SORBONNE UNIVERSITÉ

PSFPN

Benchmarking of linear solvers : link between simulation and high performance computing

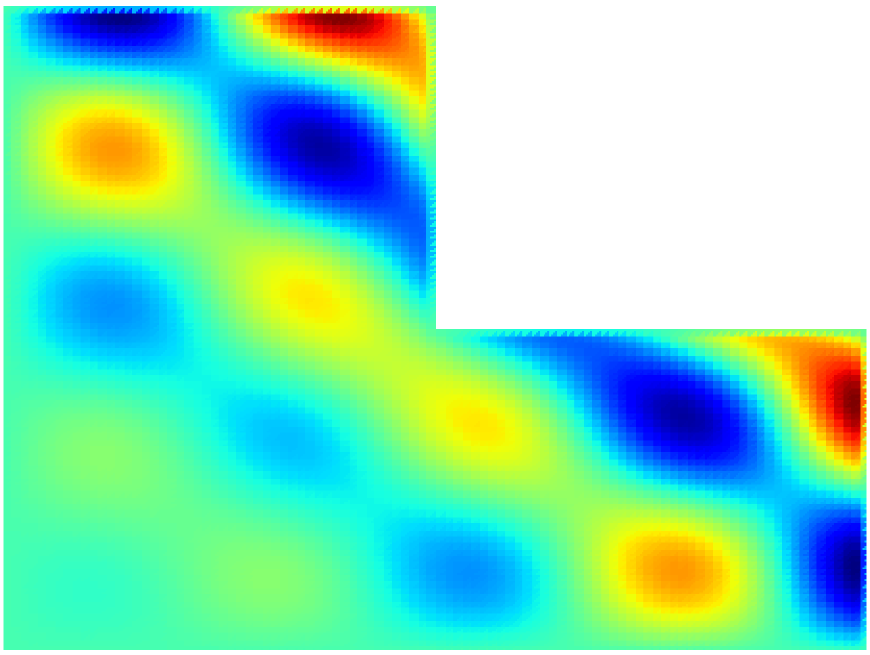


Figure 1: illustration of solving a 2D Poisson equation

Méline TROCHON
Anatole VERCELLONI

Supervisors:
Théo Mary and Pierre Jolivet

février-mai 2023

Contents

1 Introduction **2**

2 Sparse linear system solvers **2**

2.1 Linear system . . . 2
2.2 The Poisson equation . . . 2
2.3 Sparse systems . . . 3
2.4 Residual . . . 3
2.5 The matrices . . . 3

3 MUMPS: MUltifrontal Massively Parallel Solver **3**

3.1 What is MUMPS? . . . 3
3.2 How does MUMPS works? . . . 3
3.2.1 Analysis phase . . . 4
3.2.2 Factorization phase . . . 4
3.2.3 Solution phase . . . 4
3.3 Format . . . 5
3.4 Symmetry . . . 5

4 Benchmarking **5**

4.1 The matrices . . . 6
4.2 ICNTL(7): Symmetric permutation . . . 6
4.2.1 Poisson 2D equation . . . 6
4.2.2 Poisson 3D equation . . . 7
4.3 Block Low-Rank (BLR) ICNTL(35) and CNTL(7) . . . 7
4.3.1 Poisson2D equation . . . 8
4.3.2 Poisson3D equation . . . 8
4.4 Iterative Refinement ICNTL(10) and CNTL(2) . . . 9

5 How we managed to benchmark **10**

6 Conclusion **10**

7 Bibliography **10**

1 Introduction

The theme of this project covers many possibilities for benchmarking. Therefore, we had to decide what we wanted to focus on. A big part of this project was to research, with the help of our supervisors, what could be interesting for testing. We finally ended up with a project dedicated to the MUMPS solver on concrete problems, solving a Poisson equation in 2D and 3D.

2 Sparse linear system solvers

2.1 Linear system

Let us consider the following problem:

$$\begin{cases} find\, x \in \mathbb{C}^n \, s.t. \\ Ax = b \end{cases} \tag{1}$$

where $A \in \mathbb{C}^{n \times n}$, $b \in \mathbb{C}^n$, $n \in \mathbb{N}$

Solving the linear system (1) is a problem that appears in many fields. Indeed, it is very common that in order to find the best solution to a problem, it is necessary to solve a linear system. This is why scientists need efficient and precise methods and tools.

To illustrate this idea, all along this project, we will focus on physics problems, in particular Poisson's equation which is widely used (electrostatics, fluid mechanics...).

2.2 The Poisson equation

In this part, we will briefly present the method to solve a 2D Poisson equation with the Jacobi's method. The equations that we will study in this report follow the same idea.

$$\begin{cases} find\, \varphi \in H^1(\Omega) \, s.t. \\ -\partial_x^2 \varphi - \partial_y^2 \varphi = f \, on \, \Omega \\ \varphi = g \, on \, \partial\Omega \end{cases} \tag{2}$$

Thanks to Taylor's expansion, we have :

$$\partial_x^2 \varphi = \frac{\varphi_{i-1,j} - \varphi_{i+1,j} + 2\varphi_{i,j}}{h^2} \qquad with\, i, j = 1, .., n$$

and

$$\partial_y^2 \varphi = \frac{\varphi_{i,j-1} - \varphi_{i,j+1} + 2\varphi_{i,j}}{h^2} \qquad with\, i, j = 1, .., n$$

At the end, we can write (2) as the following linear system:

$$A\varphi = h^2 f \tag{3}$$

with A = $\begin{pmatrix} 4 & -1 & 0 & \cdots \\ -1 & 4 & -1 & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & -1 & 4 \end{pmatrix}$ and f = $\begin{pmatrix} f_{0,0} & f_{1,0} & f_{2,0} & \cdots \\ f_{0,1} & f_{1,1} & f_{2,1} & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ f_{0,n} & \cdots & f_{n-1,n} & f_{n,n} \end{pmatrix}$

We can notice that A has a particular shape, i.e. most of its coefficients are located on the tridiagonal part. In particular, it is a sparse, diagonally dominant and symmetric matrix with low rank properties. This is a form of matrix that appears in many problems, so we want to understand how to use a solver that is as good as possible for this type of problem. We will see later that all these properties on the matrix are useful if we want to use a solver. Indeed, by taking them into account, we can get better results. We can also notice that for the 3D Poisson equation, we have similar results but we obtain in the end a denser matrix, which explains the differences with the 2D matrix.

We have chosen for this project to benchmark with the MUMPS solver. It is one of the most used for this type of application.

Low rank matrices are important for the BLR (BLocks low rank) (we will explain this below).

2.3 Sparse systems

As remarked previously, we can notice that A is a sparse matrix. And because we are talking about benchmarking with a solver for sparse systems, we have to say few words about the behaviour in sparse systems.

We will see later how MUMPS deals with that but it follows this idea (the multifrontal method):

We denote A_{pre} the preprocessed matrix.

1. compute A_{pre}
2. factorize A_{pre}
3. solve $A_{pre}x_{pre} = b$ then find x from x_{pre}

We will detail a bit more these steps for MUMPS later.

Then, we also want to talk a bit about the complexity in sparse systems to discuss after the results that we will obtain.

In our case, we can assume that the number of non-zero elements nnz by row is constant. We should expect that, like in dense linear algebra, the complexity of step 2. (which resemble to the Gaussian elimination) will be of a higher order than linear $(O(n^\alpha), \alpha > 1)$. However, for the step 1, we are expecting something more linear.

2.4 Residual

In this report, we will talk about the accuracy of our results, we have to first define what is the meaning of that.

Let us denote x the computed solution of (1). Let us define the scaled residual:

$$r = \frac{\|Ax - b\|_\infty}{\|A\|_\infty \|x\|_\infty} \tag{4}$$

The numerator part is the classic residual, and we scale it with the norm of A and x because otherwise the error depends also of the size of A and x.

For all the tests that we did with MUMPS, we implemented our own scaled residual (MUMPS has one). Indeed, we used the single and the double precision (smumps and dmumps). So to compare properly these results, we needed a scaled residual computed after MUMPS.

2.5 The matrices

In this section we will talk a little about how we managed to find matrices for the benchmarks. First of all, we implemented a C program to build a basic tridiagonal matrix. However, we made the decision to choose matrices given by our supervisors or on the web because building more interesting large sparse matrices would be another project in itself. We implemented a loader to switch from Matrix market format and matlab matrices to the format required by MUMPS.

3 MUMPS: MUltifrontal Massively Parallel Solver

3.1 What is MUMPS?

The MUltifrontal Massively Parallel Solver (MUMPS) is a software that solves large sparse linear system. The software implements the multifrontal method, which is a version of Gaussian elimination for large sparse systems of equations. It is written in FORTRAN but it also has the possibility to be used with the C language which is what we did.

3.2 How does MUMPS works?

In the langage C, MUMPS is implemented in a function, it can be used for double or single floating points and double or single complex points.

MUMPS is separated in three phase :
- The analysis phase
- The factorization phase
- The solve and check phase

Each phase has its own parameters and functions that we will explicit below.

3.2.1 Analysis phase

The analysis phase is the most important one for sparse system's solver. Indeed, if we process the factorization without this preprocessing step, we could lose a lot of the sparsity (this phenomenon is called fill-in) of A which is disappointing. That is why MUMPS does the symbolic factorization which consists in simulating a factorization and predict the fill-in that will occur. And then, thanks to heuristic methods (otherwise, searching for the minimiser is a NP-complete problem) we are able two compute A_{pre} such that the fill-in is reduced.

We can see how one of this reordering method works on the following figure (available here)

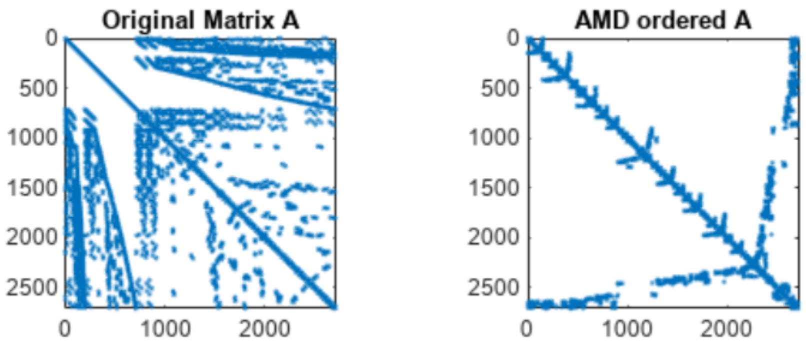


Figure 2: Illustration of a reordering method (AMD) to reduce the fill-in

So we compute A_{pre} such that:

$$A_{pre} = P.D_r.A.Q_c.D_c.P^T$$

Where P is a symmetric permutation matrix, Q_c is a permutation matrix to create a zero-free diagonal, D_r and D_c are row and column scaling matrix. Then, an elimination tree is created for A_{pre} that will be solved during the second phase.

At the end of the analysis phase, we are able to estimate the number of operations and memory necessary for factorization and solution.

3.2.2 Factorization phase

This is the phase where the computation takes part. Since we know that the factorization for dense matrix is very slow, we can suppose this should takes the huge amount of time during our tests.

During factorization phase, a factorization of the matrix is computed (LU or LDL^T) depending on the symmetry of A_{pre}. The factorization works using the multifrontal method which is a method to compute the elimination tree created during the analysis phase, but we will not go further in detail in this project.

3.2.3 Solution phase

During the solution phase, we compute the solution x_{pre} of

$$LUx_{pre} = b_{pre} \qquad or \qquad LDL^T x_{pre} = b_{pre}$$

Let us denote $y := Ux_{pre}$ or $y := L^T x_{pre}$
Then, we can compute x_{pre} in a forward elimination:

$$Ly = b_{pre} \qquad or \qquad LDy = b_{pre} \tag{5}$$

and then a backward elimination:

$$Ux_{pre} = y \qquad or \qquad L^T x_{pre} = y \tag{6}$$

Finally we postprocess x from x_{pre}.

3.3 Format

MUMPS is a solver created for sparse matrices. That is why, to use MUMPS, we have to use sparse matrix format. There exists a lot of formats for sparse matrices, but in most of the cases, we are using a COO format (In fact, there is a specific parameter for the format implemented in MUMPS).

Let $A \in \mathbb{C}^{n \times n}$ be a sparse Matrix with nnz non-zero elements $nnz \in \mathbb{N}$, we can write A with 3 arrays: ICN, JCN and V such that:

$A_{ICN[i],JCN[i]} = V[i], \qquad \forall i \leq nnz$

We can illustrate this by the following example:

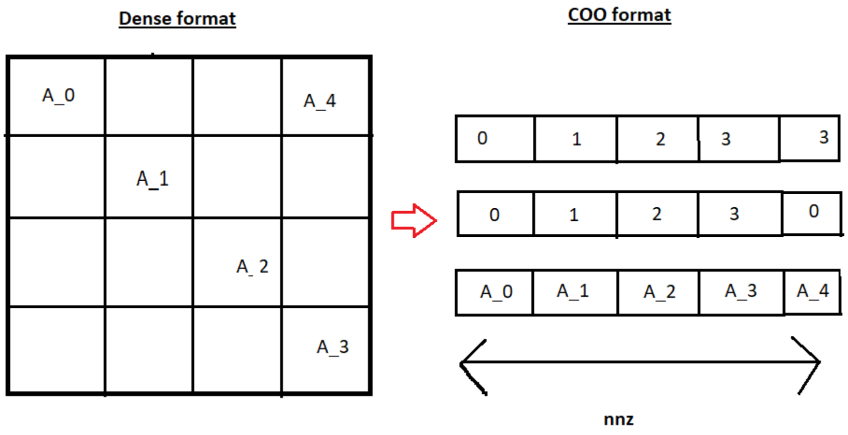


Figure 3: The format used in MUMPS

So we only need, for the memory, to store 3 * nnz elements, which needs way less memory space than the dense format (since A is sparse). So we have a really important memory gain for sparse matrices.

3.4 Symmetry

Symmetric matrices have a lot of particular properties which can be used to reduce the memory and the computational time (like Cholesky or LDL^T method).

Therefore MUMPS has a parameter SYM. It is an integer which needs to be filled by the user by 0, 1 or 2 for general matrices, symmetric postive definite (SPD) matrices or general symmetric matrices.

A particularity of using 1 or 2 for SYM is that the user has to give only once a element and its transpose (which has the same value A_i by definition of the symmetric matrices).

4 Benchmarking

For this project, we mostly focused on finding the best ways to resolve, for a certain precision, the 2D and 3D Poisson equation problem (1.2) with the minimum amount of time and space.

To do so, we used MUMPS for single precision (smumps) and double precision (dmumps). We also computed, after MUMPS has been executed, the scaled residual defined previously (1.4).

Then, we tried several parameters of MUMPS and we searched for the best results regarding the time and memory space.

MUMPS can only be called for three different functions. (The other jobs are either a combination of the first jobs, either a save and restore feature). But, since sparse matrices can have a lot of different properties, and depending on what we want to focus on. MUMPS has in total 38 integers control parameters (ICNTL) and 7 real/complex control parameters (CNTL). This can lead, if well settle, to pretty good results, or to the contrary, it can also prevent the users from getting what they want.

For this project, we will essentially focus on the sequential parameters that are useful for our problem.

4.1 The matrices

In this part, we present the matrices that we are using
The 2D Poisson equation matrix:
(1002001×1002001) with 7006001 nnz
The 2D Poisson equation matrix:
(1560896×1560896) with 23091886 nnz

What we have to said and it is very relevant for the following parts is that these two matrices have the same order of n but the 3D one have way more non-zero elements. That implies that the 3D matrix is a harder problem in comparison to the 2D.

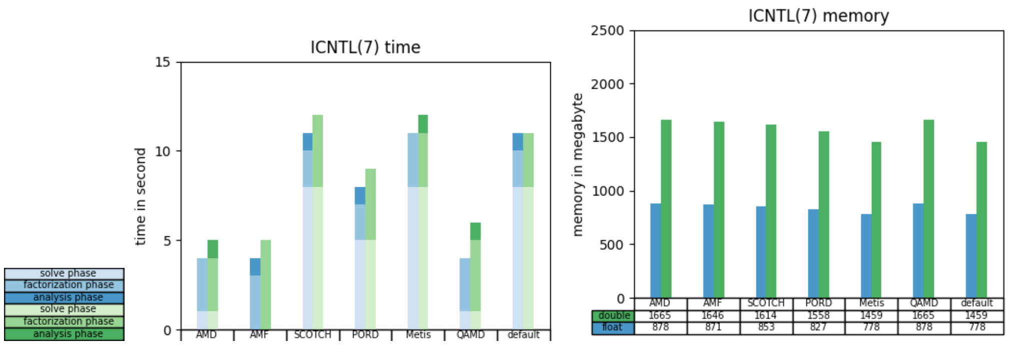
4.2 ICNTL(7): Symmetric permutation

This parameter is used in the analysis phase, it is the matrix P that we saw. It defines how the ordering of the pivots is going to be done in the preprocessed matrix.

In the following figures, we measured, on the left, the computational time during the 3 phases. And, on the right, the memory allocated during the solve phase.The blue is for the single precision and the green for the double one. All this depending on the permutation matrix.

In MUMPS, there is 8 possibilities for computing P (ICNTL(7)). We did not consider the case ICNTL(7) = 1 because it is a manual way of building P. As we said before, all these possibilities are heuristics more or less elaborated and that is why it is interesting to have a look on the results. The parameter ICNTL(7) = 7 is the automatic choice which is made depending on what is installed on the machine. If everything is present, we observe that this parameter is set to 5 which is Metis. PORD, Metis and SCOTCH are software packages and AMD (Approximate minimum degree), AMF (Approximate minimum fill) and (QAMD) Approximate Minimum Degree with automatic quasi-dense row detection are methods (a priori less elaborated).

4.2.1 Poisson 2D equation



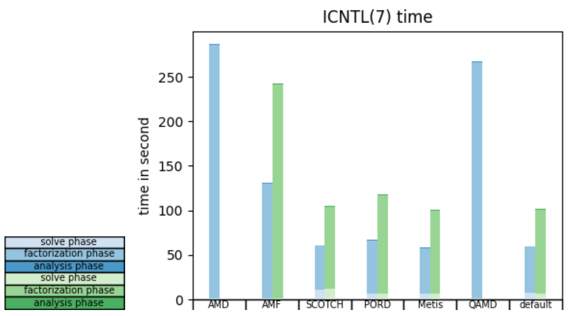
	AMD	AMF	SCOTCH	PORD	Metis	QAMD	default
double	1665	1646	1614	1558	1459	1665	1459
float	878	871	853	827	778	878	778

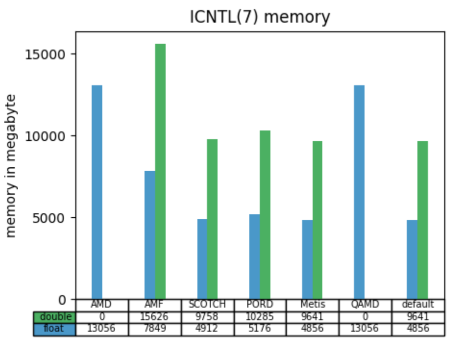
Figure 4: Time then Memory depending on the choice of ICNTL(7) for the 2D Poisson equation

We can observe plenty of things. First of all, we can notice that the default parameter is not the better one and more generally that the "elaborated software" are worst. We have more than a factor 2 between AMD and metis. For the memory, software are better, but the difference is not that big. A first conclusion on that would be that the user lose a lot of time if he lets MUMPS setting this parameter.
However, we can explain such a difference. Indeed, if we look at the time for each phase, we can see that the software take a lot of time during the analysis phase and quite the same amount of time (a bit less) during the factorization phase. In fact, they "invest time" in the analysis phase to reduce the factorization phase. And, it is a good way to reduce the computational time because as we said before the complexity is of an inferior order during the analysis phase. In this case, it is not really good because the matrix used is too small. Kind of the same can be said for the memory part..

So we will test our hypothesis on the poisson 3D equation matrix in the next part.

4.2.2 Poisson 3D equation





	AMD	AMF	SCOTCH	PORD	Metis	QAMD	default
🟩 double	0	15626	9758	10285	9641	0	9641
🟦 float	13056	7849	4912	5176	4856	13056	4856

Figure 5: Time then Memory depending on the choice of ICNTL(7) for the 3D Poisson equation

First thing to say with this new test is that, for the AMD and the QAMD way of computing P(for the double precision), the test did not work because of a too large amount of memory requested. That is why there are empty spaces. And this leads to our conclusion, the idea said before seems exact. Indeed, here the factorization phase is so important that the analysis phase is barely visible. As a consequence, in this situation it is the opposite and the software are well better. For the memory, it starts to be not possible to use the methods.

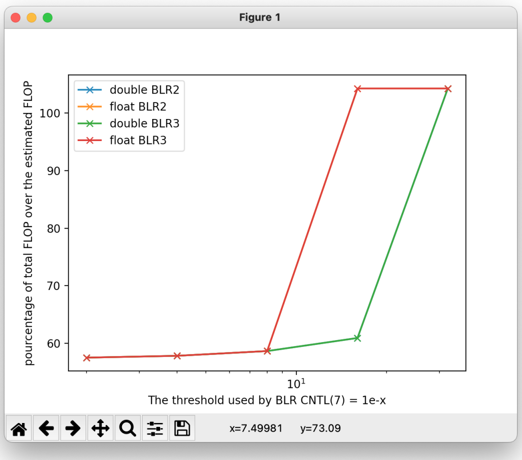
Finally, we can say that MUMPS has been design to solve large system that is why the default parameter is suited for that. However, in the particular case of our 2D matrix we would like more to modify these parameters.

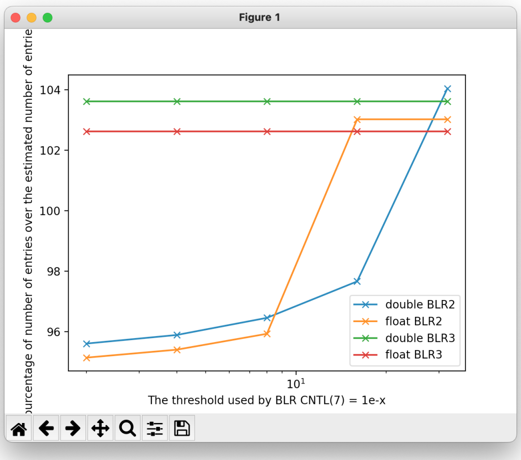
4.3 Block Low-Rank (BLR) ICNTL(35) and CNTL(7)

BLR is a feature created for low-rank matrices, it divides the matrix A to small dense matrices. Then by using a truncated QR and column pivoting, it compresses the BLR blocks. It also takes as parameter a threshold ε and after computing the compressed blocks, it suppresses any value smaller than ε. So, this feature should, for a big enough ε use less memory and also maybe takes a smaller amount of time (since the number of operations is directly influenced by the number of values).

ICNTL(35) takes 4 differents values :
- 0 does not use BLR.
- 3 uses BLR only in the factorization phase (so there is no memory gain since we cannot keep the compressed blocks).
- 2 uses BLR in the factorization and the solve and check phase.
- 1 is the automatic choice (which uses almot everytime 2).

4.3.1 Poisson2D equation





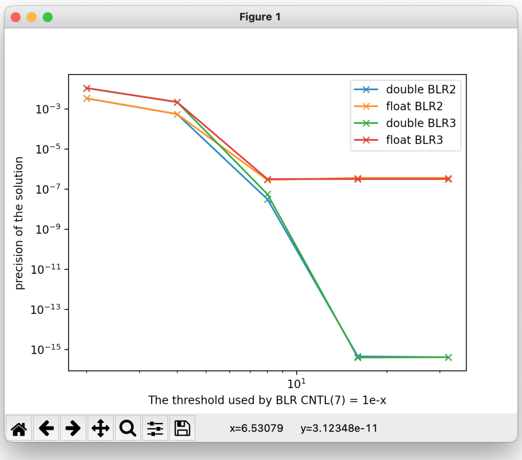


Figure 6: The number of operation, the memory space then the scaled residual depending on ϵ for the 2D Poisson problem

We can see on the plots that BLR2 and BLR3 has pratically no differences except for the memory. So, unless the user wishes to keep the data after the factorization phase, they should use BLR2.

Also, we can see that, for a small enough ϵ (depending on the epsilon-machine of the variable used), BLR needs more operations and memory than without using BLR.

But, if we do not need too much precision, BLR leads to pretty good results in our case. Especially for the number of operations.

And we can supose that we didn't gain a lot of memory space because the Poisson problem is too sparse, and the number of non-zero elements is too close to the rank of the matrix.

4.3.2 Poisson3D equation

After that, we tried to solve the 3D Poisson problem with different ϵ, but the execution failed several time, probably because it needed too much memory space. We only had at the end, these results (which were exactly the same for dmumps and smumps).

ϵ	1e-2	1e-4	1e-8
percentage of total over estimated FLOP	3.3	3.8	7.2
percentage of total over estimation entries	17.9	23.3	40.1
scaled residual	5.37e-3	5.53e-4	1.5e-7

Table 1: The percentage of FLOP and entries and the scaled residual for 3D Poisson equation

As we can see, the BLR feature allows the users to solves bigger linear systems in a shorter amount of time. Also, we can remark that the results above are almost identical either we use double or single precision, so it is important in this case to use single precision (because it will use less memory space and time, since an entry and a FLOP will be shorter or faster than for double precision).

4.4 Iterative Refinement ICNTL(10) and CNTL(2)

This parameter is used during the solve and check phase. It improves the accuracy of the solution after the factorization phase. It uses for each step a large amount of time, but even with only two or three steps, it can improve a lot the precision.

ICNTL(10) controls the number of steps of the iterative refinement. It can be either negative or positive :
- If negative, it computes exactly the number of step, without even checking if the residual is decreasing or not.
- If positive, it computes at most the number of step, and it also stops if the decreasing factor of the residual is too small or if the wanted accuracy is reached (this can be control by CNTL(2)).

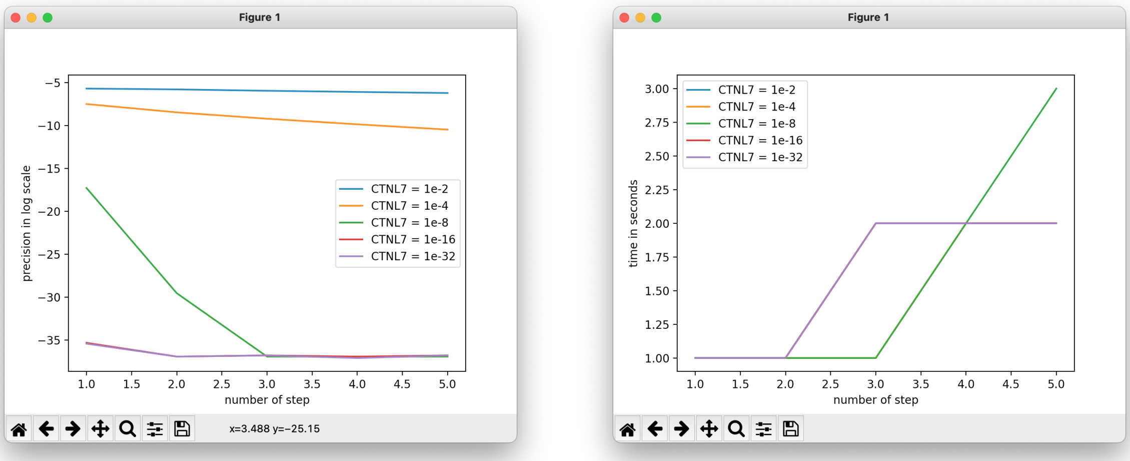


Figure 7: For 2D Poisson equation, the scaled residual then the time of the solve phase depending on the number of step

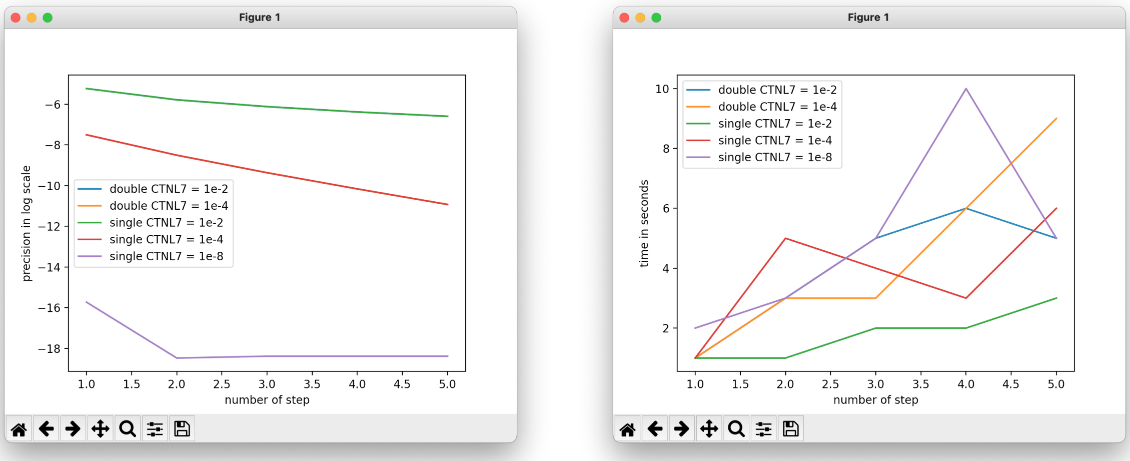


Figure 8: For 3D Poisson equation, the scaled residual then the time of the solve phase depending on the number of step

As we can see, if the ϵ of the BLR feature is too large, then, even with a lot of iterations, the scaled residual doesn't converge or very slowly (in those case, if we would have put a positive number, the iterations would have been stopped because the decreasing factor of the residual is too small).

On the other hand, when the refinement iteration converges, a very few number of iteration is enough to almost reached the epsilon-machine, also we can see that even with $\epsilon = 0$, we will still get a better result by doing a single step of the iterative refinement.

5 How we managed to benchmark

To benchmark, we had to have the access to a good machine, but as a regular student we did not have access to the LIP6 machine. The solution that we found to deal with that was to write a script and send it to our supervisor. Meaning that it was to get used to MUMPS we had to install it on our machine and this step of the project was tedious. We finally managed this installation thanks to PETSc (the Portable, Extensible Toolkit for Scientific Computation)

6 Conclusion

To conclude, we can find, for a precision ε (scaled residual) and a fixed matrix what are the best parameters (among those we studied) to have an efficient computation (by efficient, we are trying to minimise the memory space and the computation time).

For the 2D Poisson equation, we can see that it is better to use AMD instead of the default value. But for the 3D Poisson equation, Metis leads to the best results and since it is the default value, we don't need to change anything.

Also, according to our results, if we do not need a precision smaller than 1e-8, it is better to use smumps (since it will use less memory space for the same number of entries, and even if the number of operations is the same, they will be a bit faster).

Finally, we saw that the refine iteration steps did not converge when it was used on a BLR with ε too big.

Let ε be the precision the user wishes to reached, we are going to note μ the value for ε for which the user is ready to use the memory space (used by BLR), and the time taken by the post-processing refinement is not too much. (In our case, depending on the user wishes, it is between 1e-4 to 1e-8). The best parameters we found for our problems is :

For 2D : AMD (ICNTL(7) = 0) instead of default, and:
- $\varepsilon > \mu$: smumps, BLR (set to ε)
- $\varepsilon \leq \mu$: dmumps, BLR (set to μ) + iteration step (set to up to 10 steps, and ε)

for 3D : default value for ICNTL(7) and:
- $\varepsilon > \mu$: smumps, BLR (set to ε)
- $\varepsilon \leq \mu$: dmumps, BLR (set to μ) + iteration step (set to up to 10 steps, and ε)

Going further in this project is possible in a lot of aspects. We could try what we did with other matrices to see what are the behaviour or increasing n or nnz in the Poisson equation. We could also test with the parallel feature of MUMPS. An other interesting possibility that we could explore is to test what we did with other solver like SuperLU or pastix.

7 Bibliography

Website that we used for the repport:
MUMPS website
Block Low-Rank multifrontal solvers: complexity, performance, and scalability