Fault Tolerance in the Parareal Method

[Regular Paper]

Allan S. Nielsen, Jan S. Hesthaven
École Polytechnique Fédérale de Lausanne (EPFL)
Chair of Computational Mathematics and Simulation Science
Bâtiment MA, CH-1015 Lausanne
allan.nielsen@epfl.ch, jan.hesthaven@epfl.ch

ABSTRACT

Parallel-in-time integration is an often advocated approach for extracting parallelism in the solution of PDEs beyond what is possible using spacial domain decomposition techniques. Due to the comparatively low parallel efficiency of parallel-in-time integration techniques, they are primarily of interest as an extension for classical approaches at parallelism. As such, potential applications are expected to scale across several hundreds, or possibly thousands of nodes, making algorithmic resilience towards hardware induced errors highly relevant. In this work we develop a scheduling scheme for the parareal algorithm that is resilient to node-loss. The fault-tolerant scheme is based on a popular approach introduced by E. Aubanel in [1], modified with a set of MPI interface extensions for implementing recovery strategies available in the ULFM framework. In addition, we demonstrate how the parareal algorithm may be made resilient towards Silent-Data-Corruption (SDC) errors by viewing it as a point-iterative method, locally monitoring the residual between consecutive iterations so to discard potentially corrupt iterations.

CCS Concepts

•Computing methodologies → Parallel algorithms; •Applied computing → Physical sciences and engineering; •Computer systems organization → Dependable and fault-tolerant systems and networks;

Keywords

Resilience; Parallel-in-time; Parareal; Exascale; HPC; Fault-tolerance; Silent-Data-Corruption; Parallel Computing

1. INTRODUCTION

Parallel-in-time integration is a promising technique for extracting additional parallelism in the solution of evolution problems beyond what is possible using standard spacial domain decomposition methods. By introducing a decomposition of the time domain it is possible, for certain classes of problems, to greatly increase the number of nodes that may be used to accelerate the solution of a problem of fixed size. A space-time parallel algorithm presented in [14] is able to scale to 458,752 cores on a benchmark problem, the full size of the 5 Petaflop/s JUQUEEN cluster. This more than 3 times the number of nodes the parallel multi-grid solver alone could scale. In a recent report released by the Exascale Mathematics Working Group at Lawrence Livermore National Laboratory, time-parallel integration techniques are highlighted as a potential path in overcoming the limitations of strong scaling in evolution problems and calls for more research in this direction [5]. In this paper we demonstrate how a popular method for time-parallel integration, Parareal, with slight modifications, may be made resilient towards hardware faults. We begin this section with a short introduction to the Parareal method, and to algorithmic resilience. In the two sections that follow, a fault-tolerant Parareal algorithm is developed and tested, while the final section contains a short summary of our findings.

FTXS '16 May 31 – June 4, 2016, Kyoto, Japan
© 2016 ACM. ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

1.1 Time-domain parallelism

Solving time dependent PDEs is often done in a methods-of-line approach where the spatial components are discretized in some appropriate manner and a numerical integration technique is applied to advance in time. The approach extends trivially to distributed memory machines by applying domain decomposition, letting independent nodes communicate boundary information of their local sub-domains. The limitation to the approach lies in the strong scaling limit, i.e. increasing the number of nodes for a fixed problem size. As spacial sub-domains decrease in size, nodes will increasingly spend time on communicating boundary information rather than computing derivatives. With the large machines comprising thousands of nodes available to research today, this is a substantial bottleneck in scaling application efficiently, clearly new algorithmic developments are required.

A potential path to increased parallelism in the solution procedure is to attempt parallel time-integration. Once the methods-of-lines approach have reduced the partial differential equation to a large system of ordinary differential equations that needs to be integrated over time, the problem is traditionally viewed as a sequential process. However, various attempts to extract parallelism do exists. The Parareal method, first proposed in [9], is one such method. The algorithm borrows idea from spatial domain decomposition to construct an iterative approach for solving the temporal problem in a parallel global-in-time approach. It has been applied with success in a range of different applications from plasma physics to many-body problems[17, 15]. To present the method, consider the problem

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + \mathcal{A}(t, \mathbf{u}) = 0 \\ \mathbf{u}(T_0) = \mathbf{u}_0 \quad t \in [T_0, T] \end{cases} \tag{1}$$

where $\mathcal{A} : \mathbb{R} \times V \to V'$ is a general operator depending on $\mathbf{u} : \Omega \times \mathbb{R}^+ \to V$ with V being a Hilbert space and V' its dual. Now, assume there exists a unique solution $\mathbf{u}(t)$ to (1) and decompose the time domain of interest into N individual time slices

$$T_0 < T_1 < \cdots < T_{N-1} < T_N = T. \tag{2}$$

Let $T_n = \Delta T n$ with $n \in \mathbb{N}$. We now define a numerically accurate solution operator $\mathcal{F}_{\Delta T}$ which for any $t > T_0$ advances the solution as

$$\mathcal{F}_{\Delta T}(T_n, \mathbf{u}(T_n)) = \mathbf{U}_{T_n + \Delta T} \approx \mathbf{u}(T_n + \Delta T) \tag{3}$$

To solve (1) on $[T_0, T_0 + N \cdot \Delta T]$ we define $M_{\mathcal{F}}$, $\bar{\mathbf{U}}$ and $\bar{\mathbf{U}}_0$ as operators on the form

$$M_{\mathcal{F}} = \begin{bmatrix} 1 & & & \\ -\mathcal{F}_{\Delta T}^{T_0} & \ddots & & \\ & \ddots & \ddots & \\ & & -\mathcal{F}_{\Delta T}^{T_{N-1}} & 1 \end{bmatrix} \tag{4}$$

with $\bar{\mathrm{U}} = [\mathrm{U}_0, \ldots, \mathrm{U}_N]$ and $\bar{\mathrm{U}}_0 = [u(T_0), 0, \ldots, 0]$. The sequential solution procedure is then equivalent to solving $M_{\mathcal{F}} \bar{\mathbf{U}} = \bar{\mathbf{U}}_0$ for $\bar{\mathbf{U}}$ to recover $\mathbf{U}_0 \cdots \mathbf{U}_N$ as approximations to $\mathbf{u}(T_0) \cdots \mathbf{u}(T_N)$ by forward substitution. The lower bi-diagonal nature of (4) express the explicit and local nature of the approach. If we instead seek to solve the system using a point-iterative approach i.e., we seek the solution on form $\bar{U}^{k+1} = \bar{U}^k + \left(\bar{\mathbf{U}}_0 - M_{\mathcal{F}} \bar{\mathbf{U}}^k\right)$, we observe that at the beginning of each iteration $\bar{\mathbf{U}}^k$ is known, allowing that in each iteration we may compute $\mathcal{F}_{\Delta T}^{T_1} \cdots \mathcal{F}_{\Delta T}^{T_N}$ on all intervals in parallel. Note that the computational complexity of every iteration is strictly larger than that of the sequential solution procedure, so reduced time to solution is possible only if the number of iterations k_{conv} needed for convergence is much smaller than the number of time sub-domains N. A question that remains open is what would be an appropriate preconditioner to accelerate the iteration. A typical approach is to create an approximation $M_{\mathcal{G}} \approx M_{\mathcal{F}}$, where $M_{\mathcal{G}}$ is cheap to apply, hence allowing us to solve the preconditioned system $(M_{\mathcal{G}})^{-1} M_{\mathcal{F}} \bar{\mathbf{U}} = (M_{\mathcal{G}})^{-1} \bar{\mathbf{U}}_0$. In the case considered here, we can readily create such an $M_{\mathcal{G}}$ by defining a new operator $\mathcal{G}_{\Delta T}$ as

$$\mathcal{G}_{\Delta T}(T_n, \mathbf{u}(T_n)) = \mathbf{U}_{T_n + \Delta T} \approx \mathbf{u}(T_n + \Delta T) \tag{5}$$

and relax the requirements on the accuracy of $\mathcal{G}_{\Delta T}$, by using a coarser grid or a different numerical model. Solving the system using a standard preconditioned Richardson iterations, one recovers

$$\bar{\mathbf{U}}^{k+1} = \bar{\mathbf{U}}^k + (M_{\mathcal{G}})^{-1} \left(\bar{\mathbf{U}}_0 - M_{\mathcal{F}} \bar{\mathbf{U}}^k\right) \tag{6}$$

We can write this as

$$\begin{bmatrix} 1 & & & \\ -\mathcal{G}_{\Delta T}^{T_0} & \ddots & & \\ & \ddots & \ddots & \\ & & -\mathcal{G}_{\Delta T}^{T_{N-1}} & 1 \end{bmatrix} \begin{bmatrix} \mathrm{U}_0^{k+1} \\ \mathrm{U}_1^{k+1} \\ \vdots \\ \mathrm{U}_N^{k+1} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mathcal{F}_{\Delta T}^{T_0} - \mathcal{G}_{\Delta T}^{T_0} & \ddots & & \\ & \ddots & \ddots & \\ & & \mathcal{F}_{\Delta T}^{T_{N-1}} - \mathcal{G}_{\Delta T}^{T_{N-1}} & 1 \end{bmatrix} \tag{7}$$

to recover the Parareal algorithm in the form that it is typically presented

$$\mathbf{U}_{n+1}^{k+1} = \mathcal{G}_{\Delta T}^{T_n} \mathbf{U}_n^{k+1} + \mathcal{F}_{\Delta T}^{T_n} \mathbf{U}_n^k - \mathcal{G}_{\Delta T}^{T_n} \mathbf{U}_n^k \tag{8}$$

with $\mathbf{U}_{n+1}^0 = \mathcal{G}_{\Delta T}^{T_n} \mathbf{U}_n^0$ and $\mathbf{U}_0^k = \mathbf{u}(T_0)$. Other parallel-in-time methods may be derived by simply constructing a new preconditioner for the system (4). In this paper, we focus the analysis and implementation on the Parareal algorithm where the preconditioner have the same lower bi-diagonal structure as the matrix $M_{\mathcal{F}}$. However, similar fault-tolerant implementations may be constructed for other fixed-point iteration type time-parallel domain-decomposition methods. A comprehensive introduction to parareal can be found in [13] and important contributions to the analysis of the method can be found in [2, 6]. A recent example of how time-parallel methods may be made resilient to silent data corruption can be found in [8] where it is demonstrated how the Spectral Deferred Correction based Parareal algorithm may be made resilient by introducing a special strategy for monitoring the residual inside the SDC iterations. The SDC time-parallel algorithm introduced in [11] is a special case of the Parareal method that uses Spectral Deferred Correction for constructing the integration operators $\mathcal{F}_{\triangle T}^n$ and $\mathcal{G}_{\triangle T}^n$, while interleaving the iterations within the SDC integrator with Parareal corrections to obtain a higher parallel efficiency

1.2 Resilience

In [5], resilience to faults is identified as being critical for future exascale HPC systems. The techniques needed to achieve a thousand fold increase in computational capacity expected over the next decade, are predicted to also increase the rate of faults on large systems. This posing substantial new challenges in terms of how to effectively use the machines, and on how to assess and assure the correctness of numerical simulation results. In the context of parallel integration techniques, the issue of algorithmic resilience is of particular relevance since methods of parallel integration are developed primarily with the focus of extracting parallelism in the solution of PDE's beyond what is possible using standard domain decomposition techniques, i.e., simulations involving a very large number of compute nodes. An extensive overview of challenges in addressing faults at exascale computing is given in [16]. In the white paper, faults caused by malfunctioning hardware are placed into two overall categories, soft and hard node errors. The most significant source of soft errors are from energetic particles interacting with the silicon subtract that either flip the state of a storage element or disrupt the operation of a combinational logic circuit. Such events typically leads to a silent data corruption (SDC), i.e., no warning or exception is raised when such an event happens. Depending on the location of the SDC, it may lead to an event that over the course of many compute cycles turns into a hard error. Hard errors being faults that lead to the complete failure of a node. For current parallel applications based on MPI, the approach for dealing with the loss of a process is to kill all remaining processes and restart the application at nearest check-point. However, this approach is costly as many modern clusters now scale to thousands of nodes, the I/O cost of a check-point/restart procedure may be prohibitively costly alone. Ideally, a local failure should permit local recovery.

Unlike hard errors, soft errors have the potential to corrupt the simulation result in ways that are not immediately obvious to the application scientist. Despite this, the currently most common approach to SDC resilience is to assume that errors are so rare that they are better ignored, favoring the simple solution of doing a re-run in the odd case of a corrupted solution. This approach raises questions of the trustworthiness of numerical simulations performed, as an SDC type error may not necessarily lead to a result that is obviously wrong in the eyes of the application scientist. Also, it is worth noting that both the cost of an SDC induced re-run and the probability of needing such a re-run scales linearly with the size of the machine, therefore, this simple approach may not be acceptable on future exascale systems. In this paper we seek to develop a variant of the Parareal algorithm for time parallel integration that is resilient to both soft and hard errors. As presented in [7] some parallel integration methods are intimately related, and we therefore conjecture that our ideas may extend to other techniques of time-parallel integration.

2. RECOVERING FROM NODE-LOSS

The Parareal correction (8) may be implemented in different ways. The simplest approach is to divide work into two phases; a purely sequential phase, computing $\mathbf{U}_{n+1}^{k+1}$ from $\mathcal{G}_{\Delta T}^{T_n}\mathbf{U}_n^{k+1}$ with the correction (8), and a parallel phase where $\mathcal{F}_{\Delta T}^{T_n}\mathbf{U}_n^k$ is computed in parallel on $n \in N$ nodes. Ideally, the wall-time $T_\mathcal{G}$ for a node group to compute $\mathcal{G}_{\Delta T}^{T_n}\mathbf{U}_n^{k+1}$ is much smaller than the wall-time $T_\mathcal{F}$ to compute $\mathcal{F}_{\Delta T}^{T_n}\mathbf{U}_n^k$, and the limiting factor in obtainable speed-up will be the number of iterations $k_{conv} < N$ needed for convergence. In practice however, it is seldom possible to construct a coarse operator $\mathcal{G}_{\Delta T}^{T_n}$ so computationally cheap that its cost may be ignored. Fortunately, there exists many other ways for scheduling the computational work than having two strictly separated phases, switching between computing $\mathcal{G}_{\Delta T}^{T_n}\mathbf{U}$ sequentially and $\mathcal{F}_{\Delta T}^{T_n}\mathbf{U}$ in parallel. For example, $\mathcal{G}_{\Delta T}^{T_0}\mathbf{U}_0^0$ and $\mathcal{F}_{\Delta T}^{T_0}\mathbf{U}_0^0$ may be computed concurrently. By exploiting such independencies, it is possible, to some extend, to mitigate the effects of a relatively expensive coarse operator $\mathcal{G}_{\Delta T}^{T_n}$. The most widely cited scheduler for the Parareal algorithm was proposed by Aubanel [1]. The "Fully-Distributed" scheduler is near optimal in exploiting independencies, while at the same time being fairly simple to implement. In Figure 1, the scheduler is schematically visualized for a small problem, $N = 6$ time sub-domains, and convergence in $k_{conv} = 3$ corrections. We note how, as the first time-subdomains converge, the node-groups remain idle while waiting for the application to finish. In this section we will introduce a new variant of the scheduler proposed by Aubanel, that uses features of the UFLM MPI framework [4] to build a fault tolerant algorithm. Here idle node-groups may be used as spares for the event that an active node-group fails.

Algorithm 1 Pseudocode for a Fault Tolerant Version of a "fully-distributed" parareal implementation.

```
 1: $convergeNext \leftarrow$ FALSE
 2: $id_{\Delta T} \leftarrow id_{NG}$
 3: recv_intercomm $\leftarrow$ intercomm $id_{NG} - 1$
 4: send_intercomm $\leftarrow$ intercomm $id_{NG} + 1$
 5: if $id_{\Delta T} = 0$ then
 6:     $\tilde{U}_0^0 \leftarrow y_0$
 7:     $\tilde{U}_1^0 \leftarrow \mathcal{G}_{\Delta T}\tilde{U}_0^0$
 8:     RC $\leftarrow$ send $\tilde{U}_1^0$ on send_intercomm
 9:     check_send(RC,$converge$)
10:     $\hat{U}_1^0 \leftarrow \mathcal{F}_{\Delta T}\tilde{U}_0^0$
11:     $U_1^1 \leftarrow \hat{U}_1^0$
12:     $converge \leftarrow$ TRUE
13:     RC $\leftarrow$ send $converge$ and $U_1^1$ on send_intercomm
14:     check_send(RC,$converge$)
15:     spare \\ $id_{NG} = 0$ completed $id_{\Delta T} = 0$, now spare
16:     exit \\ node-group $id_{NG} = 0$ completed execution
17: else
18:     RC $\leftarrow$receive $\tilde{U}_{id_{\Delta T}}^0$ on recv_intercomm
19:     check_recv(RC,$converge$)
20:     $\tilde{U}_{id_{\Delta T}+1}^0 \leftarrow \mathcal{G}_{\Delta T}\tilde{U}_{id_{\Delta T}}^0$
21:     if $id_{\Delta T}! = N - 1$ then
22:         RC $\leftarrow$send $\tilde{U}_{id_{\Delta T}+1}^0$ on send_intercomm
23:         check_send(RC,$converge$)
24:     end if
25: end if
26: $U_{id_{\Delta T}}^0 \leftarrow \tilde{U}_{id_{\Delta T}}^0$
27: for $k = 1$ to $K_{max}$ do
28:     $\hat{U}_{id_{\Delta T}+1}^{k-1} \leftarrow \mathcal{F}_{\Delta T}U_{id_{\Delta T}}^{k-1}$
29:     if $convergeNext$ then
30:         $converge \leftarrow$ TRUE
31:         $U_{id_{\Delta T}+1}^k \leftarrow \hat{U}_{id_{\Delta T}+1}^{k-1}$
32:         if $id_{\Delta T}! = N - 1$ then
33:             RC $\leftarrow$ send $converge$, $U_{id_{\Delta T}+1}^k$ on send_intercomm
34:             check_send(RC,$converge$)
35:         end if
36:         spare \\ enter spare mode
37:         exit \\ completed succesfully
38:     end if
39:     RC $\leftarrow$receive $converge$ and $U_{id_{\Delta T}}^k$ on recv_intercomm
40:     check_recv(RC,$converge$)
41:     $\tilde{U}_{id_{\Delta T}+1}^k \leftarrow \mathcal{G}_{\Delta T}U_{id_{\Delta T}}^k$
42:     $U_{id_{\Delta T}+1}^k \leftarrow \tilde{U}_{id_{\Delta T}+1}^k + \hat{U}_{id_{\Delta T}+1}^{k-1} + \tilde{U}_{id_{\Delta T}+1}^{k-1}$
43:     if $converge$ & $|U_{id_{\Delta T}+1}^k - U_{id_{\Delta T}+1}^{k-1}| > \epsilon$ then
44:         $converge \leftarrow$ FALSE
45:         $convergeNext \leftarrow$ TRUE \\ converges in $k = k + 1$
46:     end if
47:     if $id_{\Delta T}! = N - 1$ then
48:         RC $\leftarrow$send $converge$, $U_{id_{\Delta T}+1}^k$ on send_intercomm
49:         check_send(RC,$converge$)
50:     end if
51:     if $converge$ then
52:         spare \\ enter spare mode
53:         exit \\ completed succesfully
54:     end if
55: end for
```

Algorithm 2 Pseudo-Code for **spare()** function

```
1: if $id_{\Delta T} = N - 1$ then
2:     $work \leftarrow$ FALSE
3:     procedure: Send $exit$ to all node-groups.
4: else
5:     procedure: Wait for $exit$ or $work$ signal from any node-group $n$.
6: end if
7: if $work$ then
8:     recv_intercomm $\leftarrow$ intercomm $n$
9:     RC $\leftarrow$ receive $k$ and $id_{\Delta T}$ on recv_intercomm
10:    $id_{\Delta T} \leftarrow id_{\Delta T} + 1$
11:    RC $\leftarrow$ receive $converge$ and $U^k_{id_{\Delta T}}$ on recv_intercomm
12:    check_recv(RC,$converge$)
13:    $\tilde{U}^k_{id_{\Delta T}+1} \leftarrow \mathcal{G}_{\Delta T} U^k_{id_{\Delta T}}$
14:    $U^k_{id_{\Delta T}+1} \leftarrow \tilde{U}^k_{id_{\Delta T}+1}$
15:    Revoke and free send_intercomm
16:    if Is $id_{\Delta T} + 1$ being processed on any n.-g.? then
17:        procedure: Find the node-group $n$ processing time-subdomain $id_{\Delta T} + 1$.
18:        send_intercomm $\leftarrow$ intercomm $n$
19:    else
20:        check_send(1,$converge$)
21:    end if
22:    $K \leftarrow k + 1$
23:    for $k = K$ to $K_{max}$ do
24:        procedure: Execute pseudocode Algorithm 1, line 28 to 54.
25:    end for
26: end if
```

2.1 A Fault-Tolerant Scheduler

In Algorithm 1, pseudo code of our proposed Fault-Tolerant Parareal algorithm is presented. The only difference between it and the original "Fully-Distributed" scheme introduced by Aubanal in [1] is the introduction of the function calls check_send, check_recv, and spare. The guideline for our recovery strategy for the fault-tolerant implementation is summarized. In figure 2 the recovery procedure is schematically visualized for a small problem, $N = 6$ time sub-domains, and convergence in $k_{conv} = 3$ corrections with node-groups lost at $\{id_{\Delta T}, k_{err}\} = \{3, 2\}, \{4, 2\}$.

- **Spare Mode.** When $\mathbf{U}^{k+1}_n$ on a time sub-domain handled by a node-group converges, the node-group will become a spare node-group, ready to receive new instructions.

- **Push Strategy.** Recovery upon failure is initiated by *check_send()* . It searches for available spares to continue the work on the time sub-domain that was handled by the node-group that failed. If no spares are available, the application fails globally. If *check_send()* on $id_{\Delta T}$ successfully connects to a spare node-group, it returns a new inter-communicator for sending to the node-group working on $id_{\Delta T} + 1$.

- **Receiving.** A failed *check_recv()* will wait $cT_{\mathcal{G}}$, $c > 1$, for a signal to connect. If no signal appears it initiates global failure. Thus, the loss of a node-group is only recoverable if the loss happens during the computation of $\mathcal{F}^{T_n}_{\Delta T}\mathbf{U}$. If a node-group is in the process of doing the correction (8) when the group fails, the local loss of a node-group will lead to global failure. If *check_recv()* on $id_{\Delta T}$ successfully connects to a spare node-group, it returns a new inter-communicator for receiving from the node-group working on $id_{\Delta T} - 1$ and the converge flag is set zero so that no intervals following a node-group failure converges in the current iteration.

- **Convergence** In the unprotected algorithm, $\mathbf{U}^{k+1}_n$ on the node-group working on the lowest time sub-domain $id_{\Delta T}$ will converge, i.e, during each iteration at least one time sub-domain converges. In the Fault-Tolerant implementation, we require this to be the case as well, and $\mathcal{F}^{T_n}_{\Delta T}\mathbf{U}$ must complete on the node-group with the lowest $id_{\Delta T}$ among active node-groups. This condition is naturally enforced by the wait condition on *check_recv()* and thus simplifies the algorithm while also ensuring that it converges in a maximum of $k_{conv} = N$ iterations as in the unprotected algorithm.

Pseudo code for *check_send*, *check_recv* and *spare*_node is given in Algorithm 2, 3 and 4 respectively. In each algorithm, a number of procedures are outlined. The procedures involve querying node-groups that are actively computing $\mathcal{F}^{T_n}_{\Delta T}\mathbf{U}$ or $\mathcal{G}^{T_n}_{\Delta T}\mathbf{U}$ for information on their current status. That is, retrieving the local values of k and $id_{\Delta T}$ on node-groups without explicit synchronization. The ideal way of doing so is to use RMA features introduced in the MPI 3.0 standard. However, for our test implementation, this was not possible since ULFM-1.1 in its current implementation is based on OpenMPI 1.7. Instead, we chose a solution where all node-groups spawn two Pthreads in the beginning of the application. One for doing the work as outlined in Algorithm 1, another solely for handling signals and returning

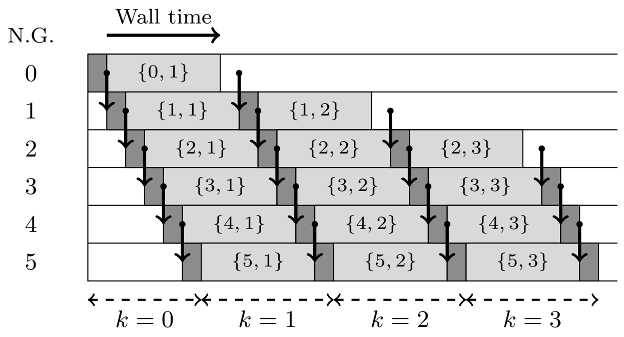


Figure 1: The "Fully-Distributed" work scheduling of the parareal algorithm as proposed in [1], light grey indicate a node-group computing $\mathcal{F}^{T_n}_{\Delta T}\mathbf{U}$, dark gray $\mathcal{G}^{T_n}_{\Delta T}\mathbf{U}$.

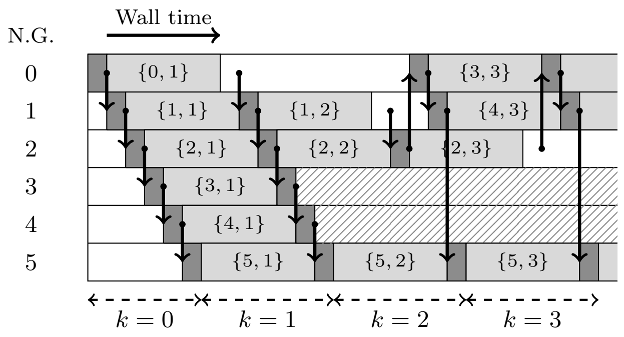


Figure 2: Schematic visualization of the recovery procedure of the Fault-Tolerant algorithm with $N = 6, k_{conv} = 3$ and failed node-groups at $\{id_{\Delta T}, k_{err}\} = \{3, 2\}, \{4, 2\}$.

Algorithm 3 Pseudocode for **check_send**(RC,*converge*)

```
1: if RC != 0 then
2:   Revoke and free send_intercomm
3:   if converge then
4:     converge ← FALSE
5:     $id_{\Delta T} \leftarrow id_{\Delta T} + 1$
6:     $\tilde{U}^k_{id_{\Delta T}+1} \leftarrow \mathcal{G}_{\Delta T} U^k_{id_{\Delta T}}$
7:     $U^k_{id_{\Delta T}+1} \leftarrow \tilde{U}^k_{id_{\Delta T}+1}$
8:     if Is $id_{\Delta T} + 1$ being processed on any n.-g.? then
9:       procedure: Find the node-hroup $n$ processing
         time-subdomain $id_{\Delta T} + 1$.
10:      send_intercomm ← intercomm $n$
11:    else
12:      check_send(1,converge)
13:    end if
14:    RC ←send converge, $U^k_{id_{\Delta T}+1}$ on send_intercomm
15:    check_send(RC,converge)
16:    $k \leftarrow k + 1$
17:    $\hat{U}^{k-1}_{id_{\Delta T}+1} \leftarrow \mathcal{F}_{\Delta T} U^{k-1}_{id_{\Delta T}}$
18:    $U^k_{id_{\Delta T}+1} \leftarrow \hat{U}^{k-1}_{id_{\Delta T}+1}$
19:    converge ← TRUE
20:    RC ← send converge, $U^k_{id_{\Delta T}+1}$ on send_intercomm
21:    check_send(RC,converge)
22:  else
23:    if Any node-group in spare-mode? then
24:      procedure: Find node-group $n$ in spare-mode
         and send work signal.
25:      send_intercomm ← intercomm $n$
26:      RC ← send $k$, $id_{\Delta T}$ on send_intercomm
27:      RC ← send converge, $U^k_{id_{\Delta T}+1}$ on send_intercomm
28:      check_send(RC,converge)
29:    else
30:      procedure: Send exit to all node-groups.
31:      exit \\ application failure
32:    end if
33:  end if
34: end if
```

Algorithm 4 Pseudocode for **check_recv**() function

```
1: if RC != 0 then
2:   procedure: Wait for exit or work signal from any
     node-group $n$. If wait-time exceeds $\Delta T_{\mathcal{G}}$, assume fail-
     ure and abort program.
3:   if work then
4:     recv_intercomm ← intercomm $n$
5:     RC ←receive converge and $U^k_{id_{\Delta T}}$ on
       recv_intercomm
6:     check_recv(RC,converge)
7:   end if
8: end if
```

information on the local node-group's current k and $id_{\Delta T}$ that may be requested by other node-groups. The Fault-Tolerant version must create $\frac{N}{2}(N-1)$ inter-communicators between N intra-communicators at the onset of the algorithm. For the unprotected algorithm even the loss of a single node within a node-group will lead to global failure. No rearranging will ever be needed, and creating $N-1$ inter-communicators between the intra-communicators of N adjacent node-groups is thus sufficient. It is not an option to simply create the new inter-communicator during the recovery process since this would require a global synchronization to shrink the global communicator so to create a new inter-communicator. In addition to the cost associated with creating the $\left(\frac{N}{2}-1\right)(N-1)$ extra inter-communicators, the Fault-Tolerant algorithm performs a check after each receive and send operation on the intercommunicators. A check must involve an agreement operation across the local intra-communicator so to ensure that all send/recv completed successfully. In the section that follow, we present a small numerical experiment to examine the cost associated with the added operations.

2.2 Numerical Experiments

For testing purposes, the proposed Fault-Tolerant variant and the unprotected algorithm are wrapped around the parallel-in-time integration of a simple wave-problem ODE system, $\frac{d}{dt}\mathbf{u} = \Lambda\mathbf{u}$ using an implicit Euler integration scheme on the interval $T = [0, 10]$ with $\mathbf{u}_0 = [1, \ldots, 1]$, Λ being a complex valued diagonal matrix, the dimension of which is given by the number of ranks in space. The computational complexity of this type of problem is very light, so the actual ratio $\frac{T_{\mathcal{F}}}{T_{\mathcal{G}}}$ is controlled by a sleep function rather than the compute capacity of the node. This approach allows for k_{conv}, $r = \frac{T_{\mathcal{F}}}{T_{\mathcal{G}}}$ and the number of time sub-domains N to be controlled independently of each other. This mimicking any possible problem, while at the same time accurately measuring the associated costs of creating a large number of inter-communicators and performing agreement operations on the send/recv operations on inter-communicators between time sub-domains. In Figure 3, measurements for a problem with $N = 16$ time sub-domains and a ratio $r = 16$ with $T_{\mathcal{G}} = 2s$ and $\mathcal{G}^{T_n}_{\Delta T}$ fine enough that $\left|\mathbf{U}^{k+1}_n - \mathbf{U}^k_n\right| < \epsilon$ after $k_{conv} = 3$ corrections is presented for multiple different error scenarios. Comparing case (b) and (c), we note that the cost associated with a recovery operation is fairly small, and that the cost due to loss of information, possibly forcing the algorithm to make another iteration or two before converging, is an order or two higher. Likewise, the initial added cost of setting up the inter-communicators and threads for handling signaling is comparatively small.

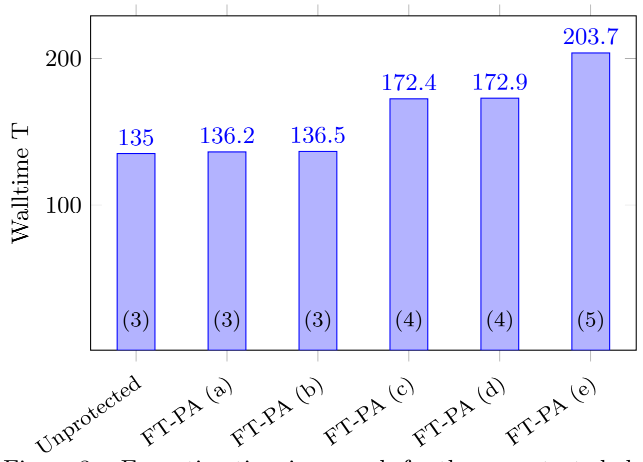


Figure 3: Execution time in seconds for the unprotected algorithm, and for the proposed fault-tolerant algorithm with one or multiple node-group losses located at $\{id_{\Delta T}, k_{err}\}$. **(a)** No errors **(b)** $\{15, 1\}$ **(c)** $\{15, 3\}$ **(d)** $\{4, 1\}, \{5, 2\}$ **(e)** $\{11, 2\}, \{12, 2\}, \{15, 3\}$. ; The number in parenthesis indicate iterations to convergence.

2.3 Failure Analysis

The proposed Fault-Tolerant variant of the Fully- Distributed Parareal work scheduling algorithm may fail to recover when subjected to node-group losses under certain circumstances. As outlined in Section 2.1, there is a limit on how many node-groups that may be lost at a given iteration, as well as the limitation that all correction operations (8), and computation of $\mathcal{G}_{\Delta T}^{T_n}\mathbf{U}$ must not fail. In this section we derive a lower bound on the probability that the Fault-Tolerant algorithm will execute successfully. A key assumption in our derivation is that the occurrence of node-losses may be assumed to be a Poisson point process, and that on the cluster running the algorithm, statistics on the average time between node failure is available. Let $\mu_{\Delta T}^{\mathcal{G}}$ be the average number of times on a time interval $T_{\mathcal{G}}$ that any node within a node-group will fail, and assume that $\mu_{\Delta T}^{\mathcal{G}} \ll 1$. Since the zeroth iteration consists solely of the computation of N, $\mathcal{G}_{\Delta T}^{T_n}\mathbf{U}$, that all must survive, the probability of successfully executing the zeroth iteration is equal the probability of zero node-losses occurring

$$\mathcal{P}_0^{N,0} = e^{-N\cdot\mu_{\Delta T}^{\mathcal{G}}} \tag{9}$$

For the iterations to follow, the derivation is less trivial. Due to our "push-strategy" for recovery, all subsequent iterations k must have zero node-losses during the correction phase, the probability of which is given by $\exp\left((k-N)\cdot\mu_{\Delta T}^{\mathcal{G}}\right)$. During the computation of $\mathcal{F}_{\Delta T}^{T_n}\mathbf{U}$, several node-groups may be lost whilst still being recoverable. The probability that n node-groups are lost at iteration k may be expressed as $\frac{\left(r\cdot(N-k)\cdot\mu_{\Delta T}^{\mathcal{G}}\right)^n}{n!}\exp\left(r\cdot(k-N)\cdot\mu_{\Delta T}^{\mathcal{G}}\right)$ where $r = \frac{T_{\mathcal{F}}}{T_{\mathcal{G}}}$. Finally, due to our requirement that the algorithm must converge in a maximum of $k = N$ iterations, $\mathcal{F}_{\Delta T}^{T_n}\mathbf{U}$ on the first active time sub-domain in any iteration k must execute successfully, the probability of which is $\exp\left(-r\cdot\mu_{\Delta T}^{\mathcal{G}}\right)$, independent of k. The probability that the algorithm for N time sub-domains successfully completes iteration k with n node-group loses is then given by

$$\mathcal{P}_n^{N,k} = \frac{\left(r\left(N-k\right)\mu_{\Delta T}^{\mathcal{G}}\right)^n}{n!}e^{(1+r)(k-N)\mu_{\Delta T}^{\mathcal{G}}-r\mu_{\Delta T}^{\mathcal{G}}} \tag{10}$$

Using the above expression, we may write the probability that the unprotected algorithm executes successfully as

$$P_{\text{PA}}^{N,k_c} = \prod_{i=0}^{k_c}\mathcal{P}_0^{N,i} \tag{11}$$

For the unprotected algorithm, each iteration must be completed with $n = 0$ node-group losses. In the case of the Fault-Tolerant algorithm, at any iteration k, the fault-tolerant algorithm may recover from up to $l_1\left(N,k,n_p\right) = \min[k-n_p, N-k]$ node-group failures, n_p being the sum of node-group failures in previous iterations $1\ldots k-1$. The limitation is due to the need for a spare node-group to be available at node-loss. When a node-group is lost, depending on the location and iteration, it may or may not lead to the need for an added iteration before convergence. For the purpose of deriving a bound on the probability of the fault tolerant algorithm executing successfully, we assume worst case scenario, that the loss of a node-group will always lead to an added iteration, up until the limit that convergence will happen in no less than N iterations. Hence, we define yet another limiter $l_2\left(N,k_c,n_p\right) = \min\left(k_c+n_p, N\right)$, this on the number of iterations needed for convergence.

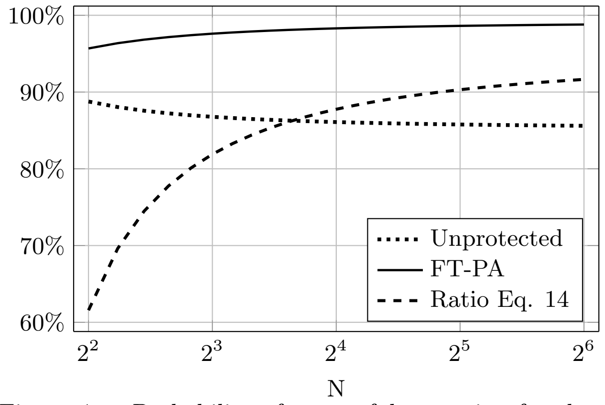


Figure 4: Probability of successful execution for the unprotected and the Fault-Tolerant algorithm. $N = 16$, $r = 32$, $k_{conv} = 2$ and failure rate $\mu_{\Delta T}^{\mathcal{G}} = 0.0001$. The dashed line indicate proportion of failures of the unprotected algorithm that the Fault-Tolerant version may recover from.

For any given problem, a lower bound on the probability of successful execution may be computed as the sum of the products of each possible path to success. To compute the possible paths for problems with a large number of time sub-domains, we define a recursive choice function as

$$\Phi_{n_p}^{N,k} = \begin{cases}\sum_{i=0}^{l_1\left(N,k,n_p\right)}\mathcal{P}_i^{N,k}\Phi_{n_p+i}^{N,k+1} & \text{if } k \le l_2\left(N,k_c,n_p\right)\\ 1 & \text{otherwise}\end{cases} \tag{12}$$

The lower bound on the probability that the fault tolerant algorithm will execute successfully may then be written as

$$P_{\text{FT-PA}}^{N,k_c} \ge \mathcal{P}_0^{N,0}\Phi_0^{N,1} \tag{13}$$

In Figure 4, the probability of successful execution for the unprotected and for the Fault-Tolerant, Algorithm 1, is presented for a problem $N = 16$, $r = 32$ and $k_{conv} = 3$, with on average one node-loss within a node-group per 10000 time-intervals $T_{\mathcal{G}}$, i.e., $\mu_{\Delta T}^{\mathcal{G}} = 0.0001$. In addition, the figure contains a plot of the percentage of failures of the unprotected algorithm that is successfully executed by the Fault-Tolerant algorithm. We denote this ratio R

$$R = \frac{P_{\text{FT-PA}}^{N,k_c} - P_{\text{PA}}^{N,k_c}}{1-P_{\text{PA}}^{N,k_c}} \tag{14}$$

3. GUARDING AGAINST SDC ERRORS

In the previous section we approached the issue of node failures. We now consider the other major source of faults in HPC applications, silent errors in the form of silent data corruption. When subjected to this type of error, an application may provide an incorrect output without any indication that the application has malfunctioned. Algorithmic resilience towards SDC type errors is an active area of research, and [8] provides a recent example in the context of time integration. In their paper the authors demonstrate how spectral deferred correction for solving ODE's may be made resilient to SDC type errors and in [3], an auxiliary checking scheme is introduced to form a fairly generic approach to silent error detection in numerical time-stepping schemes. In this work we extend the Parareal algorithm to make SDC resilience an integral part of the algorithm, regardless of the SDC resilience properties of the underlying operators $\mathcal{F}_{\triangle T}$ and $\mathcal{G}_{\triangle T}$.

Numerical algorithms have traditionally been developed under the assumption that all underlying algebraic operations are carried out accurately, subject only to the limitation of machine accuracy. In the following work we stray away from this assumption, i.e., if a matrix vector product results in x, then there is a non-zero probability of computing $x + \tilde{x}$, with $\tilde{x}$ being a random variable. In the field of numerical analysis, a main focus is on the analysis of error and convergence, but since our error is now a random variable, how should we approach the analysis? A natural idea is to define convergence in terms of the statistical moments of the error, indeed this is the idea being presented in [19], where they consider a method to be convergent with respect to hardware error, if for every $\epsilon > 0$, a finite amount of work will make $E\left[e^h\right] < \epsilon$ and $Var\left[e^h\right] < \epsilon^2$.

3.1 SDC resilient Parareal

In building an SDC resilient algorithm for iterative methods, it is natural to look at the difference between consecutive iterations to detect whether or not an SDC-type error occurred. This is the approach taken by Stoyanov and Webster in [19], where a generic approach for making fixed-point iterative methods resilient towards SDC-type errors is proposed. In their approach, they argue that if the iteration matrix is a contraction, the norm of the difference between successive iterates should reduce at the same rate as the rate of convergence of the algorithm, thus rejecting iterates if they fail to do so. As presented in the introduction, Parareal is in essence also a fixed point iteration, but with a non-normal iteration matrix, the elements of which are potentially non-linear operators. Due to the non-normal structure of the iteration matrix, it is not possible to provide a general guarantee that the iteration matrix will be a contraction. However, since the upper bound on parallel efficiency of the algorithm scales as $1/k_{conv}$, we find it reasonable to assume that for any practical application, $\mathcal{G}_{\Delta T}^{T_n}$ is constructed sufficiently close to $\mathcal{F}_{\Delta T}^{T_n}$ so that the iteration matrix will remain a contraction on $\mathbf{U}_n^k$ from $k = 0$ and onwards. Hence, the approach of [19] is directly applicable for making the Parareal method SDC resilient. However, this approach is limited by the fact that $\bar{\mathbf{U}}^k$ is needed, imposing the need for a synchronization stage between consecutive iterations. This limits the possible scheduling of work to a slow manager-worker type model. As previously discussed in Section 2.1, such a model is not used in practice as the limitations it imposes on obtainable speed-up are too severe. Fortunately, due to the special structure of the iteration matrix (6), we may construct a local approach without the need for a synchronization between iterations. First, define the residual between two consecutive iterations on the node-group local time sub-domain n as

$$e_n^{k+1} = \left\| \mathbf{U}_n^{k+1} - \mathbf{U}_n^k \right\|_\infty \quad (15)$$

For an SDC resilient model, the above e_n^{k+1} must be computed at iteration $k+1$ on each time sub-domain n, and communicated along with *converge*, see Algorithm 1, so that the node-group responsible for the n'th time sub-domain at the $k+1$'th iteration can access $e_i^{k+1} \forall i \in 1 \ldots n$. Then, if at any iteration for any time sub-domain

$$\max_{i=0\ldots n} e_n^{k+1} \geq \beta \max_{i=0\ldots n} e_n^k \quad (16)$$

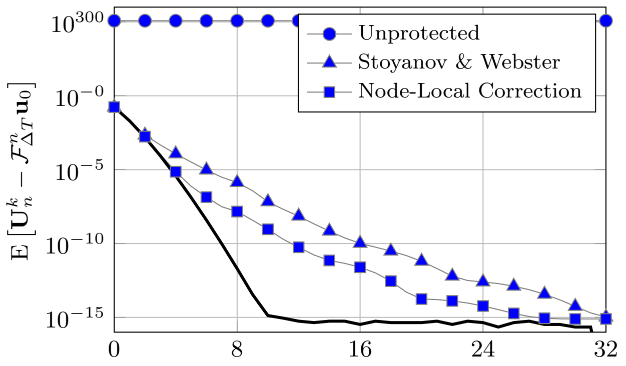
is true, we reject $\mathbf{U}_n^{k+1}$ and replace it with

$$\mathbf{U}_n^{k+1} = \mathbf{U}_n^{k-1}, \quad e_n^{k+1} = e_n^{k-1} \quad (17)$$

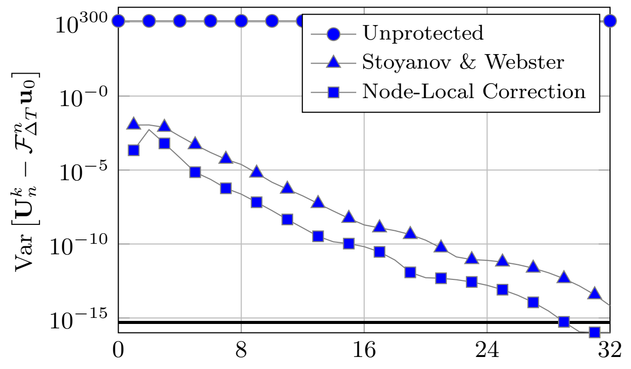
where $\beta \leq 1$ is an upper bound to the contraction factor. If no upper bound is available, using $\beta = 1$ appears to work well. To avoid stagnation due to false rejection, we reject the previous two local iterates. In [19] other approaches for guarding against false rejections are discussed. These approaches only discard a single iterate, but need tunable parameters, or estimates on the Parareal iteration matrix that may not be available in general. In our experience the above approach appears to be near-optimal for avoiding stagnation, while at the same time being parameter-free and easy to implement.

3.2 Numerical Experiments

For the numerical experiments, a strategy to introduce data corruption during the solution process is needed. Various studies have attempted to quantify the rate of soft errors leading to SDC's on clusters. Despite a sizable amount of research into this topic, a consensus seems yet to have been established. A reasonable estimate appears to be that the frequency of SDC type errors is roughly an order of magnitude lower than that of errors leading to node failure [10, 18]. On modern day clusters, DRAM memory and CPU caches are almost always protected at the architectural level using some form of error correction. The major source of faults leading to SDCs are therefore suspected to be caused by radiation strikes that effect logic elements in the CPU or GPU. It is not trivial to deduce how a fault in a logic units



(a)



(b)

Figure 5: Convergence rate when solution procedure is subjected to silent-data corruption. The unmarked line indicate the convergence rate for the error-free solution procedure.

will effect the output of the operation, or the statistical nature of the outputs. In [20], a quantitative comparison between the accuracy of direct fault-injection at the assembly code level with that of fault injection in high level code is presented. They demonstrate that faults leading to SDC type errors are well approximated by high level injection of single bit flips. We proceed with this error model for our test-case. At each time-step, within both operators $\mathcal{G}_{\Delta T}^{T_n}$ and $\mathcal{F}_{\Delta T}^{T_n}$, every element in the state vector $\mathbf{U}_n^k$ will be subject to a bit-flip with probability P at a random location in the 64bit wide double. As a test-case for SDC-type resilience, we use the time-parallel integration over a wave-period of the 1D advection-diffusion equation with periodic boundaries and advection-diffusion coefficients $\alpha = 1, \kappa = 0.01$. $\mathcal{F}_{\Delta T}^{T_n}$ is constructed as in [12] using a 4th order compact finite-difference stencil for discretizing the spacial derivatives and $\mathcal{C}^1$-spline collocation for solving the linear system of ODEs, and $\mathcal{G}_{\Delta T}^{T_n}$ as with first order finite difference approximations in space and time. We test the solution procedure with a high rate of errors $P = 10^{-6}$, and measure the mean and variance as a function of iterations averaged over 1000 realizations. We present the results in Figure 5. Clearly, the proposed node-local correction strategy is not only preferable in the sense that it is generally applicable regardless of the work-distribution model used. As an added bonus, it also converges faster than the approach proposed by Stoyanov and Webster, this due to the fact that less information is discarded upon rejection.

4. SUMMARY

Time-domain parallelism is receiving increasing attention as a viable way to extend the limits of strong scaling in solving evolution-type PDE problems, and offer a potential path to scaling at exascale. We have demonstrated how a novel method of time-domain parallelism, the Parareal method, may be made resilient towards hard errors in the form of node-losses when a fault-tolerant supporting API for distributed memory computing such as ULFM is used. In addition, we have shown that due to the special structure of the iteration matrix, it is possible to monitor the residual between consecutive iterations locally. This leading to an SDC resilient correction strategy that may be applied regardless of the work distribution model used.

5. REFERENCES

[1] E. Aubanel. Scheduling of tasks in the parareal algorithm. *Parallel Computing*, 37:172–182, 2011.

[2] G. Bal. On the convergence and the stability of the parareal algorithm to solve partial differential equations. In *Domain decomposition methods in science and engineering*, pages 425–432. Springer Berlin Heidelberg, 2005.

[3] A. R. Benson, S. Schmit, and R. Schreiber. Silent error detection in numerical time-stepping schemes. *IJHPCA*, 2014.

[4] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. An evaluation of user-level failure mitigation support in mpi. *Computing*, 95(12):1171–1184, 2013.

[5] J. Dongarra and et al. Applied mathematics research for exascale computing. Technical Report No. LLNL-TR-651000, Lawrence Livermore National Laboratory (LLNL), Livermore, CA„ 2014.

[6] M. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM Journal on Scientific Computing*, 29(2):556–578, 2007.

[7] M. J. Gander. 50 years of time parallel time integration. In *Householder Symposium XIX June 8-13, Spa Belgium*, page 81, 2015.

[8] R. Grout, H. Kolla, M. Minion, and J. Bell. Achieving algorithmic resilience for temporal integration through spectral deferred corrections. *arXiv preprint arXiv:1504.01329*, 2015.

[9] J.-L. Lions, Y. Maday, and G. Turinici. Résolution d'edp par un schéma en temps pararéel. *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7):661–668, 2001.

[10] S. E. Michalak, A. J. DuBois, C. B. Storlie, H. M. Quinn, W. N. Rust, D. H. DuBois, D. G. Modl, A. Manuzzato, and S. P. Blanchard. Assessment of the impact of cosmic-ray-induced neutrons on hardware in the roadrunner supercomputer. *Device and Materials Reliability, IEEE Trans.*, 12(2):445–454, 2012.

[11] M. Minion. A hybrid parareal spectral deferred corrections method. *COMCoS*, 5(2):265–301, 2011.

[12] A. Mohebbi and M. Dehghan. High-order compact solution of the one-dimensional heat and advection–diffusion equations. *Applied Mathematical Modelling*, 34(10):3071–3084, 2010.

[13] A. Nielsen. Feasibility study of the parareal algorithm. Diss. msc thesis, Technical University of Denmark, 2012. IMM-134.

[14] D. Ruprecht, R. Speck, M. Emmett, M. Bolten, and R. Krause. Extreme-scale space-time parallelism.

[15] D. Samaddar, D. E. Newman, and R. Sánchez. Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. *Journal of Computational Physics*, 229(18):6558–6573, 2010.

[16] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, et al. Addressing failures in exascale computing. *IJHPCA*, 2014.

[17] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, and P. Gibbon. A massively space-time parallel n-body solver. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 92. IEEE Computer Society Press, 2012.

[18] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. Feng shui of supercomputer memory positional effects in dram and sram faults. In *International Conference for HPC, Networking, Storage and Analysis (SC)*. IEEE, 2013.

[19] M. Stoyanov and C. Webster. Numerical analysis of fixed point algorithms in the presence of hardware faults. Technical Report TM-2013/283, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN, 2013.

[20] J. Wei, A. Thomas, G. Li, and K. Pattabiraman. Quantifying the accuracy of high-level fault injection techniques for hardware faults. In *Dependable Systems and Networks, 44th Annual IEEE/IFIP Conference*, pages 375–382. IEEE, 2014.