Study Towards Enhanced Performance Analysis In QR MUMPS Task-Based Sparse Factorization

Marcelo Cogo Miletto, Lucas Mello Schnorr
Institute of Informatics, Federal University of Rio Grande do Sul - UFRGS, Porto Alegre, Brazil

***Abstract*—Linear algebra solvers are commonly present in scientific computing. They help in the research process through a cheaper way of experimentation that numerical applications provide. As the problems in this context are continually growing in size and complexity, the time spent to compute these experiments increase as well. For keeping the use of these applications viable, parallel high-performance solutions are an essential tool nowadays. In this work, we study a sparse direct solver called qr_mumps. It uses the StarPU library to explore the task-based parallelism. We focus on finding information that can be used to enable better performance analysis of the application.**

I. Introduction

Many research areas depend on linear algebra fundamentals. Numerical applications used in areas such as economy, mechanics, and geophysics, continuously end up with the problem of solving systems of linear equations [1]. These applications play an essential role in the research process, they allow simulating the reality using a computational environment, thus, providing a cheaper way of experimentation from which we can validate hypothesis or make predictions.

This kind of application is characterized by intensive computational operations, demanding a lot of computing power to its executions. As these applications play an essential role in scientific computing. The linear algebra methods developed a long time ago to solve such problems, needed to be updated to face the size of the ever-increasing problems that arise in this context. Thus, to reach good performance, many solutions rely on parallel computing, exploring multiple cores and accelerator devices (GPUs) to speed up calculations providing results faster. Ondes3D [2] for example, uses a cluster to predict earthquake ground motion, and RAFEM [3] uses GPU to accelerate the simulation of a medical procedure to treat hepatic cancer.

Besides the computational intensity, most of the real problems lead to large and sparse matrices because of the form they are structured. This adds a new level of complexity, implying in the use of efficient data structure to represent the matrix and strategies to take advantage of its sparsity, saving computational effort and memory. Depending on how the solver performs operations and on the matrix pattern, all the sparsity can be lost due to the introduced fill-ins. This way, parallel implementations of both direct and iterative classes of solvers depend on the permutation or reordering of rows and columns [4] to optimize what they can get from sparsity. Parallel algorithms for solving sparse systems must consider this irregularity characteristic in the workload when partitioning the data. Otherwise, poor resource utilization can bound application performance. A good way to fight this problem is by adopting a task-based approach, breaking the problem into a set of smaller tasks and letting a runtime system responsible for their scheduling. For this, libraries like StarPU [5] can be used to handle data partitioning, providing a dynamic runtime system and different scheduling policies.

Some of these runtime scheduling policies are based on user given scheduling hints, such as the performance model of application tasks. These hints help to eliminate the source of load imbalance by using a measure to quantify the computation done by each processing unit according to task costs. Thus, it helps to distribute the workload evenly among them. This strategy is commonly used in high-performance libraries [5]. However, besides this scheduling utility, the performance model of an application can be used both to faithfully simulate the application execution to help developers on its optimization, and to compare a real execution of the application with a prediction based on the model.

The qr_mumps [6] is a task-based solver that uses the StarPU library. It already has a performance model that was implemented in [7]. As StarPU enables tracing the application behavior, what we propose in this work is to enrich the generated traces with this performance model, attaching the quantified computational cost of the application tasks in the traces generated by a real execution. This way, we can detect anomalous tasks by comparing the time one task took to execute in the real platform with the expected time given by the model, enabling a more in-depth analysis of the application characteristics.

The rest of the paper is organized as follows: Section II presents libraries and sparse solvers, the task-based programming model, details of qr_mumps and the performance model. Section III presents our motivation and work proposal describing our methodology. Finally, Section IV concludes the paper and presents future directions for this work.

II. Background

This section presents information about parallel sparse solvers, the StarPU library, going trough its programming model characteristics and listing its capabilities in terms of scheduling task workloads. Also, we provide a description of the qr_mumps concepts and implementation details, plus an overview of its performance model.

A. Parallel Sparse Solvers

With the advent of multicore and the accelerators devices, also considering them in a distributed memory system. New challenges arose in the developing of linear algebra solvers for such complex heterogeneous systems. As the developed solution should adapt a complex workload to the different speeds and capacities of the computing resources and different costs for communications, this problem becomes even harder. Load balancing, communication, memory consumption, and portability are some of the essential aspects that should be considered when developing such algorithms.

Numerous libraries and packages implement high-performance sparse solvers. They all adopt a standard set of basic routines defined by BLAS [8], which have implementations with hand-tunned code for specific devices like for example Intel MKL, AMD BLIS and openBLAS for CPUs, CUBLAS and MAGMA BLAS for GPUs. Libraries like PARDISO [9] and MAGMA [10] provides several solvers focusing both on shared and distributed memory multiprocessors and heterogeneous platforms containing multiple cores and GPUs. MAGMA uses a hybrid methodology that breaks the problem into tasks which a dynamic runtime system is responsible for scheduling. Besides these packages that offer numerous sparse solver methods, we can also look into individually developed solvers like MUMPS [11]. It was one of the first multifrontal QR solvers, and it was first developed to work on shared-memory multiprocessors, exploring the parallelism using the multifrontal method [12], based on the concept of an elimination tree [13]. This solver uses a hand-coded task queuing system to orchestrate tasks in this tree structure along with a multithreaded BLAS implementation to explore parallelism further.

Employing a task-based approach to sparse multifrontal solvers enable to handle the irregular workloads that come with the different task granularities and characteristics. The qr_mumps is a further step when compared to MUMPS. This newer solver explores more fine-grained parallelism using StarPU to handle the task-based parallelism. Trough it, various architectures can be targetted as there are the different BLAS implementations that can be used in the application, allied with the StarPU dynamic runtime system which is capable of scheduling tasks over heterogeneous platforms, using a wide range of scheduling policies.

B. Task-Based Programming Model in StarPU

The task-based programming paradigm has a simple yet expressive way to describe problems, offering a portable way to deliver performance of complex workloads over many cores [14]. The application can be described using the concept of a Directed Acyclic Graph (DAG), dividing the whole computational workload into smaller tasks. The DAG nodes represent computational tasks, and the edges between them are their dependencies. A task can only execute if all of its dependencies are satisfied. Runtime systems are responsible for managing and distributing the DAG tasks among the available processing units during execution time.

StarPU is a C/C++ task programming library for heterogeneous platforms that has a dynamic scheduling runtime system, supporting multicore CPU/GPU in both shared and distributed memory systems. Its programming model is based on the concept of a codelet, which is a piece of code that defines what instructions a given task executes. Task dependencies are described in terms of the access pattern of a task to a specified data handle, which is the structure used to partition the problem data. Additional task information like priority and performance model can be associated with tasks. This extra information can be used by some of StarPU scheduling algorithms to achieve better performance, for example, deciding which tasks are going to be executed by the CPU and the GPU. The available scheduling policies algorithms can be into two categories:

- **Non performance modelling policies**: The eager scheduler uses a central task queue from which all the workers (computational resources) get tasks to work on. The random scheduler have a task queue per worker and distribute them randomly according to workers overall performance. The work stealing have a task queue per worker. When a worker becomes idle, it steals from the most loaded queue. The local work stealing policy consider neighbor workers for stealing and also takes into account task priorities. The prio scheduler sort the tasks by priority in a central queue. Heteroprio can define different priorities for different processing units.
- **Performance model-based task scheduling policies**: The DM (deque model) policy tries to minimize tasks finishing time, this is done as soon as the tasks become available. DMDA (DM data aware) also consider the data transfers cost. The DMDAR (DMDA ready) gives privilege to tasks whose data is already available on the target device. The DMDAS (DMDA sorted) consider task priority besides the data transfer cost. The DMDASD (DMDAS decision) is similar to the DMDAS but considers priority to find the minimum completion time.

C. Multifrontal QR Factorization

Direct methods like LU, Cholesky, and QR, are preferable because of their robustness when compared to iterative methods, whose efficiency depend on the numerical properties of the input matrix [1]. Among the direct methods, the QR factorization and its variations are very popular because of the capacity to achieve high performance and its numerical robustness, especially the dense Householder QR factorization and the multifrontal QR [15]. It can be used to solve sparse systems of linear equations and the least-squares problem.

Given a sparse matrix, we can take advantage of its sparsity in the following ways: (1) economize memory not storing all of its zero values explicitly, (2) perform cheaper operations because the zero values can be skipped, and (3), parallelize operations that modify distinct nonzero subsets of the matrix. The last item is the one responsible for building the elimination

tree concept, which is the main idea behind the multifrontal technique. The elimination tree nodes hold smaller dense matrices called frontal matrices or fronts, and the tree is then traversed bottom-up, enabling the processing of different branches in parallel. This source of parallelism in the multifrontal method is called tree parallelism [12]. Other factors that influence the sparse solvers performance is the ordering applied to the sparse matrix structure. It can harshly impact the tree structure, leading to unbalanced trees in the case of the multifrontal method and can control the generated fill-in level. To treat this, applications use fill-reducing matrix ordering techniques like the Cuthill-McKee, Average Minimum Degree, and the Nested Dissection orderings [16].

The elimination tree can be seen as a DAG of tasks, where the nodes are factorization tasks, and they depend on the previous child nodes factorizations. In the classical approach, the multifrontal method is divided into two steps: assembly and factorization. The assembly task groups a set of rows that have nonzero elements in the pivot column (the one that is being eliminated) along with the other elements that are affected by the factorization, originating a front. Then, in the factorization step, this front matrix is factorized through the Householder method, producing one row of the R factor, a part of the Q factor, and the contribution block that is incorporated to its parent front.

D. QR MUMPS: Fine-Grained Task-Based Multifrontal QR

The previously detailed approach has two limitations. As we go upward the tree, the tree parallelism shrinks, and depending only on this parallelism source does not produce good speedups [17]. Also, threads cannot start processing a parent node until all of its children were factorized, limiting, even more, the parallelism and hampering reaching a good load balance. This way, qr_mumps goes further by exploring fine-grained tasks in the multifrontal method on top of StarPU, exploring efficient ways to perform the nodes factorization using 1D and 2D block factorization algorithms, allowing starting the computation of a father node concurrently with its children.

The elimination tree structure in qr_mumps is diluted in a finer-grained set of tasks. The task creation and the matrix partitioning are fully dynamic, depending on the front matrix size, the corresponding tree node is partitioned into 1D block columns or in 2D tiles. Also, it can be pruned, which means that the node or a pruned subtree will be computed sequentially by a single thread. Only nodes that represent a small portion of the factorization cost are pruned. The LAPACK-based factorization routines in qr_mumps were slightly modified by its authors to allow the user to pass a parameter defining a value for the internal block size. Helping thus to reduce the generated fill-in by the blocked operations. There are nine types of tasks that the application threads can execute:

- **init/clean front**: initialize and assemble a frontal matrix or clean it, deallocating from memory.
- **init/clean block**: initialize and clean blocks of a specific frontal matrix.
- **geqrt**: computes the QR factorization in a front matrix block, generating a set of Householder reflectors and part of the R factor.
- **gemqrt**: update the matrices block at the right of the diagonal with the previously calculated reflectors by geqrt.
- **tpqrt**: eliminate the blocks below the diagonal factorized by geqrt.
- **tpmqrt**: updates the remaining blocks in the front matrix.
- **do subtree**: define the factorization of an entire subtree of the elimination tree to be computed sequentially using the previous described routines.

To generate this fine-grained version of the elimination tree, the application performs a sequence of steps: given a sparse input matrix A, first, a fill-reducing ordering is applied, for this, it uses libraries such as SCOTCH, METIS and COLAMD. Then, a symbolic factorization step computes the generated fill-in and the structures of all the fronts to be incorporated in the tree. Pruning rules define which tree nodes will be pruned. With the structure of the tree and the fronts defined, the front factorization is partitioned into a set of tasks defined by the block sizes. Those tasks are submited following the bottom-up traversal of the tree, and just the tasks that are part of initialized frontal matrices are visible to the runtime system.

E. The Application Performance Model

Performance modeling of individual computational tasks is of great value to overcome some problems [7]. It enables to obtain trustful performance predictions using low-cost simulations, helps in application tunning, and supports complex scheduling algorithms. In the StarPU context, it is possible to emulate/simulate StarPU applications with SimGrid [18]. Also, the StarPU library supports performance model-based scheduling algorithms, which can provide better scheduling decisions by knowing the targeted architecture capabilities, communication costs, and by estimating the computational weight of each task.

Although we know that simulations are useful in many ways and because it is hard for a model to capture every factor that can affect performance, there can be differences in the predicted and measured values. This way, performance models values can be used to compare expected results with the real obtained results, highlighting anomalous tasks where some interference in a real experiment perturbed the task duration. For qr_mumps a reliable performance model was already made in [7]. For each factorization task, there is a function that is called in the moment of its creation. The function reads the data structure which the task will work on and calculate the floating point operations. This value is then associated to the task and write in the application trace.

III. Motivation and Methodology

Given the application and its performance model, we want to incorporate this estimated cost per task from the model to the StarPU generated traces for real executions. This way, we have the real execution information collected by StarPU

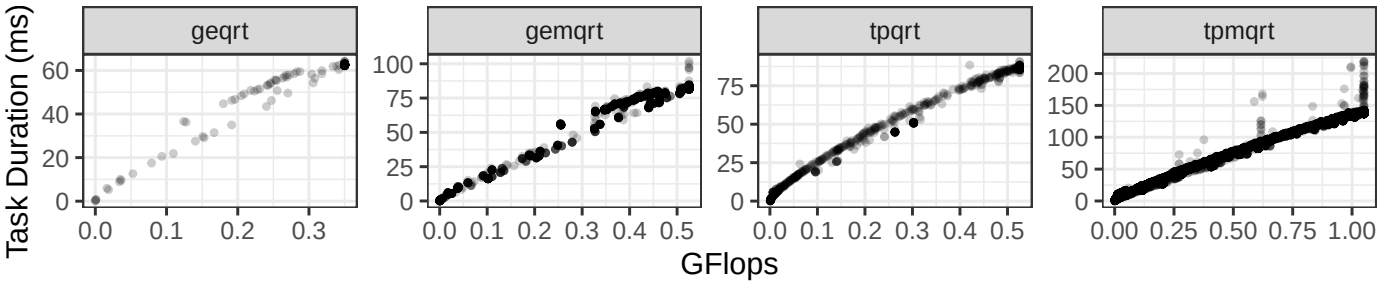

Fig. 1. Duration of tasks according to their calculated GFlops.

plus the estimated cost in floating-point operations (flops) from the model. Since the expected results are trustable, we can estimate the time that a given task should take to execute in a specific computing unit and compare it with the real execution time obtained in the trace. This extra information allows us to detect anomalous tasks in a real execution. By doing so, we can enhance existing application behavioral visualizations like in StarVZ [19].

To achieve our goal of detecting anomalous tasks, we need to combine the information of the expected flops and the task execution time from an execution trace. A model of analysis of variance (ANOVA) can be used to assess how far a real task execution is from what the model predicted. Having this data, we can use visualization techniques to highlight tasks that have a duration that was not expected by the performance model. The Figure 1 show the relation between task duration and its computational weight. We can see that there is a linearly increasing pattern, but some points lie outside of that line. That means that for a similar amount of flops that a task have, there are different duration times.

IV. Conclusion

This work presented an initial description of the qr_mumps application concepts and its performance model. Our study aims to use the model information included into the execution traces generated by the StarPU library. With this new information on the traces, we can enhance application visualization tools to detect anomalous tasks which can lead to further investigations.

Future work needs to explore the floating-point operations values integrated into the StarPU tracing system and work on developing and using visualization techniques to highlight identified anomalous tasks. For the classification of these tasks, an ANOVA model can be employed. With the identification of anomalous tasks, a more in-depth analysis can be made to assess the reason that it presented this abnormal behavior.

References

[1] F. Lopez, "Task-based multifrontal qr solver for heterogeneous architectures," Ph.D. dissertation, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2015.

[2] R. Keller Tesser, L. Mello Schnorr, A. Legrand, F. C. Heinrich, F. Dupros, and P. O. Navaux, "Performance modeling of a geophysics application to accelerate over-decomposition parameter tuning through simulation," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 11, p. e5012, 2019.

[3] M. C. Miletto, "Acelerando uma aplicação de simulação computacional para o processo de ablação por radiofrequência usando gpu," 2018.

[4] Y. Saad, *Iterative methods for sparse linear systems*. siam, 2003, vol. 82.

[5] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[6] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez, "Multifrontal qr factorization for multicore architectures over runtime systems," in *European Conference on Parallel Processing*. Springer, 2013, pp. 521–532.

[7] L. Stanisic, E. Agullo, A. Buttari, A. Guermouche, A. Legrand, F. Lopez, and B. Videau, "Fast and accurate simulation of multithreaded sparse linear algebra solvers," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2015, pp. 481–490.

[8] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "An extended set of fortran basic linear algebra subprograms," *ACM Transactions on Mathematical Software (TOMS)*, vol. 14, no. 1, pp. 1–17, 1988.

[9] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with pardiso," *Future Generation Computer Systems*, vol. 20, no. 3, pp. 475–487, 2004.

[10] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "Accelerating numerical dense linear algebra calculations with gpus," *Numerical Computations with GPUs*, pp. 1–26, 2014.

[11] P. R. Amestoy, I. S. Duff, and J.-Y. L'excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Computer methods in applied mechanics and engineering*, vol. 184, no. 2-4, pp. 501–520, 2000.

[12] I. S. Duff and J. K. Reid, "The multifrontal solution of unsymmetric sets of linear equations," *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 3, pp. 633–641, 1984.

[13] J. W. Liu, "The role of elimination trees in sparse factorization," *SIAM journal on matrix analysis and applications*, vol. 11, no. 1, pp. 134–172, 1990.

[14] J. Dongarra, S. Tomov, P. Luszczek, J. Kurzak, M. Gates, I. Yamazaki, H. Anzt, A. Haidar, and A. Abdelfattah, "With extreme computing, the rules have changed," *Computing in Science & Engineering*, vol. 19, no. 3, p. 52, 2017.

[15] A. Buttari, "Fine-grained multithreading for the multifrontal qr factorization of sparse matrices," *SIAM Journal on Scientific Computing*, vol. 35, no. 4, pp. C323–C345, 2013.

[16] G. H. Golub and C. Loan, "Matrix computations, forth edition," 2013.

[17] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[18] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[19] V. Garcia Pinto, L. Mello Schnorr, L. Stanisic, A. Legrand, S. Thibault, and V. Danjean, "A visual performance analysis framework for task-based parallel applications running on hybrid clusters," *CCPE*, vol. 30, no. 18, p. e4472, 2018.