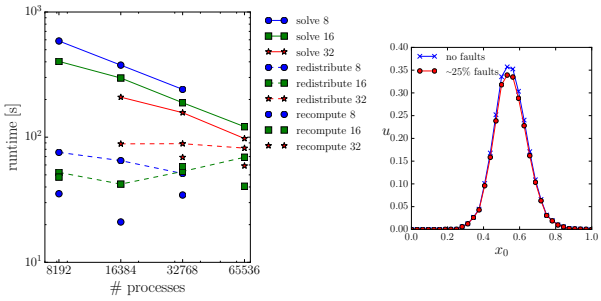


Fig. 3. Scaling results (left) and solution after fault recovery (right).

## IV. Soft Faults

We simulate SDC in our algorithm by altering the values of a random component solution by different orders of magnitude, as proposed in [7]. Depending on when SDC occurs, the error might propagate with time, and if we do not detect wrong component solutions before combining them together, our extrapolated solution $u_{\mathbf{n}}^{(c)}$ might be ruined.

| Pair | Norm |
|---|---|
| (1, 4) (1, 3) | 2.50e-02 |
| **(1, 4) (2, 3)** | **8.71e+04** |
| **(1, 3) (2, 3)** | **3.12e+04** |
| (1, 3) (2, 2) | 5.02e-02 |
| **(2, 3) (2, 2)** | **5.01e+04** |
| (2, 2) (3, 2) | 6.03e-02 |
| (2, 2) (4, 1) | 3.88e-02 |
| (3, 2) (3, 1) | 2.49e-02 |
| (3, 1) (4, 1) | 2.54e-02 |

TABLE I
NORMS FOR DIFFERENT PAIRS OF SOLUTIONS. SOLUTION $u_{(2,3)}$ SEEMS TO HAVE BEEN AFFECTED BY SDC.

We have come up with two approaches to detect SDC [8]. The first idea consists in looking at each of the sparse grid points individually, since for most grid points we have several versions of the solution (from the different component solutions). We know that at a given point all versions of the solution should be nearly constant, so with enough samples we can detect outliers. Although this allows us to identify most errors, it requires substantial additional computations and some tuning of the outlier sensitivity parameters, which might be problem-dependent and thus not optimal. We are currently working on a more general, problem-independent method based on theoretical error bounds.

The second idea is to compute a certain norm for different pairs of component solutions. This norm tells us how different two solutions are from each other. If one of the solutions is wrong, we should be able to detect it by examining the resulting list of norms generated for the different pairs of component solutions (see Table I). We then compare the measured norms with the theoretical error bounds, trying to fir the measured norms to the theoretical model using robust least square regression. Robust fits give us residuals that can be normalized (for example, with respect to a robust median), and large normalized residuals (¿ 2.5) indicate possible outliers. These outlier solutions are removed from the combination, just as we do when hard faults occur. This approach, which will be submitted for reviewing soon, has a small overhead and can be easily parallelized.

## V. Conclusions

The SGCT can be used to recover from hard faults by removing faulty groups and combining the surviving solutions with alternative weights. It can also deal with SDC by exploiting the data redundancy among the different component solutions: if one solution is wrong, it will be different from the others, and this can be detected.

## References

[1] M. Snir, R. W. Wisniewski, J. A. Abraham *et al.*, "Addressing failures in exascale computing," *International Journal of High Performance Computing Applications*, vol. 28, pp. 129–173, 2014.

[2] J. Elliott, M. Hoemmen, and F. Mueller, "Tolerating silent data corruption in opaque preconditioners," *arXiv preprint arXiv:1404.5552*, 2014.

[3] M. Griebel, M. Schneider, and C. Zenger, "A combination technique for the solution of sparse grid problems," in *Iterative Methods in Lin. Alg.*, 1992, pp. 263–281.
[4] B. Harding, M. Hegland, J. Larson, and J. Southern, "Fault tolerant computation with the sparse grid combination technique," *SIAM Journal on Scientific Computing*, vol. 37, no. 3, pp. C331–C353, 2015.
[5] D. Pflüger, *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010.
[6] M. Heene, A. Parra Hinojosa, H.-J. Bungartz, and D. Pflüger, "A massively-parallel, fault-tolerant solver for time-dependent pdes in high dimensions," in *Euro-Par 2016*, Grenoble, Jun. 2016, accepted.
[7] J. Elliott, M. Hoemmen, and F. Mueller, "Evaluating the impact of SDC on the GMRES iterative solver," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 1193–1202.
[8] A. Parra Hinojosa, B. Harding, M. Hegland, and H.-J. Bungartz, "Handling silent data corruption with the sparse grid combination technique," in *Proceedings of SPPEXA Symposium*, ser. Lecture Notes in Computational Science and Engineering. Springer-Verlag, 2016, accepted.