Enhancing sparse direct solver scalability through runtime system automatic data partition

Alycia Lisito, Mathieu Faverge, Grégoire Pichon, Pierre Ramet

▶ **To cite this version:**

Alycia Lisito, Mathieu Faverge, Grégoire Pichon, Pierre Ramet. Enhancing sparse direct solver scalability through runtime system automatic data partition. WAMTA 2024 - Workshop on Asynchronous Many-Task Systems and Applications 2024, Feb 2024, Knoxville, United States. pp.105-110, 10.1007/978-3-031-61763-8_10 . hal-04527103

HAL Id: hal-04527103

https://inria.hal.science/hal-04527103v1

Submitted on 3 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Enhancing sparse direct solver scalability through runtime system automatic data partition.

Alycia Lisito[1], Mathieu Faverge[1], Grégoire Pichon[2], and Pierre Ramet[1]

[1] Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400 Talence
[2] Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France
{firstname.lastname}@inria.fr

Abstract. With the ever-growing number of cores per node, it is critical for runtime systems and applications to adapt the task granularity to scale on recent architectures. Among applications, sparse direct solvers are a time-consuming step and the task granularity is rarely adapted to large many-core systems. In this paper, we investigate the use of runtime systems to automatically partition tasks in order to achieve more parallelism and refine the task granularity. Experiments are conducted on the new version of the PaStiX solver, which has been completely rewritten to better integrate modern task-based runtime systems. The results demonstrate the increase in scalability achieved by the solver thanks to the adaptive task granularity provided by the StarPU runtime system.

Keywords: Sparse Direct Solver · Task scheduling · Data partitioning · Runtime systems.

1 Introduction

In this paper, we are interested in solving linear systems of the form $Ax = b$ where the matrix A is usually large and sparse. This kind of problem appears in many scientific applications, such as electromagnetism or computational fluid dynamics, and is in general one of the most expensive operations in numerical models. In order to solve this problem, a wide range of methods is available: direct, iterative and hybrid methods. In this work, we will focus on direct methods, which are more time and memory consuming, but provide more robust solutions. The objective of direct methods is to factorize A into LL^h, LDL^h or LU with L, D and U respectively unit lower triangular, diagonal and upper triangular matrices, depending on the numerical properties of the problem.

The common approach to solve a sparse system is divided into four main steps: 1) ordering of the unknowns, 2) block-symbolic factorization, 3) numerical factorization, and 4) triangular system solves. The two initial steps focus on computing a blocked representation of the factorized matrix to take advantage of efficient Level-3 BLAS operations [6] while minimizing the computational complexity of the last two steps. Therefore, these first two steps determine the granularity of the computation without taking into account the target architectures, which may have an impact on performance and/or scalability. There are

two ways to address this issue: using parallel implementation of the computational kernels to delegate the optimization to a third-party library, or splitting the workload internally to feed the computational resources. In this paper, we focus on the latter to work with one task per core and use a runtime system to schedule the tasks. We show how the use of an external task-based runtime system such as StarPU [2] helps to easily divide the workload without adding complexity to the original algorithm.

Section 2 describes the global approach of sparse direct solvers and discusses the different levels of task granularity that can be expressed in terms of code complexity in such an algorithm. Section 4 shows some preliminary results of the new implementation of the PaStiX solver with the internal scheduler, and with the help of an external runtime system. Section 5 presents related work in other sparse direct solvers. Finally, Section 6 concludes and presents future work.

2 Task-based sparse factorization

Sparse direct solvers are by nature very similar to dense solvers to perform LL^h *(Cholesky)*, LDL^h, or LU factorization. They can simply be summed-up by the classic three nested loop presented in Figure 1, and the difference lies in the non continuous range of the inner loops due to the sparsity. When one wants to apply task-based parallelism to these algorithms, three choices, from large to fine, naturally appear.

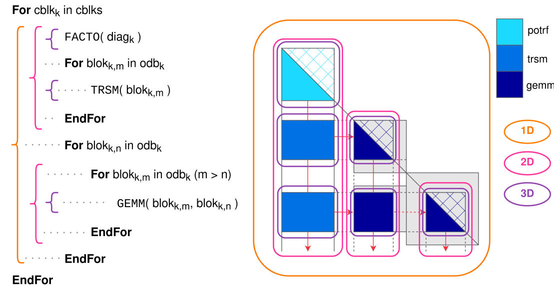


Fig. 1: Pseudo code of the factorization algorithm and visual representation of a factorization iteration with the *1D*, *2D* and *3D task level* represented in orange, pink and purple respectively.

First, one can define a task as being a complete iteration of the first loop level, *1D task level*, as shown by the orange brace in Figure 1. This kind of task distribution works well when the top left diagonal block to factorize is small because it groups together many small updates (*gemm*) and it allows using multi-threaded BLAS for large tasks. This is close to what is commonly used in sparse solvers using multi-frontal algorithms. Sparse solvers based on the multifrontal approach commonly use this task level as frontal matrices compact contributions

into a temporary contiguous buffer, allowing better parallelization for additional memory consumption. One of the problems here is also that due to the large scattering of the contributions, it requires a fine protection mechanism to allow multiple tasks to be computed in parallel. It also presents a large number of data dependencies that are not suitable for all sequential task flow runtime systems (StarPU [2], OpenMP [3], PaRSEC-DTD [8]. . .).

Then, to create more parallelism, some implementations consider the task granularity to be defined by the second loop level, *2D task level*, shown in pink in Figure 1. This creates two types of tasks, those that factorize a panel (set of columns) of the matrix, and those that perform panel updates (one per off-diagonal block). This solution has nice advantages: the number of data dependencies is fixed for all tasks (1 or 2 panels), so it is suitable for a task-based runtime system, and the computation of the number of contributions made to each panel is straightforward. However, at the end of the factorization, the problem becomes dense, and as it has been shown in many dense linear algebra libraries, this can be insufficient to achieve a good level of performance. It is also a problem when used on the small panels at the beginning of the factorization due to the overhead it generates with respect to the computational tasks.

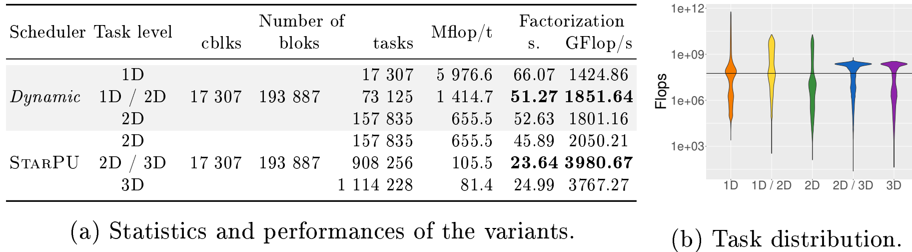
Finally, the task-based approach of dense linear algebra can be applied and tasks can be defined at the inner loop level, *3D task level*, shown in purple in Figure 1. This model is well suited when the solver needs to manage its own parallelism and use sequential tasks to reduce iteration synchronization as much as possible. As before, the number of data dependencies per task is limited, making it well suited for task-based runtime systems, and the task granularity can be properly adapted to many-core architectures on the final large blocks. However, in the case of sparse systems, the extremely large number of tasks can flood the runtime system or make it difficult and expensive to manage with a dedicated scheduler.

3 Implementation within the PaStiX solver

The PaStiX solver implements the LL^h *(Cholesky)*, LDL^h, and LU factorizations, and has been completely rewritten in its version 6 to improve modularity and support for external runtime systems. The numerical factorization is currently implemented using the first two levels of task parallelism, and in this paper we propose to study the use of the StarPU runtime systems to enable the third level of parallelism when needed.

PaStiX already offers a choice between two solutions: a *Dynamic* internal scheduler and a version built on top of the StarPU runtime system. The *Dynamic* scheduler uses a mixture of the first two levels *1D* and *2D* to achieve the best computational efficiency with enough parallelism to serve a large number of cores. For each factorization task, the scheduler checks the size of the update operation. If it is larger than a given threshold, the update is split and submitted with the *3D task level* scheme to be performed in parallel by other threads, otherwise the operation is performed immediately by the current working thread.

This limits oversized tasks without creating too many microtasks. However, the task size can still be too large, especially for the GEMM operation. The StarPU version uses the *2D task level* algorithm by default due to the complexity of managing the large set of data dependencies in *1D*. In this paper, we have extended this version to use StarPU's automatic data partitioning to divide panels into blocks to switch for a *3D task level* algorithm. Note that we could have done this without this feature, but by using it we can dynamically select the best strategy in the same way as the *Dynamic* scheduler. In fact, based on a user threshold, the panel is split or not in order to use the best task granularity, StarPU then automatically handles the data dependencies to scatter/gather the data as needed for the different operations.



Scheduler	Task level	cblks	Number of bloks	tasks	Mflop/t	Factorization s.	GFlop/s
	1D			17 307	5 976.6	66.07	1424.86
Dynamic	1D / 2D	17 307	193 887	73 125	1 414.7	**51.27**	**1851.64**
	2D			157 835	655.5	52.63	1801.16
	2D			157 835	655.5	45.89	2050.21
StarPU	2D / 3D	17 307	193 887	908 256	105.5	**23.64**	**3980.67**
	3D			1 114 228	81.4	24.99	3767.27

(a) Statistics and performances of the variants.

(b) Task distribution.

Fig. 2: Statistics of all the variants for the *Cube_ Coup_ dt0* matrix factorization on 4 `bora` nodes.

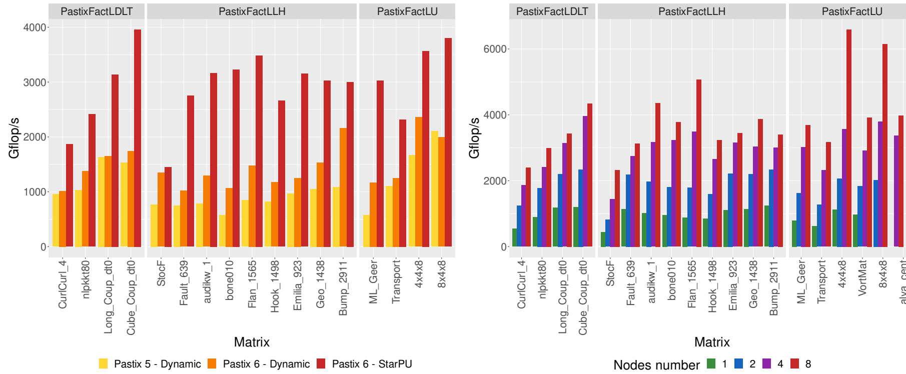
Figure 2 illustrates the effect of the different strategies. As expected, adding extra levels of tasks reduces the average number of flops per task and also the maximum number of flops per task. Figure 2a illustrates the fact that mixed 2D/3D and 2D strategies drastically reduce the number of tasks above the defined threshold. Here we target tasks of similar size to those used in dense linear algebra on CPU cores, which corresponds to dense matrix multiply of 384×384 or $56, 6$MFlops. On the other hand, it can be observed that increasing the task level drastically increases the number of tasks in the systems, which leads to an overload of both the *Dynamic* scheduler and the StarPU runtime system, and too many fine-grained tasks that degrade performance.

4 Experiments

Experiments were conducted on the `bora` cluster of the *Plafrim*[3] platform. Each `bora` node is equipped with two Intel `CascadeLake` 18-cores running at 2.60 GHz and 192 GB of memory. The environment was handled with GUIX and used: Intel MKL 2020 for BLAS kernels, OpenMPI `4.1.5`, StarPU `1.4.3`, and Scotch `7.0.1` to perform the ordering of the unknowns.

[3] https://www.plafrim.fr

The experiments are based on PaStiX `6.3.2` available on the public git repository [4] and used a set of 19 matrices issued from The SuiteSparse Matrix Collection [5] ranging from $600K$ to $10M$ unknowns and requiring from 3.64 to 636 TFlops for the factorization.



(a) Comparison of the versions studied on 4 nodes.

(b) Scalability study with the StarPU 2D/3D algorithm.

Fig. 3: Performance study of the factorization performance on the `bora` cluster with a set of 19 matrices.

Figure 3a compares the performance of the *Dynamic* scheduler in former and new releases of PaStiX with the proposed StarPU implementation. All versions use mixed strategies. While the new modular version of PaStiX slightly improves the performance, the proposed StarPU strategy largely increases the performance, especially in distributed, thanks to the extra level of parallelism exposed. This result is also illustrated by the scalability of the StarPU version shown in Figure 3b that reaches an almost perfect speedup of 2.01 to 3.94 on 4 nodes, while keeping a very high scalability on 8 nodes despite the rather small problem sizes targeted.

5 Related work

Other sparse direct solvers have also tried to solve the granularity vs. task size problem with their own solutions. In [9], they work on the CHOLMOD solver to propose an approach similar to the one proposed in this paper to create more or less parallelism. However, this solver relies on a left-looking approach that focuses on parallelizing dot-product operations, while we have a right-looking algorithm that parallelizes an outer-product operation. symPACK [4] is the closest work as it uses a similar 2D/3D task scheme with an internally developed

[4] https://gitlab.inria.fr/solverstack/pastix

UPC++ scheduler targeting high performance communication for distributed systems. However, it focuses only on symmetric problems. In [1], they study a similar strategy on QR sparse factorization to partition the larger tasks and introduce parallelism in the QR kernels. This work was a precursor to automatic partitioning in StarPU and used manual data partitioning.

6 Conclusion

In this paper, we have shown that we can use a task-based runtime system to automatically refine the task granularity at low cost to improve performance by a factor of 2 to 4. However, this strategy, when pushed to its limit, generates a large number of tasks that may be difficult to process efficiently by the scheduler. In a future work, we plan to explore the possibility given by StarPU's hierarchical tasks [7] to take dynamically the decision to refine, or not, the granularity. Additionally, it would help to delay the submission of fine-grained tasks to reduce the overhead of the scheduler. The final results will be studied with respect to other sparse direct solvers and algorithm variants.

References

1. Agullo, E., Buttari, A., Guermouche, A., Lopez, F.: Multifrontal qr factorization for multicore architectures over runtime systems. In: Euro-Par 2013 Parallel Processing: 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings 19. pp. 521–532. Springer (2013)
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience **23**(2), 187–198 (2011)
3. Ayguade, E., Copty, N., Duran, A., Hoeflinger, J., Lin, Y., Massaioli, F., Teruel, X., Unnikrishnan, P., Zhang, G.: The design of openmp tasks. IEEE Transactions on Parallel and Distributed Systems **20**(3), 404–418 (2009). https://doi.org/10.1109/TPDS.2008.105
4. Bellavita, J., Jacquelin, M., Ng, E.G., Bonachea, D., Corbino, J., Hargrove, P.H.: sympack: A gpu-capable fan-out sparse cholesky solver. In: Proceedings of the SC'23 Workshops. p. 1171–1184. New York, NY, USA (2023)
5. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (Dec 2011). https://doi.org/10.1145/2049662.2049663
6. Dongarra, J., Croz, J.D., Hammarling, S., Duff, I.S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**(1), 1–17 (1990)
7. Faverge, M., Furmento, N., Guermouche, A., Lucas, G., Namyst, R., Thibault, S., Wacrenier, P.a.: Programming Heterogeneous Architectures Using Hierarchical Tasks. Concurrency and Computation: Practice and Experience **35**(25) (2023). https://doi.org/10.1002/cpe.7811, https://hal.science/hal-04088833
8. Hoque, R., Herault, T., Bosilca, G., Dongarra, J.: Dynamic task discovery in parsec: a data-flow task-based runtime. In: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems. ScalA '17 (2017)
9. Le Fèvre, V., Usui, T., Casas, M.: A selective nesting approach for the sparse multi-threaded cholesky factorization. In: 2022 IEEE/ACM 7th Intl. Workshop ESPM2. pp. 1–9 (2022)