Parallel Computing 000 (2017) 1–12

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner

F. Rizzi[a,*], K. Morris[a], K. Sargsyan[a], P. Mycek[b], C. Safta[a], O. Le Maître[c], O.M. Knio[b,d], B.J. Debusschere[a]

[a] Sandia National Labs, Livermore, CA, USA
[b] Duke University, Durham, NC, USA
[c] LIMSI, Orsay, France
[d] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

ARTICLE INFO

Article history:
Received 29 June 2016
Revised 12 April 2017
Accepted 20 May 2017
Available online xxx

Keywords:
Resiliency
Server-client programming model
Dynamic voltage/frequency scaling
PDE
Domain-decomposition
Silent data corruption

ABSTRACT

We discuss algorithm-based resilience to silent data corruptions (SDCs) in a task-based domain-decomposition preconditioner for partial differential equations (PDEs). The algorithm exploits a reformulation of the PDE as a sampling problem, followed by a solution update through data manipulation that is resilient to SDCs. The implementation is based on a server-client model where all state information is held by the servers, while clients are designed solely as computational units. Scalability tests run up to $\sim 51K$ cores show a parallel efficiency greater than 90%. We use a 2D elliptic PDE and a fault model based on random single and double bit-flip to demonstrate the resilience of the application to synthetically injected SDC. We discuss two fault scenarios: one based on the corruption of all data of a target task, and the other involving the corruption of a single data point. We show that for our application, given the test problem considered, a four-fold increase in the number of faults only yields a 2% change in the overhead to overcome their presence, from 7% to 9%. We then discuss potential savings in energy consumption via dynamic voltage/frequency scaling, and its interplay with fault-rates, and application overhead.

1. Introduction

The evolution of computing platforms towards exascale presents key challenges related to resiliency, power, memory access, concurrency and heterogeneous hardware [1–5]. There is no consensus on what a "typical" exascale architecture might look like [2]. One of the main concerns is understanding how hardware will affect future computing systems in terms of reliability, energy consumption, communication and computational models.

The main constraint to making exascale computing a reality is energy consumption [4]. The current target is to build an exascale machine consuming $\sim 20MW$ by 2020. Significant technological advances are required to make this objective feasible, since current systems cannot be simply scaled up to reach this goal. These advancements need to span different hardware aspects, ranging from underlying circuits, to power delivery as well as cooling technologies. Hardware-oriented research should be complemented by cross-cutting efforts tackling energy efficiency at the algorithm and programming

* Corresponding author.
E-mail address: fnrizzi@sandia.gov (F. Rizzi).

http://dx.doi.org/10.1016/j.parco.2017.05.005
0167-8191/

model level. There is consensus that a coordination of efforts is required between advances in programming systems and the development of hardware to enable applications to run efficiently and correctly on exascale machines [1,3].

Exascale simulations are expected to rely not only on thousands of CPU cores running up to a billion threads, but also on extensive use of accelerators, e.g. Graphics Processing Units (GPUs) [1,3,5]. This framework will necessarily lead to systems with a large number of components. The presence of many components, the variable operational modes (e.g. lower voltage to address energy requirements) and the increasing complexity of these systems (e.g. more and smaller transistors) can become a liability in terms of system faults. Exascale systems are expected to suffer from errors and faults more frequently than the current petascale systems [5,6]. Current parallel programming models and applications will require a resilient infrastructure to be suitable for fault-free simulations across many cores for reasonable amounts of time. It will become increasingly more important to develop resilient-aware applications for exascale, where fault-tolerance is explored and quantified to assess whether or not they can tolerate expected failure rates.

Energy and resilience are tightly linked challenges. For instance, high resilience could be achieved through extensive hardware redundancy, but this approach would yield a large power overhead, e.g. three times more expensive for triple-redundancy. Checkpointing is currently the approach most widely used to recover from faults, but it is expected to become unfeasible for exascale applications given the higher failure rates [3,5]. To address resilience without an excess power and/or performance costs will require innovations and coordinated efforts across all system levels. At the application level, one approach would be to design applications such that they are structured into stages having different resilience requirements. This would allow one to isolate those data and computational units requiring resilience from other data and work units where resilience is less needed.

This work presents a new task-based resilient domain-decomposition partial differential equation (PDE) preconditioner implemented with a server-client programing model. The problem is reformulated such that the PDE solver is reduced to a number of independent tasks to benefit concurrency and parallelism. The algorithm enables the application to be resilient to silent data corruption (SDC), while the server-client model (SCM) enables resiliency to hard faults. Our implementation uses the *User Level Fault Mitigation* MPI (MPI-ULFM) [7], a fault tolerance capability proposed for the MPI standard that enables a fault-tolerant MPI framework. In this work, however, we don't focus on hard faults, whose analysis will be the subject of a separate study, but limit our attention to SDCs. Our application can be seen as a preconditioner that will enable today's solvers to be used effectively on future architectures by operating on subdomain levels. Scalability tests run up to $\sim 51K$ cores show a parallel efficiency greater than 90%.

The server-client programming model provides a task-based application with an infrastructure that can potentially address some of the concerns related to energy consumption and resiliency. The work we present here assumes a SCM running on a machine with different capacity cores assigned to servers and clients. The idea pushed forward is that high-end high-capacity/voltage/reliability nodes are reserved for the servers which hold all the state information of the application, while lower-voltage higher-fault-rate components are used for clients which are in charge of the computation. This separation of data and computation enables the overall reduction of energy consumption for large scale machines, provided that the number of nodes hosting the servers is negligible compared to that hosting the clients, and the overhead associated with clients with higher fault rates is sufficiently small.

The paper is organized as follows. In Section 2, we describe the mathematical formulation; in Section 3, we present the implementation details; in Section 4, we discuss the results, focusing on the scalability Section 4.1, and resilience Section 4.2; in Section 5, we analyze the interplay between energy and resilience. Finally, Section 6 presents the conclusions.

2. Mathematical formulation

We present the formulation for a generic 2D elliptic PDE of the form

$$\mathcal{L}y(\mathbf{x}) = g(\mathbf{x}), \tag{1}$$

where $\mathcal{L}$ is an elliptic differential operator, $g(\mathbf{x})$ is a given source term, and $\mathbf{x} = \{x_1, x_2\} \in \Omega \subset \mathbb{R}^2$, with Ω being the target domain region. We focus on Dirichlet boundary condition $y(\mathbf{x})|_{\mathbf{x}\in\Gamma} = y_\Gamma$ along the boundary Γ of domain Ω. A formulation of the algorithm focusing on 1D elliptic PDEs can be found in [8]. Elliptic equations are chosen as test case because they are a fundamental problem in physics.

Fig. 1 shows a high-level schematic of the algorithm's workflow. The starting point is the *discretization* of the computational domain. In general, the choice of the discretization method is arbitrary, potentially heterogeneous across the domain, e.g. uniform, or non-uniform rectangular grid, or a finite-element triangulation, etc.

The second step is the *partitioning* stage. The target 2D domain, Ω, is partitioned into a grid of $n_1 \times n_2$ *overlapping* regions (or subdomains), with n_k being the number of subdomains along the x_kth axis. The size of the overlap does not need to be equal and uniform among all partitions, and can vary across the domain. The partitioning stage yields a set of $n_1 \times n_2$ subdomains Ω_{ij}, and their corresponding boundaries $\Gamma_{s_{ij}}$, for $i = 0, \ldots, n_1 - 1$, and $j = 0, \ldots, n_2 - 1$, where $\Gamma_{s_{ij}}$ represents the boundary set of the ijth subdomain Ω_{ij}.

We define as our object of interest the set of solution fields along the boundaries, which we denote $y(\mathbf{x})|_{\mathbf{x}\in\Gamma_{s_{ij}}}$ for $i = 0, \ldots, n_1 - 1$, and $j = 0, \ldots, n_2 - 1$. Due to the overlapping, each subdomain Ω_{ij} includes *inner* boundaries, $\Gamma^{in}_{s_{ij}}$, i.e. the parts of the boundaries contained within Ω_{ij} that belong to the intersecting (neighboring) subdomains. The core of the algorithm

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

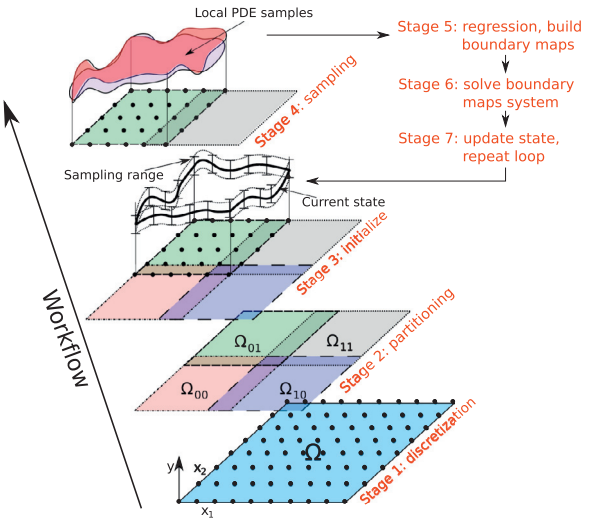

Fig. 1. Schematic of the workflow of the algorithm. For clarity, starting with stage 2 we only show the steps for Ω_{01} but the same "operations" are applied to all subdomains.

relies on exploiting within each subdomain Ω_{ij} the *map* relating the solution at the subdomain boundaries, $y(\mathbf{x})|_{\mathbf{x}\in\Gamma_{s_{ij}}}$, to the solution along the inner boundaries, $y(\mathbf{x})|_{\mathbf{x}\in\Gamma^{in}_{s_{ij}}}$. These maps can be written compactly as

$$y(\mathbf{x})|_{\mathbf{x}\in\Gamma^{in}_{s_{ij}}} = f^{(ij)}\left(y(\mathbf{x})|_{\mathbf{x}\in\Gamma_{s_{ij}}}\right), \tag{2}$$

for $i = 0, \ldots, n_1 - 1$, and $j = 0, \ldots, n_2 - 1$. The system of equations assembled from these *boundary-to-boundary maps* collected from all subdomains, combined with the boundary conditions on the full domain $y(\mathbf{x})|_{\mathbf{x}\in\Gamma}$, yields a fixed-point problem of the form

$$\mathbf{y}(\mathbf{x}) = \mathcal{F}\mathbf{y}(\mathbf{x}), \tag{3}$$

where $\mathbf{y}$ represents the vector of the solution values at all subdomains boundaries. This problem is only satisfied by the true solution. We remark that these boundary maps $f^{(ij)}$ relate the y-values, since they are built from the restrictions of the subdomain solutions at the corresponding boundaries.

In this work, rather than solving Eq. 3 directly, we construct *approximations* (or surrogates) of the boundary-to-boundary maps, which we call $\tilde{f}^{(ij)}$. One of the main features of the algorithm is that the construction of these maps can be done for each subdomain *independently* from all the others. This allows us to satisfy data locality which is key to achieve scalability on extreme scale machines. To build these surrogate maps, given a current "state" of the solution at the subdomains boundaries, we use a sampling strategy that involves solving the target PDE equation locally within each subdomain for sampled values of the boundary conditions on that subdomain, see stage 3 in Fig. 1. These samples are used within a regression approach to "infer" the approximate boundary-to-boundary maps. In general, for non-linear problems the maps are non-linear, while for linear PDEs the boundary maps are linear [8]. Following the construction of the surrogate boundary-to-boundary maps, we can then solve the *approximate* version of the fixed point system in Eq. (2), which provides us with the new solution state at all the subdomains boundaries and represents an approximation of the true solution. An important measure of the accuracy of the current solution $y(\mathbf{x})|_{\mathbf{x}\in\Gamma_{s_{ij}}}$ is the *residual* vector, defined as

$$\mathbf{z} = \mathcal{F}\mathbf{y} - \mathbf{y}, \tag{4}$$

which can be computed by extra subdomain solves using boundary conditions defined by the current solution $\mathbf{y}$, and subtracting the corresponding current solutions $\mathbf{y}$ from the resulting values at all boundaries. Given the fixed-point problem in Eq. (3), the residual (4) vanishes if the current solution $\mathbf{y}$ is the exact solution. In the case of linear PDEs, because the boundary-to-boundary maps are linear, and assuming that all the regressions complete successfully, the algorithm converges in one iteration.

The construction of the boundary-to-boundary maps plays a key role for ensuring resilience against potential silent data corruption (SDC) affecting the PDE samples. As shown in [8], when inferring linear maps, using a ℓ_1-noise model one can seamlessly filter out the effects of few corrupted data. The ℓ_1 noise model yields the solution with as few non-zero residuals as possible. Under the assumption that faults are rare, the inferred maps will fit the non-corrupted data exactly while effectively ignoring the corrupted data. In the present work, we employ an iteratively re-weighted least squares (IRLS) method, which is effectively equivalent to a ℓ_1 minimization [9]. Fig. 2 shows a test proving the resilience of the regression stage. The PDE used to generate these results is described later in the results section. Panel (a) shows sample PDE solutions generated

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

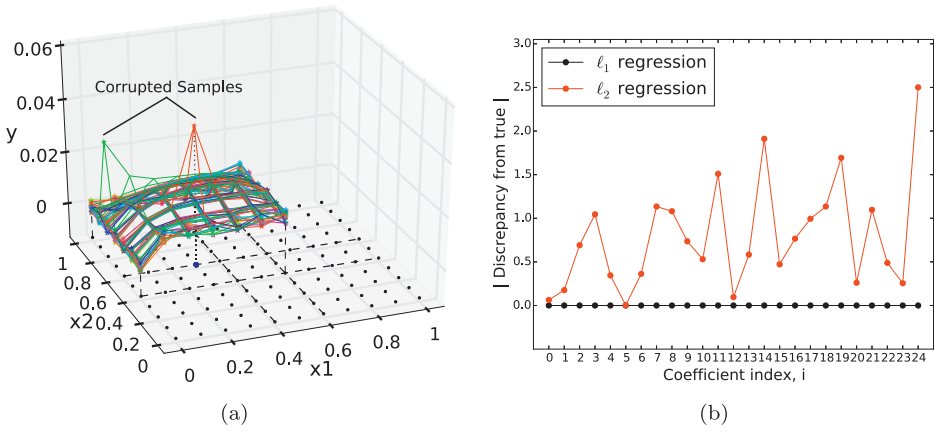

Fig. 2. Simple test proving the resilience of the ℓ_1 regression using the PDE described later in the results section. Panel (a) shows sample PDE solutions generated over a target subdomain for sampled boundary conditions. Panel (b) shows the distance of the approximate map obtained through ℓ_1 and ℓ_2 regressions, to the "true" map for the target blue point in (a). The "true" map is obtained using regression with uncorrupted samples, while the approximate map uses the data presented in (a), where two samples are corrupted. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

over a target subdomain for sampled boundary conditions. We synthetically corrupt two samples using a random bit flip: in one case, shown by the green surface, we corrupt one boundary condition point *before* the task execution; for the second corrupted sample, shown by the red surface, we corrupt a single point in the inner grid *after* the task execution. These two types of corruptions can both be categorized as SDC, because they do not cause the termination of the application.

From the PDE samples above, we collect the subset obtained at a test inner location $\mathbf{x}_* = (0.3, 0.6)$ (shown as a blue circle in the figure), and infer the approximate linear map $y_*(\mathbf{x}) = c_0 + \sum_{i=1}^{N} c_i y_i(\mathbf{x})$, where i enumerates the points along the subdomain perimeter. For this test, we generated a total of 30 PDE samples: 25 is the minimum number to have a well-posed linear regression given the size of the subdomain ($N = 24$ for this specific case), while 5 additional samples are added to guard against possible faulty data. We perform the regression using both the ℓ_1 and ℓ_2 models, and report in panel (b) the error between the approximate map and the one obtained using the uncorrupted samples. The results show that the ℓ_1-based regression matches exactly the uncorrupted result, being completely unaffected by the corrupted data points. On the contrary, using ℓ_2 yields the wrong answer since the corrupted data have substantial effect. The key underlying point demonstrated is that even in the presence of corrupted PDE samples, provided we have enough samples, we do not need to waste resources and energy to identify them in order to have a successful regression. The correctness of the result is ensured by the mathematical properties of the regression model.

3. Implementation details

We have developed a parallel, C++ implementation of the algorithm using a server-client model (SCM). This section describes the SCM, its resilience properties, and how we implement each stage of the algorithm to exploit the SCM model.

3.1. Server-Client model

Fig. 3 shows a schematic of our SCM structure. We adopt a cluster-based model where the MPI ranks are grouped into separate clusters, with each cluster containing a server and, for resource balancing purposes, the same number of clients. All servers can communicate between each other, while the clients within a cluster are only visible to the server within that cluster. Moreover, within any given cluster, clients are independent, i.e. at a given time instant, each client is handling a different work unit and they cannot communicate with each other. This design choice allows a client to fail without affecting other clients. Only the work being executed by the failed client is affected.

The data is distributed among the servers, and these are assumed to be highly resilient (safe or under a "sandbox" model implementation). The sandbox model assumed for the servers can be supported by either software or hardware. In the case of software support, this can be accomplished by a programming model relying on data redundancy and strategic synchronization [10–12]. The latter assumption is supported by hardware specifications on the variable levels of resilience that can be allowed within large computer systems.

Since the servers hold the data, they are responsible for generating work in the form of tasks, dispatching them to their pool of available clients, as well as receiving and processing tasks. A client is defined as a set of MPI processes, and is designed solely to accept and perform work without any assumption on its reliability. To optimize communication, the root rank of a client is in charge of receiving work from the server, and then distributing it among the children ranks within that client. This paradigm can be exploited in certain hardware configurations because leveraging local communication within a

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

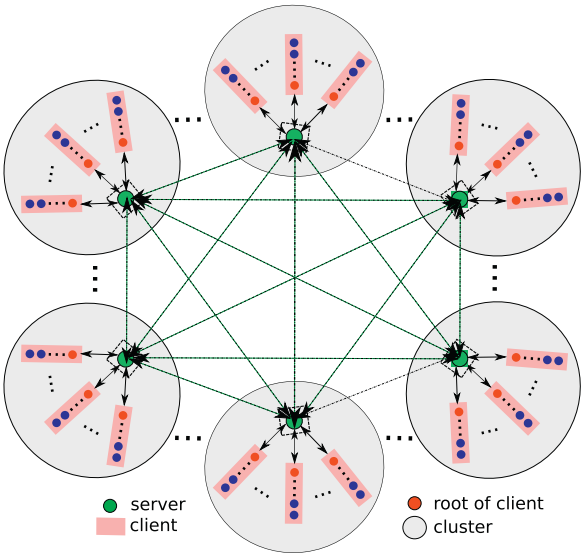

Fig. 3. Schematic of the server-client structure.

client is more efficient than having the server communicate a task to all the MPI ranks in a client. One example is the case where all ranks of a client live in the same node, so that one can exploit in-node parallelism and faster memory access.

A key property of the SCM structure is the inherent resiliency to hard faults in the sense that clients crashing do not affect the state safely owned by the servers. This is because clients crashing (partial or complete node failures) only translates into missing tasks. In order to take advantage of the SCM model, an application should be designed to handle missing data. Our SCM is implemented using the *User Level Fault Mitigation* MPI (MPI-ULFM) [7], a fault tolerance capability proposed for the MPI standard that enables a fault-tolerant MPI framework.

The proposed SCM has the potential to be extremely compatible with hardware designs targeting energy efficiency through approaches like dynamic voltage/frequency scaling (DVFS) or heterogeneous micro-architectures (HMs) [13,14]. One can envision an architecture with servers operating at the maximum allowed voltage/energy requirements for best resilience, while the clients (in charge of doing all the computations) are adaptively operated at various levels of voltage/frequency to decrease the overall energy consumption. This settings provides a suitable avenue for energy reduction given that the servers are expected to occupy a minimal part of the machine (e.g. less than 10%), while the clients occupy most of the machine (e.g. more than 90%).

3.2. Algorithm implementation

The algorithm described in Section 2 involves four main stages: sampling, regression, fixed-point solve, and updating. As mentioned before, sampling and regression can be performed independently and concurrently across all subdomains. This feature reveals their task-based nature, and are therefore implemented in the form of tasks executed by the clients. As such, they are vulnerable to the failures occurring on the clients. On the other hand, the fixed-point solve of the boundary-to-boundary maps system and the updating of the subdomains are safely executed by the servers, since they fully own the state information. The system of equations built from the boundary maps is much smaller than original discretized PDE system over the full domain grid, and so it fits on a small number of servers. Moreover, the servers are assumed to be "sandboxed", allowing us to circumvent any potential data corruption during these operations. This design choice follows the concept of target reliability [15], where some parts of the algorithm are assumed to be handled in a reliable manner. This can be accomplished either by making the hardware more reliable or by incorporating it within the algorithm itself.

4. Results

The results presented below are based on the following 2D linear elliptic PDE

$$\frac{\partial}{\partial x_1}\left(k(\mathbf{x})\frac{\partial y(\mathbf{x})}{\partial x_1}\right)+\frac{\partial}{\partial x_2}\left(k(\mathbf{x})\frac{\partial y(\mathbf{x})}{\partial x_2}\right)=g(\mathbf{x}), \tag{5}$$

where $\mathbf{x}=\{x_1,x_2\}$, the field variable is $y(x_1, x_2)$, $k(x_1, x_2)$ is the diffusivity, and $g(x_1, x_2)$ is the source term, over the unit square $(0, 1)^2$ with homogeneous Dirichlet boundary conditions. The diffusivity and source fields are defined as

$$k(x_1,x_2)=8.0*\exp(-d(x_1,x_2)/0.025)+2.0, \tag{6}$$

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

Table 1
Scalability parameters.

Weak Scaling Parameters	
Subdomains	12^2, 18^2, 24^2, 30^2, 36^2, 42^2
Subdomain grid size	180^2
Num. of Servers	16, 36, 64, 100, 144, 196
Num. of clients/server	64
Size of client	4 MPI ranks
Total Cores	4112, 9252, 16448, 25700, 37008, 50,372

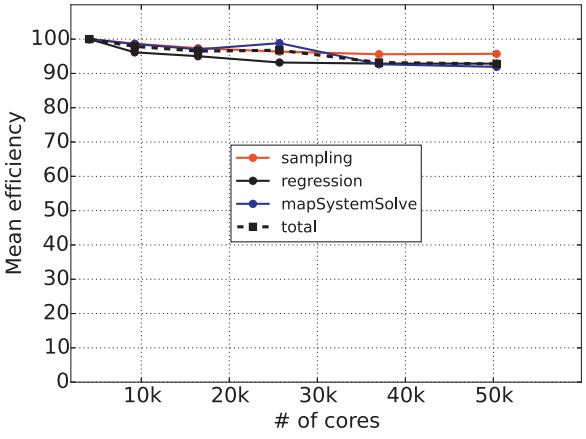

Fig. 4. Nominal weak scaling results: the mean efficiency is defined as t_{ref}/t^*100, where t_{ref} is the execution time for the smallest case.

$$g(x_1, x_2) = 2.0 * \exp(-d(x_1, x_2)/0.050) - 1.0, \tag{7}$$

where $d(x_1, x_2) = (x_1 - 0.35)^2 + (x_2 - 0.35)^2$. To solve the above PDE within each subdomain, we employ a structured grid and second-order finite differences to discretize Eq. (5). The resulting linear system stemming from the finite-difference discretization is solved using the parallel solver AztecOO in Trilinos [16].

4.1. Nominal scalability

We tested the scalability of our application on Edison at NERSC[1], a Cray XC30, with Peak performance of 2.57 Petaflops, Cray Aries high-speed interconnect with Dragonfly topology with approximately ~8GB/sec MPI bandwidth. Table 1 lists the parameters used for the scalability runs. These runs use the Edison's native Cray–MPICH.

The weak scaling is setup by fixing the number of clients per server and the amount of data owned by each server, and increasing the problem size by adding increasingly more clusters, as shown in Table 1. This design imposes no constraint on the problem size, since larger problems can be tackled by simply adding more clusters. Fig. 4 presents the results up to ~51K cores, specifically highlighting the scaling of the most important stages of the algorithm, as well as the scaling of the full application. The results show an excellent behavior for each individual stage and for the full application, with efficiency above 90%.

4.2. Resiliency

Thissection describes and demonstrates the resilience properties of our PDE solver, specifically focusing on resilience to SDC. We evaluate the resilience against SDC affecting numerical data used in the algorithm, i.e. we exclude other types of faults e.g., in data structures or control flow, since these issues represent a different problem [17]. We model SDC as bit-flips, and the results below are based on a random bit-flip model to inject the faults. Specifically, we explore the case where a SDC is caused by a single random bit-flip, as well as two random bits being flipped. The latter is more rare than the former, but also more dangerous because current technology like ECC can only handle single bit-flips [18]. Contrary to the work by [17,19], we do not characterize only the effect of the worst case scenario, or very outrageous faults. We believe, in fact, that in many scientific applications, the biggest problems might not come from the occurrence of an outrageous fault, but from a small corruption in the data at some point during the simulation. One example is scientific simulations that tie to chaos

[1] http://www.nersc.gov

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

theory, like climate models and/or turbulence: in such cases, the simulations are very sensitive to, e.g., initial conditions, where even small variations of the initial data can yield large deviations in the model predictions. As will be shown below, in fact, this is the case for the present algorithm. When an outrageous fault occurs, it is easily filtered out. However, when a more subtle corruption occurs, the solver takes more effort to converge towards the right solution. The variability of corruptions possible by using a bit flip model is a useful resource allowing us to test and assess our applications under all these various scenarios.

Test Problem and Execution

As a test problem for resilience, we adopt the PDE introduced in Eq. (5), solved over a structured uniform grid with 201^2 grid points over the unit square domain. We partition the domain using $n_1 = n_2 = 4$ subdomains, with an overlapping of 4 grid cells between neighboring subdomains. This setting yields a total of 16 subdomains, each with a local grid of 54^2 grid points. Nominally, this problem involves $N^s_{nom} = 3408$ sampling tasks, and $N^r_{nom} = 2496$ regression tasks. The SCM structure involves a single server holding the data, i.e. subdomains, and uses 14 clients each with size 2.

Fault Injection

Our goal is to evaluate how the application behaves as we increase the number of faults. We leverage the task-based nature of our algorithm by choosing the number of faults to inject as a percentage of the nominal number of tasks to execute. Since we know in advance how many tasks are needed by our test problem, we randomly select off-line the set of task IDs that will be hit by a fault during the execution. One advantage of this method is that the number of faults hitting the system is well-known, and it eliminates any dependency between faults occurrence and the execution time, since the latter is machine-dependent. Moreover, if needed, this setting still allows us to extract an average fault rate, given a total execution time and the known number of faults. We explore various levels of corruption, namely 0.25%, 0.5%, 0.75%, and 1.0% of the nominal number of tasks. For each percentage of corruption, we run 150 runs to have a statistically meaningful result.

Exploiting a *selective reliability* approach [15,17,19,20], which lets algorithm developers isolate faults to certain parts of the algorithm, in this paper we focus on the results obtained by injecting the faults during the *sampling stage* only. The results obtained for the other scenarios, e.g. involving faults hitting the regression are left for future extensions. Faults are only injected in the clients, consistently with the SCM described previously, where servers are assumed to be reliable, while no assumption is made on the reliability of the clients.

Fault Model

In this work, we analyze two cases: one involves corrupting all elements in the array holding boundary conditions contained in the task, while the other involves corrupting only a single array element. The reason behind this distinction is that when the whole data-set is corrupted, provided the amount of data is sufficiently large, it is likely that at least one large bit (e.g. exponent) is flipped, causing the value to become “outrageous”. One the contrary, when a single data variable is corrupted, the likelihood of flipping a bit in the exponent is lower. We believe that both scenarios are important to examine, since they provide information on the algorithm’s sensitivity to different levels of data corruption.

When a fault needs to be injected, we adopt the following procedure: we draw a value, u, from a standard uniform distribution, and if $u \leq 0.5$, the task data is corrupted *before* the execution; if $u > 0.5$, the task data is corrupted *after* its execution. This allows us to mimic corruptions that occur when tasks are being transmitted to and from a client, as well as those happening during execution. If a task is corrupted before the execution, this translates into corrupting all or a single point in the boundary conditions owned by that sampling task, since the boundary conditions are the only information needed to run a solve of the elliptic PDE. If a task is corrupted after execution, this translates into corrupting all or a single point in the solution, which means that even the inner points of a subdomain can be affected.

Handling Faults

Given the fault model described above, several fault scenarios unfold. The mechanisms that we incorporate in the algorithm to make it resilient are kept to a minimum in order to reduce the overhead. To guarantee the resilience of the algorithm towards faults in the sampling, the key condition to be satisfied is that out of the samples used in the regression, the number of uncorrupted samples has to be greater than the minimum set needed to have a mathematically well-posed regression problem. To this end, we mitigate the impact of faults in the sampling by generating, within each subdomain, more samples than the minimum set. This is accomplished by defining an oversampling factor, $\rho > 1$, such that the target number of samples to generate is $N = \rho N^s_{nom}$. Moreover, during the sampling stage, we apply a filter on the task data returned to the server to check that it is within the interval $(-100, 100)$ before the data is stored within the corresponding subdomain. Based on physical insight into the problem, solutions outside this range are clearly the result of corrupted data. Moreover, only very few cases return results outside of this range as a result of random bit-flips. This interval is arbitrary, but can be estimated by either a domain expert or by known physical bounds on the solution. In general, this bound does not have to be tight. If the problem under consideration has some related experimental data, then those could be easily used to derive some bounds. Otherwise, one can resort to some theoretical analysis to derive bounds for the PDE solution under study, see e.g. [21]. This is the only active “filter” that is needed by the application. Any other corruption during the sampling does not need to be actively detected, since it is seamlessly filtered out thanks to the mathematical model used in the regression as shown in Fig. 2.

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

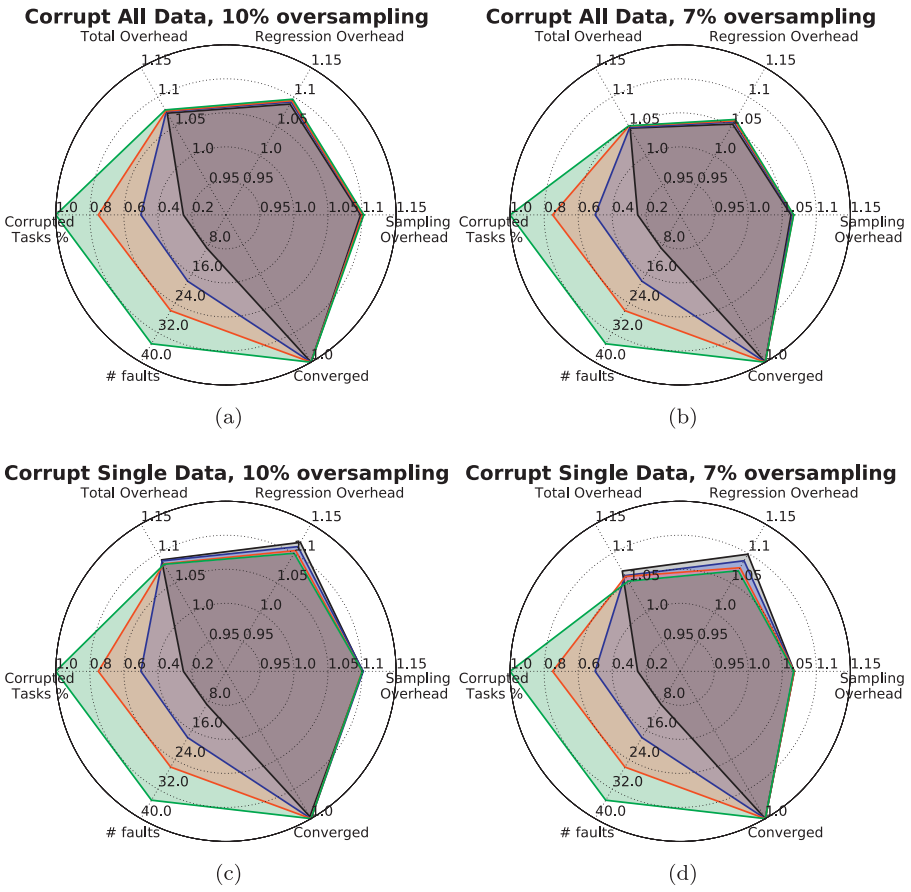

Fig. 5. Statistical results obtained from the ensemble runs performed for the resilience analysis. The radar-plots correspond to the following four cases: first row shows results when all data in a sampling task is corrupted, with $\rho = 1.1$ (a) and $\rho = 1.07$ (b); second row shows results for single data point corruption in a sampling task, with $\rho = 1.1$ (c) and $\rho = 1.07$ (d).

4.2.1. Resilience analysis

This section focuses on the resilience results obtained for the runs described above. Fig. 5 summarizes the main statistics from the ensemble runs performed for the *single* bit-flip scenario. Specifically, we show four radar-plots, corresponding to the following four different combinations: first row shows the results when all data in a sampling task is corrupted, with $\rho = 1.1$ (a) and $\rho = 1.07$ (b); second row shows results for single data point corruption in a sampling task, with $\rho = 1.1$ (c) and $\rho = 1.07$ (d). Each radar-plot displays the average value over 150 replicas of six key quantities in the following clock-wise order: the total overhead of the application runtime compared to the no-fault runs, the regression and sampling overhead, the boolean value identifying convergence, the number of tasks affected by a fault, and the percentage of tasks corrupted. In all cases, the runs converge to the correct answer. Convergence is verified by checking that the root-mean-square error computed for the residual in Eq. (4) is below a specified threshold of 10^{-13}. This proves that the application is resilient to faults occurring during sampling.

For a fixed value of ρ, the plots show that the overhead with respect to the no-fault case is smaller when all the data in a sampling task is corrupted. As it was mentioned before, this is explained by the fact that when every data element is hit by a bit-flip, it is more likely that it is filtered by the server during the task post-processing check. On the contrary, when a single bit-flip hits a single data point, the ℓ_1 regression has to work harder to obtain the correct boundary-to-boundary map. For $\rho = 1.1$, the total overhead is ~7% when all data is corrupted, only slightly changing as the number of fault increases, see Fig. 5a. For the single data point corruption, the total overhead increases from ~7.5% to 9% when the number of faults increases from 9 to 35, see Fig. 5c. A similar trend is observed for $\rho = 1.07$. In this case, however, there is a wider discrepancy between the overhead obtained when all the data is corrupted, and that of a single point corruption, see Fig. 5b and d.

As expected, the plots confirm that as the oversampling is reduced from 10% to 7%, the overhead incurred by the application decreases. A greater reduction is observed when all data is corrupted, as shown by Fig. 5a and b. The plots show also that the largest variability is observed for the regression overhead. This is expected, because of the additional computations required to compute the correct boundary-to-boundary maps given the presence of faulty samples.

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

Table 2
Overhead comparison between single bit-flip (SBF) and double bit-flip (DBF) corruption for the case where a single data point is corrupted and 7% oversampling.

Corrupted tasks %	0.25% SBF DBF	0.5% SBF DBF	0.75% SBF DBF	1.0% SBF DBF
Regression Overhead (%)	1.099 1.111	1.088 1.092	1.076 1.085	1.071 1.071
Total Overhead (%)	1.070 1.075	1.062 1.069	1.061 1.063	1.053 1.054

We repeated a similar analysis for the case when SDCs are caused by double random bit-flips. For brevity, we discuss here only the results obtained when a single data element is corrupted and the oversampling is set to 7%. Table 2 shows a comparison of the overhead incurred when SDCs are caused by single and double bit-flips, for various percentages of corrupted tasks. The results show that the differences are minimal, demonstrating that the algorithm formulation and the underlying ℓ_1 regression model used make the application minimally sensitive to the number of bit-flips occurring.

5. Trade-off between energy and resilience

In this section, we discuss how the resilience properties of our application allow us to draw conclusions about potential savings in the energy consumption. Resilience and energy consumption are tightly linked [22,23]. It has been shown that voltage decrease is linked to higher faults rates, see [22] and references therein. We demonstrate below how lowering the energy consumption during the *sampling stage* by means of voltage scaling allows us to save energy and still run the application successfully despite being affected by more frequent system faults. This framework can be enabled because of the SCM, which allows us to separate state from computation. Decreasing the energy consumption is possible via variable-voltage CPUs, which can reduce power consumption quadratically at the expense of linearly reduced speed [22]. The reason for this is that circuit delay is almost linearly related to $1/V$, where V is the voltage, so for systems to function correctly, the operating frequency needs to decrease linearly with respect to the decrease of the supply voltage.

We compare two scenarios: (A) involves running the application assuming that all machine components run at full operational capacity/speed; (B) is based on decreasing the energy consumption of the clients only during the sampling stage by reducing the operational efficiency of the corresponding processing units. Moreover, to compare the two scenarios, they tackle the same problem, have the same number of servers and clients, are run on the same machine. The servers always run at full capacity to keep the state safe. We remark that the only difference between the two cases lies in how the sampling stage is run. The other stages of the algorithm are equivalent. The analysis below is carried out without accounting for the energy consumed by the network and/or system cooling. We are aware that the energy consumption for these HPC systems' components contributes to the total energy budget, but we limit the focus of this work to highlighting the relationship between system faults, algorithmic resilience and computing energy.

To lay the ground of the analysis, we define how to estimate the energy consumption following the work in [22,23]. The power consumption, P, during activity can be modeled as [22]:

$$P = \hat{P} + CV^2 f \tag{8}$$

where $\hat{P}$ is the non-dynamic contribution, C is the switch capacitance, V is the voltage, and f is the frequency. In general, this $\hat{P}$ includes leakage and shortcircuit power. In current systems, with improved technologies, the dynamic component dominates. For the purposes of this work, as in [22], we assume $\hat{P}$ to be independent of V and f, and small compared to the dynamic part. Note that as in [22], we have neglected the contribution of the sleep power since it does not have any effect on the energy savings and we assume the system to be always on. The energy consumed by an operation running over the time interval $T = t_2 - t_1$ is then

$$E = (\hat{P} + CV^2 f)T. \tag{9}$$

We now proceed by estimating the energy consumption of our application in both scenarios. The nominal (or full energy) scenario involves clients operating at maximum voltage, V_m, and frequency, f_m, and includes N_1 sampling tasks, with each task execution taking a time T_1. For this reference scenario, the total energy consumed by the clients to execute N_1 samples is

$$E_1^s = N_1\left(\hat{P}T_1 + CV_m^2 f_m T_1\right). \tag{10}$$

For the reduced energy case, as mentioned above, both voltage and frequency are lowered to $V_2 < V_m$, and $f_2 < f_m$, such that $f_m/f_2 = V_m/V_2$. This implies that the execution time of a task is $T_2 = T_1 \frac{f_m}{f_2}$, as it depends linearly on the frequency. This assumption is, in general, valid for compute-intensive tasks. Due to the interplay between voltage and reliability, we expect for this low-energy scenario a higher probability of faults occurring during the sampling. We mitigate the effect of these faults by generating more samples, i.e. we assume $N_2 = \rho N_1$, where $\rho > 1$ is the oversampling factor. This oversampling is needed for the algorithm, as shown in the resilience results from the previous section, to guard against potential data

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

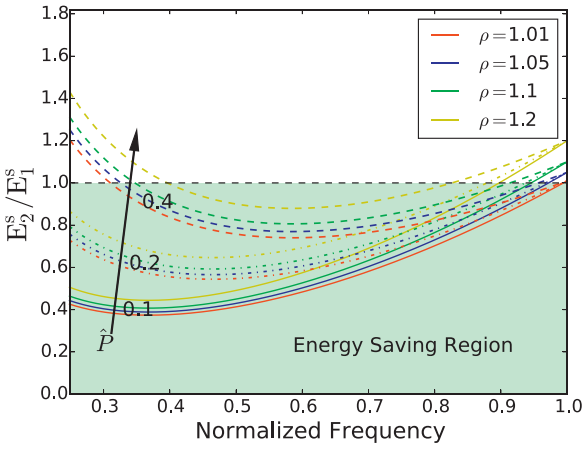

Fig. 6. Energy ratio E_2^s/E_1^s between the reduced, E_2, and full, E_1, case as a function of the normalized frequency. We show the curves obtained for $\hat{P} = \{0.1, 0.2, 0.4\}$, and varying oversampling factor $\rho = \{1.01, 1.05, 1.1, 1.2\}$. (Frequencies below 0.25 are not shown assuming 0.25 to be a reasonable value for the lowest operational frequency of a processor).

corruption. The total energy consumed by the clients to execute N_2 samples in the reduced energy case is

$$\begin{aligned} E_2^s &= N_2\left(\hat{P}T_2 + CV_2^2 f_2 T_2\right) \\ &= N_2\left(\hat{P}T_1\frac{f_m}{f_2} + CV_m^2 f_m T_1 \frac{f_2^2}{f_m^2}\right), \end{aligned} \tag{11}$$

where Eq. (11) has been obtained by making some substitutions and rearranging terms to explicitly highlight the relationship with the nominal case. It is easy to see that the energy function has a minimum at $f_2^* = (0.5\hat{P}/C)^{1/3}$. Below, we assume that voltages and frequencies are normalized, i.e. we set $V_m = 1$ and $f_m = 1$, such that any other voltage or frequency is in the interval (0, 1).

Fig. 6 shows the ratio E_2^s/E_1^s as a function of the frequency f_2. We show the results obtained for $\hat{P} = \{0.1, 0.2, 0.4\}$, and also vary the oversampling factor $\rho = \{1.01, 1.05, 1.1, 1.2\}$. This figure allows us to make conclusion about how much energy we are able to save by leveraging the reduced-energy scenario as opposed to the full one. We can draw the following observations. First, the lower the value of $\hat{P}$, the more energy we are able to save. This is clear from Eq. 11, since the contribution from $\hat{P}$ is proportional to the execution time. Second, the results show that as $\hat{P}$ decreases, the minimum of the energy curves is obtained for smaller and smaller frequencies. For the three cases shown, the optimal frequencies are $f_2^* = \{0.369, 0.464, 0.585\}$ for $\hat{P} = \{0.1, 0.2, 0.4\}$. Third, we see that for a given value of $\hat{P}$, the energy curves shift upward as the oversampling ratio ρ increases. Intuitively, increasing ρ increases the number of tasks to execute, and therefore we are able to save less and less energy. The results show, e.g., that if we assume $\hat{P} = 0.4$, and run at the optimal frequency $f_2^* = 0.585$, the energy saving ranges from ∼30% when $\rho = 1.01$, to ∼15% when $\rho = 1.2$. The frequency range shown in Fig. 6 was chosen to reveal the full trend of each curve for the selected values of $\hat{P}$ and ρ. But we remark that operating at the optimal (or too small) frequency is not always possible if this frequency is smaller than the minimum operational frequency, f_{low}, allowed by the processor, i.e. $f_2^* < f_{low}$, for a given voltage. To the best of our knowledge[2], we think that a reasonable assumption would be $0.2 < f_{low} < 0.4$. The operational minimum energy efficient frequency that is feasible for a processor is $max\{f_{low}, f_2^*\}$. Hence, the energy saved might be slightly smaller, but the plot shows that it is still considerable.

Beside increasing the execution time, voltage scaling also causes fault rates in the processors (including logic core and cache) to increase exponentially [22–24]. Assuming a Poisson process for the faults [22], the relationship between the fault rate, λ (# faults/sec), the voltage V and frequency f can be expressed as:

$$\lambda = \lambda_0 10^{\left(d\frac{1-f}{1-max\{f_{low}, f_2^*\}}\right)}, \tag{12}$$

where λ_0 is the fault rate corresponding to V_m and f_m, d is a constant such that the larger its value, the more sensitive the fault rate is to scaling, and f is the frequency corresponding to voltage V, such that $V_m/V = f_m/f$. Also, the dependence on $\hat{P}$ appears through the optimal frequency f_2^*, since $f_2^* = (0.5\hat{P}/C)^{1/3}$ as shown earlier. Similarly to [22], in this work we assume $\lambda_0 = 10^{-6}$ (# faults/sec), and $d = 4$. We remark that the expression above might not be exactly applicable to current systems, but it is consistent with recent work on this topic, see e.g. [25]. The analysis below discusses the potential workflow that one would have to follow to calibrate the oversampling for tolerating an expected number of faults. It does not provide an exact formula for the dependence of the fault rate on the frequency.

Fig. 7 shows how the fault rate varies as a function of the frequency for the three different values of $\hat{P}$ selected before. For clarity, the y-axis is plotted in log-scale. The first observation is that as the frequency decreases, the fault rate increases.

[2] https://www.pugetsystems.com/labs/articles/Is-CPU-Base-Frequency-Still-a-Relevant-Spec-512/

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

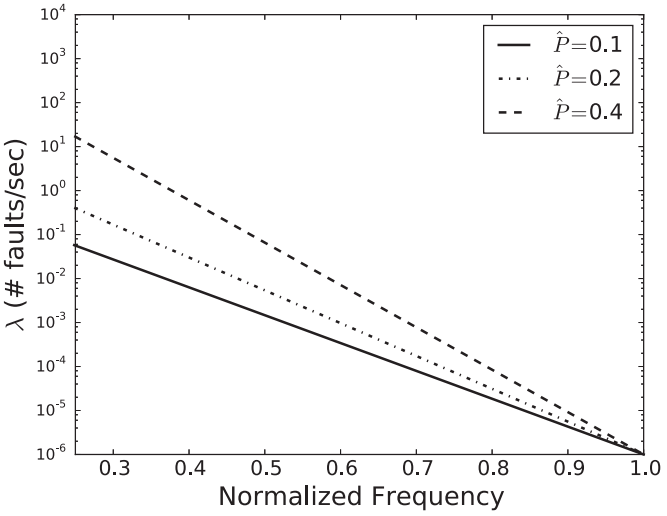

Fig. 7. Fault rate, λ, as a function of the frequency, for three different values of $\hat{P}$, see Eq. (12). For clarity, the y-axis is plotted in log-scale.

From the figure, we can see that if the operating frequency is $f_2^* = \{0.369, 0.464, 0.585\}$, the corresponding expected fault rate is $\lambda = 0.01$, namely about one fault every 100 seconds. The key question arising is: what is the trade-off between the energy saved due to voltage scaling and the energy spent to recover from the more frequent faults occurring due to lower operation power? This is where the resilience plays a key role, especially if the algorithm is inherently resilient to faults. If an application had a small overhead associated with recovering from faults, then most of the energy saved by running at reduced speed would be gained. On the other hand, if the overhead of the application to recover from a fault is substantial, then some of the energy saved by running at reduced speed would be offset, potentially eliminating any energy savings.

To calibrate a particular run, a user will have to choose a lower frequency f_2 to run the sampling stage, then use Fig. 6 to determine the energy savings for a corresponding oversampling factor ρ. The expected fault rate can be computed using Eq. (12). The user must determine if the oversampling factor chosen is large enough to compensate for the expected number of faults and adjust the oversampling factor accordingly. If it is allowed by the machine, the target low-energy frequency f_2 should be chosen as close as possible to the optimal frequency f_2^* to ensure that the energy saved during the sampling stage is not offset by the overhead of the regression stage.

6. Conclusion

We discussed algorithm-based resilience to silent data corruption (SDC) in a task-based domain-decomposition preconditioner for partial differential equations (PDEs).

The algorithm involves the following main steps: first, the domain of the PDE is split into overlapping subdomains; second, the PDE is solved on each subdomain for sampled values of the local current boundary conditions; third, the resulting subdomain solution samples are fed into a regression step to build boundary-to-boundary maps; finally, the intersection of these maps yields the updated state at the subdomain boundaries.

The implementation is based on a server-client model where all state information is held by the servers, while clients are designed solely as computational units. We tested weak scaling up to $\sim 51K$ cores, showing an efficiency greater than 90%.

We used a 2D elliptic PDE, a fault model based on random single bit-flip and target reliability assumption to show that the application is resilient to SDC injected during the sampling stage. The resilience to SDC was shown to be feasible thanks to the ℓ_1 model adopted in the regression stage. We discussed how the overhead in the regression due to the presence of faults is larger when a single data point is corrupted in a sampling task than the case where all data is corrupted. We explained this result in terms of the likelihood of having an outrageous corruption, which would simply be filtered/discarded by the server, versus having small corruptions passing the filter test, but causing the regression to work harder to obtain the ℓ_1 result. We showed that because of the algorithm formulation and the ℓ_1 regression model used in the regression, the algorithm is nearly insensitive to type of corruption, namely single or double bit-flip.

Finally, we showed how the inherent resilience to SDC in the sampling and the small associated overhead can be leveraged to achieve potential energy savings via dynamics voltage/frequency scaling during the sampling. We anticipate that in the case of undervolting only, namely reducing voltage while keeping frequency constant, the potential energy-saving gain would be even higher because there would be no increase in the computational time. We are currently planning further studies to complement the presented energy estimates with experimental data.

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Numbers 13-016717. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] DOE-ASCR, Exascale Programming Challenges, Technical Report, 2011. URL http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/ProgrammingChallengesWorkshopReport.pdf.
[2] J.A. Ang, R.F. Barrett, R.E. Benner, D. Burke, C. Chan, J. Cook, D. Donofrio, S.D. Hammond, K.S. Hemmert, S.M. Kelly, H. Le, V.J. Leung, D.R. Resnick, A.F. Rodrigues, J. Shalf, D. Stark, D. Unat, N.J. Wright, Abstract machine models and proxy architectures for exascale computing, in: Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing, in: Co-HPC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 25–32, doi:10.1109/Co-HPC.2014.4.
[3] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R.S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R.S. Williams, K. Yelick, Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead, 2008.
[4] DOE-ASCR, Top Ten Exascale Research Challenges, Technical Report, 2014.
[5] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, M. Snir, Toward exascale resilience: 2014 update, Supercomputing frontiers and innovations 1 (1) (2014). URL http://superfri.org/superfri/article/view/14.
[6] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, M. Snir, Toward exascale resilience, Int. J. High Perform. Comput. Appl. 23 (4) (2009) 374–388.
[7] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, J. Dongarra, Post-failure recovery of mpi communication capability: design and rationale, Int. J. High Perform. Comput. Appl. 27 (3) (2013) 244–254, doi:10.1177/1094342013488238.
[8] K. Sargsyan, F. Rizzi, P. Mycek, C. Safta, K. Morris, H. Najm, O.L. Maȋtre, O. Knio, B. Debusschere, Fault resilient domain decomposition preconditioner for pdes, SIAM J. Sci. Comput. 37 (5) (2015) A2317–A2345, doi:10.1137/15M1014474.
[9] I. Daubechies, R. DeVore, M. Fornasier, C.S. Güntürk, Iteratively reweighted least squares minimization for sparse recovery, Commun. Pure Appl. Math. 63 (1) (2010) 1–38, doi:10.1002/cpa.20303.
[10] M.-L. Li, P. Ramachandran, S.K. Sahoo, S.V. Adve, V.S. Adve, Y. Zhou, Understanding the propagation of hard errors to software and implications for resilient system design, SIGOPS Oper. Syst. Rev. 42 (2) (2008) 265–276, doi:10.1145/1353535.1346315.
[11] P.G. Bridges, K.B. Ferreira, M.A. Heroux, M. Hoemmen, Fault-tolerant linear solvers via selective reliability, (2012). ArXiv e-prints arXiv:1206.1390
[12] C. Engelmann, T. Naughton, Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems, in: Parallel Processing (ICPP), 2013 42nd International Conference on, 2013, pp. 960–969, doi:10.1109/ICPP.2013.114.
[13] T. Miller, X. Pan, R. Thomas, N. Sedaghati, R. Teodorescu, Booster: Reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips, in: High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on, 2012, pp. 1–12, doi:10.1109/HPCA.2012.6168942.
[14] J. Lee, V. Sathisha, M. Schulte, K. Compton, N.S. Kim, Improving throughput of power-constrained gpus using dynamic voltage/frequency and core scaling, in: Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on, 2011, pp. 111–120, doi:10.1109/PACT.2011.17.
[15] P.G. Bridges, K.B. Ferreira, M.A. Heroux, M. Hoemmen, Fault-tolerant linear solvers via selective reliability, ArXiv e-prints arXiv:1206.1390 (2012).
[16] M. Heroux, R. Bartlett, V.H.R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, An Overview of Trilinos, Technical Report SAND2003-2927, Sandia National Laboratories, 2003.
[17] J. Elliott, M. Hoemmen, F. Mueller, Evaluating the impact of sdc on the gmres iterative solver, in: Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, in: IPDPS '14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 1193–1202, doi:10.1109/IPDPS.2014.123.
[18] M. Snir, R.W. Wisniewski, J.A. Abraham, S.V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A.A. Chien, P. Coteus, N. DeBardeleben, P.C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, E.V. Hensbergen, Addressing failures in exascale computing, IJHPCA (2014) 129–173.
[19] J. Elliott, M. Hoemmen, F. Mueller, A numerical soft fault model for iterative linear solvers, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, in: HPDC '15, ACM, New York, NY, USA, 2015, pp. 271–274, doi:10.1145/2749246.2749254.
[20] J.J. Elliott, M.F. Hoemmen, F. Mueller, Tolerating Silent Data Corruption in Opaque Preconditioners., 2014.
[21] P. Mycek, F. Rizzi, O.L. Maȋtre, K. Sargsyan, K. Morris, C. Safta, B. Debusschere, O. Knio, Discrete a priori bounds for the detection of corrupted pde solutions in exascale computations, SIAM J. Sci. Comput. 39 (1) (2017) C1–C28, doi:10.1137/15M1051786.
[22] D. Zhu, R. Melhem, D. Mosse?, The effects of energy management on reliability in real-time embedded systems, in: Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on, 2004a, pp. 35–40, doi:10.1109/ICCAD.2004.1382539.
[23] D. Zhu, R. Melhem, D. Mosse, E. Elnozahy, Analysis of an energy efficient optimistic tmr scheme, in: Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on, 2004b, pp. 559–568, doi:10.1109/ICPADS.2004.1316138.
[24] V. Chandra, R. Aitken, Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos, in: Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on, 2008, pp. 114–122, doi:10.1109/DFT.2008.50.
[25] L. Tan, S.L. Song, P. Wu, Z. Chen, R. Ge, D.J. Kerbyson, Investigating the interplay between energy efficiency and resilience in high performance computing, in: 2015 IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 786–796, doi:10.1109/IPDPS.2015.108.

Please cite this article as: F. Rizzi et al., Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.05.005