The Performance Model of an Enhanced Parallel Algorithm for the SOR Method

Italo Epicoco[1,2] and Silvia Mocavero[2]

[1] University of Salento, Lecce, Italy
italo.epicoco@unisalento.it
[2] Euro-Mediterranean Center for Climate Change (CMCC), Lecce, Italy
silvia.mocavero@cmcc.it

Abstract. The Successive Over Relaxation (SOR) is a variant of the iterative Gauss-Seidel method for solving a linear system of equations $Ax = b$. The SOR algorithm is used within the NEMO (Nucleus for European Modelling of the Ocean) ocean model for solving the elliptical equation for the barotropic stream function. The NEMO performance analysis shows that the SOR algorithm introduces a significant communication overhead. Its parallel implementation is based on the Red-Black method and foresees a communication step at each iteration. An enhanced parallel version of the algorithm has been developed by acting on the size of the overlap region to reduce the frequency of communications. The overlap size must be carefully tuned for reducing the communication overhead without increasing the computing time. This work describes an analytical performance model of the SOR algorithm that can be used for establishing the optimal size of the overlap region.

Keywords: SOR, NEMO, Performance Model.

1 Introduction

The ocean engine of NEMO (Nucleus for European Modelling of the Ocean) [1] is a primitive equation model adapted to regional and global ocean circulation problems. It is a flexible tool for studying the ocean and its interactions with the other components of the earth climate system over a wide range of space and time scales. Prognostic variables are the three-dimensional velocity field, the sea surface height, the temperature and the salinity. In the horizontal direction, the model uses a curvilinear orthogonal grid and in the vertical direction, a full or partial step z-coordinate, or s-coordinate, or a mixture of the two. The model time stepping environment is a three level scheme in which the tendency terms of the equations are evaluated either centered in time, or forward, or backward depending on the nature of the term. The model is spatially discretized on a staggered grid (Arakawa C grid) masking the land points. Vertical discretization depends on both how the bottom topography is represented and whether the free surface is linear or not. Explicit, split-explicit and filtered free surface formulations are implemented for solving the prognostic equations for the active tracers and the momentum. A number of numerical schemes are available for the momentum advection, for the computation of the pressure gradients, as well as for the advection of the tracers (second or higher order advection schemes, including positive ones). When the filtered sea surface height option is used, a new force that can be interpreted as a diffusion of the vertically integrated volume flux divergence is added in the momentum equation. The equation is solved implicitly and it represents an elliptic equation for which two solvers are available: the SOR and the Preconditioned Conjugate Gradient (PCG) schemes. The SOR has been retained because it is a linear solver very useful when using the adjoint model of NEMO. The NEMO model with the MFS16 [2] configuration has been evaluated on the MareNostrum platform at the Barcelona Supercomputing Center. The routine named *dyn_spg* is the most time consuming one; it computes the surface pressure gradient term using the SOR scheme.

The paper is organized as follows: next section introduces the SOR (Successive Over Relaxation) method. Section 3 describes our parallel approach, while the latter sections, 4 and 5, show respectively the analytical performance model of the parallel algorithm and the iso-efficiency analysis.

2 SOR Overview

The iterative methods for solving the linear equation systems $Ax = b$ iteratively generates a sequence $\{p_k\}$ of approximate solutions such that the residual vector ($r_k = b - Ap_k$) converges to 0. The Gauss-Seidel algorithm [3] is an example of iterative method for solving a linear equation system. The method can be applied only if the matrix A is strictly diagonally dominant. Each equation is solved by the unknown on the diagonal and the approximated values for the other unknowns are plugged in. The process is then iterated until convergence. The Gauss-Seidel method is easily derived by examining separately each of the n equations in the linear system. Let the i-th equation given by:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \tag{1}$$

At the iteration k, it can be solved by (2) for the value of $x_i^{(k)}$ assuming the approximation of the previous iteration ($x_{j\neq i}^{(k-1)}$) for the other unkowns $x_{j\neq i}$.

$$x_i^{(k)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) \tag{2}$$

There are two important characteristics of the Gauss-Seidel method that should be noted. Firstly, the computation appears to be serial: since each component at the new iteration depends on all of the previously computed components, the updates cannot be done simultaneously as in the Jacobi method [4]. Secondly, the new iterate value $x^{(k)}$ depends upon the order in which the equations are examined. If it changes, the values at the new iteration (and not just their order) change accordingly.

The definition of the Gauss-Seidel method can be expressed using the following matrix notation:

$$x^{(k)} = (D - L)^{-1}(Ux^{(k-1)} + b) \tag{3}$$

where the matrices D, $-L$, and $-U$ represent the diagonal, the strictly lower triangular, and the strictly upper triangular parts of A, respectively. The SOR [5] is an iterative method for solving a linear system of equations derived by extrapolating the Gauss-Seidel algorithm. This extrapolation takes the form of a weighted average between the previous iteration and the Gauss-Seidel component computed at the current iteration. Given a value for the weight ω the component at iteration k is given by:

$$x_i^{(k)} = \omega \overline{x}_i^{(k)} + (1 - \omega)x_i^{(k-1)} \tag{4}$$

where $\overline{x}$ denotes a Gauss-Seidel approximation. The idea is to choose a value for ω within the interval $(0, 2)$ that will accelerate the rate of convergence to the solution. In general, it is not possible to compute in advance the value of ω that will maximize the rate of convergence of the SOR. Frequently, some heuristic estimate is used, such as $\omega = 2 - O(h)$ where h is the mesh spacing of the discretization of the underlying physical domain.

In matrix terms, the SOR algorithm can be written as follows:

$$x^{(k)} = (D - \omega L)^{-1}[\omega U + (1 - \omega)D]x^{(k-1)} + \omega(D - \omega L)^{-1}b \tag{5}$$

3 Parallel Algorithm

While the matrix notation for the SOR algorithm is useful for a theoretical analysis, a practical implementation requires an explicit formula to be defined [6]. Let's consider a general second-order elliptic equation in x and y, finite differenced on a square. Each row of the matrix A is an equation of the form:

$$a_{i,j}u_{i+1,j} + b_{i,j}u_{i-1,j} + c_{i,j}u_{i,j+1} + d_{i,j}u_{i,j-1} + e_{i,j}u_{i,j} = f_{i,j} \tag{6}$$

The iterative procedure is defined by solving the following equation for $u_{i,j}$.

$$u^*_{i,j} = \frac{f_{i,j} - a_{i,j}u_{i+1,j} - b_{i,j}u_{i-1,j} - c_{i,j}u_{i,j+1} - d_{i,j}u_{i,j-1}}{e_{i,j}} \tag{7}$$

Then, considering the (4), the $u^{new}_{i,j}$ is a weighted average given by:

$$u^{new}_{i,j} = \omega u^*_{i,j} + (1 - \omega)u^{old}_{i,j} \tag{8}$$

If we consider that the residual at any stage of the iteration is given by:

$$\begin{aligned} \xi_{i,j} =& a_{i,j}u_{i+1,j} + b_{i,j}u_{i-1,j} + c_{i,j}u_{i,j+1} + d_{i,j}u_{i,j-1} + \\ & e_{i,j}u_{i,j} - f_{i,j} \end{aligned} \tag{9}$$

we can calculate the new value at each iteration given by:

$$u_{i,j}^{new} = u_{i,j}^{old} - \omega \frac{\xi_{i,j}}{e_{i,j}} \tag{10}$$

This formulation is very easy to program, and the norm of the residual vector $\xi_{i,j}$ can be used as a criterion for terminating the iteration. The need to reduce the time spent by the SOR algorithm without increasing the number of iterations to reach convergence has been the main goal of several previous works. Different multi-color ordering techniques, such as the Red-Black [7] method for two dimensional problems, have been investigated; they allow the parallelization of operations on the same color. Other techniques, overlapping computation and communication or allowing an optimal scheduling of available processors, have been designed and implemented producing parallel versions of SOR [8]. Parallel SOR algorithms, suitable for use on an asynchronous MIMD computer, are presented since 1984 [9]. In the last years, the BPSOR [10], characterized by a new mesh domain partition and ordering, allows retaining the same convergence rate of the sequential SOR method with an easy parallel implementation on an MIMD parallel computing.

This work analyzes a parallel algorithm for the SOR based on the Red-Black method that supposes to divide the mesh into odd and even cells, like in a checkerboard. Equation (10) shows that the odd point values depend only on the even points, and vice versa. Accordingly, we can carry out one half-sweep updating the odd points and then another half-sweep updating the even points with the new odd values. The parallel algorithm uses a 2D domain decomposition based on checkerboard blocks. Let n_i and n_j be respectively the number of rows and columns of the global domain, and p_i and p_j respectively the number of processes along i and j directions, then each process will compute a subdomain made of n_i/p_i x n_j/p_j elements. If we consider only one overlap line between neighbors, each parallel process must exchange the computed values at the border at each iteration of the SOR. Two communication steps must be performed for each iteration (for the odd and for the even points). At each iteration, the generic process computes the odd points inside its domain, exchanges the odd points with its neighbors and updates the boundaries values, computes the inner even points and finally updates even points on the boundaries exchanging with neighbors (see Fig. 1). At each iteration, the generic parallel process will then communicate twice for each neighbor. In order to reduce the frequency of communication, the size of the overlap region could be increased [11]. In that case the neighboring processes would exchange a wider overlap region, but the values exchanged can be used for further iterations without the need of communication. Each process, after exchanging the data, computes a total number of lines given by $N_{inner} + N_{ol} - 1$, where N_{inner} and N_{ol} are respectively the total number of lines in the inner domain and the number of overlap lines. At each iteration only one line of the overlap expires so that the process has no need to exchange for $N_{ol} - 1$ iterations. The convergence rate of the algorithm does not change, since the ordering and partition is the same of the original SOR algorithm. The algorithm is explained by the following pseudo-code fragment.

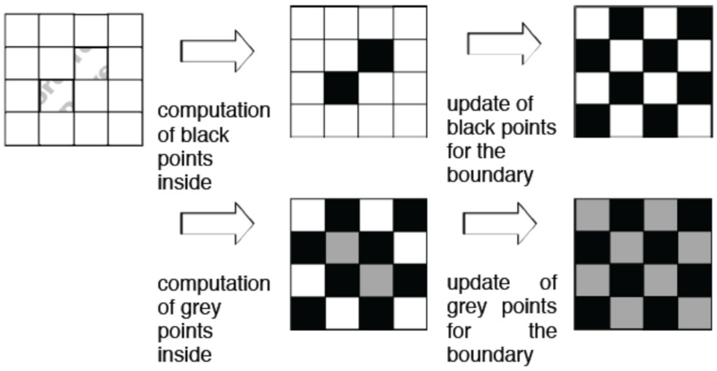
computation of black points inside
update of black points for the boundary
computation of grey points inside
update of grey points for the boundary

Fig. 1. SOR Red-Back computing algorithm

```
Require: $u$ //result matrix with initial value
  $a, b, c, d, e$ //coefficient matrix
  $f$ //known term

  $ol\_exp \leftarrow ol$
  while $\xi(i, j)$ is not enough small do
    if $ol\_exp == 0$ then
      call data_exch //exchange odd points over the overlap
    end if
    for all even points do
      $tmp \leftarrow (f(i, j) - a(i, j) * u(i, j - 1) - b(i, j) * u(i, j + 1) - c(i, j) * u(i - 1, j) - d(i, j) * u(i + 1, j))/e(i, j)$
      $\xi(i, j) \leftarrow tmp - u(i, j)$
      $u(i, j) \leftarrow \omega * tmp + (1 - \omega) * u(i, j)$
    end for
    if $ol\_exp == 0$ then
      call data_exch //exchange even points over the overlap
      $ol\_exp \leftarrow ol$
    end if
    for all odd points do
      $tmp \leftarrow (f(i, j) - a(i, j) * u(i, j - 1) - b(i, j) * u(i, j + 1) - c(i, j) * u(i - 1, j) - d(i, j) * u(i + 1, j))/e(i, j)$
      $\xi(i, j) \leftarrow tmp - u(i, j)$
      $u(i, j) \leftarrow \omega * tmp + (1 - \omega) * u(i, j)$
    end for
    call convergence_test ($\xi$)
    $ol\_exp \leftarrow ol\_exp - 1$
  end while
```

A similar approach has been used by the HYCOM ocean model [12] where a maximum number of wide halo lines can be added to reduce the halo communication overhead.

4 The Analytical Performance Model

The SOR algorithm has been implemented in a test program made by the main *sor* routine that (i) calls the *data_exch* routine for exchanging the data between the neighbors and (ii) evaluates the convergence. Both the routines are then characterized by two kind of operations: computing and communication. *data_exch* performs some data buffering operations and the actual send and receive of the data on the boundaries. The size of the overlap region directly impacts on the frequency of the *data_exch* invocation. The *sor* routine computes the result matrix and performs a collective communication during the convergence test. If we increase the size of the overlap, the computation increases, while the time for collective communication does not change. The total time is the sum of these four components, three of them depending on the size of the overlap. Which is the best value of the overlap to get the best benefit? The optimal value is related to some architectural aspects (i.e. the processor speed and the network bandwidth and latency) and changes with both the number of parallel processes and the domain decomposition. A performance model for estimating the behavior of the SOR algorithm has been defined, such as in [13]. It takes into consideration the four above mentioned aspects. The total time spent by the solver (T_{sor}) is given by: (i) the communication time spent in the *sor* routine for the convergence test (T_{c_sor}); (ii) the computing time spent in the *sor* routine for evaluating the result matrix at each iteration (T_{u_sor}); (iii) the computing time spent in the *data_exch* routine for managing the data buffer used for the data transmission (T_{u_data}) and (iv) the communication time in the *data_exch* for the data transfer to the neighbors (T_{c_data}). The number of calls of *data_exch* depends on both the overlap size (l) and the number of iterations (m) needed to reach the convergence. The performance model is summarized as follows:

$$T_{sor} = T_{c\_sor} + T_{u\_sor} + \left(\frac{2m}{l} + 1\right)(T_{c\_data} + T_{u\_data}) \tag{11}$$

The four timing components can be modeled as in (12)(13)(14).

The time spent by the collective communication depends only on the number of parallel processes ($p_i p_j$). The convergence test is performed after the first 100 iterations and has a frequency of 10 iterations through an *allreduce* MPI collective communication, where the maximum residual value is exchanged among all of the parallel processes. The amount of data exchanged is constant (it is independent from the subdomain dimension) and, considering the implementation of the *allreduce* with the butterfly parallel scheme, we have a number of communication steps logarithmic to the total number of processes.

$$T_{c\_sor} = O(\frac{m-100}{10} \log p_i p_j) \tag{12}$$

The computing time of the *sor* is related to the domain dimension: d_i and d_j are the dimensions of the biggest subdomain along the i and j directions respectively and they are given by $d_i = n_i/p_i$ and $d_j = n_j/p_j$. For each iteration of the SOR a complete sweep of the subdomain elements plus the overlap region is performed.

$$T_{u\_sor} = O(m(d_i + l)(d_j + l)) \tag{13}$$

The communication is implemented with four point-to-point sends/receives hence the communication time is directly proportional to the number of exchanged elements. Here we consider a parallel process with four neighbors, but not all of the processes have four neighbors like those in the border of the global domain.

$$\begin{aligned} T_{c\_data} &= O(L_i + L_j) \\ T_{u\_data} &= O(L_i + L_j) \end{aligned} \tag{14}$$

L_i and L_j represent the total number of elements exchanged between neighbors, $L_i = (d_i + 2l)l$, $L_j = (d_j + 2l)l$.

Considering all of the previous equations, the parallel time of the whole algorithm can be expressed as follows:

$$T_{sor} = O(\frac{n_i n_j}{p_i p_j} + \frac{n_i l}{p_i} + \frac{n_j l}{p_j} + l^2 + \log p_i p_j) \tag{15}$$

If we consider a square global domain then $n_i = n_j = n$, we can also impose $p_i = p_j = \sqrt{p}$. The (15) can be simplified:

$$T_{sor} = O(\frac{n^2}{p} + \frac{nl}{\sqrt{p}} + l^2 + \log p) \tag{16}$$

The evaluation of the analytical equation for the performance model has been defined experimentally on an IBM Power6 cluster. It has 30 IBM p575 nodes, each of them equipped with 16 Power6 dual-core CPUs at 4.7GHz and 128GB of shared memory (4GB per core). The nodes are interconnected by an Infiniband network.

The minimum square method on a set of several runs, for which we fixed the global domain of 871x253 grid points and a predefined domain decomposition and modified the overlap size, has been used. For both the *data_exch* terms, when the overlap size changes, the number of exchanged element changes accordingly: figs. 2 and 3 show the communication and the computing time with the least square equation used to evaluate the multiplicative coefficients for the T_{c_data} and T_{u_data} terms. Regarding the analysis of the *sor* terms, we have taken into consideration only the trend of the computing time. Indeed, the communication time does not depend on the overlap size. Figure 4 shows the computing trend of the *sor* routine without considering the time spent calling the *data_exch*. The optimal value of the overlap size can be analytically defined setting at zero the derivate of the total execution time respect to the overlap size l, hence solving (17) by l.

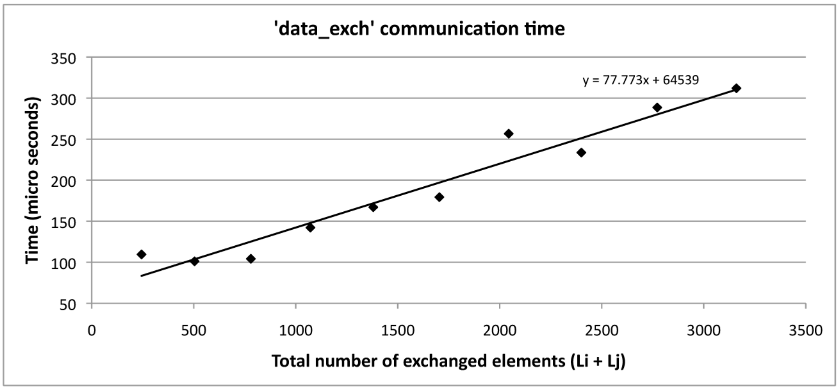

Fig. 2. *data_exch* communication time depending on the number of elements exchanged among all of the neighbors

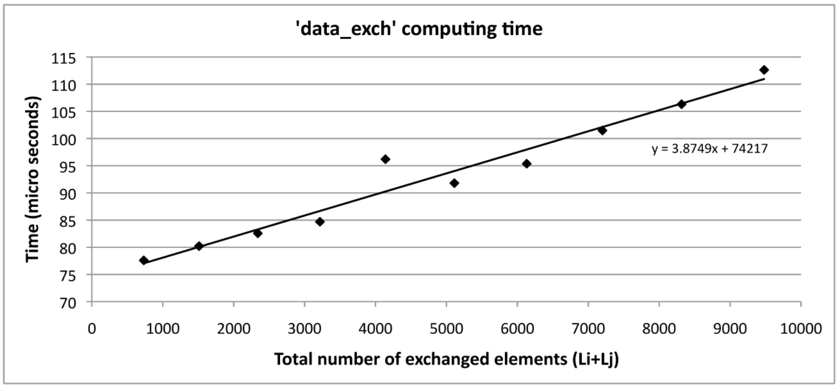

Fig. 3. *data_exch* computing time depending on the number of elements exchanged among all of the neighbors

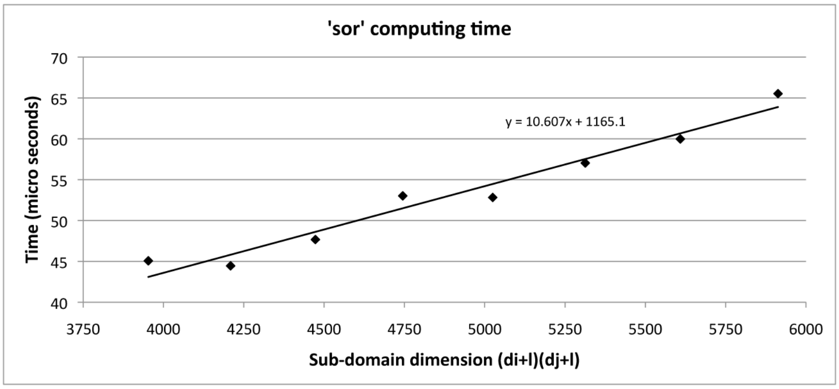

Fig. 4. *sor* computing time depending on the subdomain and the overlap size

$$\frac{dT_{sor}(n,p,l)}{dl} = 0 \tag{17}$$

As alternative we can easily derive the analytical expression of l considering that T_{u_sor} increases with l since the computation must be performed also on the elements of the overlap region (let T^+ be the positive delta time) while the communication start-up time decreases with l (let T^- be the negative delta time). The optimal value of l is such that $T^+ = T^-$. Considering that T^+ and T^- are given by the following two equations:

$$T^+ = 4(l-1)\left[\frac{n}{\sqrt{p}} + 2(l-1)\right] t_c \tag{18}$$

$$T^- = (l-1)t_s \tag{19}$$

where t_c denotes the time for executing the SOR operations on one matrix element and t_s is the communication start-up time. Hence the optimal overlap size is given by:

$$l = \frac{t_s}{8t_c} - \frac{n}{2\sqrt{p}} \tag{20}$$

and it is strictly dependent on the architectural parameters and namely on the ratio between the network latency and CPU speed. Moreover, l decreases with the subdomain perimeter increasing. Indeed when the subdomain perimeter grows up, the added time T^+ increases accordingly while T^- depends only on the number of communication steps.

4.1 Model Validation

The model has been tested and evaluated using three domain sizes: 871x253, $2K$x$2K$ and $10K$x10K grid points. The first domain size has been chosen since it is used in the NEMO ocean model with MFS16. This is a production configuration of strategical scientific interest for the Euro-Mediterranean Center for Climate Change (CMCC) and it is employed for production experiments. Figure 5 reports a comparison between the model and the experimental data for the 871x253 domain size with a decomposition of 16x4 parallel processes. The theoretical model approximates accurately the real behavior with a deviation that is limited to the 5% of the execution time for $l > 1$. As we can notice, reducing the number of calls to *data_exch*, we consequently reduce the communication time. After a threshold, it is not convenient to increase the overlap size and the model helps us to define the value of this threshold when the decomposition changes. It depends on the domain decomposition, and more in particular on the balance between computing and communication time.

The parallel speedup of the algorithm has been evaluated: for each domain decomposition the optimal value for the overlap region has been applied.

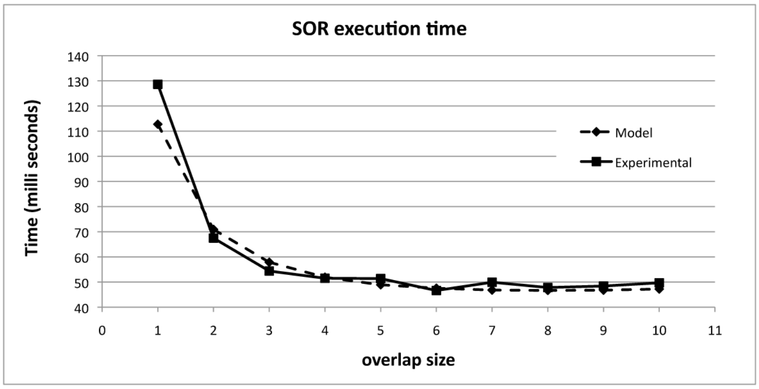

Fig. 5. Validation of the analytical performance model

5 Iso-efficiency Analysis

Even if the main goal of this work is to find the optimal value of the overlap size, the analytical performance model also allows us to study the parallel efficiency of the SOR algorithm. Given a parallel algorithm with problem size n, executed on p processes, the iso-efficiency [14] relation is given by:

$$T(n,1) \geq CT_o(n,p) \tag{21}$$

where $T(n,1)$ is the execution time of the sequential algorithm and $T_o(n,p)$ is the time to execute all of those operations introduced by the parallelization, and namely is given by:

$$T_o(n,p) = (p-1)\sigma(n) + pk(n,p) \tag{22}$$

where $\sigma(n)$ is the sequential part of the algorithm and $k(n,p)$ the communication time of each processor. The constant C is strictly related to the parallel efficiency. In order to maintain the same level of efficiency as the number of processes increases, the parallel system must be *cost-optimal* [15]. The SOR sequential algorithm has a computational complexity of $O(n^2)$. The sequential time is given by:

$$T(n,1) = O(n^2) \tag{23}$$

The operations added by the parallel algorithm are: (i) the computing of the elements within the overlap region with a complexity of $O(nl\sqrt{p})$; (ii) the exchange of the overlap region values among neighbors with a complexity of $O(pnl/l\sqrt{p})$ and (iii) the communications for evaluating the convergence with a complexity of $O(\log p)$. The total overhead time is given by:

$$T_o(n,p) = O\left(nl\sqrt{p} + \frac{np}{\sqrt{p}} + \log p\right) \tag{24}$$

The logarithmic term can be ignored since it has a complexity lower than $\sqrt{p}$. The iso-efficiency relation is hence given by:

$$\begin{aligned} n^2 &\geq nl\sqrt{p} + n\sqrt{p} \\ n &\geq \sqrt{p}(l+1) \end{aligned} \tag{25}$$

Considering for l the optimal value given by (20) we can rewrite the (25) as follows:

$$n \geq \frac{2\sqrt{p}}{3}\left(\frac{t_s}{8t_c} + 1\right) = K\sqrt{p} \tag{26}$$

$n \geq f(p)$ denotes the iso-efficiency equation. The complexity of the problem (n^2) is proportional to the number of processes (p), then the parallel system is cost-optimal. The results, showed in Fig. 6, confirm that the parallel scalability is very poor for small problem sizes while it is good for bigger problems. The analytical results in Fig. 6 refer to some cases for which the matrix is not square and the matrix order (n) is not multiple of the number of processes (p). This could introduce a displacement from the iso-efficiency theoretical analysis.

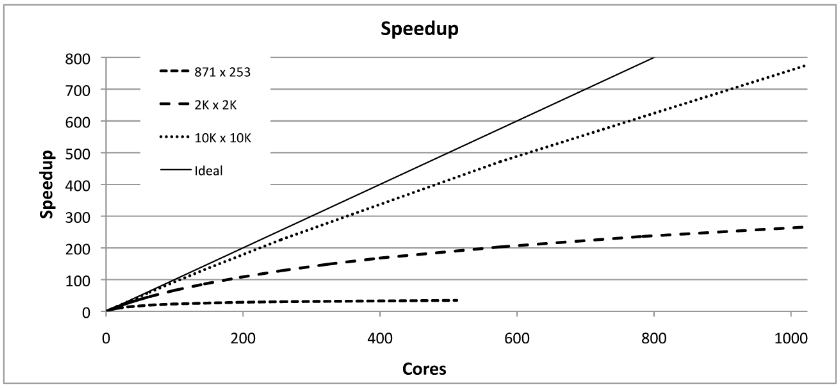

Fig. 6. Speedup of the SOR algorithm using the optimal size for the overlap region

The maximum problem size we can solve could be limited by the amount of the available main memory. Considering $M(n)$ the amount of the memory required to store the data for a problem of size n, the scalability function is given by $M(f(p))/p$. It indicates how the amount of memory used per process must increase as a function of p in order to maintain the same level of efficiency. For the SOR algorithm the amount of memory required is proportional to the number of elements in the following result matrix:

$$M(n) = O(n^2) \tag{27}$$

and hence we have:

$$M(f(p))/p = O((K\sqrt{p})^2/p) \tag{28}$$

hence $M(f(p))/p = O(1)$. This implies that the amount of memory required to solve problems with size that increases according to the iso-efficiency equation, is constant. The weak scalability is not limited by the given primary memory per process. We can conclude that the algorithm is perfectly scalable.

6 Conclusions

In this work, we presented the definition of an analytical performance model to establish the optimal overlap size for the SOR algorithm. The parallel algorithm can be optimally tuned acting on the size of the overlap region. The optimal value is given by a trade-off between computing and communication time. The use of the proposed performance model drives the user decision making strategies in the choice of the overlap size. An implementation of the algorithm has been evaluated on an IBM Power6 parallel architecture. The theoretical analysis of the parallel algorithm demonstrated a perfect weak scalability. Some criticality is evident when the number of parallel processes is so big that the dimension of the subdomain is smaller than the optimal value of the overlap region. For these cases a communication among processes that are not directly bordered is required. For the future, we plan to (i) modify the SOR parallel algorithm implementation in order to extend the exchange of data not only to the neighboring processes and (ii) introduce an overlapping between communication and computing operations leveraging on the Red-Black method.

References

1. Madec, G., et al.: NEMO ocean engine. Note du Pole de modellisation, Institut Pierre-Simon Laplace (IPSL), France, No 27 (2008) ISSN No 1288-1619
2. Tonani, T., Pinardi, N., Dobricic, S., Pujol, I., Fratianni, C.: A high-resolution free-surface model of the Mediterranean Sea. Ocean Science 4, 1–14 (2008)
3. Hageman, L., Young, D.: Applied Iterative Methods. Academic Press, New York (1981)
4. Bronshtein, I.N., Semendyayev, K.A.: Handbook of Mathematics, 3rd edn., p. 892. Springer, New York (1997)
5. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., Van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd edn. SIAM, Philadelphia, PA (1994)
6. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Successive Overrelaxation (SOR). Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd edn., pp. 866–869. Cambridge University Press, Cambridge (1992)
7. Yavneh, I.: On Red Black SOR Smoothing in Multigrid. SIAM Journal on Scientific Computing 17(1), 180–192 (1994)
8. Niethammer, W.: The SOR method on parallel computers. Numer. Math. 56, 247–254 (1989)
9. Evans, D.J.: Parallel SOR iterative methods. Parallel Computing 1, 3–18 (1984)
10. Xie, D.: A New Block Parallel SOR Method and its Analysis. SIAM Journal on Scientific Computing 27(5), 1513–1533 (2006)

11. Benshila, R., et al.: Optimization of the SOR solver for parallel run. Technical report Version 1, LOCEAN-IPSL, France (2005)
12. Wallcraft, A.J., Chassignet, E.P., Hurlburt, H.E., Townsend, T.L.: 1/25° Atlantic Ocean Simulation Using HYCOM. DoD HPCMP Users (2005)
13. Meng, J., Skadron, K.: Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs. In: Proceedings of the 23rd International Conference on Supercomputing, ICS 2009 (2009) ISBN: 978-1-60558-498-0, doi:10.1145/1542275.1542313
14. Grama, A.Y., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing, 2nd edn. Addison-Wesley, Harlow (2003)
15. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distributed Technology Systems Applications 1(3), 12–21 (1993) ISSN: 10636552