

Cots Sparse Matrix Utilization in Distribution Power Flow Applications

Dino Ablakovic, Izudin Dzafic and Hans-Theo Neisius

Abstract Sparse matrix is used in many modern industrial applications, to provide low cost and high performance solution. Unlike a number of previous works in the field of distribution networks, which analyzed the sparse matrix calculations as integral part of its calculation engine, this paper presents the considerations and analysis for the utilization of commercial of-the-shelf sparse matrix solvers (COTS SMS). Analysis is given for the parallel direct sparse solver utilized in unbalanced and unsymmetrical Distribution System Power Flow solution, which maximizes parallelization on multi-core processors. Optimal matrix population algorithm and container class from the power network object model is proposed for the fast population, search and access to the matrix elements. Optimal use cases of symmetrical and unsymmetrical matrices for the given method are analyzed. Test results for the Current Injection—Power Flow algorithm on a Three-Phase large scale system are presented which show the performance scales for symmetric and unsymmetrical matrices over a scope of system sizes and for different network topologies.

Keywords Power flow • Sparse matrix • Distribution network • Three-phase

D. Ablakovic (✉) · I. Dzafic · H.-T. Neisius
Siemens AG, Humboldtstrasse 59, 90459 Nuernberg, Germany
e-mail: dino.ablakovic@siemens.com

I. Dzafic
e-mail: izudin.dzafic@siemens.com

H.-T. Neisius
e-mail: hans-theo.neisius@siemens.com

1 Introduction

Distribution Power Networks have changed immensely in recent years, becoming larger and more complex every day. Distribution generation increased the problem complexity by far. Networks became much larger with the need for interconnection between different networks on distribution and medium voltage level. Three-Phase unbalanced and unsymmetrical networks, typical for the USA, require three-phase analysis. All of this requires better, and more performant DSPF application.

Advanced DSPF solutions can utilize sparse matrix as the system impedance/admittance matrix depending on the methods used. Introduced in 1960s to the power systems calculation [1], sparse matrix utilization evolved to present times as the crucial element of DSPF. Nodal matrix method, Newton–Raphson and Fast-Decoupled DSPF algorithms in all different variations make a great use of sparse matrix. However, almost all previous DSPF solutions utilize sparse matrix internally, and all operations such as factorization and eventually forward backward substitution are deeply integrated into the DSPF algorithm.

It has long been realized that DSPF parallelization with different technologies and hardware can provide significant performance improvements [2–4].

Different factorization algorithms can be differently parallelized. From the perspective of DSPF application, it has become very hard to choose the right algorithm, hardware and implementation technique. The problem for the industrial real-time DSPF application is exponentially more complex when design and development time, and costs are considered. The aim of this paper is to analyze the utilization of Commercial Of-the-shelf Sparse Matrix Solvers (COTS SMS) for the DMS and more specifically DSPF, which are now more than ever, to maximize the power of parallel processing through multi-core CPU processors or even GPUs [5].

2 Cots Sparse Solvers in DSPF

Unlike so many previous works in research and industry which implement their own direct sparse solvers for the DSPF, COTS solvers offer equally performant, but much faster to implement and more feasible solution. A SWOT analysis of using COTS SS, which are elaborated here, is given in Table 1.

Two types of sparse solvers in general use in industry are: Direct and Iterative solvers. Both types have advantages and disadvantages, depending on the application. For the Power Systems in general, research up-to-date shows that Iterative method is still not developed enough, having convergence problems with different pre-conditioners. Direct sparse solvers have more predictable convergence, but require large memory to hold the factorized matrix. They are however, more stable, robust and reliable, and for those reasons found greater acceptance in DSPF applications and power systems in general.

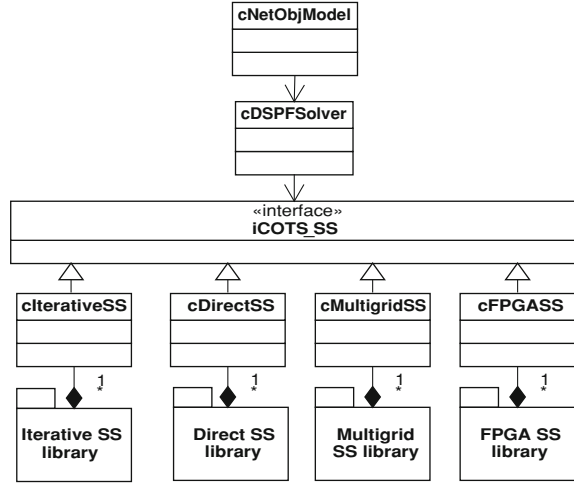
Table 1 SWOT analysis of COTS SMS

Strengths	Weaknesses
Developed by dedicated sparse solver entities	Special interface needed
Very feasible, no special development needed	Wrappers for all used COTS SS necessary
Clear interface to network applications	Special container class needed, additional memory used
Updateable without additional development	Dependency on 3rd party entity
<i>Opportunities</i>	<i>Threats</i>
Make best use of parallel processing, multi-core CPU or GPU, FPGA	Eventual bugs in 3rd party software
Ability to chose different SS for different hardware, operating systems and projects on demand	Product update and support discontinuity
Updateable without additional development at small cost	Compatibility issues
	Portability issues

Numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations [5], analyses 10 different COTS solvers and provides the performance profile for different matrix types for each of them. Although the referenced benchmarking work is done only for symmetrical solvers with real numbers, many of the tested ones support also unsymmetrical matrices, and almost all support complex numbers. This is crucial consideration for choosing the right solver, depending on the DSPF implementation.

A few direct sparse solvers are developed directly by the processor vendors which can only emphasize the maximum output from their most deep knowledge of the hardware on which the solver runs. According to the Moore's law the processor performance is doubled every 18–24 months [6]. With currently available 8 and 12-core processors, parallel sparse solvers are inevitable in serious DSPF application. Other than processor vendors, there are serious research entities dedicated to the sparse solver development which are more concentrated on the algorithms for improving the solvers, by constantly finding better solutions for ordering, pivoting and factorization algorithms. This dedicated development of sparse solvers is very time and resource consuming and can be too big task for Power Systems development groups. Given that hardware development is so fast that it constantly requires revision of the sparse solver algorithms and re-implementation and the above stated arguments it can be concluded that COTS SMS can be feasible, performant, up-to-date solution for DSPF. From all of the above, requirements for the COTS SMS in DSPF are defined as follows:

1. Direct solver type
2. Support float and double complex numbers
3. Must support symmetrical and unsymmetrical matrices
4. Platform independent (Windows/Linux)
5. Acceptable licensing/pricing model
6. Acceptable upgrade/development roadmap

Fig. 1 COTS SMS interface

3 Interface and Wrappers of Cots Sparse Solver

The aim is to enable DSPF and DMS to integrate any COTS SMS, and to replace it at lowest possible cost when better one is available, or even to be able to use more than one solver depending on the available hardware platform, operating system. Solution must be so robust to integrate iterative, direct and even GPU and FPGA solvers.

Since there are a number of different solvers encapsulated in different libraries, which are created in different programming languages, the Interface must be defined with specific minimum functionalities, each solver must provide. Class wrappers must be used to encapsulate the solvers, which will then in Object Orientated Design OOD inherit from the defined interface. If C++ is the chosen language, as it is widely used in the DMS development, Fig. 1 presents the proposed interface.

Template capabilities of C++ are to be used for the input of data types of the solver. The first step is defining the iCOTS_SS interface and its functions, which means that all inheriting wrappers will have to provide those. Solvers main functions are obviously factorization and then solving the system of linear equations with factorized matrix.

4 Sparse Matrix Container

There are many methods for storing the matrix data, i.e.: compressed row, compressed column, block compressed row storage, diagonal storage, jagged diagonal storage etc. All of those sparse storage schemes use the continuous storage

memory allocation for the nonzero elements of the matrix, or eventually a limited number of zeros when indefinite matrices are used. Overview of the standard formats given in [8, 9] shows that the compressed row and column storage formats are the simplest because they don't consider the scarcity structure of the matrix, and they don't store any unnecessary elements. Because of it, they are on the other hand not very efficient, needing an indirect addressing step for every single scalar operation in a matrix–vector product or preconditioned solve. An example of compressed row format for the symmetric matrix with real numbers is given in (1) where only upper triangular elements are stored saving the space of both 0 and symmetric values.

$$[A] = \begin{bmatrix} 1 & 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 6 & 0 \\ 3 & 0 & 7 & 0 & 0 \\ 0 & 6 & 0 & 3 & 4 \\ 0 & 0 & 0 & 4 & 9 \end{bmatrix} \quad (1)$$

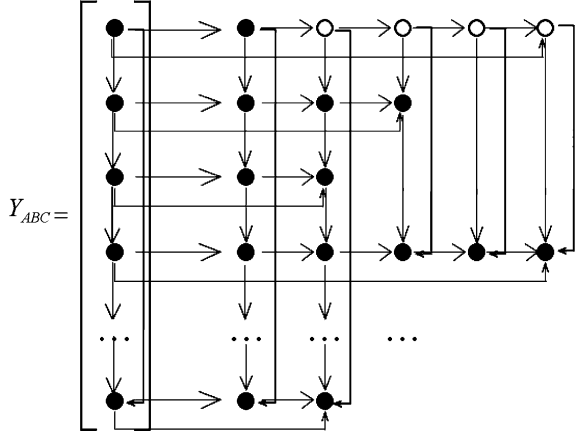
value	[V]	1	3	2	6	7	3	4	9
column index	[C]	1	3	2	4	3	4	5	5
row pointer	[R]	1	3	5	6	8			

The length of the value and column index arrays is actually the number of non-zero elements in the upper triangular matrix, and the row pointer array gives the location of the first non-zero element within a row.

Power networks topology is dynamic in its nature. Switching equipment changing its statuses define the network topology, combining and separating sub-networks. Having different sub-networks and topologies requires an implementation of tracing or graph theory coloring algorithm to collect all galvanically connected network elements, from which a sub-network is composed, at one point in time, by which the network object model is created. Bus bars are essentially graph vertices and branches, i.e. lines and transformers are graph edges. Switches are in general DSPF calculation ignored because their impedances are, when compared to branches, negligible.

When one sub-network object model is created, sparse admittance matrix vectors need to be populated. Because of its stacked sparse format, population process requires special sparse matrix container class which will provide fast population and access to matrix elements. Since the benchmark solver is using compressed row format shown in (1), it means for both algorithms that matrix must be populated by sweeping columns for each row, one by one. Because of virtually unknown number of non-zero elements the container must have dynamic size and grow as elements are inserted. This means no continuous memory allocation, but use of single linked lists. This container class is essentially made of one vector-array containing the first column of struct elements defined above. The array itself uses continual memory allocation and therefore the access to each row can be done in a single step. Later when row is found, iteration over the elements in the row by using a pointer to each next element and comparing the column value is needed. It is represented on Fig. 2.

Fig. 2 Advanced sparse matrix container structure



5 Symmetrical Matrix in DSPF

Unbalanced and unsymmetrical DSPF calculations in phase domain require admittance sparse matrix Y_{ABC} of size $3n$, where n is the number of electrical nodes and 3 is a phase multiplier. Unsymmetrical matrix factorization is in general slower than the symmetrical matrix factorization because of the more complex permutation calculation and larger fill-in calculation. Full unsymmetrical matrix size is n^2 and symmetrical $(n^2 + n)/2$. To achieve the best utilization of the matrix in DSPF, with the proposed modeling two main goals are:

1. Make only one Y admittance matrix factorization
2. Make the Y admittance matrix symmetrical

Nodal admittance matrix Y_{ABC} contains the branch admittances, i.e. lines and transformers. Loads are modeled in an injection vector, because they are dependent on the calculated voltage, which in the end makes this method iterative calculation process. When local control algorithm is implemented with DSPF, tap changer positions and capacitor banks are also considered in an injection vector.

To improve the method solvability, matrix numerical conditioning and decrease the number of iterations, advance modeling can be utilized in following:

3. Model constant impedance loads in the matrix
4. Model initial tap positions and capacitor banks in the matrix, and final tap positions simulate as current injections

Therefore general equation for the Current Injection method is presented in (3)

$$[Y_{ABC}] \times [V] = [I_L] + [I_D] - [I_S] = [I] - [I_S] \quad (2)$$

where:

- I_L —Load injection vector calculated from the voltage dependent loads.
- I_D —Injection vector calculated from tap positions of load tap changers and capacitor calculated in local control simulation.
- I_S —Injection source (Slack) compensation vector.

To avoid first calculation iteration, instead of filling the slack voltage values to (V) and doing one forward–backward calculation, I_L vector is initially set as given in (4). Because of mutual coupling between phases and the affect on the angle, initial Slack values at node k must be set with 120° phase shift.

$$\begin{aligned} I_{kA} &= 1\angle 0^\circ = 1 + j0p.u \\ I_{kB} &= 1\angle -120^\circ = -0.5 + j - 0.8660p.u \\ I_{kC} &= 1\angle 120^\circ = -0.5 + j0.8660p.u \end{aligned} \quad (3)$$

I_S is a compensation vector which ensures that Injection sources—Slack Buses are kept at the referent voltage during the DSPF iterations, keeping the matrix symmetrical at the same time.

$$I_{Sj} = Y_{ABC_{jk}} \times I_k^{n=0} \quad (4)$$

Where:

- n —iteration number = 0
- k —Index of the Slack Bus

$$\begin{bmatrix} 1 & 0 \\ 0 & [\tilde{Y}_{ABC}] \end{bmatrix} \times \begin{bmatrix} V_S \\ V_{1-n} \end{bmatrix} = \begin{bmatrix} 1 \\ I_{1-n} \end{bmatrix} - \begin{bmatrix} I_{S1} \\ I_{S1-n} \end{bmatrix} \quad (5)$$

When all Current injection vectors are added it still has to be ensured that values for Slack Buses are as given in (5), which means all branches and loads on the Slack Buses are ignored. Important to note is that I_S vector must be calculated only once and in the first iteration of the Power Flow when only Slack values are populated. The Load vector I_L is calculated in each Power Flow iteration and Delta I_D in each Local Control Iteration.

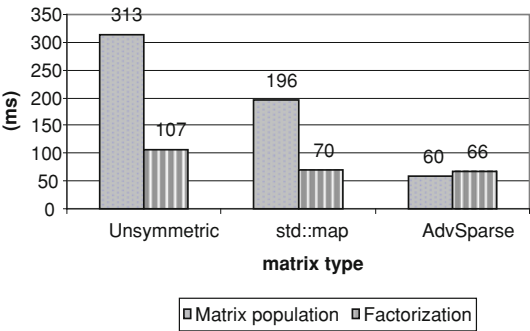
6 Result and Analysis

This paper presents the results of DSPF Admittance matrix population and factorization with implemented proposed sparse matrix container against the balanced tree container found in C++ Standard Template Library (STL). The test network is created from modified IEEE 34 test feeder [10] by multiplying the feeder 148

Table 2 Number of equipment units in the network

Type of equipment	Number of equipments units
Bus bars	5.071
Transformers	390
Lines	4.770
Loads	2.250
Capacitors	180
Switching devices	10.381

Fig. 3 Matrix population and factorization times for radial network



times and adding them in parallel to the same feeder head bus bar. Additionally connection line segments on Buses 864 and 840 are added in order to provide the meshed network topology. In Meshed configuration two loops are made between each two feeders on the given buses. The network size in total is given in Table 2.

Test Application: DSPF—Distribution System Power Flow
Benchmark Test Server: 2 × Intel Xenon(R) E5420 CPU 2.5 GHz, RAM 24 GB
Num of processor cores: 2 × 4 = 8 cores
Operating System: Windows Server 2003 SP2
Sparse solver: Intel MKL Pardiso [7]

Test Tool/Methods:

- Win32 API timing functions for profiling used directly in code utilizing number of processor ticks with precision to millisecond (ms).
- No code optimization and no SSE2 instructions in compiler optimization.
- Average calculation times on 10 runs are presented

Results presented show that both matrix population and factorization times can be significantly improved with usage of symmetrical matrix alone, even with usage of Standard Template library collections. Utilizing the Advanced Sparse Matrix container, population time is far better and shows the importance of giving so much attention to the entire process of sparse matrix utilization. Results are presented on Figs. 3 and 4, Table 3.

Fig. 4 Matrix population and factorization times for meshed network

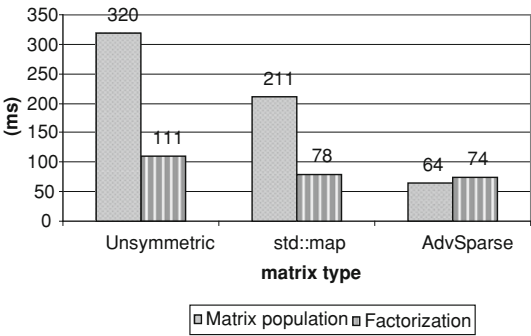


Table 3 Results of comparison tests for radial and meshed networks

		Unsymmetrical	Symmetric std::map	Symmetric AdvSparseMat
Radial networks	Population	313	196	60
	Factorization	107	70	66
Meshed networks	Population	320	211	64
	Factorization	111	78	74

Finally, from the presented results it can also be seen that results are worse for Meshed network types, which is result of more off-diagonal elements and spread matrix structure. A topic for further research is to determine exactly how the network topology affects the matrix population and factorization times.

References

1. Tinney WF, Walker JW (1967) Direct solutions of sparse network equations by optimally ordered triangular factorization. In: Proceedings of the IEEE, Nov 1967

2. Wu JQ, Bose A (1995) Parallel solution of large sparse matrix equations and parallel power flow. IEEE Trans Power Syst 10(3):1343 (August)

3. Feng Z, Zeng Z, Li P (2010) Parallel on-chip power distribution network analysis on multi-core-multi-GPU platforms. IEEE Trans Very Large Scale Integr VLSI Syst, vol PP, Issue: 99

4. Johnson J, Nagvajara P, Nwankpa C (2004) Sparse linear solver for power system analysis using FPGA, Drexel University, HPEC

5. Gould NIM, Hu Y, Scott JA (2005) A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations, Technical report, RAL-TR-2005-005, CCLRC Rutherford Appleton laboratory

6. Ramanathan RM Intel® Multi-core processors: making the move to quad-core and beyond, White Paper. Intel Corporation

7. Intel® Math Kernel library reference manual—March 2009 630813-031US

8. Templates for the solution of algebraic eigenvalue problems: a practical guide (2000) In: Bai Z, Demmel J, Dongarra J, Ruhe A, van der Vorst H (eds) SIAM, Philadelphia

9. Distributed sparse data structures for linear algebra operations (1992) Technical report CS 92-169, Computer science department, University of Tennessee, Knoxville, TN, LAPACK working note
10. IEEE 34 node test feeder, IEEE distribution system analysis subcommittee

Computer Science and Convergence

CSA 2011 & WCC 2011 Proceedings

Park, J.J.H.; Chao, H.-C.; Obaidat, M.S.; Kim, J. (Eds.)

2012, LV, 843 p. 329 illus., Hardcover

ISBN: 978-94-007-2791-5