

NUMERICAL ACCELERATIONS FOR POWER SYSTEMS TRANSIENT STABILITY SIMULATIONS

Florent Pruvost
Ecole Centrale Paris
Châtenay-Malabry, France

Thomas Cadeau
Ecole Centrale Paris
Châtenay-Malabry, France

Pascal Laurent-Gengoux
Ecole Centrale Paris
Châtenay-Malabry, France

Frédéric Magoulès
Ecole Centrale Paris
Châtenay-Malabry, France
frederic.magoules@hotmail.com

François-Xavier Bouchez
Tractebel Engineering SA
Brussels, Belgium

Bertrand Haut
Tractebel Engineering SA
Brussels, Belgium

Abstract - This paper analyzes the efficiency of some sequential and parallel methods for very large scale power systems simulations. The authors aim to assess flexible numerical strategies and solvers taking into account some power systems specificities but without considering too much intrusive methods. This study is illustrated using large realistic testcases. In addition, the simulations are performed on up to date computational softwares modified for this analysis. More precisely, the authors assess the efficiency of nowadays sequential and parallel sparse direct solvers (on multicores and multiprocessors computers), testing them and giving numerical results when applied to large power systems. The authors also investigate the possibility to replace the LU solver by a robust preconditioned iterative solver. Finally, the parallel in space approach is studied: the relaxation methods which enable to solve sub-systems on several cores or processors simultaneously. The authors aim at showing that these methods equipped with an adequate preconditioning technique can bring high parallel performances on a shared memory parallel architecture.

Keywords - Power system transient stability analysis, parallel computing, sparse direct solver, iterative algorithm, relaxation method, preconditioning

1 INTRODUCTION

THE power system stability study computes the system evolution after some events. If the system is not well designed, some events or sequences of outages can propagate and lead to significant damages and/or load shedding.

Nowadays, more and more very large power system simulations (such as the USA, China or the European interconnected transmission network) are carried out. In this case, the number of equations to solve can reach hundreds of thousands variables, making the simulation too much time consuming with actual algorithms.

1.1 Power systems differential algebraic equations

The power system stability study leads to the solution of a differential-algebraic initial value problem of the following form:

$$\begin{cases} y' &= f(y, z, t), t \in [0, T] \\ 0 &= g(y, z, t) \end{cases} \quad (1)$$

where y is the set of state variables defined by differential equations, z is the set of state variables defined by algebraic equations, and t the time variable. The system $y' = f(y, z, t)$ is a set of first-order differential equations system corresponding mainly to alternator equations and the corresponding controllers (excitation field and mechanical torque). It is assumed that each machine is coupled to the other machines through the network only. In presence of coupled machines (such as a TGV combined cycle) these machines are simply aggregated and seen as a larger machine. The corresponding differential system $y' = f(y, z, t)$ forms thus a set of completely independent subsystems. The algebraic set $0 = g(y, z, t)$ corresponds to the stator equations of each machine and to the equations of the network and loads. The vector y represents the differential variables arising in the controllers and the fluxes in the alternators. The vector z represents the algebraic variables arising in the controllers and the voltages of each bus. The functions f and g are strongly nonlinear.

Due to some events such as breaker openings/closings, the mathematical system copes with large discontinuities impacting the algebraic transmission network equations. Other smaller discontinuities frequently happen. Due to automatic control operations some variables can be bounded and the corresponding functions are not differentiable or even non-continuous. The system is considered as a “rough” problem not only due to the discontinuities but also because of its stiffness. Indeed, this stiffness of the differential system is related with the range of time constants which are also the rates of response of the differential variables. In power systems, the problem is stiff because the ratio between the largest and smallest time constants is large. Note that it corresponds also to the ratio between the largest and smallest eigenvalues of the linearized system, i.e., the system is ill-conditioned.

1.2 Integration methods

In this paper, neither the Differential Algebraic Equations (DAE) integration schemes choice [2], nor the timestepping choice [14] are of any consideration. It is only assumed that the integration methods use implicit schemes and variable stepsize. Usually, these methods involve less stability problems and thus allow to use larger integration steps that we denote *timesteps*. Indeed, for short-term phenomena, the stiffness of the problem requires the use of

small timesteps to obtain the solution, say 10^{-4} seconds or even smaller to ensure reasonable accuracy. On the opposite, for long-term phenomena, timestep sizes can be much larger, up to several seconds and, the whole time interval to solve may be long, up to 20 minutes and more.

Implicit methods usually use an explicit predictor and an implicit corrector. This implicit corrector involves the solution of a nonlinear system. Here, we denote by $t_n \in [0, T]$ the integration times, $h_n = t_n - t_{n-1}$ the timesteps and $x_n = (y_n, z_n)$ the state vector at time t_n . At each time the new state x_n is computed by solving the nonlinear system formulated as follows:

$$F(x) = C(x_{n-1}) \quad (2)$$

where $F(x)$ and $C(x)$ are functions depending on state variables. Since the function F is nonlinear, the Newton's method is usually used to solve (2):

$$[Jac(F(x_n^k))]^{-1}(x_n^{k+1} - x_n^k) = -F(x_n^k) + C(x_{n-1}) \quad (3)$$

Newton's iterative method requires the evaluation of the sparse Jacobian matrix, $Jac(F)$, and the solution of a linear system at each iteration k . The initial state is computed with an explicit predictor which should be very close to the solution at t_n . This algorithm is known to have a quadratic convergence close to the solution. In practice, Newton's algorithm can be applied with an approximation of the Jacobian matrix (most of the time a former Jacobian). The method is then called *Very DisHonest Newton* method (VDHN) or *Approximated Newton* and it is widely used in industrial programs.

In literature, to solve the nonlinear system (2), some methods use Newton's method on the global system and others apply relaxation techniques on subparts of the system. Direct methods, based on Lower Upper (LU) factorization, are broadly applied to solve the linear system (3). For power system applications, specific works have been done to reduce [4] or to parallelize the LU factorization. Reference [11] proposes to re-order the Jacobian matrix into Bordered Block Diagonal form and then to process the factorization in parallel. However, the limited size of the matrices and load imbalance due to the matrix structure significantly limit scalability. Iterative methods, like BiConjugate Gradient (BiCG) [7] and Generalized Minimal RESidual (GMRES) [3], could substitute the LU factorization methods if an adapted and cheap preconditioner was found [10]. On another side, relaxation methods have been often studied to solve the nonlinear system. First, the alternating implicit approach [16] makes use of the natural parallelism between the machines with the help of a relaxation between the machine blocks and the network. Reference [18] shows that electrotechnical characteristics can be used to efficiently apply the method on distributed machines. Subdomains relaxation methods allow to increase the parallelization granularity [19], [9], by partitioning the global system into independent weakly coupled subsystems.

1.3 Context and testcases presentation

In this paper, two testcases inspired by real power systems are considered. Synchronous machines are modeled using relatively complex generic models of Automatic Voltage Regulator (AVR) and speed governors. These controllers introduce between 20 and 25 variables each. Table 1 presents the dimension of the two testcases used in this paper. This table details the system characteristics, the equations properties and the Jacobian $Jac(F)$.

Testcases	Medium	Large
Power Systems features		
# buses	682	15350
# branches	1144	21990
# loads	301	4853
# machines	250	3824
DAE Systems features		
# Differential variables	7374	73228
# Algebraic variables	3327	50235
# Total N	10701	123463
Jacobian matrices features		
# Non-zeros entries n_{nz}	54447	673494
Fill-in ratio $\frac{n_{nz}}{N}$	5.08	5.45

Table 1: Testcases characteristics.

The main focus of this paper is to analyze the efficiency of some methods based on experiments with these two testcases.

In our context, the timestep is fixed by the required precision, see [17], [2]. To ensure Newton's algorithm convergence and, as an additional constraint, in a few number of iterations, experiments show that the approximated Jacobian should be very close to the exact one. Convergence problem appears quickly if the Jacobian is not updated after a significant timestep change or if a function has undergone a discontinuity.

In this paper, we analyze different methods for power system simulation. We focus on methods sufficiently generic so that they will be maintained and supported on future hardware. Costly tasks identified during a power system integration are mainly: linear system resolution, Jacobian evaluation/factorization and nonlinear functions evaluation. Evaluations are trivially parallelizable but it is not so easy for the LU factorizations and the forward and back substitution so that the linear systems resolutions represent the main bottleneck. In this study dedicated to large power systems, the compared costs of the main operations are different from analysis in other domains. For example, LU factorization cost is not so prohibitive compared to other elementary tasks, see Table 2. Using these cost evaluations, we propose a new analysis of different classical resolution methods.

The discussion is organized as follows. First, we present the large linear systems arising in power systems and their resolution by classical algorithms such as LU factorization and iterative methods. Then, we investigate the relaxation methods such as the Jacobi block relaxation.

2 DIRECT-BASED METHODS

FIRST, a brief analysis of the main operation costs in our context is provided. Based on this analysis, some hypothesis are taken which justify the choices made in the following sections. Three main steps can be considered:

1. The symbolic factorization analyzes the sparse structure of the matrix to compute a reordering which will reduce the fill-in. The structure of the matrices L and U are mainly fixed. This process is only done when some topology changes occur (a branch opening, a machine shut-down...).
2. The numeric factorization computes the coefficients of the matrices L and U. This process is necessary each time the Jacobian changes.
3. The forward and back substitution by using the L and U factors computes the solution.

The most expensive computational process is the matrix factorization, it is thus very efficient to solve successive linear systems by using the same factorization several times with different right hand sides.

2.1 Context and hypothesis

To present our context, we summarize in Table 2 some costs of the elementary operations measured for the large testcase; the architecture is a 2 x Intel Xeon E5410 (4 cores - 2.3GHz), 4 Gb RAM 1066 MHz, Windows Vista. The sparse linear solver used to handle the LU factorization here is KLU 1.0 [6].

We denote by C_J the Jacobian evaluation, C_{LU} the Jacobian numeric factorization, C_F the functions evaluation and $C_{f/b}$ the forward and back substitution.

Operation	Time (ms)
C_J	87
C_{LU}	117
C_F	47
$C_{f/b}$	12

Table 2: CPU time for the elementary operations

Table 2 allows to analyze the ratio cost between the tasks considered to be very costly and the other operations. We make the following assumptions:

- the LU factorization cost τ is close to the Jacobian matrix evaluation and represents only twice the functions evaluation cost. Moreover, the forward and back substitution is about $\frac{\tau}{10}$.
- C_J and C_{LU} are usually considered to be some very costly tasks which computation should be saved. We denote *direct*, the method that used a VDHN with LU factorization.
- The integration is performed between two events changing the system topology, the symbolic factorization is computed only once.
- The number of Jacobian evaluations and factorizations is N_J .

- There is one function evaluation and one forward/back substitution per Newton's iteration. The algorithm needs 2 Newton's iterations per timestep on average.

We can ask ourselves the following question: how many timesteps N_T resolution without Jacobian evaluation/factorization are equivalent, in term of cost, to an evaluation+factorization of the Jacobian? Regarding the main tasks, the cost is:

$$C_{direct} = N_J(C_J + C_{LU}) + N_T(2(C_{f/b} + C_F)).$$

Thus, on a time interval where $N_J = 1$, the number of time steps can be deduced from:

$$N_T = \frac{C_J + C_{LU}}{2(C_{f/b} + C_F)}.$$

With the CPU times given above in Table 2 $N_T \approx 1.7$. These values are surprisingly small. For example, for 10 timesteps integration, the updating cost of Jacobian factorization is less than $\frac{1}{5}$ of other costs. Obviously, C_J and C_F could be smaller using other power system softwares but C_{LU} and $C_{f/b}$ depends only on the linear solver. For the matrix sizes considered in this application, we tested sequential sparse unsymmetric linear solvers like UMF-PACK [5], SuperLU [8] and KLU [6]. Furthermore, parallel solvers such as MUMPS [1] can be used to perform the task in parallel on a multiprocessors machine or PAR-DISO [15] for a multicores architecture.

In [20], the LU factorization cost is assumed to evolve superlinearly with the system size. Indeed, experimental results using the above solvers assess that $C_{LU} = O(N^\alpha)$ with $\alpha \approx 1.2$ (see Figure 1). Costs evolution can therefore be considered as quasi linear.

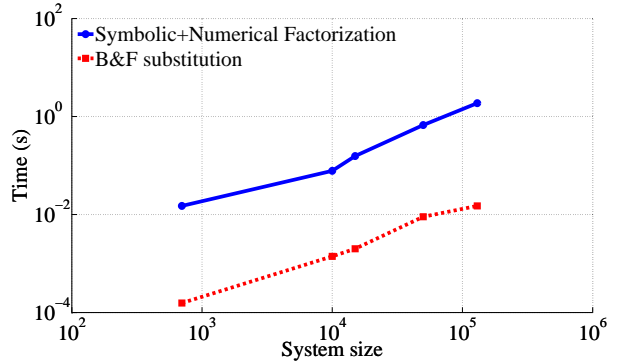


Figure 1: LU factorization cost evolution, log/log coordinates

Since the cost of Jacobian and function evaluations are almost linear with respect to the system size, it indicates that the above analysis concerning the factorization update is valid for system of other sizes.

2.2 Results analysis

The matrices issued from the medium and large testcases are very sparse, with a fill-in ratio close to 5. Moreover there exists efficient reordering techniques, that is why the factors L and U remain almost as sparse as the original matrix (see Table 3 and Figure 2) contrary to the matrices arising from finite-element discretizations for which most sparse direct solvers have been designed.

Ordering methods	AMD	COLAMD	MMD	METIS	PORD
Medium testcase	(KLU)	(KLU)	(SUPERLU)	(PARDISO)	(MUMPS)
# Non-zeros entries in L+U	67515	118427	150453	87729	89869
Fill-in ratio	1.24	2.17	2.76	1.61	1.65
CPU time cost (ms)	5.61	8.27	30.26	61.45	484.00
Large testcase	(KLU)	(UMFPACK)	(PARDISO)	(PARDISO)	(MUMPS)
# Non-zeros entries in L+U	903865	1072416	1087235	1267609	1148051
Fill-in ratio	1.34	1.59	1.61	1.88	1.70
CPU time cost (ms)	72.39	370.97	718.00	885.76	7238.00

Table 3: Cost of ordering methods

Sparse direct solvers	UMFPACK 5.2	KLU 1.0	MUMPS 4.8	SuperLU 3.1	PARDISO 3.2
Medium testcase					
Num. Fact. Time (ms)	34.3	6.8	98.3	10.9	93.0
Resolution Time (ms)	4.5	0.4	14.9	0.9	16.0
Large testcase					
Num. Fact. Time (ms)	578.8	117.0	1201.0	1263.0	328.0
Resolution Time (ms)	81.7	12.3	156.0	31.0	62.0

Table 4: Sequential costs of LU factorization methods

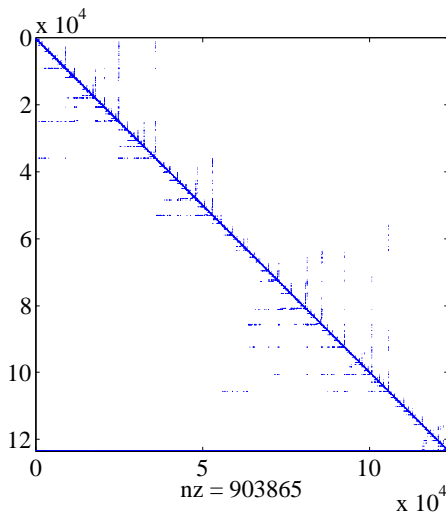


Figure 2: Large testcase matrix pattern after AMD reordering

	MUMPS 4.8		PARDISO 3.2	
	Fact. (ms)	Solve (ms)	Fact. (ms)	Solve (ms)
1p.	1201	156	328	62
2p.	1545	328	206	64
3p.	1638	327	148	65
4p.	2075	343	156	62
5p.	2200	327	114	69
6p.	2652	344	118	58
7p.	2636	219	84	56
8p.	3098	187	131	75

Table 5: Parallel costs of LU factorization methods on the large testcase

The following times have been obtained using the machine described in Section 2.1. Note that UMFPACK and KLU are sequential solvers when the three others can perform both sequential and parallel computations. For many of them, a large set of options are available, we tune them to get the best CPU time performances. As mentioned before, the factors of the decomposition remain very sparse and the factorization costs computed in sequential are al-

ready small. Because they are extremely sparse, dense matrix kernels like Basic Linear Algebra Subprograms (BLAS) are not really efficient. Dense matrix kernels are useful for the supernodal (SuperLU, PARDISO) and multifrontal (UMFPACK, MUMPS) methods. Power systems matrices typically do not have large supernodes because the nodes interconnections are not similar to all the nodes of the network. KLU solver gives very good results since it does not use BLAS optimization techniques. In Table 5, we present some results of LU factorization using parallel solvers. As it can be seen, the results are not promising for a nowadays large power system matrix although PARDISO solver improves slightly the efficiency. The limited size of matrices and the inefficiency of BLAS parallelization explain why KLU performances are comparable to parallel ones.

2.3 Direct methods conclusions

The fill-in of the LU factors is so weak (almost 1) that the number of non-zero elements in L+U is almost proportional to the number of non-zero elements in the original Jacobian matrix. It implies that the LU factorization cost is almost linear with respect to the matrix size. Thus, using a well suited recent sparse direct solver, the factorization computation cost is of the same order of magnitude than the Jacobian evaluation and only a factor 2 compared to the functions evaluation cost. As a consequence, the Jacobian update can be performed quite often and in our context every 5 iterations. The LU factorization method is a very competitive method in this context and it remains a reference to efficiently solve the linear system.

3 ITERATIVE-BASED METHODS

THE main operations in a preconditioned iterative method are, for one iteration, one matrix-vector

multiplication and one system resolution (in fact a forward/back substitution). The advantage of using an iterative method in an implicit integration relies on the possibility to update the Jacobian without factorizing it and/or parallelizing the matrix vector multiplication. However, the Jacobian matrices arising in power systems are far from having favorable properties for iterative methods. First experiments show that a strong preconditioning is required to get a fast convergence. Usual preconditioners like Incomplete LU are inefficient. Indeed, using BiCG or GMRES algorithm, the number of iterations to converge reaches easily 100 on the large testcase. The Jacobian $Jac(x(t))$ computed at one time t is a cheap preconditioner for solving the linear systems for the next timesteps in a time interval $[t, t + \Delta t]$. Of course, $Jac(x(t))$ has to be factorized into the LU form.

3.1 Computational costs

We compare the ‘‘Newton/LU factorization’’ method, denoted *direct* presented in section 2, with the ‘‘Newton/iterative’’ method, denoted *iter*. We consider that Jacobian evaluations are done under the same conditions than for *direct* and thus that we get the same number of Jacobian evaluations N_J . Actually, a first study was done evaluating the Jacobian at each time step, then evaluating the Jacobian only under some conditions. During the linear system resolution, the costly tasks of the iterative algorithm are: a product matrix·vector with the Jacobian (denoted $J.x$) and forward/back substitutions denoted f/b to apply the preconditioner denoted LU. Note that the cost of f/b is equivalent to the multiplication of the matrix (L+U) by a vector. As a consequence, because of the weak fill-in of the LU factorization, the cost of f/b is equivalent to the cost of $J.x$, and in the general case, taking into account the good parallelization of $J.x$, the relation $C_{f/b} \geq C_{J.x}$ holds.

There is one functions evaluation per Newton’s iteration as for *direct*. If during the experiment we get on $[t, t + \Delta t]$, N_{LU} preconditioner’s LU factorizations and N_{it} solver iterations (sum on Newton’s iteration and on the time), the cost of *iter* is:

$$C_{iter} = N_J C_J + N_{lu} C_{LU} + N_T (2C_F) + N_{it} (C_{J.x} + C_{f/b}).$$

Then *iter* is competitive in comparison with *direct* if $C_{iter} \leq C_{direct}$ which becomes

$$N_{lu} C_{LU} + N_{it} (2C_{J.x}) \leq N_J C_{LU} + N_T (2C_{J.x})$$

and finally

$$N_{it} \leq (N_J - N_{LU}) \frac{C_{LU}}{2C_{J.x}} + N_T. \quad (4)$$

The most favorable case for the iterative method is when the Jacobian matrix is evaluated at each timestep ($N_J = N_T$) and if the preconditioner is factorized only once ($N_{LU} = 1$). With such an assumption, to be effective,

the iterative solver should make less than $N_T \frac{C_{LU}}{2C_{J.x}}$ iterations and thus, less than $\frac{C_{LU}}{2C_{J.x}}$ iterations per integration step on average.

The ratio cost between methods $R = \frac{C_{iter}}{C_{direct}}$ is:

$$R = \frac{1 + \frac{(N_{it}(C_{J.x} + C_{f/b}) + C_{LU})}{N_T (C_J + 2C_F)}}{1 + \frac{(C_{LU} + C_{f/b})}{(C_J + 2C_F)}}.$$

3.2 Application to the large testcase

The iterative solver used for the experiments is a GMRES routine, FGMRES of the Intel Math Kernel Library (MKL) 10.1. On the large testcase, ratio $\frac{C_{LU}}{2C_{J.x}}$ is about 5 which is not very high. Consequently, here GMRES must not perform more than 5 iterations per timestep (according to table 2) to be equivalent to *direct* method.

A minorant of R is $1/(1 + \frac{C_{LU} + C_{f/b}}{C_J + 2C_F})$ and evaluated with costs presented in Table 2 is about 0.6. With $N_J = N_T = 5$, $N_{LU} = 1$ and $N_{it} = 2N_T$, $R = 0.8$. Consequently, iterative method cost represents at best 0.8 direct method cost which indicates that this alternative method is in the most favorable case quasi equivalent to the direct method considered in section 2. On parallel and distributed architectures, the speedup of the different tasks will not change, because of the linear complexity of the tasks, and the conclusion remains the same.

Table 6 presents a sequence of this simulation where the Jacobian is evaluated at all the timesteps and the preconditioner LU factorization is performed all 4 timesteps. As can be seen in table 6, the number of GMRES iterations increases fast when the preconditioner is not exact. In the first part of the table, the number of GMRES iterations is often smaller than 5, that is why the iterative method is competitive. Nevertheless, as the timesteps are increasing the method becomes less interesting since the number of iterations is larger than 5.

Time (s)	Timestep (s)	J	LU	N_{newton}	N_{gmres}
10.0050	0.0014	1	1	2	2
10.0064	0.0014	1	0	2	2
10.0078	0.0014	1	0	1	4
10.0092	0.0014	1	0	1	4
10.0150	0.0057	1	1	2	2
10.0208	0.0057	1	0	2	6
10.0266	0.0057	1	0	1	4
10.0324	0.0057	1	0	1	4
10.1446	0.0119	1	1	2	2
10.1536	0.0090	1	0	2	8
10.1627	0.0090	1	0	2	9
10.1717	0.0090	1	0	2	9
10.1866	0.0148	1	1	2	2
10.2014	0.0148	1	0	2	9
10.2162	0.0148	1	0	2	10
10.2310	0.0148	1	0	2	12

Table 6: Two simulation sequences after a fault at $t = 10s$ on the large testcase, solver tolerance= 1.10^{-6} . The preconditioner is factorized un-

der the following conditions: big timestep change, number of solver iterations > 12

3.3 Iterative methods conclusions

Iterative methods can hardly outperform the *direct* methods. Let us note that this conclusion, which can be different for other differential systems, relies on the fact that for power systems the Jacobian matrix is not only very sparse but also has a very small fill-in in its LU factorization (factor less than 1.5) which explains well why the ratio $\frac{C_{LU}}{C_{f/b}}$ is so weak. Thereby, iterative method seems to be too costly if used instead of the classic LU factorization method during the whole simulation. Nevertheless, it presents some interest for time intervals, where the timesteps are small and corresponding to a situation after an event.

4 RELAXATION METHODS

RELAXATION methods are known to be flexible and easy to implement. Their efficiency is however often limited due to a slow convergence for large and stiff DAEs. In this paper, preconditioning techniques are considered in order to improve their convergence and to make them competitive.

4.1 Relaxation methods principle

The aim is to solve the *global* nonlinear system (2) of size N which is formulated as $F_l(x) = C_l(x_{n-1})$, $l = 1, \dots, N$. Let $\Omega = \{1, \dots, N\}$, the set of indexes of the state variables x , be the *global domain*. We define a partition of the global domain Ω in p *subdomains*:

$$\begin{cases} \Omega_i \subset \Omega, \\ \cup_{i=1}^p \Omega_i = \Omega, \\ \Omega_i \cap \Omega_j = \emptyset \quad \forall i \neq j \in \{1, \dots, p\}. \end{cases} \quad (5)$$

Let $\bar{\Omega}_i \supset \Omega_i$ be a partition with overlap between subdomains so that some variable indexes can be found in several subsets Ω_i and Ω_j $i \neq j$. The algorithm convergence is improved by increasing the overlap rate but this leads to more calculations. The best overlap size is thus the result of a tradeoff.

We define a graph \mathcal{G} with vertex set Ω and with edge $(l, m) \in \mathcal{G}$ if x_m appears explicitly in equation $F_l(x) = C_l(x_{n-1})$. In fact, this graph is the sparse matrix graph of the Jacobian. We denote the *boundary* $\Omega_{ib} = \{m \in \Omega - \bar{\Omega}_i / \exists l \in \bar{\Omega}_i \text{ with } (l, m) \in \mathcal{G}\}$ i.e. the set of indexes out of $\bar{\Omega}_i$ connected to indexes in $\bar{\Omega}_i$. Let $x_i = \{x_l, l \in \bar{\Omega}_i\}$ be the *local* variables and $x_{ib} = \{x_l, l \in \Omega_{ib}\}$ the boundary variables. Functions are grouped following the partitioning: $F_i(x_i, x_{ib}) = \{F_l(x), l \in \bar{\Omega}_i\}$. The Jacobi relaxation method relies on the iterations

$$F_i(x_i^{k+\frac{1}{2}}, x_{ib}^k) = C_i(x_{n-1}) \quad i \in \{1, \dots, p\} \quad (6)$$

where $x_i^{k+\frac{1}{2}}$ is calculated with *direct* method. Then, x^{k+1} is built by merging the restrictions of $x_i^{k+\frac{1}{2}}$ to Ω_i . The resolution of subsystems (6) can be performed simultaneously in parallel and needs to communicate boundary

states x_{ib} . To obtain high speedup, the resolution cost ratio between the global and local systems should be high and the communication cost should be controlled. The objectives are then to build a large number of subdomains and to keep the total boundary size small. Besides, to be competitive with the previously described methods, the number of iterations must be very small.

The alternating implicit approach [16] uses a partition without overlap where Ω_1 is the set of network variable indexes and Ω_i , $i > 1$ are the sets of machine variable indexes. These methods exploit the natural parallelism between the machines and perform the network partitioning in different ways, based for example on geographical heuristics. A sequential relaxation is also possible between the machines and the network. The main drawbacks of such methods are the task granularity and the convergence speed.

4.2 General Domain Decomposition Methods

Generally speaking, partitioning [12] a problem and solving it through domain decomposition methods [13] is not a trivial process. To ensure good performances of the Jacobi relaxation algorithm, one main point is to find an appropriate partitioning. Indeed, a bad distribution of the variables can lead to singular matrices or affects the convergence. In addition, a strong unbalance between subdomains sizes causes strong unbalance in computation times between processors. Furthermore, partitioning algorithms aim at creating small interfaces between subdomains to minimize the data exchanged and thus the cost of communications. Libraries, such as SCOTCH, METIS, JOSTLE and CHACO, aim at building partitions based on several of these criteria. For power system application, the input is the network graph where the machine variables are aggregated. The overlapping subdomains are then built by successive additions of boundary nodes layers.

4.3 Partitioning results

Table 7 shows some results for an example of partitioning in 8 domains with METIS: the differences between partitions with and without overlap are the size of subdomains, the min and max number of indexes in subdomain boundaries $[nb_{min}, nb_{max}]$ and the total number of indexes N_b in these boundaries.

Testcase Overlap	Medium testcase		Large testcase	
	No overlap	1 layer	No overlap	1 layer
Min $\#\Omega_i$	1289	1304	14873	14965
Max $\#\Omega_i$	1376	2017	15888	16037
$[nb_{min}, nb_{max}]$	[12, 32]	[58, 148]	[60, 186]	[212, 624]
N_b	202	714	866	2990

Table 7: Partitioning results in 8 subdomains on medium and large test-case, with and without overlap. $\#\Omega_i$ denotes the number of indexes.

It can be noticed in table 7 that the boundary of each subdomain remains small and the load balancing is respected.

4.4 Preconditioning Methods

For the basic method described in 4.1, when the number of subdomains becomes large the number of iterations

required to converge increases. This is due to global phenomena which link variables far from each other in the partitioning. Moreover, the number of iterations needed to converge increases with the timestep size. A preconditioning should be a global (in the sense of the DAE system) and cheap operation which allows to compute the global phenomena approximately. After one iteration of system (6), the global state x^{k+1} satisfies the equations locally (in the interior of Ω_i), but not the boundary equations. Since the machines are only linked through the network, if the voltages values are globally exact then the machines variables will be exactly computed. To improve equation solving on the boundaries, a Newton's iteration on the global algebraic system can be performed. This procedure needs the algebraic part of the Jacobian matrix $Q = \frac{\partial g(y,z)}{\partial z}$ and an evaluation of the algebraic functions $g(y,z)$. The operation is then to compute an update of the algebraic state by globally solving

$$z^{k+1} = z^k - Q^{-1} * g(y^k, z^k)$$

where $x^k = (y^k, z^k)$ are the states computed after an iteration of (6). Practically, the Jacobian matrix Q does not change a lot during time integration. We can therefore use a fixed LU factorization of Q to compute the preconditioning steps. The function g needs to be evaluated just for the boundary states, this contributes to the performance of the method.

4.5 Numerical results

A solver has been implemented to test the relaxation algorithm described above. The convergence criterion is the same as for the global Newton algorithm. Figure 3 shows that the solution quality is the same between 1 and 32 domains.

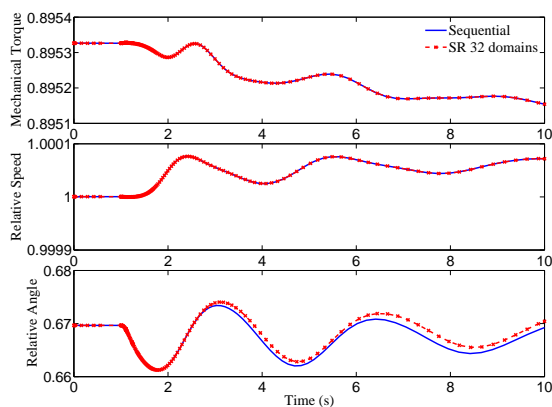


Figure 3: Large testcase solution after a short-circuit at $t = 1$ s

Figure 4 shows a comparison between the basic and preconditioned methods for the medium and large testcases. The timestep is variable and bounded to 0.02s.

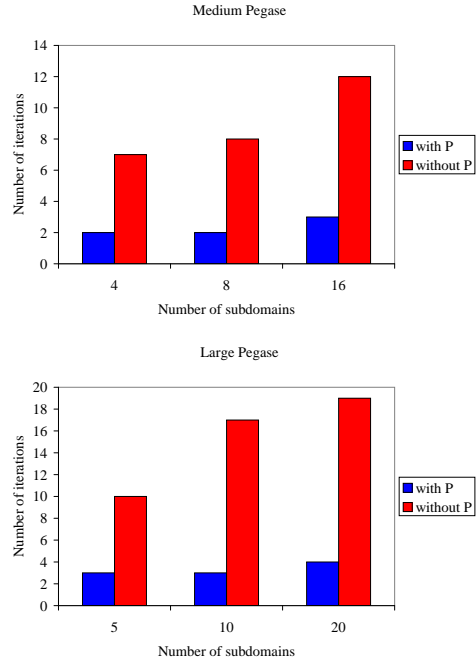


Figure 4: Simulation of the testcases. Fault occurs at $t = 10$ s. The overlap is one layer.

One can see that with preconditioning the number of iterations is small (2 or 3) and does not increase while the number of domains increases.

4.6 Relaxation methods conclusions

The local system are computed with *direct* method with a bounded number of 2 Newton's iterations. Knowing that the *direct* method cost evolves quasi linearly with N , the ratio cost between local and global resolutions is only $\frac{1}{N}$. Since increasing too much the number of subdomains increases the communication costs and because system sizes are limited, we cannot use a very large number of subsystems, say 100, 1000. As a consequence, to be competitive with *direct* methods, the number of iterations must be smaller than 3 or 4 iterations. Obviously, the basic method cannot compete with *direct* method. The use of the preconditioning technique presented in Section 4.4 gives better results for timesteps $h_n \leq 0.02$. However, even with this technique, when the timestep increases the number of iterations increases dramatically. New preconditioning methods aiming at obtaining fast convergence whatever the step size are currently under investigation.

5 CONCLUSIONS

IN this paper, authors assess some generic methods applied to power systems implicit integration. LU factorization, iterative linear solver and subdomain relaxation methods have been investigated. The best results are obtained for flexible tools using only sparse direct linear solvers. Subdomain relaxation methods present promising results using a preconditioning on the algebraic part of the system in a parallelized context. New preconditioning methods that take into account the dynamical behaviors have to be investigated in order to get fast convergence whatever the step size. With a good precondition-

ing, waveform relaxation methods will allow the granularity to increase by switching time integration and subdomains relaxation loops. It will be the aim of a further publication.

6 ACKNOWLEDGEMENTS

THE authors acknowledge partial financial support from the European Community's under the Seventh Framework Programme (FP7); grant agreement number 211407, PEGASE project <http://www.fp7-pegase.eu/>. The authors also acknowledge P. Panciatici and C. Merckx for helpful discussions and comments.

REFERENCES

- [1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multi-frontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.
- [2] J. Astic, A. Bihain, and M. Jerosolimski. The mixed Adams-BDF variable step size algorithm to simulate transient and long term phenomena in power systems. *IEEE Transactions on Power Systems*, 9(2):929 – 935, 1994.
- [3] Y. Chen, C. Shen, and J. Wang. Distributed transient stability simulation of power systems based on a jacobian-free newton-gmres method. *IEEE Transactions on Power Systems*, 24(1):146–156, 2009.
- [4] T. V. Cutsem and D. Fabozzi. Localization and latency concepts applied to time simulation of large power systems. In *Bulk Power System Dynamics and Control (iREP) - VIII (iREP), 2010 iREP Symposium*, 2010.
- [5] T. A. Davis. Algorithm 832: Umfpack v4.3— an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.
- [6] T. A. Davis and E. P. Natarajan. Algorithm 907: Klu, a direct sparse solver for circuit simulation problems. *ACM Transactions on Mathematical Software (TOMS)*, 37(3), 2010.
- [7] I. Decker, D. Falcao, and E. Kaszkurewicz. Conjugate gradient methods for power system dynamic simulation on parallel computers. *IEEE Transactions on Power Systems*, 11:1218 – 1227, 1996.
- [8] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.
- [9] V. Jalili-Marandi and V. Dinavahi. Instantaneous relaxation-based real-time transient stability simulation. *IEEE Transactions on Power Systems*, 24:1327 – 1336, 2009.
- [10] S. K. Khaitan and J. D. McCalley. A class of new preconditioners for linear solvers used in power system time-domain simulation. *IEEE Transactions on Power Systems*, 25:1835 – 1844, 2010.
- [11] D. P. Koester. *Parallel Block-Diagonal-Bordered Sparse Linear Solvers for Power Systems Applications*. PhD thesis, Syracuse University, 1995.
- [12] F. Magoulès, editor. *Mesh Partitioning Techniques and Domain Decomposition Methods*. Saxe-Coburg Publications, Stirlingshire, UK, 2007. Hardcover 368 pages.
- [13] F. Magoulès, editor. *Substructuring Techniques and Domain Decomposition Methods*. Saxe-Coburg Publications, Stirlingshire, UK, 2010. Hardcover 272 pages.
- [14] V. Savcenco, W. Hundsdorfer, and J. G. Verwer. A multirate time stepping strategy for stiff ordinary differential equations. *BIT*, 47:137–155, 2007.
- [15] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Computer Systems*, 20:475–487, 2004.
- [16] J. Shu, W. Xue, and W. Zheng. A parallel transient stability simulation for power systems. *IEEE Transactions on Power Systems*, 20:1709–1717, 2005.
- [17] M. Stubbe, A. Bihain, J. Deuse, and J. Baader. A new unified software program for the study of the dynamic behaviour of electrical power systems. *IEEE Transactions on Power Systems*, 4(1):129–138, 1989.
- [18] M. A. Tomin. *Parallel Computation of Large Power System Networks Using the Multi-Area Thévenin Equivalents*. PhD thesis, The University Of British Columbia, 2009.
- [19] J. White and A. Sangiovanni-Vincentelli. *Relaxation Techniques for the Simulation of VLSI Circuits*. Kluwer Academic Publishers, 1987.
- [20] W. Xue, J. Shu, and W. Zheng. Parallel transient stability simulation for national power grid of china. *Lecture Notes in Computer Science*, 3358:765–776, 2004.