Julio César MOREIRA MEIRA, Edelmiro MÍGUEZ GARCÍA, Carlos VILACHÁ PÉREZ, Antonio FERNÁNDEZ OTERO

University of Vigo, Departament of Electrical Engineering

Parallelization of an optimal power flow with a multicore symmetric shared memory computer

Abstract. *The boom of multicore microprocessors seen in recent years has made it so that any personal computer (PC) can be used as a small parallel computer. In this new environment a review of the algorithms and the tools used for the analysis of electrical systems is needed in order to take advantage of all the computing power available today on a PC. This paper presents the parallelization of an optimal power flow (OPF) which is solved with a predictor-corrector interior point method (PCIPM). OpenMP, a programming model for symmetric machines with shared memory, will be used for the parallelization.*

Streszczenie. *Rozkwit procesorów wielordzeniowych procesorów w ostatnich latach umożliwia zastosowanie przeciętnego komputera osobistego (PC) jako małego komputera równoległego. Przeglądu algorytmów oraz narzędzi używanych do analizy systemów elektrycznych w środowisku procesorów wielordzeniowych jest niezbędny w celu skorzystania z mocy obliczeniowej dostępnej obecnie na komputerze PC. W artykule przedstawiono paralelizację optymalnego przepływu energii (OFE), który został rozwiązany za pomocą algorytmu predyktor-korektor metody punktu wewnętrznego (PCIPM). Zastosowano do paralelizacja OpenMP, model programowania dla symetrycznych maszyn o pamięci współdzielonej. (**Paralelizacja optymalnego rozpływu mocy za pomocą wielordzeniowego symetrycznego komputera o współdzielonej pamięci)*.

Keywords: parallel computing, optimal power flow, interior-point method, predictor-corrector.
Słowa kluczowe: przetwarzanie równoległe, optymalny rozpływ mocy, metoda punktu wewnętrznego, predyktor-korektor.

Introduction

In recent years, PCs have undergone major advances in terms of parallel computing. Nowadays standard PCs come equipped with a quad-core microprocessor and almost all laptops are dual-core. In this new environment where parallelization is present in every PC, it is necessary to redefine the problems we can run locally without resorting to other more complex architectures such as grids or clusters.

With the increased computing power of PCs and the popularization of multicore microprocessors very soon a PC will be able to emulate a small cluster. It will therefore be necessary to rewrite the new limits regarding the size and complexity of the problems we can process without the need for an external computing source and the modification of existing algorithms to adapt to the new environment.

This paper examines the parallelization of one of the most important tools in the analysis of power grids, an OPF, and uses a predictor-corrector interior point method to solve it. This article tries to discover how to benefit from its execution in parallel and how it affects an increase in computing cores and also how this affects the computing time. One option for solving an OPF using a parallel computer is to divide the network into unconnected areas, solve them separately, and then, using some type a synchronizing mechanism, correct the deviations between variables shared by the different areas that must converge to the same final value.

In [1–3] they describe how to cut up the new network into subregions, using the "'Auxiliary Problem Principle"', that initially are joined via interconnecting lines. Once the network is broken down, each subnetwork begins to solve in parallel.

Another very common method used for breaking down into areas is the Lagrangian relaxation. The duality theorem is applied to the lines that form the border between the regions [4–9]. More recently, in [10] a breakdown of a network is shown by duplicating the nodes that form the border between the different regions.

One of the problems of breaking down the network into subregions is detecting and limiting the areas that comprise it. On occasions it is possible to have networks of a considerable size without them being divided into areas with clear borders. In this paper, solving the whole system and parallelizing the numerical resolution method has been chosen.

The three contributions presented by the article in the parallelization of an OPF that we believe that not have been published previously are:

- The first one is to present the execution times that could be obtained by solving an OPF on a shared memory machine, such as a personal computer, without using any technique to divide the network, simply paralleling interior point algorithm.
- Another contribution is to test the usefulness of OpenMP for the parallelization of an OPF. An important advantage of OpenMP is the ease with which it can be implemented in existing commercial codes.
- Finally, a third contribution is to provide execution times of a highly optimized OPF. In the literature there are many examples of networks solved with non-optimized codes (Matlab, etc.) with these codes is not possible to get execution times for comparison with other techniques in real situations.

Optimal Power Flow

Since the 60s the OPF has become an indispensable tool in the planning and operation of power systems, [11]. Generally an OPF can be represented as:

$$\begin{aligned} &\min f(x) \\ &g(x) = 0 \\ &\underline{h} \le h(x) \le \overline{h} \end{aligned} \tag{1}$$

- x is the variable vector; generally voltages and angles are the dependent variables, and usually the control variables are the generated power, transformers taps, etc.
- $g(x)$ are the equations of the nodal powers.
- $h(x)$ are functions that limit the variables.

The function to minimize $f(x)$ can be the active power losses, the generation costs, the reactive generation, etc. In this paper, the minimization of losses has been selected as the objective function for the realization of the numerical experiments. The results obtained can be extrapolated to any other problem defined by another cost function.

An OPF is a nonlinear optimization problem, and different techniques such as sequential quadratic

programming, successive linear programming, the increased Lagrangian method, interior point methods, etc. can be used to solve it. Due to its robustness and efficiency, the latter has been selected as a resolution method: a primal-dual method, a predictor-corrector (PCIPM) and multiple-centrality corrections (MCCIPM) interior point methods.

Primal-Dual Method

The development of the equations and the algorithm described below are based on [12] where a more detailed explanation can be found.

The PCIPM algorithm is an iterative process. In each step a system of equations given by the optimality conditions of Karush-Kuhn-Tucker (KKT) is solved. This can be seen in the equations;

$$(2)\quad \nabla_y L_\mu(\mathbf{y}) = \begin{pmatrix} \pi - \mu^k \mathbf{S}^{-1} e \\ \upsilon - \mu^k \mathbf{Z}^{-1} e \\ \mathbf{s} + \mathbf{z} - \overline{\mathbf{h}} + \underline{\mathbf{h}} \\ \mathbf{h}(x) + \mathbf{z} - \overline{\mathbf{h}} \\ \nabla_x f(x) - \mathbf{J}_g(x)^T \lambda + \mathbf{J}_h(x)^T \upsilon \\ -\mathbf{g}(x) \end{pmatrix} = 0$$

To solve the system of nonlinear equations (2) the Newton iterative method is used which can be written in this compact manner:

$$(3)\qquad A\Delta\overline{x} = \overline{r}_x$$

where

$$\Delta\overline{x} = \begin{pmatrix} \Delta s & \Delta z & \Delta\pi & \Delta\upsilon & \Delta x & \Delta\lambda \end{pmatrix}^t$$
$$\overline{r}_x = \begin{pmatrix} r_s & r_z & r_\pi & r_\upsilon & r_x & r_\lambda \end{pmatrix}^t$$

and the matrix of A coefficients is:

$$(4)\qquad A = \begin{pmatrix} \Pi & 0 & S & 0 & 0 & 0 \\ 0 & \hat{\Upsilon} & Z & Z & 0 & 0 \\ I & I & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & J_h & 0 \\ 0 & 0 & 0 & J_h^T & \nabla_x^2 L_\mu & -J_g^T \\ 0 & 0 & 0 & 0 & -J_g & 0 \end{pmatrix}$$

$\nabla_x^2 L_\mu$ is comprised of the Hessian sum of the equality, inequality and objective functions.

$$\nabla_x^2 L_\mu = \nabla_x^2 f(x) - \sum_{j=1}^{m} \lambda_j \nabla_x^2 g_j(x) + \sum_{j=1}^{p} \upsilon_j \nabla_x^2 h_j(x)$$

The variables s, z are slack variables, π, υ, λ are the Lagrange multipliers and the vector of independent terms is

$$(5)\qquad \begin{pmatrix} r_s \\ r_z \\ r_\pi \\ r_\upsilon \\ r_x \\ r_\lambda \end{pmatrix} = \begin{pmatrix} -\mathbf{S}\pi + \mu^k e \\ -\mathbf{Z}\hat{\upsilon} + \mu^k e \\ -\mathbf{s} - \mathbf{z} + \overline{\mathbf{h}} - \underline{\mathbf{h}} \\ -\mathbf{h}(x) - \mathbf{z} + \overline{\mathbf{h}} \\ -\nabla_x f(x) + \mathbf{J}_g(x)^T \lambda - \mathbf{J}_h(x)^T \upsilon \\ \mathbf{g}(x) \end{pmatrix}$$

Once the solution to the system is obtained (3), the variables are updated with the relations

$$x^{k+1} = x^k + \alpha_P^k \Delta x \quad \pi^{k+1} = \pi^k + \alpha_D^k \Delta\pi$$
$$\lambda^{k+1} = \lambda^k + \alpha_D^k \Delta\lambda \quad z^{k+1} = z^k + \alpha_P^k \Delta z$$
$$s^{k+1} = s^k + \alpha_P^k \Delta s \quad \upsilon^{k+1} = \upsilon^k + \alpha_D^k \Delta\upsilon$$

using a step $\alpha = (\alpha_P, \alpha_D)$

$$\alpha_P^k = \min\left\{1, \gamma \min_i \left\{ \frac{-s_i^k}{\Delta s_i} \mid \Delta s_i < 0, \frac{-z_i^k}{\Delta z_i} \mid \Delta z_i < 0 \right\}\right\},$$

$$\alpha_D^k = \min\left\{1, \gamma \min_i \left\{ \frac{-\pi_i^k}{\Delta \pi_i} \mid \Delta \pi_i < 0, \frac{-\hat{\upsilon}_i^k}{\Delta \hat{\upsilon}_i} \mid \Delta \hat{\upsilon}_i < 0 \right\}\right\}.$$

With the new variable values, the following termination conditions are checked;

$$(6)\qquad \begin{gathered} max\left\{h_l, h_h, \left\|g(x)\right\|_\infty\right\} \le \epsilon_1 \\ \frac{\left\|\nabla_x f(x) - J_g(x)^T \lambda + J_h(x)^T \upsilon\right\|_\infty}{1 + \left\|x\right\|_2 + \left\|\lambda\right\|_2 + \left\|\upsilon\right\|_2} \le \epsilon_1 \\ \frac{\rho}{1 + \left\|x\right\|_2} \le \epsilon_2 \\ \frac{|f(x^k) - f(x^{k-1})|}{1 + |f(x^k)|} \le \epsilon_2 \end{gathered}$$

where $h_l = max\left\{\underline{h} - h(x)\right\}$ and $h_h = max\left\{h(x) - \overline{h}\right\}$. If the conditions are met (6), the algorithm ends, otherwise The μ^k barrier parameter value is updated and the iteration is reinitiated.

The most time-consuming process in each iteration is the system solution (3). To reduce the number of iterations and improve the computing time, higher-order methods like the aforementioned Predictor-Corrector method [13] or the Multiple Centrality method [14] have been developed. The fundamental idea of these methods is to try to improve the search direction in each iteration in order to perform updates with bigger steps and reduce the number of iterations.

In all these methods, the factorization of the matrix (4) is reused in the process of calculating the new direction. The effort to improve the search direction at each iteration should be compensated by a reduction of the total iterations performed; otherwise the total computing time is not improved.

Predictor-Corrector Steps

To obtain a solution to (3), first we obtain the system solution:

$$(7)\qquad A\Delta\overline{x}^{af} = \overline{r}^{af}$$

where

$$\Delta\overline{x}^{af} = \begin{pmatrix} \Delta s^{af} & \Delta z^{af} & \Delta\pi^{af} & \Delta\upsilon^{af} & \Delta x^{af} & \Delta\lambda^{af} \end{pmatrix}^t$$
$$\overline{r}^{af} = \begin{pmatrix} -\Delta S\pi & -\Delta Z\hat{\upsilon} & r_\pi & r_\upsilon & r_x & r_\lambda \end{pmatrix}^t$$

The result is the affine direction or predictor step that, once the (α_P, α_D) steps are calculated, it is used to estimate the barrier parameter using the relationships:

$$(8)\qquad \begin{aligned} \rho_{af}^k = (s^k + \alpha_P^{af} \Delta s_{af})^T (\Pi^k + \alpha_D^{af} \Delta\Pi_{af}) + \\ + (z^k + \alpha_P^{af} \Delta z_{af})^T (\hat{\upsilon}^k + \alpha_D^{af} \Delta\hat{\upsilon}_{af}) \end{aligned}$$

and

$$\mu_{af}^{k} = min\left\{\left(\frac{\rho_{af}^{k}}{\rho^{k}}\right)^{2}, 0.2\right\}\frac{\rho_{af}^{k}}{2p} \quad (9)$$

Once the affine direction and the barrier parameter estimates and second order errors are obtained, the system below, where the predictor and corrector steps are integrated, is solved.

$$A\begin{pmatrix}\Delta s\\ \Delta z\\ \Delta\pi\\ \Delta\upsilon\\ \Delta x\\ \Delta\lambda\end{pmatrix} = \begin{pmatrix}-S\pi + \mu_{af}^{k}e - \Delta S_{af}\Delta\Pi_{af}\\ -Z\upsilon + \mu_{af}^{k}e - \Delta Z_{af}\Delta\upsilon_{af}\\ r_{\pi}\\ r_{\upsilon}\\ r_{x}\\ r_{\lambda}\end{pmatrix} \quad (10)$$

The systems discussed (7) and (10) can be solved directly or, taking advantage of the coefficient matrix structure (4), can propose the following reduced equivalent system;

$$\begin{pmatrix}J_d & -J_g^T\\ -J_g & 0\end{pmatrix}\begin{pmatrix}\Delta x\\ \Delta\lambda\end{pmatrix} = \begin{pmatrix}\overline{r}_x\\ r_{\lambda}\end{pmatrix} \quad (11)$$

where J_d and $\overline{r}_x$ are

$$J_d = \nabla_x^2 L_{\mu}(y) + J_h(x)^T(S^{-2} + Z^{-2})J_h(x)$$
$$\overline{r}_x = r_x - J_h^T(x)B$$

with

$$B = Z^{-1}r_z - S^{-1}\left(r_s - \Pi r_{\pi}\right) - \left(Z^{-1}\hat{\Upsilon} + S^{-1}\Pi\right)r_{\upsilon}$$

which is the option chosen for this paper.

From the solution obtained, the remaining unknowns are found using the following relationships:

$$\begin{aligned}\Delta z &= r_v - J_h(x)\Delta x\\ \Delta s &= r_{\pi} - \Delta z\\ \Delta\upsilon &= Z^{-1}(r_z - \hat{\Upsilon}\Delta z - Z\Delta\pi)\\ \Delta\pi &= S^{-1}(r_s - \Pi\Delta s)\\ &= S^{-1}(r_s - \Pi r_{\pi} + \Pi\Delta z)\end{aligned}$$

A summary of the steps comprising the algorithm can be seen below;

1. Initialization, error vectors, initial Jacobians and Hessians.
2. Calculating the reduced system matrix (11).
3. Numerical factorizations of the reduced system matrix.
4. Calculating the affine system solution (7).
5. Calculating the μ_{af} barrier parameter and 2nd orden errors: $\Delta S_{af}, \Delta\pi_{af}, \Delta Z_{af}, \Delta\hat{\upsilon}_{af}$.
6. Calculating predictor-corrector system solution (10).
7. Updating the $\nabla_x^2 L_{\mu}$, J_g and J_h Jacobian and Hessian matrices and calculating the error vectors.
8. Error Checking (6) and in the case of non-compliance with maximum allowable errors, return to step 2.

The system (11) can be solved by an iterative or direct method. When the system of equations is very large, iterative methods are usually preferred because they require less system memory. When the system of equations is small, direct methods are used. This paper will use a direct method to solve the systems.

Multiple-Centrality Corrections

In this method, the calculation of the affine direction $(\Delta\overline{x}^{af})$ and barrier parameter (μ_{af}^{k}) is performed in the same way as in the predictor-corrector method, and the correction vector (Δx_{co}) is calculated as follows.

First, the values of (α_P, α_D) are modified with the expression

$$\tilde{\alpha} = \min\left\{\alpha_{af} + \delta_{\alpha}, 1\right\}, \quad \delta_{\alpha} \in \{0, 0.15\}$$

and with these values the point $\tilde{x}$ is calculated

$$\tilde{x} = x_k + \hat{\alpha}\Delta x_{af}$$

It is possible that multiple components of this new point do not fulfill the condition $(s, z, \pi, \hat{\upsilon}) > 0$. In this case, the task of correcting address is to compensate the negative components and try to bring the values to the central path.

From $\tilde{x}$, the following vectors are obtained

$$\tilde{q} = \tilde{S}\tilde{\pi} \quad \tilde{r} = \tilde{Z}\tilde{\upsilon}$$

and projected into the hypercube

$$H = [\beta_{min}\mu_{af}, \beta_{max}\mu_{af}]^{2p}$$

(tipical values of β_{min} and β_{max} are 0.1 and 10) with the following expressions

$$q_i^t = \begin{cases}\beta_{min}\mu_{af} & if \quad \widetilde{q}_i < \beta_{min}\mu_{af}\\ \beta_{max}\mu_{af} & if \quad \widetilde{q}_i > \beta_{max}\mu_{af}\\ \widetilde{q}_i & \text{otherwise}\end{cases}$$

$$r_i^t = \begin{cases}\beta_{min}\mu_{af} & if \quad \widetilde{r}_i < \beta_{min}\mu_{af}\\ \beta_{max}\mu_{af} & if \quad \widetilde{r}_i > \beta_{max}\mu_{af}\\ \widetilde{r}_i & \text{otherwise}\end{cases}$$

Once the vectors (q_i^t, r_i^t) are obtained, the corrector direction (Δx_{co}) is calculated solving the system:

$$A\Delta x_{co} = \begin{pmatrix}q_t - \tilde{q}\\ r_t - \tilde{r}\\ 0\\ 0\\ 0\\ 0\end{pmatrix}$$

and is added to the affine direction

$$\Delta x = \Delta x_{af} + \Delta x_{co}$$

With this new direction, the error vectors, updating of matrices, etc, are calculated as in the predictor-corrector method.

OPF Parallelization

In this paper the programming model used for the parallelization is OpenMP [15], a tool for programming in a symmetric shared memory multiprocessor system such as a personal computer.

The main reason for using OpenMP is the few changes that must be made in the sequential code for its conversion to a parallel algorithm; this greatly reduces the cost of updating and subsequent maintenance. Most of the work is done by the compiler. The complexity associated with the assignment of tasks, and the creation/destruction of threads is hidden from the programmer.

OpenMP supports the so-called fork-join programming model [16]. The main thread when it reaches a region where there is code that can be executed in parallel, is divided into many threads as specified. These threads are executed in parallel and are destroyed when they have completed their assigned tasks, Fig. 1.

Parallelized code using this technique must have a minimum computational load to compensate the threads creation time and their destruction, when finish the parallelized block.

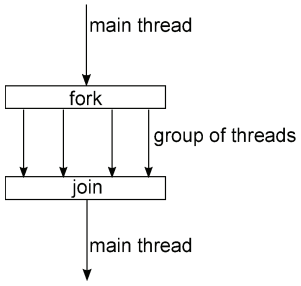

Fig. 1. OpenMP Model

Normally there will be many threads as cores have the PC, because if we start a lower number of threads, it will be idle cores. Nevertheless, starting more threads than cores will be costly context switches.

In both methods (PCIPM and MCCIPM), the part that consumes a greater amount of computing time in each iteration is the system matrix factorization (11) and the substitutions that must be made to obtain the affine direction (7) and the final direction with the step corrector (10). It is logical that the first objective to reduce the computing time using parallelization be these two operations. For the matrix factorization processes and substitutions, the well-known Pardiso libraries are used [17], which use OpenMP.

The third most time-consuming block of operations after the factorization and the substitutions is the updating of the Jacobian and Hessian matrices present in (11) and (12). But updating the elements of the matrices are independent processes that can also be done in parallel using OpenMP.

The pseudocode for calculating the elements of the Jacobians in parallel using OpenMP are:

```
#pragma omp parallel for
  for(int row=0;row<JacbRows;row++){
  CalcJacb(row);
}
```

and similarly for elements of the Hessian

```
#pragma omp parallel for
  for(int row=0;row<HessRows;row++){
  CalcHess(row);
}
```

The only modification done on the original code to add the OpenMP parallelization is the line `#pragma omp parallel for`. This simple modification allows the parallel calculation of the elements of the Jacobian and Hessian matrices.

Numerical Experiments

The code is written in C ++ and compiled with Intel software using the MKL libraries v.10.2.6. The hardware used for the tests is a machine with two Quad-Core Xeon 5540 microprocessors with 16GB of RAM operating on Windows Server 2008 SP2.

For this study the well-known network, IEEE-300 will be used in addition to various systems based on the network of IEEE-118 nodes (1185ZxN) built automatically to have high dimensional networks. These systems start with a network connecting five networks of 118 nodes (as shown in Fig. 2(a)) using the nodes indicated in the illustration. The parameters of the interconnecting lines are shown in table 1.

Table 1. Values of the Interconnection Lines in p.u.

Nodes	R	X	B
117Z0-117Z1	0.05	0.1	0.05
39Z0-39Z1	0.06	0.09	0.06
53Z0-53Z1	0.07	0.11	0.05
101Z0-101Z2	0.06	0.09	0.06
88Z0-88Z2	0.05	0.1	0.05
108Z0-108Z3	0.07	0.1	0.05
51Z0-51Z3	0.09	0.13	0.07
83Z0-83Z4	0.05	0.1	0.05
3Z0-3Z4	0.03	0.07	0.005

Once the base network is built (590 nodes) with the five areas, as many networks are created as necessary. They are situated (figuratively speaking) one on top of the other, as shown in Fig. 2(b), and are joined via the nodes of the Z1, Z2, Z3 and Z4 areas with an equal number of nodes located in the layer immediately below, using the same lines between nodes as those described in Table 1. This way, as large a network as needed can be created by simply adding another layer to the network.

This system generates three cases in addition to the IEEE-118 and IEEE-300 test cases, the 118Z5x24, the 118Z5x48, and the 118Z5x96 with 14160, 28320 and 56640 nodes, respectively.

The results in table 2 were obtained using the parallelization method with the predictor-corrector algorithm, the times obtained with the multiple centrality-corrections method do not vary significantly.

Table 2. Executions Times (ms) vs number of cores.

System	Cores	Total	Fact.	Subs.	Upd.	Other
IEEE118	1	32.2	15.2	11.4	2.5	3.0
	2	26.8	11.6	11.2	1.5	2.4
	3	23.0	9.8	9.8	1.0	2.4
	4	23.0	9.4	9.9	1.0	2.7
IEEE300	1	56.7	29.3	17.0	5.8	4.4
	2	45.8	21.9	15.3	4.1	4.4
	3	37.5	16.1	13.4	3.2	4.8
	4	35.5	15.2	12.9	2.5	4.8
IEEE118Z5x24	1	3190	1648	777	408	355
	2	2283	1082	620	251	329
	3	1823	784	527	196	315
	4	1674	661	503	186	323
IEEE118Z5x48	1	6558	3380	1617	867	692
	2	4727	2237	1277	547	666
	3	3850	1680	1110	414	644
	4	3460	1430	1011	345	673
IEEE118Z5x96	1	13508	6934	3384	1724	1464
	2	9806	4644	2642	1128	1390
	3	8007	3496	2292	858	1359
	4	7245	2992	2086	743	1423

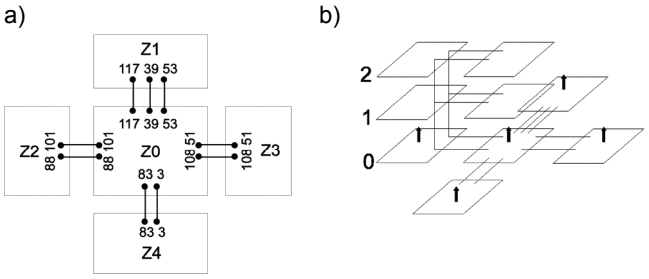

Fig. 2 Scheme for the base layer (118Z5) and expansion mechanism (118Z5xN)

The average times the different systems take to run sequentially (1 core) and the parallelization times obtained from the different proposed parts (factorizations, substitutions, updates and other operations) can be seen in table 2. In all cases the units of time are milliseconds.

One method used to assess the speed of the parallel algorithm and compare it to the sequential algorithm is the speedup ($Sp = Ts/Tp$), which is the ratio between the time spent on sequential execution and time spent on the

parallel execution. Ideally it should be equal to the number of cores p. Another method used to test the efficiency is the ratio Sp/p, whose ideal value is 1.

The efficiency of the parallelization process of the IEEE5Zx24 system for 2, 4 and 8 cores is 0.7, 0.44 and 0.24 respectively, and its speedup can be seen in Fig. 3. Data are from the 1185Zx24 system but no differences have been found with the other cases studied, Fig. 4.

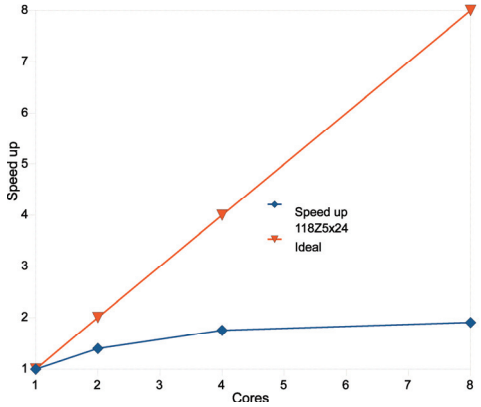

Fig. 3 Speedup

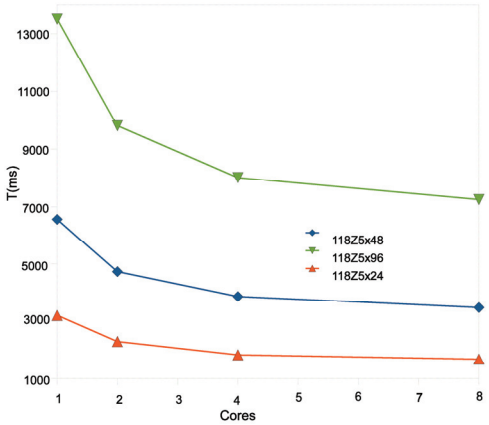

Fig. 1 Cores vs execution times

Conclusion

The execution times decrease as the number of cores used to solve the OPF increases. This demonstrates the need to upgrade the routines present in the OPF algorithms for a more efficient use of the power that currently all personal computers have today, and which will no doubt increase in the future.

The standard machines are 2 and 4 cores for notebook and desktop computers, respectively. Although nowadays computers can be bought with 6-core chips and configurations of two microprocessors that can reach up to 12 cores, they are still not widely available in late 2010.

An important aspect of parallelization is system scalability. It can be assumed to be good in the case of 2 cores and acceptable in 4 cores, with an efficiency of 70% and 44%, respectively. Nevertheless, beyond a quad-core, parallelization does not efficiently take advantage of available resources, and a drop in efficiency of up to 24% for 8 cores is experienced.

In the case of small systems, a limit could be imposed of between 200 and 300 nodes; the use of parallelism does not seem to have a practical justification with a high number of cores, as can see in the case of IEEE-118 nodes. Sequentially, these systems are solved so quickly that parallelization is clearly inefficient.

To obtain a greater scalability, future papers will study the possibility of using a hybrid technique. First, a partition of the network into areas with a technique like one of those described in the bibliography, and then each one of them will be solved with a reduced group of cores (2 or 4), where parallelization efficiency using OpenMP is greater.

Acknowledgment

This work was supported by the Galician autonomous government under the contract INCITE08PXIB303262PR.

REFERENCES

[1] B. H. Kim and R. Baldick, “Coarse-grained distributed optimal power flow,” IEEE Trans. Power Syst., vol. 12, no. 2, pp. 932–939, May 1997.

[2] R. Baldick, B. H. Kim, C. Chase, and Y. Luo, “A fast distributed implementation of optimal power flow,” IEEE Trans. Power Syst., vol. 14, no. 3, pp. 858–864, Aug 1999.

[3] B. Kim and R. Baldick, “A comparison of distributed optimal power flow algorithms,” IEEE Trans. Power Syst., vol. 15, no. 2, pp. 599–604, May 2000.

[4] A. J. Conejo and J. A. Aguado, “Multi-area coordinated decentralized dc optimal power flow,” IEEE Trans. Power Syst., vol. 13, no. 4, pp. 1272–1278, Nov 1998.

[5] J. A. Aguado, V. H. Quintana, and A. J. Conejo, “A computational comparison of two different approaches to solve the multi-area optimal power flow problem,” in Conference Proceedings. IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.98TH8341). IEEE, 1998, pp. 681–684.

[6] ——, “Optimal power flows of interconnected power systems,” in 199 IEEE Power Engineering Society Summer Meeting. Conference Proceedings (Cat. No.99CH36364). IEEE, 1999, pp. 814–819.

[7] J. A. Aguado and V. H. Quintana, “Inter-utilities powerexchange coordination: a market-oriented approach,” IEEE Trans. Power Syst., vol. 16, no. 3, pp. 513–519, Aug 2001.

[8] A. G. Bakirtzis and P. N. Biskas, “A decentralized solution to the dc-opf of interconnected power systems,” IEEE Trans. Power Syst., vol. 18, no. 3, pp. 1007–1013, August 2003.

[9] P. N. Biskas, A. G. Bakirtzis, N. I. Macheras, and N. K. Pasialis, “A decentralized implementation of dc optimal power flow on a network of computers,” IEEE Trans. Power Syst., vol. 20, no. 1, pp. 25–33, January 2005.

[10] W. Yan, L. Wen, W. Li, C. Y. Chung, and K. P. Wong, “Decomposition-coordination interior point method and its application to multi-area optimal reactive power flow,” Int. J. Electr. Power Energy Syst., vol. In Press, Corrected Proof, 2010.

[11] H. Dommel and W. Tinney, "Optimal power flow solutions," IEEE Transactions on Power Apparatus and Systems, vol. PAS-87, no. 10, pp. 1866-1876, Oct. 1968.

[12] G. Torres and V. Quintana, “On a nonlinear multiple-centralitycorrections interior-point method for optimal power flow,” IEEE Trans. Power Syst., vol. 16, no. 2, pp. 222–228, May 2001.

[13] S. Mehrotra, “On the implementation of a primal-dual interior point method,” SIAM J. on Optimiz., vol. 2 (1992), 575-601.

[14] J. Gondzio, “Multiple centrality corrections in a primal-dual method for linear programming,” Computational Optimization and Applications, vol. 6, no. 2, pp. 137-156-156, Sep. 1996.

[15] “OpenMP.org.” [Online]. Available: http://openmp.org/wp/

[16] B. Chapman, G. Jost, and R. Van Der Pas, “Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation).” The MIT Press, Oct. 2007.

[17]“PARDISO solver project.” [Online]. Available: http://www.pardiso-project.org/

***Authors**: M.Sc. J. C. Moreira, Ph.D. E. Míguez, M.Sc. C. Vilachá and Ph.D. Antonio F. Otero are with the Department of Electrical Engineering of University of Vigo, Galicia, Spain, E-mails: jcmeira@uvigo.es, edelmiro@uvigo.es; cvilacha@uvigo.es; afotero@uvigo.es.*