Application-Based Fault Tolerance for Numerical Linear Algebra at Large Scale

CAMILLE COTI, LAURE PETRUCCI, and DANIEL ALBERTO TORRES GONZÀLEZ, LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, FRANCE

Large scale architectures provide us with high computing power, but as the size of the systems grows, computation units are more likely to fail. Fault-tolerant mechanisms have arisen in parallel computing to face the challenge of dealing with all possible errors that may occur at any moment during the execution of parallel programs. Algorithms used by fault-tolerant programs must scale and be resilient to software/hardware failures. Recent parallel algorithms have demonstrated properties that can be exploited to make them fault-tolerant. In my thesis, we design, implement and evaluate parallel and distributed fault-tolerant numerical computation kernels for dense linear algebra. We take advantage of intrinsic algebraic and algorithmic properties of communication-avoiding algorithms in order to make them fault-tolerant. We are focusing on dense matrix factorization kernels: we have results on LU and preliminary results on QR. Using performance evaluation and formal methods, we are showing that they can tolerate crash-type failures, either re-spawning new processes on-the-fly or ignoring the error.

ACM Reference Format:
Camille Coti, Laure Petrucci, and Daniel Alberto Torres Gonzàlez. 2021. Application-Based Fault Tolerance for Numerical Linear Algebra at Large Scale. 1, 1 (September 2021), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

High Performance Computing (HPC) systems continue growing exponentially; the number of processors and nodes is increasing. *Top500* is a statistical list with ranks and details of the 500 world's most powerful supercomputers. The November 2020 Top500 ranking shows that 5 machines feature more than a million of cores and all 500 machines listed have more than 10 000 cores (not including accelerators). Meanwhile, as the number of hardware components increases, the overall system Mean Time Between Failures (MTBF) is reduced to only a few hours [12]. For instance, the supercomputer Blue Waters located at the National Center for Supercomputing Applications (NCSA) at the University of Illinois had an MTBF of approximately 4.8 hours [11]. Therefore, fault tolerance is necessary for such large scale systems to ensure that computational intensive applications can survive failures with a small overhead.

The total number of hardware and software components, the complexity of these components and the system reliability, availability and scalability are factors to deal with in HPC systems, because hardware or software failures may occur anytime during the execution of high parallel applications [9].

Authors' address: Camille Coti, camille.coti@lipn.univ-paris13.fr; Laure Petrucci, laure.petrucci@lipn.univ-paris13.fr; Daniel Alberto Torres Gonzàlez, daniel.torres@lipn.univ-paris13.fr, LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, 99, avenue Jean-Baptiste Clément F-93430, Villetaneuse, FRANCE, 93430.

XXXX-XXXX/2021/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

We are working in the context of *fail-stop* failures. Several approaches exist to handle failures. System-level fault tolerance is transparent to the user: the distributed run-time environment implements mechanisms such as rollback recovery and the application does not need to be modified [5]. In the thesis, we are following an *application-based* approach: our goal is to provide computation kernels that can survive failures. We use the *User-Level Failure Mitigation* model [2]. Moreover, in order to make sure that the application can survive failures at any moment of the execution, we verify reliability properties of our algorithms with formal methods.

2 GENERAL DESCRIPTION

Our work focuses on adding fault-tolerant mechanisms to dense linear algebra algorithms to make them able to survive in volatile environments in spite of failures. Our work is based on communication-avoiding algorithms, in which we take advantage of properties that can be exploited to design new scalable and robust fault-tolerant algorithms. For instance, introducing redundancy of intermediate results [3, 7].

Fault-tolerant algorithms must be designed and evaluated considering how robust they are and how much computational overhead they introduce with respect to a non-fault-tolerant algorithm [4].

To model and validate the robustness and the resilience of my algorithms, we use formal methods (Coloured Petri Net model). A formal model developed in this thesis can be seen in [6]. It helps proving reliability and correct functioning of a fault-tolerant tall and skinny algorithm. To measure the cost of our fault-tolerant mechanisms on the performance, we first consider a failure-free execution with no fault-tolerance mechanism as the baseline; we measure the overhead of the fault-tolerance mechanism on a failure-free execution; last, we measure the cost of the recovery procedure by injecting a random failure during the execution. Results show that these fault-tolerant algorithms introduce very little computational overhead.

3 LU FACTORIZATION ALGORITHMS AND EXPERIMENTS

Many applications in linear algebra rely on a LU factorization, either for a tall-and-skinny matrix or on a wider, potentially square, matrix. The TSLU algorithm was designed for a *tall and skinny* input matrix (i.e. a matrix with M rows and N columns, with $M \gg N$); the data is distributed between processes along a 1D distribution, allowing each of them holding complete lines.

The first phase consists of finding *pivot rows* to improve the numerical stability of the computation. In TSLU we are using a specific algorithm, called *tournament pivoting*, in order to find the best row-pivots to factor the entire matrix at low communication cost. The *Communication-Avoiding LU* (CALU) [10] algorithm also factors a matrix as $A = LU$, taking a potentially square matrix as input. It iterates over block-column sub-matrices called *panels*. A panel is the leftmost block-column sub-matrix. Since a panel is a tall and skinny matrix, CALU uses TSLU to compute the LU factorization of each panel. CALU uses a 2D grid of processes dividing the square matrix into smaller sub-blocks and assigning each sub-block to be calculated to one process on the grid [1]. At each iteration, it takes the leftmost non-processed panel and computes its LU factorization [8, 13].

FT-TSLU and FT-CALU are the fault-tolerant versions of TSLU and CALU proposed in this research. They can recover from crash-type process failures at run-time and proceed with the computation beyond them. When an error is detected by the run-time environment, they re-spawn all the failed processes at once, repair the communicator used by the processes to exchange information and the current state on the calculation of a matrix. To achieve this last property, they keep track on the intermediate results obtained at each step backing them up on memory or on a storage device. Hence, all processes are able to share their previous known results with a process to be restored. The restoration procedure proposed in this work has been designed to allow any

process in the global communicator to detect errors at any point of the algorithm, independently from the task a process is in charge of.

We implemented our algorithms and evaluated their performance on the Grid'5000 platform. We used the Gros cluster, which has 124 nodes, featuring one Intel Xeon Gold 5220 CPU, 18 cores/CPU, 96GB of RAM, two SSDs of 447GB and a 894GB SSD, and 2 × 25Gb Ethernet NICs each. We used OpenMPI for non-fault-tolerant versins and ULFM 4.1.0u1a1 for the fault-tolerant ones. Input matrix sizes are 32k×32k, 64k×64k and 100k×100k. Failures are injected by sending a SIGKILL signal to the processes. In this case, the operating system sends closing notifications on the TCP sockets used by the run-time environment and the failures are detected immediately. In real life, this cannot happen when a failure occurs and we need to rely on a more advanced failure detection mechanism. Although there exist more realistic techniques to inject failures, we chose not to use them in order to isolate the algorithmic cost from the system cost, and evaluate the performance of our algorithms separately from some system-specific costs.

In the poster presentation we will show execution times for CALU/FT-CALU, showing that our algorithms scale satisfactorily as the number of processes increases. We will also show that non-fault-tolerant and fault-tolerant failure-free executions are similar. Thus, added fault tolerance mechanisms generate a small overhead over non-fault-tolerant algorithms, but it is minimal compared to the total execution time.

We have designed algorithms for QR and Cholesky factorizations following a similar approach, and we are currently calculating their performance evaluation.

REFERENCES

[1] E. Agullo, C. Coti, J. Dongarra, T. Hérault, and J. Langem. 2010. QR factorization of tall and skinny matrices in a grid computing environment. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. 1–11.

[2] Wesley Bland, Aurelien Bouteiller, Thomas Herault, Joshua Hursey, George Bosilca, and Jack J. Dongarra. 2012. An Evaluation of User-Level Failure Mitigation Support in MPI. In *Recent Advances in the Message Passing Interface*, Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra (Eds.). Springer Berlin Heidelberg, 193–203.

[3] C. Coti. 2016. Exploiting Redundant Computation in Communication-Avoiding Algorithms for Algorithm-Based Fault Tolerance. In *2016 IEEE BigDataSecurity, IEEE HPSC, and IEEE IDS*. 214–219.

[4] C. Coti. 2016. Scalable, Robust, Fault-Tolerant Parallel QR Factorization. In *2016 IEEE CSE and IEEE EUC and DCABES*. 626–633.

[5] C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello. 2006. Blocking vs. Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI. In *SC '06*. 18–18.

[6] Camille Coti, Laure Petrucci, and Daniel Alberto Torres Gonzalez. [n. d.]. Fault-tolerant matrix factorisation: a formal model and proof. 6th International Workshop on Synthesis of Complex Parameters (SynCoP) 2019.

[7] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. 2008. Communication-avoiding parallel and sequential QR factorizations. *CoRR* abs/0806.2159 (2008).

[8] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. 2012. Communication-optimal Parallel and Sequential QR and LU Factorizations. *SIAM J. Sci. Comput.* 34, 1 (Feb. 2012), 206–239.

[9] Jack Dongarra et al. 2011. The international exascale software project roadmap. *International Journal of High Performance Computing Applications* (2011).

[10] Laura Grigori, James W. Demmel, and Hua Xiang. 2011. CALU: A Communication Optimal LU Factorization Algorithm. *SIAM J. Matrix Anal. Appl.* 32, 4 (Nov. 2011), 1317–1350.

[11] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. 2014. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters. In *IEEE/IFIP DSN'14*. 610–621.

[12] Daniel A. Reed, Charng Da Lu, and Celso L. Mendes. 2006. Reliability challenges in large systems. *Future Generation Computer Systems* 22, 3 (1 2 2006), 293–302.

[13] Edgar Solomonik and James Demmel. 2011. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. In *Euro-Par 2011*, Emmanuel Jeannot, Raymond Namyst, and Jean Roman (Eds.). Springer Berlin Heidelberg, 90–109.