Parallel Computing 73 (2018) 16–27

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 25 May 2017

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